HITACHI
Inspire the Next

VisiBroker Version 5

Borland(R) Enterprise Server VisiBroker(R)　プログラマーズリファレンス

文法書

3021-3-J31

前書き

■ 対象製品

●適用 OS：Windows Server 2016，Windows Server 2019，Windows 10 x64

P-2964-AF64 Cosminexus TPBroker 05-24-01

●適用 OS：AIX V7.1，AIX V7.2

P-1M64-CF61 Cosminexus TPBroker 05-24-01

●適用 OS：Red Hat Enterprise Linux 7.1 (AMD/Intel 64)，Red Hat Enterprise Linux 8.1 (AMD/Intel 64)

P-9S64-AF61 Cosminexus TPBroker 05-25

これらのプログラムプロダクトのほかにも，このマニュアルをご利用になれる場合があります。詳細は「リリースノート」でご確認ください。

■ 輸出時の注意

本製品を輸出される場合には、外国為替及び外国貿易法の規制並びに米国輸出管理規則など外国の輸出関連法規をご確認の上、必要な手続きをお取りください。

なお、不明な場合は、弊社担当営業にお問い合わせください。

■ 商標類

HITACHI，Cosminexus，TPBroker は，株式会社 日立製作所の商標または登録商標です。

AMD は，Advanced Micro Devices, Inc.の商標です。

IBM，AIX は，世界の多くの国で登録された International Business Machines Corporation の商標です。

Intel は，アメリカ合衆国および / またはその他の国における Intel Corporation またはその子会社の商標です。

Itanium は，アメリカ合衆国および / またはその他の国における Intel Corporation またはその子会社の商標です。

Linux は，Linus Torvalds 氏の日本およびその他の国における登録商標または商標です。

Microsoft は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。

Oracle と Java は，Oracle Corporation 及びその子会社，関連会社の米国及びその他の国における登録商標です。文中の社名，商品名等は各社の商標または登録商標である場合があります。

Red Hat, and Red Hat Enterprise Linux are registered trademarks of Red Hat, Inc. in the United States and other countries. Linux(R) is the registered trademark of Linus Torvalds in the U.S. and other countries.

Red Hat，および Red Hat Enterprise Linux は，米国およびその他の国における Red Hat, Inc.の登録商標です。Linux(R)は，米国およびその他の国における Linus Torvalds 氏の登録商標です。

UNIX は，The Open Group の商標です。

VisiBroker は，英国，米国またはその他の国における Micro Focus またはその子会社もしくは関連会社の商標または登録された商標です。

Windows は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。

Windows Server は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。

その他記載の会社名，製品名などは，それぞれの会社の商標もしくは登録商標です。

■ 発行

2020 年 3 月 3021-3-J31

■ 著作権

All Rights Reserved. Copyright (C) 2020, Hitachi, Ltd.

COPYRIGHT (C) 1992-2004 Borland Software Corporation. All rights reserved.

はじめに

このマニュアルは，Borland Enterprise Server VisiBroker が提供しているクラスとインタフェースの情報，プログラマツール，およびコマンドラインオプションについて説明したものです。Borland Enterprise Server VisiBroker は，分散オブジェクトベースのアプリケーションを，Common Object Request Broker Architecture（CORBA）仕様に従って開発し運用できるようにします。

■ 対象読者

Borland Enterprise Server VisiBroker 関連の製品をインストールして，システムを運用する方，管理ツールやアプリケーションを実行する方，および CORBA の仕様に基づく分散アプリケーションを開発する方を対象としています。

■ 文法の記号

このマニュアルで使用する文法記述記号を説明します。文法記述記号は文法の記述形式について説明する記号です。

文法記述記号	意味
背景色付きのボールド体（**boldface**）	背景色付きのボールド体は，記述どおりに構文をタイプすることを示します。また，コードサンプル部分を強調表示する場合にも使用されます。
イタリック体（*italics*）	イタリック体は，構文解析図の変数のように，ユーザまたはアプリケーションが提供する情報を示します。
|	横に並べられた複数の項目に対し，項目間の区切りを示し，「または」の意味を示します。 （例） A | B | C は「A，B，または C」を意味します。
[ ]	この記号で囲まれている項目は省略してもよいことを意味します。複数の項目が横に並べて記述されている場合には，すべてを省略するか，どれか一つを選択します。 （例） [A | B] では「何も指定しない」か，「A または B と指定する」ことを意味します。
{ }	この記号で囲まれている項目は，一つの構文の要素として扱うことを意味します。
< >	この記号で囲まれている項目は，該当する要素を指定することを意味します。
…	記述が省略されていることを示します。

目次

前書き　2
はじめに　4

第 1 編　Java 言語編

1　プログラマツール（Java）　35
1.1　オプション　36
1.1.1　一般オプション　36
1.2　idl2ir　37
1.2.1　構文　37
1.2.2　説明　37
1.3　ir2idl　39
1.3.1　構文　39
1.3.2　説明　39
1.4　idl2java　40
1.4.1　構文　40
1.4.2　説明　40
1.5　java2idl　44
1.5.1　構文　44
1.5.2　説明　44
1.6　java2iiop　47
1.6.1　構文　48
1.6.2　説明　48
1.7　vbj　52
1.7.1　構文　52
1.7.2　説明　52
1.7.3　オプション　53
1.8　vbjc　54
1.8.1　構文　54
1.8.2　オプション　54
1.9　クラスパスの指定　55
1.10　Java VM の指定　57
1.11　その他のツール　58

2 IDL から Java へのマッピング 60
2.1 名前 61
2.2 予約名 62
2.3 予約語 63
2.4 モジュール 64
2.5 基本型 65
2.5.1 IDL 型拡張 66
2.5.2 Holder クラス 66
2.5.3 boolean 70
2.5.4 char 70
2.5.5 octet 70
2.5.6 string 70
2.5.7 wstring 71
2.5.8 整数型 71
2.5.9 浮動小数点型 71
2.6 Helper クラス 72
2.7 定数 74
2.7.1 インタフェース内の定数 74
2.7.2 インタフェース内にない定数 74
2.8 構造型 76
2.8.1 enum 76
2.8.2 struct 78
2.8.3 union 79
2.8.4 sequence 81
2.8.5 array 82
2.9 interface 84
2.9.1 abstract インタフェース 86
2.9.2 ローカルインタフェース 86
2.9.3 パラメタの受け渡し 87
2.9.4 継承によるサーバインプリメンテーション 88
2.9.5 デリゲーションを使用したサーバインプリメンテーション 89
2.9.6 インタフェーススコープ 90
2.10 例外のマッピング 91
2.10.1 ユーザ定義例外 91
2.10.2 システム例外 92
2.11 Any 型のマッピング 93
2.12 ネストされた型のマッピング 94
2.13 typedef のマッピング 95
2.13.1 シンプル IDL 型 95

2.13.2 複合 IDL 型 95

3 生成されるインタフェースとクラス（Java） 96
3.1 概要 97
3.1.1 Signature クラスと Operations クラス 97
3.1.2 補助クラス 97
3.1.3 ポータビリティスタブインタフェースとポータビリティスケルトンインタフェース 98
3.2 <interface_name>Operations 99
3.3 <type_name>Helper 100
3.3.1 Helper のメソッド 100
3.3.2 インタフェース用に生成されるメソッド 101
3.3.3 オブジェクトラッパー用に生成されるメソッド 102
3.4 <type_name>Holder 104
3.4.1 メンバデータ 105
3.4.2 Holder のメソッド 105
3.5 _<interface_name>Stub 106
3.6 <interface_name>POA 107
3.7 <interface_name>POATie 108
3.7.1 tie のメソッド 108

4 コアインタフェースとクラス（Java） 109
4.1 CompletionStatus 110
4.1.1 IDL の定義 110
4.1.2 CompletionStatus のメンバ 110
4.1.3 CompletionStatus のメソッド 110
4.2 Context 111
4.2.1 IDL の定義 111
4.2.2 Context のメソッド 112
4.3 InvalidName 114
4.4 Object 115
4.4.1 org.omg.CORBA.Object の定義 115
4.4.2 org.omg.CORBA.Object のメソッド 116
4.4.3 VisiBroker での Object の継承 119
4.4.4 VisiBroker での Object のメソッドの継承 119
4.5 ORB 121
4.5.1 JDK での ORB の定義 121
4.5.2 JDK での ORB メソッド 123
4.5.3 OMG による ORB の定義 132
4.5.4 VisiBroker での ORB の継承 132

4.5.5 ORB に対する VisiBroker の拡張機能 132
4.5.6 ORB の初期化と終了 134
4.6 Policy 136
4.6.1 IDL の定義 136
4.6.2 Policy のメソッド 136
4.7 PortableServer.AdapterActivator 137
4.7.1 import 文 137
4.7.2 PortableServer.AdapterActivator のメソッド 137
4.8 PortableServer.Current 138
4.8.1 import 文 138
4.8.2 PortableServer.Current のメソッド 138
4.9 PortableServer.POA 139
4.9.1 import 文 139
4.9.2 PortableServer.POA のメソッド 139
4.10 PortableServer.POAManager 149
4.10.1 import 文 149
4.10.2 PortableServer.POAManager のメソッド 150
4.11 PortableServer.POAManagerPackage.State 151
4.11.1 import 文 151
4.11.2 PortableServer.POAManagerPackage.State のメンバ 151
4.12 PortableServer.ServantActivator 152
4.12.1 import 文 152
4.12.2 PortableServer.ServantActivator のメソッド 152
4.13 PortableServer.ServantLocator 154
4.13.1 import 文 154
4.13.2 PortableServer.ServantLocator のメソッド 154
4.14 PortableServer.ServantManager 156
4.14.1 import 文 156
4.15 PortableServer.ForwardRequest 157
4.15.1 import 文 157
4.15.2 ForwardRequest の変数 157
4.15.3 ForwardRequest のメソッド 157

5 動的インタフェースとクラス（Java） 158

5.1 Any 159
5.1.1 Any のメソッド 159
5.1.2 Any の抽出メソッド 160
5.1.3 Any の挿入メソッド 161
5.2 ARG_IN 162

5.2.1 ARG_IN の変数 162
5.3 ARG_INOUT 163
5.3.1 ARG_INOUT の変数 163
5.4 ARG_OUT 164
5.4.1 ARG_OUT の変数 164
5.5 ContextList 165
5.5.1 IDL の定義 165
5.5.2 ContextList のメソッド 165
5.6 DynAny 167
5.6.1 注意事項 167
5.6.2 DynAny のメソッド 168
5.6.3 DynAny の抽出メソッド 169
5.6.4 DynAny の挿入メソッド 169
5.7 DynArray 171
5.7.1 注意事項 171
5.7.2 DynArray のメソッド 171
5.8 DynAnyFactory 172
5.8.1 注意事項 172
5.8.2 DynAnyFactory のメソッド 172
5.9 DynEnum 173
5.9.1 注意事項 173
5.9.2 DynEnum のメソッド 173
5.10 DynSequence 174
5.10.1 注意事項 174
5.10.2 DynSequence のメソッド 174
5.11 DynStruct 176
5.11.1 注意事項 176
5.11.2 DynStruct のメソッド 176
5.12 DynUnion 178
5.12.1 注意事項 178
5.12.2 DynUnion のメソッド 178
5.13 DynamicImplementation 180
5.13.1 DynamicImplementation のメソッド 180
5.14 Environment 181
5.14.1 Environment のメソッド 181
5.15 ExceptionList 182
5.15.1 IDL の定義 182
5.15.2 ExceptionList のメソッド 182
5.16 InputStream 184

5.16.1 InputStream のメソッド 184
5.17 NamedValue 186
5.17.1 IDL の定義 186
5.17.2 NamedValue のメソッド 186
5.18 NameValuePair 187
5.18.1 NameValuePair の変数 187
5.18.2 NameValuePair のコンストラクタ 187
5.19 NVList 188
5.19.1 IDL の定義 188
5.19.2 NVList のメソッド 188
5.20 OutputStream 190
5.20.1 OutputStream のメソッド 190
5.21 Request 192
5.21.1 IDL の定義 192
5.21.2 Request のメソッド 193
5.22 ServerRequest 196
5.22.1 IDL の定義 196
5.22.2 ServerRequest のメソッド 196
5.23 TCKind 198
5.23.1 IDL の定義 198
5.23.2 TCKind のメソッド 198
5.24 TypeCode 199
5.24.1 IDL の定義 199
5.24.2 TypeCode のメソッド 200
5.25 UnknownUserException 204

6 インタフェースリポジトリインタフェースとクラス（Java） 205

6.1 AbstractInterfaceDef 206
6.1.1 AbstractInterfaceDef のメソッド 206
6.2 AliasDef 209
6.2.1 AliasDef のメソッド 209
6.3 ArrayDef 210
6.3.1 ArrayDef のメソッド 210
6.4 AttributeDef 211
6.4.1 AttributeDef のメソッド 211
6.5 AttributeDescription 212
6.5.1 AttributeDescription の変数 212
6.5.2 AttributeDescription のメソッド 212
6.6 AttributeMode 214

6.6.1 AttributeMode の要素 214
6.7 ConstantDef 215
6.7.1 ConstantDef のメソッド 215
6.8 ConstantDescription 216
6.8.1 ConstantDescription の変数 216
6.8.2 ConstantDescription のメソッド 216
6.9 Contained 218
6.9.1 IDL の定義 218
6.9.2 Contained のメソッド 218
6.10 ContainedPackage.Description 221
6.10.1 ContainedPackage.Description の変数 221
6.10.2 ContainedPackage.Description のメソッド 221
6.11 Container 222
6.11.1 IDL の定義 222
6.11.2 Container のメソッド 224
6.12 ContainerPackage.Description 232
6.12.1 ContainerPackage.Description の変数 232
6.12.2 ContainerPackage.Description のメソッド 232
6.13 DefinitionKind 233
6.13.1 DefinitionKind のメソッド 233
6.13.2 DefinitionKind の列挙値 233
6.14 EnumDef 235
6.14.1 EnumDef のメソッド 235
6.15 ExceptionDef 236
6.15.1 ExceptionDef のメソッド 236
6.16 ExceptionDescription 237
6.16.1 ExceptionDescription の変数 237
6.16.2 ExceptionDescription のメソッド 237
6.17 FixedDef 239
6.17.1 FixedDef のメソッド 239
6.18 FullValueDescription 240
6.18.1 FullValueDescription の変数 240
6.18.2 FullValueDescription のメソッド 241
6.19 IDLType 243
6.19.1 IDL の定義 243
6.19.2 IDLType のメソッド 243
6.20 InterfaceDef 244
6.20.1 IDL の定義 244
6.20.2 InterfaceDef のメソッド 245

6.21 InterfaceDefPackage.FullInterfaceDescription 247
6.21.1 InterfaceDefPackage.FullInterfaceDescription の変数 247
6.21.2 InterfaceDefPackage.FullInterfaceDescription のメソッド 247
6.22 InterfaceDescription 249
6.22.1 InterfaceDescription の変数 249
6.22.2 InterfaceDescription のメソッド 249
6.23 IRObject 251
6.23.1 IDL の定義 251
6.23.2 IRObject のメソッド 251
6.24 LocalInterfaceDef 252
6.25 ModuleDef 253
6.26 ModuleDescription 254
6.26.1 ModuleDescription の変数 254
6.26.2 ModuleDescription のメソッド 254
6.27 NativeDef 256
6.28 OperationDef 257
6.28.1 OperationDef のメソッド 257
6.29 OperationDescription 259
6.29.1 OperationDescription の変数 259
6.29.2 OperationDescription のメソッド 259
6.30 OperationMode 261
6.31 ParameterDescription 262
6.31.1 ParameterDescription の変数 262
6.31.2 ParameterDescription のメソッド 262
6.32 ParameterMode 264
6.33 PrimitiveDef 265
6.33.1 PrimitiveDef のメソッド 265
6.34 PrimitiveKind 266
6.34.1 PrimitiveKind のメソッド 266
6.34.2 PrimitiveKind の定数 266
6.35 Repository 268
6.35.1 Repository のメソッド 268
6.36 SequenceDef 270
6.36.1 SequenceDef のメソッド 270
6.37 StringDef 271
6.37.1 StringDef のメソッド 271
6.38 StructDef 272
6.38.1 StructDef のメソッド 272
6.39 StructMember 273

6.39.1 StructMember の変数 273
6.39.2 StructMember のメソッド 273
6.40 TypedefDef 274
6.41 TypeDescription 275
6.41.1 TypeDescription の変数 275
6.41.2 TypeDescription のメソッド 275
6.42 UnionDef 277
6.42.1 UnionDef のメソッド 277
6.43 UnionMember 278
6.43.1 UnionMember の変数 278
6.43.2 UnionMember のメソッド 278
6.44 ValueBoxDef 280
6.44.1 ValueBoxDef のメソッド 280
6.45 ValueDef 281
6.45.1 ValueDef のメソッド 281
6.46 ValueDescription 285
6.46.1 ValueDescription の変数 285
6.46.2 ValueDescription のメソッド 285
6.47 ValueMemberDef 287
6.47.1 ValueMemberDef のメソッド 287
6.48 WstringDef 288

7 活性化インタフェースとクラス（Java） 289

7.1 ActivationImplDef 290
7.1.1 ActivationImplDef のメソッド 290
7.2 Activator 291
7.2.1 Activator のメソッド 291
7.3 CreationImplDef 293
7.3.1 IDL の定義 293
7.3.2 活性化ポリシー 293
7.3.3 例 294
7.3.4 環境変数 295
7.3.5 明示的に渡される環境変数と暗黙的に渡される環境変数 295
7.3.6 CreationImplDef のメソッド 296
7.4 ImplementationDef 299
7.5 OAD 300
7.5.1 ImplementationStatus 301
7.5.2 OAD のメソッド 301

8 ネーミングサービスインタフェースとクラス（Java） 307
8.1 NamingContext 308
8.1.1 IDL の定義 308
8.1.2 NamingContext のメソッド 308
8.2 NamingContextExt 315
8.2.1 IDL の定義 315
8.2.2 NamingContextExt のメソッド 315
8.3 Binding と BindingList 318
8.3.1 IDL の定義 318
8.4 BindingIterator 319
8.4.1 IDL の定義 319
8.4.2 BindingIterator のメソッド 319
8.5 NamingContextFactory 321
8.5.1 IDL の定義 321
8.5.2 NamingContextFactory のメソッド 321
8.6 ExtendedNamingContextFactory 323
8.6.1 IDL の定義 323
8.6.2 ExtendedNamingContextFactory のメソッド 323

9 例外クラス（Java） 324
9.1 概要 325
9.2 システム例外 326
9.2.1 SystemException の属性 326
9.3 ユーザ例外 328
9.3.1 import 文 328
9.3.2 UserException のコンストラクタ 328

10 ポータブルインタセプタインタフェースとクラス（Java） 329
10.1 概要 330
10.2 ClientRequestInfo 331
10.2.1 import 文 332
10.2.2 ClientRequestInfo のメソッド 332
10.3 ClientRequestInterceptor 335
10.3.1 import 文 335
10.3.2 ClientRequestInterceptor のメソッド 335
10.4 Codec 338
10.4.1 import 文 338
10.4.2 Codec のメンバ 338
10.4.3 Codec のメソッド 338

10.5 CodecFactory 340
10.5.1 import 文 340
10.5.2 CodecFactory のメンバ 340
10.5.3 CodecFactory のメソッド 340
10.6 Current 341
10.6.1 import 文 341
10.6.2 Current のメソッド 341
10.7 Encoding 343
10.7.1 import 文 343
10.7.2 Encoding のメンバ 343
10.8 ForwardRequest 344
10.8.1 import 文 344
10.8.2 ForwardRequest の変数 344
10.8.3 ForwardRequest のメソッド 344
10.9 Interceptor 346
10.9.1 import 文 346
10.9.2 Interceptor のメソッド 346
10.10 IORInfo 347
10.10.1 import 文 347
10.10.2 IORInfo のメソッド 347
10.11 IORInfoExt 350
10.11.1 import 文 350
10.11.2 IORInfoExt のメソッド 350
10.12 IORInterceptor 351
10.12.1 import 文 351
10.12.2 IORInterceptor のメソッド 351
10.13 ORBInitializer 353
10.13.1 import 文 353
10.13.2 ORBInitializer のメソッド 353
10.14 ORBInitInfo 354
10.14.1 import 文 354
10.14.2 ORBInitInfo のメンバ 354
10.14.3 ORBInitInfo のメソッド 354
10.15 Parameter 358
10.15.1 import 文 358
10.15.2 Parameter のメンバ 358
10.16 PolicyFactory 359
10.16.1 import 文 359
10.16.2 PolicyFactory のメソッド 359

10.17 RequestInfo 360
10.17.1 import 文 360
10.17.2 RequestInfo のメソッド 360
10.18 ServerRequestInfo 364
10.18.1 import 文 365
10.18.2 ServerRequestInfo のメソッド 365
10.19 ServerRequestInterceptor 368
10.19.1 import 文 368
10.19.2 ServerRequestInterceptor のメソッド 368

11 VisiBroker 4.x インタセプタおよびオブジェクトラッパーのインタフェースとクラス（Java） 371
11.1 概要 372
11.2 インタセプタマネージャ 373
11.3 IOR テンプレート 374
11.4 InterceptorManager 375
11.5 InterceptorManagerControl 376
11.5.1 import 文 376
11.5.2 InterceptorManagerControl のメソッド 376
11.6 BindInterceptor 377
11.6.1 import 文 377
11.6.2 BindInterceptor のメソッド 377
11.7 BindInterceptorManager 380
11.7.1 import 文 380
11.7.2 BindInterceptorManager のメソッド 380
11.8 ClientRequestInterceptor 381
11.8.1 import 文 381
11.8.2 ClientRequestInterceptor のメソッド 381
11.9 ClientRequestInterceptorManager 384
11.9.1 import 文 384
11.9.2 ClientRequestInterceptorManager のメソッド 384
11.10 POALifeCycleInterceptor 385
11.10.1 import 文 385
11.10.2 POALifeCycleInterceptor のメソッド 385
11.11 POALifeCycleInterceptorManager 387
11.11.1 import 文 387
11.11.2 POALifeCycleInterceptorManager のメソッド 387
11.12 ActiveObjectLifeCycleInterceptor 388
11.12.1 import 文 388

11.12.2 ActiveObjectLifeCycleInterceptor のメソッド 388
11.13 ActiveObjectLifeCycleInterceptorManager 390
11.13.1 import 文 390
11.13.2 ActiveObjectLifeCycleInterceptorManager のメソッド 390
11.14 ForwardRequestException 391
11.14.1 ForwardRequestException の変数 391
11.15 ServerRequestInterceptor 392
11.15.1 import 文 392
11.15.2 ServerRequestInterceptor のメソッド 392
11.16 ServerRequestInterceptorManager 395
11.16.1 import 文 395
11.16.2 ServerRequestInterceptorManager のメソッド 395
11.17 IORCreationInterceptor 396
11.17.1 import 文 396
11.17.2 IORCreationInterceptor のメソッド 396
11.18 IORCreationInterceptorManager 397
11.18.1 import 文 397
11.18.2 IORCreationInterceptorManager のメソッド 397
11.19 Location 398
11.19.1 import 文 398
11.19.2 IDL の定義 398
11.19.3 Location のメンバ 398
11.20 Closure 399
11.21 ExtendedClosure 400
11.22 ChainUntypedObjectWrapperFactory 401
11.22.1 import 文 401
11.22.2 ChainUntypedObjectWrapperFactory のメソッド 401
11.23 UntypedObjectWrapper 403
11.23.1 UntypedObjectWrapper のメソッド 403
11.24 UntypedObjectWrapperFactory 405
11.24.1 import 文 405
11.24.2 UntypedObjectWrapperFactory のメソッド 405

12 QoS インタフェースとクラス（Java） 406

12.1 概要 407
12.2 PolicyManager 408
12.2.1 IDL の定義 408
12.2.2 PolicyManager のメソッド 408
12.3 PolicyCurrent 410

12.3.1 IDL の定義 410
12.4 Object 411
12.4.1 org.omg.CORBA.Object のメソッド 411
12.4.2 com.inprise.vbroker.CORBA.Object のメソッド 412
12.5 RebindPolicy 413
12.5.1 IDL の定義 413
12.5.2 ポリシーの値 414
12.6 RebindForwardPolicy 416
12.6.1 IDL の定義 416
12.7 RelativeConnectionTimeoutPolicy 417
12.7.1 IDL の定義 417
12.8 Messaging.RelativeRequestTimeoutPolicy 418
12.8.1 IDL の定義 418
12.9 Messaging.RelativeRoundtripTimeoutPolicy 419
12.9.1 IDL の定義 419
12.10 DeferBindPolicy 420
12.10.1 IDL の定義 420
12.11 ExclusiveConnectionPolicy 421
12.11.1 IDL の定義 421
12.12 SyncScopePolicy 422
12.12.1 IDL の定義 422
12.12.2 SyncScope ポリシーの値 422
12.13 QoS 例外 424

13 IOP および IIOP のインタフェースとクラス（Java） 425
13.1 IIOP.ProfileBody 426
13.1.1 IDL の定義 426
13.1.2 IIOP.ProfileBody の変数 426
13.1.3 IIOP.ProfileBody のコンストラクタ 427
13.2 IOP.IORValue 428
13.2.1 IDL の定義 428
13.2.2 IOP.IORValue の変数 428
13.2.3 IOP.IORValue のメソッド 428
13.3 IOP.ServiceContext 430
13.3.1 IDL の定義 430
13.3.2 IOP.ServiceContext の変数 430
13.3.3 IOP.ServiceContext のコンストラクタ 430
13.4 IOP.TaggedProfile 431
13.4.1 IDL の定義 431

13.4.2 IOP.TaggedProfile の変数 431
13.4.3 IOP.TaggedProfile のコンストラクタ 431

14 RMI インタフェースとクラス（Java） 433
14.1 PortableRemoteObject 434
14.1.1 PortableRemoteObject のコンストラクタ 434
14.1.2 PortableRemoteObject のメソッド 434

15 URL ネーミングインタフェースとクラス（Java） 436
15.1 Resolver 437
15.1.1 Resolver のメソッド 437

16 ロケーションサービスインタフェースとクラス（Java） 439
16.1 Agent 440
16.1.1 IDL の定義 440
16.1.2 Agent のメソッド 440
16.2 Desc 444
16.2.1 IDL の定義 444
16.2.2 Desc の変数 444
16.2.3 Desc のコンストラクタ 444
16.2.4 Desc のメソッド 445
16.3 Fail 446
16.3.1 Fail 変数 446
16.4 TriggerDesc 447
16.4.1 IDL の定義 447
16.4.2 TriggerDesc の変数 447
16.4.3 TriggerDesc のコンストラクタ 447
16.4.4 TriggerDesc のメソッド 448
16.5 TriggerHandler 449
16.5.1 IDL の定義 449
16.5.2 TriggerHandler のメソッド 449

17 コマンドラインオプション（Java） 450
17.1 ORB オプションの設定方法 451
17.1.1 vbj コマンドでコマンドラインパラメタを使用する 451
17.1.2 vbj コマンドを使用して Java アプリケーションを起動する 451
17.1.3 アプレットにオプションを設定する 451
17.1.4 メソッドでプログラム的にオプションを設定する 452
17.2 ORB.init( )メソッド 453
17.2.1 ORB オプション 453

17.3 ロケーションサービスオプション 457

18 Borland Enterprise Server VisiBroker プロパティ（Java） 458
18.1 プロパティの設定方法 459
18.1.1 vbj コマンドでコマンドラインパラメタを使用する 459
18.1.2 vbj コマンドを使用して Java アプリケーションを起動する 459
18.1.3 アプレットにプロパティを設定する 459
18.1.4 メソッドでプログラム的にプロパティを設定する 460
18.2 RMI-IIOP プロパティ 461
18.3 osagent（スマートエージェント）プロパティ 462
18.4 ORB プロパティ 464
18.5 POA プロパティ 469
18.6 ロケーションサービスプロパティ 470
18.7 ネーミングサービスプロパティ 471
18.8 OAD プロパティ 472
18.9 インタフェースリポジトリプロパティ 473
18.10 URL ネーミングプロパティ 474
18.11 クライアント側コネクションプロパティ 475
18.12 クライアント側プロセス内コネクションプロパティ 476
18.13 サーバ側エンジンプロパティ 477
18.14 サーバ側スレッドセッション IIOP_TS プロパティ，および IIOP_TS コネクションプロパティ 478
18.15 サーバ側スレッドプール IIOP_TP プロパティ，および IIOP_TP コネクションプロパティ 479
18.16 双方向通信をサポートするプロパティ 481

第 2 編 C++言語編

19 プログラマツール（C++） 482
19.1 引数とオプション 483
19.1.1 一般オプション 483
19.1.2 プログラマツールの動作環境 483
19.2 idl2cpp 485
19.2.1 構文 485
19.2.2 説明 485
19.3 idl2ir 489
19.3.1 構文 489
19.3.2 説明 489
19.4 ir2idl 491
19.4.1 構文 491
19.4.2 説明 491

20 IDL から C++言語へのマッピング 492
20.1 基本データ型 493
20.2 文字列 494
20.2.1 String_var クラス 494
20.2.2 WString_var クラス 495
20.3 定数 496
20.3.1 定数を含む特別なケース 496
20.4 列挙体 498
20.5 型定義 499
20.6 module 句 501
20.7 複合データ型 502
20.8 構造体 503
20.8.1 固定長構造体 503
20.8.2 可変長構造体 504
20.8.3 union 505
20.8.4 Sequence 507
20.8.5 配列 510
20.8.6 Principal 512
20.9 Valuetype 513
20.9.1 Valuebox 516
20.10 抽象インタフェース 517

21 生成されるインタフェースとクラス（C++） 519
21.1 概要 520
21.2 <interface_name> 521
21.3 <interface_name>ObjectWrapper 523
21.4 POA_<class_name> 525
21.5 _tie_<class_name> 526
21.6 <class_name>_var 527

22 コアインタフェースとクラス（C++） 528
22.1 PortableServer::AdapterActivator 529
22.1.1 インクルードファイル 529
22.1.2 PortableServer::AdapterActivator のメソッド 529
22.2 PortableServer 530
22.2.1 インクルードファイル 530
22.2.2 PortableServer のメソッド 530
22.3 CompletionStatus 531
22.3.1 CompletionStatus のメンバ 531

22.4 CORBA 532
22.4.1 インクルードファイル 532
22.4.2 CORBA のメソッド 532
22.5 Context 534
22.5.1 インクルードファイル 534
22.5.2 Context のメソッド 534
22.6 PortableServer::Current 537
22.6.1 インクルードファイル 537
22.6.2 PortableServer::Current のメソッド 537
22.7 Exception 538
22.7.1 インクルードファイル 538
22.8 Object 539
22.8.1 インクルードファイル 539
22.8.2 CORBA::Object のメソッド 539
22.8.3 CORBA::Object に対する VisiBroker の拡張機能 542
22.9 ORB 544
22.9.1 インクルードファイル 544
22.9.2 CORBA::ORB のメソッド 544
22.9.3 CORBA::ORB に対する VisiBroker の拡張機能 550
22.10 Policy 553
22.10.1 インクルードファイル 553
22.10.2 Policy のメソッド 553
22.11 PortableServer::POA 554
22.11.1 インクルードファイル 554
22.11.2 PortableServer::POA のメソッド 554
22.12 PortableServer::POAManager 564
22.12.1 インクルードファイル 564
22.12.2 PortableServer::POAManager のメソッド 565
22.13 PortableServer::POAManager::State 567
22.13.1 インクルードファイル 567
22.13.2 PortableServer::POAManager::State のメソッド 567
22.14 PortableServer::RefCountServantBase 568
22.14.1 インクルードファイル 568
22.14.2 PortableServer::RefCountServantBase のメソッド 568
22.15 PortableServer::ServantActivator 569
22.15.1 インクルードファイル 569
22.15.2 PortableServer::ServantActivator のメソッド 569
22.16 PortableServer::ServantBase 571
22.16.1 インクルードファイル 571

22.16.2 PortableServer::ServantBase のメソッド 571
22.17 PortableServer::ServantLocator 572
22.17.1 インクルードファイル 572
22.17.2 PortableServer::ServantLocator のメソッド 572
22.18 PortableServer::ServantManager 574
22.18.1 インクルードファイル 574
22.19 PortableServer::ForwardRequest 575
22.19.1 インクルードファイル 575
22.19.2 PortableServer::ForwardRequest のメソッド 575
22.20 SystemException 576
22.20.1 インクルードファイル 576
22.20.2 SystemException のメソッド 576
22.21 UserException 579
22.22 VISPropertyManager 580
22.22.1 インクルードファイル 580
22.22.2 VISPropertyManager のメソッド 580

23 動的インタフェースとクラス（C++） 582

23.1 Any 583
23.1.1 インクルードファイル 583
23.1.2 Any のメソッド 583
23.1.3 初期化演算子 584
23.1.4 抽出演算子 585
23.2 ContextList 586
23.2.1 ContextList のメソッド 586
23.3 DynamicImplementation 588
23.3.1 DynamicImplementation のメソッド 588
23.4 DynAny 589
23.4.1 インクルードファイル 589
23.4.2 注意事項 590
23.4.3 DynAny のメソッド 590
23.4.4 DynAny の抽出メソッド 591
23.4.5 DynAny の挿入メソッド 592
23.5 DynAnyFactory 593
23.5.1 DynAnyFactory のメソッド 593
23.6 DynArray 594
23.6.1 注意事項 594
23.6.2 DynArray のメソッド 594
23.7 DynEnum 596

23.7.1 注意事項 596
23.7.2 DynEnum のメソッド 596
23.8 DynSequence 598
23.8.1 注意事項 598
23.8.2 DynSequence のメソッド 598
23.9 DynStruct 600
23.9.1 注意事項 600
23.9.2 DynStruct のメソッド 600
23.10 DynUnion 602
23.10.1 注意事項 602
23.10.2 DynUnion のメソッド 602
23.11 Environment 604
23.11.1 インクルードファイル 604
23.11.2 Environment のメソッド 604
23.12 ExceptionList 606
23.12.1 ExceptionList のメソッド 606
23.13 NamedValue 608
23.13.1 インクルードファイル 608
23.13.2 NamedValue のメソッド 608
23.14 NVList 610
23.14.1 インクルードファイル 610
23.14.2 NVList のメソッド 610
23.15 Request 614
23.15.1 インクルードファイル 614
23.15.2 Request のメソッド 614
23.16 ServerRequest 618
23.16.1 インクルードファイル 618
23.16.2 ServerRequest のメソッド 618
23.17 TCKind 620
23.18 TypeCode 622
23.18.1 インクルードファイル 622
23.18.2 TypeCode のコンストラクタ 622
23.18.3 TypeCode のメソッド 623

24 インタフェースリポジトリインタフェースとクラス（C++） 627
24.1 AliasDef 628
24.1.1 AliasDef のメソッド 628
24.2 ArrayDef 629
24.2.1 ArrayDef のメソッド 629

24.3 AttributeDef 630
24.3.1 AttributeDef のメソッド 630
24.4 AttributeDescription 631
24.4.1 AttributeDescription のメンバ 631
24.5 AttributeMode 632
24.5.1 AttributeMode の値 632
24.6 ConstantDef 633
24.6.1 ConstantDef のメソッド 633
24.7 ConstantDescription 634
24.7.1 ConstantDescription のメンバ 634
24.8 Contained 635
24.8.1 インクルードファイル 635
24.8.2 Contained のメソッド 635
24.9 Container 638
24.9.1 インクルードファイル 638
24.9.2 Container のメソッド 638
24.10 DefinitionKind 646
24.10.1 DefinitionKind の列挙値 646
24.11 Description 648
24.11.1 Description のメンバ 648
24.12 EnumDef 649
24.12.1 EnumDef のメソッド 649
24.13 ExceptionDef 650
24.13.1 ExceptionDef のメソッド 650
24.14 ExceptionDescription 651
24.14.1 ExceptionDescription のメンバ 651
24.15 FixedDef 652
24.15.1 FixedDef のメソッド 652
24.16 FullInterfaceDescription 653
24.16.1 FullInterfaceDescription のメンバ 653
24.17 FullValueDescription 654
24.17.1 FullValueDescription の変数 654
24.18 IDLType 656
24.18.1 インクルードファイル 656
24.18.2 IDLType のメソッド 656
24.19 InterfaceDef 657
24.19.1 インクルードファイル 657
24.19.2 InterfaceDef のメソッド 658
24.20 InterfaceDescription 661

24.20.1 InterfaceDescription のメンバ 661
24.21 IRObject 662
24.21.1 インクルードファイル 662
24.21.2 IRObject のメソッド 662
24.22 ModuleDef 663
24.23 ModuleDescription 664
24.23.1 ModuleDescription のメンバ 664
24.24 NativeDef 665
24.25 OperationDef 666
24.25.1 インクルードファイル 666
24.25.2 OperationDef のメソッド 666
24.26 OperationDescription 669
24.26.1 OperationDescription のメンバ 669
24.27 OperationMode 670
24.27.1 OperationMode の値 670
24.28 ParameterDescription 671
24.28.1 ParameterDescription のメンバ 671
24.29 ParameterMode 672
24.29.1 ParameterMode の値 672
24.30 PrimitiveDef 673
24.30.1 PrimitiveDef のメソッド 673
24.31 PrimitiveKind 674
24.31.1 PrimitiveKind の定数 674
24.32 Repository 675
24.32.1 インクルードファイル 675
24.32.2 Repository のメソッド 675
24.33 SequenceDef 678
24.33.1 SequenceDef のメソッド 678
24.34 StringDef 679
24.34.1 StringDef のメソッド 679
24.35 StructDef 680
24.35.1 StructDef のメソッド 680
24.36 StructMember 681
24.36.1 StructMember のメンバ 681
24.37 TypedefDef 682
24.38 TypeDescription 683
24.38.1 TypeDescription のメンバ 683
24.39 UnionDef 684
24.39.1 UnionDef のメソッド 684

24.40 UnionMember 685
24.40.1 UnionMember のメンバ 685
24.41 ValueBoxDef 686
24.41.1 ValueBoxDef のメソッド 686
24.42 ValueDef 687
24.42.1 ValueDef のメソッド 687
24.43 ValueDescription 691
24.43.1 ValueDescription の変数 691
24.44 WstringDef 692
24.44.1 WstringDef のメソッド 692

25 活性化インタフェースとクラス（C++） 693
25.1 CreationImplDef 694
25.1.1 インクルードファイル 694
25.1.2 C++のサンプル 694
25.1.3 環境変数 695
25.1.4 CreationImplDef のメンバ 695
25.2 ImplementationDef 697
25.2.1 インクルードファイル 697
25.3 ImplementationStatus 698
25.3.1 インクルードファイル 698
25.3.2 ImplementationStatus のメンバ 698
25.4 OAD 699
25.4.1 インクルードファイル 700
25.4.2 OAD のメソッド 700
25.5 ObjectStatus 704
25.5.1 インクルードファイル 704
25.5.2 ObjectStatus のメンバ 704
25.6 ObjectStatusList 705
25.6.1 インクルードファイル 705
25.6.2 ObjectStatusList のメソッド 705
25.7 StringSequence 706
25.7.1 インクルードファイル 706
25.7.2 StringSequence のメソッド 706
25.7.3 StringSequence に関連するメソッド 707

26 ネーミングサービスインタフェースとクラス（C++） 709
26.1 NamingContext 710
26.1.1 NamingContext のメソッド 710

26.2 NamingContextExt 716
26.2.1 NamingContextExt のメソッド 716
26.3 Binding と BindingList 719
26.4 BindingIterator 720
26.4.1 BindingIterator のメソッド 720
26.5 NamingContextFactory 722
26.5.1 NamingContextFactory のメソッド 722
26.6 ExtendedNamingContextFactory 724
26.6.1 ExtendedNamingContextFactory のメソッド 724

27 ポータブルインタセプタインタフェースとクラス（C++） 725
27.1 概要 726
27.2 ClientRequestInfo 727
27.2.1 インクルードファイル 728
27.2.2 ClientRequestInfo のメソッド 728
27.3 ClientRequestInterceptor 731
27.3.1 インクルードファイル 731
27.3.2 ClientRequestInterceptor のメソッド 731
27.4 Codec 734
27.4.1 インクルードファイル 734
27.4.2 Codec のメンバ 734
27.4.3 Codec のメソッド 734
27.5 CodecFactory 736
27.5.1 インクルードファイル 736
27.5.2 CodecFactory のメンバ 736
27.5.3 CodecFactory のメソッド 736
27.6 Current 737
27.6.1 インクルードファイル 737
27.6.2 Current のメソッド 737
27.7 Encoding 739
27.7.1 インクルードファイル 739
27.7.2 Encoding のメンバ 739
27.8 ExceptionList 740
27.8.1 インクルードファイル 740
27.9 ForwardRequest 741
27.9.1 インクルードファイル 741
27.10 Interceptor 742
27.10.1 インクルードファイル 742
27.10.2 Interceptor のメソッド 742

27.11 IORInfo 743
27.11.1 インクルードファイル 743
27.11.2 IORInfo のメソッド 743
27.12 IORInfoExt 746
27.12.1 インクルードファイル 746
27.12.2 IORInfoExt のメソッド 746
27.13 IORInterceptor 747
27.13.1 インクルードファイル 747
27.13.2 IORInterceptor のメソッド 747
27.14 ORBInitializer 748
27.14.1 インクルードファイル 748
27.14.2 ORBInitializer のメソッド 748
27.15 ORBInitInfo 749
27.15.1 インクルードファイル 749
27.15.2 ORBInitInfo のメンバ 749
27.15.3 ORBInitInfo のメソッド 749
27.16 Parameter 752
27.16.1 インクルードファイル 752
27.16.2 Parameter のメンバ 752
27.17 ParameterList 753
27.17.1 インクルードファイル 753
27.18 PolicyFactory 754
27.18.1 インクルードファイル 754
27.18.2 PolicyFactory のメソッド 754
27.19 RequestInfo 755
27.19.1 インクルードファイル 755
27.19.2 RequestInfo のメソッド 755
27.20 ServerRequestInfo 759
27.20.1 インクルードファイル 760
27.20.2 ServerRequestInfo のメソッド 760
27.21 ServerRequestInterceptor 763
27.21.1 インクルードファイル 763
27.21.2 ServerRequestInterceptor のメソッド 763

28 VisiBroker 4.x インタセプタおよびオブジェクトラッパーのインタフェースとクラス（C++） 766

28.1 概要 767
28.2 インタセプタマネージャ 768
28.3 IOR テンプレート 769

28.4 InterceptorManager 770
28.5 InterceptorManagerControl 771
28.5.1 インクルードファイル 771
28.5.2 InterceptorManagerControl のメソッド 771
28.6 BindInterceptor 772
28.6.1 インクルードファイル 772
28.6.2 BindInterceptor のメソッド 772
28.7 BindInterceptorManager 775
28.7.1 インクルードファイル 775
28.7.2 BindInterceptorManager のメソッド 775
28.8 ClientRequestInterceptor 776
28.8.1 インクルードファイル 776
28.8.2 ClientRequestInterceptor のメソッド 776
28.9 ClientRequestInterceptorManager 779
28.9.1 インクルードファイル 779
28.9.2 ClientRequestInterceptorManager のメソッド 779
28.10 POALifeCycleInterceptor 780
28.10.1 インクルードファイル 780
28.10.2 POALifeCycleInterceptor のメソッド 780
28.11 POALifeCycleInterceptorManager 782
28.11.1 インクルードファイル 782
28.11.2 POALifeCycleInterceptorManager のメソッド 782
28.12 ActiveObjectLifeCycleInterceptor 783
28.12.1 インクルードファイル 783
28.12.2 ActiveObjectLifeCycleInterceptor のメソッド 783
28.13 ActiveObjectLifeCycleInterceptorManager 785
28.13.1 インクルードファイル 785
28.13.2 ActiveObjectLifeCycleInterceptorManager のメソッド 785
28.14 ForwardRequestException 786
28.14.1 ForwardRequestException の変数 786
28.15 ServerRequestInterceptor 787
28.15.1 インクルードファイル 787
28.15.2 ServerRequestInterceptor のメソッド 787
28.16 ServerRequestInterceptorManager 790
28.16.1 インクルードファイル 790
28.16.2 ServerRequestInterceptorManager のメソッド 790
28.17 IORCreationInterceptor 791
28.17.1 インクルードファイル 791
28.17.2 IORCreationInterceptor のメソッド 791

28.18 IORCreationInterceptorManager 792
28.18.1 インクルードファイル 792
28.18.2 IORCreationInterceptorManager のメソッド 792
28.19 ExtendedClosure 793
28.20 VISClosure 794
28.20.1 インクルードファイル 794
28.20.2 VISClosure のメンバ 794
28.21 VISClosureData 795
28.22 ChainUntypedObjectWrapperFactory 796
28.22.1 インクルードファイル 796
28.22.2 ChainUntypedObjectWrapperFactory のメソッド 796
28.23 UntypedObjectWrapper 799
28.23.1 インクルードファイル 799
28.23.2 UntypedObjectWrapper のメソッド 799
28.24 UntypedObjectWrapperFactory 801
28.24.1 インクルードファイル 801
28.24.2 UntypedObjectWrapperFactory のコンストラクタ 801
28.24.3 UntypedObjectWrapperFactory のメソッド 802
28.25 EventQueueManager 803
28.25.1 インクルードファイル 803
28.25.2 EventQueueManager のメソッド 803
28.26 ConnEventListeners 804
28.26.1 インクルードファイル 804
28.26.2 ConnEventListeners のメソッド 804
28.27 ConnInfo 805
28.27.1 インクルードファイル 805
28.27.2 ConnInfo のメンバ 805

29 QoS インタフェースとクラス（C++） 806
29.1 概要 807
29.2 CORBA::PolicyManager 808
29.2.1 IDL の定義 808
29.2.2 CORBA::PolicyManager のメソッド 808
29.3 CORBA::PolicyCurrent 810
29.3.1 IDL の定義 810
29.4 CORBA::Object 811
29.4.1 IDL の定義 811
29.4.2 CORBA::Object のメソッド 811
29.5 Messaging::RebindPolicy 814

29.5.1 IDL の定義 814
29.5.2 ポリシーの値 814
29.6 QoSExt::DeferBindPolicy 816
29.6.1 IDL の定義 816
29.7 QoSExt::ExclusiveConnectionPolicy 817
29.7.1 IDL の定義 817
29.8 QoSExt::RelativeConnectionTimeoutPolicy 818
29.8.1 IDL の定義 818
29.9 Messaging::RelativeRequestTimeoutPolicy 819
29.9.1 IDL の定義 819
29.10 Messaging::RelativeRoundtripTimeoutPolicy 820
29.10.1 IDL の定義 821
29.11 Messaging::SyncScopePolicy 822
29.11.1 IDL の定義 822
29.11.2 SyncScope ポリシーの値 822
29.12 QoS 例外 824

30 IOP および IIOP のインタフェースとクラス（C++） 825
30.1 GIOP::MessageHeader 826
30.1.1 MessageHeader のメンバ 826
30.2 GIOP::CancelRequestHeader 827
30.2.1 CancelRequestHeader のメンバ 827
30.3 GIOP::LocateReplyHeader 828
30.3.1 LocateReplyHeader のメンバ 828
30.4 GIOP::LocateRequestHeader 829
30.4.1 LocateRequestHeader のメンバ 829
30.5 GIOP::ReplyHeader 830
30.5.1 インクルードファイル 830
30.5.2 ReplyHeader のメンバ 830
30.6 GIOP::RequestHeader 831
30.6.1 インクルードファイル 831
30.6.2 RequestHeader のメンバ 831
30.7 IIOP::ProfileBody 832
30.7.1 ProfileBody のメンバ 832
30.8 IOP::IOR 833
30.8.1 インクルードファイル 833
30.8.2 IOR のメンバ 833
30.9 IOP::TaggedProfile 834
30.9.1 TaggedProfile のメンバ 834

31 マーシャルバッファインタフェースとクラス（C++） 835
31.1 CORBA::MarshalInBuffer 836
31.1.1 インクルードファイル 836
31.1.2 CORBA::MarshalInBuffer のコンストラクタとデストラクタ 836
31.1.3 CORBA::MarshalInBuffer のメソッド 837
31.1.4 CORBA::MarshalInBuffer の演算子 840
31.2 CORBA::MarshalOutBuffer 841
31.2.1 インクルードファイル 841
31.2.2 CORBA::MarshalOutBuffer のコンストラクタとデストラクタ 841
31.2.3 CORBA::MarshalOutBuffer のメソッド 842
31.2.4 CORBA::MarshalOutBuffer の演算子 844

32 ロケーションサービスインタフェースとクラス（C++） 845
32.1 Agent 846
32.1.1 IDL の定義 846
32.1.2 インクルードファイル 847
32.1.3 Agent のメソッド 847
32.2 Desc 852
32.2.1 IDL の定義 852
32.2.2 Desc のメンバ 852
32.3 Fail 854
32.3.1 Fail のメンバ 854
32.4 TriggerDesc 855
32.4.1 IDL の定義 855
32.4.2 TriggerDesc のメンバ 855
32.5 TriggerHandler 856
32.5.1 IDL の定義 856
32.5.2 インクルードファイル 856
32.5.3 TriggerHandler のメソッド 856
32.6 <type>Seq 857
32.6.1 <type>Seq のメソッド 857
32.7 <type>SeqSeq 859
32.7.1 <type>SeqSeq のメソッド 859

33 初期化インタフェースとクラス（C++） 861
33.1 VISInit 862
33.1.1 インクルードファイル 862
33.1.2 VISInit のメソッド 862

34 コマンドラインオプション（C++） 864
34.1 ORB_init()メソッド 865
34.1.1 ORB オプション 865
34.2 ロケーションサービスオプション 868

35 Borland Enterprise Server VisiBroker プロパティ（C++） 869
35.1 プロパティの設定方法 870
35.2 osagent（スマートエージェント）プロパティ 871
35.3 ORB プロパティ 873
35.4 ロケーションサービスプロパティ 876
35.5 OAD プロパティ 877
35.6 インタフェースリポジトリリゾルバのプロパティ 878
35.7 ネーミングサービスプロパティ 879
35.8 TypeCode のプロパティ 880
35.9 クライアント側 IIOP のコネクションプロパティ 881
35.10 サーバ側エンジンプロパティ 882
35.11 サーバ側スレッドセッションコネクションのプロパティ 883
35.12 サーバ側スレッドプールコネクションのプロパティ 885
35.13 双方向通信をサポートするプロパティ 887

付録 888
付録 A このマニュアルの参考情報 889
付録 A.1 関連マニュアル 889
付録 A.2 このマニュアルでの表記 889
付録 A.3 英略語 891

索引 893

1 プログラマツール（Java）

この章では，Borland Enterprise Server VisiBroker が提供する Java 言語用のプログラマツールについて説明します。

コマンドとキーワードで構成される構文を中心に説明します。キーワードとは，オプションとファイル名で構成されるものを指します。コマンドとオプションだけで構成される構文は，この章では説明しません。ただし，vbj と vbjc は，構文上はコマンドとオプションだけで構成されますが，この章で説明します。

1.1 オプション

Borland Enterprise Server VisiBroker のプログラマツールには一般オプションと個別オプションの両方が設定できます。ツール固有の個別オプションについては，ツールごとに説明します。

記載されているオプションはすべてハイフンで始まり，デフォルトで有効になります。無効にするには，-[no_]を使用するか，ハイフンを削除します。例えば，次のオプションは，#pragma が認識されない場合に警告を表示するときのデフォルトです。

```
warn_unrecognized_pragmas
```

デフォルトを無効にするには，次のオプションを指定します。

```
-no_warn_unrecognized_pragmas
```

1.1.1 一般オプション

次に示すオプションはすべてのプログラマツールに共通のオプションです。

オプション

- -J<*java_option*>
  java_option を直接 Java VM に渡します。
- -VBJversion
  Borland Enterprise Server VisiBroker のバージョンを出力します。
- -VBJdebug
  Borland Enterprise Server VisiBroker のデバッグ情報を出力します。
- -VBJclasspath
  CLASSPATH 環境変数の前にクラスパスを指定します。
- -VBJprop <*name*> [=<*value*>]
  対になった名前と値を Java VM に渡します。
- -VBJjavavm <*jvmpath*>
  Java VM へのパスを指定します。
- -VBJaddJar <*jarfile*>
  Java VM を実行する前に CLASSPATH に jarfile を付けます。

1.2 idl2ir

このコマンドは，インタフェース定義言語（IDL）ソースファイルで定義されたオブジェクトをインタフェースリポジトリに格納します。

1.2.1 構文

```
idl2ir [options] {filename}
```

例

```
idl2ir -irep my_repository -replace java_examples/bank/Bank.idl
```

1.2.2 説明

idl2ir コマンドは入力として IDL ファイルを使用します。idl2ir コマンドは，インタフェースリポジトリサーバにバインドして，idl filename に指定した IDL でリポジトリを生成します。リポジトリに IDL ファイルの項目と同じ名前の項目がある場合，古い項目は変更されます。

キーワード

キーワードは，次のオプションと処理対象の IDL 入力ファイルの両方を含みます。

オプション

- -D, -define *foo*[=*bar*]
  foo に指定したプリプロセサマクロを定義します。bar で値を指定することもできます。
- -I, -include <*dir*>
  インクルードファイルを検索するディレクトリを指定します。
- -P, -no_line_directives
  行番号情報の生成を抑止します。デフォルトは off です。
- -H, -list_includes
  インクルードファイルのパスを標準出力に出力します。デフォルトは off です。
- -C, -retain_comments
  Java コードが生成されるときに，IDL ファイルのコメントを保持します。コメントを保持しない場合，コメントは Java コードで表示されません。
  コメントは次の形式で記述してください。ほかの形式で記述した場合，コメントが反映されません。

（例）

/**
（コメント）

*/

- -U, -undefine *foo*
  foo に指定したプリプロセサマクロの定義を解除します。
- -[no_]back_compat_mapping
  VisiBroker 3.x 互換のマッピングを使用することを指定します。
- -[no_]idl_strict
  IDL ソースに対して厳密に OMG 標準規格を適用することを指定します。デフォルトは off です。
- -[no_]preprocess
  解析前に IDL ファイルの前処理をします。デフォルトは on です。
- -[no_]preprocess_only
  前処理の終了後に，IDL ファイルの解析を中止します。このオプションを使用すると，コンパイラで前処理フェーズの結果を stdout に生成できます。デフォルトは on です。
- -[no_]warn_all
  警告をすべて抑止します。デフォルトは off です。
- -[no_]warn_unrecognized_pragmas
  #pragma が認識されない場合に警告を表示します。デフォルトは on です。
- -deep
  ディープマージを適用します。デフォルトは off です。ディープマージはシャロウマージに対する言葉です。
- -h, -help, -usage, -?
  ヘルプ情報を出力します。
- -irep <*irep name*>
  インタフェースリポジトリの名前を指定します。
- -replace
  マージしないでリポジトリ全体を置き換えます。デフォルトは off です。
- -version
  Borland Enterprise Server VisiBroker のバージョンを表示します。
- file1 [file2] ...
  処理対象のファイルを一つ以上指定します。処理対象のファイルに stdin を指定するときは，「-」を指定します。

このほか，idl2ir のオプションとして，vbj のオプションを使用できます。vbj のオプションについては，「1.7　vbj」を参照してください。

1.3 ir2idl

このコマンドは，インタフェースリポジトリから取得したオブジェクトで IDL ソースファイルを作成します。

1.3.1 構文

```
ir2idl [options] filename
```

例

```
ir2idl -irep my_repository -o my_file
```

1.3.2 説明

ir2idl コマンドはインタフェースリポジトリにバインドしてその内容を IDL 形式で出力します。

キーワード

キーワードは，次のオプションを含みます。

オプション

- `-irep <irep name>`
  インタフェースリポジトリの名前を指定します。
- `-o <file>`
  出力ファイルの名前を指定します。または，標準出力に出力させる場合は「`-`」を指定します。
- `-strict`
  OMG の規格に厳密に従ってコード生成することを指定します。デフォルトは off です。
- `-version`
  Borland Enterprise Server VisiBroker のバージョン番号を表示します。
- `-h, -help, -usage, -?`
  ヘルプ情報を出力します。

1.4 idl2java

このコマンドは，IDL ソースファイルから Java ソースコードを生成します。

生成したスタブ・スケルトンを，J2EE サーバプロセス上で使用する場合は，Cosminexus Component Container の「J2EE アプリケーション実行時間の監視と J2EE アプリケーションの強制停止機能」※1 の対象外として，明示的に指定する必要があります。次のファイル※2 に，生成したスタブ・スケルトンをフルパッケージ指定で記述してください。

ファイル格納先：

<Cosminexus インストールディレクトリ>¥CC¥server¥usrconf¥criticalList.cfg

注※1

詳細については，「Cosminexus V11 アプリケーションサーバ 機能解説 運用／監視／連携編」を参照してください。

注※2

詳細については，「Cosminexus V11 アプリケーションサーバ リファレンス 定義編(サーバ定義)」を参照してください。

1.4.1 構文

```
idl2java [options]{filename}
```

例

```
idl2java -no_tie Bank.idl
```

1.4.2 説明

idl2java は Java ベースのプリプロセサで，IDL ソースファイルをコンパイルして，IDL 宣言に対する Java マッピングを含むディレクトリ構造体を生成します。一般的に，一つの IDL ファイルが複数の Java ファイルにマッピングされます。それは，Java が一つのファイルに対して，一つのパブリックインタフェース，またはクラスだけを許可しているためです。IDL ファイル名には，拡張子.idl を付けてください。

java2idl を実行する際にシステムプロパティ file.encoding を指定しないでください。システムプロパティ file.encoding を指定して文字エンコーディングを変更した場合の動作は保証しません。

生成したスタブ・スケルトンをコンパイルする際には，vbjc コマンド，または javac コマンドの引数に"-source 1.4"を指定してください。"-source 1.4"を指定しないでコンパイルすると，JDK5.0 からのコンパイル時の型チェックの強化によって，次の警告が表示されます。

```
「注: 入力ファイルの操作のうち、未チェックまたは安全ではないものがあります。
注: 詳細については、-Xlint:unchecked オプションを指定して再コンパイルしてください。」
```

idl2java コマンドの引数に日本語や 2 バイトコード文字を含むパスを指定しないでください。指定した場合はエラーになります。

キーワード

キーワードは，次のオプションと処理対象の IDL 入力ファイルの両方を含みます。

オプション

- `-D, -define foo[=bar]`
  foo に指定したプリプロセサマクロを定義します。bar で値を指定することもできます。
- `-I, -include <dir>`
  インクルードファイルのディレクトリを絶対パスまたは相対パスで指定します。インクルードファイルを検索するときに指定します。
- `-P, -no_line_directives`
  行番号情報の生成を抑止します。デフォルトは off です。
- `-H, -list_includes`
  インクルードファイルのパスを標準出力に出力します。
- `-C, -retain_comments`
  Java コードが生成されるときに，IDL ファイルのコメントを保持します。コメントを保持しない場合，コメントは Java コードで表示されません。デフォルトは off です。
- `-compilerflags`
  設定できるフラグを指定します。
- `-compiler`
  コンパイラオプションを指定します。
- `-U, -undefine foo`
  foo に指定したプリプロセサマクロの定義を解除します。
- `-[no_]idl_strict`
  IDL ソースに対して厳密に OMG 標準規格を適用することを指定します。デフォルトは off です。
- `-[no_]warn_unrecognized_pragmas`
  #pragma が認識されない場合に警告を表示します。デフォルトは on です。
- `-[no_]back_compat_mapping`
  VisiBroker 3.x 互換のマッピングを使用することを指定します。
- `-[no_]comments`
  コード中にコメントを生成するのを抑止します。デフォルトは on です。
- `-[no_]examples`

_example クラスの生成を抑止します。デフォルトは off です。

- -gen_included_files
  インクルードファイルのコードを生成します。デフォルトは off です。
- -list_files
  コード生成時に書き込まれたファイルの一覧を表示します。デフォルトは off です。
- -[no_]obj_wrapper
  オブジェクトラッパーのサポートを生成します。デフォルトは off です。
- -root_dir <*path*>
  生成したファイルを格納するディレクトリを指定します。
- -[no_]servant
  サーバント（サーバ側）コードを生成します。デフォルトは on です。
- -tie
  _tie クラスを生成します。デフォルトは on です。
- -[no_]warn_missing_define
  あらかじめ宣言されたファイル名を定義しなかった場合に警告します。デフォルトは on です。
- -[no_]bind
  生成した Helper クラスに bind( )メソッドを生成するのを抑止します。デフォルトは off です。
- -[no_]compile
  on に設定すると，Java ファイルを自動的にコンパイルします。デフォルトは off です。
- -dynamic_marshal
  マーシャリングに DSI モデルまたは DII モデルを使用することを指定します。デフォルトは off です。
- -idl2package <*IDL name*> <*pkg*>
  指定した IDL コンテナ型のデフォルトパッケージをオーバーライドします。
- -[no_]invoke_handler
  EJB の呼び出しハンドラクラスを生成します。デフォルトは off です。
- -[no_]narrow_compliance
  ナロウイングに従ったコードを生成します。デフォルトは on です。このオプションを off に設定すると，下位互換のコードを生成します。
- -[no_]Object_methods
  java.lang 固有のメソッドをオーバーライドします。デフォルトは on です。
- -package <*pkg*>
  生成したコードのルートパッケージを指定します。
- -stream_marshal

マーシャリングにストリームモデルを使用することを指定します。デフォルトは on です。

- `-strict`
  OMG の規格に厳密に従ってコード生成することを指定します。デフォルトは off です。
- -version
  Borland Enterprise Server VisiBroker のバージョン番号を表示します。
- `-map_keyword` 〈*kwd*〉〈*replacement*〉
  使用を避けるキーワードとそれに置き換わる文字列を指定します。
- `-[no_]copy_local_values`
  CORBA メソッドに同じプロセス内で呼び出しがあった場合に，値をコピーします。デフォルトは off です。
- `-[no_]preprocess`
  入力ファイルをパーシングする前にプリプロセシングします。デフォルトは on です。
- `-[no_]preprocess_only`
  プリプロセシングしたあとで，入力ファイルをパーシングしないようにします。デフォルトは off です。
- `-[no_]warn_all`
  すべての警告メッセージが出力されるようにします。デフォルトは off です。
- `-h, -help, -usage, -?`
  ヘルプ情報を出力します。

このほか，idl2java のオプションとして，vbj のオプションを使用できます。vbj のオプションについては，「1.7　vbj」を参照してください。

1.5 java2idl

このコマンドは，Java クラスファイル（バイトコード）から IDL ファイルを生成します。複数の Java クラス名を入力できます。複数のクラス名を入力する場合，クラス名とクラス名の間に必ず空白を入れてください。

Java のリモートインタフェース定義に org.omg.CORBA.IDLEntity を継承するクラスを使用する場合は，次のファイルとクラスを使用してください。

- 使用するクラスの型に対応する IDL 定義を格納した IDL ファイル
  org.omg.CORBA.IDLEntity インタフェースは，Java にマッピングされた IDL データ型をすべてマークするシグニチャインタフェースであるためです。
- OMG の「CORBA 2.4 IDL2Java 仕様」に従う，関連する（サポート対象の）すべてのクラス

Java リモートインタフェース定義で，org.omg.CORBA.IDLEntity を継承するクラスを使用する場合は，Java2idl ツールのコマンドラインに-import <IDL files>ディレクティブを使用してください。

詳細については，「CORBA 2.4 IDL2Java 仕様」を参照してください。

注

このコマンドを使用するには，JDK 1.1 以降のバージョンをサポートする Java VM が必要です。

1.5.1 構文

```
java2idl [options] {filename}
```

例

```
java2idl -o final.idl Account Client Server
```

1.5.2 説明

Java バイトコードから IDL ファイルを生成したい場合，java2idl を使用します。

Java バイトコードがある場合にこのコマンドを使用すると，その Java バイトコードから IDL ファイルを生成して C++などのプログラミング言語で使用できるようになります。

構文の例で示したように-o オプションを使用すると，三つの Java バイトコードファイル（Account, Client，および Server）が final.idl ファイルに出力されます。デフォルトでは，出力結果は画面に表示されます。

キーワード

キーワードは，次のオプションと処理対象の Java バイトコードファイルの両方を含みます。

オプション

- -D, -define *foo*[=*bar*]
  foo に指定したプリプロセサマクロを定義します。bar で値を指定することもできます。
- -I, -include <*dir*>
  インクルードファイルのディレクトリを絶対パスまたは相対パスで指定します。インクルードファイルを検索するときに指定します。
- -H, -list_includes
  インクルードファイルのパスを標準出力に出力します。
- -[no_]idl_strict
  IDL ソースに対して厳密に OMG 標準規格を適用することを指定します。デフォルトは off です。
- -[no_]builtin <*TypeCode*/*Principal*>
  TypeCode または Principal という固有の型を生成します。
  引数に TypeCode または，Principal のどちらかを指定してください。
  デフォルトは on です。
- -[no_]warn_unrecognized_pragmas
  #pragma が認識されない場合に警告を表示します。デフォルトは on です。
- -[no_]back_compat_mapping
  VisiBroker 3.x 互換のマッピングを使用することを指定します。
- -import <*IDL* file *name*>
  IDL 定義を追加してロードします。
- -o <file>
  出力ファイルの名前を指定します。または，標準出力に出力させる場合は「-」を指定します。
- -strict
  OMG の規格に厳密に従ってコード生成することを指定します。デフォルトは off です。
- -version
  Borland Enterprise Server VisiBroker のバージョン番号を表示します。
- -[no_]preprocess_only
  プリプロセシングしたあとで，入力ファイルをパーシングしないようにします。デフォルトは off です。
- -[no_]idlentity_array_mapping
  IDLEntity の配列を boxedRMI 型での boxedIDL にマッピングします。
  デフォルトは off です。
- -h, -help, -usage, -?

ヘルプ情報を出力します。

このほか，java2idl のオプションとして，vbj のオプションを使用できます。vbj のオプションについては，「1.7　vbj」を参照してください。

1.6 java2iiop

このコマンドは，IDL ではなく Java 言語を使用して IDL インタフェースを定義します。複数の Java ファイル名（Java バイトコード）を入力できます。複数の Java ファイル名を入力する場合は，間に空白を入れてください。完全にスコープを定めたクラス名を使用してください。

注

このコマンドを使用するには，JDK 1.1 以降のバージョンをサポートする Java VM が必要です。

Java のリモートインタフェース定義に org.omg.CORBA.IDLEntity を継承するクラスを使用する場合は，次に示すファイルとクラスを使用してください。

- 使用するクラスの型に対応する IDL 定義を格納した IDL ファイル。org.omg.CORBA.IDLEntity インタフェースは，Java にマッピングされた IDL データ型をすべてマークするシグニチャインタフェースだからです。
- OMG の「CORBA 2.4 IDL2Java 仕様」に従う，関連する（サポート対象の）すべてのクラス。

Java リモートインタフェース定義で，org.omg.CORBA.IDLEntity を継承するクラスを使用する場合は，Java2iiop ツールのコマンドラインに-import <IDL files>ディレクティブを使用してください。

スーパーインタフェースで定義した final 値をサブインタフェースで再定義し，そのサブインタフェースを java2iiop でコンパイルするとエラーが発生します。final 値をサブインタフェースで再定義しないでください。

詳細については，「CORBA 2.4 IDL2Java 仕様」を参照してください。

生成したスタブ・スケルトンを，J2EE サーバプロセス上で使用する場合は，Cosminexus Component Container の「J2EE アプリケーション実行時間の監視と J2EE アプリケーションの強制停止機能」※1 の対象外として，明示的に指定する必要があります。次のファイル※2 に，生成したスタブ・スケルトンをフルパッケージ指定で記述してください。

ファイル格納先：

<Cosminexus インストールディレクトリ>¥CC¥server¥usrconf¥criticalList.cfg

注※1

詳細については，「Cosminexus V11 アプリケーションサーバ 機能解説 運用／監視／連携編」を参照してください。

注※2

詳細については，「Cosminexus V11 アプリケーションサーバ リファレンス 定義編(サーバ定義)」を参照してください。

1.6.1 構文

```
java2iiop [options] {class name}
```

例

```
java2iiop -no_tie Account Client Server
```

1.6.2 説明

分散オブジェクトを使用するために利用したい Java バイトコードがある場合や IDL を記述したくない場合に，java2iiop を使用してください。このコマンドを使用すると，必要なコンテナクラス，クライアントスタブ，およびサーバスケルトンを Java バイトコードから生成できます。

java2idl を実行する際にシステムプロパティ file.encoding を指定しないでください。システムプロパティ file.encoding を指定して文字エンコーディングを変更した場合の動作は保証しません。

生成したスタブ・スケルトンをコンパイルする際には，vbjc コマンド，または javac コマンドの引数に"-source 1.4"を指定してください。"-source 1.4"を指定しないでコンパイルすると，JDK5.0 からのコンパイル時の型チェックの強化によって，次の警告が表示されます。

```
「注: 入力ファイルの操作のうち、未チェックまたは安全ではないものがあります。
注: 詳細については、-Xlint:uncheckedオプションを指定して再コンパイルしてください。」
```

注

java2iiop コンパイラは，CORBA インタフェース上のオーバーロードメソッドをサポートしません。パッケージ名と同一名称のインタフェースを定義しないでください。java2iiop でのコンパイル時にコンパイルエラーになります。

キーワード

キーワードは，次のオプションと処理対象の Java バイトコードファイルの両方を含みます。

オプション

- `-D, define foo[=bar]`
  foo に指定したプリプロセサマクロを定義します。bar で値を指定することもできます。
- `-I, -include <dir>`
  インクルードファイルのディレクトリを絶対パスまたは相対パスで指定します。インクルードファイルを検索するときに指定します。
- `-H, -list_includes`
  インクルードファイルのパスを標準出力に出力します。
- `-[no_]idl_strict`
  IDL ソースに対して厳密に OMG 標準規格を適用することを指定します。デフォルトは off です。

- -[no_]builtin <*TypeCode/Principal*>
  TypeCode または Principal という固有の型を生成します。
  引数に TypeCode または，Principal のどちらかを指定してください。
  デフォルトは on です。
- -[no_]warn_*unrecognized*_pragmas
  #pragma が認識されない場合に警告を表示します。デフォルトは on です。
- -[no_]back_*compat*_mapping
  VisiBroker 3.x 互換のマッピングを使用することを指定します。
- -import <*IDL file name*>
  IDL 定義を追加してロードします。
- -[no_]*comments*
  コード中のコメントの生成を抑止します。デフォルトは on です。
- -[no_]*examples*
  _example クラスの生成を抑止します。デフォルトは off です。
- -gen_*included*_files
  インクルードファイルのコードを生成します。デフォルトは off です。
- -list_*files*
  コード生成時に書き込まれたファイルの一覧を表示します。デフォルトは off です。
- -[no_]*obj*_wrapper
  オブジェクトラッパーのサポートを生成します。デフォルトは off です。
- -root_*dir* <*path*>
  生成したファイルを格納するディレクトリを指定します。
- -[no_]servant
  サーバント（サーバ側）コードを生成します。デフォルトは on です。
- -[no_]*tie*
  _tie クラスを生成します。デフォルトは on です。
- -[no_]*bind*
  生成した Helper クラスに bind( )メソッドを生成するのを抑止します。デフォルトは off です。
- -[no_]*compile*
  Java ファイルを自動的に生成します。on に設定すると，Java ファイルを自動的に生成してコンパイルします。デフォルトは off です。
- -*compiler*
  Java コンパイラを使用することを指定します。このオプションは-compile オプションを設定しないと無視されます。
- -*compilerflags*

Java コンパイラフラグを Java コンパイラに渡すことを指定します。このオプションは-compile オプションを設定しないと無視されます。

- -dynamic_*marshal*

  マーシャリングに DSI モデルまたは DII モデルを使用することを指定します。デフォルトは off です。

- -idl2package <*IDL name*> <*pkg*>

  指定した IDL コンテナ型のデフォルトパッケージをオーバーライドします。

- -[no_]invoke_*handler*

  EJB の呼び出しハンドラクラスを生成します。デフォルトは off です。

- -[no_]narrow_*compliance*

  ナロウイングに従ったコードを生成します。デフォルトは on です。

- -package <*pkg*>

  生成したコードのルートパッケージを指定します。

- -stream_*marshal*

  マーシャリングにストリームモデルを使用することを指定します。デフォルトは on です。

- -strict

  OMG の規格に厳密に従ってコード生成することを指定します。デフォルトは off です。

- -version

  Borland Enterprise Server VisiBroker のバージョン番号を表示します。

- -map_keyword <*kwd*> <*replacement*>

  使用しないキーワードとそれに置き換わる文字列を指定します。

- -sealed <*pkg*> <*dest_pkg*>

  指定したパッケージ<*pkg*>に関連するコードを，<*dest_pkg*>配下に出力します。
  <*dest_*pkg>ディレクトリがない場合は，カレントディレクトリに出力します。

- -[no_]copy_*local*_values

  CORBA メソッドに同じプロセス内で呼び出しがあった場合に，値をコピーします。デフォルトは off です。

- -[no_]preprocess_only

  プリプロセシングしたあとで，入力ファイルをパーシングしないようにします。デフォルトは off です。

- -[no_]*idlentity*_array_mapping

  IDLEntity の配列を boxedRMI 型での boxedIDL にマッピングします。
  デフォルトは off です。

- -h, -help, -*usage*, -?

  ヘルプ情報を出力します。

このほか，java2iiop のオプションとして，vbj のオプションを使用できます。vbj のオプションについては，「1.7　vbj」を参照してください。

1.7 vbj

このコマンドは，ローカルマシンにある Java インタプリタを起動します。

1.7.1 構文

```
vbj [options] [arguments normally sent to java VM] {class}
    [arg1 arg2...]
```

構文の説明

{class}：実行されるクラス名を指定します。

[arg1 arg2...]：クラスに渡されるパラメタです。

例

```
vbj Server
```

1.7.2 説明

Java アプリケーションには，ほかの言語で書かれたアプリケーションにはない制限がありますが，vbj はこの制限の幾つかに対処するオプションを提供しています。また，vbj は Borland Enterprise Server VisiBroker のアプリケーションを起動するのに望ましい方法でもあります。vbj の機能は次のとおりです。

- 環境変数と Windows レジストリ設定をチェックします。
- UDP ブロードキャストサポートのない Java VM のために，オプションで osagent を探索できます。
- CLASSPATH を自動的に設定し，Java ランタイムに対して Borland Enterprise Server VisiBroker を正しく動作させます。

さらに，vbj は二つの VisiBroker ORB プロパティを設定して，VisiBroker ORB ランタイムに渡します。

- vbroker.agent.addr
  OSAGENT_ADDR 環境変数の値またはレジストリ設定を格納します。
- vbroker.agent.port
  OSAGENT_PORT 環境変数の値またはレジストリ設定を格納します。

注

vbj を使用しない場合は，Borland Enterprise Server VisiBroker が正しく動作するように，vbroker.agent.addr と vbroker.agent.port の値を明示的に設定してください。

1.7.3 オプション

- -h, -help, -usage, -?

  ヘルプ情報を出力します。

- -VBJprop *name=value*

  実行された Java に-D<name>[=<value>]パラメタとして追加することで，システムプロパティとしての Java VM にプロパティ名と値を渡します。

- -VBJjavavm *vmpath*

  使用する Java VM を指定します。

- -VBJaddJar <*jarfile*>

  指定した jar ファイルを CLASSPATH に追加します。jar ファイルは，<インストールディレクトリ>/lib に格納されている必要があります。

- -VBJclasspath *classpath*

  クラスパスの明示的設定を指定します。

- -VBJdebug

  デバッグ情報をオンにします。

このコマンドに渡される追加オプションは，システムにインストールされた Java VM で定義されます。このコマンドのすべてのオプションを表示させるには，オプションなしで次のように java と入力してください。

```
prompt>java
```

Java インタプリタで利用できるオプション一覧が表示されます。

1.8 vbjc

このコマンドは，VisiBroker のクラスをインポートする可能性のある Java ソースコードをコンパイルするときに使用します。このコマンドの処理を次に示します。

1. Borland Enterprise Server VisiBroker のライブラリパスを検索します。
2. Borland Enterprise Server VisiBroker 標準の jar ファイルを CLASSPATH に追加します。
3. javac を起動します。

1.8.1 構文

```
vbjc [arguments normally passed to javac]
```

例

```
vbjc Server.java
```

1.8.2 オプション

- `-h, -help, -usage, -?`
  ヘルプ情報を出力します。
- `-VBJdebug`
  デバッグ情報をオンにします。
- `-VBJclasspath classpath`
  クラスパスの明示的設定を指定します。

1.9 クラスパスの指定

クラスパスに関連があるソースを次に示します。ここでは，UNIX でのパスを示しています。Windows の場合は，/を¥に置き換えてください。

- vbroker ライブラリファイル（$VBROKERDIR※/lib/*.jar）
- JDK ライブラリファイル（$JAVAHOME※/lib/*.jar，$JAVAHOME※/jre/lib/*.jar，および classes.zip）
- CLASSPATH 環境変数（$CLASSPATH）
- -VBJclasspath <path string> パラメタ
- -classpath <path string> パラメタ
- -Djava.class.path=<path string> パラメタ
- -Denv.class.path=<path string> パラメタ
- -VBJaddJar <jar file name> パラメタ（jar ファイルは vbroker/lib ディレクトリに格納されている必要があります）

注※

$VBROKERDIR は Borland Enterprise Server VisiBroker のインストールディレクトリを，$JAVAHOME は Java のホームディレクトリを表します。

通常，Java VM の起動前に，これらのサブセットが一つのクラスパスにマージされます。また，クラスパスは，プラットフォームによって生成方法が異なります。

マージされるソースと順序を次に示します。

1. -VBJclasspath に指定されたクラスパス
2. エクスポートされた$CLASSPATH
3. VisiBroker 標準の jar ファイル（検索された vbj の位置に基づいて決定します）
4. VBJaddJar で追加され，<vbroker>/lib ディレクトリにあると想定される jar ファイル
5. カレントディレクトリ

(UNIX)

マージ後のクラスパスは Java VM にエクスポートされ，パラメタとしては渡されません。ほかのクラスパスソースは，クラスパス以外のパラメタとともに Java VM にそのまま渡されます。

(Windows)

マージ後のクラスパスは-Djava.class.path で渡されます。ほかのクラスパスは，一般的なパラメタとしてそのまま渡されます。

注

サービスとして Windows に定義されているツールは，-classpath を認識しません。-classpath を無視するツールもありますが，nameserv，irep などでは，-classpath で処理が中断しエラーが発生します。

1.10 Java VM の指定

Java VM の指定方法は，プラットフォームによって異なります。

(UNIX)

デフォルト Java VM は java ですが，次のパラメタを指定すると Java VM を変更できます。

```
-VBJjavavm <jvm name>
```

どの Java VM を指定した場合でも，指定した Java VM があるかどうかが確認されます。指定した Java VM が見つからない場合は，プログラムが異常終了します。デフォルトの Java VM の検索はされません。

(Windows)

Java VM を明示的に指定しなかった場合，Borland Enterprise Server VisiBroker は，環境設定にパス情報があるかどうかを調べます。PATH 変数が設定されている場合は，¥bin を含む PATH の各ディレクトリに Java VM の dll があるかどうかを調べます。PATH が設定されているのに Java VM が見つからない場合，Borland Enterprise Server VisiBroker は異常終了します。

PATH が設定されていない場合，Borland Enterprise Server VisiBroker は，現在インストールされている Java VM の位置を Windows のレジストリで調べます。

Java VM の検索に使用する dll ファイル名は，Java VM 1.2 の場合は jvm.dll で，Java VM 1.1 の場合は javai.dll です。

次のパラメタを指定すると Java VM を変更できます。

```
-VBJjavavm <jvm name>
```

<jvm name>には，アプリケーションの絶対パスを次の形式で指定します。

```
*¥**¥*java*
```

*は，任意の文字列を意味します。空文字列も指定できます。例えば，「C:¥jdk¥bin¥java」などのパスを指定できます。<jvm name>に指定した文字列の形式が正しくない場合や，<jvm name>に指定したファイルが見つからない場合，プログラムはエラーメッセージを出力して異常終了します。

1.11　その他のツール

Borland Enterprise Server VisiBroker と同時にインストールされるツールを次に示します。これらのツールを使用して，ユーザが開発したクライアントやサーバを実行できます。

- osagent

  osagent（スマートエージェント）は，ユーザのクライアントプログラムの通信相手であるオブジェクトインプリメンテーションを検索するサービスです。ユーザのローカルネットワークに接続された一つ以上のホストで起動してください。スマートエージェントのオプションの詳細については，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「スマートエージェント（osagent）の起動」の記述を参照してください。

- locserv

  ロケーションサービスを使用すると，特定の属性を持つオブジェクトインスタンスを検索できます。ロケーションサービスは，Borland Enterprise Server VisiBroker のスマートエージェントと連携して，ネットワーク上の，どのオブジェクトが現在アクセスできるのかをユーザに通知します。ロケーションサービスの詳細については，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「ロケーションサービスの使用」の記述を参照してください。

- irep

  インタフェースリポジトリ（irep）はユーザの CORBA オブジェクトのインタフェースを格納し，クライアントによるランタイムアクセスのために使用します。ユーザはブラウザでリポジトリ内の情報を参照できます。インタフェースリポジトリの詳細については，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「インタフェースリポジトリの使用」の記述を参照してください。

- oad

  オブジェクト活性化デーモン（OAD）を使用すると，クライアントのアクセスによって自動的に開始するオブジェクトを登録できます。オブジェクト活性化デーモンの詳細については，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「オブジェクト活性化デーモンの使用」の記述を参照してください。

- oadutil

  OAD ユティリティは，使用しているシステムで利用できるオブジェクトインプリメンテーションの登録，登録解除，一覧表示をユーザが任意に実行するためのコマンドを提供します。このコマンドは，コマンドラインから入力することも，スクリプトに記述することもできます。このツールとオプションの詳細については，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「オブジェクト活性化デーモンユーティリティの使用」の記述を参照してください。

- osfind

  osfind はコマンドラインから実行するツールです。このツールを実行すると，現在実行中のプロセスおよびオブジェクトの概要を知ることができます。このツールとオプションの詳細については，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「すべてのオブジェクトとサービスの報告」の記述を参照してください。

- gatekeeper
  ゲートキーパーは，アプレットをネットワーク経由でオブジェクトサーバと通信させるためのツールです。ゲートキーパーは，Web ブラウザおよびファイアウォールによるセキュリティ制限に違反しないで通信を実現します。ゲートキーパーの詳細については，マニュアル「Borland Enterprise Server VisiBroker　ゲートキーパーガイド」を参照してください。

2 IDL から Java へのマッピング

この章では，Borland Enterprise Server VisiBroker が現在使用している IDL/Java 言語マッピングの概要を説明します。この IDL/Java 言語マッピングは idl2java コンパイラがインプリメントします。Borland Enterprise Server VisiBroker は「OMG IDL/Java 言語マッピング仕様」に従っています。

完全な情報，特に次の情報については，最新版の「OMG IDL/Java 言語マッピング仕様」を参照してください。

・擬似オブジェクトの Java へのマッピング

・サーバ側マッピング

・Java ORB ポータビリティインタフェース

2.1 名前

一般的に，IDL 名と識別子は変更されないで Java 名と識別子にマッピングされます。

マッピングされた Java コードで名前の衝突が発生する場合，マッピングされた名前の先頭にアンダースコア（_）を付けて解決します。

さらに，Java 言語の特質によって，単一の IDL 構造体は複数の（異なる名前の）Java 構造体にマッピングされることがあります。「追加」名は，記述接尾語を付けることで構成されます。例えば，IDL インタフェースの AccountManager は，Java インタフェースの AccountManager および追加の Java クラス AccountManagerOperations，AccountManagerHelper，ならびに AccountManagerHolder にマッピングされます。

「追加」名がほかのマッピングされた IDL 名と衝突する例外の場合は，上記の解決法がそのほかのマッピングされた IDL 名に適用されます。つまり，先にネーミングと必要な「追加」名が使用されます。

例えば，名前が fooHelper または fooHolder のインタフェースは，foo と名づけられたインタフェースがあってもなくても，_fooHelper または_fooHolder にそれぞれマッピングされます。インタフェース fooHelper の Helper クラスおよび Holder クラスは，_fooHelperHelper および_fooHelperHolder と名づけられます。

通常，Java の予約語と衝突する Java 識別子に，変更されないでマッピングされる IDL 名には，適用される衝突規則があります。

2.2 予約名

マッピングはマッピング本来の目的のために，複数の名前を予約しています。これらの名前をユーザ定義 IDL 型名，またはインタフェース名（正当な IDL 名とも仮定される）で使用すると，マッピング名では先頭にアンダースコア（_）が付きます。予約名を次に示します。

- Java クラス<type>Helper
  <type>は，IDL ユーザ定義型名です。
- Java クラス<type>Holder
  <type>は，（typedef エイリアスのような例外のある）IDL ユーザ定義型名です。
- Java クラス<basicJavaType>Holder
  <basicJavaType>は，IDL 基本データ型のどれかで使用される Java 基本データ型の一つです。
- ネストされたスコープの Java パッケージ名<interface>Package
  <interface>は，IDL インタフェース名です。
- Java クラス<interface>Operations，<interface>POA，および<interface>POATie
  <interface>は IDL インタフェース型です。

2.3 予約語

マッピングでは，幾つかの名前が特定の目的のために予約されています。その予約語をユーザ定義の IDL，またはインタフェースの語句（IDL としての文法は正しい）に使用すると，先頭にアンダースコア（_）が付けられてマッピングされます。Java 言語の予約キーワードを次に示します。

```
abstract        abstractBase    boolean         break
byte            case            catch           char
class           const           continue        default
do              double          else            extends
false           final           finally         float
for             goto            if              implements
import          instanceof      int             interface
local           long            native          new
null            package         private         protected
public          return          short           static
super           switch          synchronized    this
throw           throws          transient       true
try             void            volatile        while
```

2.4 モジュール

IDL モジュールは，同じ名前で Java パッケージにマッピングされます。モジュール内のすべての IDL 型宣言は，生成されたパッケージ内の該当する Java クラスまたはインタフェース宣言にマッピングされます。

モジュール内にない IDL 宣言は，（名前を付けられていない）Java グローバルスコープにマッピングされます。

IDL モジュール内で宣言された型に対して生成された Java コードをコードサンプル 2-1 に示します。

コードサンプル 2-1　IDL モジュールの Java パッケージへのマッピング

```
/* From Example.idl: */
module Example { .... };

// Generated java
package Example;
...
```

2.5 基本型

定義された IDL 型が，どのように基本 Java 型にマッピングされるかを次の表に示します。

表 2-1 基本型マッピング（Java）

IDL 型	Java 型
boolean	boolean
char	char
wchar	char
octet	byte
string	java.lang.String
wstring	java.lang.String
short	short
unsigned short	short
long	int
unsigned long	int
longlong	long
unsigned longlong	long
float	float
double	double

IDL 型と，マッピングされた Java 型との間で潜在的な不一致がある場合，標準 CORBA 例外が発生します。ほとんどの場合，発生した例外は，次の二つのカテゴリに分類されます。

- Java 型の範囲が IDL 型の範囲より大きい場合
  例えば，Java chars が IDL chars のスーパーセットである場合が該当します。
- Java がアンサインドの型をサポートしていない場合
  開発者は，アンサインドの IDL 型の大きな値が Java で負の整数として正しく処理されることを確実にする責任があります。
- IDL のインタフェースに，配列，enum，struct，union 型のどれかを引数に持つ operation を使用している，かつ，配列，enum，struct，union 型のデータとして null オブジェクトを指定している場合，データの送信時に BAD_PARAM 例外ではなく，NullPointerException または UNKNOWN 例外が発生します。

2.5.1　IDL 型拡張

ここでは，IDL 型拡張のための Borland Enterprise Server VisiBroker のサポートについて説明します。サポートされる IDL 拡張の概要を次の表に示します。

表 2-2　サポートされる IDL 拡張の概要（Java）

型	Borland Enterprise Server VisiBroker でのサポート
longlong	有
unsigned longlong	有
long double	無※1
wchar	有※2
wstring	有※2
fixed	無※1

注※1

Borland Enterprise Server VisiBroker では，OMG がインプリメント方法を決定してからサポートする予定です。

注※2

Unicode は「on the wire」で使用されます。

また，新しい型のサポートについて次の表に示します。

表 2-3　新しい型の IDL 拡張（Java）

新型の IDL	説明
longlong	64 ビットサインドの 2 の補数
unsigned longlong	64 ビットアンサインドの 2 の補数
long double	IEEE 標準 754-1985 ダブル拡張浮動小数点
wchar	ワイドキャラクタ
wstring	ワイド文字列
fixed	固定小数点 10 進演算（31 有効けた数）

2.5.2　Holder クラス

Holder クラスは，OUT および INOUT パラメタ受け渡しモードをサポートし，org.omg.CORBA パッケージのすべての基本 IDL データ型で利用できます。Holder クラスは，typedefs で定義されたものを除

いて，すべての名前付きユーザ定義型で生成されます。Holder クラスの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

ユーザ定義 IDL 型では，Holder クラス名は，その型のマッピングされた Java 名の後ろに Holder を付けて構成されます。

基本 IDL データ型では，マッピングされたデータ型に対する Java 型名（最初の文字が大文字）の後ろに Holder を付けた Holder クラス名になります。例えば，IntHolder がそうです。

各 Holder クラスには，インスタンスからのコンストラクタ，デフォルトコンストラクタがあり，型付きの value というパブリックインスタンスメンバがあります。デフォルトコンストラクタは，Java 言語によって定義されたように，値フィールドを型のデフォルト値に設定します。

- boolean：false
- 値型：null
- 数値と char 型：0
- 文字列：null
- オブジェクトリファレンス：null

ポータブルスタブ，およびスケルトンをサポートするには，ユーザ定義型の Holder クラスも org.omg.CORBA.portable.Streamable インタフェースをインプリメントします。

基本型の Holder クラスは，次のコードサンプルで定義されます。これらは org.omg.CORBA パッケージにあります。

コードサンプル 2-2　Holder クラス

```
//Java
package org.omg.CORBA;

final public class ShortHolder implements Streamable {
    public short value;
    public ShortHolder( ){}
    public ShortHolder(short initial){
        value =initial;
    }
    ...//implementation of the streamable interface
}

final public class IntHolder implements Streamable {
    public int value;
    public IntHolder( ){}
    public IntHolder(int initial){
        value =initial;
    }
    ...//implementation of the streamable interface
}

final public class LongHolder implements Streamable {
    public long value;
```

```
    public LongHolder( ){}
    public LongHolder(long initial){
        value =initial;
    }
    ...//implementation of the streamable interface
}

final public class ByteHolder implements Streamable {
    public byte value;
    public ByteHolder( ){}
    public ByteHolder(byte initial){
        value =initial;
    }
    ...//implementation of the streamable interface
}

final public class FloatHolder implements Streamable {
    public float value;
    public FloatHolder( ){}
    public FloatHolder(float initial){
        value =initial;
    }
    ...//implementation of the streamable interface
}

final public class DoubleHolder implements Streamable {
    public double value;
    public DoubleHolder( ){}
    public DoubleHolder(double initial){
        value =initial;
    }
    ...//implementation of the streamable interface
}

final public class CharHolder implements Streamable {
    public char value;
    public CharHolder( ){}
    public CharHolder(char initial){
        value =initial;
    }
    ...//implementation of the streamable interface
}

final public class BooleanHolder implements Streamable {
    public boolean value;
    public BooleanHolder( ){}
    public BooleanHolder(boolean initial){
        value =initial;
    }
    ...//implementation of the streamable interface
}

final public class StringHolder implements Streamable {
    public java.lang.String value;
    public StringHolder( ){}
    public StringHolder(java.lang.String initial){
        value =initial;
    }
```

```
    ...//implementation of the streamable interface
}

final public class ObjectHolder implements Streamable {
    public org.omg.CORBA.Object value;
    public ObjectHolder( ){}
    public ObjectHolder(org.omg.CORBA.Object initial){
        value =initial;
    }
    ...//implementation of the streamable interface
}

final public class ValueBaseHolder implements Streamable {
    public java.io.Serializable value;
    public ValueBaseHolder( ){}
    public ValueBaseHolder(java.io.Serializable initial){
        value =initial;
    }
    ...//implementation of the streamable interface
}

final public class AnyHolder implements Streamable {
    public Any value;
    public AnyHolder( ){}
    public AnyHolder(Any initial){
        value =initial;
    }
    ...//implementation of the streamable interface
}

final public class TypeCodeHolder implements Streamable {
    public TypeCode value;
    public typeCodeHolder( ){}
    public TypeCodeHolder(TypeCode initial){
        value =initial;
    }
    ...//implementation of the streamable interface
}

final public class PrincipalHolder implements Streamable {
    public Principal value;
    public PrincipalHolder( ){}
    public PrincipalHolder(Principal initial){
        value =initial;
    }
    ...//implementation of the streamable interface
}
```

ユーザ定義型<foo>の Holder クラスを次に示します。

コードサンプル 2-3　ユーザ定義型の Holder クラス

```
// Java
final public class <foo>Holder
   implements org.omg.CORBA.portable.Streamable {
   public <foo> value;
   public <foo>Holder( ) {}
```

```
    public <foo>Holder(<foo> initial) {}
    public void _read(org.omg.CORBA.portable.InputStream i)
     {...}
    public void _write(org.omg.CORBA.portable.OutputStream o)
     {...}
    public org.omg.CORBA.TypeCode _type( ) {...}
}
```

Java null

Java null は，nullCORBA オブジェクトリファレンスと valuetype（再帰型の valuetype を含む）を表すためだけに使用されます。例えば，空の文字列を表すときは，null を使わないで長さ 0 の文字列を使用してください。これは配列にも，valuetype を除く構造型にも当てはまります。構造体用に null を渡そうとすると，NullPointerException が発生します。

2.5.3 boolean

IDL 型 boolean は Java 型 boolean にマッピングされます。IDL 定数 TRUE および FALSE は，Java 定数 true および false にマッピングされます。

2.5.4 char

Java 文字が Unicode を表す 16 ビットのアンサインドの精度であるのに対して，IDL 文字は文字セットの要素を表す 8 ビットの精度です。タイプ・セーフにするため，Java CORBA ランタイムは，パラメタがメソッド呼び出し中にマーシャリングされるときに，IDL char からマッピングされたすべての Java chars の範囲が妥当であることをチェックします。

char が文字セットで定義された範囲外である場合，CORBA::DATA_CONVERSION 例外となります。

IDL wchar は Java char 型にマッピングします。

2.5.5 octet

8 ビットの精度である IDL 型 octet は，Java 型 byte にマッピングされます。

2.5.6 string

バウンド，またはアンバウンデッドの IDL 型 string は，Java 型 java.lang.String にマッピングされます。文字列のバウンドチェックと同じく，文字列内の文字の範囲がマーシャリング時にチェックされます。

2.5.7 wstring

Unicode 文字列を表すために使用される IDL 型 wstring は，Java 型 java.lang.String にマッピングされます。文字列のバウンドチェックはマーシャリング時に実施されます。

2.5.8 整数型

IDL short および unsigned short は，Java 型 short にマッピングします。

IDL long および unsigned long は Java 型 int にマッピングします。

注

Java はアンサインドの型をサポートしていません。そのため，開発者は，Java の負の整数がアンサインドの大きな値として正しく処理されることを確実にする責任があります。

2.5.9 浮動小数点型

IDL 浮動小数点型 float と double は，該当するデータ型を含む Java クラスにマッピングします。

2.6 Helper クラス

すべてのユーザ定義 IDL 型には，生成された型名の後ろに Helper が付いた，追加の「helper」Java クラスがあります。型を処理するために必要な幾つかの次のような静的メソッドが提供されています。

- 型の Any 挿入，および抽出オペレーション
- リポジトリ ID の取得
- タイプコードの取得
- ストリームからの型の読み出し，またはストリームへの型の書き込み

ユーザ定義 IDL 型<typename>では，型用に生成された次の Java コードがあります。マッピングされた IDL インタフェースの Helper クラスには，専用の narrow オペレーションがあります。

abstract interface を継承する interface を使用している，かつ，その interface の Helper クラスの narrow オペレーションの引数に null を指定した場合に，BAD_PARAM 例外が発生します。

コードサンプル 2-4　Helper クラス：ユーザ定義型に生成された Java コード

```
// generated Java helper
public class <typename>Helper {
    public static void
       insert(org.omg.CORBA.Any a, <typename> t);
    public static <typename> extract(org.omg.CORBA.Any a);
    public static org.omg.CORBA.TypeCode type( );
    public static String id( );
    public static <typename> read(
         org.omg.CORBA.portable.InputStream istream);
       {...}

    public static void write(
               org.omg.CORBA.portable.OutputStream ostream,
               <typename> value)
        {...}

   // only for interface helpers
   public static
       <typename> narrow(org.omg.CORBA.Object obj);
}
```

コードサンプル 2-5　名前付き型の Java Helper クラスへのマッピング

```
// IDL - named type
struct st {long f1, String f2};

// generated Java
public class stHelper {
    public static void insert(org.omg.CORBA.Any any,
      st s); {...}
    public static st extract(org.omg.CORBA.Any a) {...}
    public static org.omg.CORBA.TypeCode type( ) {...}
    public static String id( ) {...}
    public static st read(
```

```
                        org.omg.CORBA.InputStream is) {...}
    public static void write(org.omg.CORBA.OutputStream os,
       st s) {...}
}
```

コードサンプル 2-6　typedef 列の Java Helper クラスへのマッピング

```
// IDL - typedef sequence
typedef sequence <long> IntSeq;

// generated Java helper
public class IntSeqHelper {
    public static void insert(org.omg.CORBA.Any any,
                              int[ ] seq);
    public static int[ ] extract(org.omg.CORBA.Any a){...}
    public static org.omg.CORBA.TypeCode type( ){...}
    public static String id( ){...}
    public static int[ ] read(
               org.omg.CORBA.portable.InputStream is)
      {...}
    public static void write(
               org.omg.CORBA.portable.OutputStream os,
               int[ ] seq)
      {...}
}
```

2.7　定数

定数のマッピング方法は出現するスコープによって異なります。

2.7.1　インタフェース内の定数

IDL インタフェース内で宣言される定数は，IDL インタフェースに対応する Java インタフェース Operations クラス内の，public static final フィールドにマッピングされます。

コードサンプル 2-7　モジュール内の IDL 定数の Java クラスへのマッピング

```
/*From Example.idl:*/
module Example {
    interface Foo {
        const long aLongerOne =-321;
    };
};

//Foo.java
package Example;
public interface Foo extends
    com.inprise.vbroker.CORBA.Object,
    Example.FooOperations,
    org.omg.CORBA.portable.IDLEntity {
}
//FooOperations.java
package Example;
public interface FooOperations {
    public final static int aLongerOne =(int)-321;
}
```

2.7.2　インタフェース内にない定数

IDL モジュール内で宣言される定数は，定数と同名で，value と名づけられた public static final フィールドを含む，パブリックインタフェースにマッピングされます。

このフィールドは定数の値を保持します。

注

通常，Java コンパイラはクラスがほかの Java コードで使用されると，値をインライン（組み入れ）します。

コードサンプル 2-8　モジュール内の IDL 定数の Java クラスへのマッピング

```
/* From Example.idl: */
module Example {
   const long aLongOne = -123;
};
```

```
// Generated java
package Example;
public interface aLongOne {
    public static final int value = (int) (-123L);
}
```

2.8 構造型

IDL 構造型には，enum，struct，union，sequence，および array があります。sequence および array は Java array 型にマッピングされます。IDL 構造型の enum，struct，および union は，IDL 型のセマンティックスをインプリメントする Java クラスにマッピングされます。生成された Java クラスには，元の IDL 型と同じ名前が付きます。

2.8.1 enum

IDL enum は，value( )メソッド，ラベルごとの二つの静的データメンバ，整数変換メソッド，およびプライベートコンストラクタを宣言する，enum 型と同名の Java ファイナルクラスにマッピングされます。Java ファイナルクラスにマッピングされる IDL enum の例を次に示します。

コードサンプル 2-9　Java ファイナルクラスにマッピングされる IDL enum

```
// Generated java

public final class <enum_name> {
    //one pair for each label in the enum
    public static final int _<label> = <value>;
    public static final <enum_name> <label> =
      new <enum_name>(_<label>);

    public int value( ) {...}

    //get enum with specified value
    public static <enum_name> from_int(int value);

    //constructor
    protected <enum_name>(int) {...}
}
```

メンバの一つは，IDL enum ラベルと同名の public static final です。そのほかのメンバは，先頭にアンダースコア（_）を付けられ，switch 文で使用されます。

value( )メソッドは整数値を返します。値は，0 から順に割り当てられます。enum に value と名づけられたラベルがある場合，Java の value ( )メソッドとの衝突はありません。

enum のインスタンスは一つだけです。一つのインスタンスしかないので，ポインタ等価テストが正しく動作します。つまり，デフォルト java.lang.Object インプリメンテーションの equals( )，および hash( )が，自動的に enum のシングルトンオブジェクトで正しく動作します。

enum の Java クラスには追加メソッド from_int( )があり，これは enum を指定された値で返します。

enum の Holder クラスも生成されます。その名前は，次のように enum のマッピングされた Java クラス名の後ろに Holder を付けたものです。

コードサンプル 2-10　enum の Holder クラス

```
public class <enum_name>Holder implements
     org.omg.CORBA.portable.Streamable {
    public <enum_name> value;
    public <enum_name>Holder( ) {}
    public <enum_name>Holder(<enum_name> initial) {...}
    public void _read(org.omg.CORBA.portable.InputStream i)
        {...}
    public void _write(
                    org.omg.CORBA.portable.OutputStream o)
        {...}
    public org.omg.CORBA.TypeCode _type( ) {...}
}
```

コードサンプル 2-11　enum の Java にマッピングされた IDL

```
//IDL
module Example {
    enum EnumType {first,second,third };
};

//generated Java
public final class EnumType
        implements org.omg.CORBA.portable.IDLEntity {
    public static final int _first =0;
    public static final int _second =1;
    public static final int _third =2;

    public static final EnumType first =
                                    new EnumType(_first);
    public static final EnumType second =
                                   new EnumType(_second);
    public static final EnumType third =
                                    new EnumType(_third);
    protected EnumType(final int _vis_value){...}

    public int value( ){...}
    public static EnumType from_int (final int _vis_value)
                                                    {...}
    public java.lang.String toString( ){...}
}

public final class EnumTypeHolder
        implements org.omg.CORBA.portable.Streamable {
    public OtherExample.EnumType value;
    public EnumTypeHolder( ){...}
    public EnumTypeHolder(
                final OtherExample.EnumType _vis_value){...}
    public void _read(
             final org.omg.CORBA.portable.InputStream input)
        {...}
    public void _write(
          final org.omg.CORBA.portable.OutputStream output)
        {...}
    public org.omg.CORBA.TypeCode _type( ){...}
```

```
    public boolean equals(java.lang.Object o) {...}
}
```

2.8.2 struct

IDL struct は，IDL のインスタンス変数，およびすべての値のメンバを順序どおりに，同名のファイナル Java クラスにマッピングします。ヌルコンストラクタも提供され，これによって構造体フィールドをあとで初期化することもできます。struct の Holder クラスも生成されます。その名前は，次のように sturct がマッピングされた Java クラス名の後ろに Holder を付けたものです。

コードサンプル 2-12　struct の Holder クラス

```
final public class <class>Holder implements
      org.omg.CORBA.portable.Streamable {
    public <class> value;
    public <class>Holder( ) {}
    public <class>Holder(<class> initial) {...}
    public void _read(
                    org.omg.CORBA.portable.InputStream i)
    {...}
    public void _write(
                   org.omg.CORBA.portable.OutputStream o)
    {...}
    public org.omg.CORBA.TypeCode _type( ) {...}
}
```

コードサンプル 2-13　IDL struct の Java へのマッピング

```
/*From Example.idl:*/
module Example {
    struct StructType {
        long field1;
        string field2;
    };
};

//generated Java
public final class StructType
        implements org.omg.CORBA.portable.IDLEntity {
    public int field1;
    public java.lang.String field2;
    public StructType( ){...}
    public StructType(final int field1,
    final java.lang.String field2){...}
    public java.lang.String toString( ) {...}
    public boolean equals(java.lang.Object o) {...}
}
public final class StructTypeHolder
        implements org.omg.CORBA.portable.Streamable {
    public Example.StructType value;
    public StructTypeHolder( ){...}
    public StructTypeHolder(
                      final Example.StructType _vis_value)
        {...}
```

```
    public void _read(
           final org.omg.CORBA.portable.InputStream input)
        {...}
    public void _write(
         final org.omg.CORBA.portable.OutputStream output)
        {...}
    public org.omg.CORBA.TypeCode _type ( ){...}
}
```

2.8.3 union

IDL union はファイナル Java クラスと同じ名前が与えられ，ファイナル Java クラスにマッピングされます。IDL union は，次に示すコンストラクタとメソッドを提供します。

- デフォルトコンストラクタ
- union のディスクリミネータのアクセッサメソッドの discriminator( )
- 各ブランチのアクセッサメソッド
- 各ブランチの変更メソッド
- 複数のケースラベルのある各ブランチの変更メソッド
- 必要に応じたデフォルトの変更メソッド

マッピングされた union 型名，または任意のフィールド名と名前の衝突がある場合，通常の名前衝突解決規則（ディスクリミネータの前にアンダースコア（_）を付ける）が使用されます。

ブランチアクセッサ，および変更メソッドがオーバーロードとなり，ブランチの後ろに名前が付けられます。アクセッサメソッドは，予想されるブランチが設定されないと，CORBA::BAD_OPERATION システム例外を発生させます。

ブランチに該当する複数のケースラベルがある場合，そのブランチのシンプル変更メソッドが最初のケースラベルの値に対してディスクリミナントを設定します。さらに，明示的ディスクリミネータパラメタを取る追加の変更メソッドが生成されます。

ブランチが default ケースラベルに該当する場合，変更メソッドはほかのどのケースラベルにも一致しない値にディスクリミナントを設定します。

ケースラベルのセットがディスクリミナントの可能値を完全に満たす場合，デフォルトケースラベルの union を指定することは不当です。この状況を検出し，不当なコードの生成を拒否するのは，Java コードジェネレータ（例えば，IDL コンパイラ，またはほかのツール）の責任です。

デフォルトの変更メソッドである_default( )が生成されるのは，デフォルトケースラベルが明示的に指定されていない場合と，すべてのケースラベルがディスクリミナントの可能値を完全には満たしていない場合です。_default( )メソッドは，union の値を範囲外に設定します。

union の Holder クラスも生成されます。その名前は，次のように union がマッピングされた Java クラス名の後ろに Holder を付けたものです。

コードサンプル 2-14　union の Holder クラス

```
final public class <union_class>Holder
       implements org.omg.CORBA.portable.Streamable {
    public <union_class> value;
    public <union_class>Holder( ) {}
    public <union_class>Holder(<union_class> initial) {...}
    public void _read(org.omg.CORBA.portable.InputStream i)
        {...}
    public void _write(
                    org.omg.CORBA.portable.OutputStream o)
        {...}
    public org.omg.CORBA.TypeCode _type( ) {...}
}
```

コードサンプル 2-15　IDL union の Java へのマッピング

```
/*From Example.idl:*/
module Example {
    enum EnumType {first,second,third,fourth,fifth,sixth};
    union UnionType switch (EnumType){
        case first:long win;
        case second:short place;
        case third:
        case fourth:octet show;
        default:boolean other;
    };
};

//Generated java
final public class UnionType {
    //constructor
    public UnionType( ) {...}
    //discriminator accessor
    public int discriminator( ) {...}

    //win
    public int win( ) {...}
    public void win(int value) {...}

    //place
    public short place( ) {...}
    public void place(short value) {...}

    //show
    public byte show( ) {...}
    public void show(byte value) {...}
    public void show(int discriminator,byte value) {...}

    //other
    public boolean other( ) {...}
    public void other(boolean value) {...}
    public java.lang.String to String( ) {. . .}
    public boolean equals(java.lang.Object o) {. . .}
```

```
}

final public class UnionTypeHolder {
        implements org.omg.CORBA.portable.Streamable {
    public UnionType value;
    public UnionTypeHolder( ) {}
    public UnionTypeHolder(UnionType initial) {...}
    public void _read(
                    org.omg.CORBA.portable.InputStream i)
        {...}
    public void _write(
                   org.omg.CORBA.portable.OutputStream o)
        {...}
    public org.omg.CORBA.TypeCode_type( ) {...}
}
```

2.8.4 sequence

IDL sequence は同じ名前で Java 配列にマッピングされます。マッピングでは，sequence 型が必要な場所に，sequence 要素のマッピングされた型の配列が使用されます。

sequence の Holder クラスも生成されます。その名前は，次のように sequence がマッピングされた Java クラス名の後ろに Holder を付けたものです。

コードサンプル 2-16　sequence の Holder クラス

```
final public class <sequence_class>Holder {
   public <sequence_element_type>[ ] value;
   public <sequence_class>Holder( ) {};
   public <sequence_class>Holder(
             <sequence_element_type>[ ] initial) {...};
   public void _read(org.omg.CORBA.portable.InputStream i)
        {...}
   public void _write(
                   org.omg.CORBA.portable.OutputStream o)
        {...}
   public org.omg.CORBA.TypeCode _type( ) {...}
}
```

コードサンプル 2-17　IDL sequence の Java へのマッピング

```
// IDL
typedef sequence<long>UnboundedData;
typedef sequence<long, 42>BoundedData;

// generated Java
final public class UnboundedDataHolder
        implements org.omg.CORBA.portable.Streamable {
    public int[ ] value;
    public UnboundedDataHolder( ) {};
    public UnboundedDataHolder(int[ ] initial) {...};
    public void _read(org.omg.CORBA.portable.InputStream i)
        {...}
    public void _write(
```

```
                    org.omg.CORBA.portable.OutputStream o)
        {...}
    public org.omg.CORBA.TypeCode type( ) {...}
}

final public class BoundedDataHolder
        implements org.omg.CORBA.portable.Streamable {
    public int[ ] value;
    public BoundedDataHolder( ) {};
    public BoundedDataHolder(int[ ] initial) {...};
    public void _read(org.omg.CORBA.portable.InputStream i)
        {...}
    public void _write(
                    org.omg.CORBA.portable.OutputStream o)
        {...}
    public org.omg.CORBA.TypeCode_type( ) {...}
}
```

2.8.5 array

IDL array は，IDL のバウンデッドシーケンスと同じようにマッピングされます。マッピングでは，配列型が必要な場所に，配列要素がマッピングされた型の配列が使用されます。Java では，通常の Java 添字指定演算子がマッピングされた配列に適用されます。配列の長さは，定数規則でマッピングされる IDL 定数によって上限値を与えることで，Java で利用できます。

配列の Holder クラスも生成されます。その名前は，次のように配列がマッピングされた Java クラス名の後ろに Holder を付けたものです。

コードサンプル 2-18　配列の Holder クラス

```
final public class <array_class>Holder
        implements org.omg.CORBA.portable.Streamable {
    public <array_element_type>[ ] value;
    public <array_class>Holder( ) {}
    public <array_class>Holder(
              <array_element_type>[ ] initial) {...}
    public void _read(org.omg.CORBA.portable.InputStream i)
        {...}
    public void _write(
                    org.omg.CORBA.portable.OutputStream o)
        {...}
    public org.omg.CORBA.TypeCode _type( ) {...}
}
```

コードサンプル 2-19　配列のマッピング

```
// IDL
const long ArrayBound = 42;
typedef long larray[ArrayBound];

// generated Java
final public class larrayHolder
        implements org.omg.CORBA.portable.Streamable {
```

```
    public int[ ] value;
    public larrayHolder( ) {}
    public larrayHolder(int[ ] initial) {...}
    public void _read(org.omg.CORBA.portable.InputStream i)
        {...}
    public void _write(
                    org.omg.CORBA.portable.OutputStream o)
        {...}
    public org.omg.CORBA.TypeCode_type( ) {...}
}
```

2.9 interface

IDL のインタフェースは，次に示す二つの Java パブリックインタフェースにマッピングされます。

- IDL インタフェースに宣言されたオペレーションと定数だけを格納する Operations インタフェース
- 該当するインタフェースオペレーション，そのすべてのベースインタフェースオペレーション，および org.omg.CORBA.Object を継承する，CORBA オブジェクト宣言

拡張子 Helper の付いた追加 Helper クラス名がインタフェース名に付加されます。Java インタフェースはマッピングしたベースインタフェース org.omg.CORBA.Object を継承します。

Java インタフェースは，マッピング後のオペレーションシグニチャを格納します。メソッドは，Java インタフェースのオブジェクトリファレンスについて呼び出せます。

Helper クラスは静的 narrow メソッドを宣言します。このメソッドによって，org.omg.CORBA.Object のインスタンスをさらに詳細な型のオブジェクトリファレンスにナロウイングできます。オブジェクトリファレンスが要求型をサポートしていないためナロウイングに失敗すると，IDL 例外 CORBA::BAD_PARAM が発生します。別の種類のエラーを報告するためには，異なるシステム例外が発生します。null のナロウイングは常に null を返して成功します。

特別な「nil」オブジェクトリファレンスはありません。Java の null は，オブジェクトリファレンスに，いつでも自由に渡されます。

属性は Java アクセッサと変更メソッドのペアにマッピングされます。これらのメソッドには IDL 属性と同じ名前があり，オーバーロードされます。IDL readonly 属性に対する変更メソッドはありません。

interface の Holder クラスも生成されます。その名前は，次のようにインタフェースがマッピングされた Java クラス名の後ろに Holder を付けたものです。

コードサンプル 2-20　interface の Holder クラス

```
final public class <interface_class>Holder
       implements org.omg.CORBA.portable.Streamable {
    public <interface_class> value;
    public <interface_class>Holder( ) {}
    public <interface_class>Holder(
                    <interface_class> initial) {
        value = initial;
    }
    public void _read(org.omg.CORBA.portable.InputStream i)
        {...}
    public void _write(
                    org.omg.CORBA.portable.OutputStream o)
        {...}
    public org.omg.CORBA.TypeCode _type( ) {...}
}
```

コードサンプル 2-21　IDL interface の Java へのマッピング

```
/*From Example.idl:*/
module Example {
    interface Foo {
        long method(in long arg)raises(AnException);
        attribute long assignable;
        readonly attribute long nonassignable;
    };
};

//Generated java
package Example;

public interface Foo extends
    com.inprise.vbroker.CORBA.Object,
    Example.FooOperations,
    org.omg.CORBA.portable.IDLEntity {
}
public interface FooOperations {
    public int method(int arg) throws Example.AnException;
    public int assignable( );
    public void assignable(int assignable);
    public int nonassignable( );
}

public final class FooHelper {
    //...other standard helper methods
    public static Foo narrow(org.omg.CORBA.Object obj)
        {...}
    public static Example.Foo bind(org.omg.CORBA.ORB orb,
        java.lang.String name,
        java.lang.String host,
        com.inprise.vbroker.CORBA.BindOptions _options)
                                                {...}
    public static Example.Foo bind(org.omg.CORBA.ORB orb,
        java.lang.String fullPoaName, byte[ ] oid)) {...}
    public static Example.Foo bind(org.omg.CORBA.ORB orb,
        java.lang.String fullPoaName,byte[ ] oid,
        java.lang.String host,
        com.inprise.vbroker.CORBA.BindOptions _options)
                                                {...}
    public Foo read(org.omg.CORBA.portable.InputStream in)
                                                {...}
    public void write(org.omg.CORBA.portable.OutputStream
                      out, Foo foo) {...}
    public Foo extract(org.omg.CORBA.Any any) {...}
    public void insert(org.omg.CORBA.Any any, Foo foo)
                                               {...}
}

public final class FooHolder
        implements org.omg.CORBA.portable.Streamable {
    public Foo value;
    public FooHolder( ) {}
    public FooHolder(final Foo initial) {...}
    public void _read(org.omg.CORBA.portable.InputStream i)
                                                 {...}
```

```
    public void _write(
                    org.omg.CORBA.portable.OutputStream o)
        {...}
    public org.omg.CORBA.TypeCode_type( ) {...}
}
```

注

リポジトリ ID の長さが，512 を 2 の（サーバエンジン数 -1）乗で割ったサイズ（単位：バイト）以下になるように，interface を IDL に記述してください。IDL で#pragma を使用して，リポジトリ ID を指定している場合も同様です。

上記のサイズを超える値を指定した場合，osagent との通信が失敗することがあります。

2.9.1 abstract インタフェース

IDL の abstract インタフェースは，Java の同名のパブリックインタフェースに 1 対 1 でマッピングされます。マッピングの規則は，abstract 以外の IDL インタフェースに Java のオペレーションインタフェースを生成する場合の規則と同様です。ただし，このインタフェースもシグニチャインタフェースとして動作するため，org.omg.CORBA.portable.IDLEntity を継承します。マッピング後の Java インタフェースは，IDL インタフェースと同じ名前になります。また，指定した型のインタフェースがほかのインタフェースで使用されている場合，メソッド宣言でシグニチャ型としても使用されます。マッピング後の Java インタフェースには，マッピングされたオペレーションシグニチャであるメソッドが格納されています。

Holder クラスは abstract 以外のインタフェースに対して生成されます。詳細については，「2.5.2 Holder クラス」を参照してください。

Helper クラスも通常の規則に従って生成されます。詳細については，「2.6 Helper クラス」を参照してください。

2.9.2 ローカルインタフェース

IDL のローカルインタフェースは，ローカル以外のインタフェースと同様にマッピングされます。ただし，org.omg.CORBA.LocalInterface でマーキングされる点がローカル以外のインタフェースと異なります。ローカルインタフェースはマーシャル処理されない可能性があるため，インプリメンテーションでは，特殊なベース org.omg.CORBA.LocalObject を継承し，生成されたシグニチャインタフェースをインプリメントする必要があります。Java のマッピングでは，LocalObject クラスは，ローカルインタフェースのインプリメンテーションのベースクラスとして使用されます。ローカルインタフェースインプリメンテーションのインスタンスは，通常の Java オブジェクトと同様に，Java 演算子 New を使用して生成します。

Helper クラスも通常の規則に従って生成されます。詳細については，「2.6 Helper クラス」を参照してください。

ローカルオブジェクトのマーシャル処理を実行しようとすると，VisiBroker ORB のインプリメンテーションによって検出され，CORBA::MARSHAL 例外が発生します。

2.9.3　パラメタの受け渡し

IDL の in パラメタは Java では通常の実パラメタにマッピングされます。IDL オペレーションの結果は，該当する Java メソッドの結果として返されます。

IDL の out パラメタおよび inout パラメタは，Java パラメタ受け渡しメカニズムに直接マッピングされません。Java でこれらのパラメタモードをインプリメントするために使用されるすべての IDL 基本型とユーザ定義型を，マッピングが定義します。IDL の out パラメタまたは inout パラメタが（コールバイバリューで）渡されるホルダ Java クラスのインスタンスを，クライアントが提供します。ホルダインスタンスの内容（インスタンス自身ではない）は呼び出しによって変更され，呼び出しが返されると，クライアントは変更（された可能性のある）内容を使用します。

コードサンプル 2-22　Java 実パラメタへの in パラメタのマッピング

```
/*From Example.idl:*/
module Example {
    interface Modes {
        long operation(in long inArg, out long outArg,
                       inout long inoutArg);
    };
};

//Generated Java:
package Example;
public interface Modes extends
    com.inprise.vbroker.CORBA.Object,
    Example.ModesOperations,
    org.omg.CORBA.portable.IDLEntity {
}
public interface ModesOperations {
    public int operation(int inArg,
        org.omg.CORBA.IntHolder outArg,
        org.omg.CORBA.IntHolder inoutArg);
}
```

コードサンプル 2-22 では，結果は通常の結果として返され，実際の in パラメタは通常の値だけで返されます。しかし，out パラメタおよび inout パラメタでは，該当するホルダが構成されなければなりません。一般的な使用例をコードサンプル 2-23 に示します。

コードサンプル 2-23　out パラメタおよび inout パラメタの Holder

```
// user Java code
// select a target object
Example.Modes target = ...;
// get the in actual value
int inArg = 57;
// prepare to receive out
IntHolder outHolder = new IntHolder( );
// set up the in side of the inout
IntHolder inoutHolder = new IntHolder(131);
// make the invocation
int result = target.operation(
                          inArg, outHolder, inoutHolder);
```

```
// use the value of the outHolder
... outHolder.value ...
// use the value of the inoutHolder
... inoutHolder.value ...
```

メソッド呼び出し前に，実際のパラメタとなるホルダインスタンスの中で，inout パラメタの入力値が設定されなければなりません。inout ホルダは，値からの新ホルダを構成するか，または該当する型の既存のホルダの値に割り当てると満たされます。呼び出し後，クライアントは outHolder.value を使用して out パラメタの値にアクセスし，inoutHolder.value を使用して inout パラメタの出力値にアクセスします。IDL オペレーションのリターン結果は，呼び出しの結果として利用できます。

2.9.4　継承によるサーバインプリメンテーション

サーバをインプリメントするいちばん簡単な方法は，継承を使用することです。それは，サーバオブジェクトとオブジェクトリファレンスが同じように見えて，同じように動作し，まったく同じコンテキストで使用できるからです。サーバオブジェクトがクライアントと同じプロセス内にある場合，メソッド呼び出しは，トランスポート，インディレクション（間接），または各種デリゲーションのない通常の Java 関数呼び出しになります。

各 IDL インタフェースがマッピングされるのは，対応する Java 版 IDL インタフェースをインプリメントする POA の abstract クラスです。

注

POA クラスが（実際に）IDL インタフェースを継承しないということは，POA は CORBA オブジェクトではないということです。POA は，CORBA サーバントであり，（実際の）CORBA オブジェクトを作成するときに使用できます。POA クラスの詳細については，「4.　コアインタフェースとクラス (Java)」を参照してください。

ユーザ定義サーバクラスはコードサンプル 2-24 に示すとおり，<インタフェース名>POA クラスを継承することで VisiBroker ORB にリンクされます。

注

POA クラス自体は abstract 型であり，実体化できません。POA クラスを実体化するには，宣言した IDL インタフェースオペレーションをインプリメントしてください。

コードサンプル 2-24　継承を使用する Java でのサーバインプリメンテーション

```
/*From Bank.idl:*/
module Bank {
    interface Account {
    };
};

// Generated java
package Bank;
public abstract class AccountPOA extends
    org.omg.PortableServer.Servant implements
    org.omg.CORBA.portable.InvokeHandler,
    Bank.AccountOperations {...}
```

```
// Linking an implementation to the ORB :
public class AccountImpl extends Bank.AccountPOA {...}
```

2.9.5　デリゲーションを使用したサーバインプリメンテーション

サーバをインプリメントするための継承には一つ欠点があります。サーバクラスは POA スケルトンクラスを継承するため，ほかの目的でインプリメンテーション継承を使用できません。それは，Java が単一の継承だけをサポートしているからです。サーバクラスが，ほかの目的で利用できる単独の継承リンクを使用しなければならない場合，デリゲーションする必要があります。

サーバクラスがデリゲーションを使用してインプリメントされる場合，追加のコードが生成されます。

- 各インタフェースは，POA スケルトンを継承する Tie クラスにマッピングされ，デリゲーションコードを提供します。
- 各インタフェースは，Tie クラスがデリゲーションしているオブジェクトの型を定義するために使用される Operations インタフェースにもマッピングされます。

デリゲートされたインプリメンテーションは Operations インタフェースをインプリメントする必要があり，Tie クラスのインスタンスに格納しなければいけません。Operation インタフェースのインスタンスを Tie オブジェクトに格納するのは，Tie クラスで提供されるコンストラクタです。デリゲーションの使用例をコードサンプル 2-25 に示します。

コードサンプル 2-25　デリゲーションを使用した Java でのサーバインプリメンテーション

```
/*From Bank.idl:*/
module Bank {
    interface AccountManager {
        Account open(in string name);
    };
};

//Generated java
package Bank;
public interface AccountManagerOperations {
    public Example.Account open(java.lang.String name);
}
//Generated java
package Bank;
public class AccountManagerPOATie extends
        AccountManagerPOA {
    public AccountManagerPOATie(
            final Bank.AccountManagerOperations _delegate)
          {...}
    public AccountManagerPOATie(
            final Bank.AccountManagerOperations _delegate,
            final org.omg.PortableServer.POA _poa){...}
    public Bank.AccountManagerOperations _delegate ( ){...}
    public void _delegate(
            final Bank.AccountManagerOperations delegate)
                                                {...}
```

```
    public org.omg.PortableServer.POA _default_POA( ){...}
    public float open( ){...}
}
//Linking an implementation to the ORB :
//classAccountImpl implements AccountManager Operations
public class Server {
    public static main(String args){
        //...
        AccountManagerPOAtie managerServant =
            new AccountManagerPOATie(
                new AccountManagerImpl( ));
        //...
}
```

2.9.6 インタフェーススコープ

OMG の IDL/Java 言語マッピング仕様は，インタフェースのスコープ内で宣言をネストしたり，パッケージとインタフェースに同じ名前を付けたりすることを認めていません。そこで，インタフェーススコープは，「Package」が後ろに付いた同じ名前のパッケージにマッピングされます。

2.10　例外のマッピング

IDL 例外は structs とまったく同じようにマッピングされます。例外のフィールドのインスタンス変数やコンストラクタを提供する Java クラスに，IDL 例外はマッピングされます。

CORBA システム例外はチェックされない例外です。CORBA システム例外は，java.lang.RuntimeException から（間接的に）継承されます。

ユーザ定義例外はチェックされる例外です。ユーザ定義例外は，java.lang.Exception から（間接的に）継承されます。

2.10.1　ユーザ定義例外

ユーザ定義例外は，org.omg.CORBA.UserException を継承するファイナル Java クラスにマッピングされます。そうでない場合，Helper クラスおよび Holder クラスの生成がある IDL struct 型のようにマッピングされます。

例外が，ネストされた IDL スコープ内（特に，インタフェース内）で定義される場合，その Java クラス名は特殊スコープ内に定義されます。そうでない場合，その Java クラス名は，例外のある IDL モジュールに対応する Java パッケージのスコープ内に定義されます。

コードサンプル 2-26　ユーザ定義例外のマッピング

```
//IDL
module Example {
    exception AnException {
        string reason;
    };
};

//Generated Java
package Example;
public final class AnException extends
    org.omg.CORBA.UserException {
    public java.lang.String extra;
    public AnException( ) {...}
    public AnException(java.lang.String extra) {...}
    public AnException(java.lang.String _reason,
                       java.lang.String extra) {...}
    public java.lang.String to String( ) {...}
    public boolean equals(java.lang.Object o) {...}
}

public final class AnExceptionHolder implements
        org.omg.CORBA.portable.Streamable {
    public Example.AnException value;
    public AnExceptionHolder( ){}
    public AnExceptionHolder(
               final Example.AnException _vis_value) {...}
    public void _read(final org.omg.CORBA.portable.
```

```
                                    InputStream input) {...}
    public void _write(final org.omg.CORBA.portable.
                                   OutputStream output) {...}
    public org.omg.CORBA.TypeCode _type( ) {...}
}
```

2.10.2 システム例外

標準 IDL システム例外は，org.omg.CORBA.SystemException を継承するファイナル Java クラスにマッピングされます。例外の理由を記述する文字列と同様に，IDL のメジャーおよびマイナー例外コードへのアクセスを提供します。org.omg.CORBA.SystemException のパブリックコンストラクタはありません。それを継承するクラスだけが実体化できます。

各標準 IDL 例外の Java クラス名は，その IDL 名と同じで，org.omg.CORBA パッケージにあるよう宣言されます。デフォルトコンストラクタは，マイナーコードに 0，完了コードに COMPLETED_NO，理由文字列に空の文字列（""）を提供します。理由文字列を取りながら，ほかのフィールドにデフォルトを使用するコンストラクタもあれば，三つのパラメタに指定が必要なコンストラクタもあります。

2.11　Any 型のマッピング

IDL 型 Any は，Java クラス org.omg.CORBA.Any にマッピングされます。このクラスには，事前に定義された型のインスタンスを挿入，または抽出するために必要な，すべてのメソッドがあります。抽出オペレーションに不一致型があると，CORBA::BAD_OPERATION 例外が発生します。

さらに，ポータブルスタブとスケルトンで使用される高速インタフェースを提供するため，Holder クラスを取る挿入メソッドおよび抽出メソッドが定義されます。非基本 IDL 型の場合を処理するために，一般的なストリーム可能な型用のメソッドがあるほかに，各基本 IDL 型用に定義された挿入メソッド，および抽出メソッドがあります。挿入メソッドおよび抽出メソッドの詳細については，「5.1.2　Any の抽出メソッド」および「5.1.3　Any の挿入メソッド」を参照してください。

挿入オペレーションは指定された値を設定し，必要に応じて Any の型をリセットします。

type( )アクセッサでのタイプコードの設定は，値を消去します。値が設定される前に抽出しようとすると，CORBA::BAD_OPERATION 例外が発生します。このオペレーションは，主に IDL out パラメタに型が正しく設定されるようにするために提供されています。

2.12　ネストされた型のマッピング

IDL では型宣言をインタフェース内でネストできます。Java ではクラスをインタフェース内でネストできません。そのため，インタフェースのスコープ内で宣言される IDL 型は，Java クラスにマッピングされる際，特殊なスコープパッケージ内にマッピングされます。

これらの型宣言を含む IDL インタフェースは，スコープパッケージを生成し，マッピングされた Java クラス宣言を含みます。スコープパッケージ名は，IDL 型名に「Package」を付けて構成されます。

コードサンプル 2-27　ネストされた型のマッピング

```
// IDL
module Example {
    interface Foo {
      exception e1 {};
    };
};

// generated Java
package Example.FooPackage;
final public class e1 extends
    org.omg.CORBA.UserException {...}
```

2.13　typedef のマッピング

Java 2 SDK 1.4 の言語仕様の変更によって，パッケージのないクラスの import 文は，コンパイルエラーとなるようになりました。IDL ファイル内で typedef 宣言を行う場合は，module 文の中で宣言するようにしてください。

2.13.1　シンプル IDL 型

シンプル Java 型にマッピングされる IDL 型は，Java にサブクラス化されないことがあります。このため，シンプル型の型宣言である typedef は，typedef 型が現れるオリジナル（マッピングされた型）にマッピングされます。シンプル IDL 型では，すべての typedef で Helper クラスが生成されます。

2.13.2　複合 IDL 型

非配列および非シーケンスの typedef は，各種非 typedef のシンプル IDL 型，またはユーザ定義 IDL 型が現れるまで，オリジナルの型に「展開」されません。

Holder クラスは，シーケンス typedef および配列 typedef で生成されます。

コードサンプル 2-28　複合 idl typedef のマッピング

```
// IDL
struct EmpName {
   string firstName;
   string lastName;
};
typedef EmpName EmpRec;

// generated Java
    // regular struct mapping for EmpName
    // regular helper class mapping for EmpRec

final public class EmpName {
    ...
}

public class EmpRecHelper {
    ...
}
```

3 生成されるインタフェースとクラス（Java）

この章では，idl2java コンパイラで生成されるクラスについて説明します。

3.1 概要

org.omg.CORBA パッケージのほとんどのクラスには，Helper クラスと Holder クラスが提供されています。

また，ユーザが定義した型に対しては，idl2java コンパイラが Helper クラスと Holder クラスを生成します。Helper クラスと Holder クラスには，型に対して生成されたクラスの名前の末尾に「Holder」または「Helper」を付けた名前が付けられます。例えば，MyType と名づけられたユーザ定義型を指定すると，idl2java コンパイラでは次のように生成されます。

- public class MyType
- public class MyTypeOperations
- public class MyTypeHelper
- public class MyTypeHolder
- public class MyTypePOA
- public class MyTypePOATie
- public class _MyTypeStub

3.1.1 Signature クラスと Operations クラス

これらのクラス（MyType と MyTypeOperations）は，Java にマッピングされると，ユーザの IDL インタフェースの完全なシグニチャを提供します。

Signature クラス
: Signature クラスは，IDL ファイルに宣言するインタフェースごとにシグニチャインタフェースを定義します。

Operations クラス
: Operations クラスは，オブジェクトインプリメンテーションでインプリメントされなければならないすべてのメソッドを定義します。このクラスは，tie 機能を使用する場合，対応する tie クラスのデリゲーションオブジェクトとして動作します。

3.1.2 補助クラス

すべてのユーザ定義型には Helper クラスと Holder クラスが生成されます。

Helper クラス
: abstract Helper クラスは，idl2java コンパイラが生成し，関連オブジェクトで動作させるためのユティリティメソッドを提供します。Helper クラスの目的は，必要のないときにクラスが提供するメソッドをロードしないようにするためです。

マッピングした構造体，列挙体，共用体，例外，valuetype，valuebox などのオブジェクトに対して，Helper クラスはストリームからオブジェクトを読み込んだり，書き込んだり，オブジェクトのリポジトリ ID を返したりするためのメソッドを提供しています。インタフェース用に生成された Helper クラスには，bind や narrow のようなメソッドもあります。

Holder クラス

Java 言語では，パラメタは参照型ではなく，値でだけ渡されます。Holder クラスは，オペレーションリクエストに対応する out パラメタ，および inout パラメタの受け渡しだけをサポートするために使用されます。Holder クラスのインタフェースは，すべての型で統一されています。

3.1.3 ポータビリティスタブインタフェースとポータビリティスケルトンインタフェース

IDL から Java 言語へのマッピングには，オブジェクトのローカル呼び出しとリモート呼び出しの両方に使用できるスタブクラスが定義されています。スケルトンクラスはストリーム単位または DSI 単位で生成されます。

注

ローカルインタフェースには，スタブクラスもスケルトンクラスもありません。ローカルインタフェースの詳細については，「2.9.2 ローカルインタフェース」を参照してください。

スタブクラス

スタブクラスは，クライアントが呼び出す<interface_name>にスタブのインプリメンテーションを提供します。すべてのスタブは org.omg.CORBA.portable.ObjectImpl から継承します。

POA クラス

ストリームベースのスケルトンは org.omg.PortableServer.Servant を継承し，DSI ベースのスケルトンは org.omg.PortableServer.DynamicImplementation を継承します。abstract クラスの POA は，サーバ側のクラスのインプリメンテーションへのインタフェースを提供します。

POATie クラス

POATie クラスは<interface_name>POA クラスを継承します。POATie クラスによって，<interface_name>Operations メソッドの呼び出しを，同じメソッドをインプリメントするほかのオブジェクトにデリゲートできます。POATie クラスを使用しなくてもかまいませんが，POATie クラスを使用しないときは，<interface_name>Impl クラスに直接 abstract クラスの<interface_name>POA を継承させなければならないことがあります。

3.2 <interface_name>Operations

```
public interface <interface_name>Operations
```

idl2java コンパイラが生成する Operations クラスは，IDL ファイルに記述したインタフェース（<interface_name>）に対して宣言したメソッドと定数のインタフェース定義を格納します。

3.3 <type_name>Helper

```
abstract public class <type_name>Helper
```

Helper クラスは，org.omg.CORBA パッケージのほとんどのクラスに提供されます。また，Helper クラスは，すべてのユーザ定義型用に idl2java コンパイラで生成されます。型用に生成されるクラス名の後ろには Helper が追加されます。各種静的メソッドがクラスを処理するために提供されています。

3.3.1 Helper のメソッド

- public static <interface_name> **extract**(
  org.omg.CORBA.Any **any**)
  このメソッドは，指定された Any オブジェクトから型を抽出します。
  - any
    オブジェクトを含んでいる Any オブジェクト
- public static String **id** ( )
  このメソッドは，オブジェクトのリポジトリ ID を取得します。
- public static void **insert**(
  org.omg.CORBA.Any **any**,
  <type_name> **value**)
  このメソッドは，指定された Any オブジェクトに型を挿入します。
  - any
    型を含むための Any オブジェクト
  - value
    挿入する型
- public static <type_name> **read**(
  org.omg.CORBA.portable.InputStream **input**)
  このメソッドは，指定された入力ストリームから型を読み出します。
  - input
    オブジェクトが読み出される入力ストリーム
- public static org.omg.CORBA.TypeCode **type**( )
  このメソッドは，該当するオブジェクトに対応する TypeCode を返します。リターン値の一覧については，「5.23　TCKind」を参照してください。
- public static void **write**(
  org.omg.CORBA.portable.OutputStream **output**,

<type_name> value)
このメソッドは，指定された出力ストリームに型を書き込みます。

- output
  オブジェクトが書き込まれる出力ストリーム
- value
  出力ストリームに書き込まれる型

3.3.2 インタフェース用に生成されるメソッド

- public static <interface_name> bind(
  org.omg.CORBA.ORB orb)
  このメソッドは，<interface_name>型のオブジェクトの任意のインスタンスにバインドします。
- public static <interface_name> bind(
  org.omg.CORBA.ORB orb,
  java lang String name)
  このメソッドは，指定したインスタンス名のある<interface_name>型のオブジェクトにバインドします。
  - name
    指定したオブジェクトのインスタンス名
- bind(
  org.omg.CORBA.ORB orb,
  java.lang.String fullPoaName, byte[ ] oid)
  - orb
    サーバが使用する VisiBroker ORB。ORB.init メソッドの呼び出しによって取得されます。
  - fullPoaName
    絶対パスで記述された POA 名
  - oid
    リファレンスを生成するオブジェクトの識別子
- public static <interface_name> narrow(
  org.omg.CORBA.Object object)
  このメソッドは，org.omg.CORBA.Object リファレンスを，<interface_name>型のオブジェクトにナロウイングします。オブジェクトリファレンスをナロウイングできない場合，CORBA.BAD_PARAM 例外または null が返されます。
  - object
    <interface_name>型にナロウイングされるオブジェクト

3.3.3　オブジェクトラッパー用に生成されるメソッド

次に示すメソッドは，idl2java を-obj_wrapper オプションで発行する場合に，Helper クラスに生成されます。-obj_wrapper オプションの詳細については，「1.4　idl2java」を参照してください。オブジェクトラッパー機能の詳細については，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「オブジェクトラッパーの使用」の記述を参照してください。

- public static void addClientObjectWrapperClass(
  org.omg.CORBA.ORB orb, java.lang.Class c)
  クライアントアプリケーションからタイプドオブジェクトラッパーを追加します。タイプドオブジェクトラッパーが複数インストールされている場合は，登録された順に追加されます。

  注

  このメソッドは，クライアントアプリケーションだけで呼び出してください。

  - orb
    クライアントが使用する VisiBroker ORB。ORB.init メソッドの呼び出しで取得されます。ただし，このバージョンでは，orb パラメタは使用できません。
  - c
    追加するオブジェクトラッパー
- public static void addServerObjectWrapperClass(
  org.omg.CORBA.ORB orb, java.lang.Class c)
  サーバアプリケーションからタイプドオブジェクトラッパーを追加します。タイプドオブジェクトラッパーが複数インストールされている場合は，登録された順に追加されます。

  注

  このメソッドは，サーバアプリケーションだけで呼び出してください。

  - orb
    サーバが使用する VisiBroker ORB。ORB.init メソッドの呼び出しで取得されます。ただし，このバージョンでは，orb パラメタは使用できません。
  - c
    追加するオブジェクトラッパー
- public static void removeClientObjectWrapperClass(
  org.omg.CORBA.ORB orb, java.lang.Class c)
  クライアントアプリケーションからタイプドオブジェクトラッパーを削除します。

  注

  このメソッドは，クライアントアプリケーションだけで呼び出してください。

  - orb
    クライアントが使用する VisiBroker ORB。ORB.init メソッドの呼び出しで取得されます。
  - c

削除するオブジェクトラッパー

- public static void removeServerObjectWrapperClass(
  org.omg.CORBA.ORB orb, java.lang.Class c)
  サーバアプリケーションからタイプドオブジェクトラッパーを削除します。

  注

  このメソッドは，サーバアプリケーションだけで呼び出してください。

  - orb
    サーバが使用する VisiBroker ORB。ORB.init メソッドの呼び出しで取得されます。
  - c
    削除するオブジェクトラッパークラス

3.4 <type_name>Holder

```
public final class <type_name>Holder
```

Holder クラスは，org.omg.CORBA パッケージ内のすべての基本 IDL 型に提供されます。また，idl2java コンパイラによって，すべてのユーザ定義型に対しても Holder クラスが生成されます。ユーザ定義型のために生成されたクラス名の後ろに Holder が追加されます。各 Holder には，コンストラクタと型付きの値であるメンバのセットがあります。

基本型の Holder クラスは，次のように定義されます。基本型の Holder クラスは，org.omg.CORBA パッケージ内にあります。

- public class ShortHolder
- public class IntHolder
- public class LongHolder
- public class ByteHolder
- public class FloatHolder
- public class DoubleHolder
- public class CharHolder
- public class BooleanHolder
- public class StringHolder
- public class ObjectHolder
- public class AnyHolder
- public class TypeCodeHolder
- public class PrincipalHolder

ユーザ定義型<type_name>の Holder クラスをコードサンプル 3-1 に示します。

コードサンプル 3-1　Holder クラス

```
// Java
final public class <type_name>Holder
   implements org.omg.CORBA.portable.Streamable {
   public <type_name> value;
   public <type_name>Holder( ) {}
   public <type_name>Holder(<type_name> initial)
                                                    {}
   public void _read(org.omg.CORBA.portable.InputStream i)
       {...};
   public void _write(
                    org.omg.CORBA.portable.OutputStream o)
       {...};
```

```
    public org.omg.CORBA.TypeCode _type( ) {...}
}
```

3.4.1 メンバデータ

```
public <type_name> value
```

この値は，このオブジェクトが含む型を表します。

3.4.2 Holder のメソッド

- public <type_name> Holder( )
  デフォルトコンストラクタは，out パラメタで使用すると便利です。デフォルトコンストラクタは，Java 言語の定義に従って，値フィールドに型のデフォルト値を設定します。値は，boolean 型では false に設定されます。整数型および char 型では 0 に，文字列およびオブジェクトリファレンスでは null が設定されます。
- public <type_name> Holder(
  <interface_name> initial)
  値コンストラクタは，inout パラメタで使用すると便利です。value フィールドは指定された Any オブジェクトの value フィールドからコピーされます。
  - initial
    Holder が含んでいる，そのほかのオブジェクト

3.5 _<interface_name>Stub

```
abstract public class _<interface_name>Stub
```

idl2java コンパイラが生成するスタブクラスは，クライアントが呼び出す<interface_name>にスタブのインプリメンテーションを提供します。このクラスが提供するインプリメンテーションは，オブジェクトのインプリメンテーションを透過的に動作させます。

3.6 <interface_name>POA

```
abstract public class <interface_name>POA
```

このクラスは，指定したインタフェースに POA スケルトン（サーバント）クラスを提供します。このクラスは<interface_name>Operations インタフェースにインプリメンテーションを提供しません。<interface_name>Impl クラスをインプリメントする際に，このクラスを継承してください。

3.7 <interface_name>POATie

```
abstract public class <interface_name>POATie
```

idl2java コンパイラが生成する tie クラスは，<interface_name>POA にデリゲータクラスを作成します。

3.7.1 tie のメソッド

- public <interface_name>**POATie**(
  final <interface_name>Operations **_delegate**)
  このメソッドはコンストラクタです。<interface_name>Operations インタフェースをインプリメントしてデリゲートオブジェクトとして使用するオブジェクトをこのメソッドに渡します。
- public <interface_name>**POATie**(
  final <interface_name>Operations **_delegate**,
  final org.omg.PortableServer.POA **_poa**)
  このメソッドは，デリゲートオブジェクトとデフォルト POA サーバントを初期化するコンストラクタです。

4 コアインタフェースとクラス（Java）

この章では，Java 言語のコアインタフェースとクラスについて説明します。

4.1 CompletionStatus

```
public final class org.omg.CORBA.CompletionStatus extends
    java.lang.Object implements org.omg.CORBA.portable.IDLEntity
```

このクラスは，SystemException で動作し，例外が発生する前にオペレーションが完了したかどうかを示します。

4.1.1 IDL の定義

```
enum CompletionStatus {
    COMPLETED_YES,
    COMPLETED_NO,
    COMPLETED_MAYBE
};
```

4.1.2 CompletionStatus のメンバ

- COMPLETED_YES
  オペレーションが無事完了したことを表します。
- COMPLETED_NO
  例外，またはエラーの発生のため，オペレーションが開始されなかったことを表します。
- COMPLETED_MAYBE
  例外，またはエラーの発生のため，オペレーションが完了しなかった可能性があることを表します。

4.1.3 CompletionStatus のメソッド

このメソッドの詳細については，「2.8.1 enum」を参照してください。

```
public final static int _COMPLETED_YES
public final static int _COMPLETED_NO
public final static int _COMPLETED_MAYBE
public final static CompletionStatus COMPLETED_YES
public final static CompletionStatus COMPLETED_NO
public final static CompletionStatus COMPLETED_MAYBE
public int value( )
public static CompletionStatus from_int(int value)
```

4.2 Context

```
public abstract class org.omg.CORBA.Context extends
    java.lang.Object
```

このクラスには，クライアントのプロパティリストを含みます。このプロパティリストは，クライアントがリクエストをするときにサーバにプロパゲーションされます。CORBA 仕様では Context の内容を定義しないため，これらのプロパティを定義するのは，ユーザと実装者に任されます。Context オブジェクトは，親コンテキストに対するポインタをそれぞれ含むツリーとして編成されます。ルートコンテキストは global default context で，親は null になります。デフォルトコンテキストは，ORB.get_default_context メソッドを使用することで取得されます。ORB.get_default_context メソッドの詳細については，「4.5 ORB」を参照してください。

org.omg.CORBA.Context に設定するプロパティの値に string 以外の IDL 型を指定すると，invoke 時に org.omg.CORBA.BAD_PARAM が発生します。

org.omg.CORBA.Context に設定するプロパティの flags に 0 以外の値を指定しても，invoke 時に org.omg.CORBA.INV_FLAGS が発生しません。

4.2.1 IDL の定義

```
interface Context {
    CORBA::Identifier context_name( );
    CORBA::Context parent( );
    void set_one_value(
        in CORBA::Identifier prop_name,
        in any value
    );
    void set_values(
        in CORBA::NVList values
    );
    CORBA::NVList get_values(
        in CORBA::Identifier start_scope,
        in boolean restrict_scope,
        in CORBA::Identifier prop_name
    );
    void delete_values(
        in CORBA::Identifier prop_name
    );
    CORBA::Context create_child(
        in CORBA::Identifier context_name
    );
};
```

4.2.2 Context のメソッド

- public java.lang.String **context_name**( )
  このメソッドは，この Context 名を返します。
- public Context **create_child**(
  java.lang.String **context_name**)
  このメソッドは，指定された親 Context で子 Context を生成します。また，新しく生成された子コンテキストを返します。
  - context_name
    生成される子コンテキスト名
- public void **delete_values**(
  java.lang.String **prop_name**)
  このメソッドは，カレントの Context から指定された名前ですべてのプロパティを削除します。prop_name のいちばん後ろには，ワイルドカードとしてアスタリスク（*）を使用できます。
  - prop_name
    削除されるプロパティ名
- public org.omg.CORBA.NVList **get_values**(
  java.lang.String **start_scope**,
  int **restrict_scope**,
  java.lang.String **prop_name**)
  このメソッドは，カレントの Context に対応するプロパティを名前と値のペアである NVList として返します。Context 検索の範囲は，start_scope パラメタおよび restrict_scope パラメタを使用して制限できます。prop_name のいちばん後ろには，ワイルドカードとしてアスタリスク（*）を使用できます。
  このメソッドは，指定された検索の結果の名前と値のリストを返します。start_scope が null でなく，該当する Context が見つからない場合，このメソッドは BAD_PARAM 例外となります。
  また，このメソッドは，prop_name に空の文字列を指定すると BAD_PARAM 例外となります。
  prop_name に指定したプロパティ名が見つからない場合は，要素数が 0 の org.omg.CORBA.NVList が返ります。
  メモリの確保に失敗した場合は，NO_MEMORY 例外になります。
  - start_scope
    検索を始めるコンテキスト名
  - restrict_scope
    オペレーションフラグ。設定可能なフラグの一つは CTX_RESTRICT_SCOPE です。このフラグが設定されると，メソッドによる探索は設定された start_scope 内または Context オブジェクト内に制限されます。
  - prop_name

返されるプロパティ名

- public org.omg.CORBA.Context **parent**( )
  このメソッドは，このオブジェクトの親 Context を返します。オブジェクトがデフォルトグローバルコンテキストである場合，NULL を返します。
- public void **set_one_value**(
  java.lang.String **prop_name**,
  org.omg.CORBA.Any **value**)
  このメソッドは，新規プロパティをカレントの Context に追加します。プロパティの値は，Any クラスで表示されます。Any の詳細については，「5.1　Any」を参照してください。
  - prop_name
    新規プロパティ名
  - value
    新規プロパティの値を含む Any オブジェクト
- public void **set_values**(
  org.omg.CORBA.NVList **values**)
  このメソッドは，複数の名前と値のペアを含む，提供された NVList を使用して，カレントの Context のプロパティを設定します。NVList については，「5.19　NVList」を参照してください。
  - values
    Context のプロパティリスト

4.3 InvalidName

```
public final class org.omg.CORBA.ORBPackage.InvalidName extends
    org.omg.CORBA.UserException
```

ORB.resolve_initial_references メソッドでこの例外が発生します。詳細については，「4.5.2 JDK での ORB メソッド」の resolve_initial_references メソッドを参照してください。

このクラスの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

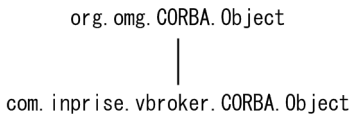
4.4 Object

```
public interface org.omg.CORBA.Object
```

このインタフェースは，CORBA 継承階層のルートです。IDL で定義されたすべてのインタフェースは，このインタフェースから継承されます。このインタフェースは，プラットフォームに依存しないランタイムの型情報とオブジェクトリファレンスの等価テストを提供します。

VisiBroker では org.omg.CORBA.Object にメソッドを幾つか追加して拡張しています。この拡張の階層関係を次の図に示します。

図 4-1　org.omg.CORBA.Object の階層と Object インタフェースの位置づけ



この二つのインタフェースの詳細を次に説明します。

4.4.1 org.omg.CORBA.Object の定義

```
package org.omg.CORBA;
public interface Object {
    Request _create_request(
        Context ctx,
        String operation,
        NVList arg_list,
        NamedValue result
    );

    Request _create_request(
        Context ctx,String operation,
        NVList arg_list,
        NamedValue result,
        ExceptionList exclist,
        ContextList ctxlist
    );
    org.omg.CORBA.Object _duplicate( );
    DomainManager [ ] _get_domain_managers( );
    org.omg.CORBA.Object _get_interface_def( );
    Policy _get_policy(int policy_type);
    int _hash(int maximum);
    boolean _is_a(String repositoryIdentifier);
    boolean _is_equivalent(org.omg.CORBA.Object other);
    boolean _non_existent( );
    void _release( );
    Request _request(String operation);
    org.omg.CORBA.Object _set_policy_override(
        Policy [ ] policies,
        SetOverrideType set_add
```

```
    );
}
```

4.4.2　org.omg.CORBA.Object のメソッド

- public org.omg.CORBA.Request _create_request(
  org.omg.CORBA.Context ctx,
  java.lang.String operation,
  org.omg.CORBA.NVList arg_list,
  org.omg.CORBA.NamedValue result)
  このメソッドは，指定されたパラメタで初期化される動的起動リクエストを生成します。デフォルトコンテキストの取得については，「4.5.2　JDK での ORB メソッド」の get_default_context メソッドを参照してください。
  - ctx
    動的起動リクエストに使用される Context
  - operation
    呼び出されるオペレーション名
  - arg_list
    NamedValue 項目のリスト。オペレーションに渡される各パラメタには，NamedValue が一つあります。
  - result
    オペレーションの結果
- public org.omg.CORBA.Request _create_request(
  org.omg.CORBA.Context ctx,
  java.lang.String operation,
  org.omg.CORBA.NVList arg_list,
  org.omg.CORBA.NamedValue result,
  org.omg.CORBA.ExceptionList exceptions,
  ContextList contexts)
  このメソッドは，指定パラメタで初期化される動的起動リクエストを生成します。動的起動リクエストの生成には，リクエストで発生する例外のリストが含まれます。デフォルトコンテキストの取得については，「4.5.2　JDK での ORB メソッド」の get_default_context メソッドを参照してください。
  - ctx
    動的起動リクエストに使用される Context
  - operation
    呼び出されるオペレーション名

- arg_list

  NamedValue 項目のリスト。オペレーションに渡される各パラメタには，NamedValue が一つあります。

- result

  オペレーションの結果

- exceptions

  該当するリクエストで発生する例外を表す，Typecode オブジェクトのリスト

- contexts

  Context オブジェクトのリスト。Context オブジェクトのリストは，コンテキスト名をチェックする型をサポートします。

- public int _hash(

  int maximum)

  このメソッドは，0 から maximum の範囲の，このオブジェクトのハッシュ値を算出します。常に正の値を返します。

  - maximum

    返される最大ハッシュ値

- public boolean _is_a(

  java.lang.String repid)

  このメソッドは，指定されたインタフェースをインプリメントしているかどうかを知るためにオブジェクトに問い合わせます。このメソッドは，インプリメンテーションオブジェクトがインタフェースをサポートしている場合，true を返します。そうでない場合，false を返します。

  - repid

    要求するインタフェースのリポジトリ ID を含む文字列

  注

  次の例に示すように，指定されたサーバは多重継承で複数のインタフェースを同時にインプリメントできます。そのため，このメソッドの呼び出しは，インプリメンテーションオブジェクトに対する呼び出しとなる場合があります。

```
module M {
    interface A {
        void opA( );
    };
    interface B {
        void opB( );
    };
    interface C : A ,B {
    };
};
```

モジュール M にインタフェース A が指定されている場合（M::A），該当するリポジトリ ID は，一般的に IDL:M/A:1.0 です。リポジトリ ID は，IDL ファイルの中で#pragmas を使用して，任意文字列にも設定されます。

- public boolean _is_equivalent(
  org.omg.CORBA.Object other_object)
  このメソッドは，このオブジェクトのインターオペラブルオブジェクトリファレンス（IOR）を指定されたオブジェクトの IOR と比較し，IOR が同じ値である場合 true を返します。そうでない場合，false を返します。同じインプリメンテーションオブジェクトが複数の IOR で参照される場合，false を取得することがあります。
  - other_object
    このオブジェクトと比較されるオブジェクトのリファレンス
- public boolean _non_existent( )
  このメソッドは，インプリメンテーションオブジェクトが活性化されているかどうかを調べるためにインプリメンテーションオブジェクトを ping します。このメソッドは，インプリメンテーションオブジェクトが現在，活性化されている場合（または，サーバを活性化したあとで），false を返します。そうでない場合，true を返します。このメソッドは，クライアントのプロキシオブジェクトがほかのサーバに再バインドする原因にはなりません。つまり，強制的にバインドをしますが，強制的に再バインドはしません。サーバにアクセスできない場合は，TRANSIENT 例外を返します。
  vbroker.orb.compliantExceptions プロパティを false に設定すると，例外ではなく true を返します。これによって，下位互換性が保たれます。
  リバインドポリシーが VB_TRANSPARENT と VB_NOTIFY_REBIND の場合に，クライアントがサーバに対して通信障害を検知した状態で，同一リファレンスを使用して org.omg.CORBA.Object._non_existent( )を発行すると，暗黙的な接続を試みずに TRANSIENT 例外が発生します。この状態で再接続による確認を行うためには，org.omg.CORBA.Object._non_existent( )以外のユーザメソッドを使用してください。
- public org.omg.CORBA.Request _request(
  java.lang.String operation)
  このメソッドは，空の動的起動リクエストを生成します。オペレーションの IN パラメタおよび INOUT パラメタの型と値，OUT パラメタの型，およびリターン値の型は，リクエスト送信前に初期化する必要があります。オペレーションによるユーザ例外が発生する場合は，ユーザ例外のタイプコードのリストをリクエスト送信前に初期化する必要があります。コンテキストを使用する場合は，コンテキストリストをリクエスト送信前に初期化する必要があります。動的起動リクエストの初期化と送信の詳細については，「5.21　Request」を参照してください。
  - operation
    呼び出されるオペレーション名

4.4.3 VisiBroker での Object の継承

```
Package com.inprise.vbroker.CORBA;

public interface Object extends org.omg.CORBA.Object {
    public void _bind( );
    public org.omg.CORBA.Policy _get_client_policy(
                                        int policy_type);
    public org.omg.CORBA.Policy[ ] _get_policy_overrides(
                                        int[ ] types);
    public com.inprise.vbroker.IOP.IOR _ior( );
    public com.inprise.vbroker.IOP.IORValue _ior_value( );
    public boolean _is_bound( );
    public boolean _is_local( );
    public boolean _is_persistent( );
    public boolean _is_remote( );
    public java.lang.String _object_name( );
    public org.omg.CORBA.ORB _orb( );
    public java.lang.String _repository_id( );
    public org.omg.CORBA.Object _resolve_reference(
                                    java.lang.String id);
    public boolean _validate_connection(
        org.omg.CORBA.PolicyListHolder
                                    inconsistent_policies
    );
}
```

4.4.4 VisiBroker での Object のメソッドの継承

- public boolean _is_bound( )

  このメソッドは，TCP コネクションがインプリメンテーションオブジェクトで設定されている場合，true を返します。そうでない場合，false を返します。

- public boolean _is_local( )

  このメソッドは，このオブジェクトがローカルアドレス空間内でインプリメントされたオブジェクトを参照する場合，true を返します。そうでない場合，false を返します。

- public boolean _is_persistent( )

  このメソッドは，このオブジェクトリファレンスがオブジェクトをインプリメントするプロセスの存続期間を超えて有効である場合に true を返します。そうでない場合，false を返します。

- public boolean _is_remote( )

  このメソッドは，このオブジェクトがリモートアドレス空間でインプリメントされたオブジェクトを参照している場合に true を返します。そうでない場合，false を返します。

- public java.lang.String _repository_id( )

  このメソッドは，オブジェクトインプリメンテーションのインタフェースのリポジトリ ID を返します。

- public org.omg.CORBA.Object _resolve_reference(
  java.lang.String id)

このメソッドは，指定されたサービス識別子でサーバ側のインタフェースを解決するため，クライアントアプリケーションからオブジェクトリファレンスに対して呼び出されます。このメソッドは，サーバ側で ORB.resolve_initial_references メソッドを呼び出させます。ORB.resolve_initial_references メソッドの詳細については，「4.5.2　JDK での ORB メソッド」の resolve_initial_references メソッドを参照してください。クライアントが特定のサーバ型にナロウイングできる，オブジェクトリファレンスを返します。

- id

  サーバ側で解決されるインタフェース名

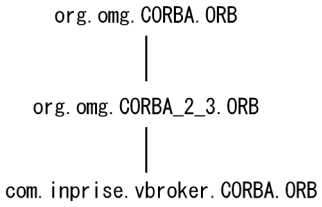
4.5 ORB

```
public abstract class org.omg.CORBA.ORB extends java.lang.Object
```

このクラスは，コードサンプル 4-1 に示すように，CORBA の基盤を初期化するメソッドを提供します。ORB は，クライアントとサーバの両方で使用される各種メソッドを提供します。

JDK には org.omg.CORBA.ORB クラスがバンドルされています。VisiBroker は，org.omg.CORBA.ORB クラスにメソッドを幾つか追加しています。この拡張の階層関係を次の図に示します。

図 4-2 org.omg.CORBA.ORB の階層と ORB インタフェースの位置づけ



コードサンプル 4-1 ORB クラスのクライアント使用例

```
public class SimpleClientProgram {
    public static void main(String args[ ]) {
        try {
          org.omg.CORBA.ORB orb =
                        org.omg.CORBA.ORB.init(args,null);
          org.omg.CORBA.Object object =
                           orb.string_to_object(args[0]);
          System.out.println(
                          "Contacted object: " + object);
        }
        catch(org.omg.CORBA.SystemException se) {
          System.out.println("Failure: " + se);
        }
    }
}
```

4.5.1 JDK での ORB の定義

```
abstract public class ORB {
    public void connect(org.omg.CORBA.Object obj);
    public org.omg.CORBA.TypeCode
             create_abstract_interface_tc(
        String id,
        String name
    );
    abstract public TypeCode create_alias_tc(
        String id,
        String name,
```

```
    TypeCode original_type
);
abstract public Any create_any( );
abstract public TypeCode create_array_tc(int length,
    TypeCode element_type);
abstract public ContextList create_context_list( );
abstract public TypeCode create_enum_tc(String id,
     String name, String[ ] members);
abstract public Environment create_environment( );
abstract public ExceptionList
                              create_exception_list( );
abstract public TypeCode create_exception_tc(
    String id,
    String name,
    StructMember[ ] members
);
public org.omg.CORBA.TypeCode create_fixed_tc(
    short digits,
    short scale);
abstract public TypeCode create_interface_tc(
    String id, String name);
abstract public NVList create_list(int count);
abstract public NamedValue create_named_value(
    String s, Any any,
    int flags);
public org.omg.CORBA.TypeCode create_native_tc(
    String id,
    String name);
public NVList create_operation_list(
                    org.omg.CORBA.Object oper);
abstract public org.omg.CORBA.portable.OutputStream
    create_output_stream( );
public org.omg.CORBA.Policy create_policy(int type,
    org.omg.CORBA.Any val)
    throws org.omg.CORBA.PolicyError;
abstract public TypeCode create_recursive_sequence_tc(
    int bound,
    int offset);
public org.omg.CORBA.TypeCode create_recursive_tc(
    String id);
abstract public TypeCode create_sequence_tc(int bound,
    TypeCode element_type);
abstract public TypeCode create_string_tc(int bound);
abstract public TypeCode create_struct_tc(
    String id,
    String name,
    StructMember[ ] members
);
abstract public TypeCode create_union_tc(
    String id,
    String name,
    TypeCode discriminator_type,
    UnionMember[ ] members
);
public org.omg.CORBA.TypeCode create_value_box_tc(
    String id,
    String name,
    TypeCode boxed_type
```

```
    );
    public org.omg.CORBA.TypeCode create_value_tc(
        String id,
        String name,
        short type_modifier,
        TypeCode concrete_base,
        ValueMember[ ] members
    );
    abstract public TypeCode create_wstring_tc(int bound);
    public void disconnect(org.omg.CORBA.Object obj);
    public org.omg.CORBA.Current get_current( );
    abstract public Context get_default_context( );
    abstract public Request get_next_response( )
                                  throws WrongTransaction;
    abstract public TypeCode
                         get_primitive_tc(TCKind tcKind);
    public boolean get_service_information(
        short service_type,
        ServiceInformationHolder service_info
    );
    public static ORB init(
                        String[ ] args, Properties props);
    public static ORB init(Applet app, Properties props);
    abstract public String[ ] list_initial_services( );
    abstract public String object_to_string(
                                org.omg.CORBA.Object obj);
    public void perform_work( );
    abstract public boolean poll_next_response( );
    abstract public org.omg.CORBA.Object
        resolve_initial_references(String object_name)
        throws InvalidName;
    public void run( );
    abstract public void send_multiple_requests_oneway(
                                          Request[ ] req);
    abstract public void send_multiple_requests_deferred(
                                          Request[ ] req);
    abstract protected void set_parameters(Applet app,
                                       Properties props);
    abstract protected void set_parameters(String[ ] args,
        Properties props);
    public void shutdown(boolean wait_for_completion);
    abstract public org.omg.CORBA.Object string_to_object(
                                             String str);
    public boolean work_pending( );
}
```

4.5.2 JDK での ORB メソッド

- public TypeCode create_abstract_interface_tc(
  String id, String name)
  IDL の抽象インタフェースの TypeCode オブジェクトを生成して返します。
  - id

抽象インタフェース型の論理 ID

- name
  抽象インタフェース型の名前

- abstract public org.omg.CORBA.TypeCode **create_alias_tc**(
  java.lang.String **repository_id**,
  java.lang.String **type_name**,
  org.omg.CORBA.TypeCode **original_type**)
  このメソッドは，IDL の alias に対応する TypeCode を生成して返します。
  - repository_id
    IDL で型を指定するリポジトリ ID
  - type_name
    アンスコープ型名
  - original_type
    エイリアスを付けられた型
- abstract public org.omg.CORBA.Any **create_any**( )
  このメソッドは，NULL タイプコードを持つ空の Any オブジェクトを生成します。
- public static org.omg.CORBA.TypeCode **create_array_tc**(
  int **bound**, TypeCode **element_type**)
  この静的メソッドは，配列に対して TypeCode を動的に生成します。
  - bound
    配列要素の最大数
  - element_type
    配列が格納している要素の型
- abstract public org.omg.CORBA.TypeCode **create_array_tc**(
  int **length**, org.omg.CORBA.TypeCode **element_type**)
  このメソッドは，IDL の配列に対応する TypeCode を生成して返します。
  - length
    配列の長さ
  - element_type
    配列に含まれる要素の型
- public abstract ContextList **create_context_list**( )
  このメソッドは，空の ContextList を生成して返します。
- public org.omg.CORBA.DynAny **create_dyn_any**(
  org.omg.CORBA.Any **value**)

指定した値で初期化した DynAny オブジェクトを作成します。

注

DynAny オブジェクトは，オペレーションリクエストおよび DII リクエストのパラメタとしては使用できません。また，ORB.object_to_string メソッドで外部化することもできません。詳細については，「5.6　DynAny」を参照してください。

- value
  該当するオブジェクトの初期化に使用する Any オブジェクト

- abstract public org.omg.CORBA.TypeCode **create_enum_tc**(
  java.lang.String **repository_id**,
  java.lang.String **type_name**,
  java.lang.String **members**[ ])
  このメソッドは，IDL の列挙体に対応する TypeCode を生成して返します。
  - repository_id
    IDL で型を指定するリポジトリ ID
  - type_name
    アンスコープ型名
  - members
    型のメンバを定義する文字列の配列
- abstract public org.omg.CORBA.Environment **create_environment**( )
  このメソッドは，空の Environment を生成して返します。
- abstract public org.omg.CORBA.TypeCode **create_exception_tc**(
  java.lang.String **repository_id**,
  java.lang.String **type_name**,
  org.omg.CORBA.StructMember **members**[ ])
  このメソッドは，IDL の exception に対応する TypeCode を生成して返します。
  - repository_id
    IDL で型を指定するリポジトリ ID
  - type_name
    アンスコープ型名
  - members
    型のメンバを定義する文字列の配列
- abstract public org.omg.CORBA.TypeCode **create_interface_tc**(
  java.lang.String **repository_id**, java.lang.String **type_name**)
  このメソッドは，IDL の interface に対応する TypeCode を生成して返します。
  - repository_id

IDL で型を指定するリポジトリ ID

- type_name
  アンスコープ型名

- abstract public org.omg.CORBA.NVList **create_list**(
  int **length**)
  このメソッドは，指定された長さの NVList を生成して返します。NVList の詳細については，「5.19 NVList」を参照してください。
  - length
    生成されるリストの長さ
- abstract public org.omg.CORBA.NamedValue **create_named_value**(
  java.lang.String **name**, org.omg.CORBA.Any **value**, int **flags**)
  このメソッドは，動的起動インタフェースの新規 NamedValue を生成して返します。
  - name
    NamedValue 名
  - value
    NamedValue の値
  - flags
    NamedValue のフラグ：IN，OUT，または INOUT
- abstract public org.omg.CORBA.NVList **create_operation_list**(
  org.omg.CORBA.OperationDef **operationDef**)
  このメソッドは，動的起動インタフェースリクエストで使用される新規 NVList を生成して返します。
  - operationDef
    指定されなければならないオペレーションの記述
- public org.omg.CORBA.Policy **create_policy**(
  int **type**,
  org.omg.CORBA.Any **val**)
  throws
  org.omg.CORBA.PolicyError
  このメソッドは，指定した初期状態を持つ指定した型のポリシーオブジェクトを作成します。ポリシーオブジェクトの作成に失敗した場合，PolicyError 例外が発生します。
  - type
    作成するポリシーオブジェクトの型
  - val
    作成するポリシーオブジェクトの初期状態を設定するための値
- abstract public org.omg.CORBA.TypeCode

create_recursive_sequence_tc(int length, int offset)
このメソッドは，IDL の sequence に対応する TypeCode を生成して返します。

- length
  生成される列の長さ。0 の場合，アンバウンデッドシーケンスであることを示します。
- offset
  タイプコード（リカーシブ）定義へのオフセット

- abstract public org.omg.CORBA.TypeCode create_sequence_tc(
  int length, org.omg.CORBA.TypeCode element_type)
  このメソッドは，IDL の sequence に対応する TypeCode を生成して返します。
  - length
    生成される列の長さ。0 の場合，アンバウンデッドシーケンスであることを示します。
  - element_type
    シーケンスに含まれる要素の型
- abstract public org.omg.CORBA.TypeCode create_string_tc(
  int length)
  このメソッドは，IDL の String に対応する TypeCode を生成して返します。
  - length
    生成される文字列の長さ。0 の場合，アンバウンデッド string であることを示します。
- abstract public org.omg.CORBA.TypeCode create_struct_tc(
  java.lang.String repository_id,
  String type_name,
  org.omg.CORBA.StructMember members[ ])
  このメソッドは，IDL の struct に対応する TypeCode を生成して返します。
  - repository_id
    IDL で型を指定するリポジトリ ID
  - type_name
    アンスコープ型名
  - members
    型のメンバを定義する構造体の配列
- abstract public org.omg.CORBA.TypeCode create_union_tc(
  java.lang.String repository_id,
  java.lang.String type_name,
  org.omg.CORBA.TypeCode discriminator_type,
  org.omg.CORBA.UnionMembers members[ ])
  このメソッドは，IDL の union に対応する TypeCode を生成して返します。

- repository_id
  IDL で型を指定するリポジトリ ID
- type_name
  アンスコープ型名
- discriminator_type
  ディスクリミネータの型。ディスクリミネータは，switch 文で使用される型です。
- members
  型のメンバを定義する構造体の配列

- public TypeCode **create_value_box_tc**(
  String **id**,
  String **name**,
  TypeCode **boxed_type**)
  IDL 値ボックスの TypeCode オブジェクトを生成して返します。
  - id
    値型の論理 ID
  - name
    値型の名前
  - boxed_type
    型の TypeCode
- abstract public org.omg.CORBA.TypeCode **create_wstring_tc**(
  int **length**)
  このメソッドは，IDL wString，または Unicode 文字列に対応する TypeCode を生成して返します。
  ルートコンテキストに対して一度もリクエストが実行されていない状態で，nameserv に対して list_all_roots メソッドを呼び出した場合，戻り値である NamingContext[ ]の要素にルートコンテキストではなく，null が入ります。
  - length
    生成される文字列の長さ。0 の場合，アンバウンデッド string であることを示します。
    負数を指定しても，BAD_PARAM 例外は発生しません。引数が負数でないことを確認する処理を入れてください。
- abstract public org.omg.CORBA.Context **get_default_context**( )
  このメソッドは，グローバルデフォルト Context を返します。デフォルトコンテキストは共有リソースであるため，デフォルトコンテキストの同期を取ってください。
- abstract public org.omg.CORBA.Request **get_next_response**( )
  このブロッキングメソッドは，遅延されたオペレーションリクエストへの応答ができるまで待ちます。完了した Request を返します。send_multiple_requests_deferred メソッドを参照してください。

- abstract public org.omg.CORBA.TypeCode get_primitive_tc(
  TCKind kind)
  このメソッドは，型に対応する基本タイプコードを返します。kind が範囲外，または基本データ型用でないと，org.omg.CORBA.BAD_PARAM 例外が発生します。
  - kind
    TCKind に定義されたタイプコードの種類
- public static ORB init(
  Strings[ ] args, Properties props)
  このメソッドは，使用する ORB をアプリケーションで初期化し，ORB の新しいインスタンスを返します。
  - args
    プログラムに渡すコマンドラインパラメタ
  - props
    ORB の動作をカスタマイズするために設定できるプロパティ
- public static ORB init(
  Applet app, Properties props)
  このメソッドは，使用する ORB をアプレットで初期化し，ORB の新しいインスタンスを返します。
  - app
    ORB の該当するインスタンスと対応づけるアプレット
  - props
    ORB の動作をカスタマイズするために設定できるプロパティ
- abstract public java.lang.String[ ] list_initial_services( )
  このメソッドは，最初にプロセスで使用できる任意のオブジェクトサービス名のリストを返します。サービスには，ロケーションサービス，インタフェースリポジトリ，またはネームサービスがあります。
- abstract public java.lang.String object_to_string(
  org.omg.CORBA.Object obj)
  このメソッドは，オブジェクトリファレンスを String に変換して返します。String はサーバが存続している間有効です。または，インプリメンテーションが活性化デーモンで登録されている場合，登録および活性化デーモンが存続している間有効です。このメソッドは，文字列化した IOR を返します。
  - obj
    変換されるオブジェクトリファレンス
- void perform_work( )
  メインスレッドから呼び出された場合，このメソッドは作業単位を一つ実行します。そうでない場合は，何も実行しません。
  メインスレッドが設定されていない場合，このオペレーションは，呼び出し元スレッドをメインスレッドとみなします。呼び出し元スレッドがメインスレッドとみなされるのは，perform_work の存続時間

の間だけです。work_pending オペレーションと perform_work オペレーションは，ORB などのアクティビティの中でメインスレッドを多重化するためのポーリングループを記述するのに使用できます。

- abstract public boolean **poll_next_response**( )

  このメソッドは，遅延されたオペレーションリクエストへの応答ができる場合に true を返します。そうでない場合，false を返します。send_multiple_requests_deferred メソッドも参照してください。

- abstract public org.omg.CORBA.Object **resolve_initial_references**(

  java.lang.String **identifier**)

  throws

  org.omg.CORBA._ORB.InvalidName

  このメソッドは，list_initial_services メソッドによって，該当するインプリメンテーションオブジェクトに返される名前の一つを分析し，得られたオブジェクトを返します。このオブジェクトは特定のサーバの型にナロウイングできます。指定された名前が見つからない場合，
  org.omg.CORBA.InvalidName 例外が発生します。

  - identifier

    識別子はサービス名で，最初のオブジェクトリファレンスを分析するために使用されます。識別子は，（Helper.bind メソッドで指定される）オブジェクト名ではありません。

    次のどれかを指定した場合に，StringIndexoutOfBoundsException 例外が発生します。

    (1)空文字列("")

    (2)"V"

    (3)"VB"

  ORB で提供されるイニシャルサービスのリストがあります。これらのサービスで，プログラムは ORB の内部機能にアクセスできるようになります。これらの機能は resolve_initial_references メソッドで使用できます。「4.4.4　VisiBroker での Object のメソッドの継承」の_resolve_reference メソッドも参照してください。

  アドオンとして ORB から提供されるイニシャルサービスには，インタフェースリポジトリ，ハンドラレジストリ，3 種類のインタセプタサービス，2 種類のアンタイプドオブジェクトラッパーサービス，および URL ネーミング（Web ネーミング）があります。次にアドオンサービスの詳細を示します。

  アドオンサービスの詳細

  ```
  ChainUntypedObjectWrapperFactory
  DistributedService
  DynAnyFactory
  InterfaceRepository
  LocationService
  NameService
  ORBPolicyManager
  PolicyCurrent
  POACurrent
  ```

RootPOA

URLNamingResolver

VBRootPOA

VisiBrokerInterceptorControl

- abstract public void **run**( )

  このオペレーションは，VisiBroker ORB に内部関数を実行するための実行リソースを提供します。メインスレッドモデルを使用している場合は，メインスレッドが処理リクエストを発行します。メインスレッドが設定されていない場合は，このオペレーションは，呼び出し元スレッドをメインスレッドとみなします。このオペレーションは，ORB が終了するまで実行を抑止します。

- abstract public void **send_multiple_requests_deferred**(
  org.omg.CORBA.Request **reqs**[ ])

  このノンブロッキングメソッドは，複数のオペレーションリクエストを送信します。リターン値は，poll_next_response，および get_next_response メソッドを使用して取得できます。

  - reqs

    オペレーションリクエスト

- abstract public void **send_multiple_requests_oneway**(
  org.omg.CORBA.Request **reqs**[ ])

  このメソッドは，複数の一方向オペレーションリクエストを送信します。リターン値は，一方向リクエストでは提供されません。

  - reqs

    活性化されるサーバのインプリメンテーションオブジェクト

- abstract public org.omg.CORBA.Object **string_to_object**(
  java.lang.String **ior**)

  このメソッドは，String をオブジェクトリファレンスに変換します。返される Object は特定のインタフェースにナロウイングされます。ior パラメタがローカルアドレススペースのインプリメンテーションオブジェクトを参照する場合，結果オブジェクトは，インプリメンテーションオブジェクトの直接ポインタリファレンスとなります。ior パラメタが不当であると，org.omg.CORBA.INV_OBJREF 例外，もしくは org.omg.CORBA.BAD_PARAM 例外が発生します。

  - ior

    object_to_string メソッドで事前に生成されたインターネットオブジェクトリファレンス

- public boolean **work_pending**( )

  このメソッドは，ORB がメインスレッドによる処理を必要とする場合は true を返し，必要としない場合は false を返します。

4.5.3　OMG による ORB の定義

```
package org.omg.CORBA_2_3;

abstract public class ORB extends org.omg.CORBA.ORB {
    public org.omg.CORBA.portable.ValueFactory
        lookup_value_factory(String id);
    public org.omg.CORBA.portable.ValueFactory
        register_value_factory(
            String id,
            org.omg.CORBA.portable.ValueFactory factory
    );
    public void set_delegate(java.lang.Object object);
    public void unregister_value_factory(String id);
}
```

4.5.4　VisiBroker での ORB の継承

```
package com.inprise.vbroker.CORBA;
public abstract class ORB extends org.omg.CORBA_2_3.ORB {
    public org.omg.CORBA.Object bind(
        String fullPoaName,
        byte[ ] oid,
        String host_name,
        BindOptions bind_options
    );
    public org.omg.CORBA.Object bind(
        String repository_id,
        String object_name,
        String host_name,
        BindOptions bind_options
    );
    public com.inprise.vbroker.CORBA.portable.InputStream
     create_input_stream(
        byte[ ] bytes
    );
    public com.inprise.vbroker.CORBA.portable.OutputStream
     create_output_stream(
        byte[ ] bytes
    );
}
```

4.5.5　ORB に対する VisiBroker の拡張機能

- public org.omg.CORBA.Object **bind**(
  java.lang.String **fullPoaName**,
  byte[ ] **oid**,
  java.lang.String **host_name**,

com.inprise.vbroker.CORBA.BindOptions **bind_options**)
このメソッドは，VisiBroker ORB オブジェクトにバインドし，オブジェクトリファレンスを取得します。

- fullPoaName
  POA 名を識別する文字列
- oid
  オブジェクト ID を識別する文字列
- host_name
  VisiBroker ORB オブジェクトを探すホスト名を識別する文字列
- bind_options
  このオブジェクトのバインドオプション

- public org.omg.CORBA.Object **bind**(
  java.lang.String **repository_id**,
  java.lang.String **object_name**,
  java.lang.String **host_name**,
  com.inprise.vbroker.CORBA.BindOptions **bind_options**)
  このメソッドは，VisiBroker ORB オブジェクトにバインドし，オブジェクトリファレンスを取得します。
  - repository_id
    リポジトリ ID を識別する文字列
  - object_name
    VisiBroker ORB オブジェクト名を識別する文字列
  - host_name
    VisiBroker ORB オブジェクトを探すホスト名を識別する文字列
  - bind_options
    このオブジェクトのバインドオプション
- abstract public org.omg.CORBA.portable.InputStream
  create_input_stream (byte[ ] **bytes**)
  このメソッドは，bytes で指定した byte 配列を含む IIOP 入力ストリームを生成します。
  - bytes
    生成する入力ストリームに含まれる byte 配列
- abstract public org.omg.CORBA.portable.OutputStream
  create_output_stream( )
  このメソッドは，IIOP 出力ストリームを生成します。IIOP バッファを構成するバイトの配列は，ストリームから抽出されます。

- public static org.omg.CORBA.ORB init( )
  このメソッドは，ORB シングルトンを返します。

4.5.6 ORB の初期化と終了

(1) ORB のインスタンス

ORB.init（String[ ],Properties）は ORB のインスタンスを新たに生成します。ORB のインスタンスごとに TPBroker 内部で使用するスレッドなどを生成し，コネクションを管理しています。これらが複数存在することで，性能の問題やメモリを圧迫するという問題が発生しますので，同時に保持する ORB インスタンス数は必要最小限にしてください。また，ORB インスタンスはアプリケーションで参照をクリアしてもガベージコレクタによってメモリ上から破棄されません。そのため，必要がなくなった ORB インスタンスに対しては，リソースとメモリの解放を行う org.omg.CORBA.ORB クラスの destroy( )メソッドを発行してください。ただし，destroy( )メソッドは使用中のコネクションも即時に解放するため，例外が発生することがあります。発行前には通信していないことを確認してください。ORB を必要とするクラスやメソッドが多く存在する場合や，Servlet のように一つのプロセス内のスレッドとしてユーザアプリケーションが動作するような場合には，引数ありの ORB.init( )の発行回数を抑止するための工夫が必要です。その方法として，一度取得した ORB のインスタンスを static 変数に保持しておき，それを使いまわすという方法が考えられます。

コーディング例：
```
static org.omg.CORBA.ORB _orb=null;
public static synchronized org.omg.CORBA.ORB getORB(String[ ] args,Properties props){
  if(_orb==null){
  _orb=org.omg.CORBA.ORB.init(args,props);
  }
  return _orb;
}
```

ORB のインスタンスが必要な場合は上記の getORB メソッドを呼び出すようにすることで，引数ありの ORB.init の発行回数を抑止できます。しかし，クラスがリロードされると static 変数（上記例の_orb）が再度初期化されることに注意してください。

Java 2 SDK，Standard Edition では，クラスをロードしたクラスローダが終了した場合にクラスがアンロードされ，リロードされたときに static 変数が初期化されます。Servlet などクラスのリロード機能を持つアプリケーションでは，ユーザが意識していないところで Servlet などが独自のクラスローダを終了させる場合があります。上記を考慮して，ORB.init の発行回数を抑止する方法として次の 2 とおりの方法が考えられます。

方法 1（推奨）：

ORB のインスタンスを static 変数で保持するクラスを，「リロード不可能なパス」に配置します。リロード機能を持つアプリケーションが「リロード可能なパス」として示す個所にクラスを配置するので

はなく，CLASSPATH など Java VM がロードできるパスにクラスを配置することで，「リロード不可能なパス」に配置することになります。

方法 2：

ORB のインスタンスを static 変数で保持するコーディングではなく，ORB のインスタンスをシステムプロパティで保持するようなコーディングにすることで，クラスのリロードが起こっても ORB インスタンスを保持することができます。

コーディング例：

```
public static synchronized org.omg.CORBA.ORB getORB(String[] args,Properties props){
  Properties sys_props=System.getProperties( );
  org.omg.CORBA.ORB _orb=(org.omg.CORBA.ORB)sys_props.get("MyORB");
  if(_orb==null)
  {
   _orb=org.omg.CORBA.ORB.init(args,props);
  }
  sys_props.put("MyORB",_orb);
  return _orb;
}
```

この方法を使用する場合は，次の点に注意してください。

- ORB.init( )を複数回発行する必要がある場合は，必ず ORB インスタンスをシステムプロパティに登録する前に ORB.init( )の発行を行ってください。
- ORB インスタンスをシステムプロパティに登録した場合，ORB インスタンスを登録したキーを引数にして java.lang.System.getProperty( )，または java.util.Properties.getProperty( )を行うと，null オブジェクトが返されます。

(2) ORB.shutdown

org.omg.CORBA.ORB.shutdown( )を発行する場合は，次の点に注意してください。

- org.omg.CORBA.ORB.shutdown( )はすべてのメソッドコールが終了したあと，発行してください。
- org.omg.CORBA.ORB.shutdown( )実施後は，org.omg.CORBA.ORB.shutdown( )した ORB インスタンスから生成したオブジェクトリファレンスを使用することができなくなるため，リファレンスを解放してください。
- org.omg.CORBA.ORB.shutdown( )した ORB インスタンスは使用しないでください。
- サーバメソッド内で org.omg.CORBA.ORB.shutdown(true)を発行すると，BAD_INV_ORDER 例外が発生する場合があります。

4.6 Policy

```
public interface org.omg.CORBA.Policy extends
    org.omg.CORBA.PolicyOperations, org.omg.CORBA.Object,
    org.omg.CORBA.portable.IDLEntity
```

Policy インタフェースは，オペレーションに影響を与える，特定の情報の利用を許可するための機構を，ORB およびオブジェクトサービスに提供します。この情報は，org.omg.CORBA.Policy インタフェースから派生するインタフェースを使用することによって，構造化された方法でアクセスできます。

4.6.1 IDL の定義

```
module CORBA {
    typedef unsigned long PolicyType;
    // Basic IDL definition
    interface Policy {
         readonly attribute PolicyType policy_type;
         Policy copy();
         void destroy();
    };
    typedef sequence <Policy> PolicyList;
    typedef sequence <PolicyType> PolicyTypeSeq;
};
```

4.6.2 Policy のメソッド

- public Policy **copy**()
  このメソッドは Policy オブジェクトのコピーを返します。
- public void **destroy**()
  このメソッドは Policy オブジェクトを削除します。
- public int **policy_type**()
  このメソッドは Policy オブジェクトの型を返します。

4.7 PortableServer.AdapterActivator

```
public interface org.omg.PortableServer.AdapterActivator extends
    org.omg.PortableServer.AdapterActivatorOperations,
    org.omg.CORBA.Object, org.omg.CORBA.portable.IDLEntity
```

アダプタアクティベータはポータブルオブジェクトアダプタ（POA）と対応し，オンデマンドで子 POA を作成する機能を POA に提供します。この機能は，子 POA を指定したリクエストを受信するとき，または活性化パラメタを True に設定して find_POA メソッドを呼び出すときに使用される機能です。

4.7.1 import 文

コード内に「`import org.omg.PortableServer.*;`」と記述してください。

4.7.2 PortableServer.AdapterActivator のメソッド

- boolean **unknown_adapter**(
  org.omg.PortableServer.POA **parent**,
  java.lang.String **name**)
  存在しない POA のオブジェクトリファレンスリクエストを受信すると，ORB がこのメソッドを呼び出します。作成する必要のある POA ごとに，ORB はこのメソッドを一度呼び出して作成します。その際，ルート POA に最も近い祖先の POA 下から作成します。
  - parent
    このメソッドを生成したアダプタアクティベータに対応する親 POA
  - name
    作成する POA の名前（親 POA に対応する POA 名）

4.8 PortableServer.Current

```
public interface org.omg.PortableServer.Current extends
    org.omg.PortableServer.CurrentOperations, org.omg.CORBA.Current,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，メソッド呼び出し先オブジェクトへアクセスするためのメソッドを提供します。このインタフェースは，複数のオブジェクトをインプリメントするサーバントをサポートするために提供されていますが，任意のサーバントに対して，POA がディスパッチしたメソッドを呼び出すコンテキストの中で使用できません。

4.8.1 import 文

コード内に「`import org.omg.PortableServer.*;`」と記述してください。

4.8.2 PortableServer.Current のメソッド

- PortableServer.ObjectId get_object_id( )

  このメソッドは，呼ばれたコンテキストのオブジェクトの ObjectId を返します。POA によってディスパッチされたメソッドのコンテキストの外部でこのメソッドが呼び出された場合は，NoContext 例外が発生します。

- PortableServer.POA get_POA( )

  このメソッドは，コンテキストの中でこのメソッドを呼び出しているオブジェクトをインプリメントしている POA のリファレンスを返します。POA によってディスパッチされたメソッドのコンテキストの外部でこのメソッドが呼び出された場合は，NoContext 例外が発生します。

4.9 PortableServer.POA

```
public interface org.omg.PortableServer.POA extends
    org.omg.PortableServer.POAOperations, org.omg.CORBA.Object,
    org.omg.CORBA.portable.IDLEntity
```

POA クラスのオブジェクトがオブジェクトインプリメンテーション群を管理します。POA はこれらのオブジェクトを ObjectId で識別するネームスペースをサポートします。一つの POA でも他 POA のネームスペースを提供します。そのネームスペースの中では，POA は既存 POA の子 POA として作成され，ルート POA から始まる階層を形成します。

POA オブジェクトを他プロセスにエクスポートしたり，文字列化したりしてはいけません。POA オブジェクトのエクスポートや文字列化を試みると MARSHAL 例外が発生します。

4.9.1 import 文

コード内に「`import org.omg.PortableServer.*;`」と記述してください。

4.9.2 PortableServer.POA のメソッド

- byte[ ] activate_object(
  org.omg.PortableServer.Servant p_servant)
  このメソッドは，バイト配列で構成されるオブジェクト ID を生成して返します。
  生成したオブジェクト ID と p_servant に指定したサーバントはアクティブオブジェクトマップに登録されます。POA に UNIQUE_ID ポリシーがあり，p_servant に指定したサーバントがすでにアクティブオブジェクトマップにある場合は，ServantAlreadyActive 例外が発生します。
  このメソッドを使用するには，POA に SYSTEM_ID ポリシーと RETAIN ポリシーが必要です。SYSTEM_ID ポリシーと RETAIN ポリシーがない場合は，WrongPolicy 例外が発生します。
  - p_servant
    アクティブオブジェクトマップに登録するサーバント
- void activate_object_with_id(
  byte[ ] id, org.omg.PortableServer.Servant p_servant)
  このメソッドは，id で指定したオブジェクトを活性化して，そのオブジェクトを p_servant で指定したサーバントにアクティブオブジェクトマップの中で対応づけます。アクティブオブジェクトマップの中ですでにそのオブジェクトにサーバントがバインドされている場合は，ObjectAlreadyActive 例外が発生します。POA に UNIQUE_ID ポリシーがあり，p_servant に指定したサーバントがすでにアクティブオブジェクトマップにある場合は，ServantAlreadyActive 例外が発生します。
  このメソッドを使用するには，POA に RETAIN ポリシーが必要です。RETAIN ポリシーがない場合は WrongPolicy 例外が発生します。

- id
  活性化するオブジェクトのオブジェクト ID
  null を指定しても，BAD_PARAM 例外は発生しません。
  注
  オブジェクト ID の長さは，512 を 2 の（サーバエンジン数 -1）乗で割ったサイズ（単位：バイト）以下にしてください。
  上記のサイズを超える値を指定した場合，osagent との通信が失敗することがあります。
- p_servant
  アクティブオブジェクトマップに登録するサーバント

- org.omg.PortableServer.ImplicitActivationPolicy
  create_implicit_activation_policy(
  org.omg.PortableServer.ImplicitActivationPolicyValue value)
  このメソッドは，指定した値の ImplicitActivationPolicy オブジェクトを返します。
  POA 生成時に ImplicitActivationPolicy を指定しなかった場合のデフォルトは NO_IMPLICIT_ACTIVATION です。
  - value
    IMPLICIT_ACTIVATION を設定すると，POA はサーバントの暗黙的活性化をサポートします。この場合，SYSTEM_ID ポリシーと RETAIN ポリシーも必要です。
    NO_IMPLICIT_ACTIVATION を設定すると，POA はサーバントの暗黙的活性化をサポートしません。
- org.omg.CORBA.Object create_reference(
  String intf)
  このメソッドは，POA が生成した ObjectId の値と intf に指定した値とをカプセル化したオブジェクトリファレンスを生成して返します。intf は生成されるオブジェクトリファレンスの type_id になります。intf に NULL 文字列を指定することもできます。このメソッドを呼び出しても活性化は起こりません。intf に指定した値が，オブジェクトの最も派生したインタフェースを識別するものでもなく，派生元インタフェースのどれかを識別するものでもない場合は，このメソッドの動作は不定です。
  このメソッドを使用するには，POA に SYSTEM_ID ポリシーが必要です。SYSTEM_ID ポリシーがない場合は WrongPolicy 例外が発生します。
  - intf
    生成するオブジェクトのクラスのリポジトリインタフェース ID
- org.omg.CORBA.Object create_reference_with_id(
  byte[ ] oid, String intf)
  このメソッドは，oid に指定した値と intf に指定した値とをカプセル化したオブジェクトリファレンスを生成して返します。intf は生成されるオブジェクトリファレンスの type_id になります。intf に NULL を指定することもできます。intf に指定した値が，オブジェクトの最も派生したインタフェースを識別するものでもなく，派生元インタフェースのどれかを識別するものでもない場合は，このメソッドの動作は不定です。このメソッドを呼び出しても活性化は起こりません。必要であれば，このメソッドが返

すオブジェクトリファレンスを複数のクライアントに渡して，そのリファレンスをクライアントからリクエストすることによってオブジェクトを活性化したり，デフォルトサーバントを使用したりできます。それは適用するポリシーに依存します。

- oid
  リファレンスを生成するオブジェクトのオブジェクト ID
- intf
  生成するオブジェクトのクラスのリポジトリインタフェース ID

- org.omg.PortableServer.IdAssignmentPolicy
  create_id_assignment_policy(
  org.omg.PortableServer.IdAssignmentPolicyValue value)
  このメソッドは，指定した値の IdAssignmentPolicy オブジェクトを返します。
  POA 生成時に IdAssignmentPolicy を指定しなかった場合のデフォルトは SYSTEM_ID です。
  - value
    USER_ID を設定すると，POA で生成したオブジェクトに，アプリケーションだけがオブジェクト ID を割り当てます。SYSTEM_ID を設定すると，POA で生成したオブジェクトに，POA だけがオブジェクト ID を割り当てます。
- org.omg.PortableServer.IdUniquenessPolicy
  create_id_uniqueness_policy(
  org.omg.Po　rtableServer.IdUniquenessPolicyValue value)
  このメソッドは，指定した値の IdUniquenessPolicy オブジェクトを返します。
  POA 生成時に IdUniquenessPolicy を指定しなかった場合のデフォルトは UNIQUE_ID です。
  - value
    UNIQUE_ID を設定すると，POA が活性化するサーバントはすべてオブジェクト ID を一つだけサポートします。MULTIPLE_ID を設定すると，POA が活性化するサーバントはそれぞれ一つ以上のオブジェクト ID をサポートできます。
- org.omg.PortableServer.LifespanPolicy create_lifespan_policy(
  org.omg.PortableServer.LifespanPolicyValue value)
  このメソッドは，指定した値の LifespanPolicy オブジェクトを返します。
  POA 生成時に LifespanPolicy を指定しなかった場合のデフォルトは TRANSIENT です。
  - value
    TRANSIENT を設定した場合，POA にインプリメントされたオブジェクトは，そのオブジェクトを最初に作成した POA インスタンスより長くは存続できません。一度トランジェント POA を非活性化したあとでその POA から生成したオブジェクトリファレンスを使用しようとすると，OBJECT_NOT_EXIST 例外が発生します。PERSISTENT を設定した場合，POA にインプリメントされたオブジェクトは，そのオブジェクトを最初に作成した POA インスタンスより長く存続できます。
- org.omg.PortableServer.POA create_POA(

String adapter_name,

org.omg.PortableServer.POAManager a_POAManager,

org.omg.CORBA.PolicyList[ ] policies)

このメソッドは，a_POAManager に指定したオブジェクトの子オブジェクトとして POA を adapter_name に指定した名前で新規作成します。新規作成した POA は，create_POA が呼び出された POA オブジェクトの子オブジェクトです。指定した親オブジェクトにすでに同名の子 POA がある場合は，PortableServer.AdapterAlreadyExists 例外が発生します。

指定したポリシーは，新規作成した POA に対応づけられ，新規作成した POA の動作を制御します。

- adapter_name
  新規作成する POA を指定する名前

注

POA 名（完全階層パス名）の長さは，512 を 2 の（サーバエンジン数 -1）乗で割ったサイズ（単位：バイト）以下にしてください。

上記のサイズを超える値を指定した場合，osagent との通信が失敗することがあります。

- a_POAManager
  新規作成する POA のマネージャ
- policies
  新規作成する POA に適用するポリシーの一覧

末尾の 2 個のパラメタには，NULL を指定できます。PortableServer.POAManager を指定しなかった場合は，自動的に作成されて新規作成した POA と対応づけられます。ポリシーを指定しなかった場合は，デフォルトのポリシーが使用されます。

ただし，次のどちらかの矛盾した設定の場合は，InvalidPolicy 例外が発生しません。

- BindSupport ポリシーに BY_POA を指定していて，ライフスパンポリシーに PERSISTENT を指定していない場合。
- BindSupport ポリシーに BY_INSTANCE を指定していて，ライフスパンポリシーに PERSISTENT が指定していない場合。

- org.omg.PortableServer.RequestProcessingPolicy
  create_request_processing_policy(
  org.omg.PortableServer.RequestProcessingPolicyValue value)
  このメソッドは，指定した値の RequestProcessingPolicy オブジェクトを返します。
  POA 生成時に RequestProcessingPolicy を指定しなかった場合のデフォルトは USE_ACTIVE_OBJECT_MAP_ONLY です。
  - value
    USE_ACTIVE_OBJECT_MAP_ONLY を設定し，オブジェクト ID がアクティブオブジェクトマップにない場合は，OBJECT_NOT_EXIST 例外をクライアントに返します。RETAIN ポリシーも必要です。

USE_DEFAULT_SERVANT を設定し，オブジェクト ID がアクティブオブジェクトマップになく，または NON_RETAIN ポリシーがあり，デフォルトサーバントが set_servant メソッドで POA に登録されている場合は，リクエストをデフォルトサーバントにディスパッチします。デフォルトサーバントが登録されていない場合は，OBJ_ADAPTER 例外をクライアントに返します。MULTIPLE_ID ポリシーも必要です。

USE_SERVANT_MANAGER を設定し，オブジェクト ID がアクティブオブジェクトマップになく，または NON_RETAIN ポリシーがあり，サーバントマネージャが set_servant_manager メソッドで POA に登録されている場合は，そのサーバントマネージャはサーバントを探し出すか，例外を発生します。サーバントマネージャが登録されていない場合は，OBJ_ADAPTER 例外をクライアントに返します。また，サーバントマネージャを使用する場合は，サーバントで Helper.narrow( )と POA.deactivate_object( )を実行しないでください。二つ以上のクライアントからそのサーバントを呼び出した場合に，サーバがデッドロック状態になることがあります。

- org.omg.PortableServer.ServantRetentionPolicy
  create_servant_retention_policy(
  org.omg.PortableServer.ServantRetentionPolicyValue value)
  このメソッドは，指定した値の ServantRetentionPolicy オブジェクトを返します。
  POA 生成時に ServantRetentionPolicy を指定しなかった場合のデフォルトは RETAIN です。
  - value
    RETAIN を設定した場合，POA は活性化したサーバントをアクティブオブジェクトマップの中に保持します。NON_RETAIN を設定した場合，POA はサーバントを保持しません。
- org.omg.PortableServer.ThreadPolicy create_thread_policy(
  org.omg.PortableServer.ThreadPolicyValue value)
  このメソッドは，指定した値の ThreadPolicy オブジェクトを返します。
  POA 生成時に ThreadPolicy を指定しなかった場合のデフォルトは ORB_CTRL_MODEL です。
  - value
    ORB_CTRL_MODEL を設定すると，ORB が制御する POA のリクエストを ORB がスレッドに割り当てます。マルチスレッド環境では，同時に複数のリクエストが複数のスレッドに割り当てられることもあります。SINGLE_THREAD_MODEL を設定すると，POA へのリクエストは一つずつ順次処理されます。マルチスレッド環境では，POA からサーバントとサーバントマネージャに出されたすべてのリクエストは，マルチスレッドに透過的なコードを保証する方法で実行されます。
- void deactivate_object(
  byte[ ] oid)
  このメソッドは，oid で指定したオブジェクトを非活性化します。非活性化したあとも，そのオブジェクトに対するアクティブなリクエストがすべてなくなるまでは，そのオブジェクトはリクエストの処理を続けます。そのオブジェクトに対して実行中のリクエストがすべて終了したときに，そのオブジェクトの ObjectId がアクティブオブジェクトマップから削除されます。
  サーバントマネージャが POA に対応づけられている場合は，ObjectId がアクティブオブジェクトマップから削除されたあとで，ServantActivator.etherealize メソッドがそのオブジェクトと，対応づけられたサーバントに対して呼び出されます。エーテライズが完了するまで，必要に応じてそのオブジェク

トの再活性化は抑止されます。しかし，このメソッドは，リクエストやエーテライズの完了を待たないで常に，指定したオブジェクトを非活性化した直後にリターンします。

このメソッドを使用するには，POA に RETAIN ポリシーが必要です。もし RETAIN ポリシーがなければ，WrongPolicy 例外が発生します。

- oid

  非活性化するオブジェクトのオブジェクト ID

- void **destroy**(

  boolean **etherealize_objects**, boolean **wait_for_completion**)

  このメソッドは，該当する POA オブジェクトとそのすべての子 POA を破棄します。最初に子 POA を破棄し，最後にカレントコンテナ POA を破棄します。必要であれば，破棄したあとで同じプロセスに同じ名前で POA を作成することもできます。

  - etherealize_objects

    このパラメタに True を設定し，POA に RETAIN ポリシーがあり，サーバントマネージャが POA に登録されている場合，アクティブオブジェクトマップ中の各アクティブオブジェクトで etherealize メソッドが呼び出されます。etherealize メソッドが呼び出される前に，見かけ上，POA の破棄が発生します。このため，POA にメソッドを呼び出す etherealize メソッドは，OBJECT_NOT_EXIST 例外を発生します。

  - wait_for_completion

    このパラメタに True を設定し，該当する POA と同じ ORB に属する他 POA からディスパッチされた呼び出しコンテキストにカレントスレッドがない場合，destroy メソッドは，アクティブなリクエストと etherealize メソッドの呼び出しがすべて完了したあとでリターンします。

    このパラメタに True を設定し，該当する POA と同じ ORB に属する他 POA からディスパッチされた呼び出しコンテキストにカレントスレッドがある場合，BAD_INV_ORDER 例外が発生し，POA の破棄は発生しません。

- org.omg.PortableServer.POA **find_POA**(

  String **adapter_name**,

  boolean **activate_it**)

  このメソッドの呼び出し先の POA オブジェクトが，指定した adapter_name の POA の親である場合は，子 POA を返します。

  - adapter_name

    該当する POA に対応する AdapterActivator の名前

  - activate_it

    このパラメタに True を設定し，adapter_name で指定した POA の子 POA がない場合は，POA の AdapterActivator（NULL 以外の場合）が呼び出され，子 POA の活性化に成功した場合は，その POA を返します。それ以外の場合は，AdapterNonExistent 例外が発生します。

- org.omg.PortableServer.Servant **get_servant**( )

  このメソッドは，POA に対応づけられたデフォルトサーバントを返します。対応づけられたサーバントがない場合は，NoServant 例外が発生します。

RETAIN ポリシーまたは USE_DEFAULT_SERVANT_MANAGER ポリシーがなければ，WrongPolicy 例外が発生します。

- org.omg.PortableServer.ServantManager get_servant_manager( )

  このメソッドは，POA に対応づけられた ServantManager オブジェクトを返します。対応づけられたサーバントマネージャがない場合は，NULL を返します。

  このメソッドを使用するには，POA に USE_SERVANT_MANAGER ポリシーが必要です。USE_SERVANT_MANAGER ポリシーがない場合は WrongPolicy 例外が発生します。

- org.omg.CORBA.Object id_to_reference(

  byte[ ] oid)

  このメソッドは，oid に指定したオブジェクトが現在アクティブであればオブジェクトリファレンスを返します。アクティブでなければ，ObjectNotActive 例外が発生します。

  このメソッドを使用するには，POA に RETAIN ポリシーが必要です。RETAIN ポリシーがない場合は WrongPolicy 例外が発生します。

  - oid

    リファレンスの取得対象オブジェクトのオブジェクト ID

- org.omg.PortableServer.Servant id_to_servant(

  byte[ ] oid)

  このメソッドの動作には次の 3 種類があります。

  - POA に RETAIN ポリシーがあり，アクティブオブジェクトマップに指定したオブジェクトがある場合は，そのオブジェクトに対応づけられたサーバントを返します。
  - POA に USE_DEFAULT_SERVANT があり，POA にデフォルトサーバントが登録されている場合は，登録されているデフォルトサーバントを返します。
  - 上記以外の場合は，ObjectNotActive 例外を出力します。

  このメソッドを使用するには，POA に RETAIN ポリシーまたは USE_DEFAULT_SERVANT ポリシーが必要です。RETAIN ポリシーまたは USE_DEFAULT_SERVANT ポリシーがなければ WrongPolicy 例外が発生します。

  パラメタの意味を次に示します。

  - oid

    サーバントの取得対象オブジェクトのオブジェクト ID

- org.omg.PortableServer.Servant reference_to_servant(

  org.omg.CORBA.Object reference)

  このメソッドの動作には次の 3 種類があります。

  - POA に RETAIN ポリシーがあり，アクティブオブジェクトマップに指定したオブジェクトがある場合は，そのオブジェクトに対応づけられたサーバントを返します。
  - POA に USE_DEFAULT_SERVANT ポリシーがあり，POA にデフォルトサーバントが登録されている場合は，登録されているデフォルトサーバントを返します。

- 上記以外の場合は，ObjectNotActive 例外を出力します。

このメソッドを使用するには，POA に RETAIN ポリシーまたは USE_DEFAULT_SERVANT ポリシーが必要です。どちらのポリシーもない場合は WrongPolicy 例外が発生します。

reference_to_servant で引数となる reference が，その POA で作られていないと WrongAdapter 例外が発生します。

パラメタの意味を次に示します。

- reference
  サーバントの取得対象オブジェクト

- byte[ ] reference_to_id(
  org.omg.CORBA.Object reference)
  このメソッドは，reference に指定したオブジェクトにカプセル化されている ObjectId の値を返します。このメソッドは，reference に指定したオブジェクトを作成した POA に対して呼び出した場合だけ有効です。このメソッドの呼び出し先 POA が，指定したオブジェクトを作成した POA と異なる場合は，WrongAdapter 例外が発生します。このメソッドを呼び出すときに，reference パラメタで指定するオブジェクトがアクティブになっている必要はありません。

  このメソッドが WrongPolicy 例外も出力できるように IDL では規定されていますが，それは将来的な拡張性を考慮してのことです。

  - reference
    ObjectId の取得対象オブジェクト

- byte[ ] servant_to_id(
  org.omg.PortableServer.Servant p_servant)
  このメソッドの動作には次の 4 種類があります。

  - POA に UNIQUE_ID ポリシーがあり，p_servant に指定したサーバントがアクティブである場合，そのサーバントに対応づけられた ObjectId を返します。
  - POA に IMPLICIT_ACTIVATION ポリシーがあり，MULTIPLE_ID ポリシーがあるか p_servant に指定したサーバントがアクティブでない場合，POA が生成した ObjectId，およびサーバントに対応づけられたリポジトリインタフェース ID を使用してサーバントを活性化して，その ObjectId を返します。
  - POA に USE_DEFAULT_SERVANT ポリシーがあり，p_servant に指定したサーバントがデフォルトサーバントである場合，現在の呼び出しに対応づけられた ObjectId を返します。
  - 上記以外の場合は，ServantNotActive 例外を出力します。

  このメソッドを使用するには，USE_DEFAULT_SERVANT ポリシーが必要です。このポリシーがない場合でも，RETAIN ポリシーと，UNIQUE_ID ポリシーまたは IMPLICIT_ACTIVATION ポリシーの組み合わせがあればこのメソッドを使用できます。この条件を満たしていない場合は，WrongPolicy 例外が発生します。

  パラメタの意味を次に示します。

  - p_servant

ObjectId の取得対象サーバント

- org.omg.CORBA.Object **servant_to_reference**(
  org.omg.PortableServer.Servant **p_servant**)
  このメソッドの動作には次の 4 種類があります。
  - POA に RETAIN ポリシーと UNIQUE_ID ポリシーがあり，p_servant に指定したサーバントがアクティブでない場合，そのサーバントを活性化するための情報をカプセル化しているオブジェクトリファレンスを返します。
  - POA に RETAIN ポリシーと IMPLICIT_ACTIVATION ポリシーがあり，さらに MULTIPLE_ID ポリシーがあるか p_servant に指定したサーバントがアクティブでない場合，POA が生成した ObjectId，およびサーバントに対応づけられたリポジトリインタフェース ID を使用してサーバントを活性化して，オブジェクトリファレンスを返します。
  - このメソッドが，p_servant に指定したサーバントへのリクエストを実行するコンテキストの中で呼び出された場合は，現在の呼び出しに対応づけられたリファレンスを返します。
  - 上記以外の場合は，ServantNotActive 例外を出力します。

  POA がディスパッチしたメソッドのコンテキストの外でこのメソッドを呼び出す場合は，RETAIN ポリシーと，UNIQUE_ID ポリシーまたは IMPLICIT_ACTIVATION ポリシーが必要です。このメソッドを p_servant に指定したサーバントに対するリクエストを実行するコンテキストの中で呼び出さないで，上記のポリシーもない場合は，WrongPolicy 例外が発生します。
  パラメタの意味を次に示します。
  - p_servant
    リファレンスの取得対象サーバント
- void **set_servant**(
  org.omg.PortableServer.Servant **p_servant**)
  このメソッドは，POA に対応するデフォルトサーバントを設定します。指定したサーバントは，アクティブオブジェクトマップの中にサーバントが登録されていないリクエストすべてに対して適用されます。
  このメソッドを使用するには，POA に USE_DEFAULT_SERVANT ポリシーが必要です。このポリシーがない場合は，WrongPolicy 例外が発生します。
  - p_servant
    デフォルトとして使用する，POA に対応するサーバント
- void **set_servant_manager**(
  org.omg.PortableServer.ServantManager **imagr**)
  このメソッドは，POA に対応するデフォルトサーバントマネージャを設定します。このメソッドは，POA が作成されているときだけ呼び出せます。POA に対応するデフォルトサーバントマネージャがすでに設定されているときにこのメソッドを呼び出すと，BAD_INV_ORDER 例外が発生します。
  このメソッドを使用するには，POA に USE_SERVANT_MANAGER ポリシーが必要です。USE_SERVANT_MANAGER ポリシーがなければ WrongPolicy 例外が発生します。

- imagr
  POA のデフォルトとして使用するサーバントマネージャ

- org.omg.PortableServer.AdapterActivator the_activator( )
  このメソッドは，POA に対応づけられたアダプタアクティベータを返します。POA の作成直後は，POA にはアダプタアクティベータがありません。つまり，属性が NULL になります。システムによっては，ルート POA の持つアクティベータの一つをアプリケーションに割り当てさせることができます。

- void the_activatorv(
  org.omg.PortableServer.AdapterActivator the_activator)
  このメソッドは，the_activator に指定したアダプタアクティベータを，POA に対応づける AdapterActivator オブジェクトとして設定します。アプリケーションは，アクティベータをルート POA に割り当てることができます。
  - the_activator
    POA に対応づけるアダプタアクティベータ

- String the_name( )
  このメソッドは，POA を識別する属性を返します。この属性は，親 POA を基準に POA を識別する，読み取り専用の属性です。この属性は，POA を作成したときに割り当てられたものです。ルート POA の名前はシステムに依存し，アプリケーションに依存するものではありません。

- org.omg.PortableServer.POA the_parent( )
  このメソッドは，該当する POA の親 POA を返します。ルート POA の親は NULL です。

- org.omg.Portableserver.POAManager the_POAManager( )
  このメソッドは，POA に対応づけられた POA マネージャを返します。

- org.omg.CORBA.Policy[ ] the_policies( )
  このメソッドは，該当する POA に有効なポリシーをすべて返します。
  Borland はこのメソッドを，POA のインプリメンテーションに追加しました。このメソッドは，将来的には OMG によって CORBA で POA に格納される可能性があります。

4.10 PortableServer.POAManager

```
public interface org.omg.PortableServer.POAManager extends
    org.omg.PortableServer.POAManagerOperations, org.omg.CORBA.Object,
    org.omg.CORBA.portable.IDLEntity
```

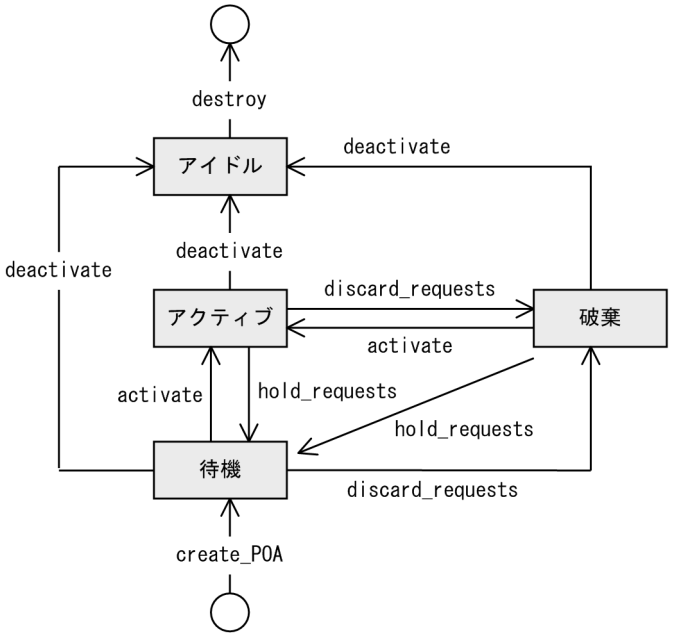
どの POA にも，POA マネージャが一つ対応づけられています。POA マネージャは一つ以上の POA オブジェクトに順番に対応づけることができます。POA マネージャは，対応づけられているすべての POA の処理状態をカプセル化しています。

POA マネージャの状態には次の 4 種類があります。

- アクティブ（ACTIVE）
- 非アクティブ（INACTIVE）
- 待機（HOLDING）
- 破棄（DISCARDING）

作成直後の POA マネージャは「待機」状態です。メソッド呼び出しによって POA マネージャの状態がどのように遷移するかを，次の図に示します。

図 4-3 Java での POA マネージャの状態遷移



4.10.1 import 文

コード内に「`import org.omg.PortableServer.*;`」と記述してください。

4.10.2 PortableServer.POAManager のメソッド

- void activate( )

  このメソッドは，POA マネージャの状態を「アクティブ」に変更します。POA マネージャが「アクティブ」状態のときは，対応づけられているすべての POA は，リクエストの処理が可能な状態となります。POA マネージャが「非アクティブ」状態のときにこのメソッドを呼び出すと，AdapterInactive 例外が発生します。

- void deactivate(

  boolean etherealize_objects,

  boolean wait_for_completion)

  このメソッドは，POA マネージャの状態を「非アクティブ」に変更します。POA マネージャが「非アクティブ」状態のときは，対応づけられているすべての POA は，新規リクエストを含む実行開始前のリクエストを拒絶します。POA マネージャが「非アクティブ」状態のときにこのメソッドを呼び出すと，AdapterInactive 例外が発生します。

- void discard_requests(

  boolean wait_for_completion)

  このメソッドは，POA マネージャの状態を「破棄」に変更します。POA マネージャが「破棄」状態のときは，対応づけられているすべての POA は，到着したリクエストを破棄します。また，キューの中で実行待ち状態にあるリクエストも破棄されます。リクエストが破棄されると，個々のリクエストを発行したクライアントにそれぞれ，TRANSIENT システム例外が返されます。POA マネージャが「非アクティブ」状態のときにこのメソッドを呼び出すと，AdapterInactive 例外が発生します。

- void hold_requests(

  boolean wait_for_completion)

  このメソッドは，POA マネージャの状態を「待機」に変更します。POA マネージャが「待機」状態のときは，対応づけられているすべての POA は，到着したリクエストをキューに保存します。このメソッドで POA マネージャを「待機」状態にする前からキューの中で実行待ち状態にあったリクエストは，継続してキューの中で実行を待ちます。POA マネージャが「非アクティブ」状態のときにこのメソッドを呼び出すと，AdapterInactive 例外が発生します。

- PortableServer.POAManager.State get_state();

  このメソッドは POAmanager の状態を返します。

4.11　PortableServer.POAManagerPackage.State

```
public class org.omg.PortableServer.POAManagerPackage.State extends
    java.lang.Object implements org.omg.CORBA.portable.IDLEntity
```

このクラスは POAManager の状態を表します。

4.11.1　import 文

コード内に「**import org.omg.PortableServer.POAManagerPackage.State;**」と記述してください。

4.11.2　PortableServer.POAManagerPackage.State のメンバ

- ACTIVE
  POAManager がアクティブであることを表します。
- INACTIVE
  POAManager が非アクティブであることを表します。
- HOLDING
  POAManager が待機中であることを表します。
- DISCARDING
  POAManager が破棄中であることを表します。

それぞれの状態遷移については，「4.10　PortableServer.POAManager」を参照してください。

4.12 PortableServer.ServantActivator

```
public interface org.omg.PortableServer.ServantActivator extends
    org.omg.PortableServer.ServantActivatorOperations,
    org.omg.PortableServer.ServantManager,
    org.omg.CORBA.portable.IDLEntity
```

POA に RETAIN ポリシーがある場合は，PortableServer.ServantActivator オブジェクトであるサーバマネージャを使用します。

4.12.1 import 文

コード内に「`import org.omg.PortableServer.*;`」と記述してください。

4.12.2 PortableServer.ServantActivator のメソッド

- void **etherealize**(
  byte[ ] **oid**,
  org.omg.PortableServer.POA **adapter**,
  org.omg.PortableServer.Servant **serv**,
  boolean **cleanup_in_progress**,
  boolean **remaining_activations**)
  このメソッドは，oid で指定したオブジェクトのサーバントを非活性化するときに，adapter に指定した POA によって呼び出されます。このメソッドは，RETAIN ポリシーと USE_SERVANT_MANAGER ポリシーがあることを前提とします。
  - oid
    非活性化するサーバントを持つオブジェクトのオブジェクト ID
  - adapter
    該当するオブジェクトが活性化していたスコープを持つ POA
  - serv
    非活性化するサーバント
  - cleanup_in_progress
    このパラメタに True を設定した場合は，etherealize_objects パラメタに True を設定した deactivate メソッドまたは destroy メソッドを呼び出したときに，このメソッドが呼び出されます。このパラメタに True を設定しない場合は，それ以外の理由でこのメソッドが呼び出されます。
  - remaining_activations

指定したサーバントが，adapter で指定した POA の他オブジェクトに対応づけられている場合は True を設定し，それ以外の場合は FALSE を設定します。

- org.omg.PortableServer.Servant incarnate(
  byte[ ] oid, org.omg.PortableServer.POA adapter)
  throws
  ForwardRequest
  このメソッドは，oid に指定した非アクティブ状態のオブジェクトへのリクエストを受け取ったときに POA によって呼び出されます。このメソッドは，RETAIN ポリシーと USE_SERVANT_MANAGER ポリシーがあることを前提とします。
  このメソッドを使用するためには，指定したオブジェクトに対応する適当なサーバントを探索および作成するサーバントマネージャのインプリメンテーションを，ユーザが提供します。このメソッドが返すサーバントは，アクティブオブジェクトマップにも登録されます。以後，アクティブなオブジェクトに対するリクエストは，サーバントマネージャを呼び出すことなく直接そのオブジェクトに対応づけられたサーバントに渡されます。
  このメソッドが，すでにほかのオブジェクトに対して活性化しているサーバントを返す場合と，POA に UNIQUE_ID ポリシーがある場合，OBJ_ADAPTER 例外が発生します。また，このメソッドでは ForwardRequest 例外を発生させることができます。ForwardRequest 例外については，「4.15 PortableServer.ForwardRequest」を参照してください。
  - oid
    活性化するサーバントを持つオブジェクトのオブジェクト ID
  - adapter
    オブジェクトを活性化するスコープを持つ POA

4.13 PortableServer.ServantLocator

```
public interface org.omg.PortableServer.ServantLocator extends
    org.omg.PortableServer.ServantLocatorOperations,
    org.omg.PortableServer.ServantManager,
    org.omg.CORBA.portable.IDLEntity
```

POA に NON_RETAIN ポリシーがある場合，POA は PortableServer.ServantLocator オブジェクトであるサーバントマネージャを使用します。POA は，このサーバントマネージャが返すサーバントは 1 件のリクエストに対してだけ使用されるということを知っているので，サーバントマネージャのメソッドに追加情報を提供できます。これによってサーバントマネージャの 1 組のメソッドは，PortableServer.ServantLocator サーバントマネージャとは異なる処理を実行できます。

4.13.1 import 文

コード内に「`import org.omg.PortableServer.*;`」と記述してください。

4.13.2 PortableServer.ServantLocator のメソッド

- org.omg.PortableServer.Servant preinvoke(
  byte[ ] oid,
  org.omg.PortableServer.POA adapter,
  String operation,
  org.omg.PortableServer.ServantLocatorPackage.CookieHolder
  the_cookie)
  throws
  ForwardRequest
  このメソッドは，POA が現在アクティブの状態にないオブジェクトへのリクエストを受け取ったときに呼び出されます。このメソッドは，NON_RETAIN ポリシーと USE_SERVANT_MANAGER ポリシーがあることを前提とします。
  可能な場合，ユーザが提供するサーバントマネージャは，oid に指定したオブジェクトに対応する適切なサーバントを探索・作成する必要があります。
  また，このメソッドでは ForwardRequest 例外を発生させることができます。ForwardRequest 例外については，「4.15 PortableServer.ForwardRequest」を参照してください。
  - oid
    到着したリクエストに対応するオブジェクト ID
  - adapter
    オブジェクトを活性化する POA

- operation
  サーバントが返されるときに POA が呼び出すオペレーションの名前
- the_cookie
  サーバントマネージャがあとで postinvoke メソッドに設定して使用できる不特定の値

- void **postinvoke**(
  byte[ ] **oid**,
  org.omg.PortableServer.POA **adapter**,
  String **operation**,
  Object **the_cookie**,
  org.omg.PortableServer.Servant **the_servant**)
  このメソッドは，サーバントがリクエストの実行を完了するときに呼び出されます。このメソッドは，POA に NON_RETAIN ポリシーと USE_SERVANT_MANAGER ポリシーがあることを前提とします。このメソッドは，オブジェクトに対するリクエストの一部とみなされます。つまり，このメソッドが正常終了したのに postinvoke メソッドがシステム例外を出力した場合は，このメソッドの正常リターンは無効となり，リクエストは例外を発生して終了します。
  POA が認識しているサーバントをデストラクトした場合，結果は不定です。
  - oid
    到着したリクエストに対応するオブジェクト ID
  - adapter
    オブジェクトを活性化する POA
  - operation
    サーバントが返されるときに POA が呼び出すオペレーションの名前
  - the_cookie
    サーバントマネージャが preinvoke メソッドに設定して使用できる不特定の値
  - the_servant
    オブジェクトに対応するサーバント

4.14 PortableServer.ServantManager

```
public interface org.omg.PortableServer.ServantManager extends
    org.omg.PortableServer.ServantManagerOperations,
    org.omg.CORBA.Object,
    org.omg.CORBA.portable.IDLEntity
```

サーバントマネージャは，POA に対応づけられています。サーバントマネージャによって POA は，POA が非アクティブ状態のオブジェクトに対するリクエストを受信したときに，オンデマンドでオブジェクトを活性化できます。

PortableServer.ServantManager インタフェースはメソッドを持たないで，ほかの二つのインタフェース（PortableServer.ServantActivator と Portableserver.ServantLocator）のベースクラスとして使用します。詳細については，「4.12 PortableServer.ServantActivator」および「4.13 PortableServer.ServantLocator」を参照してください。PortableServer.ServantActivator を使用するには，POA に RETAIN ポリシーが必要です。Portableserver.ServantLocator を使用するには，POA に NON_RETAIN ポリシーが必要です。

4.14.1 import 文

コード内に「`import org.omg.PortableServer.*;`」と記述してください。

4.15 PortableServer.ForwardRequest

```
public final class ForwardRequest extends
    org.omg.CORBA.UserException
```

この例外を使用して，別のオブジェクトにリクエストを転送するようにクライアントに指示できます。PortableServer.ServantLocator の preinvoke()メソッド，および PortableServer.ServantActivator の incarnate()メソッドで使用できます。

4.15.1 import 文

コード内に「**import org.omg.PortableServer.*;**」と記述してください。

4.15.2 ForwardRequest の変数

- public org.omg.CORBA.Object forward_reference;
  この変数は，転送するオブジェクトリファレンスを表します。

4.15.3 ForwardRequest のメソッド

- public ForwardRequest()
  このメソッドは，空のプロパティを持つ ForwardRequest オブジェクトを生成します。
- public ForwardRequest(
  org.omg.CORBA.Object ref)
  このメソッドは，指定されたプロパティを持つ ForwardRequest オブジェクトを生成します。
  - ref
    転送するオブジェクトリファレンス
- public ForwardRequest(
  java.lang.String reason, org.omg.CORBA.Object ref)
  このメソッドは，指定されたプロパティを持つ ForwardRequest オブジェクトを生成します。
  - reason
    生成する ForwardRequest オブジェクトの詳細メッセージ
  - ref
    転送するオブジェクトリファレンス

5 動的インタフェースとクラス（Java）

この章では，Java 言語の動的インタフェースとクラスについて説明します。動的インタフェースとクラスのほとんどは org.omg.CORBA パッケージに格納されています。InputStream および OutputStream は，org.omg.CORBA.portable パッケージにあります。

動的インタフェースとクラスはすべて，ランタイムのときのクライアントリクエストとオブジェクトインプリメンテーションの生成時に使用されます。

5.1 Any

```
public interface Any extends org.omg.CORBA.Any
```

このインタフェースはタイプ・セーフに Any 型の値を格納するために使用され，動的起動インタフェースで使用されます。Any に格納された型は，TypeCode で定義されます。Any は String，インタフェースオブジェクト，またはほかの Any も格納できます。含まれている値を設定，検索するためのメソッドが提供されます。VisiBroker 4.x は Fixed 型と Value 型をサポートしていませんので，これらの型を使用すると NO_IMPLEMENT 例外が発生します。

Any の生成には，org.omg.CORBA.ORB.create_any( )を使用します。詳細については，「4.5 ORB」を参照してください。

このインタフェースの Holder クラスについては，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

5.1.1 Any のメソッド

- public org.omg.CORBA.portable.InputStream **create_input_stream**( )
  このメソッドは，Any の値を含む入力ストリームを生成します。
- public org.omg.CORBA.portable.OutputStream
  create_output_stream( )
  このメソッドは，空の出力ストリームを生成します。
- public boolean **equal**(
  Any **anAny**)
  このメソッドは，Any に含まれる値がパラメタ anAny に含まれる値と同じ場合に true を返します。そうでない場合，false を返します。
  - anAny
    この Any の値と比較される値を持つ Any
- public void **read_value**(
  org.omg.CORBA.portable.InputStream **input**,
  org.omg.CORBA.Typecode **type**)
  このメソッドは，タイプコードで指定した入力ストリームからの Any の値を読み出します。Any の値だけが読み出されます。タイプコードも含めた完全な Any の定義を呼び出すには，org.omg.CORBA.portable.InputStream.read_any を使用してください。
  - input
    指定された型の値が読み出される GIOP 入力ストリーム
  - type

入力ストリームから読み出される型。このパラメタの値については，「5.24　TypeCode」を参照してください。

- public org.omg.CORBA.TypeCode **type**( )
  このメソッドは，この Any に含まれる型を表す TypeCode を返します。
- public void **type**(
  org.omg.CORBA.TypeCode **type**)
  このメソッドは，この Any に含まれる型を表す TypeCode を設定します。
  - type
    この Any オブジェクトに設定される型。このパラメタに指定できる値については，「5.24 TypeCode」を参照してください。
- public void **write_value**(
  org.omg.CORBA.portable.OutputStream **output**)
  このメソッドは，Any の値を出力ストリームに書き込みます。Any の値だけが書き込まれます。タイプコードも含めた完全な Any の定義を書き込むには，
  org.omg.CORBA.portable.OutputStream.write_any を使用してください。
  - output
    指定された Any の値が書き込まれる GIOP 出力ストリーム

5.1.2　Any の抽出メソッド

この Any に含まれる型を返すメソッドが提供されます。コードサンプル 5-1 に抽出メソッドの名前を示します。この Any に含まれる値が，使用される抽出メソッドの期待するリターンタイプと一致しない場合，StringIndexoutOfBoundsException 例外が発生します。

コードサンプル 5-1　Any クラスで提供される抽出メソッド

```
public short                extract_short( )
public int                  extract_long( )
public long                 extract_longlong( )
public short                extract_ushort( )
public int                  extract_ulong( )
public long                 extract_ulonglong( )
public float                extract_float( )
public double               extract_double( )
public boolean              extract_boolean( )
public char                 extract_char( )
public char                 extract_wchar( )
public byte                 extract_octet( )
public Any                  extract_any( )
public org.omg.CORBA.Object extract_Object( )
public java.lang.String     extract_string( )
public java.lang.String     extract_wstring( )
public org.omg.CORBA.TypeCode extract_TypeCode( )
```

5.1.3　Any の挿入メソッド

Any に特定の値の型をコピーするメソッドが提供されます。各種の型を挿入するためのメソッド一覧を，コードサンプル 5-2 に示します。一つを除いて，すべてのメソッドは，挿入する型を表すパラメタを一つ受け付けます。

最初の insert_Object メソッドは Object を挿入します。2 番目の insert_Object メソッドは，オブジェクトをさらに特定された型にナロウイングして，特定の TypeCode の Object を挿入します。2 番目のメソッドは，TypeCode の種類が TCKind.tk_objref でない場合，BAD_PARAM 例外を発生させます。

コードサンプル 5-2　Any クラスで提供される挿入メソッド

```
public void insert_short(short s)
public void insert_long(int i)
public void insert_longlong(long l)
public void insert_ushort(short s)
public void insert_ulong(int i)
public void insert_ulonglong(long l)
public void insert_float(float f)
public void insert_fixed(java.math.BigDecimal value)
public void insert_fixed(java.math.BigDecimal value,
                         org.omg.CORBA.Typecode type)
public void insert_double(double d)
public void insert_boolean(boolean b)
public void insert_char(char c)
public void insert_wchar(char c)
public void insert_octet(byte b)
public void insert_any(Any a)
public void insert_Object(org.omg.CORBA.Object o)
public void insert_Object(org.omg.CORBA.Object o,
                          org.omg.CORBA.TypeCode t)
public void insert_string(java.lang.String s)
public void insert_wstring(java.lang.String s)
void        insert_Value(java.io.Serializable v)
void        insert_Value(java.io.Serializable v,
                         org.omg.CORBA.Typecode t)
public void insert_TypeCode(org.omg.CORBA.TypeCode t)
public void insert_Streamable(
                       org.omg.CORBA.portable.Streamable s)
```

5.2 ARG_IN

```
public interface org.omg.CORBA.ARG_IN
```

このインタフェースは入力用だけに使用され，サーバで変更しない動的起動インタフェースリクエストのパラメタを指定するために使用されます。

詳細については，「5.21 Request」および「5.19 NVList」を参照してください。

5.2.1 ARG_IN の変数

- public static int value = (int) 1;

5.3 ARG_INOUT

```
public interface org.omg.CORBA.ARG_INOUT
```

このインタフェースは，入出力を目的に使用されます。また，クライアントへのリターン時にサーバによって変更される動的起動インタフェースリクエストのパラメタを指定するためにも使用されます。

詳細については，「5.21 Request」および「5.19 NVList」を参照してください。

5.3.1 ARG_INOUT の変数

- final public static int value = (int) 3;

5.4 ARG_OUT

```
public interface org.omg.CORBA.ARG_OUT
```

このインタフェースは，出力を目的に使用されます。また，クライアントへのリターン時にサーバだけが設定する動的起動インタフェースリクエストのパラメタを指定するためにも使用されます。

詳細については，「5.21 Request」および「5.19 NVList」を参照してください。

5.4.1 ARG_OUT の変数

- final public static int value = (int) 2;

5.5 ContextList

```
public interface ContextList extends org.omg.CORBA.Object
```

このインタフェースは使用されるコンテキスト文字列の変更可能なリストを保持します。

ContextList のインスタンスを生成するには，org.omg.CORBA.ORB が提供する create_context_list( ) を使用してください。詳細については，「4.5 ORB」を参照してください。

5.5.1 IDL の定義

```
interface ContextList {
   readonly attribute unsigned long count;
   void add(in string ctx);
   string item(in unsigned long index)
     raises(CORBA::Bounds);
   void remove(in unsigned long index)
     raises(CORBA::Bounds);
};
```

5.5.2 ContextList のメソッド

- public void **add**(
  String **ctx**)
  このメソッドは，文字列をコンテキストリストに追加します。
  - ctx
    コンテキストリストに追加する文字列の名前
- public int **count**( )
  このメソッドは，コンテキストリストの要素数を返します。
- public String **item**(
  int **index**)
  throws
  org.omg.CORBA.Bounds
  このメソッドは，コンテキストリスト内の項目を返します。インデックス番号が不当である場合，バウンド例外が発生します。
  - index
    項目のインデックス番号
- public void **remove**(

int index)

throws

org.omg.CORBA.Bounds

このメソッドは，コンテキストリストから項目を削除します。インデックス番号が不当である場合，バウンド例外が発生します。

- index
  項目のインデックス番号

5.6 DynAny

```
public interface DynAny extends org.omg.CORBA
```

コンパイル時にデータ型が定義されなかった場合，実行時にクライアントアプリケーションまたはサーバが DynAny オブジェクトを使用してデータ型を生成，解釈します。

DynAny は，基本型（boolean，int，float など）または複合型（struct，union など）を格納できます。DynAny に含まれる型は生成時に定義され，オブジェクトが存続している間は変更できません。

このインタフェースの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

DynAny オブジェクトは，データ型を，それぞれの値を持つ複数コンポーネントとして表すことがあります。next，rewind，および current_component メソッドを使用して，各コンポーネント間を自由に行き来できます。

次のインタフェースは DynAny から派生したもので，動的に管理される構造体をサポートします。

- Array
  「5.7 DynArray」を参照してください。
- Enumeration
  「5.9 DynEnum」を参照してください。
- Sequence
  「5.10 DynSequence」を参照してください。
- Structure
  「5.11 DynStruct」を参照してください。
- Union
  「5.12 DynUnion」を参照してください。

5.6.1 注意事項

DynAny オブジェクトは，オペレーションリクエストのパラメタまたは DII リクエストのパラメタとしては使用できません。また，ORB.object_to_string メソッドで外部化することもできません。ただし，DynAny.to_any メソッドを使用して DynAny オブジェクトを Any オブジェクトに変換すると，パラメタとして使用できるようになります。exctract_val 型と fixed 型の挿入はサポートしていません。これらのオペレーションでは BAD_OPERATION 例外が発生します。

5.6.2 DynAny のメソッド

- public void **assign**(
  org.omg.DynamicAny.DynAny **dyn_any**)
  throws
  org.omg.DynamicAny.DynAnyPackage.TypeMismatch
  このメソッドは，指定した DynAny からこのオブジェクト内のカレントコンポーネントを初期化します。Any に含まれる型がこのオブジェクトに含まれる型と一致しなかった場合，org.omg.DynamicAny.DynAnyPackage.TypeMismatch 例外が発生します。
- public org.omg.DynamicAny.DynAny **copy**( )
  このメソッドは，このオブジェクトのコピーを返します。
- public org.omg.DynamicAny.DynAny **current_component**( )
  このメソッドは，このオブジェクト内のカレントコンポーネントを返します。
- public void **destroy**( )
  このメソッドは，このオブジェクトをデストラクトします。
- public void **from_any**(
  any **value**)
  throws
  org.omg.DynamicAny.DynAnyPackage.TypeMismatch,
  org.omg.DymanicAny.DynAnyPackage.InvalidValue
  このメソッドは，指定した Any オブジェクトから，このオブジェクトのカレントコンポーネントを初期化します。
  Any に含まれる値の TypeCode が，このオブジェクトの生成時に定義した TypeCode と一致しない場合，org.omg.DynamicAny.DynAnyPackage.TypeMismatch 例外が発生します。
  - value
    このオブジェクトに設定する値を格納する Any オブジェクト
- public boolean **next**( )
  このメソッドは，コンポーネントが次にある場合は制御をそこへ移し，true を返します。コンポーネントが次にない場合は false を返します。
- public void **rewind**( )
  このメソッドは，このオブジェクトのシーケンスに含まれる先頭のコンポーネントに制御を返します。このメソッドに続いて current_component メソッドを起動すると，シーケンスの先頭のコンポーネントを返します。
  このオブジェクトにコンポーネントが一つしかなかった場合，このメソッドは何もしません。
- public boolean **seek**(
  int **index**)

このオブジェクトにコンポーネントが複数ある場合，このメソッドは指定したインデックスのコンポーネントへ制御を移し，true を返します。このメソッドに続いて current_component メソッドを起動すると，指定したインデックスのコンポーネントを返します。このオブジェクトにコンポーネントがない場合は，false を返します。

- index
  ターゲットコンポーネントのインデックス（0 から始まります）

- public org.omg.CORBA.Any to_any( )
  このメソッドは，カレントコンポーネントの値を格納する Any オブジェクトを返します。
- public org.omg.CORBA.TypeCode type( )
  このメソッドは，このオブジェクトのカレントコンポーネントが格納する値の TypeCode を返します。

5.6.3 DynAny の抽出メソッド

次に示すのは，DynAny オブジェクトのカレントコンポーネントが保持する型を返すメソッドです。コードサンプル 5-3 に抽出メソッドの名前を示します。

この DynAny に含まれる値が，使用される抽出メソッドが返す型と一致しない場合，org.omg.DynamicAny.DynAnyPackage.TypeMismatch 例外が発生します。

コードサンプル 5-3　DynAny クラスで提供される抽出メソッド

```
public org.omg.CORBA.Any get_any( )
public org.omg.DynamicAny.DynAny get_dyn_any( )
public boolean get_boolean( )
public char get_char( )
public double get_double( )
public float get_float( )
public int get_long( )
public long get_longlong( )
public byte get_octet( )
public org.omg.CORBA.Object get_reference( )
public short get_short( )
public java.lang.String get_string( )
public org.omg.CORBA.TypeCode get_typecode( )
public int get_ulong( )
public long get_ulonglong( )
public short get_ushort( )
public java.io.Serializable get_val( )
public char get_wchar( )
public java.lang.String get_wstring( )
```

5.6.4 DynAny の挿入メソッド

次に示すのは，特定の型の値をこの DynAny オブジェクトのカレントコンポーネントにコピーするメソッドです。コードサンプル 5-4 に，さまざまな型の挿入例を示します。

挿入したオブジェクトの型が DynAny オブジェクトの型と一致しなかった場合，各挿入メソッドは org.omg.DynamicAny.DynAnyPackage.TypeMismatch 例外を発生させます。

コードサンプル 5-4　DynAny クラスで提供される挿入メソッド

```
public void insert_any(org.omg.CORBA.Any value)
public void insert_dyn_any(org.omg.DynamicAny.DynAny value)
public void insert_boolean(boolean value)
public void insert_char(char value)
public void insert_double(double value)
public void insert_float(float value)
public void insert_long(int value)
public void insert_longlong(long value)
public void insert_octet(byte value)
public void insert_reference(org.omg.CORBA.Object value)
public void insert_short(short value)
public void insert_string(java.lang.String value)
public void insert_typecode(org.omg.CORBA.TypeCode value)
public void insert_ulong(int value)
public void insert_ulonglong(long value)
public void insert_ushort(short value)
public void insert_val(java.io.Serializable value)
public void insert_wchar(char value)
public void insert_wstring(java.lang.String value)
```

5.7 DynArray

```
public interface DynArray extends org.omg.DynamicAny.DynAny
```

コンパイル時にデータ型が定義されなかった場合，実行時にクライアントアプリケーションまたはサーバがこのインタフェースを使用して，配列データ型を生成，解釈します。DynArray は，一連の基本型（boolean，int，float など）または複合型（struct，union など）を格納できます。DynAny に含まれる型は生成時に定義され，オブジェクトが存続している間は変更できません。

DynamicAny から継承した next，rewind，seek，および current_component メソッドを使用して，各コンポーネント間を自由に行き来できます。

このインタフェースの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

5.7.1 注意事項

DynAny オブジェクトはオペレーションリクエストおよび DII リクエストのパラメタとしては使用できません。また，ORB.object_to_string メソッドで外部化することもできません。ただし，DynAny.to_any メソッドを使用して DynArray オブジェクトを一連の Any オブジェクトに変換すると，パラメタとして使用できるようになります。

5.7.2 DynArray のメソッド

- public org.omg.CORBA.Any[ ] get_elements( )
  このメソッドは，このオブジェクトが格納する値を含む Any オブジェクトのシーケンスを返します。
- public void set_elements(
  org.omg.CORBA.Any[ ] value)
  throws
  org.omg.DynamicAny.DynAnyPackage.TypeMismatch
  org.omg.DynamicAny.DynAnyPackage.InvalidValue
  このメソッドは，指定した Any オブジェクトのシーケンスで，このオブジェクトに含める値を設定します。
  この DynArray の要素数と value に指定した要素数が一致しない場合，
  org.omg.DynamicAny.DynAnyPackage.InvalidValue 例外が発生します。
  - value
    この DynArray に値を設定する Any オブジェクトの配列

5.8 DynAnyFactory

```
public interface DynAnyFactory extends org.omg.CORBA.Object
```

このインタフェースを使用してこのオブジェクトに対するオペレーションを呼び出すことによって，DynAny オブジェクトを任意の値から新規作成します。

5.8.1 注意事項

DynAnyFactory オブジェクトは，作成されたプロセスの中だけで有効です。したがって，DynAnyFactory オブジェクトのリファレンスをほかのプロセスにエクスポートしたり，外部化したりできません。

5.8.2 DynAnyFactory のメソッド

- public org.omg.DynamicAny.DynAny create_dyn_any(
  org.omg.CORBA.Any value)
  throws
  org.omg.DynamicAny.DynAnyFactoryPackage.InconsistentTypeCode
  このメソッドは，指定した値の DynAny オブジェクトを作成します。
  - value
    作成する DynAny オブジェクトの値
- public org.omg.DynamicAny.DynAny create_dyn_any_from_type_code(
  org.omg.CORBA.Typecode type)
  throws
  org.omg.DynamicAny.DynAnyFactoryPackage.InconsistentTypeCode
  このメソッドは，指定した型の DynAny オブジェクトを作成します。
  - type
    作成する DynAny オブジェクトの型

5.9 DynEnum

```
public interface DynEnum extends org.omg.DynamicAny.DynAny
```

コンパイル時に列挙体の値が定義されなかった場合，実行時にクライアントアプリケーションまたはサーバがこのインタフェースを使用して値を生成，解釈します。

このデータ型が含むコンポーネントは一つだけなので，DynEnum オブジェクトに対して DynAny.rewind メソッドまたは DynAny.next メソッドを起動すると，常に false を返します。

このインタフェースの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

5.9.1 注意事項

DynEnum オブジェクトはオペレーションリクエストおよび DII リクエストのパラメタとしては使用できません。また，ORB.object_to_string メソッドで外部化することもできません。ただし，DynAny.to_any メソッドを使用して DynEnum オブジェクトを Any オブジェクトに変換すると，パラメタとして使用できるようになります。

5.9.2 DynEnum のメソッド

- public java.lang.String **get_as_string**( )
  このメソッドは，DynEnum オブジェクトの値を文字列として返します。
- public void **set_as_string**(
  java.lang.String **value**)
  このメソッドは，指定した文字列で，この DynEnum に含まれる値を設定します。
  - value
    この DynEnum に値を設定するために使用する文字列
- public int **get_as_ulong**( )
  このメソッドは，DynEnum オブジェクトの値を含む int を返します。
- public void **set_as_ulong**(
  int **value**)
  このメソッドは，指定した int で，この DynEnum に含まれる値を設定します。
  - value
    この DynEnum に値を設定するために使用する整数

5.10 DynSequence

```
public interface DynSequence extends org.omg.DynamicAny.DynAny
```

コンパイル時に配列データ型が定義されなかった場合，実行時にクライアントアプリケーションまたはサーバがこのインタフェースを使用して配列データ型を生成，解釈します。DynSequence は，基本型（boolean，int，float など）または複合型（struct，union など）のシーケンスを格納できます。DynSequence に含まれる型は生成時に定義され，オブジェクトが存続している間は変更できません。

DynAny から継承した next，rewind，seek，および current_component メソッドを使用して，各コンポーネント間を自由に行き来できます。

このインタフェースの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

5.10.1 注意事項

DynSequence オブジェクトはオペレーションリクエストおよび DII リクエストのパラメタとしては使用できません。また，ORB.object_to_string メソッドで外部化することもできません。ただし，DynAny.to_any メソッドを使用して DynSequence オブジェクトを Any オブジェクトのシーケンスに変換すると，パラメタとして使用できるようになります。

5.10.2 DynSequence のメソッド

- public org.omg.CORBA.Any[ ] **get_elements**( )
  このメソッドは，このオブジェクトに格納された値を含む Any オブジェクトのシーケンスを返します。
- public int **get_length**( )
  このメソッドは，この DynSequence が格納しているコンポーネントの数を返します。
- public void **set_length**(
  int **len**)
  このメソッドは，この DynSequence が含んでいるコンポーネントの数を設定します。カレントコンポーネント数より小さい値（length）を指定すると，指定した数までのコンポーネントが格納されます。
  - len
    この DynSequence に含まれるコンポーネントの数
- public void **set_elements**(
  org.omg.CORBA.Any[ ] **value**)
  throws
  org.omg.DynamicAny.DynAnyPackage.TypeMismatch,

org.omg.org.omg.DynamicAny.DynAnyDynAnyPackage.InvalidValue

このメソッドは，指定した Any オブジェクトのシーケンスで，このオブジェクトが含める値を設定します。value に指定した要素数がこの DynSequence オブジェクトの要素数と一致しない場合，org.omg.DynamicAny.DynAnyPackage.InvalidValue 例外が発生します。

- value

  この DynArray に値が設定される Any オブジェクトの配列

5.11 DynStruct

```
public interface DynStruct extends org.omg.DynamicAny.DynAny
```

コンパイル時に構造体が定義されなかった場合，実行時にクライアントアプリケーションまたはサーバがこのインタフェースを使用して生成，解釈します。

DynAny から継承した next，rewind，seek，および current_component メソッドを使用して，各構造体メンバ間を自由に行き来できます。

DynStruct オブジェクトは，ORB.create_dyn_struct メソッドを起動することで生成されます。

このインタフェースの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

5.11.1 注意事項

DynStruct オブジェクトはオペレーションリクエストおよび DII リクエストのパラメタとしては使用できません。また，ORB.object_to_string メソッドで外部化することもできません。ただし，DynAny.to_any メソッドを使用して DynStruct オブジェクトを Any オブジェクトに変換すると，パラメタとして使用できるようになります。

5.11.2 DynStruct のメソッド

- public java.lang.String **current_member_name**( )
  throws
  org.omg.DynamicAny.DynAnyPackage.TypeMismatch,
  org.omg.org.omg.DynamicAny.DynAnyDynAnyPackage.InvalidValue
  このメソッドは，カレントコンポーネントのメンバ名を格納する文字列を返します。
- public org.omg.CORBA.TCKind **current_member_kind**( )
  throws
  org.omg.DynamicAny.DynAnyPackage.TypeMismatch,
  org.omg.org.omg.DynamicAny.DynAnyDynAnyPackage.InvalidValue
  このメソッドは，カレントコンポーネントに対応する TypeCode を返します。
- public org.omg.DynamicAny.NameValuePair[ ] **get_members**( )
  このメソッドは，この構造体のすべてのメンバを，NameValuePair オブジェクトの配列として返します。
- public void **set_members**(

org.omg.DynamicAny.NameValuePair[ ] value)
throws
org.omg.DynamicAny.DynAnyPackage.TypeMismatch,
org.omg.org.omg.DynamicAny.DynAnyDynAnyPackage.InvalidValue
このメソッドは，NameValuePair オブジェクトの配列で，構造体メンバを設定します。

- public org.omg.DynamicAny.NameDynAnyPair[ ]
  get_members_as_dyn_any( )
  このメソッドは，該当するオブジェクトが保持している値を格納している DynAny オブジェクトをすべて返します。
- public void set_members_as_dyn_any(
  org.omg.DynamicAny.NameDynAnyPair[ ] value)
  throws
  org.omg.DynamicAny.DynAnyPackage.TypeMismatch,
  org.omg.DynamicAny.DynAnyPackage.InvalidValue
  このメソッドは，DynAny オブジェクトを使用して，該当する DynStruct オブジェクトにメンバを設定します。
  value に指定した要素の順番と，該当する DynStruct オブジェクトのメンバの順番が一致しない場合，org.omg.DynamicAny.DynAnyPackage.InvalidValue 例外が発生します。

5.12 DynUnion

```
public interface DynUnion extends org.omg.DynamicAny.DynAny
```

コンパイル時に union が定義されなかった場合，実行時にクライアントアプリケーションまたはサーバがこのインタフェースを使用して，union の生成と解釈をします。DynUnion には，union の識別子と実メンバの二つの要素の列があります。

DynAny から継承した next，rewind，seek，および current_component メソッドを使用して，各コンポーネント間を自由に行き来できます。

DynUnion オブジェクトは，ORB.create_dyn_union メソッドを起動して生成します。

このインタフェースの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

5.12.1 注意事項

DynUnion オブジェクトはオペレーションリクエストおよび DII リクエストのパラメタとしては使用できません。また，ORB.object_to_string メソッドで外部化することもできません。ただし，DynAny.to_any メソッドを使用して DynUnion オブジェクトを Any オブジェクトに変換すると，パラメタとして使用できるようになります。

5.12.2 DynUnion のメソッド

- public org.omg.DynamicAny.DynAny **get_discriminator**( )
  このメソッドは，union のディスクリミネータを含む DynAny オブジェクトを返します。
- public void **set_discriminator**(
  org.omg.DynamicAny.DynAny **d**)
  throws
  org.omg.DynamicAny.DynAnyPackage.TypeMismatch
  該当する DynUnion オブジェクトの識別子を指定した値に設定します。
- public org.omg.CORBA.TCKind **discriminator_kind**( )
  このメソッドは，union のディスクリミネータのタイプコードを返します。
- public org.omg.DynamicAny.DynAny **member**( )
  throws
  org.omg.DynamicAny.DynAnyPackage.InvalidValue
  このメソッドは，union のメンバであるカレントコンポーネントの DynAny オブジェクトを返します。

- public org.omg.CORBA.TCKind member_kind( )
  throws
  org.omg.DynamicAny.DynAnyPackage.InvalidValue
  このメソッドは，union のメンバであるカレントコンポーネントのタイプコードを返します。
- public java.lang.String member_name( )
  throws
  org.omg.DynamicAny.DynAnyPackage.InvalidValue
  このメソッドは，カレントコンポーネントのメンバ名を返します。
- public void set_to_default_member( )
  throws
  org.omg.DynamicAny.DynAnyPackage.TypeMismatch
  このメソッドは，ディスクリミネータを，union のデフォルト値と一致する値に設定します。
- public boolean has_no_active_member( )
  共用体にアクティブなメンバがない場合，つまり共用体の識別子が case 文のラベルに示されない値と対応してるために，共用体が識別子だけから構成されている場合，true を返します。
- public void set_to_no_active_member( )
  throws
  org.omg.DynamicAny.DynAnyPackage.TypeMismatch
  このメソッドは，ディスクリミネータを，どの union のケースラベルにも対応しない値に設定します。

5.13 DynamicImplementation

```
public interface DynamicImplementation extends
    org.omg.CORBA.portable.ObjectImpl
```

このインタフェースは，VisiBroker ORB から任意のオブジェクトインプリメンテーションにリクエストを提供するインタフェースを提供します。オブジェクトインプリメンテーションは，コンパイル時にインプリメントするオブジェクトの型を意識しません。これは，静的なスケルトン，つまり IDL ベースのスケルトンとは異なります。このスケルトンは両方とも同じ機能を提供します。DynamicImplementation は，VisiBroker ORB が invoke メソッドでインプリメンテーションを呼び出すことで，特定のオブジェクトのすべてのリクエストをインプリメントします。

VisiBroker ORB は，ServerRequest オブジェクトを渡して，DynamicImplementation をアップコールします。ServerRequest の擬似オブジェクトは，DynamicImplementation へのリクエストの明示的状態を取得します。詳細については，「5.22 ServerRequest」を参照してください。

動的スケルトンの使用については，マニュアル「Borland Enterprise Server VisiBroker デベロッパーズガイド」の「動的スケルトンインタフェースの使用」の記述を参照してください。

5.13.1 DynamicImplementation のメソッド

- public void **invoke**(
  org.omg.CORBA.ServerRequest **request**)
  このメソッドは，サーバの機能を提供します。
  - request
    サーバが実行するリクエスト

5.14 Environment

```
public interface org.omg.CORBA.Environment
```

このインタフェースは，例外をカプセル化します。これは動的起動インタフェース（DII）と一緒に使用され，非同期 DII リクエストで発生した例外を処理するインタフェースを提供します。

Environment のインスタンスを生成するには，org.omg.CORBA.ORB で提供される create_environment メソッドを使用してください。詳細については，「4.5 ORB」を参照してください。

5.14.1 Environment のメソッド

- public void **clear**( )
  このメソッドは，カレントの Environment で発生した Exception をクリアします。これは，例外を null に設定することと同じです。
- public void **exception**(
  java.lang.Exception **exception**)
  このメソッドは，カレントの例外を設定します。設定する場合，事前に格納されていた例外は失われます。
  - exception
    カレントの Environment に設定された例外
- public java.lang.Exception **exception**( )
  このメソッドは，この Environment に設定されたカレントの Exception を返します。Exception が設定されていない場合，NULL を返します。

5.15 ExceptionList

```
public interface ExceptionList
```

このインタフェースは，動的起動インタフェース（DII）で使用され，IDL オペレーションから発生する例外について記述します。これには，タイプコードの変更可能リストを含みます。

ExceptionList のインスタンスを生成するには，org.omg.CORBA.ORB が提供する create_exception_list( )を使用してください。詳細については，「4.5 ORB」を参照してください。

5.15.1 IDL の定義

```
interface ExceptionList {
   readonly attribute unsigned long count;
   void add(in CORBA::TypeCode exc);
   CORBA::TypeCode item(in unsigned long index)
       raises(CORBA::Bounds);
   void remove(in unsigned long index)
       raises(CORBA::Bounds);
};
```

5.15.2 ExceptionList のメソッド

- public void **add**(
  TypeCode **exc**)
  このメソッドは，例外リストにタイプコードを追加します。
  - exc
    リストに追加する例外
- public int **count**( )
  このメソッドは，例外リストの項目数を返します。
- public TypeCode **item**(
  int **index**)
  throws
  org.omg.CORBA.Bounds
  このメソッドは，リストから項目を返します。インデックス番号が有効でない場合，バウンド例外が発生します。
  - index
    返される項目のインデックス番号

- public void **remove**(
  int **index**)
  throws
  org.omg.CORBA.Bounds
  このメソッドは，例外リストから項目を削除します。指定されたインデックス番号が有効でない場合，バウンド例外が発生します。
  - index
    リストから削除される項目のインデックス番号

5.16 InputStream

```
public interface InputStream
```

このインタフェースは，GIOP 入力ストリームを表します。この型のオブジェクトは，ORB.create_input_stream メソッドによって生成されます。出力ストリームに書き込まれるすべてのバイトは，入力ストリームメソッドを使用して読み出されます。各種データ型を読み出すために，幾つかのメソッドが提供されています。

create_input_stream メソッドおよび create_output_stream メソッドについては，「4.5 ORB」を参照してください。

5.16.1 InputStream のメソッド

InputStream からデータを読み出すために，次に示すメソッドが提供されています。各メソッドは，特定のデータ型を返します。

コードサンプル 5-5 InputStream からデータを読み出すためのメソッド

```
public boolean read_boolean( )
public char read_char( )
public char read_wchar( )
public byte read_octet( )
public short read_short( )
public short read_ushort( )
public int read_long( )
public int read_ulong( )
public long read_longlong( )
public long read_ulonglong( )
public float read_float( )
public double read_double( )
public String read_string( )
public String read_wstring( )
public void read_boolean_array(boolean [ ] value,
                               int offset,int length)
public void read_char_array(char [ ] value,
                               int offset,int length)
public void read_wchar_array(char [ ] value,
                               int offset,int length)
public void read_octet_array(byte [ ] value,
                               int offset,int length)
public void read_short_array(short [ ] value,
                               int offset,int length)
public void read_ushort_array(short [ ] value,
                               int offset,int length)
public void read_long_array(int [ ] value,
                               int offset,int length)
public void read_ulong_array(int [ ] value,
                               int offset,int length)
public void read_longlong_array(long [ ] value,
                               int offset,int length)
```

```
public void read_ulonglong_array(long [ ] value,
                                 int offset,int length)
public void read_float_array(float [ ] value,
                                 int offset,int length)
public void read_double_array(double [ ] value,
                                 int offset,int length)
public Object read_estruct(String expected_type)
public org.omg.CORBA.Object read_Object( )
public org.omg.CORBA.TypeCode read_TypeCode( )
public org.omg.CORBA.Any read_any( )
public org.omg.CORBA.Principal read_Principal( )
```

5.17　NamedValue

```
public interface NamedValue
```

このインタフェースは，動的起動インタフェースリクエストのパラメタと戻り値を指定するためにクライアントで使用されます。名前，値（Any），およびフラグのセットを表す整数を含みます。

NamedValue のインスタンスを生成するには，org.omg.CORBA.ORB で提供される create_named_value(String name, Any value, int flags)メソッドを使用してください。詳細については，「4.5　ORB」および「5.19　NVList」を参照してください。

5.17.1　IDL の定義

```
interface NamedValue {
    readonly attribute CORBA::Identifier name;
    readonly attribute any value;
    readonly attribute CORBA::Flags flags;
};
```

5.17.2　NamedValue のメソッド

- public int **flags**( )
  このメソッドは，この NamedValue のフラグを返します。詳細については，「5.2　ARG_IN」，「5.3　ARG_INOUT」および「5.4　ARG_OUT」を参照してください。
- public String **name**( )
  このメソッドは，この NamedValue の名前を返します。name が設定されていない場合は NULL を返します。
- public org.omg.CORBA.Any **value**( )
  このメソッドは，この NamedValue に設定されたカレントの value を表す Any を返します。値は，場合によって変更されます。

5.18 NameValuePair

```
public interface NameValuePair
```

このインタフェースは，DynStruct オブジェクトに含まれている構造体のメンバを表すために使用されます。

5.18.1 NameValuePair の変数

- public java.lang.String id
  構造体メンバ名を表します。
- public org.omg.CORBA.Any value
  構造体メンバの値と型を表します。

5.18.2 NameValuePair のコンストラクタ

- public NameValuePair( )
  空の NameValuePair を生成します。
- public NameValuePairv(
  java.lang.String id, org.omg.CORBA.Any value)
  指定のメンバ名および値で，初期化された NameValuePair を生成します。
  - id
    メンバ名
  - value
    メンバの値

5.19 NVList

```
public interface NVList
```

このインタフェースには，NamedValue オブジェクトのセットを含みます。これは，動的起動インタフェースリクエストに対応するパラメタ，およびコンテキスト値を記述するためのコンテキストルーチン内のパラメタを渡すために，クライアントアプリケーションによって使用されます。これには，NamedValues の変更可能リストを含みます。

NVList のインスタンスを生成するには，org.omg.CORBA.ORB が提供する create_list メソッドを使用してください。詳細については，「4.5 ORB」を参照してください。

5.19.1 IDL の定義

```
interface NVList {
    unsigned long count( );
    void add(in CORBA::Flags flags);
    void add_item(in CORBA::Identifier name,
                  in CORBA::Flags flags);
    void add_value(in CORBA::Identifier name,
                   in any value,
                   in CORBA::Flags flags);
    CORBA::NamedValue item(in unsigned long index);
    void remove(in unsigned long index);
};
```

5.19.2 NVList のメソッド

- public org.omg.CORBA.NamedValue **add**(
  int **flags**)
  このメソッドは，NamedValue 項目を，項目に対応する名前または値を初期化しないで，このリストに追加します。
  - flags
    追加されるパラメタのモード。許可されている値は，org.omg.CORBA.ARG_IN.value，org.omg.CORBA.ARG_OUT.value，および org.omg.CORBA.ARG_INOUT.value です。
- public org.omg.CORBA.NamedValue **add_item**(
  String **item_name**, int **flags**)
  このメソッドは，項目に対応する値を初期化しないで，このリストに NamedValue 項目を追加します。
  - item_name
    追加される項目名

- flags
  追加されるパラメタのモード。許可される値は，org.omg.CORBA.ARG_IN.value，org.omg.CORBA.ARG_OUT.value，および org.omg.CORBA.ARG_INOUT.value です。

- public org.omg.CORBA.NamedValue **add_value**(
  String **item_name**, org.omg.CORBA.Any **value**, int **flags**)
  このメソッドは，指定された名前，値，およびフラグのあるこの NVList に NamedValue を追加します。
  - item_name
    追加される NamedValue 名
  - value
    Any で表される，NamedValue の値。Any インタフェースについては，「5.1　Any」を参照してください。
  - flags
    追加されるパラメタのモード
- public int **count**( )
  このメソッドは，この NVList の NamedValue 項目数を返します。
- public org.omg.CORBA.NamedValue **item**(
  int **index**)
  throws
  org.omg.CORBA.Bounds
  このメソッドは，指定されたインデックスのあるリストから NamedValue 項目を返します。インデックスが範囲外である場合，バウンド例外が発生します。
  - index
    このリストから返される NamedValue のインデックス
- public void **remove**(
  int **index**)
  throws
  org.omg.CORBA.Bounds
  このメソッドは，このリストから，指定されたインデックスの NamedValue を削除します。インデックスが範囲外である場合，バウンド例外が発生します。
  - index
    このリストから削除される NamedValue 項目のインデックス

5.20 OutputStream

```
public interface OutputStream
```

このインタフェースは，GIOP 出力ストリームを表します。この型のオブジェクトは，ORB.create_output_stream メソッドを使用して生成されます。出力ストリームに書き込まれるすべてのバイトは，入力ストリームを使用して読み出せます。各種データ型を書き込むために幾つかのメソッドが提供されています。

create_input_stream メソッドおよび create_output_stream メソッドについては，「4.5 ORB」を参照してください。

5.20.1 OutputStream のメソッド

次に示すメソッドが，OutputStream に特定の型を書き込むために提供されています。これらの各メソッドは，書き込まれる型を表す単一パラメタを受け付けます。

コードサンプル 5-6 OutputStream に特定の型を書き込むためのメソッド

```
public inputStream create_input_stream( )
public void write_boolean(boolean value)
public void write_char(char value)
public void write_wchar(char value)
public void write_octet(byte value)
public void write_short(short value)
public void write_ushort(short value)
public void write_long(int value)
public void write_ulong(int value)
public void write_longlong(long value)
public void write_ulonglong(long value)
public void write_float(float value)
public void write_double(double value)
public void write_string(String value)
public void write_wstring(String value)
public void write_boolean_array(boolean [ ] value,
                                int offset,int length)
public void write_char_array(char [ ] value,
                                int offset,int length)
public void write_wchar_array(char [ ] value,
                                int offset,int length)
public void write_octet_array(byte [ ] value,
                                int offset,int length)
public void write_short_array(short [ ] value,
                                int offset,int length)
public void write_ushort_array(short [ ] value,
                                int offset,int length)
public void write_long_array(int [ ] value,
                                int offset,int length)
public void write_ulong_array(int [ ] value,
                                int offset,int length)
public void write_longlong_array(long [ ] value,
```

```
                              int offset,int length)
public void write_ulonglong_array(long [ ] value,
                              int offset,int length)
public void write_float_array(float [ ] value,
                              int offset,int length)
public void write_double_array(double [ ] value,
                              int offset,int length)
public org.omg.CORBA.Object write_estruct(
                                org.omg.CORBA.Object value ,
                                String expected_type)
public void write_Object(org.omg.CORBA.Object value)
public void write_TypeCode(org.omg.CORBA.TypeCode value)
public void write_any(org.omg.CORBA.Any value)
public void write_Principal(org.omg.CORBA.Principal value)
```

5.21 Request

```
public interface Request
```

このインタフェースは動的起動リクエストを表し，リクエストの初期化，送信，または応答を受信するためのメソッドを提供します。オペレーションリクエストは，同期，非同期，または応答がない oneway リクエストとして送信されます。呼び出しに対する応答は同期的にポーリングされ，取得されます。ORB インタフェースは，さらに高度な並行プロセスを可能にし，待ち時間を減少させるためにも使用されます。

このオブジェクトには次の状態情報が含まれています。

- ターゲットオブジェクト
- オペレーション名
- パラメタ型および値
- リターンタイプおよびリターン値
- Environment。詳細については，「5.14 Environment」を参照してください。
- Context。詳細については，「4.2 Context」を参照してください。

Request の生成に関連して_create_request メソッド，および_request メソッドについては，「4.4 Object」を参照してください。

5.21.1 IDL の定義

```
interface Request {
   readonly attribute CORBA::Object target;
   readonly attribute CORBA::Identifier operation;
   readonly attribute CORBA::NVList arguments;
   readonly attribute CORBA::NamedValue result;
   readonly attribute CORBA::Environment env;
   readonly attribute CORBA::ExceptionList exceptions;
   readonly attribute CORBA::ContextList contexts;

   attribute CORBA::Context ctx;

   any add_in_arg( );
   any add_named_in_arg(in string name);
   any add_inout_arg( );
   any add_named_inout_arg(in string name);

   any add_out_arg( );
   any add_named_out_arg(in string name);
   void set_return_type(in ::CORBA::TypeCode tc);
   any return_value( );

   void invoke( );
   void send_oneway( );
```

```
    void send_deferred( );
    void get_response( );
    boolean poll_response( );
};
```

5.21.2 Request のメソッド

- public Any **add_in_arg**( )
  IN パラメタをリクエストに追加します。
- public Any **add_inout_arg**( )
  INOUT パラメタをリクエストに追加します。
- public Any **add_named_in_arg**(
  String **name**)
  名前付き IN パラメタをリクエストに追加します。
  - name
    このリクエストに対応するパラメタ名
- public Any **add_named_inout_arg**(
  String **name**)
  名前付き INOUT パラメタをリクエストに追加します。
  - name
    このリクエストに対応するパラメタ名
- public Any **add_named_out_arg**(
  String **name**)
  名前付き OUT パラメタをリクエストへ追加します。
  - name
    このリクエストに対応するパラメタ名
- public Any **add_out_arg**( )
  OUT パラメタをリクエストに追加します。
- public org.omg.CORBA.NVList **arguments**( )
  このメソッドは，このリクエストのパラメタのリストを返します。これらのパラメタは，リクエスト送信前に初期化する必要があります。
- public ContextList **contexts**( )
  このメソッドは，コンテキストリストを返します。オペレーションがコンテキストを指定しないと，リストは空となります。
- public org.omg.CORBA.Context **ctx**( )

このメソッドは，このリクエストに対応するコンテキストリストを返します。詳細については，「5.5 ContextList」を参照してください。

- public void **ctx**(
  org.omg.CORBA.Context **ctx**)
  このメソッドは，このリクエストのコンテキストを設定します。
  「4.5　ORB」の get_default_context メソッドも参照してください。
  - ctx
    コンテキスト
- public org.omg.CORBA.Environment **env**( )
  このメソッドは，リクエストが呼び出される Environment を返します。サーバによって発生する例外は，リクエストの Environment に入れられます。詳細については，「5.14　Environment」を参照してください。
- public org.omg.CORBA.ExceptionList **exceptions**( )
  このメソッドは，ユーザ例外のタイプコードのリストを返します。オペレーションによるユーザ例外が発生しない場合，リストは空となります。
- public void **get_response**( )
  このブロッキングメソッドは，send_deferred メソッドによって送信された動的起動リクエストの結果を待ちます。すべての inout，out，およびリターン値はこのメソッドによって更新されます。
  ノンブロッキング poll_response メソッドは，このメソッドを呼び出す前に応答があるかどうかを判定するために使用されます。
- public void **invoke**( )
  このメソッドは，リクエストを送信したあと，処理を抑止して応答を待ちます。クライアントが応答待ちをしたくない場合は，このメソッドの代わりに send_deferred メソッドが使用できます。
- public String **operation**( )
  このメソッドは，このリクエストに対応するオペレーション名，またはメソッド名を返します。
- public boolean **poll_response**( )
  このメソッドは，リクエストに対する応答が現在ある場合に true を返します。そうでない場合，false を返します。このメソッドは，send_deferred メソッドが呼び出されたあとで，結果値を実際に読み出す get_response メソッドを呼び出す前に使用されます。
  「4.5　ORB」の poll_next_response メソッドも参照してください。
- public org.omg.CORBA.NamedValue **result**( )
  このメソッドは，リクエストの結果またはリターン値を返します。結果の型が指定されないと，型のデフォルトは void となります。リターンタイプが void でない場合，型はリクエストの送信前に初期化される必要があります。
- public Any **return_value**( )
  このメソッドは，オペレーションのリターン値を Any として取得します。

- public void **send_deferred**( )

  このメソッドは，このリクエストを送信しますが，応答を待ちません。応答を確認する場合，結果を受信するために poll_response メソッドおよび get_response メソッドが使用されます。「4.5　ORB」の send_multiple_requests_deferred メソッドも参照してください。

- public void **send_oneway**( )

  このノンブロッキングメソッドは，このリクエストを一方向 request として送信します。一方向リクエストでは，送信されるサーバからの応答はありません。

  「4.5　ORB」の send_multiple_requests_oneway メソッドも参照してください。

- public void **set_return_type**(

  TypeCode **tc**)

  このメソッドは，オペレーションを呼び出す前に返されると思われる型を設定します。

  - tc

    設定するタイプコード

- public org.omg.CORBA.Object **target**( )

  このメソッドは，このリクエストが送信されるターゲット Object を返します。ターゲット Object は，Request が生成されるときに Object の_create_request メソッドを使用して指定されます。

5.22 ServerRequest

```
public interface ServerRequest
```

動的スケルトンを使用する場合に重要である ServerRequest インタフェースは，クライアントからサーバによって受信されるリクエストを表します。これは，リクエストプロセス中に発生する Exception を反映するメソッドと同様，Context，オペレーション名，およびオペレーションパラメタを取得するメソッドを提供します。このインタフェースは，DynamicImplementation とともに動作し，動的スケルトンを提供します。DynamicImplementation の詳細については，「5.5 ContextList」を参照してください。

動的スケルトンの使用については，マニュアル「Borland Enterprise Server VisiBroker デベロッパーズガイド」の「動的スケルトンインタフェースの使用」の記述を参照してください。

5.22.1 IDL の定義

```
interface ServerRequest {
   readonly attribute CORBA::Identifier operation;
   void arguments(inout CORBA::NVList nv);
   CORBA::Context ctx();
   void set_result(in Any value);
   void set_exception(in Any value);
};
```

5.22.2 ServerRequest のメソッド

- public java.lang.String **operation**( )
  このメソッドは，このリクエストに対応するオペレーション名を返します。
- public void **arguments**(
  org.omg.CORBA.NVList **args**)
  このメソッドは，このオペレーションリクエストのパラメタを設定します。このメソッドの使用方法については，マニュアル「Borland Enterprise Server VisiBroker デベロッパーズガイド」の「動的スケルトンインタフェースの使用」の記述を参照してください。
  - args
    パラメタが格納される NVList
- public void **set_result**(
  org.omg.CORBA.Any **result**)
  このメソッドは，このオペレーションリクエストの結果を設定します。
  - result
    オペレーションリクエストに設定される結果

- public void **set_exception**(
  org.omg.CORBA.Any **except**)
  このメソッドは，request プロセス中に発生する Exception を設定するためにサーバによって使用され，クライアントに反映されます。
  - except
    発生した例外
- public org.omg.CORBA.Context **ctx**( )
  このメソッドは，この ServerRequest に対応するカレントの Context を返します。
- public String **op_name**( )
  このメソッドは，このリクエストに対応するオペレーション名を返します。
  このメソッドは，非推奨メソッドです。operation()メソッドを使用してください。
- public void **params**(
  org.omg.CORBA.NVList **params**)
  このメソッドは，このオペレーションリクエストのパラメタを設定します。このメソッドの使用方法については，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「動的スケルトンインタフェースの使用」の記述を参照してください。
  このメソッドは，非推奨メソッドです。arguments()メソッドを使用してください。
  - params
    パラメタが格納される NVList
- public void **result**(
  org.omg.CORBA.Any **result**)
  このメソッドは，このオペレーションリクエストの結果を設定します。
  このメソッドは，非推奨メソッドです。set_result()メソッドを使用してください。
  - result
    オペレーションリクエストに設定される結果
- public void **except**(
  org.omg.CORBA.Any **except**)
  このメソッドは，request プロセス中に発生する Exception を設定するためにサーバによって使用され，クライアントに反映されます。
  このメソッドは，非推奨メソッドです。set_exception()メソッドを使用してください。
  - except
    発生した例外

5.23 TCKind

```
public interface TCKind extends org.omg.CORBA.Object
```

このインタフェースには，TypeCode を定義する TypeCode オブジェクトとともに使用される定数を含みます。tk_を先頭に付けた，すべてのコードに対応できる整数定数のセットがあります。例えば，float のタイプコードは，TCKind.tk_float です。

このインタフェースの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

5.23.1 IDL の定義

```
enum TCKind {
    tk_null, tk_void,
    tk_short, tk_long, tk_ushort, tk_ulong,
    tk_float, tk_double, tk_boolean, tk_char,
    tk_octet, tk_any, tk_TypeCode, tk_Principal, tk_objref,
    tk_struct, tk_union, tk_enum, tk_string,
    tk_sequence, tk_array, tk_alias, tk_except,
    tk_longlong, tk_ulonglong, tk_longdouble,
    tk_wchar, tk_wstring, tk_fixed,
    tk_value, tk_value_box,
    tk_native,
    tk, _interface
};
```

5.23.2 TCKind のメソッド

- public int **value**( )

  このメソッドは，定数を表す整数値を返します。

- public static TCKind **from_int**(

  int **value**)

  このメソッドは，指定する値の enum インスタンスを返します。enum マッピングの詳細については，「2.8.1 enum」を参照してください。

  - value

    enum 値

5.24 TypeCode

```
public interface TypeCode
```

このインタフェースは，IDL で定義される各種型について記述し，それらがランタイム時に生成，検査できるようにします。タイプコードは，ほとんどの場合，Any オブジェクトに格納される値の型を記述するために使用されます。格納される値の型については，「5.1 Any」を参照してください。また，タイプコードは，パラメタとしてメソッド呼び出しに渡されます。

タイプコードは各種 ORB.create_<type>_tc メソッドを使用して生成されます。すべて組み込まれているタイプコードは，TCKind クラスで提供されます。詳細については，「4. コアインタフェースとクラス (Java)」を参照してください。

このインタフェースの Holder クラス，および Holder クラスのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java)」を参照してください。

5.24.1 IDL の定義

```
interface TypeCode {
    exception Bounds {
    };
    exception BadKind {
    };
    boolean equal(in CORBA::TypeCode tc);
    CORBA::TCKind kind( );
    CORBA::RepositoryId id( )
        raises(CORBA::TypeCode::BadKind);
    CORBA::Identifier name( )
        raises(CORBA::TypeCode::BadKind);
    unsigned long member_count( )
        raises(CORBA::TypeCode::BadKind);
    CORBA::Identifier member_name(in unsigned long index)
        raises(CORBA::TypeCode::BadKind,
               CORBA::TypeCode::Bounds);
    CORBA::TypeCode member_type(in unsigned long index)
        raises(CORBA::TypeCode::BadKind,
               CORBA::TypeCode::Bounds);
    any member_label(in unsigned long index)
        raises(CORBA::TypeCode::BadKind,
               CORBA::TypeCode::Bounds);
    CORBA::TypeCode discriminator_type( )
        raises(CORBA::TypeCode::BadKind);
    long default_index( )
        raises(CORBA::TypeCode::BadKind);
    unsigned long length( )
        raises(CORBA::TypeCode::BadKind);
    CORBA::TypeCode content_type( )
        raises(CORBA::TypeCode::BadKind);
    long param_count( );
    any parameter(in long index)
```

```
        raises(CORBA::TypeCode::Bounds);
};
```

5.24.2 TypeCode のメソッド

- public org.omg.CORBA.TypeCode content_type( )
  throws
  org.omg.CORBA._TypeCodePackage.BadKind
  このメソッドは，シーケンスや配列型またはエイリアス型に含まれる要素のタイプコードを返します。このメソッドは，次のタイプコードの場合有効です。
  - tk_sequence
  - tk_array
  - tk_alias

  BAD_PARAM 例外は，タイプコードが上記のどれでもない場合に発生します。
- public int default_index( )
  throws
  org.omg.CORBA._TypeCodePackage.BadKind
  このメソッドは，union のデフォルトインデックスを返します。このメソッドは，タイプコード tk_union のオブジェクトの場合だけ有効です。そうでない場合，BAD_PARAM 例外が発生します。
- public TypeCode discriminator_type( )
  throws
  org.omg.CORBA._TypeCodePackage.BadKind
  このメソッドは，union のディスクリミネータのタイプコードを返します。このメソッドは，tk_union のタイプコードでオブジェクトを呼び出す場合だけ有効です。そうでない場合は，BAD_PARAM 例外が発生します。
- public boolean equal(
  org.omg.CORBA.TypeCode tc)
  このメソッドは，このオブジェクトが tc と同等である場合に true を返します。そうでない場合，false を返します。型が同じであるかどうかは，名前ではなく型の構造体によって判定されます。同じ順序で宣言された同じフィールドの二つの構造体は，型が同じであるとみなされます。
  - tc
    このオブジェクト型と比較される TypeCode
- public String id( )
  throws
  org.omg.CORBA._TypeCodePackage.BadKind

このメソッドは，タイプコードのリポジトリ ID を返します。この文字列は IDL に使用され，型を定義します。

- public TCKind **kind**( )

  このメソッドは，このタイプコードに対応する型の種類を返します。タイプコード種類定数は，TCKind で定義されます。詳細については，「5.23　TCKind」を参照してください。

- public int **length**( )

  throws

  org.omg.CORBA._TypeCodePackage.BadKind

  このメソッドは，型が含む要素数を返します。要素数が string や sequence のようにアンバウンデッドであれば，0 を返します。このメソッドは，次のタイプコードの場合有効です。

  - tk_string
  - tk_sequence
  - tk_array

  タイプコードが上記のどれでもない場合，BAD_PARAM 例外が発生します。

- public int **member_count**( )

  throws

  org.omg.CORBA._TypeCodePackage.BadKind

  このメソッドは，型が保持しているメンバ数を返します。このメソッドは，次のタイプコードの場合有効です。

  - tk_struct
  - tk_union
  - tk_enum
  - tk_except

  タイプコードが上記のどれでもない場合，BAD_PARAM 例外が発生します。

- public Any **member_label**(

  int **index**)

  throws

  org.omg.CORBA._TypeCodePackage.BadKind,

  org.omg.CORBA._TypeCodePackage.Bounds

  このメソッドは，指定されたインデックスのあるメンバに対応するケースステートメントのラベルを返します。このメソッドは，タイプコード tk_union の場合だけ有効で，そうでない場合は，BAD_PARAM 例外が発生します。インデックスがバウンド外である場合，Bounds 例外が発生します。

  - index

    ラベルが返されるメンバのインデックス

- public String **member_name**(

int index)
throws
org.omg.CORBA._TypeCodePackage.BadKind,
org.omg.CORBA._TypeCodePackage.Bounds
このメソッドは，指定されたインデックスのあるメンバ名を返します。
このメソッドは，次のタイプコードの場合有効です。

- tk_struct
- tk_union
- tk_enum
- tk_except

タイプコードが上記のどれでもない場合，BAD_PARAM 例外が発生します。インデックスがバウンド外である場合，Bounds 例外が発生します。

- index
  名前が返されるメンバのインデックス

- public org.omg.CORBA.TypeCode member_type(
  int index)
  throws
  org.omg.CORBA._TypeCodePackage.BadKind,
  org.omg.CORBA._TypeCodePackage.Bounds
  このメソッドは，指定されたインデックスのあるメンバのタイプコードを返します。このメソッドは，次のタイプコードの場合有効です。
  - tk_struct
  - tk_union
  - tk_except

  タイプコードが上記のどれでもない場合，BAD_PARAM 例外が発生します。インデックスがバウンド外である場合，Bounds 例外が発生します。
  - index
    タイプコードが返されるメンバのインデックス
- public String name( )
  throws
  org.omg.CORBA._TypeCodePackage.BadKind
  このメソッドは，アンスコープ型名を返します。このメソッドは，次のタイプコードの場合有効です。
  - tk_objref
  - tk_struct
  - tk_union

- tk_enum
- tk_alias
- tk_except

タイプコードが上記のどれでもない場合，BAD_PARAM 例外が発生します。

5.25 UnknownUserException

```
public interface UnknownUserException extends
    org.omg.CORBA.UserException
```

クライアントが DII リクエストを発行してユーザ例外が発生すると，特定の例外がクライアントに反映されなくなり，この例外が代わりに使用されます。

6 インタフェースリポジトリインタフェースとクラス (Java)

この章では，Java 言語でインタフェースリポジトリと一緒に使用する org.omg.CORBA パッケージでのインタフェースとクラスについて説明します。

6.1 AbstractInterfaceDef

```
public interface org.omg.CORBA.AbstractInterfaceDef extends
    org.omg.CORBA.AbstractInterfaceDefOperations,
    org.omg.CORBA.InterfaceDef,
    org.omg.CORBA.portable.IDLEntity
```

AbstractInterfaceDef は，InterfaceDef インタフェースに類似したインタフェースです。InterfaceDef との違いは，abstract 型インタフェースが継承できるのは abstract 型インタフェースからだけで，concrete 型インタフェースをベースとして使用できないという点です。AbstractInterfaceDef インタフェースは，インタフェースリポジトリに格納されている IDL の abstract 型インタフェースを表すために使用します。このインタフェースは属性，オペレーション，インタフェース定義を作成するメソッド，およびベースインタフェースの設定と取得を実行するメソッドを提供します。

InterfaceDef インタフェースの詳細については，「6.20　InterfaceDef」を参照してください。

このインタフェースの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3.　生成されるインタフェースとクラス（Java）」を参照してください。

6.1.1 AbstractInterfaceDef のメソッド

- public org.omg.CORBA.AbstractInterfaceDef[ ]base_interfaces( )
  このメソッドは，このオブジェクトのベースインタフェースの空のリストを返します。
- public void base_interfaces(
  org.omg.CORBA.InterfaceDef[ ]base_interfaces)
  このメソッドは，このオブジェクトのベースインタフェースリストを設定します。
  - base_interfaces
    設定するベースインタフェースリスト
- public org.omg.CORBA.InterfaceDef create_abstract_interface(
  java.lang.String id,
  java.lang.String name,
  java.lang.String version,
  org.omg.CORBA.AbstractInterfaceDef[ ] base_interfaces)
  このメソッドは，指定した属性で AbstractInterfaceDef オブジェクトをこの Container の中に生成し，新しく生成したオブジェクトのリファレンスを返します。concrete 型の InterfaceDef とは異なり，AbstractInterfaceDef インタフェースは abstract 型と concrete 型の両方のインタフェースの定義を格納できません。継承できるのは，abstract 型インタフェースだけです。
  - id
    インタフェースのリポジトリ ID

- name
  インタフェースの名前
- version
  インタフェースのバージョン
- base_interfaces
  このインタフェースの継承元となるすべてのインタフェースの一覧

- public org.omg.CORBA.AttributeDef **create_attribute**(
  java.lang.String **id**,
  java.lang.String **name**,
  java.lang.String **version**,
  org.omg.CORBA.IDLType **type**,
  org.omg.CORBA.AttributeMode **mode**)
  このメソッドは，属性をインタフェース定義に追加します。
  - id
    属性の識別子
  - name
    属性の名前
  - version
    属性のバージョン
  - type
    属性の IDL 型
  - mode
    属性のモード。AttributeMode に指定できる値の詳細については，「6.6　AttributeMode」を参照してください。
- public org.omg.CORBA.OperationDef **create_operation**(
  java.lang.String **id**,
  java.lang.String **name**,
  java.lang.String **version**,
  org.omg.CORBA.IDLType **result**,
  org.omg.CORBA.OperationMode **mode**,
  org.omg.CORBA.ParameterDescription[ ] **params**,
  org.omg.CORBA.ExceptionDef[ ] **exceptions**,
  java.lang.String[ ] **contexts**)
  このメソッドは，オペレーションをインタフェース定義に追加します。
  - id

オペレーションの識別子

- name
  オペレーションの名前
- version
  オペレーションのバージョン
- result
  オペレーションの IDL 結果型
- mode
  オペレーションのモード。詳細については，「6.30　OperationMode」を参照してください。
- params
  このオペレーションのパラメタの一覧
- exceptions
  このオペレーションで発生する可能性のある例外の一覧
- contexts
  コンテキストの一覧

- public org.omg.CORBA.InterfaceDefPackage.
  FullInterfaceDescription **describe_interface**( )
  このメソッドは，このオブジェクトのインタフェース定義を返します。
- public boolean **is_a**(
  java.lang.String **interface_id**)
  このオブジェクトが，interface_id に指定したインタフェース識別子と互換性のあるインタフェース定義を表す場合，このメソッドは true を返します。
  - interface_id
    このオブジェクトとの比較に使用するインタフェース識別子
- public boolean **is_abstract**( )
  true が設定された場合，このメソッドは，指定したインタフェースが abstract であることを示します。
  - is_abstract
    abstract インタフェースの作成を指定します。
- public void **is_abstract**(
  boolean **is_abstract**)
  このメソッドは，このインタフェースを abstract に設定します。
  - is_abstract
    オブジェクトを abstract に設定します。

6.2 AliasDef

```
public interface org.omg.CORBA.AliasDef extends
    org.omg.CORBA.AliasDefOperations,
    org.omg.CORBA.TypedefDef,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，インタフェースリポジトリに格納されている typedef を表すために使用します。このインタフェースは，オリジナルの typedef の IDLType を設定および取得するためのメソッドを提供します。

TypedefDef の詳細については，「6.19　IDLType」を参照してください。

このインタフェースの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3.　生成されるインタフェースとクラス（Java）」を参照してください。

6.2.1 AliasDef のメソッド

- public void **original_type_def**(
  org.omg.CORBA.IDLType **original_type_def**)
  このメソッドは，このオブジェクトの IDLType を設定します。
  - original_type_def
    このオブジェクトの IDLType
- public org.omg.CORBA.IDLType **element_type_def**( )
  このメソッドは，このオブジェクトがエイリアスとなっているオブジェクトのオリジナル typedef の IDLType を返します。

6.3 ArrayDef

```
public interface org.omg.CORBA.ArrayDef extends
    org.omg.CORBA.ArrayDefOperations,
    org.omg.CORBA.IDLType,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，インタフェースリポジトリに格納されている配列を表すために使用します。このインタフェースは，配列内の要素型と配列の長さを設定および取得するためのメソッドを提供します。

このインタフェースの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.3.1 ArrayDef のメソッド

- public int **length**( )
  このメソッドは，配列内の要素数を返します。
- public void **length**(
  int **length**)
  このメソッドは，配列内の要素数を設定します。
  - length
    array 内の要素数
- public org.omg.CORBA.TypeCode **element_type**( )
  このメソッドは，配列の要素の TypeCode を返します。
- public void **element_type_def**(
  org.omg.CORBA.IDLType **element_type_def**)
  このメソッドは，配列内に格納されている要素の IDLType を設定します。
  - element_type_def
    配列内の要素の IDLType
- public org.omg.CORBA.IDLType **element_type_def**( )
  このメソッドは，配列内に格納されている要素の IDLType を返します。

6.4 AttributeDef

```
public interface org.omg.CORBA.AttributeDef extends
    org.omg.CORBA.AttributeDefOperations,
    org.omg.CORBA.Contained,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，インタフェースリポジトリに格納されているインタフェース属性を表すために使用します。このインタフェースは，属性のモードと型を設定および取得するためのメソッドを提供します。

このインタフェースの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.4.1 AttributeDef のメソッド

- public org.omg.CORBA TypeCode **type**( )
  このメソッドは，属性の型を表す TypeCode を返します。
- public void **type_def**(
  org.omg.CORBA.IDLType **type_def**)
  このメソッドは，属性の IDLType を設定します。
  - type_def
    このオブジェクトの IDLType
- public org.omg.CORBA.IDLType **type_def**( )
  このメソッドは，属性の IDLType を返します。
- public org.omg.CORBA.AttributeMode **mode**( )
  このメソッドは，属性のモードを返します。読み取り専用属性を示す AttributeMode ATTR_READONLY と，読み書き可能属性を示す AttributeMode ATTR_NORMAL のどちらかを返します。詳細については，「6.6 AttributeMode」を参照してください。
- org.omg.CORBA.AttributeDef **mode**( )
  このメソッドは，該当するモード属性の値を返します。
- public void **mode**(
  org.omg.CORBA.AttributeMode mode)
  このメソッドは，モード属性の値を設定します。

6.5 AttributeDescription

```
public final class org.omg.CORBA.AttributeDescription extends
    java.lang.Object implements org.omg.CORBA.portable.IDLEntity
```

このクラスは，インタフェースリポジトリに格納されている属性を記述します。また，AttributeDescription の struct は，インタフェースと値を完全に表現するために使用します。これらの型だけが，属性を保持できる IDL 型です。

このクラスには，Helper クラスと Holder クラスもあります。これらのクラスとそのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.5.1 AttributeDescription の変数

- public java.lang.String **name**
  この変数は，属性の名前を表します。
- public java.name.String **id**
  この変数は，属性のリポジトリ ID を表します。
- public java.lang.String **defined_in**
  この変数は，属性が定義されているインタフェースのリポジトリ ID または valuetype を表します。
- public java.lang.String **version**
  この変数は，属性のバージョンを表します。
- public org.omg.CORBA.TypeCode **type**
  この変数は，属性の TypeCode を表します。
- public org.omg.CORBA.AttributeMode **mode**
  この変数は，属性のモードを表します。

6.5.2 AttributeDescription のメソッド

- public **AttributeDescription**( )
  このメソッドは，AttributeDescription のデフォルトコンストラクタです。
- public **AttributeDescription**(
  java.lang.String **name**,
  java.lang.String **id**,
  java.lang.String **defined_in**,
  java.lang.String **version**,
  org.omg.CORBA.TypeCode **type**,

org.omg.CORBA.AttributeMode mode)

このメソッドは，指定されたパラメタを使用して，AttributeDescription を構成します。

- name
  この属性の名前
- id
  この属性のリポジトリ ID
- defined_in
  この属性が定義されたインタフェースまたは valuetype
- version
  オブジェクトのバージョン
- type
  属性の IDL タイプコード
- mode
  この属性のモード（「読み取り専用」または「読み書き可能」）。詳細については，「6.6 AttributeMode」を参照してください。

6.6 AttributeMode

```
public final class org.omg.CORBA.AttributeMode extends
    java.lang.Object implements org.omg.CORBA.portable.IDLEntity
```

このクラス（IDL の列挙体）は，属性のモードが読み取り専用か標準（読み書き可能）かを表すために使用します。モードとして次のどちらかのパラメタを指定できます。

- NORMAL
  このモードでは，この属性は読み書き可能となります。
- READONLY
  このモードでは，この属性は読み取り専用となります。

このクラスには，Helper クラスと Holder クラスもあります。これらのクラスとそのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.6.1 AttributeMode の要素

- org.omg.CORBA.AttributeMode.ATTR_NORMAL
  この変数は，標準モードの属性を定義します。
- org.omg.CORBA.AttributeMode.ATTR_READONLY
  この変数は，読み取り専用モードの属性を定義します。

6.7 ConstantDef

```
public interface org.omg.CORBA.ConstantDef extends
    org.omg.CORBA.ConstantDefOperations,
    org.omg.CORBA.Contained,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，インタフェースリポジトリに格納されている定数定義を表すために使用します。このインタフェースは，定数の型と値を設定，取得するためのメソッドを提供します。

このインタフェースの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.7.1 ConstantDef のメソッド

- public org.omg.CORBA.TypeCode **type**( )
  このメソッドは，オブジェクトの型を表す TypeCode を返します。
- public org.omg.CORBA.IDLType **type_def**( )
  このメソッドは，定数の IDLType を返します。
- public void **type_def**(
  org.omg.CORBA.IDLType **type_def**)
  このメソッドは，定数の IDLType を設定します。
  - type_def
    この定数の IDLType
- public org.omg.CORBA.Any **value**( )
  このメソッドは，定数の値を表す Any オブジェクトを返します。
- public void **value**(
  org.omg.CORBA.Any **value**)
  このメソッドは，この定数の値を設定します。
  - value
    このオブジェクトの値を表す Any オブジェクト

6.8 ConstantDescription

```
public final class org.omg.CORBA.ConstantDescription extends
    java.lang.Object implements org.omg.CORBA.portable.IDLEntity
```

このクラスは，インタフェースリポジトリに格納されている定数を記述します。

6.8.1 ConstantDescription の変数

- public java.lang.String **name**
  この変数は，定数名を表します。
- public java.lang.String **id**
  この変数は，定数のリポジトリ ID を表します。
- public java.lang.String **defined_in**
  この変数は，この定数が定義されたモジュールまたはインタフェースのリポジトリ ID を表します。
- public org.omg.CORBA.VersionSpec **version**
  この変数は，定数のバージョンを表します。
- public org.omg.CORBA.TypeCode **type**
  この変数は，定数の TypeCode を表します。
- public org.omg.CORBA.Any **value**
  この変数は，この定数の値を表します。

6.8.2 ConstantDescription のメソッド

- public **ConstantDescription**( )
  このメソッドは，ConstantDescription のデフォルトコンストラクタです。
- public **ConstantDescription**(
  java.lang.String **name**,
  java.lang.String **id**,
  java.lang.String **defined_in**,
  org.omg.CORBA.VersionSpec **version**,
  org.omg.CORBA.TypeCode **type**,
  org.omg.CORBA.Any **value**)
  このメソッドは，指定されたパラメタを使用して，ConstantDescription を構成します。
  - name

この定数の名前

- id
  この定数のリポジトリ ID
- defined_in
  この定数が定義されたモジュールまたはインタフェース
- version
  オブジェクトのバージョン
- type
  定数の IDL タイプコード
- value
  この定数の値

6.9 Contained

```
public interface org.omg.CORBA.Contained extends
    org.omg.CORBA.ContainedOperations,
    org.omg.CORBA.IRObject,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，ほかのインタフェースリポジトリオブジェクトに含まれたインタフェースリポジトリオブジェクトを表すために使用します。このインタフェースは，次に示す動作をするためのメソッドを提供します。

- オブジェクト名とバージョンの設定と取得
- このオブジェクトを含む Container の決定
- オブジェクトの絶対名，オブジェクトのリポジトリ，記述の取得
- 一つのコンテナからほかのコンテナへのオブジェクトの移動

このインタフェースの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.9.1 IDL の定義

```
interface Contained:IRObject {
    attribute RepositoryId id;
    attribute Identifier name;
    attribute VersionSpec version;

    readonly attribute Container defined_in;
    readonly attribute ScopedName absolute_name;
    readonly attribute Repository containing_repository;

    struct Description {
        DefinitionKind kind;
        any value;
    };

    Description describe( );

    void move(in Container new_container,
        in Identifier new_name,
        in VersionSpec new_version);
};
```

6.9.2 Contained のメソッド

- public java.lang.String **absolute_name**( )

このメソッドは，オブジェクトの絶対名を返します。

- public org.omg.CORBA.Repository **containing_repository**( )
  このメソッドは，このオブジェクトを含んでいるリポジトリを返します。
- public org.omg.CORBA.Container **defined_in**( )
  このメソッドは，このオブジェクトが定義された Container を返します。
- public org.omg.CORBA.ContainedPackage.Description **describe**( )
  このメソッドは，このオブジェクトの記述を返します。
  Description の詳細については，「6.12　ContainerPackage.Description」を参照してください。
- public java.lang.String **id**( )
  このメソッドは，このオブジェクトのリポジトリ ID を返します。
- public void **id**(
  String **id**)
  このメソッドは，このオブジェクトを一意に識別するリポジトリ ID を設定します。
  - id
    このオブジェクトのリポジトリ ID
- public java.lang.String **name**( )
  このメソッドは，このオブジェクトの名前を返します。
- public void **name**(
  java.lang.String **name**)
  このメソッドは，このオブジェクトの名前を設定します。
  - name
    オブジェクト名
- public java.lang.String **version**( )
  このメソッドは，このオブジェクトのバージョンを返します。
- public void **version**(
  java.lang.String **version**)
  このメソッドは，このオブジェクトのバージョンを設定します。
  - version
    オブジェクトのバージョン
- public void **move**(
  org.omg.CORBA.Container **new_container**,
  String **new_name**,
  java.lang.String **new_version**)
  このメソッドは，このオブジェクトをほかのコンテナに移動させます。

- new_container
  オブジェクトの移動先の Container
- new_name
  オブジェクトの新しい名前
- new_version
  オブジェクトの新しいバージョン設定

6.10 ContainedPackage.Description

```
public final class org.omg.CORBA.ContainedPackage.Description extends
    java.lang.Object implements org.omg.CORBA.portable.IDLEntity
```

このクラスは，Contained インタフェースから派生したインタフェースリポジトリ内の各項目の一般的な記述を提供します。

このクラスには，Helper クラスと Holder クラスもあります。これらのクラスとそのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.10.1 ContainedPackage.Description の変数

- public org.omg.CORBA.DefinitionKind **kind**
  この変数は，項目の種類を表します。
- public org.omg.CORBA.Any **value**
  この変数は，項目の値を表します。

6.10.2 ContainedPackage.Description のメソッド

- public **Description**( )
  このメソッドは，Description のデフォルトコンストラクタです。
- public **Description**(
  org.omg.CORBA.DefinitionKind **kind**,
  org.omg.CORBA.Any **value**)
  このメソッドは，指定されたパラメタを使用して，Description を構成します。
  - kind
    この項目の種類。詳細については，「6.12 ContainerPackage.Description」を参照してください。
  - value
    この項目の値を表す Any オブジェクト

6.11 Container

```
public interface org.omg.CORBA.Container extends
    org.omg.CORBA.ContainerOperations,
    org.omg.CORBA.IRObject,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，インタフェースリポジトリ内に包含階層を生成するために使用します。Container オブジェクトは，Contained クラスから派生したオブジェクト定義を保持します。Container インタフェースから派生したすべてのオブジェクト定義も，Repository クラスの例外とともに，Contained クラスから継承します。

このインタフェースの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.11.1 IDL の定義

```
interface Container:IRObject {
    Contained lookup(in ScopedName search_name);
    ContainedSeq contents(
        in DefinitionKind limit_type,
        in boolean exclude_inherited);
    ContainedSeq lookup_name(
        in Identifier search_name,
        in long levels_to_search,
        in DefinitionKind limit_type,
        in boolean exclude_inherited
    );
    struct Description {
        Contained contained_object;
        DefinitionKind kind;
        any value;
    };
    typedef sequence<Description>DescriptionSeq;

    DescriptionSeq describe_contents(
        in DefinitionKind limit_type,
        in boolean exclude_inherited,
        in long max_returned_objs);
    ModuleDef create_module,
        in RepositoryId id,
        in Identifier name,
        in VersionSpec version);

    ConstantDef create_constant(
        in RepositoryId id,
        in Identifier name,
        in VersionSpec version,
        in IDLType type,
        in any value);
    StructDef create_struct(
```

```
        in RepositoryId id,
        in Identifier name,
        in VersionSpec version,
        in StructMemberSeq members);

    NativeDef create_native(
        in RepositoryId id,
        in Identifier name,
        in VersionSpec version);

    UnionDef create_union(
        in RepositoryId id,
        in Identifier name,
        in VersionSpec version,
        in IDLType discriminator_type,
        in UnionMemberSeq members);

    EnumDef create_enum(
        in RepositoryId id,
        in Identifier name,
        in VersionSpec version,
        in EnumMemberSeq members);

    AliasDef create_alias(
        in RepositoryId id,
        in Identifier name,
        in VersionSpec version,
        in IDLType original_type);

    ExceptionDef create_exception(
        in RepositoryId id,
        in Identifier name,
        in VersionSpec version,
        in StructMemberSeq members);

    InterfaceDef create_interface(
        in RepositoryId id,
        in Identifier name,
        in VersionSpec version,
        in InterfaceDefSeq base_interfaces,
        in boolean is_abstract);

    ValueDef create_value(
        in RepositoryId id,
        in Identifier name,
        in VersionSpec version,
        in boolean is_custom,
        in boolean is_abstract,
        in ValueDef base_value,
        in boolean is_truncatable,
        in ValueDefSeq abstract_base_values,
        in InterfaceDefSeq supported_interfaces,
        in InitializerSeq initializers);
};
```

6.11.2 Container のメソッド

- public org.omg.CORBA.Contained[ ] **contents**(
  org.omg.CORBA.DefinitionKind **limit_type**,
  boolean **exclude_inherited**)
  このメソッドは，直接 Container に含まれる包含オブジェクト定義，または Container へ継承される包含オブジェクト定義のリストを返します。このメソッドを使用して，Repository でのオブジェクト定義の階層を操作できます。Repository のモジュール群に含まれるすべてのオブジェクト定義が返されたあとに，各モジュールに含まれるすべてのオブジェクト定義が返されます。
  - limit_type
    返されるインタフェースオブジェクト型。dk_all を指定すると，すべての型のオブジェクトが返されます。
  - exclude_inherited
    true を設定した場合，継承されたオブジェクトは返されません。
- public org.omg.CORBA.InterfaceDef **create_abstract_interface**(
  java.lang.String **id**,
  java.lang.String **name**,
  java.lang.String **version**,
  org.omg.CORBA.AbstractInterfaceDef[ ] **base_interfaces**)
  このメソッドは，指定した属性で AbstractInterfaceDef オブジェクトをこの Container の中に生成し，新しく生成したオブジェクトのリファレンスを返します。作成したインタフェースが格納できるのは，abstract 型のインタフェースの定義だけです。
  - id
    インタフェースのリポジトリ ID
  - name
    インタフェースの名前
  - version
    インタフェースのバージョン
  - base_interfaces
    このインタフェースの継承元となるすべてのインタフェースの一覧
- public org.omg.CORBA. AliasDef **create_alias**(
  java.lang.String **id**,
  java.lang.String **name**,
  java.lang.String **version**,
  org.omg.CORBA.IDLType **original_type**)
  このメソッドは，この Container 内に AliasDef オブジェクトを指定された属性で生成し，新しく生成したオブジェクトのリファレンスを返します。

- id
  エイリアスのリポジトリ ID
- name
  エイリアス名
- version
  エイリアスのバージョン
- original_type
  このエイリアスのオリジナルのオブジェクトの IDL 型

- public org.omg.CORBA.ConstantDef **create_constant**(
  java.lang.String **id**,
  java.lang.String **name**,
  java.lang.String **version**,
  org.omg.CORBA.IDLType **type**,
  org.omg.CORBA.Any **value**)
  このメソッドは，指定された属性でこの Container 内に ConstantDef オブジェクトを生成し，新しく生成したオブジェクトのリファレンスを返します。
  - id
    定数のリポジトリ ID
  - name
    定数名
  - version
    定数のバージョン
  - type
    定数の IDL 型
  - value
    定数の値。Any オブジェクトで表されます。
- public org.omg.CORBA.EnumDef **create_enum**(
  java.lang.String **id**,
  java.lang.String **name**,
  java.lang.String **version**,
  java.lang.String **members**[ ])
  このメソッドは，指定された属性でこの Container 内に EnumDef オブジェクトを生成し，新しく生成したオブジェクトのリファレンスを返します。
  - id
    列挙体のリポジトリ ID

- name
  列挙体名
- version
  列挙体のバージョン
- members
  列挙体の値の一覧

- public org.omg.CORBA.ExceptionDef **create_exception**(
  java.lang.String **id**,
  java.lang.String **name**,
  java.lang.String **version**,
  org.omg.CORBA.StructMember[ ] **members**)
  このメソッドは，指定された属性でこの Container 内に ExceptionDef オブジェクトを生成し，新しく生成したオブジェクトのリファレンスを返します。
  - id
    例外のリポジトリ ID
  - name
    例外名
  - version
    例外のバージョン
  - members
    例外のメンバのすべての型の一覧
- public org.omg.CORBA.InterfaceDef **create_interface**(
  java.lang.String **id**,
  java.lang.String **name**,
  java.lang.String **version**,
  org.omg.CORBA.InterfaceDef[ ] **base_interfaces**)
  このメソッドは，指定された属性でこの Container の中に concrete 型の InterfaceDef オブジェクトを生成し，新しく生成したオブジェクトのリファレンスを返します。AbstractInterfaceDef と異なり，このインタフェースは abstract 型と concrete 型の両方のインタフェースの定義を格納できます。
  - id
    インタフェースのリポジトリ ID
  - name
    インタフェース名
  - version
    インタフェースのバージョン

- base_interfaces
  このインタフェースの継承元である全インタフェースの一覧

- public org.omg.CORBA.ModuleDef **create_module**(
  java.lang.String **id**,
  java.lang.String **name**,
  java.lang.String **version**)
  このメソッドは，指定された属性でこの Container 内に ModuleDef オブジェクトを生成し，新しく生成したオブジェクトのリファレンスを返します。
  - id
    モジュールのリポジトリ ID
  - name
    モジュール名
  - version
    モジュールのバージョン
- public org.omg.CORBA.NativeDef **create_native**(
  java.lang.String **id**,
  java.lang.String **name**,
  java.lang.String **version**)
  このメソッドは，指定した属性で該当する Container オブジェクトの中に NativeDef オブジェクトを生成し，新しく生成したオブジェクトのリファレンスを返します。
  - id
    構造体のリポジトリ ID
  - name
    構造体の名前
  - version
    構造体のバージョン
- public org.omg.CORBA.StructDef **create_struct**(
  java.lang.String **id**,
  java.lang.String **name**,
  java.lang.String **version**,
  org.omg.CORBA.StructMember **members**[ ])
  このメソッドは，指定された属性で，この Container 内に StructDef オブジェクトを生成し，新しく生成したオブジェクトのリファレンスを返します。
  - id
    構造体のリポジトリ ID

- name
  構造体の名前
- version
  構造体のバージョン
- members
  構造体のフィールドの値

- public org.omg.CORBA.UnionDef **create_union**(
  java.lang.String **id**,
  java.lang.String **name**,
  java.lang.String **version**,
  org.omg.CORBA.IDLType **discriminator_type**,
  org.omg.CORBA.UnionMember[ ] **members**)
  このメソッドは，指定された属性でこの Container 内に UnionDef オブジェクトを生成し，新しく生成したオブジェクトのリファレンスを返します。
  - id
    union のリポジトリ ID
  - name
    union 名
  - version
    union のバージョン
  - discriminator_type
    union の識別値の IDL 型
  - members
    各 union のフィールドの型の一覧
- public org.omg.CORBA.ContainerPackage.Description[ ]
  describe_contents(
  org.omg.CORBA.DefinitionKind **limit_type**,
  boolean **exclude_inherited**,
  int **max_returned_objs**)
  このメソッドは，このコンテナに直接含まれているか，またはこのコンテナに継承されたすべての定義に関する記述を返します。
  - limit_type
    返されるインタフェースオブジェクトの型
  - exclude_inherited
    true を設定した場合，継承されたオブジェクトは返されません。

- max_returned_objs
  返されるオブジェクトの最大数。このパラメタに-1 を設定すると，全オブジェクトが返されます。

- public org.omg.CORBA.Contained **lookup**(
  java.lang.String **search_name**)
  このメソッドは，指定されたスコープ名で，このコンテナと相対的な定義を探します。先頭が「::」で始まる絶対スコープ名を指定すると，囲みリポジトリ内の定義を探せます。オブジェクトが見つからない場合，NULL が返されます。
  - search_name
    探す対象のオブジェクト名

- public org.omg.CORBA.Contained[ ] **lookup_name**(
  java.lang.String **search_name**,
  int **levels_to_search**,
  org.omg.CORBA.DefinitionKind **limit_type**,
  boolean **exclude_inherited**)
  このメソッドは，ある特定のオブジェクト内のオブジェクトを名前で探します。検索対象の階層内のレベル数，オブジェクトの型，継承されたオブジェクトを返すかどうか，などで検索を制限できます。
  - search_name
    探す対象のオブジェクトまたはオブジェクト群の名前
  - levels_to_search
    検索対象の階級内のレベル数。このパラメタに-1 を設定すると，すべてのレベルが検索対象となります。1 を設定すると，このオブジェクトだけを検索します。
  - limit_type
    返されるインタフェースオブジェクトの型
  - exclude_inherited
    true を設定した場合，継承されたオブジェクトは返されません。

- public org.omg.CORBA.ValueDef **create_value**(
  java.lang.String **id**,
  java.lang.String **name**,
  java.lang.String **version**,
  boolean **is_custom**,
  boolean **is_abstract**,
  org.omg.CORBA.ValueDef **base_value**,
  boolean **is_truncatable**,
  org.omg.CORBA.ValueDef[ ] **abstract_base_values**,
  org.omg.CORBA.InterfaceDef **supported_interfaces**,
  org.omg.CORBA.Initializer[ ] **initializers**)

このメソッドは，指定した属性で該当する Container オブジェクトの中に ValueDef オブジェクトを生成し，新しく生成したオブジェクトのリファレンスを返します。

- id
  構造体のリポジトリ ID
- name
  構造体の名前
- version
  構造体のバージョン
- is_custom
  true を設定すると，custom 型 valuetype を生成します。
- is_abstract
  true を設定すると，abstract 型 valuetype を生成します。
- base_value
  ベース値の定義
- is_truncatable
  true を設定すると，truncatable インタフェースを生成します。
- abstract_base_values
  abstract 型ベース定義の配列
- supported_interfaces
  サポートするインタフェース定義の配列
- initializers
  この値の型がサポートするイニシャライザの一覧

- public org.omg.CORBA.ValueBoxDef **create_value_box**(
  java.lang.String **id**,
  java.lang.String **name**,
  java.lang.String **version**,
  org.omg.CORBA.IDLType **original_type**)
  このメソッドは，指定した属性で該当する Container オブジェクトの中に ValueBoxDef オブジェクトを生成し，新しく生成したオブジェクトのリファレンスを返します。
  - id
    構造体のリポジトリ ID
  - name
    構造体の名前
  - version
    構造体のバージョン

- original_type
  該当するオブジェクトがエイリアスの場合のオリジナルオブジェクトの IDL 型

6.12 ContainerPackage.Description

```
public final class org.omg.CORBA.ContainerPackage.Description extends
    java.lang.Object implements org.omg.CORBA.portable.IDLEntity
```

このクラスは，Contained インタフェースから派生したインタフェースリポジトリ内の各項目の一般的な記述を提供します。

このクラスには，Helper クラスと Holder クラスもあります。これらのクラスとそのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.12.1 ContainerPackage.Description の変数

- public org.omg.CORBA.Contained **contained_object**
  含まれた項目です。
- public org.omg.CORBA.DefinitionKind **kind**
  項目の種類です。
- public org.omg.CORBA.Any **value**
  項目の値です。

6.12.2 ContainerPackage.Description のメソッド

- public **Description**( )
  このメソッドは，Description のデフォルトコンストラクタです。
- public **Description**(
  org.omg.CORBA.Contained **contained_object**,
  org.omg.CORBA.DefinitionKind **kind**,
  org.omg.CORBA.Any **value**)
  このメソッドは，指定されたパラメタを使用して，Description を構成します。
  - contained_object
    含まれる項目
  - kind
    この項目の種類
  - value
    この項目の値を表す Any オブジェクト

6.13 DefinitionKind

```
public class org.omg.CORBA.DefinitionKind extends
    java.lang.Object implementsorg.omg.CORBA.portable.IDLEntity
```

このクラスは，リポジトリが格納できるオブジェクトの型を列挙します。列挙体中の各値は IDL 中のデータ型を反映します。

このクラスには，Helper クラスと Holder クラスもあります。これらのクラスとそのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.13.1 DefinitionKind のメソッド

- public int value( )
  このメソッドは，DefinitionKind の整数値を返します。

6.13.2 DefinitionKind の列挙値

DefinitionKind の定数値を次の表に示します。

表 6-1 DefinitionKind の定数値（Java）

Java オブジェクト	整数定数	値
dk_AbstractInterface	_dk_AbstractInterface	abstract 型インタフェース
dk_all	_dk_all	すべての可能な型（リポジトリ lookup メソッドで使用）
dk_Alias	_dk_Alias	エイリアス
dk_Array	_dk_Array	配列
dk_Attribute	_dk_Attribute	属性
dk_Constant	_dk_Constant	定数
dk_Enum	_dk_Enum	列挙体
dk_Exception	_dk_Exception	例外
dk_Fixed	_dk_Fixed	fixed
dk_Interface	_dk_Interface	concrete 型インタフェース
dk_Module	_dk_Module	モジュール
dk_Native	_dk_Native	ネイティブ

Java オブジェクト	整数定数	値
dk_none	_dk_none	すべての型を除く（リポジトリ lookup メソッドで使用）
dk_Operation	_dk_Operation	インタフェースオペレーション
dk_Primitive	_dk_Primitive	基本型（int や long など）
dk_Repository	_dk_Repository	リポジトリ
dk_Sequence	_dk_Sequence	シーケンス
dk_String	_dk_String	文字列
dk_Struct	_dk_Struct	struct
dk_Typedef	_dk_Typedef	typedef
dk_Union	_dk_Union	union
dk_Value	_dk_Value	value
dk_ValueBox	_dk_ValueBox	ValueBox
dk_ValueMember	_dk_ValueMember	ValueMember
dk_Wstring	_dk_Wstring	Unicode 文字列

6.14 EnumDef

```
public interface org.omg.CORBA.EnumDef extends
    org.omg.CORBA.EnumDefOperations,
    org.omg.CORBA.TypedefDef,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，インタフェースリポジトリに格納されている列挙体を表すために使用します。このインタフェースは，列挙体のメンバ一覧を設定，検索するためのメソッドを提供します。

このインタフェースの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.14.1 EnumDef のメソッド

- public java.lang.String[ ] members( )
  このメソッドは，この列挙体のメンバ一覧を返します。
- public void members(
  java.lang.String members[ ])
  このメソッドは，この列挙体のメンバ一覧を設定します。
  - members
    メンバ一覧

6.15 ExceptionDef

```
public interface org.omg.CORBA.ExceptionDef extends
    org.omg.CORBA.ExceptionDefOperations,
    org.omg.CORBA.Contained,
    org.omg.CORBA.Container,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，インタフェースリポジトリに格納されている例外を表すために使用します。このインタフェースは，例外のメンバ一覧を設定，検索するためのメソッドと，例外の TypeCode を検索するためのメソッドを提供します。

このインタフェースの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.15.1 ExceptionDef のメソッド

- public org.omg.CORBA.StructMember[ ] members( )
  このメソッドは，この例外のメンバ一覧を返します。
- public void members(
  org.omg.CORBA. StructMember members[ ])
  このメソッドは，例外のメンバ一覧を設定します。
  - members
    メンバ一覧
- public org.omg.CORBA.TypeCode type( )
  このメソッドは，この例外の型を表す TypeCode を返します。

6.16 ExceptionDescription

```
public final class org.omg.CORBA.ExceptionDescription extends
    java.lang.Object implements org.omg.CORBA.portable.IDLEntity
```

このクラスは，インタフェースリポジトリに格納された例外を記述します。

このクラスには，Helper クラスと Holder クラスもあります。これらのクラスとそのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.16.1 ExceptionDescription の変数

- public java.lang.String **name**
  例外名です。
- public java.lang.String **id**
  例外のリポジトリ ID です。
- public java.lang.String **defined_in**
  この例外が定義されているモジュールまたはインタフェースのリポジトリ ID です。
- public org.omg.CORBA.VersionSpec **version**
  例外のバージョンです。
- public org.omg.CORBA.Typecode **type**
  例外の IDL 型です。

6.16.2 ExceptionDescription のメソッド

- public **ExceptionDescription**( )
  このメソッドは，ExceptionDescription のデフォルトコンストラクタです。
- public **ExceptionDescription**(
  java.lang.String **name**,
  java.lang.String **id**,
  java.lang.String **defined_in**,
  org.omg.CORBA.VersionSpec **version**,
  org.omg.CORBA.TypeCode **type**,
  org.omg.CORBA.Any **value**)
  このメソッドは，指定されたパラメタを使用して，ExceptionDescription を構成します。
  - name

この例外の名前

- id
  この例外のリポジトリ ID
- defined_in
  この例外が定義されたモジュールまたはインタフェース
- version
  オブジェクトのバージョン
- type
  例外の IDL タイプコード
- value
  この例外の値

6.17 FixedDef

```
public interface org.omg.CORBA.FixedDef extends
    org.omg.CORBA.FixedDefOperations,
    org.omg.CORBA.IDLType,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，インタフェースリポジトリに格納されている fixed 型を表すために使用します。

6.17.1 FixedDef のメソッド

- public short **digits**( )
  このメソッドは，fixed 型のけた数を返します。
- public void **digits**(
  short **digits**)
  このメソッドは，この fixed 型にけた数を設定します。
- public short **scale**( )
  スケールは，fixed 型の小数点以下のけた数です。このメソッドは，fixed 型のスケールを返します。
- public void **scale** (
  short **scale**)
  このメソッドは，fixed 型のスケールを設定します。

6.18 FullValueDescription

```
public final class org.omg.CORBA.ValueDefPackage.FullValueDescription
    extends java.lang.Object implements org.omg.CORBA.portable.IDLEntity
```

このクラスは，インタフェースリポジトリに格納されている完全な値定義を表すために使用します。詳細については，「6.45 ValueDef」を参照してください。

6.18.1 FullValueDescription の変数

- public java.lang.String **name**
  valuetype の名前です。
- public java.lang.String **id**
  valuetype のリポジトリ ID です。
- public boolean **is_abstract**
  true の場合，valuetype は abstract 型です。false の場合，valuetype は concrete 型です。
- public boolean **is_custom**
  true の場合，valuetype に対して custom 型のマーシャリングを実行します。
- public java.lang.String **defined_in**
  valuetype が定義されているモジュールのリポジトリ ID です。
- public java.lang.String **version**
  valuetype のバージョンです。
- public org.omg.CORBA.OperationDescription[ ] **operations**
  valuetype が提供するオペレーションの一覧です。
- public org.omg.CORBA.AttributeDescription[ ] **attributes**
  valuetype の属性の一覧です。
- public org.omg.CORBA.ValueMember[ ] **members**
  valuetype のメンバの配列です。
- public org.omg.CORBA.Initializer[ ] **initializers**
  イニシャライザの配列です。
- public java.lang.String[ ] **supported_interfaces**
  この valuetype がサポートするインタフェースの一覧です。
- public java.lang.String[ ] **abstract_base_values**
  この valuetype の継承元となっているすべての abstract 型ベース値の，通知 ID の一覧です。
- public boolean **is_truncatable**

true の場合，値はベースの valuetype に安全にマッピングできます。

- public java.lang.String **base_values**
  この型に対して concrete 型ベース値がある場合，その通知 ID です。
- public org.omg.CORBA.TypeCode **type**
  valuetype の IDL TypeCode です。

6.18.2 FullValueDescription のメソッド

- public org.omg.CORBA.**FullValueDescription**(
  java.lang.String **name**,
  java.lang.String **id**,
  boolean **is_abstract**,
  boolean **is_custom**,
  java.lang.String **defined_in**,
  java.lang.String **version**,
  org.omg.CORBA.OperationDescription[ ] **operations**,
  org.omg.CORBA.AttributeDescription[ ] **attributes**,
  org.omg.CORBA.ValueMember[ ] **members**,
  org.omg.CORBA.Initializer[ ] **initializers**,
  java.lang.String[ ] **supported_interfaces**,
  java.lang.String[ ] **abstract_base_values**,
  boolean **is_truncatable**,
  java.lang.String **base_values**,
  org.omg.CORBA.Typecode **type**)
  このメソッドは，FullValueDescription を構成します。
  - name
    valuetype の名前
  - id
    valuetype の識別子
  - is_abstract
    valuetype が abstract であることを表すパラメタ
  - is_custom
    valuetype が custom であることを表すパラメタ
  - defined_in
    valuetype が定義されたモジュールまたはインタフェース

- version
  valuetype のバージョン
- operations
  オペレーションの一覧
- attributes
  属性の一覧
- members
  値メンバ記述子の一覧
- initializers
  イニシャライザの一覧
- supported_interfaces
  サポートするインタフェースの一覧
- abstract_base_values
  abstract valuetype の一覧
- is_truncatable
  valuetype が truncatable であることを表すパラメタ
- base_values
  真の（abstract でない）ベース値（ある場合）
- type
  valuetype の TypeCode

6.19 IDLType

```
public interface org.omg.CORBA.IDLType extends
    org.omg.CORBA.IDLTypeOperations,
    org.omg.CORBA.IRObject,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，IDL 型を形成するリポジトリオブジェクトにマークを付けるためのインタフェースとして定義されています。例えば，EnumDef は IDL の construct 型に対応するので IDLType から継承されますが，OperationDef は正規のリポジトリオブジェクトであり IDL 型ではないため，IDLType から継承されません。各 IDLType オブジェクトは対応する TypeCode を持ちます。IDLType はオブジェクトの型を一意に識別します。詳細については，「5.24 TypeCode」を参照してください。

このインタフェースの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.19.1 IDL の定義

```
interface IDLType : CORBA::IRObject {
    readonly attribute TypeCode type;
};
```

6.19.2 IDLType のメソッド

- public org.omg.CORBA.TypeCode type( )

  このメソッドは，インタフェースリポジトリの中の IDL 型定義を表す IRObject に対応する TypeCode オブジェクトを返します。

6.20 InterfaceDef

```
public interface org.omg.CORBA.InterfaceDef extends
    org.omg.CORBA.InterfaceDefOperations,
    org.omg.CORBA.Container,
    org.omg.CORBA.Contained,
    org.omg.CORBA.IDLType,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，インタフェースリポジトリに格納されている concrete 型インタフェースを表すために使用します。このインタフェースは，ベースインタフェースを設定および取得して，属性，オペレーション，およびインタフェース記述を生成するためのメソッドを提供します。

このインタフェースの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.20.1 IDL の定義

```
interface InterfaceDef:Container,Contained,IDLType {
    attribute InterfaceDefSeq base_interfaces;
    boolean is_a(in RepositoryId interface_id);

    struct FullInterfaceDescription {
        Identifier name;
        RepositoryId id;
        RepositoryId defined_in;
        VersionSpec version;
        OpDescriptionSeq operations;
        AttrDescriptionSeq attributes;
        RepositoryIdSeq base_interfaces;
        TypeCode type;
    };
    FullInterfaceDescription describe_interface( );
    AttributeDef create_attribute(
        in RepositoryId id,
        in Identifier name,
        in VersionSpec version,
        in IDLType type,
        in AttributeMode mode);
    OperationDef create_operation(
        in CORBA::RepositoryId id,
        in Identifier name,
        in VersionSpec version,
        in IDLType result,
        in OperationMode mode,
        in ParDescriptionSeq params,
        in ExceptionDefSeq exceptions,
        in ContextIdSeq contexts);
};
```

6.20.2 InterfaceDef のメソッド

- public org.omg.CORBA.InterfaceDef[ ] base_interfaces( )
  このメソッドは，このオブジェクトのベースインタフェース一覧を返します。
- public void base_interfaces(
  org.omg.CORBA.InterfaceDef[ ] base_interfaces)
  このメソッドは，このオブジェクトのベースインタフェース一覧を設定します。
  - base_interfaces
    設定するベースインタフェースの一覧
- public org.omg.CORBA.AttributeDef create_attribute(
  java.lang.String id,
  java.lang.String name,
  java.lang.String version,
  org.omg.CORBA.IDLType type,
  org.omg.CORBA.AttributeMode mode)
  このメソッドは，インタフェース定義に属性を追加します。
  - id
    属性の識別子
  - name
    属性名
  - version
    属性のバージョン
  - type
    属性の IDL 型
  - mode
    属性のモード。指定できる値については，「6.6　AttributeMode」を参照してください。
- public org.omg.CORBA.OperationDef create_operation(
  java.lang.String id,
  java.lang.String name,
  java.lang.String version,
  org.omg.CORBA.IDLType result,
  org.omg.CORBA.OperationMode mode,
  org.omg.CORBA.ParameterDescription[ ] params,
  org.omg.CORBA.ExceptionDef[ ] exceptions,
  java.lang.String[ ] contexts)
  このメソッドは，インタフェース定義にオペレーションを追加します。

- id
  オペレーションの識別子
- name
  オペレーション名
- version
  オペレーションのバージョン
- result
  オペレーションの IDL 結果型
- mode
  オペレーションのモード。詳細については，「6.30　OperationMode」を参照してください。
- params
  このオペレーションに対するパラメタの一覧
- exceptions
  このオペレーションで発生する可能性がある例外の一覧
- contexts
  コンテキスト一覧

- public org.omg.CORBA.InterfaceDefPackage.FullInterfaceDescription
  describe_interface( )
  このメソッドは，このオブジェクトに対するインタフェースの記述を返します。
- public boolean is_a(
  java.lang.String interface_id)
  該当するオブジェクトが，指定したインタフェースと互換性のあるインタフェース定義である場合，このメソッドは，true を返します。
  - interface_id
    このオブジェクトと比較するインタフェース識別子

6.21 InterfaceDefPackage.FullInterfaceDescription

```
public final class
    org.omg.CORBA.InterfaceDefPackage.FullInterfaceDescription extends
    java.lang.Object implements org.omg.CORBA.portable.IDLEntity
```

このクラスは，インタフェースリポジトリに格納されているインタフェースを記述します。

6.21.1 InterfaceDefPackage.FullInterfaceDescription の変数

- public java.lang.String **name**
  インタフェース名です。
- public java.lang.String **id**
  インタフェースのリポジトリ ID です。
- public java.lang.String **defined_in**
  このインタフェースが定義されたモジュールのリポジトリ ID です。
- public java.lang.String **version**
  インタフェースのバージョンです。
- public org.omg.CORBA.OperationDescription[ ] **operations**
  インタフェースが提供するオペレーションの一覧です。
- public org.omg.CORBA.AttributeDescription[ ] **attributes**
  インタフェースの属性の一覧です。
- public java.lang.String[ ] **base_interfaces**
  このインタフェースが継承するベースインタフェースの一覧です。
- public org.omg.CORBA.TypeCode **type**
  この変数は，インタフェースの IDL タイプコードを表します。

6.21.2 InterfaceDefPackage.FullInterfaceDescription のメソッド

- public **FullInterfaceDescription**( )
  このメソッドは，FullInterfaceDescription のデフォルトコンストラクタを設定します。
- public **FullInterfaceDescription**(
  final java.lang.String **name**,
  final java.lang.String **id**,
  final java.lang.String **defined_in**,
  final java.lang.String **version**,

final org.omg.CORBA.OperationDescription[ ] operations,
final org.omg.CORBA.AttributeDescription[ ] attributes,
final java.lang.String[ ] base_interfaces,
final org.omg.CORBA.TypeCode type)
このメソッドは，指定されたパラメタを使用して，FullInterfaceDescription を構成します。

- name
  このインタフェースの名前
- id
  このインタフェースのリポジトリ ID
- defined_in
  この属性が定義されたモジュールのリポジトリ ID
- version
  インタフェースのバージョン
- operations
  このインタフェースが提供するオペレーションの一覧
- attributes
  このインタフェースの属性の一覧
- base_interfaces
  該当するインタフェースのベースインタフェースの一覧
- type
  インタフェースの IDL タイプコード

6.22 InterfaceDescription

```
public final class org.omg.CORBA.InterfaceDescription extends
    java.lang.Object implements org.omg.CORBA.portable.IDLEntity
```

このクラスは，インタフェースリポジトリに格納されているインタフェースの定義を提供します。

このクラスには，Helper クラスと Holder クラスもあります。これらのクラスとそのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.22.1 InterfaceDescription の変数

- public java.lang.String **name**
  インタフェース名です。
- public java.lang.String **id**
  インタフェースのリポジトリ ID です。
- public java.lang.String **defined_in**
  このインタフェースが定義されたモジュールのリポジトリ ID です。
- public java.lang.String **version**
  インタフェースのバージョンです。
- public String[ ] **base_interfaces**
  このインタフェースのベースインタフェースの一覧です。

6.22.2 InterfaceDescription のメソッド

- public **InterfaceDescription**( )
  このメソッドは，InterfaceDescription のデフォルトコンストラクタです。
- public **InterfaceDescription**(
  java.lang.String **name**,
  java.lang.String **id**,
  java.lang.String **defined_in**,
  java.lang.String **version**,
  java.lang.String[ ] **base_interfaces**)
  このメソッドは，指定されたパラメタを使用して，InterfaceDescription を構成します。
  - name
    このインタフェースの名前

- id
  このインタフェースのリポジトリ ID
- defined_in
  このインタフェースが定義されたモジュール
- version
  インタフェースのバージョン
- base_interfaces
  インタフェースのベースインタフェース一覧

6.23 IRObject

```
public interface org.omg.CORBA.IRObject extends
    org.omg.CORBA.IRObjectOperations,
    org.omg.CORBA.Object,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，インタフェースリポジトリに格納されている任意のオブジェクトに対する一般的なインタフェースを提供します。

このインタフェースの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.23.1 IDL の定義

```
interface IRObject {
   readonly attribute CORBA::DefinitionKind def_kind;
   void destroy( );
};
```

6.23.2 IRObject のメソッド

- public org.omg.CORBA.DefinitionKind **def_kind**( )
  このメソッドは，この IRObject が表している IDL 定義の種別を返します。
  定義された型の一覧については，「6.13 DefinitionKind」を参照してください。
- public void **destroy**( )
  このメソッドは，この IRObject をインタフェースリポジトリから削除します。

6.24 LocalInterfaceDef

```
public interface org.omg.CORBA.LocalInterfaceDef extends
    org.omg.CORBA.LocalInterfaceDefOperations,
    org.omg.CORBA.InterfaceDef,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，ローカルインタフェースの IDL モジュールを表すために使用します。

このインタフェースの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.25 ModuleDef

```
public interface org.omg.CORBA.ModuleDef extends
    org.omg.CORBA.ModuleDefOperations,
    org.omg.CORBA.Container,
    org.omg.CORBA.Contained,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，インタフェースリポジトリ内の IDL モジュールを表すために使用します。

このインタフェースの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.26 ModuleDescription

```
public final class org.omg.CORBA.ModuleDescription extends
    java.lang.Object implements org.omg.CORBA.portable.IDLEntity
```

このクラスは，インタフェースリポジトリに格納されているモジュールを定義します。

このクラスには，Helper クラスと Holder クラスもあります。これらのクラスとそのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.26.1 ModuleDescription の変数

- public java.lang.String **name**
  モジュール名です。
- public java.lang.String **id**
  モジュールのリポジトリ ID です。
- public java.lang.String **defined_in**
  このモジュールが定義されたモジュールのリポジトリ ID です。
- public java.lang.String **version**
  モジュールのバージョンです。

6.26.2 ModuleDescription のメソッド

- public **ModuleDescription**( )
  このメソッドは，ModuleDescription のデフォルトコンストラクタです。
- public **ModuleDescription**(
  java.lang.String **name**,
  java.lang.String **id**,
  java.lang.String **defined_in**,
  java.lang.String **version**)
  このメソッドは，指定されたパラメタを使用し，ModuleDescription を構成します。
  - name
    このインタフェースの名前
  - id
    このインタフェースのリポジトリ ID
  - defined_in

このモジュールが定義されたモジュール ID

- version
  オブジェクトのバージョン

6.27 NativeDef

```
public interface org.omg.CORBA.NativeDef extends
    org.omg.CORBA.NativeDefOperations,
    org.omg.CORBA.TypedefDef,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，インタフェースリポジトリが格納するネイティブ定義を表すために使用します。Container インタフェースは，包含されたオブジェクトとして NativeDef を生成するためのオペレーションを提供します。

6.28 OperationDef

```
public interface org.omg.CORBA.OperationDef extends
    org.omg.CORBA.OperationDefOperations,
    org.omg.CORBA.Contained,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，インタフェースリポジトリに格納されているインタフェースオペレーションを表すために使用します。このインタフェースは，オペレーションのコンテキスト，モード，パラメタ，および結果値を，設定および取得するためのメソッドを提供します。また，このオペレーションで発生する例外の一覧の検索メソッドも提供します。

このインタフェースの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.28.1 OperationDef のメソッド

- public java.lang.String[ ] contexts( )
  このメソッドは，このオペレーションに対応づけられたコンテキストを返します。
- public void contexts(
  java.lang.String[ ] contexts)
  このメソッドは，このオペレーションのコンテキスト一覧を設定します。
  - contexts
    コンテキスト一覧
- public org.omg.CORBA.ExceptionDef[ ] exceptions( )
  このメソッドは，このオペレーションで発生する例外の一覧を返します。
- public void exceptions(
  org.omg.CORBA.ExceptionDef[ ] exceptions)
  このメソッドは，このオペレーションで発生する例外の一覧を設定します。
  - exceptions
    例外の一覧
- public org.omg.CORBA.OperationMode mode( )
  このメソッドは，このオペレーションのモードを返します。
- public void mode(
  org.omg.CORBA.OperationMode mode)
  このメソッドは，このオペレーションのモードを設定します。
  - mode

設定するモード。パラメタの詳細については，「6.30　OperationMode」を参照してください。

- public org.omg.CORBA.ParameterDescription[ ] **params**( )
  このメソッドは，このオペレーションに対するパラメタの記述を返します。
- public void **params**(
  org.omg.CORBA.ParameterDescription **params**)
  このメソッドは，このオペレーションに対するパラメタの記述を設定します。
  - params
    パラメタの記述
- public org.omg.CORBA.TypeCode **result**( )
  このメソッドは，このオペレーションのリターン値の TypeCode を返します。
- public org.omg.CORBA.IDLType **result_def**( )
  このメソッドは，このオペレーションのリターン値の IDL 型を返します。
- public void **result_def**(
  org.omg.CORBA.IDLType **result_def**)
  このメソッドは，このオペレーションのリターン値の IDL 型を設定します。
  - result_def
    リターン値に対して設定する IDL 型

6.29 OperationDescription

```
public final class org.omg.CORBA.OperationDescription extends
    java.lang.Object implements org.omg.CORBA.portable.IDLEntity
```

このクラスは，インタフェースリポジトリに格納されているオペレーションの情報を表すために記述します。

このクラスには，Helper クラスと Holder クラスもあります。これらのクラスとそのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.29.1 OperationDescription の変数

- public java.lang.String **name**
  オペレーション名です。
- public java.lang.String **id**
  オペレーションのリポジトリ ID です。
- public java.lang.String **defined_in**
  このオペレーションが定義されているインタフェースまたは valuetype のリポジトリ ID です。
- java.lang.String **version**
  オペレーションのバージョンです。
- public org.omg.CORBA.TypeCode **result**
  オペレーションの結果 TypeCode です。
- public org.omg.CORBA.OperationMode **mode**
  オペレーションのモードです。
- public java.lang.String[ ] **contexts**
  このオペレーションに対応づけられたコンテキスト一覧です。
- public org.omg.CORBA.ParameterDescription[ ] **parameters**
  このオペレーションのパラメタです。
- public org.omg.CORBA.ExceptionDescription[ ] **exceptions**
  このオペレーションで発生する可能性のある例外です。

6.29.2 OperationDescription のメソッド

- public **OperationDescription**( )
  このメソッドは，OperationDescription のデフォルトコンストラクタです。
- public **OperationDescription**(

java.lang.String **name**,
java.lang.String **id**,
java.lang.String **defined_in**,
java.lang.String **version**,
org.omg.CORBA.TypeCode **result**,
org.omg.CORBA.OperationMode **mode**,
java.lang.String[ ] **contexts**,
org.omg.CORBA.ParameterDescriptions **parameters**,
org.omg.CORBA.ExceptionDescription[ ] **exceptions**)
このメソッドは，指定されたパラメタを使用して，OperationDescription を構成します。

- name
  このオペレーションの名前
- id
  このオペレーションのリポジトリ ID
- defined_in
  このオペレーションが定義されているインタフェースまたは valuetype の ID
- version
  オブジェクトのバージョン
- result
  オペレーション結果の IDL TypeCode
- mode
  オペレーションのモード
- contexts
  このオペレーションに対するコンテキスト文字列の一覧
- parameters
  このオペレーションに対するパラメタの一覧
- exceptions
  このオペレーションで発生する可能性がある例外の一覧

6.30 OperationMode

```
public final class org.omg.CORBA.OperationMode extends
    java.lang.Object implements org.omg.CORBA.portable.IDLEntity
```

このクラスは，オペレーションのモードを列挙します。モードには次の 2 種類があります。

- org.omg.CORBA.OperationMode.OP_ONEWAY
  一方向オペレーションモード。このモードでは，クライアントアプリケーションは応答を期待しません。
- org.omg.CORBA.OperationMode.OP_NORMAL
  標準リクエストモード。このモードでは，リクエストの結果を格納するオブジェクトインプリメンテーションが応答をクライアントに送信します。

このクラスには，Helper クラスと Holder クラスもあります。これらのクラスとそのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java)」を参照してください。

6.31 ParameterDescription

```
public final class org.omg.CORBA.ParameterDescription extends
    java.lang.Object implements org.omg.CORBA.portable.IDLEntity
```

このクラスは，インタフェースリポジトリに格納されているオペレーションのパラメタを記述します。

このクラスには，Helper クラスと Holder クラスもあります。これらのクラスとそのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.31.1 ParameterDescription の変数

- public java.lang.String **name**
  パラメタ名です。
- public org.omg.CORBA.TypeCode **type**
  パラメタの TypeCode です。
- public org.omg.CORBA.IDLType **type_def**
  パラメタの IDL 型です。
- public org.omg.CORBA.ParameterMode **mode**
  パラメタのモードです。パラメタの詳細については，「6.32 ParameterMode」を参照してください。

6.31.2 ParameterDescription のメソッド

- public **ParameterDescription**( )
  このメソッドは，ParameterDescription のデフォルトコンストラクタです。
- public **ParameterDescription**(
  java.lang.String **name**,
  org.omg.CORBA.TypeCode **type**,
  org.omg.CORBA.IDLType **type_def**,
  org.omg.CORBA.ParameterMode **mode**)
  このメソッドは，指定されたパラメタを使用し，ParameterDescription を構成します。
  - name
    パラメタ名
  - type
    パラメタの TypeCode
  - type_def

パラメタの IDL 型

- mode
  パラメタのモード

6.32 ParameterMode

```
public class org.omg.CORBA.ParameterMode extends
    java.lang.Object implementsorg.omg.CORBA.portable.IDLEntity
```

このクラスは，パラメタの次に示す 3 種類のモードを列挙します。

- PARAM_IN
  クライアントからサーバへの入力に使用します。
- PARAM_OUT
  サーバからクライアントへの結果の出力に使用します。
- PARAM_INOUT
  クライアントからの入力とサーバからの出力の両方に使用します。

このクラスには，Helper クラスと Holder クラスもあります。これらのクラスとそのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.33 PrimitiveDef

```
public interface org.omg.CORBA.PrimitiveDef extends
    org.omg.CORBA.PrimitiveDefOperations,
    org.omg.CORBA.IDLType,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，インタフェースリポジトリに格納されている基本型（int，long など）を表すために使用します。このインタフェースは，表示中の基本型の種類を検索するためのメソッドを提供します。

このインタフェースの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.33.1 PrimitiveDef のメソッド

- public org.omg.CORBA.PrimitiveKind kind( )
  このメソッドは，このオブジェクトが表す基本型の種類を表します。

6.34 PrimitiveKind

```
public final class org.omg.CORBA.PrimitiveKind extends
    java.lang.Object implements org.omg.CORBA.portable.IDLEntity
```

このクラスは，IDL にある基本型を列挙します。このクラスは，列挙体（_pk_で始まる名前の PrimitiveKind 列挙体）と Java オブジェクトの集合（pk_で始まる名前の PrimitiveKind Java オブジェクト）の両方を提供します。

このクラスには，Helper クラスと Holder クラスもあります。これらのクラスとそのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.34.1 PrimitiveKind のメソッド

- public int **value**( )
  このメソッドは，定数を表す整数を返します。

6.34.2 PrimitiveKind の定数

PrimitiveKind の定数値を次の表に示します。

表 6–2 PrimitiveKind の定数値（Java）

Java オブジェクト	整数定数	意味
pk_any	_pk_any	Any オブジェクト
pk_boolean	_pk_boolean	boolean
pk_char	_pk_char	文字
pk_double	_pk_double	double
pk_float	_pk_float	float
pk_long	_pk_long	long
pk_longdouble	_pk_longdouble	long double
pk_longlong	_pk_longlong	long long
pk_null	_pk_null	null
pk_octet	_pk_octet	オクテット文字列
pk_octet	_pk_octet	オクテット文字列
pk_objref	_pk_objref	オブジェクトリファレンス
pk_short	_pk_short	short

Java オブジェクト	整数定数	意味
pk_string	_pk_string	文字列
pk_TypeCode	_pk_TypeCode	TypeCode オブジェクト
pk_ulong	_pk_ulong	unsigned long
pk_ulonglong	_pk_ulonglong	unsigned long
pk_ushort	_pk_ushort	unsigned short
pk_void	_pk_void	void
pk_wchar	_pk_wchar	Unicode 文字
pk_wstring	_pk_wstring	Unicode 文字列

6.35 Repository

```
public interface org.omg.CORBA.Repository extends
    org.omg.CORBA.RepositoryOperations,
    org.omg.CORBA.Container,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，クライアントが使用できるオブジェクトの定義を格納するために使用する，インタフェースリポジトリ自体へのインタフェースを提供します。Repository インタフェースは，定義を格納，および取得するためのメソッドを提供します。Repository オブジェクトは，インタフェースリポジトリ内の階層での唯一のルートオブジェクトです。

このインタフェースの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.35.1 Repository のメソッド

- public org.omg.CORBA.ArrayDef **create_array**(
  int **length**, org.omg.CORBA.IDLType **element_type**)
  このメソッドは，指定された長さおよび要素型で，リポジトリ内に配列定義を生成します。生成された ArrayDef のリファレンスが返されます。
  - length
    配列内の要素数。この値は 0 より大きくしてください。
  - element_type
    配列が含む要素の IDL 型
- public org.omg.CORBA.SequenceDef **create_sequence**(
  int **bound**, org.omg.CORBA.IDLType **element_type**)
  このメソッドは，指定された要素数（バウンド）および要素型で，リポジトリ内に配列定義を生成します。生成された SequenceDef のリファレンスが返されます。
  - bound
    シーケンスの最大長。この値は 0 以上にしてください。
  - element_type
    シーケンスが含む要素の IDL 型
- public org.omg.CORBA.StringDef **create_string**(
  int **bound**)
  このメソッドは，指定された文字数（バウンド）で，リポジトリ内に文字列定義を生成します。生成された StringDef のリファレンスが返されます。
  - bound

文字列の最大数。この値は 0 より大きくしてください。

- public org.omg.CORBA.WstringDef **create_wstring**(
  int **bound**)
  このメソッドは，指定された文字数（バウンド）で，リポジトリ内に Unicode 文字列定義を生成します。生成された WstringDef のリファレンスが返されます。
  - bound
    文字列の最大数。この値は 0 より大きくしてください。
- public org.omg.CORBA.PrimitiveDef **get_primitive**(
  org.omg.CORBA.PrimitiveKind **kind**)
  このメソッドは，指定された PrimitiveKind に対する PrimitiveDef オブジェクトを返します。
  - kind
    基本型の種類
- public org.omg.CORBA.Contained **lookup_id**(
  java.lang.String **search_id**)
  このメソッドは，指定した検索 ID に一致するオブジェクトを，インタフェースリポジトリから検索します。一致するオブジェクトが見つからない場合，null が返されます。
  - search_id
    検索に使用する識別子
- public org.omg.CORBA.FixedDef **create_fixed**(
  short **digits**, short **scale**)
  このメソッドは，fixed 型のけた数とスケールを設定します。
  - digits
    fixed 型のけた数
  - scale
    fixed 型のスケール

6.36 SequenceDef

```
public interface org.omg.CORBA.SequenceDef extends
    org.omg.CORBA.SequenceDefOperations,
    org.omg.CORBA.IDLType,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，インタフェースリポジトリに格納されているシーケンスを表すために使用します。このインタフェースは，シーケンスのバウンドと要素型を設定および取得するためのメソッドを提供します。

このインタフェースの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.36.1 SequenceDef のメソッド

- public int **bound**( )
  このメソッドは，シーケンスのバウンドを返します。
- public void **bound**(
  int **bound**)
  このメソッドは，シーケンスのバウンドを設定します。
  - bound
    シーケンスのバウンド
- public org.omg.CORBA.TypeCode **element_type**( )
  このメソッドは，このシーケンス内の要素型を表す TypeCode を返します。
- public org.omg.CORBA.IDLType **element_type_def**( )
  このメソッドは，このシーケンスに格納されている要素の IDL 型を返します。
- public void element_type_def(
  org.omg.CORBA.IDLType **element_type_def**)
  このメソッドは，このシーケンスに格納されている要素の IDL 型を設定します。
  - element_type_def
    設定する IDL 型

6.37 StringDef

```
public interface org.omg.CORBA.StringDef extends
    org.omg.CORBA.StringDefOperations,
    org.omg.CORBA.IDLType,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，インタフェースリポジトリに格納されている String を表すために使用します。このインタフェースは，文字列のバウンドを設定および取得するためのメソッドを提供します。

このインタフェースの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.37.1 StringDef のメソッド

- public int **bound**( )

  このメソッドは，String のバウンド（最大長）を返します。アンバウンデッドの String オブジェクトの場合，0 を返します。

- public void **bound**(

  int **bound**)

  このメソッドは，String のバウンドを設定します。バウンドをアンバウンデッドに設定する場合は，0 を渡してください。

  - bound

    新しく設定する String オブジェクトのバウンド

6.38 StructDef

```
public interface org.omg.CORBA.StructDef extends
    org.omg.CORBA.StructDefOperations,
    org.omg.CORBA.TypedefDef,
    org.omg.CORBA.Container,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，インタフェースリポジトリに格納されている構造体を表すために使用します。このインタフェースは，構造体のメンバ一覧を設定および取得するためのメソッドを提供します。

このインタフェースの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.38.1 StructDef のメソッド

- public org.omg.CORBA.StructMember[ ] members( )
  このメソッドは，構造体のメンバ一覧を返します。
- public void members(
  org.omg.CORBA.StructMember[ ] members)
  このメソッドは，構造体のメンバ一覧を設定します。
  - members
    メンバ一覧

6.39 StructMember

```
public final class org.omg.CORBA.StructMember extends
    java.lang.Object implements org.omg.CORBA.portable.IDLEntity
```

このクラスは struct の各フィールドを定義するのに使用します。

6.39.1 StructMember の変数

- public java.lang.String **name**
  StructMember の名前です。
- public org.omg.CORBA.TypeCode **type**
  StructMember の IDL 型です。
- public org.omg.CORBA.IDLType **type_def**
  StructMember の IDL 型定義です。

6.39.2 StructMember のメソッド

- public **StructMember**(
  final java.lang.String **name**,
  final org.omg.CORBA.TypeCode **type**,
  final org.omg.CORBA.IDLType **type_def**)
  このメソッドは，指定したパラメタを使用して StructMember オブジェクトを生成します。
  - name
    この StructMember の名前
  - type
    StructMember の IDL タイプコード
  - type_def
    StructMember の型定義

6.40 TypedefDef

```
public interface org.omg.CORBA.TypedefDef extends
    org.omg.CORBA.TypedefDefOperations,
    org.omg.CORBA.Contained,
    org.omg.CORBA.IDLType,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，インタフェースリポジトリに格納されている，ユーザ定義の構造体を表します。次に示すインタフェースはすべてこのインタフェースから継承します。

- AliasDef
- EnumDef
- NativeDef
- StructDef
- UnionDef
- WstringDef

このインタフェースの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.41 TypeDescription

```
public final class org.omg.CORBA.TypeDescription extends
    java.lang.Object implements org.omg.CORBA.portable.IDLEntity
```

このクラスは，インタフェースリポジトリに格納されているオペレーションの型を記述した情報を含んでいます。

このクラスには，Helper クラスと Holder クラスもあります。これらのクラスとそのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.41.1 TypeDescription の変数

- public java.lang.String **name**
  型名です。
- public java.lang.String **id**
  型のリポジトリ ID です。
- public java.lang.String **defined_in**
  この型が定義されたモジュール名またはインタフェース名です。
- public java.lang.String **version**
  型のバージョンです。
- public org.omg.CORBA.Typecode **type**
  型の IDL 型です。

6.41.2 TypeDescription のメソッド

- public **TypeDescription**( )
  このメソッドは，TypeDescription のデフォルトコンストラクタです。
- public **TypeDescription**(
  java.lang.String **name**,
  java.lang.String **id**,
  java.lang.String **defined_in**,
  java.lang.String **version**,
  org.omg.CORBA.TypeCode **type**)
  このメソッドは，指定されたパラメタを使用して，TypeDescription を構成します。
  - name

この型の名前

- id
  この型のリポジトリ ID
- defined_in
  この型が定義されたモジュールまたはインタフェース
- version
  オブジェクトのバージョン
- type
  型の IDL タイプコード

6.42 UnionDef

```
public interface org.omg.CORBA.UnionDef extends
    org.omg.CORBA.UnionDefOperations,
    org.omg.CORBA.TypedefDef,
    org.omg.CORBA.Container,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，インタフェースリポジトリに格納されている union を表すために使用します。

このインタフェースの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.42.1 UnionDef のメソッド

- public org.omg.CORBA.TypeCode **discriminator_type**( )
  このメソッドは，union のディスクリミネータの TypeCode を返します。
- public org.omg.CORBA.IDLType **discriminator_type_def**( )
  このメソッドは，union のディスクリミネータの IDL 型を返します。
- public void **discriminator_type_def**(
  org.omg.CORBA.IDLType **discriminator_type_def**)
  このメソッドは，union のディスクリミネータの IDL 型を設定します。
  - discriminator_type_def
    ディスクリミネータの IDL 型
- public org.omg.CORBA.UnionMember[ ] **members**( )
  このメソッドは，union のメンバ一覧を返します。
- public void **members**(
  org.omg.CORBA.UnionMember[ ] **members**)
  このメソッドは，union のメンバ一覧を設定します。
  - members
    メンバ一覧

6.43 UnionMember

```
public final class org.omg.CORBA.UnionMember extends
    java.lang.Object implements org.omg.CORBA.portable.IDLEntity
```

このクラスは，インタフェースリポジトリに格納されている Union を記述します。

このクラスには，Helper クラスと Holder クラスもあります。これらのクラスとそのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

6.43.1 UnionMember の変数

- public java.olang.String **name**
  union メンバの名前です。
- public org.omg.CORBA.Any **label**
  メンバに対応づけられているラベルです。
- public org.omg.CORBA.Typecode **type**
  union の TypeCode です。
- public org.omg.CORBA.IDLType **type_def**
  union の IDL 型です。

6.43.2 UnionMember のメソッド

- public **UnionMember**( )
  このメソッドは，UnionMember のデフォルトコンストラクタです。
- public **UnionMember**(
  java.lang.String **name**,
  org.omg.CORBA.Any **label**,
  org.omg.CORBA.TypeCode **type**,
  org.omg.CORBA.IDLType **type_def**)
  このメソッドは，指定されたパラメタを使用して，UnionMember を構成します。
  - name
    この union の名前
  - label
    この union のラベル
  - type

Union の TypeCode

- type_def
  Union の IDL 型

6.44 ValueBoxDef

```
public interface org.omg.CORBA.ValueBoxDef extends
    org.omg.CORBA.ValueBoxDefOperations,
    org.omg.CORBA.Contained,
    org.omg.CORBA.IDLType,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，任意の IDL 型の公開メンバを一つ格納する簡易な valuetype として使用します。ValueBoxDef インタフェースは，次の valuetype を簡略化したものです。

```
value type name
    public <IDLType> value;
```

この宣言はボックス型の<IDLType>とほとんど同じですが，ValueBoxDef インタフェースは簡易な ValueTypeDef インタフェースとは異なります。

6.44.1 ValueBoxDef のメソッド

- public org.omg.CORBA.IDLType original_type_def( )
  このメソッドは，ボックス化されている型を識別します。
- public void original_type_def(
  org,omg.CORBA.IDLType original_type_def)
  このメソッドは，ボックス化する型を設定します。

6.45 ValueDef

```
public interface org.omg.CORBA.ValueDef extends
    org.omg.CORBA.ValueDefOperations,
    org.omg.CORBA.Container,
    org.omg.CORBA.Contained,
    org.omg.CORBA.IDLType,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，construct という IDL 値を記述します。このインタフェースが格納できるのは，定数，型定義，例外，オペレーション，および属性です。このインタフェースは，クラス型とよく似ていて，インタフェースリポジトリに格納されている値定義を表します。追加情報については，「6.18 FullValueDescription」を参照してください。

6.45.1 ValueDef のメソッド

- public org.omg.CORBA.Interface[ ] supported_interfaces( )
  このメソッドは，この valuetype がサポートするインタフェースの一覧を返します。
- public void supported_interfaces(
  org.omg.CORBA.interfaceDef[ ] supported_interfaces)
  このメソッドは，サポートするインタフェースを設定します。
- public org.omg.CORBA.Initializer[ ] initializers( )
  このメソッドは，イニシャライザの一覧を返します。
- public void initializers(
  org.omg.CORBA.Initializer[ ] initializers)
  このメソッドは，イニシャライザを設定します。
- public org.omg.CORBA.ValueDef base_value( )
  このメソッドは，この値の継承元 valuetype を定義します。
- public void base_value(
  org.omg.CORBA.ValueDef base_value)
  このメソッドは，この valuetype の継承元の valuetype を設定します。
- public org.omg.CORBA.ValueDef[ ] abstract_base_values( )
  このメソッドは，この値の継承元 abstract 型 valuetype の一覧を返します。
- public void abstract_base_values(
  org.omg.CORBA.ValueDef[ ] abstract_base_values)
  このメソッドは，ベースの abstract 型 valuetype の一覧を定義します。
- public boolean is_abstract( )

true を設定した場合，abstract 型の valuetype を返します。

- public void **is_abstract**(
  boolean **is_abstract**)
  このメソッドは，valuetype を abstract 型の valuetype に設定します。
- public boolean **is_custom**( )
  true を設定した場合，この値は custom 型マーシャル処理を使用します。
- public void **is_custom**(
  boolean **is_custom**)
  このメソッドは，値に対して custom のマーシャリングを実行することを指定します。
- public boolean **is_truncatable**( )
  true を設定した場合，値を継承元から安全にマッピングできます。
- public void **is_truncatable**(
  boolean **is_truncatable**)
  このメソッドは，この値に短縮属性を設定します。
- public boolean **is_a**(
  java.lang.String **value_id**)
  このメソッドの呼び出しに使用した値が，ID パラメタで定義したインタフェースまたは値と同一であるか，（直接的または間接的に）継承されたものである場合，このメソッドは true を返し，そうでない場合は false を返します。
- public org.omg.CORBA.ValueDefPackage.FullValueDescription
  describe_value( )
  このメソッドは，値に対応する FullValueDescription オブジェクトを（オペレーションと属性を含めて）返します。
- public org.omg.CORBA.ValueMemberDef **create_value_member**(
  java.lang.String **id**,
  java.lang.String **name**,
  java.lang.String **version**,
  org.omg.CORBA.IDLtype **type_def**,
  short access)
  このメソッドは，このメソッドの呼び出し対象の ValueDef オブジェクトが格納する，新しい ValueMemberDef オブジェクトを返します。
  - id
    型のリポジトリ ID
  - name
    型の名前

- version
  オブジェクトのバージョン
- type_def
  値の IDL 型
- short access
  アクセス値

- public org.omg.CORBA.AttributeDef **create_attribute**(
  java.lang.String **id**,
  java.lang.String **name**,
  java.lang.String **version**,
  org.omg.CORBA.IDLType **type**,
  org.omg.CORBA.AttributeMode **mode**)
  このメソッドはこの valuetype に新規属性定義を生成し，その定義に対応する AttributeDef オブジェクトを返します。
  - id
    この属性のリポジトリ ID
  - name
    型の名前
  - version
    オブジェクトのバージョン
  - type
    型の IDL 型
  - mode
    オブジェクトのモード
- public org.omg.CORBA.OperationDef **create_operation**(
  java.lang.String **id**,
  java.lang.String **name**,
  java.lang.String **version**,
  org.omg.CORBA.IDLtype **result**,
  org.omg.CORBA.OpeartionMode **mode**,
  org.omg.CORBA.ParameterDescription[ ] **params**,
  org.omg.CORBA.ExceptionDef[ ] **exceptions**,
  java.lang.String[ ] **contexts**)
  このメソッドは，この valuetype の新規オペレーションを生成し，対応する OperationDef オブジェクトを返します。

- id
  オペレーションのリポジトリ ID
- name
  型の名前
- version
  オブジェクトのバージョン
- result
  オペレーションの IDL 型
- mode
  オブジェクトのモード
- params
  オペレーションのパラメタの一覧
- exceptions
  オペレーションの例外の一覧
- contexts
  オペレーションのコンテキストの一覧

6.46 ValueDescription

```
public final class org.omg.CORBA.ValueDescription extends
    java.lang.Object implements org.omg.CORBA.portable.IDLEntity
```

このクラスは，インタフェースリポジトリが格納する valuetype の定義を表すために使用します。

6.46.1 ValueDescription の変数

- public java.lang.String **name**
  ValueDescription の名前です。
- public java.lang.String **id**
  ValueDescription のリポジトリ ID です。
- public boolean **is_abstract**
  true を設定した場合，ValueDescription は abstract 型の valuetype です。
- public boolean **is_custom**
  true を設定した場合，ValueDescription に custom 型マーシャル処理が実行されます。
- public java.lang.String **defined_in**
  ValueDescription が定義されているモジュールまたはインタフェースのリポジトリ ID を表します。
- public java.lang.String **version**
  ValueDescription のバージョンです。
- public java.lang.String[ ] **supported_interfaces**
  この ValueDescription がサポートするインタフェースの一覧です。
- public java.lang.String[ ] **abstract_base_values**
  この ValueDescription の継承元 abstract 型の valuetype の一覧です。
- public boolean **is_truncatable**
  この ValueDescription が継承元の valuetype に安全にマッピングできるかどうかを示す valuetype の設定です。
- public java.lang.String **base_value**
  この ValueDescription の継承元 valuetype です。

6.46.2 ValueDescription のメソッド

- public **ValueDescription**(
  java.lang.String **name**,

java.lang.String id,
boolean is_abstract,
boolean is_custom,
java.lang.String defined_in,
java.lang.String version,
java.lang.String supported_interfaces,
java.lang.String abstract_base_values,
boolean is_truncatable,
java.lang.String base_values)

このメソッドは，指定したパラメタを使用して AttributeDescription オブジェクトを生成します。

- name
  この ValueDescription の名前
- id
  この ValueDescription のリポジトリ ID
- is_abstract
  true を設定した場合，valuetype は abstract です。
- is_custom
  true を設定した場合，ValueDescription は custom 型マーシャル処理を使用します。
- defined_in
  この ValueDescription が定義されているモジュール
- version
  ValueDescription のバージョン
- supported_interfaces
  サポートするインタフェース
- abstract_base_values
  サポートする abstract ベース値
- is_truncatable
  true を設定した場合，ValueDescription はベース値に安全にマッピングできます。
- base_values
  ベース値

6.47 ValueMemberDef

```
public interface org.omg.CORBA.ValueMemberDef extends
    org.omg.CORBA.ValueMemberDefOperations,
    org.omg.CORBA.Contained,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，インタフェースリポジトリが格納する値のメンバ定義を表すために使用します。

6.47.1 ValueMemberDef のメソッド

- public org.omg.CORBA.TypeCode **type**( )
  このメソッドは，値メンバの IDL 型を返します。
- public org.omg.CORBA.IDLType **type_def**( )
  このメソッドは，IDL 型の定義を表します。
- public void **type_def**(
  org.omg.CORBA.IDLType **type_def**)
  このメソッドは，値メンバに対する IDL 型を設定します。
- public short **access**( )
  このメソッドは，オブジェクトのアクセス値を定義します。
- public void **access**(
  short **access**)
  このメソッドは，値メンバに対するアクセス値を設定します。

6.48 WstringDef

```
public interface org.omg.CORBA.WstringDef extends
    org.omg.CORBA.WstringDefOperations,
    org.omg.CORBA.IDLType,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，インタフェースリポジトリに格納されている Unicode 文字列を表すために使用します。

このインタフェースの Helper クラスと Holder クラス，およびこれらのクラスのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

7 活性化インタフェースとクラス（Java）

この章では，Java 言語の Activation パッケージが格納しているインタフェースとクラスについて説明します。説明するのは ActivationImplDef，Activator，CreationImplDef，ImplementationDef，および OAD です。これらは動的インタフェースと動的クラスでありオブジェクト活性化デーモン（OAD）とともに使用されます。

7.1 ActivationImplDef

```
valuetype ActivationImplDef
```

ActivationImplDef 型は，Activator 用の属性の集合を提供します。

```
valuetype ActivationImplDef :ImplementationDef {
    attribute string service_name;
    attribute CORBA::ReferenceData id;
    attribute extension::Activator activator_obj;
};
```

7.1.1 ActivationImplDef のメソッド

- public abstract Activator activator_obj( )
  このメソッドは，Activator の管理下でオブジェクトインプリメンテーションのオブジェクトリファレンスを取得します。
- public abstract byte[ ] id( )
  このメソッドは，インプリメンテーションのリファレンスデータ識別子を取得します。
- public abstract String service_name( )
  このメソッドは，インプリメンテーションのサービス名を取得します。

7.2 Activator

```
public interface Activator
```

オブジェクトインプリメンテーションを設計する場合，クライアントからリクエストがあるまで，VisiBroker ORB オブジェクトの活性化を延期することがあります。オブジェクトの活性化を延期すると性能が向上します。多くのオブジェクトがサーバにある場合，クライアントからオブジェクトのリクエストがあったときだけ，オブジェクトを活性化させることで，システム資源を節約できます。一つの Activator を使用して，複数のオブジェクトインプリメンテーションの活性化を延期できます。

```
interface Activator {
    Object activate(in CORBA::ImplementationDef impl);
    void   deactivate(in Object obj,
                      in CORBA::ImplementationDef impl);
};
```

7.2.1 Activator のメソッド

- public org.omg.Object **activate**(
  org.omg.CORBA.ImplementationDef **impl**)
  このメソッドを使用して，Activator が管理しているオブジェクトインプリメンテーションを活性化します。Activator の管理下で，VisiBroker ORB が一つのオブジェクトに対するクライアント要求を受信すると，VisiBroker ORB は Activator 上で activate( )メソッドを呼び出します。このメソッドでは，Activator に ImplementationDef パラメタを渡します。これで，VisiBroker ORB は活性化されたオブジェクトインプリメンテーションを一意に識別します。ここから，インプリメンテーションは一意の識別子である ref_data を取得できます。
  - impl
    ImplementationDef のインスタンスです。
- public void **deactivate**(
  org.omg. CORBA.Object **obj**,
  org.omg.CORBA.ImplementationDef **impl**)
  このメソッドは，Activator の管理下でオブジェクトインプリメンテーションを非活性化させます。このメソッドでは，Activator にオブジェクトリファレンスと ImplementationDef パラメタを渡します。これによって，VisiBroker ORB は非活性化対象のオブジェクトインプリメンテーションを一意に識別します。ここから，インプリメンテーションは一意の識別子である ref_data を取得できます。たくさんのオブジェクトを使用したインプリメンテーションの場合，オブジェクトのキャッシュに負荷が掛かったとき，deactivate( )を使用して状態データを処理できます。
  - obj
    非活性化対象のオブジェクトのオブジェクトリファレンスです。

- impl
  ImplementationDef のインスタンスです。

7.3 CreationImplDef

```
struct CreationImplDef
```

CreationImplDef は，ある特定のオブジェクトインプリメンテーション用の属性の集合を提供する IDL 構造体です。これらの属性の値の問い合わせと，設定メソッドは，この構造体で提供します。該当する属性は，_args，_env，id（リファレンスデータ用），object_name，_path_name，_policy，および repository_id です。

オブジェクト活性化デーモンは，CreationImplDef を使用してオブジェクトインプリメンテーションを一覧表示，登録，および登録解除します。コマンドラインパラメタは oadutil を使用したときに指定され，CreationImplDef で定義した属性の設定に使用します。

7.3.1 IDL の定義

```
struct CreationImplDef
    CORBA::RepositoryId repository_id;
    string object_name;
    CORBA::ReferenceData id;
    string path_name;
    CORBA::Policy activation_policy;
    CORBA::StringSequence args;
    CORBA::StringSequence env;
};
```

7.3.2 活性化ポリシー

クライアントリクエストに対してサーバを活性化する場合，CreationImplDef の値を OAD がどのように使用するかを次に示します。

CreationImplDef は，サーバの活性化ポリシーの設定メソッドを提供します。これらの活性化ポリシーは，パーシステントオブジェクトに適用できます。トランジェントオブジェクトには適用できません。活性化ポリシーを次に示します。

- SHARED_SERVER
  複数のクライアントが，同じインプリメンテーションを共用します。一度に OAD によって活性化されるサーバは一つだけです。
- UNSHARED_SERVER
  クライアントの一つだけが，活性化されたサーバにバウンドできます。複数のクライアントが同じオブジェクトインプリメンテーションにバインドしたい場合，個別のサーバがそれぞれのクライアントのために活性化されます。クライアントアプリケーションの切断，終了時に，サーバは終了します。
- SERVER_PER_METHOD

各メソッドの呼び出しで，新しいサーバが活性化されます。メソッド呼び出しが完了すると，サーバは終了します。

7.3.3 例

OAD 上の，CreationImplDef 属性から実行コマンドへの変換例を次に示します。

(1) Java の例

パラメタ CreditUnion とシステムプロパティ DEBUG を 1 に設定し，com.mycompany.Server と呼ばれる Java アプリケーション用に Borland Enterprise Server VisiBroker を起動する場合，次の属性で CreationImplDef を記述します。

```
path_name = "vbj"
args = ["com.mycompany.Server", "CreditUnion"]
env = ["DEBUG=1"]
```

上記の属性から，次のコマンドを OAD に生成させます。

```
"vbj -DOAoad_uid=<uid> -DOAactivateIOR=<OAD's ior>          ¥
     -DDEBUG=1                                              ¥
   com.mycompany.Server CreditUnion"
```

さらに，次の環境変数を OAD の環境から，生成された「vbj」実行環境に引き継ぎます。

- PATH
- CLASSPATH
- OSAGENT_PORT
- OSAGENT_ADDR
- VBROKER_ADM
- OSAGENT_ADDR_FILE
- OSAGENT_LOCAL_FILE
- TPDIR
- SHLIB_PATH（HP-UX の場合）
- LD_LIBRARY_PATH（Solaris および Linux の場合）
- LIBPATH（AIX の場合）

7.3.4　環境変数

vbj コマンドを実行して登録済み Java クラスを活性化した場合，OAD の環境は生成されたプロセスに自動的に渡されません。一部の環境変数を設定すると，これらの環境変数は OAD によって明示的に渡されます。該当する環境変数については，「7.3.5　明示的に渡される環境変数と暗黙的に渡される環境変数」を参照してください。

そのほかの環境変数は，CreationImplDef の env 属性を使用して登録する必要があります。

活性化された Java インプリメンテーションの場合，CreationImplDef の env 属性に記録されているような環境設定は，次の二つの方法でプロパゲーションされます。

- 生成された vbj コマンドの環境内でのプロパゲーション
- クラスへのシステムプロパティとして（Java VM への-D パラメタ）のプロパゲーション

したがって，生成された Java アプリケーションの場合，登録は次の実行済みコマンドにマッピングします。

```
vbj -DOAoad_uid=<uid> -DOAactivateIOR=<oad's ior>
   { -Denv1 ... -DenvN }
   className { args1 ... argsN }
```

その結果，生成された環境には，インプリメンテーション定義から指定したすべての環境変数が含まれます。また，システムを起動したときに OAD 自身の環境から取った次の環境変数の定義も含まれます。

- PATH
- CLASSPATH
- OSAGENT_PORT
- OSAGENT_ADDR
- VBROKER_ADM
- OSAGENT_ADDR_FILE
- OSAGENT_LOCAL_FILE
- TPDIR
- SHLIB_PATH（HP-UX の場合）
- LD_LIBRARY_PATH（Solaris および Linux の場合）
- LIBPATH（AIX の場合）

7.3.5　明示的に渡される環境変数と暗黙的に渡される環境変数

次に示す環境変数は，OAD の環境から生成されたサーバの環境にプロパゲーションされます。また，設定がある場合は OAD で明示的に渡されます。

- PATH
- CLASSPATH
- OSAGENT_PORT
- OSAGENT_ADDR
- VBROKER_ADM
- OSAGENT_ADDR_FILE
- OSAGENT_LOCAL_FILE
- TPDIR
- SHLIB_PATH（HP-UX の場合）
- LD_LIBRARY_PATH（Solaris および Linux の場合）
- LIBPATH（AIX の場合）
- OSAGENT_CLIENT_HANDLER_PORT（CreationImplDef 内に設定）

例えば，Java インプリメンテーションを生成する場合，生成された実行対象に OSAGENT_CLIENT_HANDLER_PORT が必要なときは，CreationImplDef 環境に OSAGENT_CLIENT_HANDLER_PORT を明示的に登録する必要があります。

7.3.6 CreationImplDef のメソッド

- public abstract org.omg.CORBA.Policy **activation_policy**()
  このメソッドは，サーバの活性化ポリシーを取得します。
- public abstract void **activation_policy**(
  org.omg.CORBA.Policy **activation_policy**)
  このメソッドは，サーバの活性化ポリシーを設定します。活性化ポリシーには，SHARED_SERVER，UNSHARED_SERVER，および SERVER_PER_METHOD があります。
  - activation_policy
    サーバの活性化ポリシー
- public abstract String[ ] **args**()
  このメソッドは，サーバに渡された引数のリストを取得します。
- public abstract void **args**(
  String **args**[ ])
  このメソッドは，サーバに渡すコマンドライン引数を設定します。最初の引数には必ずクラス名を指定してください。詳細については，「7.3.3　例」を参照してください。
  - args
    コマンドライン引数をすべて指定する文字列の並び

- public abstract String[ ] **env**()
  このメソッドは，サーバに渡した環境設定のリストを取得します。
- public abstract void **env**(
  String **env**[ ])
  このメソッドは，サーバに渡す環境設定を設定します。env 属性の設定の詳細については，「7.3.4　環境変数」を参照してください。
  - env
    環境変数をすべて指定する文字列の並び
- public abstract byte[ ] **id**()
  このメソッドは，インプリメンテーションのリファレンスデータ識別子を取得します。
- public abstract void **id**(
  byte **id**[ ])
  このメソッドは，インプリメンテーションのリファレンスデータ識別子を設定します。
  - id
    インプリメンテーションのリファレンスデータ識別子（バイトの並び）
- public abstract String **object_name**()
  このメソッドは，インプリメンテーションのオブジェクト名を取得します。
- public abstract void **object_name**(
  String **object_name**)
  このメソッドは，インプリメンテーションのオブジェクト名を設定します。
  - object_name
    インプリメンテーションのオブジェクト名を指定する文字列
- public abstract String **path_name**()
  登録の場合，このメソッドは，文字列「vbj」を取得します。
- public abstract void **path_name**(
  String **path_name**)
  このメソッドは，オブジェクトをインプリメントする実行可能プログラムの正確なパス名を設定します。プログラムの場合，パス名は必ず「vbj」にしてください。

  注
  OAD のパス設定環境変数を設定する場合は，必ずプログラム「vbj」にパスが通るように設定してください。OAD のパスはインストール時に設定します。
  - path_name
    インプリメンテーションのパス名を指定する文字列
- public abstract String **repository_id**()
  このメソッドは，インプリメンテーションのリポジトリ ID を取得します。

- public abstract void **repository_id**(
  String **repository_id**)
  このメソッドは，インプリメンテーションのリポジトリ ID を設定します。
  - repository_id
    インプリメンテーションのリポジトリ ID を指定する文字列

7.4 ImplementationDef

ImplementationDef は，ImplementationDefs の型である ActivationImplDef と CreationImplDef 用の空のベースクラスです。ImplementationDef が使用できるのは，ActivationImplDef メソッド，または CreationImplDef メソッド内のシグニチャの中だけです。

7.5 OAD

```
public interface OAD extends org.omg.CORBA.Object
```

OAD インタフェースは，OAD へのアクセスを提供します。このインタフェースは，管理ツールがオブジェクトの一覧表示，登録，および登録解除を実行するために使用します。また，OAD をプログラムから管理するために，クライアントコードがこのインタフェースを使用することもできます。

```
interface OAD {
CreationImplDef create_CreationImplDef( );

Object reg_implementation(in extension::CreationImplDef impl)
    raises(DuplicateEntry,InvalidPath);

CreationImplDef get_implementation(
                in CORBA::RepositoryId repId,
                in string object_name)
    raises(NotRegistered);

void change_implementation(
                in extension::CreationImplDef old_info,
                in extension::CreationImplDef new_info)
    raises(NotRegistered,InvalidPath,IsActive);

attribute boolean destroy_on_unregister;

void unreg_implementation(in CORBA::RepositoryId repId,
                in string object_name)
    raises(NotRegistered);

void unreg_interface(in CORBA::RepositoryId repId)
    raises(NotRegistered);

void unregister_all( );

ImplementationStatus get_status(in CORBA::RepositoryId repId,
                in string object_name)
    raises(NotRegistered);

ImplStatusList get_status_interface(
                in CORBA::RepositoryId repId)
    raises(NotRegistered);

ImplStatusList get_status_all( );

Object lookup_interface(in CORBA::RepositoryId repId,
                in long timeout)
    raises(NotRegistered,FailedToExecute,NotResponding,Busy);

Object lookup_implementation(in CORBA::RepositoryId repId,
                in string object_name,in long timeout)
    raises(NotRegistered,FailedToExecute,NotResponding,Busy);

string generated_command(in extension::CreationImplDef impl);
```

```
string generated_environment(
                          inextension::CreationImplDef impl);
};
```

7.5.1 ImplementationStatus

ImplementationStatus は，CreationImplDef からの impl と，ObjectStatusList からのステータスを含んだ構造体です。ObjectStatusList は，long 型に unique_id を指定し，State には activation_state を指定している構造体です。インプリメンテーションは，次の状態のどれかにできます。

- 活性
- 非活性
- 活性化待ち

```
module Activation
{
...
   struct ObjectStatus {
      long     unique_id;
      State    activation_state;
      Object   objRef;
   };
   typedef sequence<ObjectStatus>ObjectStatusList;
   struct ImplementationStatus {
      extension::CreationImplDef impl;
      ObjectStatusList           status;
   };
```

7.5.2 OAD のメソッド

- public void change_implementation(
  org.omg.CORBA.CreationImplDef old_info,
  org.omg.CORBA.CreationImplDef new_info)
  このメソッドは，オブジェクトのインプリメンテーションを動的に変更します。このメソッドを使用して，登録の活性化ポリシー，パス名，パラメタ設定，および環境設定を変更できます。
  - old_info
    変更したい情報
  - new_info
    old_info と差し替えたい情報

  このメソッドでは，次の例外が発生します。

NotRegistered

指定したオブジェクトは未登録です。登録済みオブジェクトを指定してください。

InvalidPath

Java クラス，または ccp が実行できるプログラムが見つかりません。

IsActive

オブジェクトインプリメンテーションは現在実行中です。オブジェクトを非活性化してから，その情報を変更してください。

注

現在活性状態にあるインプリメンテーションの情報は変更できません。このメソッドでオブジェクトのインプリメンテーション名やオブジェクト名を変更する場合，必ず注意を守ってください。クライアントアプリケーションはオブジェクトを古い名前で検索できません。

- public abstract CreationImplDef **create_CreationImplDef**( )

  このメソッドは，CreationImplDef のインスタンスを生成します。その後，属性を設定できます。詳細については，「7.3　CreationImplDef」を参照してください。

  このメソッドでは，次の例外が発生します。

  DuplicateEntry

  指定したオブジェクトはすでに登録されています。未登録のオブジェクトを指定してください。

  InvalidPath

  Java クラスが見つかりません。

- public abstract void **destroy_on_unregister**(
  boolean **destroy_on_unregister**)

  このメソッドは，OAD の destroy_on_unregister 属性を設定します。この属性に true を設定した場合，活性状態のインプリメンテーションは，登録解除時にすべてシャットダウンされます。

- public abstract boolean **destroy_on_unregister**( )

  このメソッドは，インプリメンテーションの destroy_on_unregister 属性の設定を取得します。

- public abstract String **generated_command**(
  org.omg.CORBA.CreationImplDef **impl**)

  このメソッドは，指定したインプリメンテーションに対して実行するコマンドのコマンドラインオプションを表す文字列を返します。

- public abstract String **generated_environment**(
  com.Inprise.vbroker.extension.CreationImplDef **impl**)

  このメソッドは，指定したインプリメンテーションについて生成したサーバを実行する，環境を表す文字列を返します。

- public org.omg.CORBA.CreationImplDef **get_implementation**(
  String **repository_id**, String **object_name**)

このメソッドは，指定されたリポジトリ ID とオブジェクト名に対して登録されたインプリメンテーションに関する情報を検索します。

- repository_id
  リポジトリ ID を指定する文字列
- object_name
  オブジェクト名を指定する文字列

このメソッドでは，次の例外が発生します。

NotRegistered
  指定したオブジェクトは未登録です。登録済みオブジェクトを指定してください。

- public com.inprise.vbroker.Activation.ImplementationStatus
  get_status(String **repository_id**, String **object_name**)
  このメソッドは，指定されたリポジトリ ID とオブジェクト名に対して登録されたインプリメンテーションに関するステータス情報を検索します。
  - repository_id
    リポジトリ ID を指定する文字列
  - object_name
    オブジェクト名を指定する文字列
- public Activation.ImplementationStatus[ ] **get_status_all**()
  このメソッドは，すべてのインプリメンテーションに関するステータス情報を取得します。
- public Activation.ImplementationStatus[ ] **get_status_interface**(
  String **repository_id**)
  このメソッドは，指定したリポジトリに登録されているインプリメンテーションのステータス情報を取得します。
  - repository_id
    リポジトリ ID を指定する文字列
- public Activation ImplStatusList **get_status_interface**(
  String **repository_id**)
  このメソッドは，指定したリポジトリ ID のインプリメンテーションのオブジェクトリファレンスを返します。
  - repository_id
    リポジトリ ID を指定する文字列

  このメソッドでは，次の例外が発生します。

  NotRegistered
    指定したオブジェクトは未登録です。登録済みオブジェクトを指定してください。

FailedToExecute

指定したオブジェクトは実行できません。実行中にエラーが発生しました。

NotResponding

メソッドへの応答がありません。

Busy

指定したオブジェクトは現在使用中です。

- public Activation ImplStatusList **get_status_all**( )

このメソッドは，登録されているすべてのインプリメンテーションのステータス情報を返します。

- public org.omg.CORBA.Object **lookup_interface**(
String **repository_id**,long **timeout**)

このメソッドは，指定したインプリメンテーションを検索します。

- repository_id
  リポジトリ ID を指定する文字列
- timeout

このメソッドでは次の例外が発生します。

NotRegistered

指定したオブジェクトは未登録です。登録済みオブジェクトを指定してください。

FailedToExecute

指定したオブジェクトは実行できません。実行中にエラーが発生しました。

NotResponding

メソッドへの応答がありません。

Busy

指定したオブジェクトは現在使用中です。

- public org.omg.CORBA.Object **lookup_implementation**(
String **repID**, string **object_name**, long **timeout**)

このメソッドは，指定したインプリメンテーションを検索します。クライアントが直接呼び出す必要はありません。

- repID
  リポジトリ ID を指定する文字列
- object_name
  オブジェクト名を指定する文字列
- timeout

このメソッドでは次の例外が発生します。

NotRegistered
　指定したオブジェクトは未登録です。登録済みオブジェクトを指定してください。

FailedToExecute
　指定したオブジェクトは実行できません。実行中にエラーが発生しました。

NotResponding
　メソッドへの応答がありません。

Busy
　指定したオブジェクトは現在使用中です。

- public org.omg.CORBA.Object **reg_implementation**(
  org.omg.CORBA.CreationImplDef **impl**)
  このメソッドは，インプリメンテーションを OAD と Borland Enterprise Server VisiBroker ディレクトリサービスに登録します。
  - impl
    CreationImplDef のインスタンス

  このメソッドでは，次の例外が発生します。

  DuplicateEntry
  　指定したオブジェクトは重複エントリです。未登録オブジェクトを指定してください。

  InvalidPath
  　Java クラスが見つかりません。
- public void **unreg_implementation**(
  String **repository_id**,
  String **object_name**)
  このメソッドは，リポジトリ ID とオブジェクト名でインプリメンテーションを登録解除します。destroy_on_unregister 属性に true を設定した場合，このメソッドは，指定されたリポジトリ ID およびオブジェクト名を現在インプリメントしている，すべてのプロセスを終了させます。
  - repository_id
    リポジトリ ID を指定する文字列
  - object_name
    オブジェクト名を指定する文字列

  このメソッドでは，次の例外が発生します。

  NotRegistered
  　指定したオブジェクトは未登録です。登録済みオブジェクトを指定してください。
- public void **unreg_interface**(
  String **repository_id**)

このメソッドは，リポジトリ ID に対応するすべてのインプリメンテーションを登録解除します。destroy_on_unregister 属性に true を設定した場合，このメソッドは，指定されたリポジトリ ID を現在インプリメントしている，すべてのプロセスを終了させます。

- repository_id
  リポジトリ ID を指定する文字列です。

このメソッドでは，次の例外が発生します。

NotRegistered
  指定したオブジェクトは未登録です。登録済みオブジェクトを指定してください。

- public void **unregister_all**( )
  このメソッドは，すべてのインプリメンテーションを登録解除します。destroy_on_unregister 属性に true を設定しないかぎり，活性状態のすべてのインプリメンテーションが実行を続けます。

8 ネーミングサービスインタフェースとクラス（Java）

この章では，Java 言語でプログラミングする場合に，Borland Enterprise Server VisiBroker のネーミングサービスで使用するインタフェースとクラスについて説明します。Borland Enterprise Server VisiBroker ネーミングサービスは，OMG の CORBA 2.5 の仕様に従って実装されています。

8.1 NamingContext

```
public interface NamingContext extends
    com.inprise.vbroker.CORBA.Object
```

このインタフェースを使用して，VisiBroker ORB オブジェクトまたはほかの NamingContext オブジェクトにバインドされるネームを登録したり，操作したりします。クライアントアプリケーションは，このインタフェースを使用して，コンテキスト内のすべてのネームを resolve または list によって処理します。オブジェクトインプリメンテーションは，このオブジェクトを使用して，オブジェクトインプリメンテーションまたは NamingContext オブジェクトをネームにバインドします。NamingContext の IDL 仕様を次に示します。

8.1.1 IDL の定義

```
module CosNaming {

    interface NamingContext {
        void bind(in Name n, in Object obj)
            raises(NotFound, CannotProceed, InvalidName,
                   AlreadyBound);
        void rebind(in Name n, in Object obj)
            raises(NotFound, CannotProceed, InvalidName);
        void bind_context(in Name n, in NamingContext nc)
            raises(NotFound, CannotProceed, InvalidName,
                   AlreadyBound);
        void rebind_context(in Name n, in NamingContext nc)
            raises(NotFound, CannotProceed, InvalidName);
        Object resolve(in Name n)
            raises(NotFound, CannotProceed, InvalidName);
        void unbind(in Name n)
            raises(NotFound, CannotProceed, InvalidName);
        NamingContext new_context( );
        NamingContext bind_new_context(in Name n)
            raises(NotFound, CannotProceed, InvalidName,
                   AlreadyBound);
        void destroy( )
            raises(NotEmpty);
        void list(in unsigned long how_many,
            out BindingList bl,
            out BindingIterator bi);
    };
};
```

8.1.2 NamingContext のメソッド

- public void **bind**(
  CosNaming.NameComponent[ ] **n**,

org.omg.CORBA.Object **obj**)
throws
CosNaming.NamingContextPackage.**NotFound**,
CosNaming.NamingContextPackage.**CannotProceed**,
CosNaming.NamingContextPackage.**InvalidName**,
CosNaming.NamingContextPackage.**AlreadyBound**
このメソッドは，指定されたネームを指定された Object にバインドします。このとき，最初の NameComponent に関連づけられたコンテキストを解決し，そのあと，次に示す Name を使用して新しいコンテキストにオブジェクトをバインドします。

$$Name[NameComponent_{(2)},...,NameComponent_{(n-1)},NameComponent_{(n)}]$$

解決とバインドのこの再帰的なプロセスは，NameComponent (n-1)に関連づけられたコンテキストが解決され，ネームとオブジェクトとの実際のバインドが格納されるまで続きます。パラメタ n がシンプルネームである場合には，obj は，この NamingContext 内の n にバインドされます。

- n
  オブジェクトに指定するネームで初期化される Name 構造体
- obj
  ネーミングされるオブジェクト

このメソッドでは，次の例外が発生します。

- NotFound
  Name またはそのコンポーネントの一つが見つかりません。
- CannotProceed
  シーケンスの NameComponent オブジェクトの一つが解決されていません。クライアントは，返されたネーミングコンテキストからオペレーションを継続できます。
- InvalidName
  指定された Name にはネームコンポーネントがありません。または，ID フィールドに空文字列を指定したネームコンポーネントがあります。
- AlreadyBound
  ネームが NamingContext 内の別のオブジェクトにすでにバインドされています。

- public void **rebind**(
  CosNaming.NameComponent[ ] **n**,
  org.omg.CORBA.Object **obj**)
  throws
  CosNaming.NamingContextPackage.**NotFound**,
  CosNaming.NamingContextPackage.**CannotProceed**,
  CosNaming.NamingContextPackage.**InvalidName**

このメソッドは，AlreadyBound 例外が発行されないという点を除いて，bind メソッドと同じです。指定された Name がすでに別のオブジェクトにバインドされている場合には，このバインドは，新しいバインドで置き換えられます。

- n
  オブジェクトに指定するネームで初期化される Name 構造体
- obj
  ネーミングされるオブジェクト

このメソッドでは，次の例外が発生します。

- NotFound
  Name またはそのコンポーネントの一つが見つかりません。
- CannotProceed
  シーケンスの NameComponent オブジェクトの一つが解決されていません。クライアントは，返されたネーミングコンテキストからオペレーションを継続できます。
- InvalidName
  指定された Name にはネームコンポーネントがありません。または，ID フィールドに空文字列を指定したネームコンポーネントがあります。

- public void **bind_context**(
  CosNaming NameComponent[ ] **n**,
  CosNaming.NamingContext **nc**)
  throws
  CosNaming.NamingContextPackage.**NotFound**,
  CosNaming.NamingContextPackage.**CannotProceed**,
  InvalidName,
  CosNaming.NamingContextPackage.**AlreadyBound**
  このメソッドは，指定された Name が，任意の ORB オブジェクトではなく，NamingContext に対応づけられるという点を除いて，bind メソッドと同じです。
  - n
    希望するネーミングコンテキスト名で初期化される Name 構造体。シーケンス内の最初の(n-1)個の NameComponent 構造体は，NamingContext に解決される必要があります。
  - nc
    リバインドされる NamingContext オブジェクト

  このメソッドでは，次の例外が発生します。

  - NotFound
    Name またはそのコンポーネントの一つが見つかりません。
  - CannotProceed

シーケンスの NameComponent オブジェクトの一つが解決されていません。クライアントは，返されたネーミングコンテキストからオペレーションを継続できます。

- InvalidName
  指定された Name にはネームコンポーネントがありません。または，ID フィールドに空文字列を指定したネームコンポーネントがあります。
- AlreadyBound
  ネームが NamingContext 内の別のオブジェクトにすでにバインドされています。

- public void **rebind_context**(
  CosNaming. NameComponent[ ] **n**,
  CosNaming.NamingContext **nc**)
  throws
  CosNaming.NamingContextPackage.**NotFound**,
  CosNaming.NamingContextPackage.**CannotProceed**,
  CosNaming.NamingContextPackage.**InvalidName**
  このメソッドは，AlreadyBound 例外が発行されないという点を除いて，bind_context メソッドと同じです。指定された Name がすでに別のネーミングコンテキストにバインドされている場合には，このバインドは，新しいバインドで置き換えられます。
  - n
    オブジェクトに指定するネームで初期化される Name 構造体
  - nc
    リバインドされる NamingContext オブジェクト

  このメソッドでは，次の例外が発生します。
  - NotFound
    Name またはそのコンポーネントの一つが見つかりません。
  - CannotProceed
    シーケンスの NameComponent オブジェクトの一つが解決されていません。クライアントは，返されたネーミングコンテキストからオペレーションを継続できます。
  - InvalidName
    指定された Name にはネームコンポーネントがありません。または，ID フィールドに空文字列を指定したネームコンポーネントがあります。
- public org.omg.CORBA.Object **resolve**(
  CosNaming.NameComponent[ ] **n**)
  throws
  CosNaming.NamingContextPackage.**NotFound**,
  CosNaming.NamingContextPackage.**CannotProceed**,
  CosNaming.NamingContextPackage.**InvalidName**

このメソッドは，指定された Name を解決し，オブジェクトリファレンスを返します。パラメタ n がシンプルネームである場合には，この NamingContext で解決されます。

n がコンプレックスネームである場合には，最初の NameComponent に関連づけられたコンテキストを使用して解決されます。その後，新しいコンテキストを使用して次に示す Name を解決します。

$$Name[NameComponent_{(2)} ,...,NameComponent_{(n-1)} ,NameComponent_{(n)}]$$

この再帰的なプロセスは，n 番目の NameComponent に関連づけられたオブジェクトが返されるまで続きます。

- n
  対象となるオブジェクトのネームで初期化される Name 構造体

このメソッドでは，次の例外が発生します。

- NotFound
  Name またはそのコンポーネントの一つが見つかりません。
- CannotProceed
  シーケンスの NameComponent オブジェクトの一つが解決されていません。クライアントは，返されたネーミングコンテキストからオペレーションを継続できます。
- InvalidName
  指定された Name にはネームコンポーネントがありません。または，ID フィールドに空文字列を指定したネームコンポーネントがあります。

- public void **unbind**(
  CosNaming.NameComponent[ ] **n**)
  throws
  CosNaming.NamingContextPackage.**NotFound**,
  CosNaming.NamingContextPackage.**CannotProceed**,
  CosNaming.NamingContextPackage.**InvalidName**
  このメソッドは，bind メソッドの逆で，指定された Name に対応しているバインドを削除します。
  - n
    バインドの解除をしたいネームの Name 構造体

  このメソッドでは，次の例外が発生します。
  - NotFound
    Name またはそのコンポーネントの一つが見つかりません。
  - CannotProceed
    シーケンスの NameComponent オブジェクトの一つが解決されていません。クライアントは，返されたネーミングコンテキストからオペレーションを継続できます。
  - InvalidName

指定された Name にはネームコンポーネントがありません。または，ID フィールドに空文字列を指定したネームコンポーネントがあります。

- public CosNaming.NamingContext **new_context**( )

  このメソッドは，新しいネーミングコンテキストを作成します。新しく作成されたコンテキストは，このオブジェクトと同じサーバ内でインプリメントされます。新しいコンテキストは，初期状態ではどの Name にもバインドされていません。

- public CosNaming.NamingContext **bind_new_context**(

  NamComponent[ ] **n**)

  throws

  CosNaming.NamingContextPackage.**NotFound**,

  CosNaming.NamingContextPackage.**CannotProceed**,

  CosNaming.NamingContextPackage.**InvalidName**,

  CosNaming.NamingContextPackage.**AlreadyBound**

  このメソッドは，新しいコンテキストを作成し，そのコンテキスト内で指定された Name にバインドします。

  - n

    新しく作成された NamingContext オブジェクトに対応するネームで初期化される Name 構造体

  このメソッドでは，次の例外が発生します。

  - NotFound

    Name またはそのコンポーネントの一つが見つかりません。

  - CannotProceed

    シーケンスの NameComponent オブジェクトの一つが解決されていません。クライアントは，返されたネーミングコンテキストからオペレーションを継続できます。

  - InvalidName

    指定された Name にはネームコンポーネントがありません。または，ID フィールドに空文字列を指定したネームコンポーネントがあります。

  - AlreadyBound

    ネームが NamingContext 内の別のオブジェクトにすでにバインドされています。

- public void **destroy**( )

  throws

  CosNaming.NamingContextPackage.NotEmpty

  このメソッドは，現在のネーミングコンテキストを非活性化します。以後，このオブジェクトでオペレーションを呼び出そうとすると，CORBA.OBJECT_NOT_EXIST ランタイム例外が発生します。

  このメソッドを使用する前に，unbind メソッドを使用して，現在のネーミングコンテキストに対応してバインドされているすべての Name オブジェクトをバインド解除しておく必要があります。空でない NamingContext を破棄しようとすると，NotEmpty 例外が発生します。

NamingContext.destroy( )を発行したあとに，同じ NamingContext オブジェクトに対して，_non_existent( )を発行すると，org.omg.CORBA.OBJ_ADAPTER 例外が発生します。

- public void **list**(
  int **how_many**,
  CosNaming.BindingListHolder **bl**,
  CosNaming.BindingIteratorHolder **bi**)
  このメソッドは，現在のコンテキストに含まれているすべてのバインドを返します。how_many までの Name が，BindingList によって返されます。残りのバインドは，すべて BindingIterator によって返されます。返された BindingList と BindingIterator を使用すると，ネームのリストを参照できます。BindingList の詳細については，「8.3　Binding と BindingList」を参照してください。
  - how_many
    リストに返される Name の最大数
  - bl
    呼び出しプログラムに返される Name のリスト。リストのネームの数は how_many の値を超えません。
  - bi
    残りの Name を参照するための BindingIterator オブジェクト

8.2 NamingContextExt

```
public interface NamingContextExt extends
    CosNaming.NamingContext
```

このインタフェースは，NamingContext インタフェースを継承したもので，文字列化した名前と URL を使用するときに必要なオペレーションを提供します。

8.2.1 IDL の定義

```
module CosNaming {
    interface NamingContextExt:NamingContext{
        typedef string StringName;
        typedef string Address;
        typedef string URLString;

        StringName to_string(in Name n)
            raises(InvalidName);
        Name to_name(in StringName sn)
            raises(InvalidName);

        exception InvalidAddress {};

        URLString to_url(in Address addr,in StringName sn)
            raises(InvalidAddress,InvalidName);
        Object resolve_str(in StringName n)
            raises(NotFound,CannotProceed,
                   InvalidName);
    };
};
```

8.2.2 NamingContextExt のメソッド

- public java.lang.String to_string(
  CosNaming.NameComponent[ ] n)
  throws
  CosNaming.NamingContextPackage.InvalidName
  このオペレーションは，指定した Name の文字列化表現を返します。
  - n
    対象となるオブジェクトの名前で初期化される Name 構造体

  このメソッドで発生する可能性のある例外を次に示します。
  - InvalidName

指定された Name にはネームコンポーネントがありません。または，ID フィールドに空文字列を指定したネームコンポーネントがあります。

引数に空文字列など不正な名前を指定したネームコンポーネントを指定した場合，to_string メソッドでは InvalidName 例外は発生しません。

InvalidName 例外は，to_string メソッドの戻り値を使用して，resolve メソッドなどを呼び出した時点で発生します。

- public CosNaming.NameComponent[ ] to_name(
  java.lang.String sn)
  throws
  CosNaming.NamingContextPackage.InvalidName
  このオペレーションは，指定した文字列化された名前の Name オブジェクトを返します。
  - sn
    オブジェクトの文字列化された名前

  このメソッドで発生する可能性のある例外を次に示します。
  - InvalidName
    指定した Name にはネームコンポーネントがありません。または，ID フィールドに空文字列を指定したネームコンポーネントがあります。

    引数に空文字列など不正な名前を指定したネームコンポーネントを指定した場合，to_name メソッドでは InvalidName 例外は発生しません。

    InvalidName 例外は，to_name メソッドの戻り値を使用して，resolve メソッドなどを呼び出した時点で発生します。
- public java.lang.String to_url(
  java.lang.String addr,
  java.lang.String sn)
  throws
  CosNaming.NamingContextExtPackage.InvalidAddress,
  CosNaming.NamingContextPackage.InvalidName
  このオペレーションは，指定した URL コンポーネントと文字列化オブジェクト名から，完全な URL 文字列を返します。
  - addr
    「myhost.inprise.com:800」の形式の URL コンポーネント。このパラメタの指定を省略すると，InvalidAddress 例外が発生します。
  - sn
    文字列化したオブジェクト名

  このメソッドで発生する可能性のある例外を次に示します。
  - InvalidAddress

addr パラメタに指定したアドレスが不正です。

- InvalidName
  指定された Name にはネームコンポーネントがありません。または，ID フィールドに空文字列を指定したネームコンポーネントがあります。

- public org.omg.CORBA.Object **resolve_str**(
  java.lang.String **n**)
  throws
  CosNaming.NamingContextPackage.**NotFound**,
  CosNaming.NamingContextPackage.**CannotProceed**,
  CosNaming.NamingContextPackage.**InvalidName**
  このメソッドは，文字列名を解決してオブジェクトリファレンスを返します。
  - n
    文字列化したオブジェクト名

  このメソッドで発生する可能性のある例外を次に示します。
  - NotFound
    Name またはそのコンポーネントの一つが見つかりません。
  - CannotProceed
    シーケンスの中に解決できない NameComponent オブジェクトがあります。クライアントは，返された NamingContext からオペレーションを継続できます。
  - InvalidName
    指定された Name にはネームコンポーネントがありません。または，ID フィールドに空文字列を指定したネームコンポーネントがあります。

8.3 Binding と BindingList

```
public interface Binding
```

Binding インタフェース，BindingList インタフェース，および BindingIterator インタフェースは，NamingContext オブジェクトの中に「名前とオブジェクトのバインド」を定義するために使用します。Binding 構造体は，名前とオブジェクトのペアを一組カプセル化します。binding_name フィールドは Name を表し，binding_type フィールドは ORB オブジェクトと NamingContext オブジェクトのどちらに Name をバインドするかを表します。

BindingList は，NamingContext オブジェクトに含まれる Binding 構造体のシーケンスです。

8.3.1 IDL の定義

```
module CosNaming {

    enum BindingType {
        nobject,
        ncontext
    }
    struct Binding {
        Name binding_name;
        BindingType binding_type;
    };
    typedef sequence<Binding>BindingList;
};
```

8.4 BindingIterator

```
public interface BindingIterator extends
    org.omg.CORBA.Object
```

このインタフェースを使用すると，クライアントアプリケーションは，NamingContext の list メソッドによって返されるこのオブジェクトを使用して，オブジェクトとネームのバインドを，数を指定しないで繰り返し参照できます。

「8.1.2 NamingContext のメソッド」の list メソッドも参照してください。

8.4.1 IDL の定義

```
module CosNaming {

    interface BindingIterator {
        boolean next_one(out Binding b);
        boolean next_n(in unsigned long how_many,
                       out BindingList b);
        void destroy( );
    };
};
```

8.4.2 BindingIterator のメソッド

- public boolean **next_one**(
  CosNaming.Binding **b**)
  このメソッドは，リストから次の Binding を返します。リストに次の Binding がない場合には false が返ります。それ以外の場合には true が返ります。
  - b
    リストの次の Binding オブジェクト
- public boolean **next_n**(
  int **how_many**, CosNaming.BindingList **b**)
  このメソッドは，リストから要求された Binding オブジェクトの数を含む BindingList を返します。リストに次の Binding がない場合には，返されるバインドの数が，要求された数よりも少ない場合があります。リストに次の Binding がない場合には false が返ります。それ以外の場合には true が返ります。
  - how_many
    要求される Binding オブジェクトの最大数
  - b
    要求された Binding オブジェクト数以下のオブジェクトが登録された BindingList

- public void **destroy**( )

  このメソッドは，オブジェクトを破棄し，オブジェクトに対応するメモリを解放します。このメソッドの呼び出しに失敗すると，メモリの使用量が増大します。

8.5 NamingContextFactory

```
public interface NamingContextFactory extends
    com.inprise.vbroker.CORBA.Object
```

このインタフェースは，最初の NamingContext を実体化します。クライアントは，このオブジェクトにバインドし，create_context メソッドを使用して最初のコンテキストを作成できます。最初のコンテキストが作成できたら，new_context メソッドを使用して，ほかのコンテキストを作成できます。

ネーミングサービスが起動するときに，このネーミングコンテキストファクトリのインスタンスが作成されます。

一つのルートコンテキストを自動的に作成する NamingContextFactory の作成方法については，「8.6 ExtendedNamingContextFactory」を参照してください。

8.5.1 IDL の定義

```
module CosNaming {

    interface NamingContextFactory {
        NamingContext create_context( );
        oneway void shutdown( );
    };
};
```

8.5.2 NamingContextFactory のメソッド

- public CosNaming.NamingContextExt **create_context**( )
  このメソッドによって，クライアントはネーミングコンテキストを作成します。ネーミングコンテキストにはルートコンテキストが指定されていません。そのため，NamingContextFactory を実体化しただけではネーミングコンテキストは作成されません。
- public void **shutdown**( )
  このメソッドによって，クライアントはネーミングサービスを正常に終了します。
- public CosNamingExt.ClusterManager **get_cluster_manager**( )
  このメソッドは，指定したクラスタ条件でクラスタを生成します。
- public void **remove_stale_contexts**(
  java.lang.String **password**)
  このメソッドによって，クライアントは Cluster オブジェクトから存続期間全体を通してメンバを削除します。
- public CosNaming.NamingContext[ ] **list_all_roots**(

java.lang.String password)

このメソッドによって，ルートコンテキストをすべて一覧表示できます。

引数に誤ったパスワードを指定すると UNKNOWN 例外が発生します。

8.6 ExtendedNamingContextFactory

```
public interface ExtendedNamingContextFactory extends
    CosNamingExt.NamingContextFactory
```

このインタフェースは，NamingContextFactory インタフェースを拡張し，拡張ネーミングサービスの起動時にファクトリ内でデフォルトルートの作成を可能にします。詳細については，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「クラスタの生成」の記述を参照してください。

8.6.1 IDL の定義

```
module CosNaming {
    interface ExtendedNamingContextFactory
                                    :NamingContextFactory{
        NamingContext root_context( );
    };
};
```

8.6.2 ExtendedNamingContextFactory のメソッド

- public CosNaming.NamingContextExt root_context( )

  このメソッドは，該当するオブジェクトのインスタンス作成時に，自動的に生成されたルートネーミングコンテキストを返します。

9 例外クラス（Java）

この章では，Borland Enterprise Server VisiBroker で使用する Java 言語の例外クラスについて説明します。

9.1　概要

CORBA システム例外は，java.lang.RuntimeException のサブクラスです。このため，このような例外が発生する可能性があるすべてのメソッドシグニチャで CORBA システム例外を宣言する必要はありません。

UserException は java.lang.Exception のサブクラスです。

注

　インタフェース階層の変更によって，org.omg.CORBA.Exception クラスは削除されました。

java.lang.Exception と例外の位置づけについては，「2.10　例外のマッピング」を参照してください。

9.2 システム例外

```
public class SystemException extends
    java.lang.RuntimeException
```

CORBA システム例外は，ランタイムに問題が起こると発生します。この例外は java.lang.RuntimeException から継承します。発生するすべての SystemException クラス，およびそれに対する意味を，表 9-1 に示します。

標準 IDL のシステム例外は，org.omg.CORBA.SystemException を継承するファイナル Java クラスにマッピングされ，IDL メジャー例外コード，マイナー例外コードへのアクセス，および例外の理由を記述する文字列を提供します。

注

org.omg.CORBA.SystemException のパブリックなコンストラクタはありません。これを継承するクラスだけ実体化されています。

現在，Borland Enterprise Server VisiBroker はマイナーコードの使用をサポートしていません。したがって，マイナーコードを設定する minor メソッドはありません。

9.2.1 SystemException の属性

- public CompletionStatus completed

  この属性は，オペレーションが完了したかどうかを示します。

  システム例外一覧を次の表に示します。

表 9-1 システム例外一覧（Java）

例外クラス名	説明
BAD_CONTEXT	コンテキストオブジェクトの処理エラーが発生しました。
BAD_INV_ORDER	オペレーション要求の前に，必要な前提条件オペレーションが呼び出されていません。
BAD_OPERATION	無効なオペレーションが実行されました。
BAD_PARAM	無効なパラメタが引き渡されました。
BAD_TYPECODE	ORB が不正な TypeCode を検出しました。
CODESET_INCOMPATIBLE	クライアントとサーバのコードセットに互換がないため，通信に失敗しました。
COMM_FAILURE	通信障害が発生しました。
DATA_CONVERSION	データ変換エラーが発生しました。
FREE_MEM	メモリを解放できません。

例外クラス名	説明
IMP_LIMIT	インプリメンテーションの上限に違反しました。
INITIALIZE	ORB 初期化障害が発生しました。
INTERNAL	内部エラーが発生しました。
INTF_REPOS	インタフェースリポジトリへのアクセスエラーが発生しました。
INV_FLAG	無効フラグが指定されました。
INV_IDENT	識別子の構文が無効です。
INV_OBJREF	無効なオブジェクトリファレンスが検出されました。
INV_POLICY	無効なポリシーの変更が検出されました。
INVALID_TRANSACTION	トランザクションコンテキストが不正です。
MARSHAL	マーシャルパラメタまたは結果が不当です。
NO_IMPLEMENT	オペレーションのインプリメンテーションが使用できません。
NO_MEMORY	動的メモリ割り当て障害が発生しました。
NO_PERMISSION	許可されていないオペレーションを実行しようとしました。
NO_RESOURCES	必要な資源を取得できませんでした。
NO_RESPONSE	クライアントが送信したリクエストの応答がまだありません。
OBJ_ADAPTER	オブジェクトアダプタが障害を検出しました。
OBJECT_NOT_EXIST	リクエストされたオブジェクトが存在していません。
PERSIST_STORE	パーシステントストレージ障害が発生しました。
REBIND	クライアントが，QoS ポリシーに矛盾する IOR を受信しました。
TIMEOUT	オペレーションがタイムアウトしました。
TRANSACTION_REQUIRED	リクエスト時に無効なトランザクションコンテキストがトランザクションサービスに渡されましたが，アクティブなトランザクションが必要です。
TRANSACTION_ROLLEDBACK	リクエストに対応するトランザクションがすでにロールバックされているか，またはロールバック用にマーキングされています。
TRANSIENT	通信エラーが検出されましたが，再接続できる場合があります。
UNKNOWN	未知の例外です。

9.3　ユーザ例外

```
public abstract class org.omg.CORBA.UserException extends
    java.lang.Exception implements org.omg.CORBA.portable.IDLEntity
```

UserException クラスは abstract ベースクラスであり，オブジェクトインプリメンテーションで発生する可能性がある例外を定義する場合に使用します。この例外型に対応づけられた状態情報はありませんが，派生したクラスが自分自身の状態情報を付けることがあります。コードサンプル 9-1 に示すように，このクラスの主な目的は，クライアントコード内の catch ブロックを簡単に使用できるようにすることです。

コードサンプル 9-1　システム例外とユーザ例外のキャッチ

```
try {
     proxy.operation();
}
catch(org.omg.CORBA.SystemException se) {
     System.out.println("The runtime failed: " + se);
}
catch(org.omg.CORBA.UserException ue) {
     System.out.println("The implementation failed: " + ue);
}
```

9.3.1　import 文

コード内に「**import org.omg.CORBA.*;**」と記述してください。

9.3.2　UserException のコンストラクタ

- protected org.omg.CORBA.**UserException**()
  空の UserException オブジェクトを生成します。
- protected org.omg.CORBA.**UserException**(
  String **reason**)
  指定された文字列を持つ UserException オブジェクトを生成します。
  - reason
    例外内容を表す文字列

10 ポータブルインタセプタインタフェースとクラス（Java）

この章では，OMG 標準規格で定義されたポータブルインタセプタのインタフェースとクラスの，Borland Enterprise Server VisiBroker でのインプリメンテーションについて，Java 言語でのインタフェースを説明します。これらのインタフェースとクラスの詳細については，「OMG Final Adopted 仕様」を参照してください。また，この章で説明するインタフェースを使用する前に，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「ポータブルインタセプタの使用」の記述を参照してください。

10.1 概要

VisiBroker ORB は，拡張機能をプラグインするためのインタセプタである API を提供します。例えば，トランザクションとセキュリティのサポートなどが拡張機能の一例です。インタセプタは，VisiBroker ORB の内部にフックされています。これによって，VisiBroker ORB サービスは，VisiBroker ORB の通常の実行の流れを受け取れます。Borland Enterprise Server VisiBroker がサポートしているインタセプタには，次の 2 種類があります。

- ポータブルインタセプタ

  OMG が標準化したインタセプタです。これによって，異なるベンダの ORB 間で使用できる，ポータブルなインタセプタのコードを記述できます。

- VisiBroker 4.x のインタセプタ

  VisiBroker 4.x で定義された，Borland Enterprise Server VisiBroker 独自のインタセプタです。

VisiBroker 4.x インタセプタの詳細については，マニュアル「Borland Enterprise Server VisiBroker デベロッパーズガイド」の「VisiBroker 4.x インタセプタの使用」の記述を参照してください。また，「11. VisiBroker 4.x インタセプタおよびオブジェクトラッパーのインタフェースとクラス（Java）」を参照してください。

ポータブルインタセプタには，次の 2 種類があります。

- リクエストインタセプタ

  リクエストインタセプタを使用すると，VisiBroker ORB サービスはクライアントとサーバの間でコンテキスト情報の受け渡しができるようになります。リクエストインタセプタには，クライアントリクエストインタセプタとサーバリクエストインタセプタの 2 種類があります。

- IOR インタセプタ

  IOR インタセプタを使用すると，VisiBroker ORB サービスはサーバまたはオブジェクトの ORB サービス関連の機能を定義する IOR に情報を登録できるようになります。例えば，SSL などのセキュリティサービスは，自身のタグ付きコンポーネントを IOR に登録できるようになります。これによって，そのコンポーネントを認識するクライアントは，そのコンポーネントの情報に基づいてサーバとの間にコネクションを確立できます。

ポータブルインタセプタの詳細については，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「ポータブルインタセプタの使用」の記述を参照してください。

10.2 ClientRequestInfo

```
public interface ClientRequestInfo extends RequestInfo,
    org.omg.CORBA.LocalInterface,
    org.omg.PortableInterceptor.ClientRequestInfoOperations,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，RequestInfo から派生したインタフェースで，クライアント側のインタセプトポイントに渡されます。ClientRequestInfo のメソッドは，一部のインタセプトポイントでは有効ではありません。

次の表に，属性またはメソッドの有効性を示します。無効の属性またはメソッドにアクセスすると，標準マイナーコード 14 の BAD_INV_ORDER 例外が発生します。

表 10–1 ClientRequestInfo の有効性（Java）

	send_request	send_poll	receive_reply	receive_exception	receive_other
request_id	○	○	○	○	○
operation	○	○	○	○	○
arguments	○※1	×	○	×	×
exceptions	○	×	○	○	○
contexts	○	×	○	○	○
operation_context	○	×	○	○	○
result	×	×	○	×	×
response_expected	○	○	○	○	○
sync_scope	○	×	○	○	○
reply_status	×	×	○	○	○
forward_reference	×	×	×	×	○※2
get_slot	○	○	○	○	○
get_request_service_context	○	×	○	○	○
get_reply_service_context	×	×	○	○	○
target	○	○	○	○	○
effective_target	○	○	○	○	○
effective_profile	○	○	○	○	○
received_exception	×	×	×	○	×
received_exception_id	×	×	×	○	×
get_effective_component	○	×	○	○	○

	send_request	send_poll	receive_reply	receive_exception	receive_other
get_effective_components	○	×	○	○	○
get_request_policy	○	×	○	○	○
add_request_service_context	○	×	×	×	×

（凡例）

○：有効，×：無効

注※1

ClientRequestInfo が send_request()に渡される場合，in，inout，または out の各パラメタのリストにエントリがありますが，使用できるのは inout パラメタと out パラメタだけです。

注※2

reply_status()が LOCATION_FORWARD を返さない場合，この属性にアクセスすると，標準マイナーコード 14 の BAD_INV_ORDER 例外が発生します。

10.2.1 import 文

コード内に「`import org.omg.PortableInterceptor.*;`」と記述してください。

10.2.2 ClientRequestInfo のメソッド

- public org.omg.CORBA.Object target( );

  このメソッドは，クライアントがオペレーションを実行するために呼び出したオブジェクトを返します。effective_target()メソッドも参照してください。

- public org.omg.CORBA.Object effective_target();

  このメソッドは，実際のオペレーション呼び出し元オブジェクトを返します。reply_status()が LOCATION_FORWARD を返す場合，後続のリクエストに対して target が返す内容は変わりませんが，effective_target は，フォワードされた IOR を返します。

- public org.omg.IOP.TaggedProfile effective_profile();

  このメソッドは，org.omg.IOP.TaggedProfile の形式でプロファイルを返します。返されたプロファイルは，リクエストの送信に使用されます。このオペレーションのオブジェクトにロケーションフォワードが発生し，そのオブジェクトのプロファイルが変更された場合，このプロファイルが探索結果のプロファイルとなります。

- public org.omg.CORBA.Any received_exception();

  このメソッドは，クライアントに返される例外を格納するデータを CORBA.Any の形式で返します。

  CORBA.Any に挿入できないユーザ例外の場合，例えば，未知の例外やバインディングで TypeCode が提供されない場合などは，この属性は標準マイナーコード 1 の UNKNOWN システム例外を格納す

る CORBA.Any となります。ただし，この例外の RepositoryId は，received_exception_id 属性に使用できます。

- public java.lang.String **received_exception_id**();
  このメソッドは，クライアントに返される received_exception の ID を返します。
- public org.omg.IOP.TaggedComponent **get_effective_component**(
  int **id**);
  このメソッドは，リクエストに対して選択されたプロファイルに指定された ID を持つ org.omg.IOP.TaggedComponent を返します。
  指定された ID のコンポーネントが複数ある場合に，どのコンポーネントを返すかは定義されていません。指定された ID のコンポーネントが複数ある場合は，このメソッドの代わりに get_effective_components()が呼び出されます。指定された ID のコンポーネントがない場合は，標準マイナーコード 28 の BAD_PARAM 例外が発生します。
  - id
    返されるコンポーネントの ID
- public org.omg.IOP.TaggedComponent[ ]
  get_effective_components(int **id**);
  このメソッドは，リクエストに対して選択されたプロファイルに指定された ID を持つタグ付きコンポーネントをすべて返します。org.omg.IOP.TaggedComponent の配列の形式で返されます。指定された ID のコンポーネントがない場合は，標準マイナーコード 28 の BAD_PARAM 例外が発生します。
  - id
    返されるコンポーネントの ID
- public org.omg.CORBA.Policy **get_request_policy**(
  int **type**);
  このメソッドは，オペレーションに対して有効な所定のポリシーを返します。指定した型に ORB が対応していない，または指定した型のポリシーオブジェクトがこのオブジェクトに対応づけられていないために，ポリシーの型が無効となる場合は，標準マイナーコード 2 の INV_POLICY 例外が発生します。
  - type
    返されるポリシーを指定するポリシーの型
- public void **add_request_service_context**(
  org.omg.IOP.ServiceContext **service_context**, boolean **replace**);
  このメソッドを使用すると，インタセプタで一つ以上のサービスコンテキストをリクエストに登録できます。サービスコンテキストの順序の宣言はありません。登録された順序でサービスコンテキストが表示される場合も，されない場合もあります。
  - service_context
    リクエストに登録する IOP.ServiceContext
  - replace

指定した ID のサービスコンテキストがすでに存在する場合のメソッドの動作。false の場合は，標準マイナーコード 15 の BAD_INV_ORDER 例外が発生します。true の場合は，既存のコンテキストが新しいコンテキストに置き換えられます。

10.3 ClientRequestInterceptor

```
public interface ClientRequestInterceptor extends Interceptor,
    org.omg.CORBA.LocalInterface,
    org.omg.PortableInterceptor.ClientRequestInterceptorOperations,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，ユーザ定義クライアント側インタセプタを継承するときに使用します。ClientRequestInterceptor インスタンスは，VisiBroker ORB に登録されます。詳細については，「10.4 Codec」を参照してください。

10.3.1 import 文

コード内に「`import org.omg.PortableInterceptor.*;`」と記述してください。

10.3.2 ClientRequestInterceptor のメソッド

- public void send_request(
  ClientRequestInfo ri)
  throws
  ForwardRequest;
  このインタセプトポイントを使用すると，リクエストがサーバに送信される前に，リクエスト情報の照会とサーバコンテキストの修正が，インタセプタでできます。
  このインタセプトポイントではシステム例外が発生する場合があります。システム例外が発生した場合，ほかのインタセプタの send_request()インタセプトポイントは呼び出されません。フロースタックからインタセプタが取り出され，そのインタセプタのインタセプトポイントが呼び出されます。
  このインタセプトポイントでは，ForwardRequest 例外を発生させることができます。インタセプトでこの例外が発生した場合，ほかのインタセプタの send_request メソッドは呼び出されません。フロースタックからインタセプタが取り出され，そのインタセプタの receive_other()インタセプトポイントが呼び出されます。ForwardRequest 例外については，「10.8 ForwardRequest」を参照してください。
  - ri
    インタセプタが使用する ClientRequestInfo インスタンス
- public void send_poll(
  ClientRequestInfo ri);
  このインタセプトポイントを使用すると，時間非依存呼び出し（TII）によるポーリング get reply シーケンス中に，インタセプタで情報を照会できます。

ただし，VisiBroker ORB は TII に対応していないため，この send_poll()インタセプトポイントが呼び出されることはありません。

- ri

  インタセプタが使用する ClientRequestInfo インスタンス

- public void **receive_reply**(

  ClientRequestInfo **ri**);

  このインタセプトポイントを使用すると，サーバから応答が返されてから，制御がクライアントに戻るまでの間に，インタセプタで応答の情報を照会できます。

  このインタセプトポイントでは，システム例外が発生する場合があります。システム例外が発生した場合，ほかのインタセプタの receive_reply()メソッドは呼び出されません。フロースタックからインタセプタが取り出され，そのインタセプタの receive_exception()インタセプトポイントが呼び出されます。

  - ri

    インタセプタが使用する ClientRequestInfo インスタンス

- public void **receive_exception**(

  ClientRequestInfo **ri**)

  throws

  ForwardRequest;

  このインタセプトポイントは例外が発生したときに呼び出されます。これによって，例外の情報がクライアントに通知される前にインタセプタで例外の情報を照会できます。

  このインタセプトポイントでは，システム例外が発生する場合があります。システム例外が発生した場合は，フロースタックから取り出された一連のインタセプタが，receive_exception()呼び出し時に受け取る例外が変更されます。クライアントに通知される例外は，インタセプタが通知する最後の例外です。ほかのインタセプタが例外を変更しなければ，元の例外が通知されます。

  また，このインタセプトポイントでは，ForwardRequest 例外を発生させることができます。インタセプタでこの例外が発生した場合，ほかのインタセプタの receive_exception()インタセプトポイントは呼び出されません。フロースタックからインタセプタが取り出され，そのインタセプタの receive_other()インタセプトポイントが呼び出されます。ForwardRequest 例外については，「10.8 ForwardRequest」を参照してください。

  - ri

    インタセプタが使用する ClientRequestInfo インスタンス

- public void **receive_other**(

  ClientRequestInfo **ri**)

  throws

  ForwardRequest;

  このインタセプトポイントを使用すると，リクエストの結果が，正常な応答でも例外でもない場合に使用できる情報をインタセプタで照会できます。それは，リクエストがリトライになる場合（例えば，LOCATION_FORWARD 状態で GIOP Reply を受信した場合）や，非同期呼び出し時にリクエスト

の直後に応答が返されないで，制御がクライアントに戻って終了インタセプトポイントが呼び出される場合などです。

リクエストがリトライになる場合は，適用されているポリシーによって，リトライ指示直後に新しいリクエストが発行されるときと発行されないときがあります。新しいリクエストが発行されるときは，このリクエストが新しいリクエストである間は，インタセプタに関して，元のリクエストとリトライとの間に相関性があります。制御がクライアントに戻らないため，リクエストをスコープとする Current は，元のリクエストの場合もリトライリクエストの場合も同一です。詳細については，「10.12 IORInterceptor」を参照してください。

このインタセプトポイントでは，システム例外が発生する場合があります。システム例外が発生した場合，ほかのインタセプタの receive_other()インタセプトポイントは呼び出されません。フロースタックからインタセプタが取り出され，そのインタセプタの receive_exception()インタセプトポイントが呼び出されます。

また，このインタセプトポイントでは，ForwardRequest 例外を発生させることができます。インタセプタがこの例外を出力した場合，ForwardRequest 例外が提供する新しい情報で一連のインタセプタの receive_other()メソッドが呼び出されます。

- ri

  インタセプタが使用する ClientRequestInfo インスタンス

10.4 Codec

```
public interface Codec extends
    org.omg.CORBA.LocalInterface,
    org.omg.IOP.CodecOperations,
    org.omg.CORBA.portable.IDLEntity
```

ORB サービスが使用する IOR コンポーネントとサービスコンテキストデータの形式は，IDL データ型を持つ CDR カプセル化エンコーディングインスタンスとして定義される場合があります。Codec は，IDL データ型と CDR カプセル化表現との間でコンポーネントを受け渡すための機能を持っています。

Codec は CodecFactory から取得します。CodecFactory は，
ORB.resolve_initial_references("CodecFactory")の呼び出しで取得できます。

10.4.1 import 文

コード内に「`import org.omg.IOP.*;`」と記述してください。

10.4.2 Codec のメンバ

- public final class InvalidTypeForEncoding extends org.omg.CORBA.UserException;
  この例外は，エンコーディングに不正な型が指定されている場合に，encode()または encode_value()が出力します。
- public final class FormatMismatch extends org.omg.CORBA.UserException;
  この例外は，オクテットシーケンスのデータを CORBA.Any にデコードできない場合に，decode()または decode_value()が出力します。
- public final class TypeMismatch extends org.omg.CORBA.UserException;
  この例外は，指定した TypeCode とオクテットシーケンスが適合しない場合に，decode_value()が出力します。

10.4.3 Codec のメソッド

- public byte[ ] encode(
  org.omg.CORBA.Any data)
  throws
  InvalidTypeForEncoding
  このメソッドは，この Codec に適用されているエンコーディング形式に基づいて，CORBA.Any の形式で指定したデータをオクテットシーケンスに変換します。このオクテットシーケンスには，TypeCode

と型のデータの両方が格納されます。このオペレーションでは，InvalidTypeForEncoding 例外が発生する場合があります。

- data
  オクテットシーケンスに変換する，CORBA.Any 形式のデータ

- public org.omg.CORBA.Any **decode**(
  byte[ ] **data**)
  throws
  FormatMismatch, TypeMismatch;
  このメソッドは，この Codec に適用されているエンコーディング形式に基づいて，指定したオクテットシーケンスを CORBA.Any オブジェクトにデコードします。
  オクテットシーケンスを CORBA.Any にデコードできない場合，このメソッドは FormatMismatch 例外を出力します。
  - data
    CORBA.Any に変換する，オクテットシーケンス形式のデータ
- public byte[ ] **encode_value**(
  org.omg.CORBA.Any **data**)
  throws
  InvalidTypeForEncoding;
  このメソッドは，この Codec に適用されているエンコーディング形式に基づいて，指定した CORBA.Any オブジェクトをオクテットシーケンスに変換します。CORBA.Any のデータだけをエンコードします。TypeCode のデータはエンコードしません。
  - data
    エンコード済み CORBA.Any のデータを格納するオクテットシーケンス
- public org.omg.CORBA.Any **decode_value**(
  byte[ ] **data**, org.omg.CORBA.TypeCode **tc**);
  このメソッドは，指定した TypeCode と，この Codec に適用されているエンコーディング形式に基づいて，指定したオクテットシーケンスを CORBA.Any にデコードします。
  オクテットシーケンスを CORBA.Any にデコードできない場合，このメソッドは FormatMismatch 例外を出力します。
  - data
    CORBA.Any にデコードする，オクテットシーケンス形式のデータ
  - tc
    データのデコードに使用する TypeCode

10.5 CodecFactory

```
public interface CodecFactory extends
    org.omg.CORBA.LocalInterface,
    org.omg.IOP.CodecFactoryOperations,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，Codec を取得するときに使用します。CodecFactory は，ORB.resolve_initial_references("CodecFactory")の呼び出しで取得できます。

10.5.1 import 文

コード内に「`import org.omg.IOP.*;`」と記述してください。

10.5.2 CodecFactory のメンバ

- public final class UnknownEncoding extends org.omg.CORBA.UserException

  この例外は，CodecFactory で Codec を生成できない場合に発生します。「10.5.3 CodecFactory のメソッド」の create_codec()メソッドを参照してください。

10.5.3 CodecFactory のメソッド

- public Codec create_codec(

  Encoding enc)

  throws

  UnknownEncoding

  このメソッドは，指定したエンコーディング形式の Codec を生成します。

  指定したエンコーディング形式の Codec をファクトリが生成できない場合は，UnknownEncoding 例外を出力します。

  - enc

    Codec の生成に使用するエンコーディング形式

10.6 Current

```
public interface Current extends
    org.omg.CORBA.LocalInterface,
    org.omg.PortableInterceptor.CurrentOperations,
    org.omg.CORBA.portable.IDLEntity
```

Current インタフェースは，単なるスロットテーブルです。このテーブルのスロットは，スロットのコンテキストと，リクエストまたは応答のサービスコンテキストとの間で，各サービスがコンテキストデータを受け渡すために使用されます。

Current を使用する各サービスは，初期化時に確保したスロットを，リクエストと応答の処理中に使用します。「10.14.3 ORBInitInfo のメソッド」の allocate_slot_id()メソッドを参照してください。

Current は，ORB.resolve_initial_references("PICurrent")の呼び出しで取得されます。

RequestInfo オブジェクトの get_slot()メソッドを使用すると，インタセプトポイントの内側から，スレッドスコープからリクエストスコープへ移動した Current のデータを使用できます。

10.6.1 import 文

コード内に「**import org.omg.PortableInterceptor.*;**」と記述してください。

10.6.2 Current のメソッド

- public org.omg.CORBA.Any **get_slot**(
  int **id**)
  throws
  InvalidSlot;
  このメソッドを使用すると，サービスは，PICurrent に設定したスロットデータを取得できます。
  設定されていないスロットを指定した場合は，tk_null の TCKind 値を持つタイプコードを格納する CORBA.Any を返します。
  割り当てられていないスロットに対して get_slot()を呼び出すと，InvalidSlot 例外が発生します。
  ORB イニシャライザの内側から get_slot()を呼び出すと，マイナーコード 14 の BAD_INV_ORDER 例外が発生します。ORB イニシャライザについては，「10.4 Codec」を参照してください。
  - id
    データを取り出すスロットの SlotId
- public void **set_slot**(
  int **id**, org.omg.CORBA.Any **data**)

throws
InvalidSlot;
サービスは，このメソッドを使用して，スロットにデータを CORBA.Any オブジェクトの形式で設定します。
スロットにすでにデータが設定されている場合，既存のデータは上書きされます。
割り当てられていないスロットに対して set_slot()を呼び出すと，InvalidSlot 例外が発生します。
ORB イニシャライザの内側から set_slot()を呼び出すと，マイナーコード 14 の BAD_INV_ORDER 例外が発生します。ORB イニシャライザについては，「10.4　Codec」を参照してください。

- id
  データを設定するスロットの SlotId
- data
  指定したスロットに設定する，CORBA.Any オブジェクト形式のデータ

10.7 Encoding

```
public final class Encoding implements
    org.omg.CORBA.portable.IDLEntity
```

このクラスは，Codec のエンコーディング形式を定義します。CDR カプセル化エンコーディングなどのエンコード形式，メジャーバージョン，およびマイナーバージョンを定義します。

次のエンコード形式に対応しています。

- ENCODING_CDR_ENCAPS バージョン 1.0
- ENCODING_CDR_ENCAPS バージョン 1.1
- ENCODING_CDR_ENCAPS バージョン 1.2
- GIOP の将来のバージョンすべてに対応する ENCODING_CDR_ENCAPS

10.7.1 import 文

コード内に「`import org.omg.IOP.*;`」と記述してください。

10.7.2 Encoding のメンバ

- public short format;
  このメンバは，Codec のエンコーディング形式を保持します。
- public byte major_version;
  このメンバは，Codec のメジャーバージョン番号を保持します。
- public byte minor_version;
  このメンバは，Codec のマイナーバージョン番号を保持します。

10.8 ForwardRequest

```
public final class ForwardRequest extends
    org.omg.CORBA.UserException
```

インタセプタは，ForwardRequest 例外を使用して新規オブジェクトを指定し，リクエストのリトライを ORB に指示できます。インタセプタからの ForwardRequest 例外を ORB が受信した場合だけ，リトライが指示されます。それ以外の場合に ForwardRequest 例外が発生すると，その例外は，ユーザ例外と同様に ORB を介して渡されます。

インタセプタの呼び出しに対してインタセプタから ForwardRequest 例外が出力された場合，そのインタセプトポイントに，ほかの Interceptor は呼び出されません。フロースタックに蓄積されたインタセプタに対応する終了インタセプトポイント（クライアントの receive_other，またはサーバの send_other()）が呼び出されます。receive_other()および send_other()の中では，reply_status()は LOCATION_FORWARD を返します。

10.8.1 import 文

コード内に「`import org.omg.PortableInterceptor.*;`」と記述してください。

10.8.2 ForwardRequest の変数

- public org.omg.CORBA.Object forward;
  この変数は，転送するオブジェクトリファレンスを表します。

10.8.3 ForwardRequest のメソッド

- public ForwardRequest()
  このメソッドは，空のプロパティを持つ ForwardRequest オブジェクトを生成します。
- public ForwardRequest(
  org.omg.CORBA.Object ref)
  このメソッドは，指定されたプロパティを持つ ForwardRequest オブジェクトを生成します。
  - ref
    転送するオブジェクトリファレンス
- public ForwardRequest(
  java.lang.String reason, org.omg.CORBA.Object ref)
  このメソッドは，指定されたプロパティを持つ ForwardRequest オブジェクトを生成します。

- reason
  生成する ForwardRequest オブジェクトの詳細メッセージ
- ref
  転送するオブジェクトリファレンス

10.9 Interceptor

```
public interface Interceptor extends
    org.omg.CORBA.LocalInterface,
    org.omg.PortableInterceptor.InterceptorOperations,
    org.omg.CORBA.portable.IDLEntity
```

Interceptor は，すべてのインタセプタの派生元となるベースクラスです。

10.9.1 import 文

コード内に「**import org.omg.PortableInterceptor.*;**」と記述してください。

10.9.2 Interceptor のメソッド

- public java.lang.String **name**();

  このメソッドは，インタセプタの名前を返します。個々のインタセプタには，インタセプタを並べ替えるための名前を付けられます。インタセプタ型ごとに，指定した名前を持つ唯一のインタセプタを VisiBroker ORB に登録できます。空文字列を名前に設定することで，インタセプタを匿名にすることもできます。

  VisiBroker ORB には，幾つでも匿名のインタセプタを登録できます。

- public void **destroy**();

  このメソッドは，ORB.destroy()の実行中に呼び出されます。アプリケーションによって ORB.destroy()が呼び出されると，VisiBroker ORB は次のように処理します。

  1. 処理中のリクエストがすべて完了するまで待ちます。
  2. Interceptor.destroy()メソッドをインタセプタごとに呼び出します。
  3. VisiBroker ORB のデストラクトを完了します。

  デストラクト中の VisiBroker ORB に実装されているオブジェクトのオブジェクトリファレンスに，Interceptor.destroy()の中からメソッド呼び出しをした場合の動作は保証できません。ただし，デストラクト中ではない VisiBroker ORB に，実装されているオブジェクトのメソッド呼び出しはできます。つまり，デストラクト中の VisiBroker ORB は，クライアントとしては使用できますが，サーバとしては使用できません。

10.10 IORInfo

```
public interface IORInfo extends
    org.omg.CORBA.LocalInterface,
    org.omg.PortableInterceptor.IORInfoOperations,
    org.omg.CORBA.portable.IDLEntity
```

IORInfo インタフェースによって，サーバ側の ORB サービスはコンポーネントを追加したり，IOR 構築中に適用可能なポリシーへアクセスしたりできるようになります。

ORB は，このインタフェースの ORB のインプリメンテーションのインスタンスを IORInterceptor.establish_components()にパラメタとして渡します。

次の表に，IORInterceptor に定義されたメソッドの，IORInfo での属性またはメソッドの有効性を示します。IORInfo の属性またはメソッドに，不当な呼び出しをすると，標準マイナーコード 14 の BAD_INV_ORDER 例外が発生します。

表 10-2 IORInfo の有効性（Java）

	establish_components	components_established
get_effective_policy	○	○
add_component	○	×
add_component_to_profile	○	×
manager_id	○	○
state	○	○
adapter_template	×	○
current_factory	×	○

（凡例）

○：有効，×：無効

10.10.1 import 文

コード内に「`import org.omg.PortableInterceptor.*;`」と記述してください。

10.10.2 IORInfo のメソッド

- public org.omg.CORBA.Policy **get_effective_policy**(
  int **type**);

ORB サービスのインプリメンテーションでは，get_effective_policy()メソッドを呼び出すことで，特定の型のどのサーバ側ポリシーが構築中の IOR に適用されているかを調べられます。構築中の IOR が，POA を使用して実装されたオブジェクトの IOR である場合，その POA を生成した PortableServer.POA.create_POA()呼び出しで渡された Policy オブジェクトはすべて，get_effective_policy でアクセスできます。

指定した型のポリシーを ORB が認識していない場合，このメソッドは標準マイナーコード 3 の INV_POLICY 例外を出力します。

- type
  取得するポリシーの型を指定した CORBA.PolicyType

- public void **add_ior_component**(
  org.omg.IOP.TaggedComponent **a_component**);
  このメソッドは，IOR 構築時にインクルードされるタグ付きコンポーネントのセットにメンバを追加するときに，establish_components()から呼び出されます。すべてのプロファイルにコンポーネントのセットがインクルードされます。
  同じコンポーネント ID のコンポーネントを複数存在させることもできます。
  - a_component
    追加する IOP.TaggedComponent
- public void **add_ior_component_to_profile**(
  org.omg.IOP.TaggedComponent **a_component**,int **profile_id**);
  このメソッドは，IOR 構築時にインクルードされるタグ付きコンポーネントのセットにメンバを追加するときに，establish_components()から呼び出されます。指定したプロファイルにコンポーネントのセットがインクルードされます。
  指定したプロファイル ID が既存のプロファイルを定義していない場合，およびプロファイルにコンポーネントを追加できない場合，標準マイナーコード 29 の BAD_PARAM 例外が発生します。
  - a_component
    追加する IOP.TaggedComponent
  - profile_id
    コンポーネントを追加するプロファイルの IOP.ProfileId
- public int **manager_id**();
  このメソッドは，アダプタのマネージャへの不透明なハンドルを提供する属性を返します。このメソッドは，同じアダプタマネージャに管理されているアダプタの状態変更を通知するために使用します。
- public short **state**();
  このメソッドは，アダプタの現在の状態を返します。状態として，HOLDING，ACTIVE，DISCARDING，INACTIVE，または NON_EXISTENT のどれかを返します。
- public ObjectReferenceTemplate **adapter_template**();
  IOR インタセプタが呼び出されたときは常に，このメソッドがオブジェクトリファレンスのテンプレートを取得するための属性を返します。オブジェクトリファレンスのテンプレートを直接作成する方法は

標準では提供されません。adapter_template()が返す値は，add_component()とadd_component_to_profile()の IOR インタセプタ呼び出しのために作成されるテンプレート，およびアダプタポリシーです。adapter_template()が返す値は，オブジェクトアダプタが存続している間は変更されません。

- public ObjectReferenceFactory current_factory();

  このメソッドが返す属性を使用して，アダプタがオブジェクトリファレンスを作成するときに使用するファクトリにアクセスできます。current_factory()が返す初期値は adapter_template 属性と同じ値ですが，current_factory にほかのファクトリを設定することで変更できます。オブジェクトアダプタが作成するオブジェクトリファレンスは，すべて current_factory の make_object()メソッドを呼び出して作成する必要があります。

- public void current_factory(
  ObjectReferenceFactory current_factory);

  このメソッドでは，current_factory 属性を設定します。アダプタが使用する current_factory 属性の値を設定できるのは，components_established()メソッドの呼び出し時だけです。

  - current_factory
    設定対象の current_factory オブジェクト

10.11 IORInfoExt

```
public interface IORInfoExt extends
    org.omg.CORBA.LocalInterface,
    org.omg.PortableInterceptor.IORInfo,
    com.inprise.vbroker.PortableInterceptor.IORInfoExtOperations,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースでは，Borland Enterprise Server VisiBroker のポータブルインタセプタを拡張して，POA をスコープとするサーバリクエストインタセプタをインストールします。IORInfoExt インタフェースは，IORInfo インタフェースから継承します。IORInfoExt インタフェースは，POA をスコープとするサーバリクエストインタセプタに対応するための拡張メソッドを提供します。

10.11.1 import 文

コード内に「`import com.inprise.vbroker.PortableInterceptor.*;`」と記述してください。

10.11.2 IORInfoExt のメソッド

- public void add_server_request_interceptor(
  ServerRequestInterceptor interceptor);
  このメソッドは，POA をスコープとするサーバ側リクエストインタセプタをサーバに追加するときに使用します。
  - interceptor
    追加する ServerRequestInterceptor
- public java.lang.String full_poa_name();
  このメソッドは，POA のフルネームを返します。

10.12 IORInterceptor

```
public interface IORInterceptor extends Interceptor,
    org.omg.CORBA.LocalInterface,
    org.omg.PortableInterceptor.IORInterceptorOperations,
    org.omg.CORBA.portable.IDLEntity
```

ポータブル ORB サービスインプリメンテーションでは，クライアントの ORB サービスインプリメンテーションが正しく機能するように，必要に応じてサーバまたはオブジェクトの ORB サービスに関連する機能の定義情報をオブジェクトリファレンスに追加する必要があります。それは，IORInterceptor インタフェースと IORInfo インタフェースが提供しています。IOR インタセプタは，IOR にあるプロファイルにタグ付きコンポーネントを設定するために使用します。

10.12.1 import 文

コード内に「`import org.omg.PortableInterceptor.*;`」と記述してください。

10.12.2 IORInterceptor のメソッド

- public void establish_components(
  IORInfo info);

  サーバ側 ORB は，特定のオブジェクトリファレンスの一つ以上のプロファイルにインクルードするコンポーネントの一覧を生成するときに，登録済み IORInterceptor インスタンスすべての establish_components()メソッドを呼び出します。このメソッドは，オブジェクトリファレンスごとに呼び出す必要はありません。POA の場合は，POA.create_POA()呼び出し時に毎回呼び出されます。ほかのアダプタの場合は，通常，アダプタの初期化時に呼び出されます。

  この段階では，アダプタテンプレートは使用できません。それは，アダプタテンプレートに必要な情報（コンポーネント）がまだ作成されていないためです。

  - info

    適用可能なポリシーを照会し，生成された IOR にインクルードするコンポーネントを追加するときに，ORB サービスが使用する IORInfo インスタンス

- public void components_established(
  IORInfo info);

  establish_components()メソッドがすべて呼び出されると，登録済み IOR インタセプタすべての components_established()メソッドが呼び出されます。この段階では，アダプタテンプレートが使用できます。また，current_factory 属性を取得したり設定したりすることもできます。

  components_established()で発生する例外はすべて，components_established()の呼び出し元に返されます。このため，POA の場合，create_POA()呼び出しは失敗し，標準マイナーコード 6 の OBJ_ADAPTER 例外が create_POA()の呼び出し元に返されます。

- info
  適用可能なポリシーにアクセスするために ORB サービスが使用する IORInfo インスタンス

- public void **adapter_manager_state_changed**(
  int **id**, short **state**);
  アダプタマネージャの状態が変わったときに，登録済み IOR インタセプタすべての adapter_manager_state_changed()メソッドが呼び出されます。
  状態の変化が adapter_manager_state_changed()によって通知される場合は，adapter_state_changed()では通知されません。
  - id
    適用可能なポリシーにアクセスするために ORB サービスが使用する IORInfo インスタンス
  - state
    オブジェクトアダプタの変更後の状態
- public void **adapter_state_changed**(
  ObjectReferenceTemplate[ ] **templates**, short **state**);
  アダプタマネージャの状態の変化と関連なく，一つ以上のオブジェクトアダプタの状態が変更された場合は，このメソッドに通知されます。templates パラメタは，状態が変更されたオブジェクトアダプタをテンプレート ID 情報で識別します。このパラメタには，通知対象のオブジェクトアダプタの，すべてのアダプタテンプレートが格納されます。
  - templates
    状態が変更されたオブジェクトアダプタをテンプレート ID 情報で識別します。
  - state
    オブジェクトアダプタの変更後の状態

10.13 ORBInitializer

```
public interface ORBInitializer extends
    org.omg.CORBA.LocalInterface,
    org.omg.PortableInterceptor.ORBInitializerOperations,
    org.omg.CORBA.portable.IDLEntity
```

ORBInitializer インタフェースをインプリメントする ORBInitializer オブジェクトを登録すると，対応するインタセプタが登録されます。ORB は，初期化時に登録済み ORBInitializer をすべて呼び出し，それぞれに ORBInitInfo オブジェクトを渡します。この ORBInitInfo オブジェクトは，ORBInitializer のインタセプタの登録に使用されます。

10.13.1 import 文

コード内に「`import org.omg.PortableInterceptor.*;`」と記述してください。

10.13.2 ORBInitializer のメソッド

- public void pre_init(
  ORBInitInfo info);
  このメソッドは，ORB の初期化中に呼び出されます。インタセプタによって登録された初期サービスが，ほかのインタセプタに使用される場合は，事前に ORBInitInfo.register_initial_reference()呼び出しで，その初期サービスが登録されます。
  - info
    インタセプタを登録するための初期化属性とメソッド
- public void post_init(
  ORBInitInfo info);
  このメソッドは，ORB の初期化中に呼び出されます。サービスが初期化の一部として初期リファレンスを解決する必要がある場合は，サービスは，この段階ですべての初期リファレンスを使用できるとみなします。
  post_init()メソッド呼び出しは，ORB の初期化の最後の処理ではありません。post_init()のあとに，登録済みインタセプタの一覧を ORB にアタッチするのが最後の処理です。したがって，post_init()呼び出し中は，ORB にはインタセプタが含まれていません。ORB を介する呼び出しが post_init()の内部で実行された場合，その呼び出しに対するリクエストインタセプタは呼び出されません。
  同様に，IOR を作成するメソッドが実行された場合も，IOR インタセプタは呼び出されません。
  - info
    インタセプタを登録するための初期化属性とメソッド

10.14 ORBInitInfo

```
public interface ORBInitInfo extends
    org.omg.CORBA.LocalInterface,
    org.omg.PortableInterceptor.ORBInitInfoOperations,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，インタセプタを登録するために ORBInitializer オブジェクトに渡されます。

10.14.1 import 文

コード内に「**import org.omg.PortableInterceptor.*;**」と記述してください。

10.14.2 ORBInitInfo のメンバ

- public final class DuplicateName extends org.omg.CORBA.UserException;
  インタセプタ型ごとに，指定した名前を持つ唯一のインタセプタを ORB に登録できます。既存のインタセプタと同じ名前でインタセプタを登録しようとすると，DuplicateName 例外が発生します。
  空文字列を名前に設定することで，インタセプタを匿名にすることもできます。ORB には，幾つでも匿名のインタセプタを登録できます。このため，匿名のインタセプタを登録する場合は，DuplicateName 例外は発生しません。
- public final class InvalidName extends org.omg.CORBA.UserException;
  この例外は，register_initial_reference()と resolve_initial_references()で発生します。
  register_initial_reference()で InvalidName 例外が発生するのは次の場合です。
  - このメソッドが空文字列 ID で呼び出された場合
  - このメソッドが，すでに登録されている ID で呼び出された場合（OMG 定義済みの既定の名前で呼び出された場合も含みます）

  resolve_initial_references()では，解決する名前が不正な場合に InvalidName 例外が発生します。

10.14.3 ORBInitInfo のメソッド

- public java.lang.String[ ] arguments();
  このメソッドは，ORB_init()に渡されたパラメタを返します。パラメタには，ORB のパラメタが含まれる場合も，含まれない場合もあります。
- public java.lang.String orb_id();
  このメソッドは，初期化中の ORB の ID を返します。
- public CodecFactory codec_factory();

このメソッドは，IOP_CodecFactory を返します。通常，CodecFactory は，ORB.resolve_initial_references("CodecFactory")を呼び出して取得しますが，その時点では，ORB はまだ使用できません。しかし，サービスコンテキスト処理時などに，インタセプタが Codec を必要とするため，Codec を取得する手段が ORB の初期化時に必要となります。

- public void register_initial_reference(
  java.lang.String id, org.omg.CORBA.Object obj);
  例えば，このメソッドが ID「Y」とオブジェクト「YY」で呼び出されたあとに，register_initial_reference メソッドが再び呼び出されると，オブジェクト「YY」が返されます。
  このメソッドの機能は，ORB.register_initial_reference()と同一です。このメソッドは，ORB が完全に初期化されていないためにまだ使用できない場合に，インタセプタの登録の一部として初期リファレンスが必要となるときに使用します。なお，このメソッドと ORB::register_initial_reference()との違いは，ORB のメソッドのバージョンは PIDL（CORBA.ORB.ObjectId と CORBA.ORB.InvalidName）を使用し，インタフェースのバージョンは，インタフェースに定義された IDL を使用する点です。構文に違いはありません。
  このメソッドが空文字列 ID で呼び出された場合，またはこのメソッドがすでに登録されている ID で呼び出された場合（OMG 定義済みの既定の名前で呼び出された場合も含みます），register_initial_reference()で InvalidName 例外が発生します。
  - id
    初期リファレンスを認識するための ID
  - obj
    初期リファレンス
- org.omg.CORBA.Object resolve_initial_references(
  java.lang.String id)
  throws
  InvalidName;
  このメソッドは，post_init()呼び出し時にだけ有効です。このメソッドの機能は ORB.resolve_initial_references()と同一です。このメソッドは，ORB が完全に初期化されていないためにまだ使用できない場合に，インタセプタの登録の一部として初期リファレンスが必要となるときに使用します。
  - id
    初期リファレンスを認識するための ID

  解決する名前が不正な場合，resolve_initial_references()では InvalidName 例外が発生します。
- public void add_client_request_interceptor(
  ClientRequestInterceptor interceptor)
  throws
  DuplicateName;

このメソッドは，クライアント側リクエストインタセプタの一覧に項目を追加するときに使用します。同じ名前のクライアント側リクエストインタセプタがすでに存在している場合は，DuplicateName 例外が発生します。

- interceptor
  追加する ClientRequestInterceptor

- public void **add_server_request_interceptor**(
  ServerRequestInterceptor **interceptor**)
  throws
  DuplicateName;
  このメソッドは，サーバ側リクエストインタセプタの一覧に項目を追加するときに使用します。同じ名前のサーバ側リクエストインタセプタがすでに存在している場合は，DuplicateName 例外が発生します。
  - interceptor
    追加する ServerRequestInterceptor
- public void **add_ior_interceptor**(
  IORInterceptor **interceptor**)
  throws
  DuplicateName;
  このメソッドは，IOR インタセプタの一覧に項目を追加するときに使用します。同じ名前の IOR インタセプタがすでに存在している場合は，DuplicateName 例外が発生します。
  - interceptor
    追加する IORInterceptor
- public int **allocate_slot_id**();
  このメソッドは，割り当て済みスロットのインデックスを返します。
  サービスは，allocate_slot_id を呼び出して PortableInterceptor.Current のスロットを割り当てます。
  注
  スロット ID の割り当ては，ORB イニシャライザの内部でできますが，スロット自体は初期化できません。ORB イニシャライザの内部で Current の set_slot()または get_slot()を呼び出すと，マイナーコード 14 の BAD_INV_ORDER 例外が発生します。詳細については，「10.12 IORInterceptor」を参照してください。
- public void **register_policy_factory**(
  int **type**, PolicyFactory **policy_factory**);
  このメソッドは，指定した PolicyType の PolicyFactory を登録します。
  指定した PolicyType の PolicyFactory がすでに登録されている場合は，標準マイナーコード 16 の BAD_INV_ORDER 例外が発生します。
  - type

指定した PolicyFactory の CORBA.PolicyType

- policy_factory
  指定した CORBA.PolicyType のファクトリ

10.15 Parameter

```
public final class Parameter
```

このクラスは，パラメタ情報を保持します。このクラスは，RequestInfo クラスの arguments()メソッドからパラメタ情報を渡すときに使用します。詳細については，「10.12 IORInterceptor」を参照してください。

10.15.1 import 文

コード内に「`import org.omg.Dynamic.*;`」と記述してください。

10.15.2 Parameter のメンバ

- public org.omg.CORBA.Any argument;
  このメンバは，CORBA.Any 形式でパラメタデータを格納します。
- public org.omg.CORBA.ParameterMode mode;
  このメンバは，パラメタのモードを指定します。PARAM_IN，PARAM_OUT，または PARAM_INOUT のどちらかの enum 値がこのメンバの値となります。

10.16 PolicyFactory

```
public interface PolicyFactory extends
    org.omg.CORBA.LocalInterface,
    org.omg.PortableInterceptor.PolicyFactoryOperations,
    org.omg.CORBA.portable.IDLEntity
```

ポータブル ORB サービスインプリメンテーションは，ORB 初期化時に，PolicyFactory インタフェースのインスタンスを登録します。これは，CORBA.ORB.create_policy()を使用してポリシーの型を構成できるようにするためです。POA は，この方法で ORBInitInfo に登録されたポリシーを保存するために必要です。詳細については，「10.14.3 ORBInitInfo のメソッド」の register_policy_factory()メソッドを参照してください。

10.16.1 import 文

コード内に「`import org.omg.PortableInterceptor.*;`」と記述してください。

10.16.2 PolicyFactory のメソッド

- public org.omg.CORBA.Policy create_policy(
  int type,
  org.omg.CORBA.Any value)
  throws
  org.omg.CORBA.PolicyError;
  登録済み PolicyFactory の PolicyType に CORBA.ORB.create_policy()が呼び出されると，ORB は，登録済み PolicyFactory インスタンスの create_policy()を呼び出します。create_policy()メソッドは，次に，指定した CORBA.Any に対応する値を持つ CORBA.Policy から派生したインタフェースのインスタンスを返します。失敗した場合は，CORBA.ORB.create_policy()で示す例外が発生します。
  - type
    作成するポリシーの型を指定している CORBA.PolicyType
  - value
    CORBA.Policy を構成するためのデータを格納している CORBA.Any

10.17 RequestInfo

```
public interface RequestInfo extends
    org.omg.CORBA.LocalInterface,
    org.omg.PortableInterceptor.RequestInfoOperations,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，ClientRequestInfo と ServerRequestInfo の派生元となるベースクラスです。各インタセプトポイントに与えられるオブジェクトを通じて，インタセプタはリクエスト情報にアクセスできます。クライアント側とサーバ側のインタセプトポイントでは扱う情報が異なるため，情報オブジェクトは二つあります。ClientRequestInfo はクライアント側インタセプトポイントに渡され，ServerRequestInfo は，サーバ側インタセプトポイントに渡されます。しかし，両方に共通する情報があるため，どちらも共通インタフェースの RequestInfo を継承しています。

10.17.1 import 文

コード内に「`import org.omg.PortableInterceptor.*;`」と記述してください。

10.17.2 RequestInfo のメソッド

- public int request_id();

  このメソッドは，アクティブなリクエストまたは応答シーケンスを一意に識別する ID を返します。リクエストまたは応答シーケンスが解決された場合は，ID が再使用されることもあります。

  注

  この ID は GIOP の request_id とは異なります。GIOP がトランスポート機能として使用されている場合，この ID と GIOP の request_id は必ずしも一致するとは限りません。また，これらが一致している必要はありません。

- public java.lang.String operation();

  このメソッドは，呼び出されているオペレーションの名前を返します。

- public Parameter[ ] arguments();

  このメソッドは，呼び出されているオペレーションのパラメタを格納する，Parameter 型の配列を返します。パラメタがない場合，この属性は長さが 0 のシーケンスとなります。

  すべての環境でこれらのパラメタにアクセスできるとは限りません。例えば，Java のポータブルバインディングでは，これらのパラメタは使用できません。アクセスできない環境でこの属性にアクセスしようとすると，標準マイナーコード 1 の NO_RESOURCES 例外が発生します。

- public org.omg.CORBA.TypeCode[ ] exceptions();

このメソッドは，オペレーション呼び出しで発生するユーザ例外の TypeCode を定義する CORBA.TypeCode 型の配列を返します。ユーザ例外がない場合，この属性は長さが 0 のシーケンスとなります。

すべての環境で例外リストにアクセスできるとは限りません。例えば，Java のポータブルバインディングでは，この例外リストは使用できません。アクセスできない環境でこの属性にアクセスしようとすると，標準マイナーコード 1 の NO_RESOURCES 例外が発生します。

- public java.lang.String[ ] contexts();

  このメソッドは，オペレーション呼び出しで渡される可能性のあるコンテキストを定義する java.lang.String の配列を返します。コンテキストがない場合，この属性は長さが 0 のシーケンスとなります。

  すべての環境でコンテキストリストにアクセスできるとは限りません。例えば，Java のポータブルバインディングでは，このコンテキストリストは使用できません。アクセスできない環境でこの属性にアクセスしようとすると，標準マイナーコード 1 の NO_RESOURCES 例外が発生します。

- public java.lang.String[ ] operation_context();

  このメソッドは，リクエストで送信されるコンテキストを定義する java.lang.String の配列を返します。すべての環境でコンテキストにアクセスできるとは限りません。例えば，Java のポータブルバインディングでは，このコンテキストは使用できません。アクセスできない環境でこの属性にアクセスしようとすると，標準マイナーコード 1 の NO_RESOURCES 例外が発生します。

- public org.omg.CORBA.Any result();

  このメソッドは，オペレーション呼び出しの結果を CORBA.Any 形式で返します。オペレーションのリターン型が void の場合，この属性は，TCKind 値が tk_void または値なしのタイプコードを格納する CORBA.Any となります。

  すべての環境で呼び出し結果にアクセスできるとは限りません。例えば，Java のポータブルバインディングでは，この呼び出し結果は使用できません。アクセスできない環境でこの属性にアクセスしようとすると，標準マイナーコード 1 の NO_RESOURCES 例外が発生します。

- public boolean response_expected();

  このメソッドは，応答が期待されているかどうかを示すブール値を返します。

  クライアントでは，response_expected が false の場合，応答が返らないため，receive_reply()を呼び出せません。例外が発生した場合以外は，receive_other()を呼び出します。例外が発生した場合は，receive_exception()を呼び出します。

- public short sync_scope();

  このメソッドは，Messaging 仕様で定義されている属性を返します。この属性は，response_expected が false のときだけ有効です。response_expected が true の場合，sync_scope()の値は不定です。この属性は，クライアントに制御が戻る前のリクエストの進捗状況を表します。この属性の値を次に示します。

  - Messaging.SYNC_NONE
  - Messaging.SYNC_WITH_TRANSPORT
  - Messaging.SYNC_WITH_SERVER

- Messaging.SYNC_WITH_TARGET

サーバでは，すべてのスコープで，ターゲットオペレーション呼び出しの戻り値から応答が生成されますが，その応答はクライアントには返されません。クライアントに返されなくても応答は生成されているので，通常のサーバ側インタセプトポイント，つまり，receive_request_service_contexts()，receive_request()，send_reply()，または send_exception()に従います。

SYNC_WITH_SERVER と SYNC_WITH_TARGET に関しては，ターゲットオペレーションが呼び出される前に，サーバは空の応答をクライアントに返却します。サーバ側インタセプタはこの応答を受け取りません。

- public short reply_status();

このメソッドは，オペレーションの呼び出し結果の状態を表す属性を返します。この属性の値を次に示します。

- PortableInterceptor.SUCCESSFUL =0
- PortableInterceptor.SYSTEM_EXCEPTION =1
- PortableInterceptor.USER_EXCEPTION =2
- PortableInterceptor.LOCATION_FORWARD =3
- PortableInterceptor.TRANSPORT_RETRY =4

クライアント側では，この属性の値は次のようになります。

- インタセプトポイント receive_reply の中では，この属性の値は SUCCESSFUL だけです。
- インタセプトポイント receive_exception の中では，この属性の値は SYSTEM_EXCEPTION または USER_EXCEPTION です。
- インタセプトポイント receive_other の中では，この属性の値は SUCCESSFUL，LOCATION_FORWARD，または TRANSPORT_RETRY のどれかです。SUCCESSFUL は，非同期リクエストが正常にリターンしたことを意味します。LOCATION_FORWARD は，LOCATION_FORWARD 状態が応答で返却されたことを意味します。TRANSPORT_RETRY は，トランスポート機能がリトライを指示したことを意味します。例えば，NEEDS_ADDRESSING_MODE 状態の GIOP 応答が該当します。

サーバ側では，この属性の値は次のようになります。

- インタセプトポイント send_reply の中では，この属性の値は SUCCESSFUL だけです。
- インタセプトポイント send_exception の中では，この属性の値は SYSTEM_EXCEPTION または USER_EXCEPTION です。
- インタセプトポイント send_other の中では，この属性の値は SUCCESSFUL または LOCATION_FORWARD です。SUCCESSFUL は，非同期リクエストが正常にリターンしたことを意味します。LOCATION_FORWARD は，LOCATION_FORWARD 状態が応答で返却されたことを意味します。

- public org.omg.CORBA.Object forward_reference();

reply_status()が LOCATION_FORWARD を返す場合，このメソッドは，リクエストフォワード先オブジェクトを返します。フォワードされるリクエストが実際に発生するかどうかはわかりません。

- public org.omg.CORBA.Any **get_slot**(
  int **id**)
  throws
  InvalidSlot;
  このメソッドは，リクエストのスコープにある Current の指定スロットからデータを CORBA.Any 形式で返します。
  指定したスロットが設定されていない場合，TCKind 値が tk_null のタイプコードを格納する CORBA.Any が返されます。
  ID が割り当てられていないスロットを表す場合は，InvalidSlot 例外が発生します。
  スロットと Current の詳細については，「10.12　IORInterceptor」を参照してください。
  - id
    取得するスロットの SlotId
- public org.omg.IOP.ServiceContext **get_request_service_context**(
  int **id**);
  このメソッドは，リクエストに対応づけられたサービスコンテキストのうち，指定した ID のサービスコンテキストのコピーを返します。
  リクエストのサービスコンテキストに，指定した ID のエントリが含まれていない場合は，標準マイナーコード 26 の BAD_PARAM 例外が発生します。
  - id
    取得するサービスコンテキストの ID
- public org.omg.IOP.ServiceContext **get_reply_service_context**(
  int **id**);
  このメソッドは，応答に対応づけられたサービスコンテキストのうち，指定した ID のサービスコンテキストのコピーを返します。
  応答のサービスコンテキストに，指定した ID のエントリが含まれていない場合は，標準マイナーコード 26 の BAD_PARAM 例外が発生します。
  - id
    取得するサービスコンテキストの ID

10.18 ServerRequestInfo

```
public interface ServerRequestInfo extends RequestInfo,
    org.omg.CORBA.LocalInterface,
    org.omg.PortableInterceptor.ServerRequestInfoOperations,
    org.omg.CORBA.portable.IDLEntity
```

ServerRequestInfo は，RequestInfo から派生したインタフェースです。サーバ側インタセプトポイントに渡されます。ServerRequestInfo のメソッドは，一部のインタセプトポイントでは有効ではありません。

次の表に，属性またはメソッドの有効性を示します。無効の属性またはメソッドにアクセスすると，標準マイナーコード 14 の BAD_INV_ORDER 例外が発生します。

表 10–3 ServerRequestInfo の有効性（Java）

	receive_request_service_contexts	receive_request	send_reply	send_exception	send_other
request_id	○	○	○	○	○
operation	○	○	○	○	○
arguments	×	○※1	○	×※2	×※2
exceptions	×	○	○	○	○
contexts	×	○	○	○	○
operation_context	×	○	○	×	×
result	×	×	○	×	×
response_expected	○	○	○	○	○
sync_scope	○	○	○	○	○
reply_status	×	×	○	○	○
forward_reference	×	×	×	×	○※2
get_slot	○	○	○	○	○
get_request_service_context	○	○	○	○	○
get_reply_service_context	×	×	○	○	○
sending_exception	×	×	×	○	×
object_id	×	○	○	○※3	○※3
adapter_id	×	○	○	○※3	○※3
target_most_derived_interface	×	○	×※4	×※4	×※4
get_server_policy	○	○	○	○	○
set_slot	○	○	○	○	○

	receive_request_service_contexts	receive_request	send_reply	send_exception	send_other
target_is_a	×	○	×※4	×※4	×※4
add_reply_service_context	○	○	○	○	○

（凡例）

○：有効，×：無効

注※1

ServerRequestInfo が receive_request()に渡される場合，in，inout，または out の各パラメタのリストにエントリがありますが，使用できるのは in パラメタと inout パラメタだけです。

注※2

reply_status()が LOCATION_FORWARD を返さない場合，この属性にアクセスすると標準マイナーコード 14 の BAD_INV_ORDER 例外が発生します。

注※3

Servant Locator でロケーションのフォワードや，例外が発生した場合，この属性またはメソッドは，当該インタセプトポイントで使用できないことがあります。使用できない場合は，標準マイナーコード 1 の NO_RESOURCES 例外が発生します。

注※4

このメソッドは，当該インタセプトポイントで使用できません。

それは，必要な情報を取得するためにはターゲットオブジェクトのサーバントにアクセスする必要がありますが，この時点で ORB はこのサーバントを使用できないためです。例えば，ServantLocator を使用する POA がオブジェクトのアダプタである場合，ORB は，ServantLocator.postinvoke()を呼び出したあとでインタセプトポイントを呼び出します。

10.18.1 import 文

コード内に「`import org.omg.PortableInterceptor.*;`」と記述してください。

10.18.2 ServerRequestInfo のメソッド

- public org.omg.CORBA.Any sending_exception();

  このメソッドは，クライアントに返される例外を格納するデータを CORBA.Any の形式で返します。

  CORBA.Any に挿入できないユーザ例外の場合，例えば，未知の例外やバインディングで TypeCode が提供されない場合などは，この属性は標準マイナーコード 1 の UNKNOWN システム例外を格納する CORBA.Any となります。

- public byte[ ] object_id();

このメソッドは，オペレーション呼び出しのターゲットを表す不透明な object_id を CORBA.OctetSequence の形式で返します。

- public byte[ ] **adapter_id**();
  このメソッドは，オブジェクトアダプタの不透明な識別子を CORBA.OctetSequence の形式で返します。
- public java.lang.String **target_most_derived_interface**();
  このメソッドは，サーバントのいちばん派生したインタフェースの RepositoryID を返します。
- public org.omg.CORBA.Policy **get_server_policy**(
  int **type**);
  このメソッドは，オペレーションに対して有効なポリシーのうち，指定したポリシー型を持つポリシーを返します。register_policy_factory で登録された型を持つポリシーだけが，CORBA.Policy オブジェクトとして返されます。
  指定した型のポリシーが register_policy_factory で登録されていない場合は，標準マイナーコード 3 の INV_POLICY 例外が発生します。詳細については，「10.14.3　ORBInitInfo のメソッド」の register_policy_factory()メソッドを参照してください。
  - type
    取得するポリシーを指定する CORBA.PolicyType
- public void **set_slot**(
  int **id**, org.omg.CORBA.Any **data**)
  throws
  InvalidSlot;
  このメソッドを使用すると，インタセプタでリクエストのスコープ内の Current のスロットにデータを設定できます。スロットにすでにデータが設定されている場合，既存データは上書きされます。割り当てられていないスロットを表す ID を指定した場合は，InvalidSlot 例外が発生します。スロットと Current の詳細については，「10.12　IORInterceptor」を参照してください。
  - id
    スロットの SlotId
  - data
    指定したスロットに設定する，CORBA.Any オブジェクト形式のデータ
- public boolean **target_is_a**(
  java.lang.String **id**);
  このメソッドは，指定した RepositoryId がサーバントの場合は true を返し，そうでない場合は false を返します。
  - id
    サーバントが，この CORBA.RepositoryId であるかどうかを呼び出し元が調べます。
- public void **add_reply_service_context**(

org,omg.IOP.ServiceContext service_context,boolean replace);

このメソッドを使用すると，インタセプタでサービスコンテキストを応答に登録できます。サービスコンテキストの順序の宣言はありません。登録された順序でサービスコンテキストが表示される場合も，されない場合もあります。

- service_context
  応答に登録する IOP.ServiceContext
- replace
  指定した ID のサービスコンテキストがすでに存在する場合のメソッドの動作。false の場合は，標準マイナーコード 15 の BAD_INV_ORDER 例外が発生します。true の場合は，既存のコンテキストが新しいコンテキストに置き換えられます。

10.19　ServerRequestInterceptor

```
public interface ServerRequestInterceptor extends Interceptor,
    org.omg.CORBA.LocalInterface,
    org.omg.PortableInterceptor.ServerRequestInterceptorOperations,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，ユーザ定義サーバ側インタセプタを継承するときに使用します。ServerRequestInterceptor インスタンスは，VisiBroker ORB に登録されます。詳細については，「10.4 Codec」を参照してください。

10.19.1　import 文

コード内に「`import org.omg.PortableInterceptor.*;`」と記述してください。

10.19.2　ServerRequestInterceptor のメソッド

- public void receive_request_service_contexts(
  ServerRequestInfo ri)
  throws
  ForwardRequest;
  このインタセプトポイントでは，インタセプタは受信したリクエストからサービスコンテキスト情報を取得し，Current のスロットに転送する必要があります。
  このインタセプトポイントは，サーバントマネージャよりも先に呼び出されます。オペレーションのパラメタはまだこの時点では使用できません。このインタセプトポイントは，ターゲットオペレーションの呼び出しと同じスレッドで実行される場合も，実行されない場合もあります。
  このインタセプトポイントでは，システム例外が発生する場合があります。システム例外が発生した場合，ほかのインタセプタの receive_request_service_contexts()インタセプトポイントは呼び出されません。フロースタックからインタセプタが取り出され，そのインタセプタの send_exception()インタセプトポイントが呼び出されます。
  このインタセプトポイントでは，ForwardRequest 例外が発生する場合もあります。インタセプトでこの例外が発生した場合，ほかのインタセプタの receive_request_service_contexts()メソッドは呼び出されません。フロースタックからインタセプタが取り出され，そのインタセプタの send_other インタセプトポイントが呼び出されます。ForwardRequest 例外については，「10.8　ForwardRequest」を参照してください。
  - ri
    インタセプタが使用する ServerRequestInfo インスタンス
- public void receive_request(
  ServerRequestInfo ri)

throws

ForwardRequest;

このインタセプトポイントを使用すると，メソッドのパラメタを含むすべての情報が使用できる状態になったあとで，リクエスト情報をインタセプタで照会できます。このインタセプトポイントは，ターゲットオペレーションの呼び出しと同じスレッドで実行されます。

DSI モデルでは，ユーザコードが arguments()を呼び出したときに最初にパラメタが使用できるようになるため，arguments()の内側から receive_request()が呼び出されます。DSI モデルでは，arguments()を呼び出さないようにすることもできます。

ターゲットオペレーションは arguments()より先に set_exception()を呼び出すことがあります。ORB は，arguments()と set_exception()のどちらが呼び出されても，receive_request()は 1 回だけ呼び出されることを保障します。

set_exception()で receive_request()が呼び出された場合，arguments()のリクエストは，標準マイナーコード 1 の NO_RESOURCES 例外となります。

このインタセプトポイントでは，システム例外が発生する場合があります。システム例外が発生した場合，ほかのインタセプタの receive_request()メソッドは呼び出されません。フロースタックからインタセプタが取り出され，そのインタセプタの send_exception()インタセプトポイントが呼び出されます。

このインタセプトポイントでは，ForwardRequest 例外が発生する場合もあります。インタセプトでこの例外が発生した場合，ほかのインタセプタの receive_request()メソッドは呼び出されません。フロースタックからインタセプタが取り出され，そのインタセプタの send_other インタセプトポイントが呼び出されます。ForwardRequest 例外については，「10.8　ForwardRequest」を参照してください。

- ri

  インタセプタが使用する ServerRequestInfo インスタンス

- public void **send_reply**(

  ServerRequestInfo **ri**);

  このインタセプトポイントを使用すると，ターゲットオペレーションが呼び出されてから応答がクライアントに返されるまでの間に，インタセプタで応答情報の照会と，応答サービスコンテキストの修正ができます。このインタセプトポイントは，ターゲットオペレーションと同じスレッドで実行されます。

  このインタセプトポイントでは，システム例外が発生する場合があります。システム例外が発生した場合，ほかのインタセプタの send_reply()インタセプトポイントは呼び出されません。フロースタックにあるインタセプタの send_exception()インタセプトポイントが呼び出されます。

  - ri

    インタセプタが使用する ServerRequestInfo インスタンス

- public void **send_exception**(

  ServerRequestInfo **ri**)

  throws

  ForwardRequest;

このインタセプトポイントは，例外が発生したときに呼び出されます。このインタセプトポイントを使用すると，例外がクライアントに通知される前にインタセプタでリクエスト情報の照会と，応答サーバコンテキストの修正ができます。このインタセプトポイントは，ターゲットオペレーションと同じスレッドで実行されます。このインタセプトポイントでは，システム例外が発生する場合があります。システム例外が発生した場合は，フロースタックから取り出された一連のインタセプタが，send_exception()呼び出し時に受け取る例外が変更されます。クライアントに通知される例外は，インタセプタが通知する最後の例外です。ほかのインタセプタが例外を変更しなければ，元の例外が通知されます。

このインタセプトポイントでは，ForwardRequest 例外を発生させることができます。インタセプタでこの例外が発生した場合，ほかのインタセプタの send_exception()メソッドは呼び出されません。フロースタックからインタセプタが取り出され，そのインタセプタの send_other インタセプトポイントが呼び出されます。ForwardRequest 例外については，「10.8　ForwardRequest」を参照してください。

- ri

  インタセプタが使用する ServerRequestInfo インスタンス

- public void **send_other**(

  ServerRequestInfo **ri**)

  throws

  ForwardRequest;

  このインタセプトポイントを使用すると，リクエストの結果が，正常な応答でも例外でもない場合に使用できる情報をインタセプタで照会できます。それは，リクエストがリトライになる場合（例えば，LOCATION_FORWARD 状態で GIOP Reply を受信した場合）などです。このインタセプトポイントでは，ターゲットオペレーションと同じスレッドで実行されます。

  このインタセプトポイントでは，システム例外が発生する場合があります。システム例外が発生した場合，ほかのインタセプタの send_other インタセプトポイントは呼び出されません。フロースタックにあるインタセプタの send_exception()インタセプトポイントが呼び出されます。

  このインタセプトポイントでは，FowardRequest 例外が発生する場合もあります。インタセプタがこの例外を出力した場合，ForwardRequest 例外が提供する新しい情報で一連のインタセプタの send_other()メソッドが呼び出されます。ForwardRequest 例外については，「10.8 ForwardRequest」を参照してください。

  - ri

    インタセプタが使用する ServerRequestInfo インスタンス

11 VisiBroker 4.x インタセプタおよびオブジェクトラッパーのインタフェースとクラス（Java）

この章では，VisiBroker 4.x インタセプタとオブジェクトラッパーに対して使用する，Borland Enterprise Server VisiBroker のインタフェースとクラスについて，Java 言語でのインタフェースを説明します。VisiBroker 4.x インタセプタおよびオブジェクトラッパーの生成方法と使用方法については，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「VisiBroker 4.x インタセプタの使用」および「オブジェクトラッパーの使用」の記述を参照してください。

11.1 概要

VisiBroker 4.x インタセプタは，VisiBroker 4.x に定義され，インプリメントされているインタセプタです。ポータブルインタセプタと同様に，Borland Enterprise Server VisiBroker の ORB サービスに ORB の実行フローを受け取る機能を提供します。

VisiBroker 4.x インタセプタ，およびオブジェクトラッパーの使用方法の詳細については，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「VisiBroker 4.x インタセプタの使用」および「オブジェクトラッパーの使用」の記述を参照してください。

VisiBroker 4.x インタセプタには次の 3 種類があります。

- クライアントインタセプタ

  システムレベルのインタセプタです。トランザクションやセキュリティなどの ORB サービスのフックを提供し，クライアントの ORB に処理させるために使用できます。

- サーバインタセプタ

  システムレベルのインタセプタです。トランザクションやセキュリティなどの ORB サービスのフックをサーバの ORB に処理させるために使用できます。

- オブジェクトラッパー

  ユーザレベルのインタセプタです。簡易なトレースとデータキャッシュができるような，スタブとスケルトンの呼び出しをインタセプトするための簡易な機構を提供します。

11.2 インタセプタマネージャ

インタセプタのインストールと管理は，インタセプタマネージャで実行します。InterceptorManager インタフェースは，すべてのグローバルインタセプタを管理するためのグローバルインタセプタマネージャです。

グローバルインタセプタは，ローカライズされたインタセプタをインストールするために，インタセプタマネージャが追加されて渡されることがあります。例えば，各 POA のインタセプタは POAInterceptorManager を使用します。

グローバルインタセプタマネージャのインスタンスである InterceptorManager は，文字列「VisiBrokerInterceptorControl」をパラメタとして渡すときに，ORB.resolve_initial_references を使用して取得できます。この値は，ORB が管理モードのとき，つまり ORB の初期化中にだけ使用できます。

11.3 IOR テンプレート

POALifeCycleInterceptor.create の呼び出し中に，インタセプタに加えて，インターオペラブルオブジェクトリファレンス（IOR）テンプレートを直接 POAInterceptorManager インタフェースで修正できます。IOR テンプレートは完全な IOR 値ですが，type_id が未設定で，GIOP ProfileBody 値のオブジェクトキーはすべて不完全なものです。IORCreationInterceptors の呼び出し前に，POA は type_id を設定してテンプレートのオブジェクトキーを記述します。

11.4 InterceptorManager

```
public interface Interceptor InterceptorManager
```

ほかのすべてのインタセプタマネージャは，このインタフェースから継承します。インタセプタマネージャは，インタセプタのインストールと削除を管理するために使用するインタフェースです。

11.5 InterceptorManagerControl

```
public interface InterceptorManagerControl
```

このインタフェースは，関連のあるインタセプタマネージャを一括管理するために使用します。

11.5.1 import 文

コード内に「`import com.inprise.vbroker.interceptor.*;`」と記述してください。

11.5.2 InterceptorManagerControl のメソッド

- public com.inprise.vbroker.interceptor.InterceptorManager
  get_manager(java.lang.string name)
  ORB は，このメソッドを呼び出して InterceptorManager のインスタンスを取得します。
  InterceptorManager は，マネージャを識別する文字列を返します。
  - name
    インタセプタの名前

11.6 BindInterceptor

```
public interface BindInterceptor
```

このインタフェースは，ユーザ独自のインタセプタを派生させて，クライアントアプリケーションまたはサーバアプリケーションのバインドイベントとリバインドイベントを処理させるために使用します。

バインドインタセプタは，バインドの前後にクライアント側で呼び出すグローバルインタセプタです。

バインド中に例外が発生した場合は，チェーンに登録されている残りのインタセプタは呼ばないで，それ以降のインタセプタをチェーンから削除します。bind_succeeded または bind_failed 中に発生した例外は無視されます。

11.6.1 import 文

コード内に「`import com.inprise.vbroker.interceptor.*;`」と記述してください。

11.6.2 BindInterceptor のメソッド

- public com.inprise.vbroker.IOP.IORValue bind(
  com.inprise.vbroker.IOP.IORValue ior,
  org.omg.CORBA.Object obj,boolean rebind,
  com.inprise.vbroker.interceptor.Closure closure)
  このメソッドは，すべての ORB バインドオペレーション中に呼び出されます。
  新規 IOR を使用してバインドオペレーションを継続する場合，新規 IOR を返します。それ以外の場合は null を返し，元の IOR を使用してバインドを続行します。渡されたパラメタと同じ IOR を返すことは不正なので，バインド時に例外が発生します。
  - ior
    クライアントがバインドしているサーバオブジェクトの IOR
  - obj
    サーバにバインドしているクライアントオブジェクト。このオブジェクトはこの時点では適切に初期化されていないので，オブジェクトのオペレーションはしないでください。しかし，このオブジェクトをデータ構造体に保存してバインド完了後に使用できます。
  - rebind
    リバインドを試行することを指定します。
  - closure
    バインドオペレーション用の新しい closure オブジェクト。closure は，bind_failed，または bind_succeeded に対応した呼び出しで使用します。

- public com.inprise.vbroker.IOP.IORValue **bind_failed**(
  com.inprise.vbroker.IOP.IORValue **ior**,
  org.omg.CORBA.Object **obj**,
  com.inprise.vbroker.interceptor.Closure **closure**)
  このメソッドは，バインドオペレーションが失敗したときに呼び出されます。
  この IOR に対してリバインドを試行する場合，新規 IOR を返します。それ以外の場合は null を返し，リバインドは試行されません。
  - ior
    バインドオペレーションが失敗したサーバオブジェクトの IOR
  - obj
    サーバにバインドしているクライアントオブジェクト
  - closure
    前回のバインド呼び出しで与えられた closure オブジェクト
- public void **bind_succeeded**(
  com.inprise.vbroker.IOP.IORValue **ior**,
  org.omg.CORBA.Object **obj**,int **profileIndex**,
  com.inprise.vbroker.interceptor.InterceptorManagerControl
  interceptorControl,
  com.inprise.vbroker.interceptor.Closure **closure**)
  このメソッドは，バインドオペレーションが成功すると呼び出されます。
  - ior
    バインドオペレーションが成功したサーバオブジェクトの IOR
  - obj
    サーバにバインドしているクライアントオブジェクト
  - profileIndex
    実際にバインドが成功したプロファイルのインデックス
  - interceptorControl
    このマネージャがマネージャ種別一覧を提供します。
  - closure
    前回のバインド呼び出しで与えられた closure オブジェクト
- public void **exception_occurred**,(
  com.inprise.vbroker.IOP.IORValue **ior**,
  org.omg.CORBA.Object **obj**,
  org.omg.CORBA.Environment **env**,
  com.inprise.vbroker.interceptor.Closure **closure**)

このメソッドは，bind で例外が発生すると呼び出されます。

- ior
  成功したバインドオペレーションの対象サーバオブジェクトの IOR
- obj
  サーバにバインドしているクライアントオブジェクト
- env
  発生した例外の情報
- closure
  前回のバインド呼び出しで与えられた closure オブジェクト

11.7 BindInterceptorManager

```
public interface BindInterceptorManager extends
    com.inprise.vbroker.interceptor.InterceptorManager
```

このインタフェースは，BindInterceptor を追加するために使用します。

11.7.1 import 文

コード内に「`import com.inprise.vbroker.interceptor.*;`」と記述してください。

11.7.2 BindInterceptorManager のメソッド

- public void **add**(
  com.inprise.vbroker.interceptor.BindInterceptor **interceptor**)
  このメソッドを使用して BindInterceptor を追加します。
  - interceptor
    追加するインタセプタ

11.8 ClientRequestInterceptor

```
public interface ClientRequestInterceptor
```

このインタフェースは，ユーザ独自のクライアントインタセプタを派生し，オーバーライド対象のメソッドにインプリメンテーションを提供するために使用します。

クライアントリクエストインタセプタは，バインドインタセプタの bind_succeeded 呼び出し中にインストールできます。また，コネクションが確立されている間はアクティブ状態を維持します。ユーザが派生したクラスに定義しているメソッドは，オペレーションリクエストの準備時，送信時，返信メッセージの受信時，または例外発生時に ORB によって呼び出されます。

11.8.1 import 文

コード内に「`import com.inprise.vbroker.interceptor.*;`」と記述してください。

11.8.2 ClientRequestInterceptor のメソッド

- public void **preinvoke_premarshal**(
  org.omg.CORBA.Object **target**,
  java.lang.String **operation**,
  com.inprise.vbroker.IOP.ServiceContextListHolder
  service_contexts,
  com.inprise.vbroker.interceptor.Closure **closure**)
  このメソッドは，リクエストごとに，それらがマーシャリングされる前に ORB に呼び出されます。このインタセプタで例外が発生した場合は，即時にリクエストは終了させられます。インタセプタのチェーンでは，処理済みのインタセプタだけがチェーンに残ります。その例外を起こしたリクエストは送信されないで，exception_occurred( )がチェーンに残ったすべてのインタセプタに対して呼び出されます。
  - target
    サーバへのバインドを試みたクライアントオブジェクト
  - operation
    呼び出すオペレーションの名前を識別します。
  - service_contexts
    ORB が割り当てたサービスを識別します。このサービスは OMG に規定されているものです。
  - closure
    あるインタセプタメソッドが保存したデータを格納する場合があります。その場合，このデータをほかのインタセプタメソッドがあとで取得できます。

- public void **preinvoke_postmarshal**(
  org.omg.CORBA.Object **target**,
  com.inprise.vbroker.CORBA.portable.Outstream **payload**,
  com.inprise.vbroker.interceptor.Closure **closure**)
  このメソッドは，リクエストごとに，リクエストのマーシャル処理が済んでから送信されるまでの間に呼び出されます。このメソッドで例外が発生した場合は，残りのチェーンは呼び出されません。該当するリクエストはサーバに送信されません。その後，exception_occurred( )はインタセプタのチェーン全体に呼び出されます。
  - target
    サーバへのバインドを試みたクライアントオブジェクト
  - payload
    マーシャリング済みバッファ
  - closure
    あるインタセプタメソッドが保存したデータを格納する場合があります。その場合，このデータをほかのインタセプタメソッドがあとで取得できます。
- public void **postinvoke**(
  org.omg.CORBA.Object **target**,
  com.inprise.vbroker.IOP.ServiceContext[ ] **service_contexts**,
  com.inprise.vbroker.CORBA.portable.InputStream **payload**,
  org.omg.CORBA.Environment **env**,
  com.inprise.vbroker.interceptor.Closure **closure**)
  リクエストの完了後，このメソッドが呼び出されます。
  - target
    サーバへのバインドを試みたクライアントオブジェクト
  - service_context
    ORB が割り当てたサービスを識別します。このサービスは OMG に規定されているものです。
  - payload
    マーシャリング済みバッファ
  - env
    発生した例外についての情報を格納します。
  - closure
    あるインタセプタメソッドが保存したデータを格納する場合があります。その場合，このデータをほかのインタセプタメソッドがあとで取得できます。
- public void **exception_occurred**(
  org.omg.CORBA.Object **target**,
  org.omg.CORBA.Environment **env**,

com.inprise.vbroker.interceptor.Closure closure)

このメソッドは，呼び出し前に例外が発生した場合に ORB が呼び出します。呼び出し後に発生した例外はすべて，postinvoke メソッドの Environment パラメタに収集されます。

11.9 ClientRequestInterceptorManager

```
public interface ClientRequestInterceptorManager extends
    com.inprise.vbroker.interceptor.InterceptorManager
```

このインタフェースは，ClientRequestInterceptor の追加と削除を実行するのに使用します。

11.9.1 import 文

コード内に「`import com.inprise.vbroker.interceptor.*;`」と記述してください。

11.9.2 ClientRequestInterceptorManager のメソッド

- public void **add**(
  com.inprise.vbroker.interceptor.ClientRequestInterceptor
  interceptor)
  ORB がこのメソッドを呼び出して ClientRequestInterceptor を追加します。
  - interceptor
    追加するインタセプタ

11.10 POALifeCycleInterceptor

```
public interface POALifeCycleInterceptor
```

このインタフェースは，POA が生成またはデストラクトされるたびに呼び出されるグローバルインタセプタです。ほかのサーバ側インタセプタはすべてグローバルインタセプタとして，または特定の POA に対してインストールできます。POALifeCycleInterceptor は，InterceptorManager インタフェースを使用してインストールします。POALifeCycleInterceptor は，POA の生成時またはデストラクト時に呼び出されます。

11.10.1 import 文

コード内に「**import com.inprise.vbroker.interceptor.*;**」と記述してください。また，「**import com.inprise.vbroker.PortableServerExt.*;**」と記述してください。

11.10.2 POALifeCycleInterceptor のメソッド

- public void create(
  org.omg.PortableServer.POA poa,
  org.omg.CORBA.PolicyListHolder policies,
  com.inprise.vbroker.IOP.IORValueHolder iorTemplate,
  com.inprise.vbroker.interceptor.InterceptorManagerControl
  poaAdmin)
  このメソッドは，新規 POA が create_POA の呼び出しまたは AdapterActivator によって生成されたときに，呼び出されます。AdapterActivator の場合，インタセプタは，unknown_adapter メソッドが AdapterActivator から正常にリターンしたあとにだけ呼び出されます。
  - poa
    作成する POA
  - policies
    該当する POA に指定したポリシーの一覧
  - iorTemplate
    IOR テンプレートは，type_id が未設定の完全 IOR 値であり，GIOP.ProfileBodyValue のオブジェクトキーがすべて不完全なものです。
  - poaAdmin
    ほかのインタセプタマネージャを取得するために使用する InterceptorManagerControl
- public void destroy(

org.omg.PortableServer.POA poa)

このメソッドは，POA のデストラクト時，そのすべてのオブジェクトがエーテライズされている場合に呼び出されます。create が同じ名前の POA に対して再び呼び出される前に，destroy がすべてのインタセプタに対して必ず呼び出されるようにしなければなりません。destroy オペレーションにシステム例外が発生してもそのシステム例外は無視され，残りのインタセプタは引き続き呼び出されます。

- poa
  デストラクト対象の POA

11.11　POALifeCycleInterceptorManager

```
public interface POALifeCycleInterceptorManager
```

このインタフェースは，POALifeCycleInterceptor を登録するために使用します。

11.11.1　import 文

コード内に「`import com.inprise.vbroker.interceptor.*;`」と記述してください。また，「`import com.inprise.vbroker.PortableServerExt.*;`」と記述してください。

11.11.2　POALifeCycleInterceptorManager のメソッド

- public void **add**(
  POALifeCycleInterceptor **interceptor**)
  ORB がこのメソッドを呼び出して，POALifeCycleInterceptor を追加します。
  - interceptor
    追加するインタセプタ

11.12 ActiveObjectLifeCycleInterceptor

```
public interface ActiveObjectLifeCycleInterceptor
```

このインタフェースは，オブジェクトをアクティブオブジェクトマップに追加するときと，アクティブオブジェクトマップからオブジェクトを削除するときに呼び出されます。このインタフェースは，POA に RETAIN ポリシーがあるときだけ使用できます。また，このインタフェースは，POA 生成時に POALifeCycleInterceptor が POA ごとにインストールできるインタセプタです。

11.12.1 import 文

コード内に「**import com.inprise.vbroker.interceptor.*;**」と記述してください。また，「**import com.inprise.vbroker.PortableServerExt.*;**」と記述してください。

11.12.2 ActiveObjectLifeCycleInterceptor のメソッド

- public void **create**(
  byte[ ] **oid**,
  org.omg.PortableServer.Servant **servant**,
  org.omg.PortableServer.POA **adapter**)
  このメソッドは，直接 API または ServantActivator を使用して実行した（明示的または暗黙的な）呼び出しによってオブジェクトがアクティブオブジェクトマップに追加されたあとに，呼び出されます。オブジェクトリファレンスと新規アクティブオブジェクトの POA はパラメタとして渡されます。
  - oid
    活性化するサーバントのオブジェクト ID
  - servant
    活性化するサーバント
  - adapter
    サーバントを活性化した POA
- public void **destroy**(
  byte[ ] **oid**,
  org.omg.PortableServer.Servant **servant**,
  org.omg.PortableServer.POA **adapter**)
  このメソッドは，オブジェクトが非活性化されエーテライズされたあとに，呼び出されます。該当するオブジェクトのオブジェクトリファレンスと POA はパラメタとして渡されます。
  - oid

デストラクトするサーバントのオブジェクト ID

- servant
  デストラクトするサーバント
- adapter
  サーバントをデストラクトした POA

11.13 ActiveObjectLifeCycleInterceptorManager

```
public interface ActiveObjectLifeCycleInterceptorManager
```

このインタフェースは，ActiveObjectLifeCycleInterceptor を追加するために使用します。

11.13.1 import 文

コード内に「**import com.inprise.vbroker.interceptor.*;**」と記述してください。また，「**import com.inprise.vbroker.PortableServerExt.*;**」と記述してください。

11.13.2 ActiveObjectLifeCycleInterceptorManager のメソッド

- public void **add**(
  ActiveObjectLifeCycleInterceptor **interceptor**)
  ORB がこのメソッドを呼び出して ActiveObjectLifeCycleInterceptor を追加します。
  - interceptor
    追加するインタセプタ

11.14 ForwardRequestException

```
public interface ForwardRequestException extends
    org.omg.CORBA.UserException,
```

この例外は，ServerRequestInterceptor の preinvoke メソッドで発生させることができます。preinvoke メソッドはこの例外を発生させて，リクエストをほかのオブジェクトに転送できます。

11.14.1 ForwardRequestException の変数

- public boolean is_permanent
  ロケーションの転送が恒久的であるかどうかを指定します。
- public org.omg.CORBA.Object forward_reference
  リクエストの転送先のオブジェクトのリファレンスを提供します。

11.15 ServerRequestInterceptor

```
public interface Interceptor ServerRequestInterceptor
```

このインタフェースは，POALifeCycleInterceptor が POA 生成時に POA ごとにインストールできるインタセプタです。このインタフェースを使用して，アクセス制御，サービスコンテキストの検査と挿入，およびリクエストの応答状況の変更ができます。

11.15.1 import 文

コード内に「`import com.inprise.vbroker.interceptor.*;`」と記述してください。

11.15.2 ServerRequestInterceptor のメソッド

- public void **preinvoke**(
  org.omg.CORBA.Object **target**,
  java.lang.String **operation**,
  com.inprise.vbroker.IOP.ServiceContext[ ] **service_contexts**,
  com.inprise.vbroker.CORBA.portable.InputStream **payload**,
  com.inprise.vbroker.interceptor.Closure **closure**)
  raises (ForwardRequestException)
  リクエストがサーバ側に到着するたびに，ORB がこのメソッドを呼び出します。このインタセプタで例外が発生した場合，リクエストはすぐに終了させられます。このメソッドは，Servant Locator の呼び出し前に呼び出されるため，サーバントは利用できないことがあります。
  - target
    リクエスト呼び出し対象のオブジェクト
  - operation
    呼び出すオペレーションの名前を識別します。
  - service_contexts
    ORB が割り当てたサービスをすべて識別します。このサービスは OMG に規定されているものです。
  - payload
    マーシャリング済みバッファ
  - closure
    あるインタセプタメソッドが保存したデータを格納する場合があります。このデータは，ほかのインタセプタメソッドがあとで取得できます。
- public void **postinvoke_premarshal**(

org.omg.CORBA.Object **target**,
com.inprise.vbroker.IOP.ServiceContextListHolder
service_contexts,
org.omg.CORBA.Environment **env**,
com.inprise.vbroker.interceptor.Closure **closure**)
このメソッドは，サーバントにリクエストを送り，その応答をマーシャリングする前に呼び出されます。ここで発生した例外には，チェーンを中断することによって対処します。exception_occurred( ) がチェーンのすべてのインタセプタに対して呼び出されます。

- target
  リクエスト呼び出し対象のオブジェクト
- service_contexts
  ORB が割り当てたサービスをすべて識別します。このサービスは OMG に規定されているものです。
- env
  発生した例外の情報を格納します。
- closure
  あるインタセプタメソッドが保存したデータを格納する場合があります。このデータは，ほかのインタセプタメソッドがあとで取得できます。

- public void **postinvoke_postmarshal**(
  org.omg.CORBA.Object **target**,
  com.inprise.vbroker.CORBA.portable.OutputStream **payload**,
  com.inprise.vbroker.interceptor.Closure **closure**)
  このメソッドは，応答をマーシャリングしてから，その応答をクライアントに送信する前に呼び出されます。ここで発生した例外は無視されます。チェーン全体が呼び出されるように保証されます。
  このメソッドは，ServantLocator の呼び出し後に呼び出されます。このメソッドに発生した例外は，アプリケーションに発生した例外を置換します。一方向呼び出しでは，リクエストが正常に送信されたあとで呼び出されます。
  - target
    リクエスト呼び出し対象のオブジェクト
  - payload
    マーシャリング済みバッファ
  - closure
    あるインタセプタメソッドが保存したデータを格納する場合があります。このデータは，ほかのインタセプタメソッドがあとで取得できます。
- public void **exception_occurred**(
  org.omg.CORBA.Object **target**,
  org.omg.CORBA.Environment **env**,

com.inprise.vbroker.interceptor.Closure closure)

このメソッドは，例外発生時にリクエスト処理のどこからでも呼び出せます。この呼び出し中に発生した例外によって，該当する環境の既存の例外が置き換えられなければいけません。

- target
  リクエスト呼び出し対象のオブジェクト
- env
  発生した例外の情報を格納します。
- closure
  あるインタセプタメソッドが保存したデータを格納する場合があります。このデータは，ほかのインタセプタメソッドがあとで取得できます。

11.16 ServerRequestInterceptorManager

```
public interface ServerRequestInterceptorManager extends
    com.inprise.vbroker.interceptor.InterceptorManager
```

このインタフェースは，ServerRequestInterceptor を追加するために使用します。

11.16.1 import 文

コード内に「`import com.inprise.vbroker.interceptor.*;`」と記述してください。

11.16.2 ServerRequestInterceptorManager のメソッド

- public void add(
  com.inprise.vbroker.interceptor.ServerRequestInterceptor
  interceptor)
  ORB がこのメソッドを呼び出して，ServerRequestInterceptor を追加します。
  - interceptor
    追加するインタセプタ

11.17 IORCreationInterceptor

```
public interface PortableServerExt.IORCreationInterceptor
```

このインタフェースは，POA 生成時に POALifeCycleInterceptor が POA ごとにインストールできるインタセプタです。このインタセプタを使用して，プロファイルやコンポーネントを追加することによって，IOR を修正できます。このインタフェースは通常，トランザクションやファイアウォールなどのサービスをサポートするために使用します。

このインタセプタに加えて，POA ごとに，すべての IOR を制御する IOR テンプレートを変更することもできます。詳細については，「11.3 IOR テンプレート」を参照してください。IOR の操作がリポジトリや生成するリファレンスの OID に関連がない場合は，この IOR テンプレートによる方法を使用するときがあります。

IOR を根本的に変更することはお勧めしません。

11.17.1 import 文

コード内に「`import com.inprise.vbroker.interceptor.*;`」と記述してください。また，「`import com.inprise.vbroker.PortableServerExt.*;`」と記述してください。

11.17.2 IORCreationInterceptor のメソッド

- public void create(
  org.omg.PortableServer.POA poa,
  com.inprise.vbroker.IOP.IORValueHolder ior)
  このメソッドは，POA がオブジェクトリファレンスを生成する必要がある場合にいつでも呼び出せます。このインタセプタは，プロファイルやコンポーネントを追加したり，既存のプロファイルやコンポーネントに変更を加えることによって，IORValue を修正できます。
  - poa
    オブジェクトリファレンスの生成対象の POA
  - ior
    IOR のプロファイルやコンポーネントを変更できる IOR の Holder

11.18 IORCreationInterceptorManager

```
public interface PortableServerExt.IORCreationInterceptorManager
    extends InterceptorManager
```

このインタフェースは，IORInterceptor を追加するために使用します。

11.18.1 import 文

コード内に「**import com.inprise.vbroker.interceptor.*;**」と記述してください。また，「**import com.inprise.vbroker.PortableServerExt.*;**」と記述してください。

11.18.2 IORCreationInterceptorManager のメソッド

- public void **add**(
  IORCreationInterceptor **interceptor**)
  ORB がこのメソッドを呼び出して，IORInterceptor を追加します。
  - interceptor
    追加するインタセプタ

11.19 Location

```
enum Location
```

これは，オブジェクトラッパーをクライアント側とサーバ側のどちらに登録するかを定義します。

11.19.1 import 文

コード内に「`import com.inprise.vbroker.interceptor.*;`」と記述してください。

11.19.2 IDL の定義

```
enum Location {
     CLIENT,
     SERVER,
     BOTH
};
```

11.19.3 Location のメンバ

- CLIENT
  オブジェクトラッパーをクライアント側に登録します。
- SERVER
  オブジェクトラッパーをサーバ側に登録します。
- BOTH
  オブジェクトラッパーをクライアント側，サーバ側の両方に登録します。

11.20 Closure

```
public interface Closure extends Object
```

Closure オブジェクトは，インタセプタ群の呼び出しシーケンスの開始時に，ORB が作成するオブジェクトです。一つのシーケンスの中で呼び出されるすべてのインタセプタは，同じ Closure オブジェクトによって呼び出されます。Closure オブジェクトは，java.lang.Object 型オブジェクトであるパブリックデータフィールドを一つ保持しています。このフィールドは，状態情報を保持するためにインタセプタによって設定されます。Closure オブジェクトが生成されるシーケンスは，インタセプタの型によって異なります。

コードサンプル 11-1　Closure クラス

```
class Closure {
    java.lang.Object object;
};
```

11.21 ExtendedClosure

```
public interface ExtendedClosure extends Closure {
    public RequestInfo reqInfo;
    public InputStream payload;
}
```

このインタフェースは Closure から派生したインタフェースであり，読み取り専用属性に使用する RequestInfo を格納します。

IDL サンプル 11-1 RequestInfo

```
struct RequestInfo {
    boolean response_expected;
    unsigned long request_id;
};
```

ServerRequestInterceptor および ClientRequestInterceptor に渡された Closure オブジェクトを，サブクラス ExtendedClosure にキャストできます。ExtendedClosure を使用して，RequestInfo を抽出し，さらにその RequestInfo から request_id と response_expected を抽出できます。request_id は，リクエストに割り当てられた一意の識別子です。response_expected フラグは，リクエストが一方向呼び出しであるかどうかを識別します。

```
int my_response_expected =
    ((ExtendedClosure)closure).reqInfo.response_expected;
int my_request_id = ((ExtendedClosure)closure).reqInfo.request_id;
```

詳細については，examples/interceptor/client_server にある例を参照してください。

注

InputStream を修正する場合は，ExtendedClosure の payload パラメタを使用してください。リクエストインタセプタの payload 属性は読み取り専用なので，InputStream は変更できません。

このため，ExtendedClosure には読み書き可能な InputStream payload パラメタが用意されています。payload 属性の主な用途は，既存の InputStream を新規に置き換えられるようにすることです。

examples/interceptor/encryption の例は，ExtendedClosure の payload 属性の使用方法を示しています。この例では，暗号化された InputStream をインタセプタが解読するときに，解読されたメッセージを格納するための新規 InputStream を生成しなければいけません。ExtendedClosure は InputStream のホルダの役割を果たします。payload が新規 InputStream に割り当てられると，この InputStream はリクエストに対応づけられた InputStream となります。

11.22 ChainUntypedObjectWrapperFactory

```
public interface ChainUntypedObjectWrapperFactory extends
    com.inprise.vbroker.interceptor.
    UntypedObjectWrapperFactory
```

このインタフェースは，クライアントまたはサーバのアプリケーションが UntypedObjectWrapperFactory オブジェクトを追加または削除するために使用します。UntypedObjectWrapperFactory オブジェクトは，クライアントアプリケーションがバインドするオブジェクトごとに，またはサーバアプリケーションが生成するオブジェクトインプリメンテーションごとに UntypedObjectWrapper を生成するために使用します。

コードサンプル 11-2 ChainUntypedObjectWrapperFactory

```
enum Location (CLIENT,SERVER,BOTH );

abstract interface ChainUntypedObjectWrapperFactory :
    UntypedObjectWrapperFactory {
        void add(in UntypedObjectWrapperFactory owFactory,
                 in Location loc);
        void remove(in UntypedObjectWrapperFactory owFactory,
                    in Location loc);
        long count(in Location loc);
};
```

オブジェクトラッパーの使用についての詳細は，マニュアル「Borland Enterprise Server VisiBroker デベロッパーズガイド」の「オブジェクトラッパーの使用」の記述を参照してください。

11.22.1 import 文

コード内に「**import com.inprise.vbroker.interceptor.*;**」と記述してください。

11.22.2 ChainUntypedObjectWrapperFactory のメソッド

- public void **add**(
  com.inprise.vbroker.interceptor.UntypedObjectWrapperFactory
  owFactory,
  com.inprise.vbroker.interceptor.Location **loc**)
  このメソッドは，指定したアンタイプドオブジェクトラッパーファクトリを，クライアントまたはサーバのアプリケーションに追加します。

注

クライアント側では，オブジェクトをバインドする前にアンタイプドオブジェクトラッパーファクトリをインストールしておいてください。サーバ側では，インプリメンテーションオブジェクトを生成する前にアンタイプドオブジェクトラッパーファクトリをインストールしておいてください。

- owfactory
  追加するオブジェクトラッパーファクトリ
- loc
  オブジェクトラッパーの追加先

- public void **remove**(
  com.inprise.vbroker.interceptor.UntypedObjectWrapperFactory
  owFactory,
  com.inprise.vbroker.interceptor.Location **loc**)
  このメソッドは，指定したアンタイプドオブジェクトラッパーファクトリをクライアントまたはサーバのアプリケーションから削除します。

  注

  オブジェクトラッパーファクトリをクライアントから削除しても，すでにクライアントからバインドされている同じクラスのオブジェクトには影響しません。しかし，そのあとにバインドしたオブジェクトには影響します。

  オブジェクトラッパーファクトリをサーバから削除しても，生成済みのオブジェクトインプリメンテーションには影響しません。しかし，そのあとに生成したオブジェクトインプリメンテーションには影響します。

  - owfactory
    削除するオブジェクトラッパーファクトリ
  - loc
    オブジェクトラッパーファクトリの削除対象
- long **count**(
  com.inprise.vbroker.interceptor.Location **loc**)
  このメソッドを呼び出して，アンタイプドオブジェクトラッパーファクトリを追加します。
  - loc
    count が必要な場所

11.23　UntypedObjectWrapper

```
public interface UntypedObjectWrapper
```

このインタフェースは，アンタイプドオブジェクトラッパーをユーザのクライアントアプリケーションまたはサーバアプリケーションに派生させるために使用します。このインタフェースを使用して，アンタイプドオブジェクトラッパーを派生させる場合，クライアントアプリケーションがオペレーションリクエストを発行する前，またはサーバ側のオブジェクトインプリメンテーションがそのオペレーションリクエストを処理する前に呼び出される pre_method メソッドを定義します。また，サーバ側のオブジェクトインプリメンテーションがオペレーションリクエストを処理したあと，またはクライアントアプリケーションが応答を受信したあとに呼び出される post_method メソッドも定義します。

UntypedObjectWrapperFactory インタフェースからファクトリクラスも派生させてください。詳細については，「11.24　UntypedObjectWrapperFactory」を参照してください。
UntypedObjectWrapperFactory インタフェースは，ユーザの UntypedObjectWrapper オブジェクトを生成します。

オブジェクトラッパーの使用方法の詳細については，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「オブジェクトラッパーの使用」の記述を参照してください。

IDL サンプル 11-2　UntypedObjectWrapper

```
interface UntypedObjectWrapper {
    void pre_method(
        in string operation,
        in Object target,
        in interceptor::Closure closure
    );
   void post_method(
      in string operation,
      in Object target,
      in CORBA::Environment env,
      in interceptor::Closure closure
   );
};
```

11.23.1　UntypedObjectWrapper のメソッド

- public void **pre_method**(
  java.lang.String **operation**,
  org.omg.CORBA.Object **target**,
  com.inprise.vbroker.interceptor.Closure **closure**)
  このメソッドは，クライアント側でオペレーションリクエストが送信される前，またはサーバ側のオブジェクトインプリメンテーションがそのオペレーションリクエストを処理する前に呼び出されるメソッドです。

- operation
  リクエストされているオペレーションの名前
- target
  リクエストのターゲットオブジェクト
- closure
  オブジェクトラッパーメソッド間のデータ受け渡しに使用できるクロージャオブジェクト

- public void **post_method**(
  java.lang.String **operation**,
  org.omg.CORBA.Object **target**,
  org.omg.CORBA.Environment **env**,
  com.inprise.vbroker.interceptor.Closure **closure**)
  このメソッドは，サーバ側のオブジェクトインプリメンテーションがオペレーションリクエストを処理したあと，またはクライアント側のスタブが応答メッセージを処理する前に呼び出されるメソッドです。
  - operation
    リクエストされているオペレーションの名前
  - target
    リクエストのターゲットオブジェクト
  - env
    オペレーションリクエストの処理中に発生した可能性のある例外を反映するときに使用される Environment オブジェクト
  - closure
    オブジェクトラッパーメソッド間のデータ受け渡しに使用できるクロージャオブジェクト

11.24 UntypedObjectWrapperFactory

```
public interface UntypedObjectWrapperFactory
```

このインタフェースは，ユーザのアンタイプドオブジェクトラッパーファクトリを派生させるときに使用します。ChainUntypedObjectWrapperFactory インタフェースが提供する add メソッドを使用してユーザのアンタイプドオブジェクトラッパーファクトリを登録します。

新規オブジェクトのバインド時，またはオブジェクトインプリメンテーションの生成時に，ユーザのファクトリを使用して，クライアントアプリケーションまたはサーバアプリケーション用にユーザのアンタイプドオブジェクトラッパーのインスタンスを生成します。

IDL サンプル 11-3　UntypedObjectWrapperFactory

```
interface UntypedObjectWrapperFactory {
    UntypedObjectWrapper create(
                                in Object obj,
                                in Location loc);
};
```

11.24.1 import 文

コード内に「**import com.inprise.vbroker.interceptor.*;**」と記述してください。

11.24.2 UntypedObjectWrapperFactory のメソッド

- public com.inprise.vbroker.interceptor.UntypedObjectWrapper
  create(
  org.omg.CORBA.Object obj,
  com.inprise.vbroker.interceptor.Location loc)
  このメソッドは，ユーザ任意の型で UntypedObjectWrapper のインスタンスを生成するときに呼び出します。このメソッドをインプリメントすると，バインドされたオブジェクトまたはオブジェクトインプリメンテーションの型を検査し，そのオブジェクトラッパーの生成が必要かどうかを判定します。
  - obj
    アンタイプドオブジェクトラッパーを生成中のクライアントアプリケーションがバインドしているオブジェクト。このメソッドがサーバ側で呼び出し中の場合，このパラメタは生成中のオブジェクトインプリメンテーションを表します。
  - loc
    オブジェクトラッパーを生成する場所

12 QoS インタフェースとクラス（Java）

この章では，Borland Enterprise Server VisiBroker の Java 言語での QoS インタフェースとクラスについて説明します。

12.1 概要

QoS（Quality of Service）API の Borland Enterprise Server VisiBroker でのインプリメンテーションについて説明します。QoS API によって，ポリシーを使用して，ユーザのクライアントアプリケーションとサーバとの間のコネクションを定義したり管理したりできます。ポリシーの作成方法については，「4.9 PortableServer.POA」を参照してください。

QoS は，VisiBroker ORB レベルのポリシー，スレッドレベルのポリシー，およびオブジェクトレベルのポリシーを管理するために，それぞれ次のクラスを提供しています。

- VisiBroker ORB レベルのポリシー

  VisiBroker ORB レベルのポリシーは，ローカルの PolicyManager が取り扱います。PolicyManager によって，ポリシーを設定したり，現在のポリシーのオーバーライドを参照できます。システムデフォルトをオーバーライドする VisiBroker ORB レベルでのポリシーです。

- スレッドレベルのポリシー

  スレッドレベルのポリシーは，PolicyCurrent によって設定します。PolicyCurrent は，スレッドレベルでのポリシーのオーバーライドを参照したり設定したりするための各種オペレーションを格納しています。システムデフォルトと VisiBroker ORB レベルで設定した値をオーバーライドするスレッドレベルでのポリシーです。

- オブジェクトレベルのポリシー

  オブジェクトレベルのポリシーは，継承元 Object インタフェースの QoS オペレーションにアクセスすることによって適用できます。システムデフォルト，VisiBroker ORB レベル，およびスレッドレベルで設定した値をオーバーライドするオブジェクトレベルでのポリシーです。

12.2 PolicyManager

```
public interface org.omg.CORBA.PolicyManager extends
    org.omg.CORBA.Object,
    org.omg.CORBA.PolicyManagerOperations,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，VisiBroker ORB レベルでのポリシーオーバーライドにアクセスするために使用します。VisiBroker ORB レベルで定義されたポリシーは，任意のシステムデフォルトをオーバーライドします。マネージャスレッドに属するインスタンスには，resolve_initial_references("PolicyManager") を使用して PolicyManager にナロウイングすることでアクセスできます。

12.2.1 IDL の定義

```
module CORBA {
    ...
    interface PolicyManager {
        PolicyList get_policy_overrides(in PolicyTypeSeq ts);
        void set_policy_overrides(in Policy[ ] policies,
            in SetOverrideType set_add)
            raises (InvalidPolicies);
    };
};
```

12.2.2 PolicyManager のメソッド

- Policy[ ] get_policy_overrides(
  int[ ])
  このメソッドは，要求した型のポリシーをすべて返します。引数に空のシーケンスを指定した場合，（つまり，長さ 0 の配列を指定した場合）該当するスコープのポリシーをすべて返します。指定したポリシーの型が対象 PolicyManager に設定されていない場合は，空のシーケンスを返します。
- void set_policy_overrides(
  Policy[ ] policies,
  SetOverrideType set_add)
  throws
  InvalidPolicy
  このメソッドは，指定したリストで現在のポリシーオーバーライドを更新します。このメソッドにポリシーとして空のシーケンスを処理モードとして SET_OVERRIDE を指定して呼び出すと，PolicyManager からオーバーライドをすべて削除します。このメソッドを使用してオーバーライドできるポリシーは，クライアント側でオペレーションの呼び出しに適用したポリシーだけです。それ以外のポリシーをオーバーライドしようとすると，CORBA.NO_PERMISSION 例外が発生します。リク

エストによって，対象 PolicyManager のオーバーライドポリシーに矛盾が生じる場合は，ポリシーの変更も追加も実行されないで，InvalidPolicies 例外が発生します。ほかの PolicyManager に設定されたポリシーとの互換性はチェックされません。

PolicyManager.set_policy_overrides( )のパラメタ policies に，複数の RebindPolicy を指定すると，リストの最後の RebindPolicy だけが PolicyManager に設定されます。

- policies
  Policy オブジェクトのリファレンスの列
- set_add
  org.omg.CORBA.SetOverrideType 型のパラメタ。このパラメタは，指定したポリシーを，オーバーライドがすでに登録されている PolicyManager に追加するか（ADD_OVERRIDE），オーバーライドが登録されていない PolicyManager に追加するか（SET_OVERRIDE）を示します。リクエストによって，指定した PolicyManager に矛盾が生じる場合は，ポリシーの変更も追加も実行されないで，InvalidPolicies 例外が発生します。

12.3 PolicyCurrent

```
public interface org.omg.CORBA.PolicyCurrent extends
    org.omg.CORBA.PolicyCurrentOperations,
    org.omg.CORBA.PolicyManager,
    org.omg.CORBA.Current,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，PolicyManager と Current から，新しいメソッドを追加しないで派生したインタフェースです。このため，PolicyManager インタフェースのオペレーションはすべて PolicyCurrent インタフェースでも使用できます。詳細については，「12.2 PolicyManager」を参照してください。

PolicyCurrent インタフェースを使用すると，スレッドレベルでオーバーライドされたポリシーにアクセスできます。スレッドの PolicyCurrent オブジェクトのリファレンスは，PolicyCurrent の識別子を指定した org.omg.CORBA.ORB.resolve_initial_references メソッドを呼び出すことで取得できます。

12.3.1 IDL の定義

```
interface PolicyCurrent :PolicyManager,Current {
};
```

12.4 Object

```
public interface org.omg.CORBA.Object
```

CORBA 2.4 仕様には限られた QoS のサポートしかないため，Borland Enterprise Server VisiBroker は org.omg.CORBA.Object を継承して OMG Messaging 仕様に従った拡張 QoS のサポートを提供しています。このため，2 種類の Object インタフェースがあることになっています。拡張機能を使用する場合は，org.omg.CORBA.Object を com.inprise.vbroker.CORBA.Object にキャストしてください。OMG Messaging 仕様に規定されている追加メソッドは現在の CORBA 2.4 ではまだ利用できないので，追加メソッドを com.inprise.vbroker.CORBA.Object に追加することによって，QoS の機能を実現します。

12.4.1 org.omg.CORBA.Object のメソッド

- public org.omg.CORBA.Policy _get_policy(
  int type)
  該当するオブジェクトリファレンスの有効ポリシーを返します。このポリシーは，リクエストが発行された場合に使用されるポリシーです。このメソッドは最初に，_get_client_policy が返す PolicyType の有効オーバーライドを取得することによって，有効ポリシーを判定します。
  次に，IOR に指定されたポリシーと有効オーバーライドを比較します。有効ポリシーは，有効オーバーライドと IOR に指定されたポリシーが許容する値の共通部分です。共通部分が空の場合は，INV_POLICY システム例外が発生します。空でない場合は，値が共通部分のポリシーを有効ポリシーとして返します。IOR にポリシー値の指定がない場合は，適当な値を使用してもよいことを意味します。オブジェクトリファレンスに対して_get_policy メソッドを呼び出す前に，_non_existent メソッドまたは_validate_connection メソッドを呼び出すと，返される有効ポリシーの正確さが保証されます。
  _get_policy を，オブジェクトリファレンスをバインドする前に呼び出すと，インプリメンテーション依存の有効ポリシーが返されます。この状況では，仕様に従って生成したインプリメンテーションは次のどれかの動作を実行する可能性があります。
  - CORBA.BAD_INV_ORDER 例外が発生します。
  - バインドごとに変更される可能性のある PolicyType の値を返します。
  - バインドを試みて有効ポリシーを返します。

  RebindPolicy に TRANSPARENT，VB_TRANSPARENT，または VB_NOTIFY_REBIND が設定されている場合は，透過的なリバインドによって，有効ポリシーは呼び出しごとに変わる可能性があります。
- org.omg.CORBA.Object _set_policy_override(
  const Policy[ ] _policies,
  SetOverrideType _set_add)
  throws
  org.omg.CORBA.InvalidPolicy

このメソッドは，指定したオブジェクトレベルでのポリシーオーバーライドのリストで新規オブジェクトリファレンスを返します。さらに，オブジェクト，スレッド，または VisiBroker ORB のカレントのポリシーを，指定したポリシーオーバーライドのリストで更新します。

12.4.2 com.inprise.vbroker.CORBA.Object のメソッド

- public org.omg.CORBA.Policy **_get_client_policy**(
  int **type**)
  _get_client_policy メソッドは，サーバ側ポリシーとの共通部分を考慮しないで，オブジェクトリファレンスの有効オーバーライドポリシーを返します。有効オーバーライドは，まず，指定した PolicyType のオーバーライドをオブジェクトのスコープでチェックし，次にカレントスコープでチェックし，最後に VisiBroker ORB のスコープでチェックして取得します。指定した PolicyType にオーバーライドが登録されていなければ，その PolicyType 固有のシステム依存デフォルト値を使用します。デフォルトのポリシー値は指定されないので，ポータブルなアプリケーションは，必要な「デフォルト」を VisiBroker ORB のスコープで設定することを求められています。
- org.omg.CORBA.Policy[ ] **_get_policy_overrides**(
  int[ ] **types**)
  _get_policy_overrides メソッドは，オブジェクトレベルで指定したポリシー型のポリシーオーバーライドのリストを返します。引数に空のシーケンスを指定した場合，（つまり，長さ 0 の配列を指定した場合）オブジェクトレベルのオーバーライドをすべて返します。オブジェクトレベルで PolicyTypes が一つもオーバーライドされていない場合は，空のシーケンスを返します。
- boolean **_validate_connection**(
  org.omg.CORBA.PolicyListHolder **inconsistent_policies**)
  _validate_connection メソッドは，該当するオブジェクトのカレントの有効ポリシーで呼び出しができるかどうかを Boolean 値で返します。呼び出しができる場合は true を返します。オブジェクトリファレンスがバインド済みでない場合，このメソッドはオペレーションの一環としてバインディングを実行します。バインド済みの場合でも，カレントのポリシーオーバーライドが変更されていたり，何らかの理由でバインディングが無効になっている場合は，RebindPolicy オーバーライドの設定に関係なくリバインドされます。

  カレントの有効 RebindPolicy が暗黙的なリバインドを許可していない場合，そのようなリバインドを強制的にできるのは_validate_connection だけです。バインドやリバインドの実行を試みると，VisiBroker ORB によって GIOP の LocateRequests 処理が実行されます。

  カレントの有効ポリシーで呼び出すと INV_POLICY システム例外が発生する場合，このメソッドは false を返します。カレントの有効ポリシーに互換性がない場合，out パラメタ「inconsistent_policies」に非互換性を生じさせたポリシーを格納します。このパラメタに返されたポリシーのリストはすべての原因を網羅しているわけではありません。ポリシーのオーバーライドに関係しない原因によってバインディングが失敗した場合は，その原因に応じたシステム例外が発生します。

12.5 RebindPolicy

```
public interface org.omg.Messaging.RebindPolicy extends
    org.omg.Messaging.RebindPolicyOperations,
    org.omg.CORBA.Policy,
    org.omg.CORBA.portable.IDLEntity
```

VisiBroker での RebindPolicy は，CORBA 2.5 の仕様に従って実装されています。さらに，VisiBroker ではオブジェクト障害後 osagent を使用したほかのオブジェクトの呼び出しをサポートするための拡張をしています。

RebindPolicy は，クローズしたコネクション，GIOP のロケーションフォワードメッセージ，およびオブジェクト障害をクライアント側の VisiBroker ORB がどのように処理するかを決定します。VisiBroker ORB は org.omg.CORBA.Object インスタンスにある有効ポリシーを参照して，オブジェクト障害後 osagent を使用したほかのオブジェクトの呼び出し，リバインド，および再接続を処理します。

VisiBroker ORB が対象サーバに正常にバインドされたあとで，VisiBroker ORB が透過的にリバインドするかどうかが，OMG が定義したポリシーの値によって決定されます。また，VisiBroker ORB がターゲットのオブジェクトにバインドされたあとで，VisiBroker ORB が透過的にオブジェクト障害後 osagent を使用したほかのオブジェクトの呼び出しをするかどうかが，継承されたポリシーの値によって決定されます。

注

RebindPolicy が実行されるのは，オブジェクトに正常にバインドされたあとだけです。GIOP に従ったプロトコルの場合，オブジェクトリファレンスは LocateReply メッセージが返されたあとでバインドされるとみなされます。

RebindPolicy の値によって切断，オブジェクトフォワードリクエスト，またはオブジェクト障害の対処方法が決定されます。

RebindPolicy の値には，仕様にない値を指定しないでください。仕様にない値を引数にして，ORB.create_policy( )を呼ぶと，INV_POLICY 例外が発生します。

12.5.1 IDL の定義

```
#pragma prefix ”omg.org”

module Messaging {
    typedef short RebindMode;
    const CORBA::PolicyType REBIND_POLICY_TYPE =23;
    interface RebindPolicy::CORBA::Policy {
        readonly attribute RebindMode rebind_mode;
    };
};
```

12.5.2 ポリシーの値

RebindPolicy として設定できる OMG のポリシー値について，次の表に示します。

表 12–1 OMG のポリシー値（Java）

ポリシー値	説明
TRANSPARENT	このポリシーは，リモートリクエスト実行時に，オブジェクト障害後 osagent を使用してほかのオブジェクトを呼び出すことと必要な再接続処理を VisiBroker ORB が暗黙的に実行することを許可します。これは，最も制限を加えない値です。
NO_REBIND	このポリシーは，リモートリクエスト実行時に，クローズしたコネクションの再開を VisiBroker ORB が暗黙的に実行することを許可します。ただし，クライアント側で有効になっている QoS ポリシーに変更を加えるような，透過的な呼び出し（オブジェクト障害後 osagent を使用してほかのオブジェクト呼び出すこと）は許可しません。
NO_RECONNECT	このポリシーは，オブジェクト障害後 osagent を使用してほかのオブジェクトを呼び出すことと，クローズしたコネクションの再開を VisiBroker ORB が暗黙的に実行することを禁止します。これは，最も制限を加える値です。

注

次の条件がすべて重なる場合，サーバへの呼び出しをすると MARSHAL 例外または DATA_CONVERSION 例外が発生します。

- 複数のスレッドで同じオブジェクトリファレンスを使用する。
- QoS の RebindPolicy に NO_RECONNECT 以外を指定する。
- サーバが再起動，またはサーバへの呼び出しでタイムアウトが発生することによってコネクションが切断され，透過的な再接続処理が行われる。
- 次のデータ型を使用する。

  char

  java.lang.String

  java.lang.Object

RebindPolicy に設定できる VisiBroker 固有のポリシー値について，次の表に示します。

表 12–2 VisiBroker 固有のポリシー値（Java）

ポリシー値	説明
com.inprise.vbroker.QoSExt.VB_TRANSPARENT	これは，TRANSPARENT のオブジェクト障害後 osagent を使用したほかのオブジェクトの呼び出しをする動作を継承したポリシーです。これはデフォルトポリシーです。 サーバダウンが原因でリモート呼び出しが失敗するときにこのポリシーを設定すると，VisiBroker ORB は osagent を使用してほかのサーバへの再接続を試みます。リバインドに成功すると，クライアントの VisiBroker ORB は通信障害をマスクし，呼び出し元のスレッドに例外を返しません。

ポリシー値	説明
com.inprise.vbroker.QoSExt.VB_NOTIFY_REBIND	VB_NOTIFY_REBIND は VB_TRANSPARENT と同様の動作をしますが，通信障害を検知した場合に例外を発生します。もう一度呼び出しをすると，このポリシーは，ほかのオブジェクトへの透過的な再接続を試みます。
com.inprise.vbroker.QoSExt.VB_NO_REBIND	VB_NO_REBIND は，オブジェクト障害後，osagent を使用してほかのオブジェクトを呼び出すことを有効にしません。また，オブジェクトの転送も許可しません。クライアント VisiBroker ORB が，同じサーバに対してクローズしたコネクションを再開することだけを許可します。

注

ユーザのクライアントの有効ポリシーが VB_TRANSPARENT であり，かつ状態データを保存しているサーバと連携している場合，VB_TRANSPARENT はそのクライアントを新規サーバに接続することがあります。この場合，クライアントはサーバが変更されたことを認識しないで，元のサーバに保存されていた状態データは失われます。

クライアントで RebindPolicy に NO_RECONNECT または VB_NO_REBIND を設定した場合，かつ，初回リクエストでオブジェクト転送が発生した場合，オブジェクト転送を許可します。

12.6 RebindForwardPolicy

```
public interface com.inprise.vbroker.QoSExt.RebindForwardPolicy
    extends com.inprise.vbroker.QoSExt.RebindForwardPolicyOperations,
    org.omg.CORBA.Policy,
    org.omg.CORBA.portable.IDLEntity
```

RebindForwardPolicy は，LOCATION_FORWARD 時に接続が失敗した場合に，クライアントの VisiBroker ORB がリバインドを実行するかどうかを決定します。また，クライアントが新規オブジェクトにフォワードされたとき，新規フォワード先オブジェクトへの接続を試みます。接続に失敗した場合，VisiBroker ORB は透過的に元のオブジェクト（フォワード元）に接続し直します。この再接続は，次に示す場合に実行されます。

- この時点でのフォワード数の合計が，当該ポリシーに指定した forward_count の値を超えていない場合
- 同じフォワード先への接続を試みた直後ではない場合

vbroker.orb.rebindForward プロパティは，VisiBroker ORB レベルで forward_count の値を設定します。forward_count の値は，QoS ポリシーと同様，VisiBroker ORB レベル，スレッドレベル，またはオブジェクトレベルでオーバーライドできます。

デフォルト値の 0 は，制限を設定しないことを指定します。

12.6.1 IDL の定義

```
#pragma prefix ”inprise.org”

module QoSExt{
    typedef short ForwardCount;
    const CORBA::PolicyType REBIND_FORWARD_POLICY_TYPE
                                                = 0x56495314
    interface RebindForwardPolicy::CORBA::Policy {
        readonly attribute ForwardCount forward_count;
    };
};
```

12.7 RelativeConnectionTimeoutPolicy

```
public interface com.inprise.vbroker.QoSExt.
    RelativeConnectionTimeoutPolicy extends
    com.inprise.vbroker.QoSExt.RelativeConnectionTimeoutPolicyOperations,
    org.omg.CORBA.Policy,
    org.omg.CORBA.portable.IDLEntity
```

RelativeConnectionTimeoutPolicy は，有効なエンドポイントを使用してオブジェクトと接続しようとしたときに，どれだけの時間，接続に成功しなければ処理をタイムアウトするかを表します。ファイアウォールに保護されたオブジェクトで，オブジェクトへの接続手段が HTTP トンネリング以外にない場合，タイムアウトしやすくなります。

注

このポリシーはプロセス内通信には適用されません。

insert_ulonglong 型のポリシーは，100 ナノ秒単位でタイムアウトを指定します。タイムアウトは，VisiBroker ORB が接続を試みるエンドポイントのそれぞれに適用されます。したがって，複数の接続リクエストが発行された場合，経過時間は，設定したタイムアウトの倍数となります。その精度も Java VM のインプリメンテーションによって制限されます。

12.7.1 IDL の定義

```
#pragma prefix ”inprise.com”

module QoSExt{
    const CORBA::PolicyType RELATIVE_CONN_TIMEOUT_POLICY_TYPE
                                                    = 0x56495304
    interface RelativeConnectionTimeoutPolicy : CORBA::Policy {
        readonly attribute TimeBase::TimeT relative_expiry:
    };
};
```

12.8 Messaging.RelativeRequestTimeoutPolicy

```
public interface org.omg.Messaging.RelativeRequestTimeoutPolicy
    extends org.omg.Messaging.RelativeRequestTimeoutPolicyOperations,
    org.omg.CORBA.Policy,
    org.omg.CORBA.portable.IDLEntity
```

RelativeRequestTimeoutPolicy は，リクエストが渡されるときのタイムアウトを指定します。指定した時間が経過すると，リクエストは取り消されます。このポリシーは，同期呼び出しと非同期呼び出しの両方に適用されます。指定したタイムアウト以内にリクエストが完了するとみなされるため，タイムアウトによって応答が破棄されることはありません。タイムアウトの値は 100 ナノ秒単位で指定します。

12.8.1 IDL の定義

```
#pragma prefix ”omg.org”

module Messaging {
    const CORBA::PolicyType RELATIVE_REQ_TIMEOUT_POLICY_TYPE = 31;
    interface RelativeRequestTimeoutPolicy : CORBA::Policy {
        readonly attribute TimeBase::TimeT relative_expiry;
    };
};
```

12.9 Messaging.RelativeRoundtripTimeoutPolicy

```
public interface org.omg.Messaging.RelativeRoundtripTimeoutPolicy
    extends org.omg.Messaging.RelativeRoundtripTimeoutPolicyOperations,
    org.omg.CORBA.portable.IDLEntity
```

RelativeRoundtripTimeoutPolicy は，リクエストまたはその応答が渡されるときのタイムアウトを指定します。指定した時間が経過しても応答が渡されない場合，リクエストは取り消されます。また，リクエストがすでに渡されて応答が返ってきた場合，応答はタイムアウトに指定した時間が経過したときに破棄されます。このポリシーは，同期呼び出しと非同期呼び出しの両方に適用されます。指定したタイムアウト以内にリクエストが完了するとみなされるため，タイムアウトによって応答が破棄されることはありません。タイムアウトの値は 100 ナノ秒単位で指定します。

12.9.1 IDL の定義

```
#pragma prefix ”omg.org”

module Messaging {
    const CORBA::PolicyType RELATIVE_RT_TIMEOUT_POLICY_TYPE = 32;
    interface RelativeRoundtripTimeoutPolicy : CORBA::Policy {
        readonly attribute TimeBase::TimeT relative_expiry;
    }
};
```

12.10 DeferBindPolicy

```
public interface com.inprise.vbroker.QoSExt.DeferBindPolicy extends
    com.inprise.vbroker.QoSExt.DeferBindPolicyOperations,
    org.omg.CORBA.Policy,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，リモートオブジェクトが最初に生成されたときに VisiBroker ORB がすぐにコンタクトをするのか，それとも呼び出しがあるまでコンタクトを延期するのかを決定します。デフォルトでは，VisiBroker ORB は bind()を発行したときに（リモート）オブジェクトに接続します。DeferBindPolicy に TRUE が設定されている場合は，VisiBroker ORB は呼び出しがあるまでそのオブジェクトにコンタクトしません。

クライアントオブジェクトを生成し，DeferBindPolicy を TRUE に設定すると，サーバの起動を呼び出しがあるまで延期できます。このオプションは，生成された Helper クラスの Bind メソッドのオプションとして以前からあったものです。

12.10.1 IDL の定義

```
#pragma prefix ”inprise.com”

module QoSExt{
    const CORBA::PolicyType DEFER_BIND_POLICY_TYPE =
                                            0x56495305
    interface DeferBindPolicy: CORBA::Policy {
        readonly attribute boolean value:
    }
};
```

12.11 ExclusiveConnectionPolicy

```
public interface com.inprise.vbroker.QoSExt.
    ExclusiveConnectionPolicy extends
    com.inprise.vbroker.QoSExt.ExclusiveConnectionPolicyOperations,
    org.omg.CORBA.Policy, org.omg.CORBA.portable.IDLEntity
```

ExclusiveConnectionPolicy は，指定したサーバオブジェクトとの排他接続（共有でない接続）を確立するための，VisiBroker 固有のポリシーです。このポリシーは，true または false のブール値を持ちます。true が設定された場合は，サーバオブジェクトへの排他接続をオープンします。false が設定された場合で既存のコネクションを再使用できるときは，既存のコネクションを再使用します。既存のコネクションを再使用できないときは，新しいコネクションをオープンします。デフォルトは false です。

このポリシーに true を設定した場合は，「vbroker.ce.iiop.ccm.connectionMax」が有効にならないため，このプロパティに指定した値以上のコネクションが使用される可能性があります。

このポリシーは，VisiBroker 3.x の Object._clone()と同じ動作をします。

このポリシーが有効になるのは次のような場合です。

- _bind()前に ORB または PolicyCurrent に対して Policy を設定している場合
- string_to_object で作成した Object を呼び出す前に，ORB または PolicyCurrent に対して Policy を設定している場合
- string_to_object や_bind で作成した Object に対して Policy を設定し，Policy を設定された Object に対してリクエストを行う場合

12.11.1 IDL の定義

```
module QoSExt {
  const CORBA::PolicyType EXCLUSIVE_CONNECTION_POLICY_TYPE = 0x56495320;
  interface ExclusiveConnectionPolicy :CORBA::Policy {
    /** Returns the current setting of */
    /** the ExclusiveConnectionPolicy  */
    readonly attribute boolean value;
  };
};
```

12.12 SyncScopePolicy

```
public interface org.omg.Messaging.SyncScopePolicy extends
    org.omg.Messaging.SyncScopePolicyOperations,
    org.omg.CORBA.Policy,
    org.omg.CORBA.portable.IDLEntity
```

このインタフェースは，CORBA::Policy から派生したローカルオブジェクトです。このインタフェースは，一方向オペレーションに適用され，オペレーション要求の対象への同期のスコープを示します。非一方向オペレーションが呼び出された場合は無視されます。このポリシーは，DII が INV_NO_RESPONSE のフラグで使用されているときも適用されます。それは，インタフェース定義を照会してオペレーションが一方向かどうかを調べるための DII のインプリメンテーションが不要なためです。このポリシーのデフォルトは SYNC_WITH_TRANSPORT です。各アプリケーションは，SyncScopePolicy を明示的に設定して，VisiBroker ORB インプリメンテーション間のポータビリティを保障する必要があります。SyncScopePolicy のインスタンスが作成されると，Messaging::SyncScope 型の値が CORBA::ORB::create_policy に渡されます。このポリシーは，クライアント側のオーバーライドとしてだけ適用できます。

12.12.1 IDL の定義

```
module Messaging {
    interface SyncScopePolicy :CORBA::Policy {
        readonly attribute SyncScope synchronization;
    };
};
```

12.12.2 SyncScope ポリシーの値

SyncScope ポリシーの値と動作を次の表に示します。

表 12-3 SyncScope のポリシー値（Java）

ポリシー値	説明
SYNC_NONE	VisiBroker ORB は，リクエストメッセージをトランスポートプロトコルに渡す前に（例えば，メソッド呼び出しから），クライアントに制御を返します。クライアントは，動作を抑止しないことが保障されます。サーバから応答はないため，このレベルの同期ではロケーションフォワードは実行されません。 IDL に oneway 宣言したオペレーションを，QoS の SyncScope ポリシーに SYNC_NONE を設定して実行した場合，send_oneway( )を呼び出した延長で送信するリクエスト電文が TCP の送信バッファに書き込めずプロセス内で滞留すると，オペレーションがブロッキングされます。回避策として，QoS の RelativeRequestTimeout ポリシーに有限値を設定してください。送信タイムアウト時間経過後に，oneway リクエストにて TIMEOUT を検知し終了させることができます。

ポリシー値	説明
SYNC_WITH_TRANSPORT	VisiBroker ORB は，リクエストメッセージをトランスポートプロトコルに渡したあとに，クライアントに制御を返します。サーバから応答はないため，このレベルの同期ではロケーションフォワードは実行されません。
SYNC_WITH_SERVER	サーバ側 VisiBroker ORB は，対象インプリメンテーションを呼び出す前に応答を送信します。NO_EXCEPTION の応答が送信された場合，必要なロケーションフォワードがすでに発生しています。この応答受信時に，クライアント側 VisiBroker ORB は，制御をクライアントアプリケーションに返します。クライアントは，ロケーションフォワードがすべて完了するまで動作を抑止します。POA を使用しているサーバの場合，応答は，ServantManager が呼び出されてから対象サーバントにリクエストが渡されるまでの間に送信されます。
SYNC_WITH_TARGET	CORBA 2.2 の，同期型非一方向オペレーションと同等です。サーバ側 VisiBroker ORB は，オペレーション呼び出し先でオペレーションが完了したあとに，応答メッセージの送信だけを実行します。LOCATION_FORWARD の応答は，オペレーション呼び出し前に送信済みです。SYSTEM_EXCEPTION の応答は，例外の構文に従った任意の時間に送信できる状態となります。一方向と定義されている場合でも，実際は，オペレーションは同期オペレーションと同様に動作します。この形式の同期では，対象オペレーションがリクエストを解釈し，そのリクエストに従って実行したことをクライアントが認識していることが保障されます。CORBA 2.2 と同様，この最高レベルの同期では，OTS だけを使用できます。これより低いレベルの同期で呼び出されたオペレーションでは，対象はクライアントのカレントトランザクションに入れません。

12.13 QoS 例外

QoS で発生する例外について，次の表に示します。

表 12–4 QoS で発生する例外（Java）

例外	説明
org.omg.CORBA.INV_POLICY	ポリシーオーバーライド間に互換性がない場合に発生します。
org.omg.CORBA.REBIND	RebindPolicy に NO_REBIND，NO_RECONNECT，または VB_NOTIFY_REBIND を設定し，バインドされたオブジェクトリファレンスの呼び出しがオブジェクトフォワードメッセージまたはロケーションフォワードメッセージとなった場合に発生します。
org.omg.CORBA.PolicyError	指定したポリシーがサポートされていない場合に発生します。

13 IOP および IIOP のインタフェースとクラス（Java）

この章では，General Inter-ORB Protocol の主なインタフェース，および CORBA 仕様で定義されたそのほかの構造体の，Borland Enterprise Server VisiBroker でのインプリメンテーションについて，Java 言語でのインタフェースを説明します。

13.1 IIOP.ProfileBody

```
public final class com.inprise.vbroker.IIOP.ProfileBody extends
    java.lang.Object implements org.omg.CORBA.portable.IDLEntity
```

このクラスは，オブジェクトがサポートするプロトコルについての情報を含んでいます。

このクラスには，Helper クラスと Holder クラスもあります。これらのクラスとそのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

13.1.1 IDL の定義

```
struct ProfileBody {
    ::GIOP::Version iiop_version;
    string host;
    unsigned short port;
    ::CORBA::OctetSequence object_key;
    sequence<::IOP::TaggedComponent>components;
};
```

13.1.2 IIOP.ProfileBody の変数

- public com.inprise.vbroker.GIOP.Version iiop_version
  IIOP がサポートするバージョンを表します。
- public java.lang.String host
  オブジェクトがインプリメントされるホスト名を表します。
- public short port
  オブジェクトへのコネクションを確立するために使用するポート番号を表します。
- public byte[ ] object_key
  オブジェクトリファレンスを一意に識別するために使用し，オブジェクトをインプリメントするサーバントを見つけるための情報を格納します。
  オブジェクトキーはベンダ固有の形式で格納され，IOR が生成されたときに生成されます。
- public com.inprise.vbroker.IOP.TaggedComponent[ ] components
  0 個以上の TaggedComponent オブジェクトのシーケンスを表します。このプロファイルが記述するオブジェクトの呼び出しに使用できる追加情報を格納するために使用します。

13.1.3 IIOP.ProfileBody のコンストラクタ

- public **ProfileBody**( )
  空の ProfileBody を生成します。
- public **ProfileBody**(
  com.inprise.vbroker.GIOP.Version **iiop_version**,
  java.lang.String **host**,short **port**,byte[ ] **object_key**,
  com.inprise.vbroker.IOP.TaggedComponent[ ] **components**)
  指定した IIOP バージョン，ホスト名，ポート番号，オブジェクトキー，およびコンポーネントで初期化した ProfileBody を生成します。
  - iiop_version
    サポートする IIOP のバージョン
  - host
    オブジェクトがインプリメントされるホスト名
  - port
    オブジェクトのコネクション確立時に使用するポート番号
  - object_key
    オブジェクトのキー
  - components
    0 個以上の TaggedComponent オブジェクトのシーケンス。このプロファイルが記述するオブジェクトの呼び出しに使用できる追加情報を格納するために使用します。

13.2 IOP.IORValue

```
public abstract class com.inprise.vbroker.IOP.IORValue extends
    java.lang.Object implements org.omg.CORBA.portable.StreamableValue
```

このクラスは，インターオペラブルオブジェクトリファレンスを表し，オブジェクトリファレンスについての重要な情報を提供するために使います。クライアントアプリケーションは ORB::object_to_string メソッドを呼び出して IOR を生成できます。ORB::object_to_string メソッドについては，「4.5 ORB」を参照してください。

このクラスには，Helper クラスと Holder クラスもあります。これらのクラスとそのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

13.2.1 IDL の定義

```
valuetype IORValue {
    public string type_id;
    public ProfileValueSeq profiles;
    IOR toIOR( );
    IORValue copy( );
    boolean matchesTemplate (in IORValue iorv);
};
```

13.2.2 IOP.IORValue の変数

- public java.lang.String type_id
  この IOR によって表されるオブジェクトリファレンスの型を記述します。
- public com.inprise.vbroker.IOP.ProfileValue[ ] profiles
  一つ以上の TaggedProfile オブジェクトのシーケンスを表します。そのオブジェクトには，サポートしているプロトコルについての情報が含まれています。

13.2.3 IOP.IORValue のメソッド

- public com.inprise.vbroker.IOP.IOR toIOR( )
  IOR に変換します。
- public com.inprise.vbroker.IOP.IORValue copy( )
  IORValue の複製を作成します。
- public boolean matchesTemplate(
  IORValue iorv)

IORValue がテンプレート IORValue と一致しているか調べます。

13.3 IOP.ServiceContext

```
public final class com.inprise.vbroker.IOP.ServiceContext extends
    java.lang.Object implements org.omg.CORBA.portable.IDLEntity
```

このクラスは，リクエストまたは応答を渡すときに付加するサーバ固有のコンテキスト情報を表します。

このクラスには，Helper クラスと Holder クラスもあります。これらのクラスとそのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

13.3.1 IDL の定義

```
struct ServiceContext {
    ::IOP::ServiceID context_id;
    ::CORBA::OctetSequence context_data;
};
```

13.3.2 IOP.ServiceContext の変数

- public int **Context_id**
  特定のサービスとデータ形式を識別します。
- public byte[ ] **context_data**
  context_id で識別されるサービスに対応するコンテキストデータを表します。このコンテキストデータはサービス固有の形式でエンコードされ，オクテットのシーケンスとしてカプセル化されます。

13.3.3 IOP.ServiceContext のコンストラクタ

- public **ServiceContext**( )
  空の ServiceContext を生成します。
- public **ServiceContext**(
  int **context_id**, byte[ ] **context_data**)
  指定した識別子とデータで初期化した ServiceContext を生成します。
  - context_id
    特定のサービスとデータ形式を識別します。
  - context_data
    オクテットのシーケンスとしてカプセル化したサービス固有のデータを示します。

13.4 IOP.TaggedProfile

```
public final class com.inprise.vbroker.IOP.TaggedProfile extends
    java.lang.Object implements org.omg.CORBA.portable.IDLEntity
```

このクラスは，サポートするプロトコルと，そのプロトコルがオブジェクトを識別するために必要とするカプセル化された基本情報を表します。

このクラスには，Helper クラスと Holder クラスもあります。これらのクラスとそのメソッドの詳細については，「3. 生成されるインタフェースとクラス（Java）」を参照してください。

13.4.1 IDL の定義

```
struct TaggedProfile {
    ::IOP::ProfileId tag;
    sequence <octet> profile_data;
};
```

13.4.2 IOP.TaggedProfile の変数

- public int tag
  プロファイルデータの内容を識別する，次の値のどれかを指定します。
  - TAG_INTERNET_IOP
    標準 IIOP プロトコルであることを示します。
  - TAG_MULTIPLE_COMPONENTS
    プロファイルデータが，プロトコルで利用できる ORB サービスの一覧を含んでいることを示します。
  - TAG_VB_LOCATOR
    IOR が，osagent が実 IOR を受信するまで使用される interim の擬似オブジェクトであることを示します。
  - TAG_VSGN_LIOP
    プロトコルが，ローカル IPC 機能をカバーする IOP であることを示します。
- public byte[ ] profile_data
  オブジェクトを識別するために必要なプロトコル情報をすべてカプセル化します。

13.4.3 IOP.TaggedProfile のコンストラクタ

- public TaggedProfile( )

空の TaggedProfile を生成します。

- public **TaggedProfile**(
  int **tag**, byte[ ] **profile_data**)
  指定したタグとデータで初期化した TaggedProfile を生成します。
  - tag
    プロファイルデータの内容を識別する次の値のどれかを指定します。
    TAG_INTERNET_IOP
    TAG_MULTIPLE_COMPONENTS
    TAG_VB_LOCATOR
    TAG_VSGN_LIOP
  - profile_data
    IOR に対するオペレーションを呼び出すために必要なプロトコル情報

14 RMI インタフェースとクラス（Java）

この章では，Java 言語で RMI-IIOP をサポートするために使用するインタフェースとクラスについて説明します。Borland Enterprise Server VisiBroker は現在，VisiBroker 4.x の RMI-IIOP のサーバ側でのプログラミングモデルをサポートしていません。サーバ側の各種 API をここで説明してはいますが，それらを使用すると OBJECT_NOT_EXIST 例外が発生する可能性があります。

14.1 PortableRemoteObject

```
public abstract class javax.rmi.PortableRemoteObject
    extends java.lang.Object { }
```

このクラスから継承してすべてのサーバインプリメンテーションを生成します。RMI-IIOP のサーバインプリメンテーションは javax.rmi.PortableRemoteObject から継承するか，RMI-IIOP リモートインタフェースをインプリメントしてから，exportObject メソッドを使用してサーバインプリメンテーション自体を一つサーバオブジェクトとして登録できます。クライアントは narrow メソッドを使用して，一般的なリモートインタフェースを特定のリモートインタフェースにナロウイングします。

14.1.1 PortableRemoteObject のコンストラクタ

- public static void **exportObject**(
  Remote **obj**)
  サーバオブジェクトをリモート呼び出し受信ができる状態にします。このコンストラクタが呼び出す PortableRemoteObject のサブクラスは，このメソッドを呼び出す必要はありません。
  - obj
    エクスポートするサーバオブジェクト

14.1.2 PortableRemoteObject のメソッド

- protected **PortableRemoteObject**( )
  このメソッドは，exportObject( )を呼び出すことによってオブジェクトを初期化します。
- public static Remote **toStub**(
  Remote **obj**)
  このメソッドは，指定したサーバオブジェクトのスタブを返します。サーバは，リモート通信が受信できる状態でなければなりません。このためには，PortableRemoteObject.connect(Remote，Remote) メソッドが必要な場合があります（オブジェクトがリモートメソッド呼び出しの引数として渡されていない場合)。指定したサーバオブジェクトにスタブが見つからない場合は，
  java.rmi.NoSuchObjectException 例外が発生します。
  - obj
    スタブを調べたいサーバオブジェクト。PortableRemoteObject のサブクラスであるか，PortableRemoteObject.exportObject( )を呼び出していなければなりません。
- public static void **unexportObject**(
  Remote **obj**)

このメソッドは，ランタイムからサーバオブジェクトの登録を解除します。登録を解除したサーバオブジェクトは，ガーベッジコレクションの対象になります。リモートオブジェクトが現在エクスポートされていない場合，java.rmi.NoSuchObjectException 例外が発生します。

- obj
  エクスポートするオブジェクト

- public static java.lang.Object **narrow**(
  java.lang.Object **narrowFrom**, java.lang.Class **narrowTo**)
  このメソッドは，該当する RMI-IIOP オブジェクトを，narrowTo クラスのリモートインタフェースのスタブにナロウイングします。narrowFrom を narrowTo にキャストできない場合は ClassCastException 例外が発生します。
  - narrowFrom
    型にキャストするオブジェクト
  - narrowTo
    キャストするオブジェクトの型
- public static void **connect**(
  Remote **unconnected**, Remote **connected**)
  このメソッドは，リモートオブジェクトを，リモート通信が実行できる状態にします。通常，オブジェクトをリモートメソッド呼び出しの引数として送受信する場合に暗黙的にこの状態になりますが，状況によっては，明示的にこれを実行した方がよい場合があります。connected に指定したオブジェクトが接続されていなかったり，unconnected に指定したオブジェクトがすでに接続されていたりした場合，java.rmi.RemoteException 例外が発生します。
  - unconnected
    接続するオブジェクト
  - connected
    接続済みのオブジェクト

15 URL ネーミングインタフェースとクラス（Java）

この章では，Borland Enterprise Server VisiBroker の URL ネーミングサービスで使用する Resolver インタフェースとクラスについて説明します。

以前のバージョンでは URL ネーミングサービスは，Web ネーミングサービスと呼ばれていました。

15.1 Resolver

```
public interface Resolver extends Object
```

URL ネーミングサービスを使用する場合，Resolver は ORB の resolve_initial_references で呼び出されます。Resolver 使用の詳細については，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「URL ネーミングの使用」の記述を参照してください。

```
interface Resolver {

    // Read Operations
    Object locate(in string url_s)
        raises (InvalidURL, CommFailure, ReqFailure);

    // Write Operations
    void force_register_url(in string url_s, in Object obj)
        raises (InvalidURL, CommFailure, ReqFailure);

    void register_url(in string url_s, in Object obj)
        raises (InvalidURL, CommFailure, ReqFailure,
                AlreadyExists);
};
```

15.1.1 Resolver のメソッド

- Object locate(

  String url_s)

  Resolver に接続する必要がある場合，クライアントアプリケーションは bind メソッド呼び出し時にパラメタに URL を指定します。このとき，locate( )メソッドが，bind( )メソッドから透過的に呼び出されます。

  URL が無効の場合は，InvalidURL 例外が発生します。

  - url_s

    URL の文字列

- void force_register_url(

  String url_s, Object obj)

  このメソッドは，サーバのオブジェクトの IOR を URL に対応づけて，そのオブジェクトを登録します。

  force_register_url メソッドを使用して URL とオブジェクトの IOR を対応づけようとした場合，URL がすでにそのオブジェクトに対応づけられているとき，新しい URL 対応が古い対応と入れ替わります。

  - url_s

    URL の文字列

  - obj

URL に対応づけるオブジェクト

- void **register_url**(
  String **url_s**, Object **obj**)
  このメソッドは，サーバのオブジェクトの IOR を URL に対応づけて，そのオブジェクトを登録します。
  register_url メソッドを使用して URL とオブジェクトの IOR を対応づけようとした場合に，URL がすでにオブジェクトに対応づけられているとき，AlreadyExists 例外が発生します。
  - url_s
    URL の文字列
  - obj
    URL に対応づけるオブジェクト

16 ロケーションサービスインタフェースとクラス（Java）

この章では，スマートエージェントが管理するネットワーク上にあるオブジェクトインスタンスの検索に使用できる，Java 言語のロケーションサービスのエージェントと TriggerHandler インタフェースについて説明します。ロケーションサービスの詳細については，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「ロケーションサービスの使用」の記述を参照してください。

16.1 Agent

```
public interface Agent extends
    com.inprise.vbroker.CORBA.Object
```

ロケーションサービスエージェントはスマートエージェントが管理するネットワーク上のオブジェクトインスタンスを検索できるインタフェースを提供します。Agent インタフェースのメソッドは，二つのグループに分けられます。一つは，インスタンスの情報をスマートエージェントに問い合わせます。もう一つは，トリガーの登録，および登録解除をします。

問い合わせは，インタフェースのリポジトリ ID，またはインタフェースのリポジトリ ID と，インスタンス名の組み合わせに基づいてできます。問い合わせ結果は，オブジェクトリファレンス，またはそれよりも詳細なインスタンス記述を返せます。トリガーとは，ロケーションサービスを利用しているクライアントにインスタンスの可用性の変化を通知する通知機能です。

16.1.1 IDL の定義

```
interface Agent {
    HostnameSeq all_agent_locations( )
        raises(Fail);
    RepositoryIdSeq all_repository_ids( )
        raises(Fail);
    ObjSeq all_instances(in string repository_id)
        raises(Fail);
    ObjSeq all_replica(in string repository_id,
                       in string instance_name)
        raises(Fail);
    DescSeq all_instances_descs(in string repository_id)
        raises(Fail);
    DescSeq all_replica_descs(in string repository_id,
                              in string instance_name)
        raises(Fail);
    void reg_trigger(in TriggerDesc desc,
                     in TriggerHandler handler)
        raises(Fail);
    void unreg_trigger(in TriggerDesc desc,
                       in TriggerHandler handler)
        raises(Fail);
};
```

16.1.2 Agent のメソッド

- public java.lang.String[ ] all_agent_locations( )
  throws
  com.inprise.vbroker.ObjLocation.Fail

このメソッドは，osagent が常駐しているホスト名を取得します。

このメソッドでは，次の例外が発生します。

Fail

使用できるエージェントがありません。または osagent との通信に失敗しました。

- public org.omg.CORBA.Object[ ] all_instances(
  java.lang.String repository_id)
  throws
  com.inprise.inprise.vbroker.ObjLocation.Fail
  このメソッドは，指定されたリポジトリ ID を満足させる，インタフェースのインスタンスのオブジェクトリファレンスを取得します。
  - repository_id
    リポジトリ ID を含む文字列

  このメソッドでは，次の例外が発生します。

  Fail

  リポジトリ ID が無効です。
- public com.inprise.vbroker.ObjLocation.Desc[ ] all_instances_descs(
  java.lang.String repository_id)
  throws
  com.inprise.inprise.vbroker.ObjLocation.Fail
  このメソッドは，指定されたリポジトリ ID をインプリメントするインタフェースのインスタンスに対する完全な記述情報を取得します。
  - repository_id
    リポジトリ ID を含む文字列

  このメソッドでは，次の例外が発生します。

  Fail

  リポジトリ ID が無効です。
- public org.omg.CORBA.Object[ ] all_replica(
  java.lang.String repository_id,
  java.lang.String instance_name)
  throws
  com.inprise.inprise.vbroker.ObjLocation.Fail
  このメソッドは，指定されたリポジトリ ID とインスタンス名を満足させる，インタフェースのインスタンスのオブジェクトリファレンスを取得します。
  - repository_id
    リポジトリ ID を含む文字列

- instance_name
  インスタンス名を含む文字列

このメソッドでは，次の例外が発生します。

Fail
  リポジトリ ID，またはインスタンス名が無効です。

- public com.inprise.vbroker.ObjLocation.Desc[ ] all_replica_descs(
  java.lang.String repository_id,
  java.lang.String instance_name)
  throws
  com.inprise.inprise.vbroker.ObjLocation.Fail
  このメソッドは，指定されたリポジトリ ID をインプリメントし，指定したインスタンス名を持つインタフェースの類似名インスタンスに対する完全な記述情報を取得します。
  - repository_id
    リポジトリ ID を含む文字列
  - instance_name
    インスタンス名を含む文字列

  このメソッドでは，次の例外が発生します。

  Fail
    リポジトリ ID，またはインスタンス名が無効です。
- public java.lang.String[ ] all_repository_ids( )
  throws
  com.inprise.inprise.vbroker.ObjLocation.Fail
  このメソッドは，任意の osagent が保持しているすべてのインタフェースを検索します。
  このメソッドでは次の例外が発生します。

  Fail
    リポジトリ ID が無効です。
- public void reg_trigger(
  com.inprise.vbroker.ObjLocation.TriggerDesc desc,
  com.inprise.vbroker.ObjLocation.TriggerHandler handler)
  throws
  com.inprise.inprise.vbroker.ObjLocation.Fail
  このメソッドは，トリガーハンドラを登録します。
  - desc

インスタンスの記述。インスタンス記述として指定できるのは，リポジトリ ID，インスタンス名，およびホスト名など，インスタンス情報の組み合わせです。インスタンス情報が多いほど，具体的にインスタンスの指定ができます。

- handler

  登録対象の TriggerHandler オブジェクト

このメソッドでは，次の例外が発生します。

Fail

該当するトリガーはありません。

- public void unreg_trigger(
  com.inprise.vbroker.ObjLocation.TriggerDesc desc,
  com.inprise.vbroker.ObjLocation.TriggerHandler handler)
  throws
  com.inprise.inprise.vbroker.ObjLocation.Fail
  このメソッドは，トリガーハンドラを登録解除します。

  - desc

    インスタンスの記述。インスタンス記述として指定できるのは，リポジトリ ID，インスタンス名，およびホスト名など，インスタンス情報の組み合わせです。インスタンス情報が多いほど，具体的にインスタンスの指定ができます。

  - handler

    登録解除対象の TriggerHandler オブジェクト

  注

  トリガーは何度も呼び出されることがあります。TriggerHandler は，トリガー記述を満たすオブジェクトがアクセスできるたびに呼び出されます。最初のインスタンスがいつアクセスできるかということを知りたい場合，エージェントの unreg_trigger( )メソッドを呼び出して，最初のイベントが見つかったあとはトリガーを登録解除してください。

  このメソッドでは，次の例外が発生します。

  Fail

  該当するトリガーはありません。

16.2 Desc

```
public interface Desc
```

このインタフェースは，オブジェクトの特性を記述するための情報を含んでいます。ロケーションサービスのメソッドには，Desc 構造体をパラメタとして必要とするものや，Desc 構造体または Desc 構造体の列を返すものがあります。

16.2.1 IDL の定義

```
struct Desc {
    CORBA::Object ref;
    ::IIOP::ProfileBodyValue iiop_locator;
    string repository_id;
    string instance_name;
    boolean activable;
    string agent_hostname;
};
```

16.2.2 Desc の変数

- public boolean **activable**
  オブジェクトがオブジェクト活性化デーモンに登録されている場合，true となります。オブジェクトが手動で起動され，osagent に登録されている場合，false となります。
- public org.omg.CORBA.Object **ref**
  定義するオブジェクトのリファレンスです。
- public com.inprise.vbroker.IIOP.ProfileBodyValue **iiop_locator**
  定義するオブジェクトのリファレンスです。
- public java.lang.String **repository_id**
  オブジェクトのリポジトリ ID です。
- public java.lang.String **instance_name**
  オブジェクトのインスタンス名です。
- public java.lang.String **agent_hostname**
  このオブジェクトが登録されているスマートエージェントが動作しているホストの名前です。

16.2.3 Desc のコンストラクタ

- public **Desc**(

org.omg.CORBA.Object **ref**,
com.inprise.vbroker.IIOP.ProfileBodyValue **iiop_locator**,
java.lang.String **repository_id**,
java.lang.String **instance_name**,
boolean **activable**,
java.lang.String **agent_hostname**)
指定したパラメタで初期化した Desc オブジェクトを生成します。

- ref
  記述するオブジェクトのリファレンス
- iiop_locator
  記述するオブジェクトのリファレンス
- repository_id
  オブジェクトのリポジトリ ID
- instance_name
  オブジェクトのインスタンス名
- activable
  オブジェクトがオブジェクト活性化デーモンに登録されている場合，true となります。オブジェクトが手動で起動され，osagent に登録されている場合，false となります。
- agent_hostname
  このオブジェクトが登録されているスマートエージェントが稼働しているホストの名前

16.2.4 Desc のメソッド

- public java.lang.String **toString**( )
  このメソッドは，オブジェクトの内容を含む文字列を返します。

16.3 Fail

```
public interface Fail extends org.omg.CORBA.UserException
```

この例外は，各種エラーを示すために Agent クラスが発生させます。データメンバ「FailReason」は，障害の性質を示します。

16.3.1 Fail 変数

- com.inprise.vbroker.ObjLocation.FailReason reason
  障害の性質を示す次の値のどれかを示します。
  - NO_AGENT_AVAILABLE
  - INVALID_REPOSITORY_ID
  - INVALID_OBJECT_NAME
  - NO_SUCH_TRIGGER
  - AGENT_ERROR

16.4 TriggerDesc

```
public final interface TriggerDesc
```

このインタフェースは，TriggerHandler を登録する，一つ以上のオブジェクトの特性を記述するために使用する情報を含みます。TriggerHandler については，「16.5 TriggerHandler」を参照してください。

できる限り広範囲にオブジェクトを監視するには，次に示すメンバを null に設定します。多くの情報を指定するほど，対象オブジェクトを絞り込めます。

16.4.1 IDL の定義

```
struct TriggerDesc {
   string repository_id;
   string instance_name;
   string host_name;
};
```

16.4.2 TriggerDesc の変数

- public java.lang.String host_name

  TriggerHandler で監視するオブジェクトのホスト名を表します。ネットワーク上のホストをすべて対象とするには，null を設定します。

- public java.lang.String instance_name

  TriggerHandler で監視するオブジェクトのインスタンス名を表します。監視可能なインスタンス名をすべて対象とするには，null を設定します。

- public java.lang.String repository_id

  このメンバは，TriggerHandler で監視するオブジェクトのリポジトリ ID を表します。監視可能なリポジトリ ID をすべて対象とするには，null を設定します。

16.4.3 TriggerDesc のコンストラクタ

- public TriggerDesc(java.lang.String.repository_id,
  java.lang.String instance_name,
  java.lang.String hostname)

  指定したパラメタで初期化した Desc オブジェクトを生成します。

  - repository_id

TriggerHandler が監視するオブジェクトのリポジトリ ID。監視可能なすべてのリポジトリ ID を含めるためには，null を設定する場合があります。

- instance_name

  TriggerHandler が監視するオブジェクトのインスタンス名。監視可能なすべてのインスタンス名を含めるためには，null を設定する場合があります。

- hostname

  TriggerHandler が監視するオブジェクトのあるホスト名。ネットワーク上のすべてのホストを含めるためには，null を設定する場合があります。

16.4.4 TriggerDesc のメソッド

- public java.lang.String toString( )

  このメソッドは，オブジェクトの内容を含む文字列を返します。

16.5 TriggerHandler

```
public interface TriggerHandler extends
    com.inprise.vbroker.CORBA.Object
```

このインタフェースは，トリガー記述を満たすオブジェクトがアクセスできるたびに呼び出されるコールバックオブジェクトです。TriggerHandler をインプリメントするには，_TriggerHandlerImplBase クラスを継承して，その impl_is_ready( )メソッドと impl_is_down( )メソッドをインプリメントしてください。

16.5.1 IDL の定義

```
interface TriggerHandler {
   void impl_is_ready(in Desc desc);
   void impl_is_down(in Desc desc);
};
```

16.5.2 TriggerHandler のメソッド

- public void **impl_is_ready**(
  com.inprise.vbroker.ObjLocation.Desc **desc**)
  このメソッドは，desc に一致するインスタンスがアクセスできるとロケーションサービスから呼び出されます。
  - desc
    インスタンスの記述。インスタンス記述として指定できるのは，リポジトリ ID，インスタンス名，およびホスト名など，インスタンス情報の組み合わせです。インスタンス情報が多いほど，具体的にインスタンスの指定ができます。
- public void **impl_is_down**(
  com.inprise.vbroker.ObjLocation.Desc **desc**)
  このメソッドは，desc に一致するインスタンスがアクセスできなくなると，ロケーションサービスから呼び出されます。
  - desc
    インスタンスの記述。インスタンス記述として指定できるのは，リポジトリ ID，インスタンス名，およびホスト名など，インスタンス情報の組み合わせです。インスタンス情報が多いほど，具体的にインスタンスの指定ができます。

17 コマンドラインオプション（Java）

この章では，Java 言語を使用してプログラミングするときに，オブジェクトリクエストブローカー（ORB）およびロケーションサービスで設定するオプションについて説明します。

オブジェクトリクエストブローカー（ORB）およびロケーションサービスは，VisiBroker 3.x 形式で使用できます。しかし，現在のバージョンに対応したプロパティを使用してください。VisiBroker 4.x 形式で使用する場合については，「18.　Borland Enterprise Server VisiBroker プロパティ（Java）」を参照してください。

17.1　ORB オプションの設定方法

ORB の初期設定オプションを設定するには，次に示す方法があります。

- vbj コマンドでコマンドラインパラメタを使用する
- vbj コマンドのコマンドラインパラメタを使用して Java アプリケーションを起動する
- アプレットにオプションを設定する
- メソッドでプログラム的にオプションを設定する

17.1.1　vbj コマンドでコマンドラインパラメタを使用する

vbj コマンドを使用して，アプリケーション起動時の ORB の動作をカスタマイズするコマンドラインパラメタを定義できます。コマンドラインから vbj コマンドを使用する場合は，次の例のように等号（=）を使用して値を設定します。

```
vbj -DORBdebug=true Server
```

注

java コマンドを使用して，アプリケーション起動時の ORB の動作をカスタマイズするコマンドラインプロパティを定義することもできます。vbj コマンドを使用するときは，ORB が環境変数も検証します。

17.1.2　vbj コマンドを使用して Java アプリケーションを起動する

vbj コマンドのコマンドラインパラメタを使用して，Java アプリケーションを起動することもできます。コマンドラインパラメタを指定するときは，ハイフン（-）のあとに D を付けないように注意してください。また，値を設定するときに等号（=）を使用しないように注意してください。次に例を示します。

```
vbj <Javaアプリケーション名> -ORBdebug true
```

注

vbj コマンドを使用して Java アプリケーションを起動する場合は，ORB.init を呼び出してパラメタを渡してください。詳細については，「17.2　ORB.init( )メソッド」のコードサンプル 17-1 を参照してください。

17.1.3　アプレットにオプションを設定する

アプレットにオプションを設定する場合は，パラメタ名と値を使用します。次に例を示します。

```
<param name=ORBInitRef value=NameService=corbaname::TestHost:20003/>
```

17.1.4　メソッドでプログラム的にオプションを設定する

ORB の初期化メソッド（ORB.init( )）を使用して，プログラム的にオプションを設定できます。これらのメソッドの使用方法の詳細については，「17.2　ORB.init( )メソッド」を参照してください。

17.2 ORB.init( )メソッド

```
public static ORB init(String[ ] args, Properties props)
```

ORB.init( )メソッドは，使用するスマートエージェントの IP アドレスやポート番号などのオプションを設定するときにアプリケーションが使用します。各パラメタはアプリケーション起動時にパラメタとして渡されます。

ORB.init( )メソッドに渡されるパラメタは，アプリケーションのメインルーチンに渡されるパラメタと同じです。ORB.init( )メソッドが認識できないパラメタは無視します。

コードサンプル 17-1 では，ORB.init( )にスマートエージェントのポートを指定するパラメタを渡しています。

コードサンプル 17-1　パラメタを指定した ORB.init()の使用例

```
public static void main(String[ ] args) {
    ...
    java.util.Properties props =
                              new java.util.Properties( );
    props.put("ORBagentPort", "9898");
    org.omg.CORBA.ORB orb =
                      org.omg.CORBA.ORB.init(args, props);
   ...
}
```

17.2.1 ORB オプション

次の表に，ORB.init( )メソッドのオプションの概要を示します。

表 17-1　ORB.init オプション（Java）

オプション	説明
ORBagentAddr <hostname | ip_address>	クライアントが使うスマートエージェントを実行するホストのホスト名または IP アドレスを指定します。 スマートエージェントが指定のアドレスで見つからない場合，またはこのオプションの指定がない場合，ブロードキャストメッセージでスマートエージェントを探します。
ORBagentAddrFile <file_name>	デフォルトファイルの agentaddr の代わりに使用するファイルを指定します。
ORBagentNoFailOver <false | true>	VisiBroker アプリケーションが通信している osagent が終了した場合，ほかの osagent との通信を行うかどうかを指定します。 true を指定した場合，VisiBroker アプリケーションは先に通信を行っていた osagent にだけ，再度通信を試みます。

オプション	説明
	false を指定した場合，VisiBroker アプリケーションは先に通信を行っていた osagent だけでなくほかの osagent にも再度通信を試みます。 デフォルトは false です。 不正な値を指定した場合，true を設定したときと同じ動作をします。
ORBagentPort <port_number>	スマートエージェントのポート番号を指定します。このオプションは，複数の ORB ドメインが必要なときに便利です。このオプションを指定しないと，デフォルトのポート番号 14000 が使用されます。 アプリケーション内で ORB.init( )メソッドの第 2 引数に ORBagentPort または ORBdisableLocator のプロパティを設定した場合，その効果はありません。ORB オプションの値を有効にする場合は，コマンドライン上で-D オプションで指定してください。
ORBalwaysProxy <false | true>	クライアントが常にゲートキーパーを使用して接続する必要があるかどうかを指定します。デフォルトは false です。 true を設定した場合は，ORBgatekeeperIOR オプションも設定してください。 不正な値を指定した場合，true を設定したときと同じ動作をします。
ORBalwaysTunnel <false | true>	クライアントが常に HTTP を使用してゲートキーパーに接続する必要があるかどうかを指定します。デフォルトは false です。 true を設定した場合は，ORBgatekeeperIOR オプションも設定してください。 不正な値を指定した場合，true を設定したときと同じ動作をします。
ORBconnectionMax <#>	可能なコネクションの最大数を指定します。 このオプションを指定しない場合，コネクション数は無制限です。
ORBconnectionMaxIdle <#>	コネクションが非アクティブな状態の最大監視時間を秒単位で指定します。最大監視時間を経過してもコネクションが非アクティブなままの場合，VisiBroker がコネクションを終了します。 このオプションはインターネットアプリケーションで設定します。 0 を設定した場合，監視しません。デフォルトは 0 です。 最大アイドル時間の監視は，プロパティ vbroker.orb.gcTimeout の設定値（デフォルト 30 秒）の範囲の誤差があります。
ORBdebug <false | true>	デバッグ機能を有効にします。

オプション	説明
ORBDefaultInitRef	デフォルトの初期リファレンスを指定します。nameserv には，ORBDefaultInitRef を使用しないでください。ORBInitRef を使用してください。
ORBdisableAgentCache <false | true>	スマートエージェントのキャッシュを有効にします。デフォルトは false です。 不正な値を指定した場合，true を設定したときと同じ動作をします。
ORBdisableGatekeeperCallbacks <false | true>	ゲートキーパーのコールバックを有効（または無効）にします。デフォルトは false です。 false を設定した場合は，ゲートキーパーのコールバックが有効となります。true を設定した場合は，ORBgatekeeperIOR オプションも設定してください。
ORBdisableLocator <false | true>	スマートエージェントとゲートキーパーを無効にします。 アプリケーション内で ORB.init( )メソッドの第 2 引数に ORBagentPort または ORBdisableLocator のプロパティを設定した場合，その効果はありません。ORB オプションの値を有効にする場合は，コマンドライン上で-D オプションで指定してください。 不正な値を指定した場合，true を設定したときと同じ動作をします。
ORBgatekeeperIOR <URL>	IOR に対応する URL を指定します。
ORBgcTimeout <#>	ORB のガーベッジコレクションを実行する周期（秒）を指定します。デフォルトは 30（秒）です。
ORBInitRef	初期リファレンスを指定します。
ORBmbufSize <buffer_size>	Borland Enterprise Server VisiBroker がオペレーションリクエストを処理するときに使用する中間バッファのサイズ（バイト）を指定します。 ORB は，VisiBroker の旧バージョンよりも複雑なバッファ管理で性能の向上を図っています。 送信バッファと受信バッファのデフォルトサイズは 4096 バイトです。送受信データがデフォルトサイズより大きい場合は，リクエスト／応答の実行ごとに追加バッファが割り当てられます。デフォルトサイズより大きいデータを頻繁に送信するアプリケーションを使用する場合は，このシステムプロパティでバイト数を指定してデフォルトサイズを調整すると，効率的にバッファ管理ができます。
ORBnullString <false | true>	true を設定すると，NULL 文字列のマーシャリングが有効になります。デフォルトは false です。
ORBwarn <#>	出力する警告メッセージのレベルを，0，1，2 のどれかの値で設定します。 0：デフォルトです。警告メッセージを出力しません。 1：ユーザ作成コードからの CORBA 以外の例外と，その例外のスタックトレースを出力します。

オプション	説明
	2：1 に加えて，CORBA の例外と，その例外のスタックトレースを出力します。

17.3　ロケーションサービスオプション

VisiBroker 4.x では，コマンドラインオプションの LOCdebug，LOCtimeout，および LOCverify をサポートしていません。これらのオプションの代わりに使用できるプロパティとデフォルトの詳細については，「18.6　ロケーションサービスプロパティ」を参照してください。

18 Borland Enterprise Server VisiBroker プロパティ (Java)

この章では，Java 言語でプログラミングする場合に，Borland Enterprise Server VisiBroker で設定できるプロパティについて説明します。

この章で説明しているプロパティは現在のバージョンでも使用できます。VisiBroker 3.x 形式で使用する場合については，「17.　コマンドラインオプション（Java）」を参照してください。

18.1　プロパティの設定方法

プロパティを設定するには，次に示す方法があります。

- vbj コマンドでコマンドラインパラメタを使用する
- vbj コマンドのコマンドラインパラメタを使用して Java アプリケーションを起動する
- アプレットにプロパティを設定する
- メソッドでプログラム的にプロパティを設定する

18.1.1　vbj コマンドでコマンドラインパラメタを使用する

vbj コマンドを使用して，アプリケーション起動時の ORB の動作をカスタマイズするコマンドラインパラメタを定義できます。コマンドラインから vbj コマンドを使用する場合は，次の例のように等号（=）を使用して値を設定します。

```
vbj -J-Dvbroker.agent.port=14001 Server
```

注

java コマンドを使用して，アプリケーション起動時の ORB の動作をカスタマイズするコマンドラインプロパティを定義することもできます。vbj コマンドを使用するときは，ORB が環境変数も検証します。

18.1.2　vbj コマンドを使用して Java アプリケーションを起動する

vbj コマンドのコマンドラインパラメタを使用して，Java アプリケーションを起動することもできます。コマンドラインパラメタを指定するときは，ハイフン（-）のあとに D を付けないように注意してください。また，値を設定するときに等号（=）を使用しないように注意してください。次に例を示します。

```
vbj 〈Javaアプリケーション名〉 -vbroker.agent.port  14001
```

注

vbj コマンドを使用して Java アプリケーションを起動する場合は，ORB.init を呼び出してパラメタを渡してください。詳細については，「18.1.4　メソッドでプログラム的にプロパティを設定する」のコードサンプル 18-1 を参照してください。

18.1.3　アプレットにプロパティを設定する

アプレットにプロパティを設定する場合は，パラメタ名と値を使用します。次に例を示します。

```
<param name=vbroker.agent.port value=14001>
```

18.1.4　メソッドでプログラム的にプロパティを設定する

ORB の初期化メソッド（ORB.init( )）を使用して，プログラム的にプロパティを設定できます。これらのメソッドの使用例をコードサンプル 18-1 に示します。

コードサンプル 18-1　パラメタを指定した ORB.init()の使用例

```
public static void main(String[ ] args){
   ...
   java.util.Properties props = new java.lang.util.Properties();
   props.put(”vbroker.agent.port”, ”14001”);
   org.omg.CORBA.ORB orb =org.omg.CORBA.ORB.init(args, props);
   ...
}
```

18.2 RMI-IIOP プロパティ

RMI-IIOP のプロパティを次の表に示します。

表 18–1 RMI-IIOP プロパティ（Java）

プロパティ	デフォルト	説明
javax.rmi.CORBA.StubClass	com.inprise.vbroker.rmi.CORBA.StubImpl	RMI-IIOP スタブすべての継承元である Stub ベースクラスのインプリメンテーションの名前を指定します。
javax.rmi.CORBA.UtilClass	com.inprise.vbroker.rmi.CORBA.UtilImpl	共通オペレーションを実行するためのスタブと tie を提供する Utility クラスのインプリメンテーションの名前を指定します。
javax.rmi.CORBA.PortableRemoteObjectClass	com.inprise.vbroker.rmi.CORBA.PortableRemoteObjectImpl	RMI-IIOP サーバのインプリメンテーションオブジェクトが javax.rmi.PortableRemoteObject を継承することを指定します。 または，RMI-IIOP リモートインタフェースを単にインプリメントしてから，exportObject メソッドを使用してそのオブジェクトをそれぞれサーバオブジェクトとして登録できることを指定します。
java.rmi.server.codebase	null	未知のクラスをサーバが見つけることのできる場所を指定します。 日本語や 2 バイトコード文字を含むパスを指定しないでください。指定した場合，POA の create_POA( )メソッドで DATA_CONVERSION 例外となります。
java.rmi.server.useCodebaseOnly	false	サーバが未知のクラスを見つけられるかどうかを指定します。true を設定すると，クライアントがリモートクラスをサーバに送信する場合であっても，サーバはリモートクラスを見つけることはできません。

18.3 osagent（スマートエージェント）プロパティ

osagent（スマートエージェント）のプロパティを次の表に示します。

表 18–2 osagent（スマートエージェント）プロパティ（Java）

プロパティ	デフォルト	旧プロパティ	説明
vbroker.agent.addr	null	ORBagentAddr	osagent が動作しているホストの IP アドレスまたはホスト名を，VisiBroker アプリケーションに指定します。デフォルトの null に設定すると，VisiBroker アプリケーションは OSAGENT_ADDR 環境変数の値を使用します。 OSAGENT_ADDR 環境変数が未設定のときは，osagent がローカルホストで動作していると仮定します。
vbroker.agent.addrFile	null	ORBagentAddrFile	osagent が IP アドレスまたはホスト名をどこで見つけられるかどうかを記述したファイルを指定します。
vbroker.agent.debug	false	ORBdebug	true を設定すると，システムは VisiBroker アプリケーションと osagent との間での通信のデバッグ情報を表示します。
vbroker.agent.enableCache	true	ORBagentCache	true を設定すると，VisiBroker アプリケーションは IOR をキャッシュできます。 不正な値を指定した場合，false を設定したときと同じ動作をします。
vbroker.agent.enableLocator	true	ORBdisableLocator	false を設定すると，VisiBroker アプリケーションは osagent と通信できません。 不正な値を指定した場合，false を設定したときと同じ動作をします。
vbroker.agent.failOver	true	ORBagentNoFailOver	VisiBroker アプリケーションが通信している osagent が終了した場合，ほかの osagent との通信を行うかどうかを指定します。 true を指定した場合，VisiBroker アプリケーションは先に通信を行っていた osagent だけでなくほかの osagent にも再度通信を試みます。 false を指定した場合，VisiBroker アプリケーションは先に通信を行っていた osagent にだけ，再度通信を試みます。

プロパティ	デフォルト	旧プロパティ	説明
			不正な値を指定した場合，false を設定したときと同じ動作をします。
vbroker.agent.port	14000	ORBagentPort	ネットワーク上のドメインを定義するポート番号を指定します。VisiBroker アプリケーションと osagent に同一のポート番号を設定すると，それらを連携させることができます。このプロパティは OSAGENT_PORT 環境変数と同じ働きをします。 注 0 以外を指定する場合，有効範囲は 5001～65535 の範囲の整数値です。それ以外の値を指定した場合，動作の保証はできません。

18.4 ORB プロパティ

VisiBroker ORB のプロパティを次の表に示します。

表 18–3 ORB プロパティ（Java）

プロパティ	デフォルト	旧プロパティ	説明
vbroker.orb.admDir	<環境変数 TPDIR の設定値>/adm	該当しません。	さまざまなシステムファイルが存在する管理ディレクトリを指定します。このプロパティは VBROKER_ADM 環境変数を使って設定できます。環境変数 TPDIR については，マニュアル「TPBroker ユーザーズガイド」を参照してください。
vbroker.orb.alwaysProxy	false	ORBalwaysProxy	true を設定すると，クライアントは常にゲートキーパーを使用してサーバに接続しなければならないことを指定します。
vbroker.orb.alwaysSecure	false	該当しません。	true を設定すると，クライアントは常にサーバへセキュアに接続しなければならないことを指定します。
vbroker.orb.alwaysTunnel	false	ORBalwaysTunnel	true を設定すると，クライアントは常にサーバに HTTP トンネル（IIOP のオブジェクトラッパー）を使用して接続することを指定します。
vbroker.orb.autoLocateStubs	false	該当しません。	オブジェクトリファレンスの読み込み時にスタブを検索する機能を有効にします。この機能は，渡された正式なクラス引数の一般オブジェクトやスタブではなく，オブジェクトのリポジトリ ID に基づいて，read_Object を使用して実行します。不正な値を指定した場合，true を設定したときと同じ動作をします。
vbroker.orb.bidOrder	inprocess:liop:ssl:iiop:proxy:hiop:locator	該当しません。	各トランスポートの相対的な重要度を指定します。トランスポートの優先順位は，次のように割り当てられています。 1. inprocess 2. liop 3. ssl 4. iiop 5. proxy 6. hiop 7. locator

プロパティ	デフォルト	旧プロパティ	説明
			例えば，IOR に LIOP と IIOP の両プロファイルが含まれている場合は，LIOP が優先されます。IIOP が使用されるのは，LIOP が失敗したときだけです。ただし，vbroker.orb.bidOrder と vbroker.orb.bids.critical に同時に優先順位が設定されている場合は，vbroker.orb.bids.critical の設定が有効となります。
vbroker.orb.bids.critical	inprocess	該当しません。	vbroker.orb.bidOrder と vbroker.orb.bids.critical に同時に優先順位が設定されている場合は，vbroker.orb.bids.critical の設定が有効となります。 vbroker.orb.bids.critical で複数の値が設定されている場合は，vbroker.orb.bidOrder に基づいて相対的な重要度が決まります。
vbroker.orb.defAddrMode	0（Key）	該当しません。	クライアントの VisiBroker ORB が使用するデフォルトアドレッシングモードを指定します。0 を設定するとアドレッシングモードは Key となります。1 を設定するとアドレッシングモードは Profile となります。2 を設定するとアドレッシングモードは IOR となります。
vbroker.orb.bufferCacheTimeout	6000	該当しません。	メッセージチャンクを破棄する前にキャッシュに保存しておく時間を指定します。
vbroker.orb.compliantExceptions	true	該当しません。	org.omg.CORBA.Object.non_existent メソッドの延長でサーバにアクセスできない場合，例外を返すか true を返すかを指定します。true を設定した場合は TRANSIENT 例外を返し，false を設定した場合は true を返します。
vbroker.orb.debug	false	該当しません。	true を設定すると，ORB はデバッグ情報を表示できるようになります。
vbroker.orb.defaultInitRef	null	ORBDefaultInitRef	デフォルトの初期リファレンスを指定します。
vbroker.orb.dynamicLibs	null	該当しません。	VisiBroker ORB が使用できるサービスの一覧を指定します。
vbroker.orb.embedCodeset	true	該当しません。	IOR が作成されると，VisiBroker ORB は，codeset コンポーネントを

プロパティ	デフォルト	旧プロパティ	説明
			IOR に埋め込みます。これによって問題が発生する非準拠の ORB もあります。off を設定すると，VisiBroker ORB は codeset コンポーネントを埋め込みません。false を設定すると，クライアントとサーバとの間ではネゴシエーションなしで char 型と wchar 型が変換されます。 不正な値を指定した場合，true を設定したときと同じ動作をします。
vbroker.orb.enableVB4backcompat	false	該当しません。	VisiBroker 4.x で GIOP 1.2 に準拠していない動作に対処できるようにするためのプロパティです。クライアントが VisiBroker 4.x の場合は，このフラグに true を設定する必要があります。これはサーバ側専用のフラグです。クライアント側で設定する必要はありません。
vbroker.orb.enableBiDir	none	該当しません。	選択的に双方向コネクションを確立します。 クライアント側で「vbroker.orb.enableBiDir=client」と定義し，サーバ側で「vbroker.orb.enableBiDir=server」と定義すると，ゲートキーパーの vbroker.orb.enableBiDir の値でコネクションの状態が決まります。このプロパティの値は，server，client，both，または none です。詳細については，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「双方向通信」の記述を参照してください。
vbroker.orb.enableKeyId	true	該当しません。	false を設定すると，クライアントリクエストのキー ID を無効にします。
vbroker.orb.enableNullString	false	ORBnullString	true を設定すると，NULL 文字列のマーシャリングが有効になります。 不正な値を指定した場合，true を設定したときと同じ動作をします。
vbroker.orb.fragmentSize	0	該当しません。	GIOP メッセージのフラグメントサイズを指定します。チャンクサイズの整数倍である値を指定します。0 を指定するとフラグメント化をしません。

プロパティ	デフォルト	旧プロパティ	説明
vbroker.orb.initRef	null	ORBInitRef	初期リファレンスを指定します。
vbroker.orb.streamChunkSize	4096	該当しません。	GIOP メッセージのチャンクサイズを指定します。2 の累乗を指定します。 設定できる値の最小値は 16 です。
vbroker.orb.gcTimeout	30	ORBgcTimeout	未使用の重要リソースを削除する場合のタイムアウトを秒数で指定します。
vbroker.orb.logger.appName	VBJ-Application	該当しません。	ログに記録するアプリケーション名を指定します。
vbroker.orb.logger.catalog	com.inprise.vbroker.Logging.ORBMsgs	該当しません。	ロギングが有効の場合，ORB が使用するメッセージのメッセージカタログを指定します。
vbroker.orb.logger.output	stdout	該当しません。	ログの出力先を指定します。指定できるのは標準出力またはファイル名です。
vbroker.orb.logLevel	emerg	該当しません。	ログに記録するメッセージのレベルを指定します。 デフォルトの「emerg」は，VisiBroker システムが正常に通信の処理ができないときに必要最小限のメッセージをログに記録することを指定します。
vbroker.orb.procId	0	該当しません。	サーバのプロセス ID を指定します。
vbroker.orb.rebindForward	0	該当しません。	クライアントが新規フォワード先オブジェクトへの接続に失敗した場合，元のオブジェクト（フォワード元）への再接続の回数を指定します。 指定できる値の上限値は，2147483647 です。
vbroker.orb.sendLocate	false	該当しません。	true を設定すると，VisiBroker システムは，IIOP 1.2 の対象オブジェクトを呼び出す前に強制的にロケートリクエストを送信します。
vbroker.orb.systemLibs.applet	com.inprise.vbroker.IIOP.Init, com.inprise.vbroker.LIOP.Init, com.inprise.vbroker.qos.Init,	該当しません。	アプレットにロードされたシステムライブラリのリストを提供します。

プロパティ	デフォルト	旧プロパティ	説明
	com.inprise.vbroker.URLNaming.Init, com.inprise.vbroker.HIOP.Init, com.inprise.vbroker.firewall.Init, com.inprise.vbroker.dynamic.Init, com.inprise.vbroker.naming.Init		
vbroker.orb.systemLibs.application	com.inprise.vbroker.IIOP.Init, com.inprise.vbroker.LIOP.Init, com.inprise.vbroker.qos.Init, com.inprise.vbroker.ds.Init, com.inprise.vbroker.URLNaming.Init, com.inprise.vbroker.dynamic.Init, com.inprise.vbroker.ir.Init, com.inprise.vbroker.naming.Init	該当しません。	アプリケーションにロードされたシステムライブラリのリストを提供します。
vbroker.orb.tcIndirection	true	該当しません。	タイプコードを書き込むときに間接参照を無効にすることを指定します。VisiBroker ORB をほかのベンダからインターオペレートする場合にこのプロパティが必要となることがあります。 false を設定すると，再帰型タイプコードのマーシャリングができなくなります。
vbroker.orb.warn	0	ORBwarn	出力するメッセージの警告レベルを 0，1，または 2 で指定します。

18.5 POA プロパティ

POA のプロパティを次の表に示します。

表 18–4 POA プロパティ（Java）

プロパティ	デフォルト	説明
vbroker.poa.logLevel	emerg	ログに記録するメッセージのレベルを指定します。デフォルトの「emerg」は，システムが使用できないときやパニック状態のときにメッセージをログに記録することを指定します。

18.6 ロケーションサービスプロパティ

ロケーションサービスのプロパティを次の表に示します。

表 18–5 ロケーションサービスプロパティ（Java）

プロパティ	デフォルト	説明
vbroker.locationservice.debug	false	true を設定すると，ロケーションサービスはデバッグ情報を表示できるようになります。
vbroker.locationservice.verify	false	true を設定すると，ロケーションサービスは osagent から送られたオブジェクトリファレンスで参照されるオブジェクトがあるかをチェックできるようになります。BY_INSTANCE で登録されたオブジェクトだけをチェックします。OAD または BY_POA ポリシーで登録されたオブジェクトのチェックはしません。
vbroker.locationservice.timeout	1	ロケーションサービスとの通信時の接続，受信，および送信のタイムアウトを指定します。

18.7　ネーミングサービスプロパティ

ネーミングサービスプロパティについては，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「ネーミングサービスプロパティ」の記述を参照してください。

18.8 OAD プロパティ

設定可能な OAD のプロパティを次の表に示します。

表 18–6 認定可能な OAD プロパティ（Java）

プロパティ	デフォルト	説明
vbroker.oad.spawnTimeOut	20	OAD が実行形式ファイルを生成したあと，オブジェクトからコールバックが何秒間来なければ NO_RESPONSE 例外を出力するかを指定します。
vbroker.oad.verbose	false	OAD がオペレーションの詳細情報を出力できるようにします。
vbroker.oad.readOnly	false	true を設定すると，OAD インプリメンテーションの登録，登録解除，および変更ができなくなります。
vbroker.oad.iorFile	oadj.ior	OAD の文字列化 IOR のファイル名を指定します。
vbroker.oad.quoteSpaces	false	コマンドを引用するかどうかを指定します。
vbroker.oad.killOnUnregister	false	生成したサーバの登録を解除した場合に，それらを kill するかどうかを指定します。
vbroker.oad.verifyRegistration	false	オブジェクトの登録を確認するかどうかを指定します。

プロパティファイルにオーバーライドできない OAD プロパティを次の表に示します。なお，このプロパティは，環境変数またはコマンドラインでオーバーライドできます。

表 18–7 プロパティファイルにオーバーライドできない OAD プロパティ（Java）

プロパティ	デフォルト	説明
vbroker.oad.implName	impl_rep	インプリメンテーションリポジトリのファイル名を指定します。
vbroker.oad.implPath	null	インプリメンテーションリポジトリを格納しているディレクトリを指定します。
vbroker.oad.path	null	OAD のディレクトリを指定します。
vbroker.oad.systemRoot	null	ルートディレクトリを指定します。
vbroker.oad.winDir	null	Windows ディレクトリを指定します。
vbroker.oad.vbj	vbj	VisiBroker ディレクトリを指定します。

18.9 インタフェースリポジトリプロパティ

インタフェースリポジトリ（IR）のプロパティを次の表に示します。

表 18–8 インタフェースリポジトリプロパティ（Java）

プロパティ	デフォルト	説明
vbroker.ir.debug	false	true を設定すると，IR リゾルバはデバッグ情報を表示できるようになります。
vbroker.ir.ior	null	vbroker.ir.name プロパティにデフォルト値の「null」が設定されている場合，ORB はこのプロパティを使用して IR を捜し出します。
vbroker.ir.name	null	VisiBroker ORB が IR を探すときに使用する名前を指定します。

18.10 URL ネーミングプロパティ

URL ネーミングサービスのプロパティを次の表に示します。

表 18–9 URL ネーミングプロパティ（Java）

プロパティ	デフォルト	説明
vbroker.URLNaming.allowUserInteraction	true	true を設定すると，URL ネーミングサービスは GUI の使用を開始できるようになります。 不正な値を指定した場合，false を設定したときと同じ動作をします。
vbroker.URLNaming.debug	false	true を設定すると，URL ネーミングサービスはデバッグ情報を表示できるようになります。

18.11　クライアント側コネクションプロパティ

クライアント側コネクションのプロパティを次の表に示します。

表 18–10　クライアント側コネクションプロパティ（Java）

プロパティ	デフォルト	説明
vbroker.ce.iiop.ccm.connectionMax	0	1 クライアントの合計コネクション数の上限を指定します。この値は，アクティブなコネクションの数です。デフォルトの 0 を指定すると，クライアントは以前のアクティブなコネクションをクローズしません。
vbroker.ce.iiop.ccm.connectionMaxIdle	0	コネクションが非アクティブな状態の最大監視時間を秒単位で指定します。最大監視時間を経過してもコネクションが非アクティブなままの場合，VisiBroker がコネクションを終了します。 このプロパティはインターネットアプリケーションで設定します。0 を設定した場合，監視しません。デフォルトは 0 です。
vbroker.ce.iiop.connection.tcpNoDelay	true	false を設定すると，ソケットのバッファリング機能が有効になります。true（デフォルト）では，すべてのパケットが使用可能になったときに直ちに送信されるように，ソケットのバッファリング機能を無効にします。
vbroker.ce.iiop.ccm.type	Pool	クライアントが使用するクライアントコネクション管理の種類を指定します。デフォルト値の「Pool」はコネクションプールの意味です。

18.12　クライアント側プロセス内コネクションプロパティ

クライアント側プロセス内コネクションのプロパティを次の表に示します。

表 18–11　クライアント側プロセス内コネクションプロパティ（Java）

プロパティ	デフォルト	説明
vbroker.ce.inprocess.ccm.bid	9488	POA bidder の bid 値を指定します。ここで指定した値は，VisiBroker ORB がクライアントコネクションを処理するプロトコルを選択する自動プロセスに作用します。
vbroker.ce.iiop.ccm.bid	10000	iiop bidder の bid 値を指定します。ここで指定した値は，VisiBroker ORB がクライアントコネクションを処理するプロトコルを選択する自動プロセスに作用します。

18.13　サーバ側エンジンプロパティ

サーバ側のサーバエンジンのプロパティを次の表に示します。

表 18–12　サーバ側エンジンプロパティ（Java）

プロパティ	デフォルト	説明
vbroker.se.default	iiop_tp	デフォルトサーバエンジンを指定します。
vbroker.se.<se>.scm.<scm>.listener.giopVersion	1.2	このプロパティを使用すると，古い VisiBroker ORB では未知のマイナー GIOP バージョンを正しく処理できないインターオペラビリティの問題を解決できます。このプロパティの値は，1.0，1.1，または 1.2 です。例えば，ネームサービスで GIOP 1.1 の ior を生成する場合は，次のように指定します。 `nameserv -VBJprop vbroker.se.iiop_tp.scm.iiop_tp.listener.giopVersion=1.1`

18.14 サーバ側スレッドセッション IIOP_TS プロパティ，および IIOP_TS コネクションプロパティ

サーバ側スレッドセッション IIOP_TS/IIOP_TS コネクションのプロパティを次の表に示します。

表 18–13 サーバ側スレッドセッション IIOP_TS/IIOP_TS コネクションプロパティ（Java）

プロパティ	デフォルト	説明
vbroker.se.iiop_ts.host	null	該当するサーバエンジンが使用するホスト名を指定します。デフォルト値の「null」は，システムからホスト名を使用することを指定します。 日本語や 2 バイトコード文字を含むパスを指定しないでください。指定した場合，POA の create_POA( )メソッドで DATA_CONVERSION 例外となります。
vbroker.se.iiop_ts.proxyHost	null	IOR 文字列に出力するホスト名を指定します。IOR 文字列にホスト名で出力する場合は，このプロパティにホスト名を指定してください。このプロパティの指定を省略した場合は，vbroker.se.iiop_ts.host の指定に従います。 日本語や 2 バイトコード文字を含むパスを指定しないでください。指定した場合，POA の create_POA( )メソッドで DATA_CONVERSION 例外となります。
vbroker.se.iiop_ts.scms	iiop_ts	サーバコネクションマネージャ名の一覧を指定します。
vbroker.se.iiop_ts.scm.iiop_ts.manager.type	Socket	サーバコネクションマネージャの種別を指定します。
vbroker.se.iiop_ts.scm.iiop_ts.manager.connectionMax	0	サーバが許可するコネクション数の最大値を指定します。デフォルト値の 0 は，コネクション数を制限しないことを表します。
vbroker.se.iiop_ts.scm.iiop_ts.manager.connectionMaxIdle	0	アイドルなコネクションをクローズするかどうかをサーバが判定するためのタイムアウトを秒数で指定します。最大アイドル時間の監視は，プロパティ vbroker.orb.gcTimeout の設定値（デフォルト 30 秒）の範囲の誤差があります。
vbroker.se.iiop_ts.scm.iiop_ts.listener.type	IIOP	リスナーが使用するプロトコルの種別を指定します。
vbroker.se.iiop_ts.scm.iiop_ts.listener.port	0	ホスト名プロパティに使用するポート番号を指定します。デフォルト値の 0 は，システムはポート番号を無作為に選び取ることを表します。65536 より小さい値を指定してください。65536 以上の値を指定すると，create_POA( )で IllegalArgumentException が発生します。
vbroker.se.iiop_ts.scm.iiop_ts.listener.proxyPort	0	プロキシホスト名プロパティに使用するプロキシポート番号を指定します。デフォルト値の 0 は，システムはポート番号を無作為に選択することを表します。
vbroker.se.iiop_ts.scm.iiop_ts.dispatcher.type	"ThreadSession"	サーバコネクションマネージャに使用するスレッドディスパッチャの種別を指定します。

18.15 サーバ側スレッドプール IIOP_TP プロパティ，および IIOP_TP コネクションプロパティ

サーバ側スレッドプール IIOP_TP/IIOP_TP コネクションのプロパティを次の表に示します。

表 18–14 サーバ側スレッドプール IIOP_TP/IIOP_TP コネクションプロパティ（Java）

プロパティ	デフォルト	説明
vbroker.se.iiop_tp.host	null	該当するサーバエンジンが使用するホスト名を指定します。デフォルト値の「null」は，システムからホスト名を使用することを指定します。 日本語や 2 バイトコード文字を含むパスを指定しないでください。指定した場合，POA の create_POA( )メソッドで DATA_CONVERSION 例外となります。
vbroker.se.iiop_tp.proxyHost	null	IOR 文字列に出力するホスト名を指定します。IOR 文字列にホスト名で出力する場合は，このプロパティにホスト名を指定してください。このプロパティの指定を省略した場合は，vbroker.se.iiop_tp.host の指定に従います。 日本語や 2 バイトコード文字を含むパスを指定しないでください。指定した場合，POA の create_POA( )メソッドで DATA_CONVERSION 例外となります。
vbroker.se.iiop_tp.scms	iiop_tp	サーバコネクションマネージャ名の一覧を指定します。
vbroker.se.iiop_tp.scm.iiop_tp.connection.tcpNoDelay	true	false を設定すると，ソケットのバッファリング機能が有効になります。true（デフォルト）では，すべてのパケットが使用可能になったときに直ちに送信されるように，ソケットのバッファリング機能を無効にします。
vbroker.se.iiop_tp.scm.iiop_tp.manager.type	Socket	サーバコネクションマネージャの種別を指定します。
vbroker.se.iiop_tp.scm.iiop_tp.manager.connectionMax	0	サーバが許可するキャッシュコネクション数の最大値を指定します。デフォルト値の 0 は，コネクション数を制限しないことを表します。
vbroker.se.iiop_tp.scm.iiop_tp.manager.connectionMaxIdle	0	アイドルなコネクションをクローズするかどうかをサーバが判定するためのタイムアウトを秒数で指定します。最大アイドル時間の監視は，プロパティ vbroker.orb.gcTimeout の設定値（デフォルト 30 秒）の範囲の誤差があります。
vbroker.se.iiop_tp.scm.iiop_tp.listener.type	IIOP	リスナーが使用するプロトコルの種別を指定します。
vbroker.se.iiop_tp.scm.iiop_tp.listener.port	0	ホスト名プロパティに使用するポート番号を指定します。デフォルト値の 0 は，システムはポート番号を無作為に選び取ることを表します。65536 より小さい値を指定してください。65536 以上の値を指定すると，create_POA( )で IllegalArgumentException が発生します。
vbroker.se.iiop_tp.scm.iiop_tp.listener.proxyPort	0	プロキシホスト名プロパティに使用するプロキシポート番号を指定します。デフォルト値の 0 は，システムはポート番号を無作為に選択することを表します。

プロパティ	デフォルト	説明
vbroker.se.iiop_tp.scm.iiop_tp.dispatcher.type	ThreadPool	サーバコネクションマネージャに使用するスレッドディスパッチャの種別を指定します。
vbroker.se.iiop_tp.scm.iiop_tp.dispatcher.threadMin	0	サーバコネクションマネージャが生成できるスレッド数の下限を指定します。
vbroker.se.iiop_tp.scm.iiop_tp.dispatcher.threadMax	0	サーバコネクションマネージャが生成できるスレッド数の上限を指定します。デフォルト値の 0 は，スレッド数を制限しないことを表します。
vbroker.se.iiop_tp.scm.iiop_tp.dispatcher.threadMaxIdle	300	アイドルなスレッドをデストラクトするまでのタイムアウトを秒数で指定します。0 を指定した場合は，スレッドのアイドル時間が無限になり，スレッドがデストラクトされなくなります。

18.16　双方向通信をサポートするプロパティ

双方向通信をサポートしているプロパティを次の表に示します。

注

ここで示すプロパティが評価されるのは，SCM が生成されるときの一度だけです。SCM の exportBiDir プロパティと importBiDir プロパティには，enableBiDir プロパティで優先順位が設定されます。つまり，これらのプロパティに矛盾する値が設定された場合は，SCM 固有のプロパティが有効になります。これによって，enableBiDir プロパティをグローバルに設定し，各 SCM で双方向性を無効にできます。

表 18–15　双方向通信をサポートするプロパティ（Java）

プロパティ	デフォルト	説明
vbroker.orb.enableBiDir	none	選択的に双方向コネクションを確立できます。クライアントに vbroker.orb.enableBiDir=client を設定し，サーバに vbroker.orb.enableBiDir=server を設定した場合は，ゲートキーパーの vbroker.orb.enableBiDir の値でコネクションの状態が決まります。このプロパティの値は，server，client，both，または none です。
vbroker.se.<se>.scm.<scm>.manager.exportBiDir	ORB は設定しません。	クライアント側プロパティを指定します。true を設定すると，指定したサーバエンジンの双方向コールバック POA を生成できるようになります。false を設定すると，指定したサーバエンジンの双方向コールバック POA を生成できなくなります。
vbroker.se.<se>.scm.<scm>.manager.importBiDir	ORB は設定しません。	サーバ側プロパティを指定します。true を設定すると，リクエストをクライアントに送信するために，クライアントによってすでに確立されたコネクションをサーバ側で再使用できます。false を設定した場合は，再使用できません。

19 プログラマツール（C++）

この章では，Borland Enterprise Server VisiBroker が提供する C++言語用のプログラマツールについて説明します。

19.1 引数とオプション

引数とオプションには，すべての VisiBroker プログラマツールに共通するものと，各ツールに固有のものがあります。ツール固有の引数とオプションについては，ツールごとに説明します。

19.1.1 一般オプション

次に示すオプションはすべてのプログラマツールに共通のオプションです。

オプション

- -J<*java_option*>
  java_option を直接 Java VM に渡します。
- -VBJversion
  Borland Enterprise Server VisiBroker のバージョンを出力します。
- -VBJdebug
  Borland Enterprise Server VisiBroker のデバッグ情報を出力します。
- -VBJclasspath
  CLASSPATH 環境変数の前にクラスパスを指定します。
- -VBJprop <*name*> [=<*value*>]
  対になった名前と値を Java VM に渡します。
- -VBJjavavm <*jvmpath*>
  Java VM へのパスを指定します。
- -VBJaddJar <*jarfile*>
  Java VM を実行する前に CLASSPATH に jarfile を付けます。

19.1.2 プログラマツールの動作環境

この章で説明している Borland Enterprise Server VisiBroker のプログラマツールは，動作環境が UNIX または Windows のどちらであるかによって異なります。

(1) UNIX の場合

UNIX ユーザは，次の構文でコマンドのオプションを表示できます。

```
command name -¥?
```

次のように入力します。

例： idl2cpp -¥?

(2) Windows の場合

Windows ユーザは，次の構文でコマンドのオプションを表示できます。

```
command name -?
```

次のように入力します。

例： `idl2cpp -?`

19.2 idl2cpp

このコマンドは，VisiBroker の IDL（インタフェース定義言語）から idl2cpp コンパイラを実装します。IDL から C++へのコンパイラを使って，IDL ファイルからクライアントスタブおよびサーバスケルトンコードを生成します。

19.2.1 構文

```
idl2cpp [arguments] file1 [file2]...
```

例

```
idl2cpp -hdr_suffix hx -server_ext _serv -no_tie
-no_excep-spec bank.idl
```

19.2.2 説明

idl2cpp コマンドは，IDL ファイルを入力として使用し，クライアントおよびサーバ側のための C++コード，クライアントスタブ，およびサーバスケルトンコードを生成します。

infile パラメタは，C++コードの生成の対象となる IDL ファイルを表します。argument は生成されたコードに対して機能を付加します。

(Windows)

idl2cpp コマンドが生成するスタブおよびスケルトンに基づくインプリメンテーションをリンクする場合，-DSTRICT プリプロセサオプションを使用してください。これを使用しないと，リンカは，orb.lib にコンストラクタがないことを示すエラーメッセージを表示する場合があります。

引数

- `-C, -retain_comments`

  C++コード生成時に，IDL ファイルからコメントを引き継ぎます。このオプションを指定しないと，コメントは C++コードに表示されません。デフォルトは off です。

- `-D, -define` *`foo`*`[=`*`bar`*`]`

  foo にプリプロセサマクロを定義します。bar で値を指定できます。プリプロセサマクロを複数指定するには，`-Dfoo=bar -Dhello=world` のように-D オプションを複数回使用してください。

- `-H, -list_includes`

  インクルードファイルへのパスを標準出力に出力します。デフォルトは off です。

- `-I, -include <dir>`

  #include 検索用の拡張ディレクトリを指定します。#include 検索用の拡張ディレクトリを複数指定するには，`-I/home/include -I/app/include` のように-I オプションを複数回使用してください。

- -P, -no_line_directives

  行番号情報の生成を抑止します。デフォルトは off です。

- -U, -undefine *foo*

  foo に指定されたプリプロセサマクロの定義を解除します。

- -client_ext <file_*extension*>

  生成されたクライアントファイルに対して使用するファイル拡張子を指定します。デフォルトの拡張子は_c です。拡張子なしのクライアントファイルを生成するには，<file_extension>の値に none を指定してください。

- -[no_]back_compat_mapping

  現バージョンでは，この引数は無効です。以降のバージョンで変更される可能性もあります。

- -[no_]comments

  生成されたコードにコメントを入れることを指定します。デフォルトでは，コメントは生成されたコード中に表示されます。

- -[no_]idl_strict

  IDL ソースに，OMG 標準規格を指定します。デフォルトでは OMG 標準規格は使いません。

- -[no_]obj_wrapper

  オブジェクトラッパーサポートでスタブとスケルトンを生成します。また，ほかのすべてのオブジェクトラッパーが継承するベースのタイプドオブジェクトラッパーと，アンタイプドオブジェクトラッパー呼び出しを実行するデフォルトのオブジェクトラッパーを生成します。このオプションが設定されないと，idl2cpp は，オブジェクトラッパー用のコードを生成しません。

- -[no_]preprocess

  解析前に IDL ファイルの前処理をします。デフォルトは on です。

- -[no_]preprocess_only

  前処理の終了後に，IDL ファイルの解析を中止します。このオプションを使用すると，コンパイラで前処理フェーズの結果を stdout に生成できます。デフォルトは off です。

- -[no_]pretty_print

  _pretty_print メソッドを生成します。デフォルトは on です。

- -[no_]servant

  サーバ側コードを生成します。デフォルトでは，サーバントが生成されます。

- -[no_]stdstream

  クラスのシグニチャ内に，標準の iostream クラスを使ったストリーム演算子を生成します。デフォルトは on です。

- -[no_]tie

  _tie テンプレートクラスを生成します。デフォルトでは，_tie クラスが生成されます。

- -[no_]warn_all

  警告をすべて抑止します。デフォルトは off です。

- -[no_]warn_missing_define

  あらかじめ宣言された名前を定義しなかった場合に警告します。デフォルトは on です。

- -[no_]warn_unrecognized_pragmas

  #pragma が認識されない場合に警告を生成します。

- -corba_inc <filename>

  生成されたコードに，通常の#include <corba.h>ではなく#include <filename>が挿入されます。デフォルトでは，#include <corba.h>が生成コードに挿入されます。

- -excep_spec

  メソッドの例外仕様を生成します。デフォルトでは，例外仕様は生成されません。

- -export <tag> (Windows)

  生成されたすべてのクライアント側宣言（クラス，関数など）に挿入するタグ名を定義します。idl2cpp の起動時に，-export _MY_TAG と指定すると，クラス定義が「class Bank {...}」の代わりに「class _MY_TAG Bank {...}」となります。

  デフォルトでは，クライアント側宣言のタグ名は生成されません。

- -export_skel <tag> (Windows)

  生成されたサーバ側の宣言だけに挿入するタグ名を定義します。idl2cpp の起動時に-export_skel _MY_TAG と指定すると，クラス定義が「class POA_Bank {...}」の代わりに「class _MY_TAG POA_Bank {...}」となります。

  デフォルトでは，サーバ側宣言のタグ名は生成されません。

- -gen_included_files

  #include ファイルのコードの生成を指定します。デフォルトではこのコードは生成されません。

- -h, -help, -usage, -?

  ヘルプ情報を出力します。

- -hdr_suffix <*string*>

  ヘッダファイル名の拡張子を指定します。デフォルトの拡張子は.hh です。

- -impl_inherit

  インプリメンテーション継承を生成します。デフォルトは off です。

- -list_files

  コード生成時に書き込まれたファイルのリスト表示を指定します。デフォルトではこのリストは生成されません。

- -map_keyword <keywrd> <map>

  キーワードとして<keywrd>を追加し，表示されたマッピングに対応させます。<keywrd>と矛盾する IDL 識別子は，C++で<map>にマッピングされます。これによって，C++コードで使用されるキーワードと名前間で矛盾が発生しないようにします。

  すべての C++キーワードにはデフォルトのマッピングがあります。このようなキーワードは，このオプションを使用して指定する必要はありません。

- `-namespace`

  モジュールをネームスペースとして実装します。デフォルトは off です。

- `-root_dir <path>`

  生成コードが書き込まれるディレクトリを指定します。デフォルトでは，コードはカレントディレクトリに書き込まれます。

- `-server_ext <file_extension>`

  生成されたサーバファイルに対して使用するファイル拡張子を指定します。デフォルトの拡張子は _s です。拡張子なしのサーバファイルを生成するには，<file_extension>の値に none を指定してください。

- `-src_suffix <string>`

  ソースファイルのファイル拡張子を指定します。デフォルトの拡張子は.cc です。

- `-type_code_info`

  動的起動インタフェースを使用するクライアントプログラムで必要とされるタイプコード情報を生成できるようになります。詳細については，「23. 動的インタフェースとクラス（C++）」を参照してください。デフォルトでは，タイプコード情報は生成されません。

- `-version`

  Borland Enterprise Server VisiBroker のバージョンを表示します。

- `file1 [file2] ...`

  処理対象のファイルを一つ以上指定します。または，標準入力として「-」を指定します。

19.3 idl2ir

このコマンドは，IDL ソースファイルで定義されたオブジェクトにインタフェースリポジトリを実装します。

19.3.1 構文

```
idl2ir [options] file1 [file2]...
```

例

```
idl2ir -irep my_repository -replace bank/Bank.idl
```

19.3.2 説明

idl2ir コマンドは IDL ファイル名を入力として使用します。また，idl2ir コマンドは，このコマンド自体をインタフェースリポジトリサーバにバインドし，<filename>.idl に指定された IDL 構成体にリポジトリを実装します。-replace オプションを指定した場合，リポジトリに IDL ファイルの項目と同じ名前を持つ項目がすでにあると，その項目は新しい項目と置き換えられます。

注

idl2ir コマンドが匿名の配列およびシーケンスを適切に処理できないという問題があります。この問題の対処として，すべての配列およびシーケンスに対して typedefs を使用します。

オプション

- `-D, -define` *foo*`[=`*bar*`]`
  foo に指定されたプリプロセサマクロを定義します。bar で値を指定できます。
- `-I, -include <`*dir*`>`
  #include ファイルを検索するディレクトリを指定します。
- `-P, -no_line_directives`
  行番号情報の生成を抑止します。デフォルトは off で，行の番号づけは抑止されません。
- `-H, -list_includes`
  インクルードされたファイル名を標準出力に出力します。デフォルトは off です。
- `-C, -retain_comments`
  C++コード生成時に，IDL ファイルからコメントを引き継ぎます。このオプションを指定しないと，コメントは C++コードに表示されません。
- `-U, -undefine` *foo*
  foo に指定されたプリプロセサマクロの定義を解除します。
- `-[no_]back_compat_`mapping

VisiBroker 3.x との下位互換性を持つマッピングの使用を指定します。

- -[no_]idl_strict
  IDL ソースに，OMG 標準規格を指定します。デフォルトでは OMG 標準規格は使いません。
- -[no_]preprocess
  解析前に IDL ファイルの前処理をします。デフォルトは on です。
- -[no_]preprocess_only
  前処理の終了後に，IDL ファイルの解析を中止します。このオプションを使用すると，コンパイラで前処理フェーズの結果を stdout に生成できます。デフォルトは on です。
- -[no_]warn_all
  警告をすべて抑止します。デフォルトは off です。
- -[no_]warn_unrecognized_pragmas
  #pragma が認識されない場合に警告を生成します。
- -deep
  シャロウマージではなく，ディープマージを指定します。-deep を指定した場合，新しい内容と既存内容との違いだけをマージします。-deep を指定しない場合はシャロウマージが選択されて，新しい内容が同じ名前を定義すると，すべての既存内容を新しい内容と置き換えます。デフォルトは off です。
- -h, -help, -usage, -?
  ヘルプ情報を出力します。
- -irep <*name*>
  idl2ir がバインドしようとするインタフェースリポジトリのインスタンス名を指定します。名前を指定しないと，idl2ir は，カレントドメインで見つけたインタフェースリポジトリサーバにバインドします。現在のドメインは，OSAGENT_PORT 環境変数によって定義されます。
- -replace
  定義を更新しないで，そのまま置き換えます。
- -version
  Borland Enterprise Server VisiBroker のバージョンを表示します。
- file1 [file2] ...
  処理対象のファイルを一つ以上指定します。処理対象のファイルに stdin を指定するときは，「-」を指定します。

19.4 ir2idl

このコマンドは，インタフェースリポジトリから取得したオブジェクトで IDL ソースファイルを生成します。

19.4.1 構文

```
ir2idl [options]
```

例

foo というインタフェースリポジトリの内容から，foo.idl を生成します。

```
ir2idl -irep foo -o foo.idl
```

19.4.2 説明

ir2idl コマンドは，IR の内容を抽出し，それを IDL として出力します。

オプション

- -irep <*irep name*>
  インタフェースリポジトリの名前を指定します。
- -o <*file*>
  出力ファイルの名前を指定します。または，標準出力として「-」を指定します。
- -strict
  OMG 標準規格を適用した IDL ソースファイルの生成を行うことを指定します。デフォルトは off です。
- -version
  使用中の Borland Enterprise Server VisiBroker のバージョンを表示，または出力します。
- -h, -help, -usage, -?
  ヘルプ情報を出力します。

20 IDL から C++言語へのマッピング

この章では，CORBA C++言語マッピング仕様を実装した Borland Enterprise Server VisiBroker idl2cpp コンパイラによって供給される IDL から C++言語へのマッピングについて説明します。

20.1　基本データ型

次の表に，IDL によって提供されている基本的なデータ型の概要を示します。プラットフォーム間のハードウェアの違いによって，幾つかの IDL 基本データ型がプラットフォームに依存する場合があります。そのようなデータ型には，その定義の説明に"プラットフォームに依存"と示してあります。例えば，64 ビット整数型のプラットフォームでも，CORBA::Long 型は 32 ビットとなってしまいます。プラットフォームに依存する基本データ型の正確なマッピングについては，インクルードファイル"orbtypes.h"，"vdef.h"，または"ptypedef.h"を参照してください。

表 20-1　IDL 基本型マッピング（C++）

IDL 型	VisiBroker 型	C++定義
short	CORBA::Short	short
long	CORBA::Long	プラットフォームに依存
unsigned short	CORBA::UShort	unsigned short
unsigned long	CORBA::ULong	unsigned long
float	CORBA::Float	float
double	CORBA::Double	double
char	CORBA::Char	char
wchar	CORBA::WChar	wchar_t
boolean	CORBA::Boolean	unsigned char
octet	CORBA::Octet	unsigned char
longlong	CORBA::LongLong	プラットフォームに依存
ulonglong	CORBA::ULongLong	プラットフォームに依存

注

IDL boolean 型は，1 または 0 しか使用できないように CORBA の仕様によって定義されています。それ以外の値を使用すると，未定義の動作となります。

20.2　文字列

IDL にある文字列型は，固定長でも可変長でも，C++の char*型(wstring 型は CORBA::WChar*)にマッピングされます。アプリケーションと VisiBroker が，同一のメモリ管理機能を使うようにするために，文字列を動的に割り当てる場合，および動的に解放する場合は，コードサンプル 20-1 に示している関数を使用してください。すべての CORBA 文字列型は，NULL で終わる文字列です。

コードサンプル 20-1　文字列に対するメモリの割り当ておよび解放の際に使用するメソッド

```
class CORBA
{
    ...
    static char *string_alloc(CORBA::ULong len);
    static void string_free(char *data);
    ...
    static CORBA::WChar *wstring_alloc(CORBA::ULong len);
    static void wstring_free(CORBA::WChar *);
    ...
};
```

メソッドの詳細については，「22.4.2　CORBA のメソッド」を参照してください。

20.2.1　String_var クラス

IDL の string を char *にマッピングするとき，IDL コンパイラは，String_var クラスも生成します。これは，文字列用に割り当てられたメモリを指しているポインタを保持するクラスです。String_var オブジェクトがデストラクトされたり，スコープ外になったりすると，文字列用のメモリが自動的に解放されます。コードサンプル 20-2 に，String_var クラスおよびこのクラスがサポートするメソッドを示します。_var クラスの詳細については，「21.6　<class_name>_var」を参照してください。

コードサンプル 20-2　String_var クラス

```
class CORBA {
   class String_var {
      protected:
         char*_p;
         ...
      public:
         String_var();
         String_var(const String_var& var);
         String_var(char *p);
         ~String_var();
         String_var&  operator=(const char *p);
         String_var&  operator=(char *p);
         String_var&  operator=(const String_var& s);
         operator const char *() const;
         operator char *();
         char &operator[ ](CORBA::ULong index);
         char operator[ ](CORBA::ULong index) const;
         friend ostream&  operator<<(
```

```
                            ostream&, const String_var&);
        inline friend Boolean  operator==(
                            const String_var& s1,
                            const String_var& s2);
        ...
    };
    ...
};
```

20.2.2　WString_var クラス

IDL の wstring を CORBA::WChar *にマッピングするとき，IDL コンパイラは，WString_var クラスも生成します。これは，文字列用に割り当てられたメモリを指しているポインタを保持するクラスです。WString_var オブジェクトがデストラクトされたり，スコープ外になったりすると，文字列用のメモリが自動的に解放されます。コードサンプル 20-3 に，WString_var クラスおよびこのクラスがサポートするメソッドを示します。_var クラスの詳細については，「21.6　<class_name>_var」を参照してください。

コードサンプル 20-3　WString_var クラス

```
class CORBA {
   class WString_var {
      private:
         CORBA::WChar *_p;
         ...
      public:
         WString_var();
         WString_var(const WString_var& var);
         WString_var(CORBA::WChar *p);
         WString_var(const CORBA::Whcar *p);
         ~WString_var();

         WString_var&  operator=(const CORBA::WChar *p);
         WString_var&  operator=(CORBA::WChar *p);
         WString_var&  operator=(const WString_var& s);
         operator CORBA::WChar *();
         operator const CORBA::WChar *();
         CORBA::WChar &operator[ ](CORBA::ULong index);
         CORBA::WChar operator[ ](CORBA::ULong index);

         friend ostream&  operator<<(
                            ostream&, const WString_var&);
         friend CORBA::Long compare(
                            const WString_var& s1,
                            const WString_var& s2);
         ...
      };
      ...
};
```

20.3　定数

interface の外で定義された IDL 定数がインクルードファイルで C++定数宣言にダイレクトにマッピングされます。コードサンプル 20-4～20-5 に例を示します。

コードサンプル 20-4　IDL のトップレベル定義

```
const string str_example = ”this is an example”;
const long long_example = 100;
const boolean bool_example = TRUE;
```

コードサンプル 20-5　定数の C++コード

```
const char * str_example = ”this is an example”;
const CORBA::Long long_example = 100;
const CORBA::Boolean bool_example = 1;
```

interface の中で定義された定数がインクルードファイルで宣言され，またソースファイルに値が代入されます。コードサンプル 20-6～20-8 に例を示します。

コードサンプル 20-6　example.idl ファイルの IDL 定義

```
interface example {
    const string str_example = ”this is an example”;
    const long long_example = 100;
    const boolean bool_example = TRUE;
};
```

コードサンプル 20-7　example_client.hh インクルードファイルに生成された C++コード

```
class example :: public virtual CORBA::Object
{
   ...
   static const char *str_example; /* ”this is an example” */
   static const CORBA::Long long_example; /* 100 */
   static const CORBA::Boolean bool_example; /* 1 */
   ...
};
```

コードサンプル 20-8　example_client.cc ソースファイルに生成された C++コード

```
const char *example::str_example = ”this is an example”;
const CORBA::Long example::long_example = 100;
const CORBA::Boolean example::bool_example = 1;
```

20.3.1　定数を含む特別なケース

状況によって IDL コンパイラは，IDL 定数の名前を生成するのではなく，IDL 定数の値を含んでいる C++コードを生成する必要があります。例えば，C++コードが適切にコンパイルできるようにするために，

typedef V に対して定数 length の値が生成される必要があります。例をコードサンプル 20-9～20-10 に示します。

コードサンプル 20-9　値を持つ IDL 定数の定義

```
// IDL
interface foo {
    const long length = 10;
    typedef long V[length];
};
```

コードサンプル 20-10　C++での IDL 定数の値の生成

```
class foo : public virtual CORBA::Object
{
    const CORBA::Long length;
    typedef CORBA::Long V[10];
};
```

20.4　列挙体

IDL 内の列挙体が C++列挙体へとダイレクトにマッピングされています。例をコードサンプル 20-11～20-12 に示します。

コードサンプル 20-11　列挙体の IDL 定義

```
// IDL
enum enum_type {
    first,
    second,
    third
};
```

コードサンプル 20-12　IDL 列挙体の C++enum へのダイレクトマッピング

```
// C++ code
enum enum_type {
    first,
    second,
    third
};
```

20.5　型定義

IDL 内の型定義が C++型定義にダイレクトにマッピングされています。元の IDL 型定義が複数の C++型にマッピングされる場合，IDL コンパイラは，それぞれの型に対応する C++のエイリアスを生成します。例をコードサンプル 20-13～20-14 に示します。

コードサンプル 20-13　IDL の型定義

```
// IDL
typedef octet example_octet;
typedef enum enum_values {
    first,
    second,
    third
} enum_example;
```

コードサンプル 20-14　IDL の型定義の C++へのマッピング

```
// C++
typedef octet example_octet;
enum enum_values {
    first,
    second,
    third
};
typedef enum_values enum_example;
```

その他の型定義のマッピング例を，コードサンプル 20-15～20-18 に示します。

コードサンプル 20-15　インタフェースの IDL typedef

```
// IDL
interface A1;
typedef A1 A2;
```

コードサンプル 20-16　IDL インタフェース型定義の C++へのマッピング

```
// C++
class A1;
typedef A1 *A1_ptr;
typedef A1_ptr A1Ref;
class A1_var;

typedef A1 A2;
typedef A1_ptr A2_ptr;
typedef A1Ref A2Ref;
typedef A1_var A2_var;
```

コードサンプル 20-17　シーケンスの IDL typedef

```
// IDL
typedef sequence<long> S1;
typedef S1 S2;
```

コードサンプル 20-18　IDL シーケンス型定義の C++へのマッピング

```
// C++
class S1;
typedef S1 *S1_ptr;
typedef S1_ptr S1Ref;
class S1_var;

typedef S1 S2;
typedef S1_ptr S2_ptr;
typedef S1Ref S2Ref;
typedef S1_var S2_var;
```

20.6　module 句

OMG の IDL から C++言語へのマッピングでは，IDL の module を C++の namespace にマッピングする際に同じ名前を使用するように指定します。現在，ネームスペースをサポートするコンパイラの数が限られているため，VisiBroker は，module から class へのマッピングだけをサポートしています。コードサンプル 20-19～20-20 では，Borland Enterprise Server VisiBroker の IDL コンパイラが module 定義を class へとマッピングしています。

コードサンプル 20-19　IDL モジュール定義

```
// IDL
module ABC
{
    ...
};
```

コードサンプル 20-20　C++クラスとして生成

```
// C++
class ABC
{
    ...
};
```

20.7　複合データ型

次の複合データ型が，どのように IDL から C++マッピングされるかを説明します。

- Any 型
- string 型（固定長または可変長）
- sequence 型（固定長または可変長）
- オブジェクトリファレンス
- 可変長メンバを含むその他の struct または union
- 可変長要素を持つ array
- 可変長要素を持つ typedef

複合データ型の C++マッピングについて次の表に示します。

表 20–2　複合データ型の C++マッピングの概要（C++）

IDL 型	C++マッピング
struct（固定長）	struct および_var クラス
struct（可変長）	struct および_var クラス（各可変長メンバは，その T_var クラスとともに宣言されます）
union	class および_var クラス
sequence	class および_var クラス
array	array，array_slice，array_forany，および array_var

20.8　構造体

固定長構造体，可変長構造体などについて説明します。

20.8.1　固定長構造体

Borland Enterprise Server VisiBroker の IDL コンパイラは，C++にマッピングされた固定長 IDL 構造体に，構造体と構造体に対する_var クラスを生成します。_var クラスの詳細については，「21.6 <class_name>_var」を参照してください。

コードサンプル 20-21　IDL 内の固定長構造体定義

```
// IDL
struct example {
    short a;
    long b;
};
```

コードサンプル 20-22　固定長 IDL 構造体の C++へのマッピング

```
// C++
struct example {
    CORBA::Short a;
    CORBA::Long b;
};

class example_var
{
    ...
    private:
    example *_ptr;
};
```

(1)　固定長構造体の使用

_var クラスであるフィールドにアクセスするには，->演算子が必要となります。コードサンプル 20-23 は，その例を示しています。ex2 がスコープ外となると，割り当てられていたメモリが自動的に解放されます。

コードサンプル 20-23　example 構造体および example_var クラスの使用

```
// example構造体を宣言し，領域を初期化します。
example ex1 = { 2, 5 };

// _varクラスを宣言し，新しく作成したexample構造体を代入します。
// _varは，未初期化の領域を持つ構造体を指します。
example_var ex2 = new example;

// ex1からex2の領域を初期化します。
ex2->a = ex1.b;
```

20.8.2　可変長構造体

可変長のメンバを持つ構造体か，または固定長のメンバを持つ構造体かによって，C++で生成するコードが異なります。コードサンプル 20-21 に示した example 構造体を可変長にした例を，コードサンプル 20-24～20-25 に示します。ここでは，可変長構造体に変換するため，long メンバが string と入れ替えられ，オブジェクトリファレンスが追加されています。

コードサンプル 20-24　IDL 内の可変長構造体定義

```
// IDL
interface ABC {
    ...
};
struct vexample {
    short a;
    ABC c;
    string name;
};
```

コードサンプル 20-25　可変長構造体の C++へのマッピング

```
// C++
struct vexample {
    CORBA::Short a;
    ABC_var c;
    CORBA::String_var name;
    vexample& operator=(const vexample& s);
};

class vexample_var {
    ...
};
```

ABC オブジェクトが ABC_var クラスにマッピングされています。同様に，name という string が CORBA::String_var クラスにマッピングされます。また，可変長構造体の代入演算子も生成されます。

(1)　可変長構造体でのメモリ管理

可変長構造体で_var クラスを使用すると，可変長メンバに対応するメモリが透過的に管理されます。

- 構造体がスコープ外になると，可変長メンバに対応するすべてのメモリが自動的に解放されます。
- 構造体の初期化または代入が繰り返されると，繰り返し前のデータに対応するメモリは解放されます。
- オブジェクトリファレンスに可変長メンバが割り当てられると，そのオブジェクトリファレンスのコピーが作成されます。ポインタに可変長メンバを代入した場合，そのポインタのコピーは作成されません。

20.8.3 union

IDL に指定した union は，メソッドとともに C++クラスへとマッピングされます。IDL に指定した union の各メンバは，アクセッサ，およびミューテータとして動作する関数にマッピングされます。ミューテータ関数は，データメンバの値を設定します。

あらかじめ定義されている識別型の_d という名前のデータメンバも生成されます。union が初めて作成される際には，この識別型の値は設定されていません。したがって，union を使用する前に，アプリケーションはその識別型の値を設定する必要があります。提供されているメソッドのどれかを使用してデータメンバを設定すると，それらすべてのデータメンバは自動的に識別型として設定されます。次の表は，example_union クラスで使用できるメソッドの幾つかを示しています。

表 20-3　example_union クラス用に生成されたメソッド（C++）

メソッド	説明
_d()	ディスクリミネータの値を返します。
_d(CORBA::Long)	ディスクリミネータの値を設定します。ここでは，ディスクリミネータは long 型です。入力する引数の型は，ディスクリミネータのデータ型によって異なります。
example_union()	デフォルトコンストラクタでは，識別型にデフォルト値が設定されますが，データメンバの初期化は実行されません。
example_union(const example_union& obj)	コピーコンストラクタがソースオブジェクトのディープコピーを実行します。
~example_union()	デストラクタが union によって所有されているメモリを解放します。
operator=(const example_union& obj)	代入演算子がディープコピーを実行します。また，必要な場合は，古いメモリを解放します。

コードサンプル 20-26～20-27 では，example_union の生成方法を示しています。

コードサンプル 20-26　struct を含む IDL union

```
// IDL
struct example_struct
{
   long abc;
};
union example_union switch(long)
{
   case 1: long x; // a primitive data type
   case 2: string y; // a simple data type
   case 3: example_struct z; // a complex data type
};
```

コードサンプル 20-27　IDL union の C++クラスへのマッピング

```
// C++
struct example_struct
{
         CORBA::Long abc;
```

```
};
class example_union
{
   private:
         CORBA::Long _disc;
         CORBA::Long _x;
         CORBA::String_var _y;
         example_struct _z;
   public:
         example_union();
         ~example_union();
         example_union(const example_union& obj);
         example_union& operator=(const example_union& obj);
         void x(const CORBA::Long val);
         const CORBA::Long x() const;
         void y(char *val);
         void y(const char *val);
         void y(const CORBA::String_var& val);
         const char *y() const;
         void z(const example_struct& val);
         const example_struct& z() const;
         example_struct& z();
         CORBA::Long _d();
         void _d(CORBA::Long);
         ...
};
```

(1) union の管理型

コードサンプル 20-27 に示した example_union クラスに加えて，example_union_var クラスも生成されます。_var の詳細については，「21.6 <class_name>_var」を参照してください。

(2) union でのメモリ管理

union 内の複合データ型のメモリ管理をするときは，次の点を注意してください。

- データメンバの値を設定する際にアクセッサメソッドを使用する場合，ディープコピーが実行されます。小さめの型の場合は値を，大きめの型の場合は定数リファレンスを使って，パラメタをアクセッサメソッドに渡してください。
- アクセッサメソッドを使用してデータメンバを設定する場合，そのメンバに対応していたメモリが解放されます。代入されているメンバがオブジェクトリファレンスである場合，そのオブジェクトのリファレンスカウントは，アクセッサメソッドが返される前に増やされます。
- char *アクセッサメソッドは，渡されたポインタの所有権が引き受けられる前に，すべてのメモリを解放します。
- const char *および String_var アクセッサメソッドは，新しいパラメタのメモリがコピーされる前に，すべての古いメモリを解放します。
- 配列データメンバのアクセッサメソッドは，配列スライスを指すポインタを返します。詳細については，「20.8.5(1) 配列スライス」を参照してください。

20.8.4 Sequence

固定長および可変長に関係なく IDL に指定した Sequence は，現在の長さおよび最大長を持つ C++クラスにマッピングされます。固定長シーケンスの最大長は，そのシーケンスの型によって定義されます。C++コンストラクタが呼び出される際に可変長シーケンスの最大長を指定できます。プログラミングによって，現在の長さを変更できます。コードサンプル 20-28～20-29 では，IDL シーケンスがアクセッサメソッドを含む C++クラスへとマッピングされています。

注

可変長シーケンスの長さが，指定した最大長よりも長い場合，Borland Enterprise Server VisiBroker は，バッファが大きい方を透過的に割り当てます。その際，古いバッファが新しいバッファへとコピーされ，その古いバッファに割り当てられていたメモリが解放されます。しかし，最大長が減ると，使用されなかったメモリは解放されません。

コードサンプル 20-28　IDL 可変長シーケンス

```
// IDL
typedef sequence<long> LongSeq;
```

コードサンプル 20-29　IDL 可変長シーケンスの C++クラスへのマッピング

```
// C++
class LongSeq
{
    public:
        LongSeq(CORBA::ULong max=0);
        LongSeq(CORBA::ULong max=0, CORBA::ULong length,
                CORBA::Long *data, CORBA::Boolean release = 0);
        LongSeq(const LongSeq&);
        ~LongSeq();
        LongSeq&    operator=(const LongSeq&);
        CORBA::ULong    maximum() const;
        void   length(CORBA::ULong len);
        CORBA::ULong    length() const;
        const CORBA::ULong&    operator[ ](
                                   CORBA::ULong index) const;
        ...
        static LongSeq    *_duplicate(LongSeq* ptr);
        static void    _release(LongSeq *ptr);
        static CORBA::Long    *allocbuf(CORBA::ULong nelems);
        static void    freebuf(CORBA::Long *data);
    private:
        CORBA::Long* _contents;
        CORBA::ULong   _count;
        CORBA::ULong   _num_allocated;
        CORBA::Boolean   _release_flag;
        CORBA::Long   _ref_count;
};
```

コードサンプル 20-29 に示した可変長シーケンスのために生成されたメソッド一覧を次の表に示します。

表 20-4　コードサンプル 20-29 に示した可変長シーケンスのために生成されたメソッドの一覧 (C++)

メソッド	説明
LongSeq(CORBA::ULong max=0)	可変長シーケンスのコンストラクタは，最大長を引数として使用します。固定長シーケンスには定義された最大長が含まれます。
LongSeq(CORBA::Ulong max=0, CORBA::ULong length, CORBA::Long *data, CORBA::Boolean release=0)	コンストラクタで，最大長，現在の長さ，対応するデータバッファのポインタ，解放フラグを設定できます。release が 0 でない場合にシーケンスのサイズを拡大する際，データバッファに対応するメモリを解放します。release が 0 である場合，古いデータバッファのメモリは解放されません。固定長シーケンスでは，max 以外のこれらすべてのパラメタを使用します。
LongSeq(const LongSeq&)	コピーコンストラクタは，ソースオブジェクトのディープコピーを実行します。
~LongSeq();	解放フラグが構成された際に 0 以外の値を持つ場合だけ，デストラクタがシーケンスによって所有されるメモリを解放します。
operator=(const LongSeq&)	代入演算子がディープコピーを実行します。必要な場合，古いメモリを解放します。
maximum()	シーケンスのサイズを返します。
length()	シーケンスの長さを設定したり，返したりするために，二つのメソッドが定義されます。
operator[ ] ()	シーケンス内の要素にアクセスするために二つのインデックス演算子が提供されています。一方はその要素を修正するための演算子で，もう一方は要素を読み込むためだけ（読み取り専用）の演算子です。
_release()	シーケンスを解放します。オブジェクトが生成され，シーケンス要素型が文字列およびオブジェクトリファレンスである場合で，コンストラクタの解放フラグが 0 以外の値であるときは，バッファが解放される前に各要素が解放されます。
allocbuf() freebuf()	シーケンス用のメモリを割り当てたり，解放したりするために，これら二つの静的メソッドを使用します。

(1)　シーケンスの管理型

コードサンプル 20-29 に示した LongSeq クラスに加えて，LongSeq_var クラスも生成されます。_var の詳細については，「21.6　<class_name>_var」を参照してください。また，通常の_var メソッドに加えて，シーケンスのインデックスメソッドが二つ定義されています。

コードサンプル 20-30　シーケンスを表す_var クラスに追加された二つのインデックスメソッド

```
CORBA::Long& operator[ ](CORBA::ULong index);
const CORBA::Long& operator[ ](CORBA::ULong index) const;
```

(2) シーケンスでのメモリ管理

メモリ管理をする際には，次の事項に注意してください。コードサンプル 20-31～20-32 は，次の事項を配慮しています。

- シーケンスが生成された際にその解放フラグが 0 以外の値で設定された場合，そのシーケンスがユーザのメモリの管理を引き受けます。要素が代入された際，右側にあるメモリの所有権が引き受けられる前に，古いメモリが解放されます。
- 文字列またはオブジェクトリファレンスを含むシーケンスが生成された際に，その解放フラグが 0 以外の値で設定された場合，シーケンスのコンテンツバッファが解放される前，およびそのオブジェクトがデストラクトされる前に，各要素が解放されます。
- 解放フラグに 1 が設定されていないかぎり，**[ ]**演算子を使ってシーケンス要素を代入しないでください。代入した場合，メモリ管理のエラーが発生するおそれがあります。
- 解放フラグに 0 を設定してシーケンスを生成した場合，そのシーケンスを inout パラメタとして使用しないでください。オブジェクトサーバでメモリ管理のエラーが発生するおそれがあります。
- シーケンスで使用するメモリを作成および解放する場合は，必ず allocbuf および freebuf を使用してください。

コードサンプル 20-31　固定長シーケンスの IDL 仕様

```
// IDL
typedef sequence<string, 3> StringSeq;
```

コードサンプル 20-32　二つの固定長シーケンスでのメモリ管理の例

```
// C++
char *static_array[ ] = {(char*)"1", (char*)"2", (char*)"3"};
char **dynamic_array = StringSeq::allocbuf(3);
dynamic_array[0] = CORBA::string_dup("1");
dynamic_array[1] = CORBA::string_dup("2");
dynamic_array[2] = CORBA::string_dup("3");

// シーケンスを作成し，解放フラグはデフォルトのFALSEを設定します。
StringSeq static_seq(3, static_array);

// 別のシーケンスを作成し，解放フラグはTRUEを設定します。
StringSeq dynamic_seq(3, dynamic_array, 1);

static_seq[1] = "1"; // 古い領域は，解放されないでコピーされます。

char *str = CORBA::string_dup("1");
dynamic_seq[1] = str; // 古い領域は，解放され代入されます。
```

20.8.5　配列

IDL で指定した配列は，C++配列にマッピングされます。マッピングされた配列を静的に初期化できます。IDL 配列要素が文字列またはオブジェクトリファレンスである場合，C++配列ではそれらの要素が _var 型となります。コードサンプル 20-33～20-34 は，異なる要素型を持つ三つの配列を示しています。

コードサンプル 20-33　IDL 配列定義

```
// IDL
interface Intf
{
    ...
};
typedef long L[10];
typedef string S[10];
typedef Intf A[10];
```

コードサンプル 20-34　IDL 配列の C++配列へのマッピング

```
// C++
typedef CORBA::Long L[10];
typedef CORBA::String_var S[10];
typedef Intf_var A[10];
```

文字列およびオブジェクトリファレンスに対して_var という管理型を使うことによって，配列要素が代入された際にメモリが透過的に管理できます。

(1)　配列スライス

array_slice 型は複次元配列のパラメタを渡す際に使用されます。一次元配列以外の配列に対して_slice 型が VisiBroker の IDL コンパイラによって生成されます。パラメタを渡したり，返したりする際に，_slice 型を使うと便利です。コードサンプル 20-35 とコードサンプル 20-36 は，_slice 型の例を示しています。

コードサンプル 20-35　複次元配列の IDL 定義

```
// IDL
typedef long L[10];
typedef string str[1][2][3];
```

コードサンプル 20-36　_slice 型の生成

```
// C++
typedef CORBA::Long L_slice;
typedef CORBA::String_var str_slice[2][3];
```

(2)　配列の管理型

VisiBroker の IDL コンパイラは，IDL 配列に対する C++配列だけでなく，_var クラスも生成します。このクラスは，配列の拡張機能を提供します。

- 配列要素への直感的なアクセスを可能にするため operator[ ]がオーバーロードされます。

- 配列の_slice オブジェクトに対するポインタを引数として使用するために，コンストラクタと代入演算子が提供されます。

コードサンプル 20-37　配列の IDL 定義

```
// IDL
typedef long L[10];
```

コードサンプル 20-38　配列に対して生成された_var クラス

```
// C++
class L_var
{
    public:
        L_var();
        L_var(L_slice *slice);
        L_var(const L_var& var);
        ~L_var();
        L_var&  operator=(L_slice *slice);
        L_var&  operator=(const L_var& var);
        CORBA::Long&  operator[ ](CORBA::ULong index);
        operator L_slice *();
        operator L &() const;
        ...
    private:
        L_slice*_ptr;
};
```

(3)　タイプセーフ配列

any 型にマッピングされた要素を持つ配列を処理するために，_forany という特別なクラスが生成されます。_var クラスと同様に，_forany クラスでも下位の配列型にアクセスできます。Any 型がメモリの所有権を引き受けるので，_forany クラスは，デストラクトの際にメモリを解放しません。関数を適切にオーバーロードし，ほかの型と見分けられるようにするため，_forany クラスは typedef として実装されません。

コードサンプル 20-39　IDL 配列定義

```
// IDL
typedef long L[10];
```

コードサンプル 20-40　IDL 配列に対して生成された_forany クラス

```
// C++
class L_forany
{
    public:
        L_forany();
        L_forany(L_slice *slice);
        ~L_forany();
        CORBA::Long&  operator[ ](CORBA::ULong index);
        const CORBA::Long&  operator[ ](
                                CORBA::ULong index) const;
        operator L_slice *();
        operator L &() const;
```

```
        operator const L &() const;
        operator const L&() const;
        L_forany& operator=(const L_forany obj);
        ...
    private:
        L_slice*_ptr;
};
```

(4) 配列でのメモリ管理

配列に対応するメモリの割り当て，および解放のため，VisiBroker の IDL コンパイラは四つの関数を生成します。これらの関数を使うと，VisiBroker ORB は new および delete 演算子をオーバーライドしなくてもメモリを管理できます。

コードサンプル 20-41　IDL 配列定義

```
// IDL
typedef long L[10];
```

コードサンプル 20-42　配列メモリの割り当ておよび解放のため生成されたメソッド

```
// C++
static inline L_slice* L_alloc(); // 配列を動的に割り当てます。
static inline void L_free(L_slice *data);
                                  // L_allocを使って割り当てられた
                                  // 配列のメモリを解放します。
static inline void L_copy(L_slice *_to, L_slice *_from)
                // _from配列を_to配列にコピーします。
static inline L_slice *L_dup(const L_slice *_date)
                // 新しくコピーされた_date配列を返します。
```

20.8.6 Principal

Principal は，オブジェクトインプリメンテーションに対して，オペレーション要求を発行しているクライアントアプリケーションについての情報を表します。Principal の IDL インタフェースにオペレーションはありません。Principal は，octet のシーケンスとして実装されます。Principal は，クライアントアプリケーションによって設定され，VisiBroker ORB インプリメンテーションによってチェックされます。Borland Enterprise Server VisiBroker では，Principal は不透過な型として取り扱われ，Principal の内容は VisiBroker ORB には，検査されません。

20.9 Valuetype

IDL の valuetype は，IDL の valuetype と同じ名前の C++クラスにマッピングされます。このクラスは，valuetype の状態メンバに対応する純粋仮想アクセッサとモディファイア関数，および valuetype のオペレーションに対応する純粋仮想関数を持つ抽象ベースクラスです。

C++クラス名が valuetype の完全なスコープ名称に"OBV_"を持つ，C++クラスが生成されます。この C++クラスは抽象ベースクラスのアクセッサ，およびモディファイアの，デフォルトインプリメンテーションを提供します。

valuetype インスタンスは，アプリケーションで生成します。生成後，これらのアプリケーションはポインタを使ってインスタンスを処理します。C++ valuetype インスタンスへのハンドルは，C++ポインタです。C++ポインタ，または類似のオブジェクトとして実装される C++ _ptr 型にマッピングされるオブジェクトリファレンスとは異なります。これによって，ハンドルとオブジェクトリファレンスを区別できます。

インタフェースのマッピングとは異なり，valuetype のリファレンスカウント機能は，valuetype のインタフェースごとに実装しなければなりません。valuetype の_var 型はリファレンスカウントを自動化します。コードサンプル 20-43 では，この機能を示します。

コードサンプル 20-43　リファレンスカウントを自動化する valuetype の_var 型

```
 valuetype Example {
  short op1();
 long op2( in Example x );
 private short val1;
 public long val2;
};
```

また，コードサンプル 20-44 では，IDL 定義に対して生成される三つのクラスの C++マッピングを示します。

コードサンプル 20-44　IDL 定義の C++マッピング

```
class Example : public virtual CORBA::ValueBase {
public:
   virtual CORBA::Short op1() = 0;
   virtual CORBA::Long op2(Example_ptr _x) = 0;
// すべてのパブリックな状態の純粋仮想getter/setterです。
// これらのアクセッサは，C++のunionメンバと同じように，
// リファレンスによってread-write属性でアクセスできます。
virtual void val2(const CORBA::Long _val2) = 0;
virtual const CORBA::Long val2() const = 0;

protected:
Example() {}
virtual ~Example() {}
virtual void val1(const CORBA::Short _val1) = 0;
virtual const CORBA::Short val1() const = 0;

   private:
```

```
void operator=(const Example&);
};
class OBV_Example: public virtual Example{
public:
virtual void val2(const CORBA::Long _val2) {
_obv_val2 = _val2;
}
virtual const CORBA::Long val2() const {
return _obv_val2;
}

protected:
virtual void val1(const CORBA::Short _val1) {
_obv_val1 = _val1;
}
virtual const CORBA::Short val1() const {
return _obv_val1; }
OBV_Example() {}
virtual ~OBV_Example() {}
OBV_Example(const CORBA::Short _val1,
     const CORBA::Long _val2) {
   _obv_val1 = _val1;
   _obv_val2 = _val2;
}
CORBA::Short _obv_val1;
CORBA::Long _obv_val2;
};

class Example_init : public CORBA::ValueFactoryBase {
};
```

_init クラスは，valuetype のファクトリを実装する方法を提供します。valuetype は，回線上を値で渡されます。そのため，ストリームして送信した valuetype の受信側が，ストリームから valuetype インスタンスを生成するために，ファクトリを実装します。ストリーム上で valuetype を受信する可能性がある場合は，サーバとクライアントの両方がこのファクトリを実装している必要があります。コードサンプル 20-45 で示すように，_init クラスは CORBA::ValueBase *を返す create_for_unmarshal も実装してください。コードサンプル 20-46 およびコードサンプル 20-47 では，ほかの valuetype から valuetype を派生させる例を示します。

コードサンプル 20-45　_init クラス例

```
class Example_init_impl: public Example_init{
public:
   Example_init_impl();
   virtual ~Example_init_impl();
CORBA::ValueBase * create_for_unmarshal() {
 ...// return an Example_ptr
 }
};
```

コードサンプル 20-46　ほかの valuetype から派生した valuetype の IDL

```
valuetype DerivedExample: Example{
 short op3();
};
```

コードサンプル 20-47　派生 valuetype に対して生成された C++

```
// IDL valuetype: DerivedExample
class DerivedExample  : public virtual Example {
 public:
   virtual CORBA::Short op3() = 0;
 protected:
   DerivedExample() {}
   virtual ~DerivedExample() {}
 private:
   void operator=(const DerivedExample&);
};
class OBV_DerivedExample: public virtual DerivedExample, public virtual OBV_Example{
 protected:
   OBV_DerivedExample() {}
   virtual ~OBV_DerivedExample() {}
};
class DerivedExample_init : public CORBA::ValueFactoryBase { };
```

コードサンプル 20-48 に，派生した valuetype をベースの valuetype に切り落とす例を示します。これは，ストリームの受信側が，派生 valuetype は構築できないが，ベース valuetype は構築できる場合に必要です。

コードサンプル 20-48　切り落とされた派生 valuetype

```
valuetype DerivedExample : truncatable Example { };
```

マッピングは通常の派生した valuetype と同じですが，Example というベースクラスに切り落とされることを示すため，DerivedExample クラスの Type 情報に情報が追加されるという点が異なります。

valuetype はインタフェースからは派生できませんが，インタフェースのすべてのオペレーションを提供することによって一つ以上のインタフェースをサポートできます。この場合，supports という IDL キーワードを使用できます。

コードサンプル 20-49　派生 valuetype の IDL キーワード「support」

```
interface myInterface{
 long op5();
};

valuetype IderivedExample supports myInterface {
 short op6();
};
```

コードサンプル 20-50　派生 valuetype の C++

```
// IDL valuetype: DerivedExample
class IderivedExample  : public virtual CORBA::ValueBase  {
```

```
 public:
   virtual CORBA::Short op6() = 0;
   virtual CORBA::Long op5() = 0;

 protected:
   IderivedExample() {}
   virtual ~IderivedExample() {}
 private:
   void operator=(const IderivedExample&);
};

class OBV_IderivedExample: public virtual IderivedExample{
 protected:
   OBV_IderivedExample() {}
   virtual ~OBV_IderivedExample() {}
};
```

リファレンスカウント用に，C++ マッピングは二つの標準クラスを提供します。クラス CORBA::DefaultValueRefCountBase は，任意のアプリケーションが提供する，どの IDL インタフェースからも派生しない具象 valuetype のベースクラスとして使用します。この種類の valuetype の場合，アプリケーションは独自のリファレンスカウントメカニズムを実装できます。クラス PortableServer::ValueRefCountBase は，任意のアプリケーションが提供する，一つ以上の IDL インタフェースから派生する具象 valuetype クラスのベースクラスとして使用します。

20.9.1 Valuebox

valuebox は，構造体，union，any，文字列，基本型，オブジェクトリファレンス，enum，シーケンス，および配列型に適用される valuetype です。これらの型はメソッド，継承，およびインタフェースはサポートしません。valuebox はリファレンスカウントされ，CORBA::DefaultValueRefCountBase から派生します。マッピングはベースとなる型によって異なります。すべての valuebox C++クラスはベースとなる型へのマッピング用に_boxed_in()，_boxed_out()，および_boxed_inout()を提供します。valuebox id のファクトリは，生成されたスタブが自動的に登録します。

20.10　抽象インタフェース

抽象インタフェースで，オブジェクトをリファレンス（IOR）で渡すか，または値（valuetype）で渡すかを実行時に決定するには，プリフィクス"abstract"を，インタフェース宣言前に使用します。

コードサンプル 20-51　IDL コードサンプル

```
abstract interface foo {
   void func():
};
```

抽象インタフェースをサポートする valuetype は，その抽象インタフェースとして渡せます。抽象インタフェースはコードサンプル 20-52 に示すように宣言されます。

コードサンプル 20-52　抽象インタフェースとしての valuetype

```
valuetype vt supports foo {
   ...
};
```

抽象インタフェースとして渡す必要があるインタフェースはコードサンプル 20-53 に示すように宣言されます。

コードサンプル 20-53　抽象インタフェースとしてのインタフェース

```
interface intf : foo {
};
```

コードサンプル 20-53 で宣言した foo という抽象インタフェースの C++マッピングの結果は，コードサンプル 20-54 のようなクラスになります。

コードサンプル 20-54　抽象インタフェースの C++マッピング

```
class foo_var : public CORBA::_var{
   ...
};
class foo_out{
   ...
};

class foo : public virtual CORBA:::AbstractBase{
   private:
      ...
      void operator=(const foo&) {}
   protected:
      foo();
      foo(const foo& ref) {}
      virtual ~foo() {}
   public:
      static CORBA::Object* _factory():
      foo_ptr _this();
      static foo_ptr _nil() { ... }
      static foo_ptr _narrow(CORBA::AbstractBase* _obj);
```

```
      static foo_ptr _narrow(CORBA::Object_ptr _obj);
      static foo_ptr _narrow(CORBA::ValueBase_ptr _obj);

      virtual void func() = 0;

      ...

};

class _vis_foo_stub : public virtual foo, public virtual CORBA::Object {
   public :
      _vis_foo_stub() {}
      virtual ~_vis_foo_stub() {}

      ...
      virtual void func():
};
```

_var クラス，_out クラス，およびメソッドを実装する CORBA::AbstractBase から派生したクラスがあります。

21 生成されるインタフェースとクラス（C++）

この章では，Borland Enterprise Server VisiBroker の IDL コンパイラで生成されるクラス，その使い方，および機能について説明します。

21.1 概要

Borland Enterprise Server VisiBroker の IDL コンパイラは，クライアントアプリケーションおよびオブジェクトサーバの開発をより簡単にするために，多様なクラスを生成できます。生成されたクラスのほとんどは，CORBA クラスでも使用できます。

- スタブクラス（<interface_name>）
- オブジェクトラッパークラス（<interface_name>ObjectWrapper）
- サーバントクラス（POA_<class_name>）
- tie クラス（_tie_<class_name>）
- var クラス（<class_name>_var）

21.2 <interface_name>

```
class <interface_name>
```

<interface_name>クラスは，特定の IDL インタフェースのために生成され，クライアントアプリケーションによって使用されます。このクラスは，特定の IDL インタフェースのために定義されたメソッドを提供します。クライアントが，オブジェクトリファレンスを使って，オブジェクトに対してメソッドを呼び出す場合，実際に呼び出されるのはスタブメソッドです。スタブメソッドは，クライアントのオペレーション要求をパッケージ化して，オブジェクトインプリメンテーションに送信し，その結果を反映します。この全プロセスはクライアントアプリケーションにとって透過的です。

クライアントがローカルオブジェクトリファレンスを使用してローカルオブジェクトのメソッドを呼び出す場合，スタブメソッドは使用されません。

注

IDL コンパイラによって生成されたスタブクラスの内容は変更しないでください。

- static <interface_name>_ptr _bind(
  const char *_poa_name,
  const CORBA::OctetSequence& _id,
  const char *_host_name=(const char*)NULL,
  const CORBA::BindOptions* _opt=(CORBA::BindOptions*)NULL,
  CORBA::ORB_ptr _orb=(CORBA::ORB_ptr)NULL);
  - _poa_name
    絶対パスで記述された POA 名
  - _id
    リファレンスを生成するオブジェクトの識別子
  - _host_name
    対象オブジェクトがあるホスト名
  - opt
    省略するか，または NULL だけ指定してください。
  - orb
    サーバが使用する VisiBroker ORB。ORB.init メソッドの呼び出しによって取得されます。
- static <interface_name>_ptr _narrow(
  CORBA::Object * _obj);
  このメソッドは，CORBA::Object リファレンスを，<interface_name>型のオブジェクトにナロウイングします。オブジェクトリファレンスをナロウイングできない場合，CORBA::BAD_PARAM 例外または null が返されます。

- _obj
  <interface_name>型にナロウイングされるオブジェクト

- static <interface_name>_ptr **_bind**(
  const char ***_object_name** = (const char*)NULL,
  const char ***_host_name** = (const char*)NULL,
  const CORBA::BindOptions* **_opt** = (CORBA::BindOptions*)NULL,
  CORBA::ORB_ptr **_orb** = (CORBA::ORB_ptr)NULL);
  このメソッドは，VisiBroker ORB オブジェクトにバインドし，オブジェクトリファレンスを取得します。
  - _object_name
    オブジェクト名を識別する文字列。このパラメタはオプションです。
  - _host_name
    ORB オブジェクトを探すホスト名を識別する文字列。このパラメタはオプションです。
  - _opt
    このオプションは省略するか，または NULL を指定してください。
  - _orb
    サーバが使用する VisiBroker ORB。ORB_init メソッドの呼び出しによって取得されます。

21.3 <interface_name>ObjectWrapper

```
class <interface_name>ObjectWrapper
```

このクラスは，ローカルインタフェースには適用されません。

それ以外のインタフェースでは，このクラスは，タイプドオブジェクトラッパーを派生するために使用されます。idl2cpp コマンドを-obj_wrapper オプション付きで呼び出すと，ローカルインタフェース以外のすべてのインタフェースに対して生成されます。idl2cpp コマンドについては，「19.2 idl2cpp」を参照してください。オブジェクトラッパー機能を使用する際の詳細については，マニュアル「Borland Enterprise Server VisiBroker デベロッパーズガイド」の「オブジェクトラッパーの使用」の記述を参照してください。

- static void **add**(
  CORBA::ORB_ptr **orb**,
  CORBA::WrapperFactory **factory**,
  VISObjectWrapper::Location **loc**);
  クライアントアプリケーションから，タイプドオブジェクトラッパーを追加します。複数のタイプドオブジェクトラッパーがインストールされている場合，登録された順に呼び出されます。
  - orb
    クライアントが使用したい ORB。ORB_init メソッドによって返されます。
  - factory
    追加したいオブジェクトラッパークラスのファクトリメソッド。
  - loc
    追加されるオブジェクトラッパーのロケーション。次の値のどれかです。
    `VISObjectWrapper::Client`
    `VISObjectWrapper::Server`
    `VISObjectWrapper::Both`
- static void **remove**(
  CORBA::ORB_ptr **orb**,
  CORBA::WrapperFactory **factory**,
  VISObjectWrapper::Location **loc**);
  サーバアプリケーションからタイプドオブジェクトラッパーを削除します。
  - orb
    クライアントが使用したい ORB。ORB_init メソッドによって返されます。
  - factory
    削除したいオブジェクトラッパークラスのファクトリメソッド。
  - loc

削除されるオブジェクトラッパーのロケーション。次の値のどれかです。

VISObjectWrapper::Client

VISObjectWrapper::Server

VISObjectWrapper::Both

21.4 POA_<class_name>

```
class POA_<class_name>
```

POA_<class_name>クラスは，IDL コンパイラによって生成される抽象ベースクラスです。この POA クラスを使って，オブジェクトインプリメンテーションクラスを派生させます。オブジェクトインプリメンテーションは通常，クライアントのオペレーション要求の受信，および解釈に必要なメソッドを提供するサーバントクラスから派生します。

21.5 _tie_<class_name>

```
class _tie_<class_name>
```

tie<class_name>クラスは，デリゲーションインプリメンテーションの生成をサポートするために，IDL コンパイラによって生成されます。この tie クラスを使って，ほかのオブジェクトに対するすべてのオペレーション要求をデリゲートする，オブジェクトインプリメンテーションを生成します。したがって，CORBA::Object クラスから継承させない既存のオブジェクトを使用できます。

21.6 <class_name>_var

```
class <class_name>_var
```

<class_name>_var クラスは，IDL インタフェースのために生成されるクラスで，メモリ管理の負担を軽減させるセマンティックを提供します。

22 コアインタフェースとクラス（C++）

この章では，C++言語のコアインタフェースとクラスについて説明します。

22.1 PortableServer::AdapterActivator

```
class PortableServer::AdapterActivator :
                        public CORBA::PseudoObject
```

アダプタアクティベータは POA（ポータブルオブジェクトアダプタ）と対応し，オンデマンドで子 POA を作成する機能を POA に提供します。この機能は，子 POA を指定したリクエストを受信するとき，または活性化パラメタを TRUE に設定して find_POA メソッドを呼び出すときに使用される機能です。

22.1.1 インクルードファイル

このクラスを使用するときは，poa_c.hh ファイルをインクルードしてください。

22.1.2 PortableServer::AdapterActivator のメソッド

- CORBA::Boolean **unknown_adapter**(
  POA_ptr **parent**, const char* **name**);
  存在しない POA のオブジェクトリファレンスリクエストを受信すると，VisiBroker ORB がこのメソッドを呼び出します。作成する必要のある POA ごとに，VisiBroker ORB はこのメソッドを呼び出して作成します。その際，ルート POA に最も近い祖先の POA 下から作成します。
  - parent
    このメソッドを生成したアダプタアクティベータに対応する親 POA
  - name
    作成する POA の名前（親 POA に対応する POA 名）

22.2 PortableServer

```
class PortableServer
```

22.2.1 インクルードファイル

このクラスを使用するときは，poa_c.hh ファイルをインクルードしてください。

22.2.2 PortableServer のメソッド

- char* ObjectId_to_string(
  const ObjectId& oid);
  このメソッドは，指定された ObjectId を文字列に変換します。例えば，文字列に変換された ObjectId をファイルなどに格納できます。
  - oid
    文字列に変換される ObjectId
- ObjectId* string_to_ObjectId(
  const char* str);
  このメソッドは，オブジェクトを表す文字列を ObjectId に変換します。
  - str
    ObjectId を示す文字列

22.3 CompletionStatus

```
enum CORBA::CompletionStatus
```

この列挙体は，オペレーションの完了状態を表します。

22.3.1 CompletionStatus のメンバ

- COMPLETED_YES = 0
  オペレーションが無事完了したことを表します。
- COMPLETED_NO = 1
  例外，またはエラーの発生のため，オペレーションが開始されなかったことを表します。
- COMPLETED_MAYBE = 2
  例外，またはエラーの発生のため，オペレーションが完了しなかった可能性があることを表します。

22.4 CORBA

```
class CORBA
```

CORBA クラスは，ORB を使用する上で基本的なメソッドを提供します。

22.4.1 インクルードファイル

このクラスを使用するときは，corba.h ファイルをインクルードしてください。

22.4.2 CORBA のメソッド

- static CORBA::ORB_ptr **ORB_init**(
  int& **argc**, char *const *__argv__, const char *__orb_id__ = NULL);
  このメソッドは，ORB を初期化し，クライアントおよびオブジェクトインプリメンテーションによって使用されます。また，ORB メソッドを呼び出すために使用できる ORB を指すポインタを返します。アプリケーションのメイン関数に渡される argc および argv パラメタを，直接このメソッドに渡せます。このメソッドによって受け取られる引数は，ほかのコマンドライン引数と見分けられるように名前・値ペアとなっています。指定できる ORB_init オプションの詳細については，「34. コマンドラインオプション（C++）」を参照してください。
  - argc
    渡された引数の数
  - argv
    引数を指す char ポインタの配列。二つの引数を除く，すべての引数がキーワードと値の形式になります。このメソッドは認識できないキーワードを無視します。
  - orb_id
    使用される ORB の型を識別します。デフォルトは ORB です。
- static char *__string_alloc__(
  CORBA::ULong **len**);
  文字列領域を動的に割り当て，その文字列を指すポインタを返します。割り当てが失敗すると，NULL ポインタが返されます。
  - len
    確保する文字列領域の長さを指定します。指定する文字列の長さは，NULL ターミネータを含まなくてもかまいません。
- static void **string_free**(
  char *__str__);

CORBA::string_alloc メソッドを使って割り当てた文字列に対応するメモリを解放します。

- str

  CORBA::string_alloc メソッドを使って割り当てた文字列を指すポインタです。

- static CORBA::WChar *wstring_alloc(

  CORBA::ULong len);

  ワイド文字列領域を動的に割り当て，その文字列を指すポインタを返します。割り当てに失敗すると，NULL ポインタが返されます。

  - len

    確保するワイド文字列領域の長さを指定します。指定するワイド文字列の長さは，NULL ターミネータを含まなくてもかまいません。

- static void wstring_free(

  CORBA::WChar *str);

  CORBA::wstring_alloc メソッドを使って割り当てた文字列に対応するメモリを解放します。

  - str

    CORBA::wstring_alloc メソッドを使って割り当てた文字列を指すポインタです。

22.5 Context

```
class CORBA::Context
```

Context クラスは，静的，または動的なメソッドの呼び出しの際に暗黙的なパラメタとしてサーバに渡され，クライアントアプリケーションの環境についての情報を表します。このクラスは，リクエストと対応づける必要がある特別な情報の伝達に使用できますが，このクラスはメソッドの引数の一部ではありません。

Context クラスは，名前・値のペアとして格納されているプロパティのリストによって構成されています。このクラスは，それらのプロパティの設定，および操作に必要なメソッドを提供します。Context は，NVList オブジェクトを含んでいて，名前・値ペア群を保持します。

また，Context_var クラスも使用できます。このクラスは，メモリ管理を容易にするセマンティックを提供します。

「22.9.2 CORBA::ORB のメソッド」の「void get_default_context(CORBA::Context_out contextPtr);」も参照してください。

22.5.1 インクルードファイル

このクラスを使用するときは，corba.h ファイルをインクルードしてください。

22.5.2 Context のメソッド

- const char *context_name() const;
  このメソッドは，コンテキストを示す名前を返します。このオブジェクトが生成された際に名前が提供されないと，NULL 値が返されます。
- void create_child(
  const char * name, CORBA::Context_out context);
  このメソッドは，このオブジェクトの子 Context を生成します。
  - name
    新しい Context オブジェクトの名前
  - context
    新しく生成された子 Context に対するリファレンス
- void delete_values(
  const char *name);
  このメソッドは，オブジェクトからプロパティを削除します。
  - name

削除される一つ以上のプロパティの名前。該当するプロパティを削除するために，名前にワイルドカード"*"（アスタリスク）を含めることができます。すべてのプロパティを削除するには，"*"を一つ指定します。

- static CORBA::Context_ptr _duplicate(
  CORBA::Context_ptr ctx);
  このメソッドは，指定されたオブジェクトを複製します。
  - ctx
    複製されるオブジェクト
- void get_values(
  const char *start_scope,
  CORBA::Flags flag,
  const char *name,
  CORBA::NVList_out NVList_ptr);
  このメソッドは，Context オブジェクト階層を検索し，name パラメタで指定された一つ以上の名前・値ペアを取り出します。次に，NVList オブジェクトを生成し，検索した名前・値ペアを配置し，そのオブジェクトに対するリファレンスを返します。
  start_scope パラメタで，検索が開始される最初のコンテキストの名前を指定します。プロパティが見つからない場合，Context オブジェクト階層での検索は一致するプロパティが見つかるまで，または検索対象となる Context オブジェクトがなくなるまで続きます。
  該当する Context が見つからない場合，このメソッドは BAD_CONTEXT 例外を発生させます。また，このメソッドは，name に空の文字列を指定すると BAD_PARAM 例外を発生させます。
  メモリの確保に失敗した場合は，NO_RESOURCES 例外を発生させます。
  - start_scope
    検索が開始される最初の Context オブジェクトの名前。CORBA::Context::_nil()と設定した場合，カレント Context から検索が開始されます。
  - flag
    検索スコープをカレント Context に限定するには，CORBA::CTX_RESTRICT_SCOPE と指定します。
    限定しない場合は，CORBA::CTX_RESTRICT_SCOPE(15)以外の値を指定してください。
  - name
    検索の対象となるプロパティ名。ワイルドカード"*"を使って，name と一致するすべてのプロパティを取り出せます。一致するコンテキスト名が発見できない場合，例外が発生します。
  - NVList_ptr
    見つかったプロパティのリストに対するリファレンス
- static CORBA::Context_ptr _nil();
  このメソッドは，初期化の際に使用する NULL Context_ptr を返します。
- CORBA::Context_ptr parent();

このパラメタは，親 Context を指すポインタを返します。親 Context がない場合，NULL 値が返されます。

- static void _release(
  CORBA::Context_ptr ctx);
  この静的メソッドは，指定された Context オブジェクトを解放します。オブジェクトのリファレンスカウントが 0 になると，そのオブジェクトは自動的に削除されます。
  - ctx
    解放されるオブジェクト
- void set_one_value(
  const char *name, const CORBA::Any& anAny);
  このメソッドは，指定された名前と値を使って，このオブジェクトにプロパティを追加します。
  - name
    プロパティの名前
  - anAny
    プロパティの値
- void set_values(
  CORBA::NVList_ptr _list);
  このメソッドは，NVList で指定された名前・値ペアを使って，オブジェクトに一つ以上のプロパティを追加します。このメソッドに対する in パラメタとして使用するために NVList オブジェクトを生成した場合は，Flags フィールドに 0 を設定し，NVList に追加した Any オブジェクトの TypeCode に TC_string を設定する必要があります。NVList クラスの詳細については，「23.14　NVList」を参照してください。
  - _list
    このオブジェクトに追加される名前・値ペアのリスト

22.6 PortableServer::Current

```
class PortableServer::Current : public CORBA::Current
```

Current クラスは，メソッド呼び出し先オブジェクトへアクセスするためのメソッドを提供します。このクラスは，複数のオブジェクトを実装するサーバントをサポートするために提供されていますが，任意のサーバントに対して，POA がディスパッチしたメソッドを呼び出すコンテキストの中では使用できません。

22.6.1 インクルードファイル

このクラスを使用するときは，poa_c.hh ファイルをインクルードしてください。

22.6.2 PortableServer::Current のメソッド

- PortableServer::POA *get_POA( );

  このメソッドは，コンテキストの中で，このメソッドを呼び出しているオブジェクトを実装している POA のリファレンスを返します。POA によってディスパッチされたメソッドのコンテキストの外部で，このメソッドが呼び出された場合は，NoContext 例外が発生します。

- PortableServer::ObjectId *get_object_id( );

  このメソッドは，呼び出されたコンテキストのオブジェクトの ObjectId を返します。POA によってディスパッチされたメソッドのコンテキストの外部で，このメソッドが呼び出された場合は，NoContext 例外が発生します。

22.7 Exception

```
class CORBA::Exception
```

Exception クラスは，システム例外クラス，およびユーザ例外クラスのベースクラスです。詳細については，「22.20 SystemException」を参照してください。

22.7.1 インクルードファイル

このクラスを使用するときは，corba.h ファイルをインクルードしてください。

22.8 Object

```
class CORBA::Object
```

すべての ORB オブジェクトは Object クラスから派生します。Object クラスは，次のメソッドを提供します。

- クライアントとオブジェクトをバインドするメソッド
- オブジェクトリファレンスを処理するメソッド
- オブジェクトの状態を確認したり，設定したりするメソッド

Object クラスによって提供されるメソッドは，ORB が実装します。

Borland Enterprise Server VisiBroker は，CORBA Object の仕様には規定されていない拡張機能を提供しています。詳細については，「22.8.3 CORBA::Object に対する VisiBroker の拡張機能」を参照してください。

22.8.1 インクルードファイル

このクラスを使用するときは，corba.h ファイルをインクルードしてください。

22.8.2 CORBA::Object のメソッド

- void _create_request(
  CORBA::Context_ptr ctx,
  const char *operation,
  CORBA::NVList_ptr arg_list,
  CORBA::NamedValue_ptr result,
  CORBA::Request_out request,
  CORBA::Flags req_flags);
  このメソッドは，動的起動インタフェースによる呼び出しの際に使用する，オブジェクトインプリメンテーションの Request を生成します。
  - ctx
    このリクエストに対応する Context。詳細については，「22.5 Context」を参照してください。
  - operation
    オブジェクトインプリメンテーションで実行されるオペレーションの名前
  - arg_list

オブジェクトインプリメンテーションに渡す引数のリスト。詳細については，「23.14　NVList」を参照してください。

- result
  オペレーションの結果。詳細については，「23.13　NamedValue」を参照してください。
- request
  生成された Request を指すポインタ。詳細については，「23.15　Request」を参照してください。
- req_flags
  arg_list に，一つ以上の NamedValue 項目が out 引数である場合，このフラグに OUT_LIST_MEMORY を設定します。

- void _create_request(
  CORBA::Context_ptr **ctx**,
  const char ***operation**,
  CORBA::NVList_ptr **arg_list**,
  CORBA::NamedValue_ptr **result**,
  CORBA::ExceptionList_ptr **eList**,
  CORBA::ContextList_ptr **ctxList**,
  CORBA::Request_out **request**,
  CORBA::Flags **req_flags**);
  このメソッドは，動的起動インタフェースによる呼び出しに使用する，オブジェクトインプリメンテーションの Request を生成します。
  - ctx
    このリクエストに対応する Context。詳細については，「22.5　Context」を参照してください。
  - operation
    オブジェクトインプリメンテーションで実行されるオペレーションの名前
  - arg_list
    オブジェクトインプリメンテーションに渡す引数のリスト。詳細については，「23.14　NVList」を参照してください。
  - result
    オペレーションの結果。詳細については，「23.13　NamedValue」を参照してください。
  - eList
    このリクエストの例外リスト
  - ctxList
    このリクエストの Context オブジェクトのリスト
  - request
    生成された Request を指すポインタ。詳細については，「23.15　Request」を参照してください。

- req_flags
  arg_list に，一つ以上の NamedValue 項目が out 引数である場合，このフラグに OUT_LIST_MEMORY を設定します。

- static CORBA::Object_ptr _duplicate(
  CORBA::Object_ptr obj);
  この静的メソッドは，指定された Object_ptr を複製し，そのオブジェクトを指すポインタを返します。オブジェクトのリファレンスカウントは 1 ずつ増やします。
  - obj
    複製されるオブジェクトポインタ
- CORBA::InterfaceDef_ptr _get_interface();
  このメソッドは，オブジェクトのインタフェース定義を指すポインタを返します。詳細については，「24.19　InterfaceDef」を参照してください。
- CORBA::ULong _hash(
  CORBA::ULong maximum);
  このメソッドは，オブジェクトのハッシュ値を返します。このオブジェクトが存在する間，ハッシュ値は変更されませんが，その値が一意である必要はありません。二つのオブジェクトが異なるハッシュ値を返す場合，それらのオブジェクトが同一のものでないことを表しています。
  - maximum
    返されるハッシュ値の上限（下限は 0）。
- CORBA::Boolean _is_a(
  const char *logical_type_id);
  このオブジェクトがリポジトリ ID に対応するインタフェースを実装する場合，このメソッドは TRUE を返します。そうでない場合は FALSE を返します。
  - logical_type_id
    チェックするリポジトリ ID
- CORBA::Boolean _is_equivalent(
  CORBA::Object_ptr other_object);
  指定されたオブジェクトポインタと，このオブジェクトが同一のオブジェクトインプリメンテーションを指す場合，このメソッドは TRUE を返します。そうでない場合は FALSE を返します。
  - other_object
    このオブジェクトと比較されるオブジェクトを指すポインタ
- static CORBA::Object_ptr _nil();
  この静的メソッドは，初期化に使用される NULL ポインタを返します。
- CORBA::Boolean _non_existent();
  オブジェクトが，すでに存在しないオブジェクトリファレンスによって表されている場合，このメソッドは TRUE を返します。

- CORBA::Request_ptr _request(
  const char* operation);
  このメソッドは，オブジェクトに対してメソッドを呼び出すために使われる Request を生成します。Request オブジェクトを指すポインタが返されます。詳細については，「23.15　Request」を参照してください。
  - operation
    呼び出されるオブジェクトメソッドの名前
- CORBA::Object_ptr _resolve_reference(
  const char* id);
  ユーザのクライアントアプリケーションは，指定サービス識別子のサーバ側インタフェースを解決するために，オブジェクトリファレンスに対してこのメソッドを呼び出せます。このメソッドは，指定のサービスを解決するために，サーバ側で ORB::resolve_initial_references メソッドを呼び出します。ORB::resolve_initial_references メソッドは，クライアントが該当するサーバ型にナロウできるオブジェクトリファレンスを返します。
  一般に，このメソッドは，サーバの属性を管理するクライアントアプリケーションによって使用されます。
  - id
    サーバ側で解決されるインタフェース名

22.8.3　CORBA::Object に対する VisiBroker の拡張機能

- static CORBA::Object _ptr _clone(
  CORBA::Object_ptr obj, CORBA::Boolean reset_connection = 1UL);
  このメソッドは，指定されたオブジェクトリファレンスをクローンします。
  - obj
    クローンされるオブジェクトリファレンス
  - reset_connection
    このパラメタは使用されません。
- static const CORBA::TypeInfo *_desc();
  このオブジェクトの型情報を返します。
- const char *_interface_name() const;
  このメソッドは，オブジェクトのインタフェース名を返します。
- CORBA::Boolean _is_bound() const;
  クライアントプロセスがオブジェクトインプリメンテーションと接続している場合，このメソッドは TRUE を返します。
- CORBA::Boolean _is_local() const;

オブジェクトインプリメンテーションが，クライアントアプリケーションと同じプロセス，またはアドレス空間に属している場合，このメソッドは TRUE を返します。

- CORBA::Boolean _is_persistent() const;
  オブジェクトがパーシステントオブジェクトである場合，このメソッドは TRUE を返します。トランジェントである場合は FALSE を返します。
- CORBA::Boolean _is_remote() const;
  オブジェクトインプリメンテーションが，クライアントアプリケーションと異なるプロセス，またはアドレス空間に属している場合，このメソッドは TRUE を返します。クライアントとオブジェクトインプリメンテーションは，同じホストに属しているとは限りません。
- CORBA::Long _ref_count() const;
  このオブジェクトのリファレンスカウントを返します。
- void _release();
  オブジェクトのリファレンスカウントを減らし，そのリファレンスカウントが 0 になったらオブジェクトを解放します。
- const char *_repository_id() const;
  このメソッドはオブジェクトのリポジトリ ID を返します。

22.9 ORB

```
class CORBA::ORB
```

ORB クラスは，ORB のインタフェースを提供します。このクラスは，特定の Object または Object アダプタから独立した形で，クライアントオブジェクトに対して使用できるメソッドを提供します。

Borland Enterprise Server VisiBroker は，CORBA ORB の仕様には規定されていない拡張機能を提供します。詳細については，「22.9.3 CORBA::ORB に対する VisiBroker の拡張機能」を参照してください。このクラスは，コネクションおよびスレッドを管理したり，サービスを活性化するメソッドを提供します。

22.9.1 インクルードファイル

このクラスを使用するときは，corba.h ファイルをインクルードしてください。

22.9.2 CORBA::ORB のメソッド

- CORBA::Boolean work_pending();
  このメソッドは，ORB に処理待ちの作業がある場合に true を返します。
- static CORBA::TypeCode_ptr create_alias_tc(
  const char *repository_id,
  const char *type_name,
  CORBA::TypeCode_ptr original_type);
  この静的メソッドは，指定された型，および名前を持つエイリアスの TypeCode を動的に生成します。
  - repository_id
    IDL コンパイラによって生成された識別子，または動的に構築された識別子
  - type_name
    エイリアスの型の名前
  - original_type
    このエイリアスが生成される元の型
- static CORBA::TypeCode_ptr create_array_tc(
  CORBA::Ulong length, TypeCode_ptr element_type);
  この静的メソッドは，配列の TypeCode を動的に生成します。
  - length
    配列要素の最大数

- element_type
  この配列に格納されている要素の型

- static CORBA::TypeCode_ptr **create_enum_tc**(
  const char ***repository_id**,
  const char ***type_name**,
  const CORBA::EnumMemberSeq& **members**);
  この静的メソッドは，指定された型，およびメンバを持つ，列挙体の TypeCode を動的に生成します。
  - repository_id
    IDL コンパイラによって生成された識別子，または動的に構築された識別子
  - type_name
    列挙体の型の名前
  - members
    列挙体メンバの値のリスト
- void **create_environment**(
  CORBA::Environment_out **env**);
  このメソッドは，Environment オブジェクトを生成します。
  - env
    新しく生成された Environment を指すポインタに設定されるリファレンス
- static CORBA::TypeCode_ptr **create_exception_tc**(
  const char ***repository_id**,
  const char ***type_name**,
  const CORBA::StructMemberSeq& **members**);
  この静的メソッドは，指定された型，およびメンバを持つ例外の TypeCode を動的に生成します。
  - repository_id
    IDL コンパイラによって生成された識別子，または動的に構築された識別子
  - type_name
    構造体の型の名前
  - members
    構造体メンバの値のリスト
- static CORBA::TypeCode_ptr **create_interface_tc**(
  const char ***repository_id**, const char ***type_name**);
  この静的メソッドは，指定された型を持つインタフェースの TypeCode を動的に生成します。
  - repository_id
    IDL コンパイラによって生成された識別子，または動的に構築された識別子
  - type_name

インタフェースの型の名前

- void create_list(
  CORBA::Long num, CORBA::NVList_out nvList);
  このメソッドは，指定された要素数を持つ NVList を生成し，そのリストに対するリファレンスを返します。
  - num
    リスト内の要素数
  - nvList
    新しく生成されたリストを指すポインタに初期化されます。
- void create_named_value(
  CORBA::NamedValue_out value);
  このメソッドは NamedValue オブジェクトを生成します。
- void create_operation_list(
  CORBA::OperationDef_ptr opDefPtr, CORBA::NVList_out nvList);
  このメソッドは，指定された OperationDef オブジェクトの引数リストを生成します。
- static CORBA::TypeCode_ptr create_recursive_sequence_tc(
  CORBA::Ulong bound, CORBA::Ulong offset);
  この静的メソッドは，再帰的シーケンスの TypeCode を動的に生成します。このメソッドの結果は，ほかの型の生成に使用できます。offset パラメタは，どの TypeCode がシーケンスの要素を表しているかを判別します。
  - bound
    シーケンス要素の最大数
  - offset
    現在の要素のタイプコードが生成されたバッファ内の位置
- static CORBA::TypeCode_ptr create_sequence_tc(
  CORBA::Ulong bound, CORBA::TypeCode_ptr element_type);
  この静的メソッドは，シーケンスの TypeCode を動的に生成します。
  - bound
    シーケンス要素の最大数
  - element_type
    このシーケンスに格納されている要素の型
- static CORBA::TypeCode_ptr create_string_tc(
  CORBA::Ulong bound);
  この静的メソッドは，文字列の TypeCode を動的に生成します。
  - bound

文字列の最大長

- static CORBA::TypeCode_ptr **create_struct_tc**(
  const char ***repository_id**,
  const char ***type_name**,
  const CORBA::StructMemberSeq& **members**);
  この静的メソッドは，指定された型とメンバを持つ構造体の TypeCode を動的に生成します。
  - repository_id
    IDL コンパイラによって生成された識別子，または動的に構築された識別子
  - type_name
    構造体の型の名前
  - members
    構造体メンバの値のリスト
- static CORBA::TypeCode_ptr **create_union_tc**(
  const char ***repository_id**,
  const char ***type_name**,
  CORBA::TypeCode_ptr **discriminator_type**,
  const CORBA::UnionMemberSeq& **members**);
  この静的メソッドは，指定された型，ディスクリミネータ，およびメンバを持つ union の TypeCode を動的に生成します。
  - repository_id
    IDL コンパイラによって生成された識別子，または動的に構築された識別子
  - type_name
    union の型の名前
  - discriminator_type
    union の識別型
  - members
    union メンバの値のリスト
- void **get_default_context**(
  CORBA::Context_out **contextPtr**);
  このメソッドは，VisiBroker によって保持されている，プロセスごとの Context のデフォルトを返します。デフォルトの Context は，DII リクエストを構成する際によく使用されます。詳細については，「22.5　Context」を参照してください。
  - contextPtr
    プロパティの値
- void **get_next_response**(

CORBA::Request_out **req**);

このメソッドは，遅延リクエストに対応する応答を待ちます。このメソッドを呼び出す前に，受信されることを待っている応答があるかどうかを判別するには，ORB::poll_next_response メソッドを使用します。

- req
  応答が得られたリクエスト情報を指すポインタが設定されます。

- ObjectIdList ***list_initial_services**();
  このメソッドは，アプリケーションで使用できるすべてのオブジェクトサービス名のリストを返します。それらのサービスには，ロケーションサービス，インタフェースリポジトリ，およびネームサービスが含まれます。返された名前は ORB::resolve_initial_references メソッドで，このサービスのトップレベルオブジェクトを取得するのに使用します。

- char ***object_to_string**(
  CORBA::Object_ptr **obj**);
  このメソッドは，指定されたオブジェクトリファレンスを文字列に変換します。この処理は，CORBA の仕様で「文字列化（stringification）」と呼ばれます。例えば，文字列に変換されたオブジェクトリファレンスをファイルなどに格納できます。ORB インプリメンテーションによって，オブジェクトリファレンスから文字列への変換方法が異なるので，このメソッドは ORB メソッドになっています。
  - obj
    文字列に変換されるオブジェクトを指すポインタ

- void **perform_work**();
  このメソッドは，ORB に処理の実行を指示します。

- CORBA::Boolean **poll_next_response**();
  遅延リクエストの応答を受信している場合，このメソッドは TRUE を返します。そうでない場合は FALSE を返します。このメソッドを呼び出しても遅延リクエストの応答を待ちません。

- CORBA::Object_ptr **resolve_initial_references**(
  const char * **identifier**);
  このメソッドは，ORB::list_initial_services メソッドによって返された名前と一致するインプリメンテーションオブジェクトを解決します。このメソッドよって返された解決済みオブジェクトは，適切なサーバ型にナロウできます。指定されたサービスが見つからない場合，InvalidName 例外が発生します。この節の「ObjectIdList *list_initial_services();」も参照してください。
  - identifier
    返されるトップレベルオブジェクトのサービスの名前。識別子の名前は，返されるオブジェクトの名前ではありません。
    アドオンされる ORB サービスの識別子を次に示します。
    ・ORBPolicyManager
    ・PolicyCurrent
    ・DynAnyFactory

・CodecFactory
・PlCurrent
・RootPOA
・LocationService
・InterfaceRepository
・VisiBrokerInterceptorControl
・NameService

- void **send_multiple_requests_deferred**(
  const CORBA::RequestSeq& **req**);
  このメソッドは，指定されたシーケンス内のすべてのクライアントリクエストを遅延リクエストとして送信します。ORB は，そのオブジェクトインプリメンテーションからの応答待ちをしません。クライアントアプリケーションが，ORB::get_next_response メソッドを使って各リクエストの応答を取得します。
  - req
    送信される遅延リクエストのシーケンス
- void **send_multiple_requests_oneway**(
  const CORBA::RequestSeq& **req**);
  このメソッドは，指定されたシーケンス内のすべてのクライアントリクエストを一方向リクエストとして送信します。一方向リクエストはオブジェクトインプリメンテーションからの応答を生成しないので，ORB はリクエストからの応答を待ちません。
  - req
    送信される一方向リクエストのシーケンス
- CORBA::Object_ptr **string_to_object**(
  const char ***str**);
  このメソッドは，オブジェクトを表す文字列をオブジェクトポインタに変換します。なお，その文字列は ORB::object_to_string メソッドを使って生成されたものでなければなりません。
  - str
    オブジェクトを表す文字列を指すポインタ
- static CORBA::ORB_ptr **_nil**();
  この静的メソッドは，初期化に使用できる NULL ORB ポインタを返します。
- void **run**();
  このメソッドは，ORB に処理の実行を開始させます。ORB はリクエストを受信し，ディスパッチします。この呼び出しは ORB がシャットダウンするまでプロセスを続行します。
- CORBA::ValueFactory **register_value_factory**(
  const char* **id**, CORBA::ValueFactory **factory**);
  ValueFactory を登録します。

- id
  ファクトリのリポジトリ ID
- factory
  登録する ValueFactory

- void **unregister_value_factory**(
  const char* **id**);
  ValueFactory の登録を削除します。
  - id
    ファクトリのリポジトリ ID
- CORBA::ValueFactory **lookup_value_factory**(
  const char* **id**);
  ValueFactory の登録を検索します。
  - id
    ファクトリのリポジトリ ID
- CORBA::Policy_ptr **create_policy**(
  CORBA::PolicyType **type**, const CORBA::Any& **val**);
  このメソッドは，指定された初期値で特定のタイプの Policy オブジェクトを作成するために使用します。サポートしていないポリシーを指定した場合，PolicyError 例外が発生します。
  - type
    作成される Policy オブジェクトの PolicyType
  - val
    作成される Policy オブジェクトの初期状態を設定するために使われる値
- static void **shutdown**(
  CORBA::Boolean **wait_for_completion**=0);
  このメソッドは，run メソッドをリターンさせます。すべてのオブジェクトアダプタがシャットダウンされ，そのアダプタ用のメモリが解放されます。

22.9.3 CORBA::ORB に対する VisiBroker の拡張機能

- CORBA::Object_ptr **bind**(
  const char* **logical_type_id**,
  const char ***poa_name**,
  const CORBA::OctetSequence& **oid**,
  const char ***host_name** = (const char*)NULL,
  const CORBA::BindOptions ***opt** = (CORBA::BindOptions*)NULL);

このメソッドは，VisiBroker ORB オブジェクトにバインドし，オブジェクトリファレンスを取得します。

- logical_type_id
  リポジトリ ID を識別する文字列
- poa_name
  POA 名を識別する文字列
- oid
  オブジェクト ID を識別する文字列
- host_name
  ORB オブジェクトを探すホスト名を識別する文字列。このパラメタはオプションです。
- opt
  このオプションは省略するか，または NULL を指定してください。

- CORBA::ULong **connection_count**();
  このメソッドは，クライアントアプリケーションが活性化状態のコネクションの数を返すために使用されます。
- void **connection_max**(
  CORBA::ULong **max_conn**);
  このメソッドは，クライアントアプリケーションが最大コネクション数を設定するために使用されます。また，コマンドラインオプションの-ORBconnectionMax を使ってもこのプロパティを設定できます。コマンドラインオプションの詳細については，「34. コマンドラインオプション（C++）」を参照してください。
  - max_conn
    最大コネクション数
- CORBA::ULong **connection_max**();
  このメソッドは，クライアントアプリケーションが最大コネクション数を返すために使用されます。
- static CORBA::TypeCode_ptr **create_wstring_tc**(
  CORBA::Ulong **bound**);
  この静的メソッドは，Unicode 文字列の TypeCode を動的に生成します。
  - bound
    文字列の最大長
- static VISPropertyManager_ptr **getPropertyManager**();
  このメソッドは，特定サーバエンジンによる POA の生成の際に使用する VISPropertyManager を取得するために使用されます。
  詳細については，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「これらのプロパティはいつ使用するか」を参照してください。返却された VISPropertyManager を解放しないでください。

- CORBA_Object_ptr **bind**(
  const char ***rep_id**,
  const char ***object_name** = (const char*)NULL,
  const char ***host_name** = (const char*)NULL,
  const CORBA_BindOptions ***opt**=(CORBA_BindOptions*)NULL);
  このメソッドは，VisiBroker ORB オブジェクトにバインドし，オブジェクトリファレンスを取得します。
  - rep_id
    リポジトリ ID を識別する文字列
  - object_name
    オブジェクト名を識別する文字列。このパラメタはオプションです。
  - host_name
    ORB オブジェクトを探すホスト名を識別する文字列。このパラメタはオプションです。
  - opt
    このオプションは省略するか，または NULL を指定してください。

22.10 Policy

```
class  CORBA::Policy  : public virtual CORBA::PseudoObject
```

Policy クラスは，オペレーションに影響を与える，特定の情報の利用を許可するための機構を，ORB およびオブジェクトサービスに提供します。

22.10.1 インクルードファイル

このクラスを使用するときは，corba.h ファイルをインクルードしてください。

22.10.2 Policy のメソッド

- CORBA::Policy *copy()
  このメソッドは Policy オブジェクトのコピーを返します。
- void destroy()
  このメソッドは Policy オブジェクトを削除します。
- CORBA::PolicyType policy_type()
  このメソッドは Policy オブジェクトの型を返します。

22.11 PortableServer::POA

```
class PortableServer::POA : public virtual CORBA::PseudoObject
```

POA クラスのオブジェクトがオブジェクトインプリメンテーション群を管理します。POA は，これらのオブジェクトを ObjectId で識別する，ネームスペースをサポートします。一つの POA は，ほかの複数の POA のネームスペースを提供します。そのネームスペースの中では，POA は既存 POA の子 POA として作成され，ルート POA から始まる階層を形成します。

ルート POA は，ORB::resolve_initial_references("RootPOA")を使用して取得します。

POA オブジェクトをほかのプロセスにエクスポートしたり，文字列化したりしないでください。POA オブジェクトのエクスポートや文字列化を試みると MARSHAL 例外が発生します。

22.11.1 インクルードファイル

このクラスを使用するときは，poa_c.hh ファイルをインクルードしてください。

22.11.2 PortableServer::POA のメソッド

- PortableServer::ObjectId* activate_object(
  PortableServer::Servant _p_servant);
  このメソッドは，バイト配列で構成されるオブジェクト ID を生成して返します。
  生成したオブジェクト ID と，_p_servant に指定したサーバントは，アクティブオブジェクトマップに登録されます。POA に UNIQUE_ID ポリシーがあり，_p_servant に指定したサーバントがすでにアクティブオブジェクトマップにある場合は，ServantAlreadyActive 例外が発生します。
  このメソッドを使用するには，POA に SYSTEM_ID ポリシーと RETAIN ポリシーが必要です。もし SYSTEM_ID ポリシーと RETAIN ポリシーがなければ，WrongPolicy 例外が発生します。
  - _p_servant
    アクティブオブジェクトマップに登録するサーバント
- void activate_object_with_id(
  const PortableServer::ObjectId& _oid,
  PortableServer::Servant _p_servant);
  このメソッドは，_oid で指定したオブジェクトを活性化して，そのオブジェクトを_p_servant で指定したサーバントにアクティブオブジェクトマップの中で対応づけます。アクティブオブジェクトマップの中で，すでに_oid にバインドされたサーバントがある場合は，ObjectAlreadyActive 例外が発生します。POA に UNIQUE_ID ポリシーがあり，_p_servant に指定したサーバントがすでにアクティブオブジェクトマップにある場合は，ServantAlreadyActive 例外が発生します。

このメソッドを使用するには，POA に RETAIN ポリシーが必要です。RETAIN ポリシーがなければ WrongPolicy 例外が発生します。

- _oid
  活性化するオブジェクトのオブジェクト ID
- _p_servant
  アクティブオブジェクトマップに登録するサーバント

- PortableServer::ImplicitActivationPolicy_ptr
  create_implicit_activation_policy(
  PortableServer::ImplicitActivationPolicyValue _value);
  このメソッドは，_value に指定した ImplicitActivationPolicy オブジェクトへのポインタを返します。Policy オブジェクトが不要になると，アプリケーションは，Policy オブジェクトから継承した destroy メソッドを呼び出します。
  POA 生成時に ImplicitActivationPolicy を指定しなかった場合のデフォルトは，NO_IMPLICIT_ACTIVATION です。
  - _value
    IMPLICIT_ACTIVATION を設定すると，POA はサーバントの暗黙的活性化をサポートします。この場合，SYSTEM_ID ポリシーと RETAIN ポリシーも必要です。
    NO_IMPLICIT_ACTIVATION を設定すると，POA はサーバントの暗黙的活性化をサポートしません。
- CORBA::Object_ptr create_reference(
  const char* _intf);
  このメソッドは，POA が生成した ObjectId の値と_intf に指定した値とをカプセル化したオブジェクトリファレンスを生成して返します。_intf は生成されるオブジェクトリファレンスの type_id になります。_intf に NULL を指定することもできます。このメソッドを呼び出しても活性化は起こりません。_intf に指定した値が，オブジェクトの最下位派生インタフェースを識別するものでもなく，派生元インタフェースのどれかを識別するものでもない場合は，このメソッドの動作は不定です。返されたオブジェクトの POA::reference_to_id メソッドを呼び出すと ObjectId が取得できます。
  このメソッドを使用するには，POA に SYSTEM_ID ポリシーと RETAIN ポリシーが必要です。SYSTEM_ID ポリシーと RETAIN ポリシーがなければ WrongPolicy 例外が発生します。
  - _intf
    生成するオブジェクトのクラスのリポジトリインタフェース ID
- CORBA::Object_ptr create_reference_with_id(
  const PortableServer::ObjectId& _oid, const char* _intf);
  このメソッドは，_oid に指定した値と_intf に指定した値とをカプセル化したオブジェクトリファレンスを生成して返します。_intf は生成されるオブジェクトリファレンスの type_id になります。_intf に NULL を指定することもできます。intf に指定した値が，オブジェクトの最下位派生インタフェースを識別するものでもなく，派生元インタフェースのどれかを識別するものでもない場合は，このメソッドの動作は不定です。このメソッドを呼び出しても活性化は起こりません。必要であれば，このメソッド

が返すオブジェクトリファレンスを複数のクライアントに渡して，そのリファレンスをクライアントからリクエストすることによってオブジェクトを活性化したり，デフォルトサーバントを使用したりできます。それは適用するポリシーに依存します。

POA に SYSTEM_ID ポリシーがあり，ObjectId 値がシステムによって，または POA 用に生成されていない場合は，BAD_PARAM システム例外が発生することがあります。

- _oid
  リファレンスを生成するオブジェクトのオブジェクト ID
- _intf
  生成するオブジェクトのクラスのリポジトリインタフェース ID

- PortableServer::IdAssignmentPolicy_ptr
  create_id_assignment_policy(
  PortableServer::IdAssignmentPolicyValue _value);
  このメソッドは，_value に指定した IdAssignmentPolicy オブジェクトへのポインタを返します。Policy オブジェクトが不要になると，アプリケーションは，Policy オブジェクトから継承した destroy メソッドを呼び出します。
  POA 生成時に IdAssignmentPolicy を指定しなかった場合，デフォルトは SYSTEM_ID です。
  - _value
    USER_ID を設定すると，POA で生成したオブジェクトに，アプリケーションだけがオブジェクト ID を割り当てます。SYSTEM_ID を設定すると，POA で生成したオブジェクトに，POA だけがオブジェクト ID を割り当てます。
- PortableServer::IdUniquenessPolicy_ptr
  create_id_uniqueness_policy(
  PortableServer::IdUniquenessPolicyValue _value);
  このメソッドは，_value に指定した IdUniquenessPolicy オブジェクトへのポインタを返します。Policy オブジェクトが不要になると，アプリケーションは，Policy オブジェクトから継承した destroy メソッドを呼び出します。
  POA 生成時に IdUniquenessPolicy を指定しなかった場合，デフォルトは UNIQUE_ID です。
  - _value
    UNIQUE_ID を設定すると，POA が活性化するサーバントはすべてオブジェクト ID を一つだけサポートします。MULTIPLE_ID を設定すると，POA が活性化するサーバントはそれぞれ一つ以上のオブジェクト ID をサポートできます。
- PortableServer::LifespanPolicy_ptr create_lifespan_policy(
  PortableServer::LifespanPolicyValue _value);
  このメソッドは，_value に指定した LifespanPolicy オブジェクトへのポインタを返します。Policy オブジェクトが不要になると，アプリケーションは，Policy オブジェクトから継承した destroy メソッドを呼び出します。
  POA 生成時に LifespanPolicy を指定しなかった場合，デフォルトは TRANSIENT です。

- _value
  TRANSIENT を設定した場合，POA に実装されたオブジェクトは，そのオブジェクトを最初に作成した POA インスタンスより長くは存続できません。一度トランジェント POA を非活性化したあとで，その POA から生成したオブジェクトリファレンスを使用しようとすると，OBJECT_NOT_EXIST 例外が発生します。PERSISTENT を設定した場合，POA に実装されたオブジェクトは，そのオブジェクトを最初に作成した POA インスタンスより長く存続できます。

- PortableServer::POA_ptr **create_POA**(
  const char* **_adapter_name**,
  PortableServer::POAManager_ptr **_a_POAManager**,
  const CORBA::PolicyList& **_policies**);
  このメソッドは，_a_POAManager に指定したオブジェクトの子オブジェクトとして，POA を _adapter_name に指定した名前で新規作成します。指定した親オブジェクトにすでに同名の子 POA がある場合は，PortableServer::AdapterAlreadyExists 例外が発生します。
  _policies に指定したポリシーは，新規作成した POA に対応づけられ，新規作成した POA の動作を制御します。矛盾したポリシーを設定した場合は，InvalidPolicy 例外が発生します。
  - _adapter_name
    新規作成する POA を指定する名前
  - _a_POAManager
    新規作成する POA の親 POA オブジェクト
  - _policies
    新規作成する POA に適用するポリシーの一覧

- PortableServer::RequestProcessingPolicy_ptr
  create_request_processing_policy(
  PortableServer::RequestProcessingPolicyValue **_value**);
  このメソッドは，_value に指定した RequestProcessingPolicy オブジェクトへのポインタを返します。Policy オブジェクトが不要になると，アプリケーションは，Policy オブジェクトから継承した destroy メソッドを呼び出します。
  POA 生成時に RequestProcessingPolicy を指定しなかった場合，デフォルトは USE_ACTIVE_OBJECT_MAP_ONLY です。
  - _value
    USE_ACTIVE_OBJECT_MAP_ONLY を設定し，オブジェクト ID がアクティブオブジェクトマップにない場合は，OBJECT_NOT_EXIST 例外をクライアントに返します。RETAIN ポリシーも必要です。
    USE_DEFAULT_SERVANT を設定し，オブジェクト ID がアクティブオブジェクトマップになく（または NON_RETAIN ポリシーがあり），デフォルトサーバントが set_servant メソッドで POA に登録されている場合は，リクエストをデフォルトサーバントにディスパッチします。デフォルトサーバントが登録されていない場合は，OBJ_ADAPTER 例外をクライアントに返します。MULTIPLE_ID ポリシーも必要です。

USE_SERVANT_MANAGER を設定し，オブジェクト ID がアクティブオブジェクトマップになく（または NON_RETAIN ポリシーがあり），サーバントマネージャが set_servant_manager メソッドで POA に登録されている場合は，そのサーバントマネージャはサーバントを捜し出すか，例外を発生します。サーバントマネージャが登録されていない場合，OBJ_ADAPTER 例外をクライアントに返します。

- PortableServer::ServantRetentionPolicy_ptr
  create_servant_retention_policy(
  PortableServer::ServantRetentionPolicyValue _value);
  このメソッドは，_value に指定した ServantRetentionPolicy オブジェクトへのポインタを返します。Policy オブジェクトが不要になると，アプリケーションは，Policy オブジェクトから継承した destroy メソッドを呼び出します。
  POA 生成時に ServantRetentionPolicy を指定しなかった場合，デフォルトは RETAIN です。
  - _value
    RETAIN を設定した場合，POA は活性化したサーバントをアクティブオブジェクトマップの中に保持します。NON_RETAIN を設定した場合，POA はサーバントを保持しません。
- PortableServer::ThreadPolicy_ptr create_thread_policy(
  PortableServer::ThreadPolicyValue _value);
  このメソッドは，_value に指定した ThreadPolicy オブジェクトへのポインタを返します。Policy オブジェクトが不要になると，アプリケーションは，Policy オブジェクトから継承した destroy メソッドを呼び出します。
  POA 生成時に ThreadPolicy を指定しなかった場合，デフォルトは ORB_CTRL_MODEL です。
  - _value
    ORB_CTRL_MODEL を設定すると，ORB は，ORB が制御する POA のリクエストをスレッドに割り当てます。マルチスレッド環境では，複数のスレッドを使用して，同時に複数のリクエストを送信できます。
    SINGLE_THREAD_MODEL を設定すると，POA へのリクエストは一つずつ順次処理されます。マルチスレッド環境では，POA からサーバントとサーバントマネージャに出されたすべてのリクエストは，マルチスレッドに透過的なコードを保証する方法で実行されます。
- void deactivate_object(
  const PortableServer::ObjectId& _oid);
  このメソッドは，_oid で指定したオブジェクトを非活性化します。非活性化したあとも，そのオブジェクトに対するアクティブなリクエストがすべてなくなるまでは，そのオブジェクトはリクエストの処理を続けます。そのオブジェクトに対して実行中のリクエストがすべて終了したときに，そのオブジェクトの ObjectId がアクティブオブジェクトマップから削除されます。
  サーバントマネージャが POA に対応づけられている場合は，ObjectId がアクティブオブジェクトマップから削除されたあとで，ServantActivator::etherealize メソッドが，そのオブジェクトと，対応づけられたサーバントに対して呼び出されます。エーテライズが完了するまで，必要に応じて，そのオブジェクトの再活性化は抑止されます。ただし，このメソッドは，リクエストやエーテライズの完了を待たないで，常に指定したオブジェクトを非活性化した直後にリターンします。

このメソッドを使用するには，POA に RETAIN ポリシーが必要です。RETAIN ポリシーがない場合，WrongPolicy 例外が発生します。

- _oid
  非活性化するオブジェクトのオブジェクト ID

- void **destroy**(
  CORBA::Boolean **_etherealize_objects**,
  CORBA::Boolean **_wait_for_completion**);
  このメソッドは，該当する POA オブジェクトとそのすべての子 POA を破棄します。最初に子 POA を破棄し，最後にカレントコンテナ POA を破棄します。必要であれば，破棄したあとで同じプロセスに同じ名前で POA を作成できます。
  - _etherealize_objects
    このパラメタに TRUE を設定し，POA に RETAIN ポリシーがあり，サーバントマネージャが POA に登録されている場合，アクティブオブジェクトマップ中の各アクティブオブジェクトで etherealize メソッドが呼び出されます。etherealize メソッドが呼び出される前に，見かけ上，POA の破棄が発生します。このため，POA にメソッドを呼び出す etherealize メソッドは，OBJECT_NOT_EXIST 例外を発生します。
  - _wait_for_completion
    このパラメタに TRUE を設定し，カレントスレッドが POA からディスパッチされた呼び出しでない場合，destroy メソッドは，アクティブなリクエストと etherealize メソッドの呼び出しがすべて完了したあとでリターンします。
    このパラメタに TRUE を設定し，カレントスレッドが POA からディスパッチされた呼び出しの場合，destroy メソッドは，BAD_INV_ORDER 例外が発生し，POA の破棄は発生しません。

- PortableServer::POA_ptr **find_POA**(
  const char* **_adapter_name**, CORBA::Boolean **_activate_it**);
  このメソッドの呼び出し先の POA オブジェクトが，指定した_adapter_name の POA の親である場合，子 POA を返します。
  - _adapter_name
    該当する POA に対応する AdapterActivator の名前
  - _activate_it
    このパラメタに TRUE を設定し，_adapter_name で指定した POA の子 POA がない場合は，POA の AdapterActivator（NULL 以外の場合）が呼び出されます。子 POA の活性化に成功した場合，その POA を返します。それ以外の場合は，AdapterNonExistent 例外が発生します。

- PortableServer::Servant **get_servant**();
  このメソッドは，POA に対応づけられたデフォルトサーバントを返します。対応づけられたサーバントがない場合，NoServant 例外が発生します。このメソッドを使用するには，POA に USE_DEFAULT_SERVANT ポリシーが必要です。USE_DEFAULT_SERVANT ポリシーがない場合，WrongPolicy 例外が発生します。

- PortableServer::ServantManager_ptr get_servant_manager();

  このメソッドは，POA に対応づけられた ServantManager オブジェクトへのポインタを返します。対応づけられたサーバントマネージャがない場合，NULL を返します。このメソッドを使用するには，POA に USE_SERVANT_MANAGER ポリシーが必要です。USE_SERVANT_MANAGER ポリシーがない場合，WrongPolicy 例外が発生します。

- CORBA::Object_ptr id_to_reference(

  PortableServer::ObjectId& _oid);

  このメソッドは，_oid に指定したオブジェクトがアクティブであればオブジェクトリファレンスを返します。アクティブでなければ，ObjectNotActive 例外が発生します。このメソッドを使用するには，POA に RETAIN ポリシーが必要です。RETAIN ポリシーがない場合，WrongPolicy 例外が発生します。

  - _oid

    リファレンスの取得対象オブジェクトのオブジェクト ID

- PortableServer::Servant id_to_servant(

  PortableServer::ObjectId& _oid);

  このメソッドの動作には次の 3 種類があります。

  - POA に RETAIN ポリシーがあり，アクティブオブジェクトマップに指定したオブジェクトがある場合，そのオブジェクトに対応づけられたサーバントを返します。
  - POA に USE_DEFAULT_SERVANT ポリシーがあり，POA にデフォルトサーバントが登録されている場合，登録されているデフォルトサーバントを返します。
  - 上記以外の場合は，ObjectNotActive 例外を出力します。

  このメソッドを使用するには，POA に USE_DEFAULT_SERVANT ポリシーまたは RETAIN ポリシーが必要です。どちらのポリシーもない場合，WrongPolicy 例外が発生します。

  パラメタの意味を次に示します。

  - _oid

    サーバントの取得対象オブジェクトのオブジェクト ID

- PortableServer::Servant reference_to_servant(

  CORBA::Object_ptr _reference);

  このメソッドの動作には次の 3 種類があります。

  - POA に RETAIN ポリシーがあり，アクティブオブジェクトマップに指定したオブジェクトがある場合，そのオブジェクトに対応づけられたサーバントを返します。
  - POA に USE_DEFAULT_SERVANT ポリシーがあり，POA にデフォルトサーバントが登録されている場合，登録されているデフォルトサーバントを返します。
  - 上記以外の場合は，ObjectNotActive 例外を出力します。

  このメソッドを使用するには，POA に RETAIN ポリシー，または USE_DEFAULT_SERVANT ポリシーが必要です。どちらのポリシーもない場合，WrongPolicy 例外が発生します。

reference_to_servant で引数となる reference が，その POA で作られていないと WrongAdapter 例外が発生します。

パラメタの意味を次に示します。

- _reference
  サーバントの取得対象オブジェクト

- PortableServer::ObjectId* reference_to_id(
  CORBA::Object_ptr _reference);
  このメソッドは，_reference に指定したオブジェクトにカプセル化されている ObjectId の値を返します。このメソッドは，_reference に指定したオブジェクトを作成した POA に対して呼び出した場合だけ有効です。このメソッドの呼び出し先 POA が，指定したオブジェクトを作成した POA と異なる場合，WrongAdapter 例外が発生します。このメソッドを呼び出すときに，_reference パラメタで指定するオブジェクトがアクティブになっている必要はありません。

  このメソッドが WrongPolicy 例外も出力するように IDL では規定されていますが，それは将来的な拡張性を考慮してのことです。

  - _reference
    ObjectId の取得対象オブジェクト

- PortableServer::ObjectId* servant_to_id(
  PortableServer::Servant _p_servant);
  このメソッドの動作には次の 4 種類があります。

  - POA に UNIQUE_ID ポリシーがあり，_p_servant に指定したサーバントがアクティブである場合，そのサーバントに対応づけられた ObjectId を返します。
  - POA に IMPLICIT_ACTIVATION ポリシーがあり，MULTIPLE_ID ポリシーがあるか，または _p_servant に指定したサーバントがアクティブでない場合，POA が生成した ObjectId，およびサーバントに対応づけられたリポジトリインタフェース ID を使用してサーバントを活性化して，その ObjectId を返します。
  - POA に USE_DEFAULT_SERVANT ポリシーがあり，_p_servant に指定したサーバントがデフォルトサーバントである場合，現在の呼び出しに対応づけられた ObjectId を返します。
  - 上記以外の場合は，ServantNotActive 例外を出力します。

  このメソッドを使用するには，USE_DEFAULT_SERVANT ポリシーが必要です。このポリシーがない場合でも，RETAIN ポリシーと，UNIQUE_ID ポリシー，または IMPLICIT_ACTIVATION ポリシーの組み合わせがあればこのメソッドを使用できます。この条件を満たしていない場合は，WrongPolicy 例外が発生します。

  パラメタの意味を次に示します。

  - _p_servant
    ObjectId の取得対象サーバント

- CORBA::Object_ptr servant_to_reference(
  PortableServer::Servant _p_servant);

このメソッドの動作には次の 4 種類があります。

- POA に RETAIN ポリシーと UNIQUE_ID ポリシーがあり，_p_servant に指定したサーバントがアクティブな場合，そのサーバントを活性化するための情報をカプセル化しているオブジェクトリファレンスを返します。
- POA に RETAIN ポリシーと IMPLICIT_ACTIVATION ポリシーがあり，さらに，MULTIPLE_ID ポリシーがあるか，または_p_servant に指定したサーバントがアクティブでない場合，POA が生成した ObjectId，およびサーバントに対応づけられたリポジトリインタフェース ID を使用してサーバントを活性化して，オブジェクトリファレンスを返します。
- このメソッドが，_p_servant に指定したサーバントへのリクエストを実行するコンテキストの中で呼び出された場合，現在の呼び出しに対応づけられたリファレンスを返します。
- 上記以外の場合は，ServantNotActive 例外を出力します。

POA がディスパッチしたメソッドのコンテキストの外でこのメソッドを呼び出す場合は，RETAIN ポリシーと，UNIQUE_ID ポリシーまたは IMPLICIT_ACTIVATION ポリシーが必要です。このメソッドが，_p_servant に指定したサーバントに対するリクエストを実行するコンテキストの中で呼び出されないで，さらに，上記のポリシーもない場合は，WrongPolicy 例外が発生します。

パラメタの意味を次に示します。

- _p_servant
  リファレンスの取得対象サーバント

- void **set_servant**(
  PortableServer::Servant **_p_servant**);
  このメソッドは，POA に対応するデフォルトサーバントを設定します。指定したサーバントは，アクティブオブジェクトマップの中にサーバントが登録されていないすべてのリクエストに対して適用されます。
  このメソッドを使用するには，POA に USE_DEFAULT_SERVANT ポリシーが必要です。USE_DEFAULT_SERVANT ポリシーがない場合，WrongPolicy 例外が発生します。
  - _p_servant
    デフォルトとして使用する，POA に対応するサーバント
- void **set_servant_manager**(
  PortableServer::ServantManager_ptr **_imagr**);
  このメソッドは，POA に対応するデフォルトサーバントマネージャを設定します。このメソッドは，POA が生成されたときだけ呼び出せます。POA に対応するサーバントマネージャがすでに設定されているときにこのメソッドを呼び出すと，BAD_INV_ORDER 例外が発生します。
  このメソッドを使用するには，POA に USE_SERVANT_MANAGER ポリシーが必要です。USE_SERVANT_MANAGER ポリシーがない場合，WrongPolicy 例外が発生します。
  - _imagr
    POA のデフォルトとして使用するサーバントマネージャ
- PortableServer::AdapterActivator_ptr **the_activator**();

このメソッドは，POA に対応づけられたアダプタアクティベータを返します。POA の作成直後は，POA にはアダプタアクティベータがありません。つまり，属性が NULL になります。システムによっては，ルート POA の持つアクティベータの一つをアプリケーションに割り当てさせることができます。

- void the_activator(
  PortableServer::AdapterActivator_ptr _val);
  このメソッドは，POA に対応するアダプタアクティベータを，指定されたものに設定します。
  - _val
    POA に対応づけるアダプタアクティベータ
- char* the_name();
  このメソッドは，POA を識別する属性を返します。この属性は，親 POA を基準に POA を識別する，read-only の属性です。この属性は，POA を作成したときに割り当てられたものです。ルート POA の名前はシステムに依存し，アプリケーションには依存しません。
- PortableServer::POA_ptr the_parent();
  このメソッドは，該当する POA の親 POA へのポインタを返します。ルート POA の親は null です。
- PortableServer::POAManager_ptr the_POAManager();
  このメソッドは，POA に対応づけられた POA マネージャへのポインタである read-only の属性を返します。

22.12 PortableServer::POAManager

```
class PortableServer::POAManager : public CORBA::PseudoObject
```

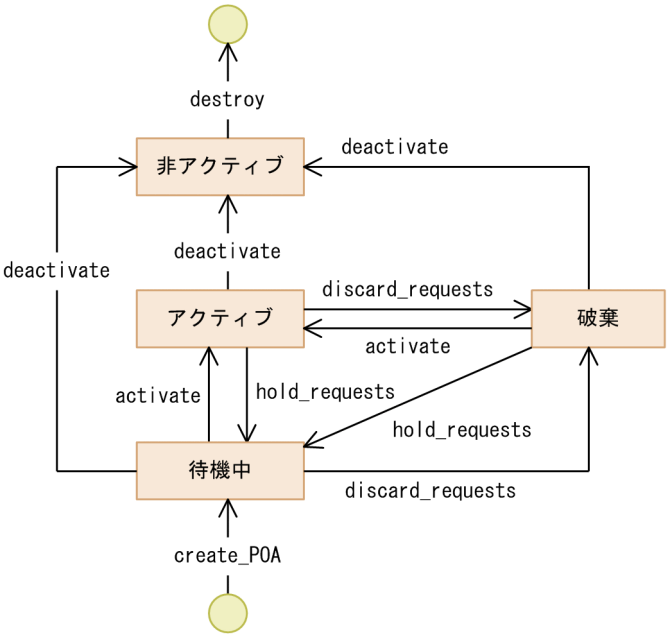
どの POA にも，POA マネージャが一つ対応づけられています。POA マネージャは一つ以上の POA オブジェクトに順番に対応づけることができます。POA マネージャは，対応づけられているすべての POA の処理状態をカプセル化しています。

POA マネージャの状態には次の 4 種類があります。

- アクティブ（ACTIVE）
- 非アクティブ（INACTIVE）
- 待機中（HOLDING）
- 破棄（DISCARDING）

作成直後の POA マネージャは「待機中」状態です。メソッド呼び出しによって POA マネージャの状態がどのように遷移するかを，次の図に示します。

図 22-1 C++での POA マネージャの状態遷移



22.12.1 インクルードファイル

このクラスを使用するときは，poa_c.hh ファイルをインクルードしてください。

22.12.2 PortableServer::POAManager のメソッド

- void activate();

  このメソッドは，POA マネージャの状態を「アクティブ」に変更します。POA マネージャに対応づけられているすべての POA は，リクエストの処理が可能な状態となります。POA マネージャが「非アクティブ」状態のときにこのメソッドを呼び出すと，AdapterInactive 例外が発生します。

- void deactivate(

  CORBA::Boolean _etherealize_objects,

  CORBA::Boolean _wait_for_completion);

  このメソッドは，POA マネージャの状態を「非アクティブ」に変更します。POA マネージャが「非アクティブ」状態のときは，対応づけられているすべての POA は，新規リクエストを含む実行開始前のリクエストを拒絶します。状態変更後の動作は，_etherealize_objects パラメタの設定値によって次のようになります。

  - 設定値が TRUE の場合，POA マネージャは，対応するサーバントマネージャで，RETAIN ポリシーと USE_SERVANT_MANAGER ポリシーが，すべてのアクティブオブジェクトに対する etherealize オペレーションを POA に実行させます。
  - 設定値が FALSE の場合，etherealize オペレーションを呼び出しません。この目的は，回復不能エラーなどの危機発生時に POA をシャットダウンする手段を開発者に提供することです。

  _wait_for_completion パラメタが FALSE の場合，このオペレーションは状態変更直後にリターンします。このパラメタが TRUE の場合で，該当する POA と同じ VisiBroker ORB に属するほかの POA がディスパッチした呼び出しコンテキスト内に，カレントスレッドがないときは，このオペレーションは，該当する POA マネージャに対応するすべての POA から，アクティブに実行中のリクエストがなくなるまで（つまり状態変更前に開始されたリクエストがすべて完了するまで），リターンしません。また，_etherealize_objects パラメタが TRUE の場合，このオペレーションは RETAIN ポリシーと USE_SERVANT_MANAGER ポリシーがあるすべての POA に対する etherealize の呼び出しが完了するまでリターンしません。このパラメタが TRUE の場合で，該当する POA と同じ VisiBroker ORB に属するほかの POA がディスパッチした呼び出しコンテキスト内に，カレントスレッドがあるときは，BAD_INV_ORDER 例外が発生し状態は変わりません。

- void discard_requests(

  CORBA::Boolean _wait_for_completion);

  このメソッドは，POA マネージャの状態を「破棄」に変更します。POA マネージャが「破棄」状態のときは，対応づけられているすべての POA は，到着したリクエストを破棄します。また，キューの中で実行待ち状態にあるリクエストも破棄されます。リクエストが破棄されると，個々のリクエストを発行したクライアントにそれぞれ，TRANSIENT システム例外が返されます。POA マネージャが「非アクティブ」状態のときにこのメソッドを呼び出すと，AdapterInactive 例外が発生します。

  _wait_for_completion パラメタが FALSE の場合，このオペレーションは状態変更直後にリターンします。このパラメタが TRUE の場合で，該当する POA と同じ VisiBroker ORB に属するほかの POA がディスパッチした呼び出しコンテキスト内に，カレントスレッドがないときは，このオペレーションは該当する POA マネージャに対応するすべての POA から，アクティブに実行中のリクエストがなくなるまで（つまり状態変更前に開始されたリクエストがすべて完了するまで），または POA マネージャ

の状態が「破棄」以外の状態に変更されるまで，リターンしません。このパラメタが TRUE の場合で，該当する POA と同じ VisiBroker ORB に属するほかの POA がディスパッチした呼び出しコンテキスト内に，カレントスレッドがあるときは，BAD_INV_ORDER 例外が発生し，状態は変わりません。

- void hold_requests(
  CORBA::Boolean _wait_for_completion);
  このメソッドは，POA マネージャの状態を「待機中」に変更します。POA マネージャが「待機中」状態のときは，対応づけられているすべての POA は，到着したリクエストをキューに保存します。このメソッドで POA マネージャを「待機中」状態にする前からキューの中で実行待ち状態にあったリクエストは，継続してキューの中で実行を待ちます。POA マネージャが「非アクティブ」状態のときにこのメソッドを呼び出すと，AdapterInactive 例外が発生します。
  _wait_for_completion パラメタが FALSE の場合，このオペレーションは状態変更直後にリターンします。このパラメタが TRUE の場合で，該当する POA と同じ VisiBroker ORB に属するほかの POA がディスパッチした呼び出しコンテキスト内に，カレントスレッドがないときは，このオペレーションは該当する POA マネージャに対応するすべての POA から，アクティブに実行中のリクエストがなくなるまで（つまり状態変更前に開始されたリクエストがすべて完了するまで），リターンしません。
- PortableServer::POAManager::State get_state();
  このメソッドは POAmanager の状態を返します。

22.13　PortableServer::POAManager::State

```
enum PortableServer::POAManager::State
```

この列挙体は POAManager の状態を表します。

22.13.1　インクルードファイル

この列挙体を使用するときは，poa_c.hh ファイルをインクルードしてください。

22.13.2　PortableServer::POAManager::State のメソッド

- ACTIVE
  POAManager がアクティブであることを表します。
- INACTIVE
  POAManager が非アクティブであることを表します。
- HOLDING
  POAManager が待機中であることを表します。
- DISCARDING
  POAManager が破棄中であることを表します。

それぞれの状態遷移については，「22.12　PortableServer::POAManager」を参照してください。

22.14 PortableServer::RefCountServantBase

```
class Portable Server::RefCountServantBase :
                        public virtual PortableServer::ServantBase
```

このクラスは，継承クラスとともに使用する PortableServer::ServantBase クラスとしてではなく，標準サーバントのリファレンスカウント用ミックスインクラスとして使用できます。「22.16 PortableServer::ServantBase」も参照してください。

22.14.1 インクルードファイル

このクラスを使用するときは，poa_c.hh ファイルをインクルードしてください。

22.14.2 PortableServer::RefCountServantBase のメソッド

- void _add_ref();
  このメソッドはリファレンスカウントを 1 ずつ増やします。正確なリファレンスカウントを提供するために，このメソッドをベースクラスからオーバーライドできます。
- void _remove_ref();
  このメソッドはリファレンスカウントを 1 ずつ減らします。正確なリファレンスカウントを提供するために，このメソッドをベースクラスからオーバーライドできます。

22.15 PortableServer::ServantActivator

```
class PortableServer::ServantActivator :
                        public PortableServer::ServantManager
```

POA に RETAIN ポリシーがある場合は，PortableServer::ServantActivator オブジェクトであるサーバマネージャを使用します。

22.15.1 インクルードファイル

このクラスを使用するときは，poa_c.hh ファイルをインクルードしてください。

22.15.2 PortableServer::ServantActivator のメソッド

- void **etherealize**(
  PortableServer::ObjectId& **oid**,
  PortableServer::POA_ptr **adapter**,
  PortableServer::Servant **serv**,
  CORBA::Boolean **cleanup_in_progress**,
  CORBA::Boolean **remaining_activations**);
  このメソッドは，oid で指定したオブジェクトのサーバントを非活性化するときに，指定したアダプタによって呼び出されます。このメソッドは，RETAIN ポリシーと USE_SERVANT_MANAGER ポリシーがあることを前提とします。
  - oid
    非活性化するサーバントを持つオブジェクトのオブジェクト ID
  - adapter
    アクティブなオブジェクトのスコープを持つ POA
  - serv
    非活性化するサーバント
  - cleanup_in_progress
    このパラメタに TRUE を設定した場合，etherealize_objects パラメタに TRUE を設定した deactivate メソッド，または destroy メソッドを呼び出したときに，このメソッドが呼び出されます。このパラメタに TRUE を設定しない場合，それ以外の理由でこのメソッドが呼び出されます。
  - remaining_activations
    serv に指定したサーバントが，adapter で指定した POA で他オブジェクトに対応づけられている場合は TRUE を設定し，それ以外の場合は FALSE を設定します。

- PortableServer::Servant incarnate(
  const PortableServer::ObjectId& oid,
  PortableServer::POA_ptr adapter);
  このメソッドは，oid に指定した非アクティブ状態のオブジェクトに対するリクエストを POA が受け取るたびに，POA によって呼び出されます。このメソッドは，RETAIN ポリシーと USE_SERVANT_MANAGER ポリシーがあることを前提とします。
  このメソッドを使用するためには，指定したオブジェクトに対応する適切なサーバントを探索，および生成するサーバントマネージャのインプリメンテーションを，ユーザが提供してください。このメソッドが返すサーバントは，アクティブオブジェクトマップにも登録されます。以後，アクティブなオブジェクトに対するリクエストは，サーバントマネージャを呼び出さないで直接そのオブジェクトに対応づけられたサーバントに送られます。
  このメソッドが返すサーバントが，すでにほかのオブジェクト ID に対してアクティブで，POA に UNIQUE_ID ポリシーがある場合，OBJ_ADAPTER 例外が発生します。
  - oid
    活性化するサーバントを持つオブジェクトのオブジェクト ID
  - adapter
    オブジェクトを活性化するスコープを持つ POA

22.16　PortableServer::ServantBase

```
class PortableServer::ServantBase
```

Portable::ServantBase クラスは，ご使用のサーバアプリケーションのベースクラスです。

22.16.1　インクルードファイル

このクラスを使用するときは，poa_c.hh ファイルをインクルードしてください。

22.16.2　PortableServer::ServantBase のメソッド

- void _add_ref();
  このメソッドは該当するサーバントのリファレンスカウントを追加します。デフォルトインプリメンテーションは何もしないので，このクラスから派生したクラスのリファレンスカウント機能を提供するには，このメソッドをオーバーライドしてください。
- PortableServer::POA_ptr _default_POA();
  このメソッドは，デフォルト VisiBroker ORB に対して，カレントプロセスのデフォルト VisiBroker ORB の，ルート POA へのオブジェクトリファレンスを返します。これは，ORB::resolve_initial_references("RootPOA")の呼び出しと同じ戻り値です。必要に応じて，PortableServer::ServantBase から派生したクラスにこのメソッドをオーバーライドして，独自に選択した POA を返せます。
- CORBA::InterfaceDef_ptr _get_interface();
  このメソッドは該当するオブジェクトのインタフェース定義を指すポインタを返します。詳細については，「24.19.2　InterfaceDef のメソッド」を参照してください。
- CORBA::Boolean _is_a(
  const char *rep_id);
  該当するサーバントがリポジトリ ID に対応するインタフェースを実装する場合，このメソッドは TRUE を返します。そうでない場合は FALSE を返します。
  - rep_id
    チェックするリポジトリ ID
- void _remove_ref();
  このメソッドは該当するサーバントのリファレンスカウントを削除します。デフォルトインプリメンテーションは何もしないので，このクラスから派生したクラスのリファレンスカウント機能を提供するには，このメソッドをオーバーライドしてください。

22.17 PortableServer::ServantLocator

```
class PortableServer::ServantLocator :
                        public PortableServer::ServantManager
```

POA に NON_RETAIN ポリシーがある場合，POA は PortableServer::ServantLocator オブジェクトであるサーバントマネージャを使用します。サーバントマネージャが返すサーバントは，1 件のリクエストにだけ使用できます。

POA は，このサーバントマネージャが返すサーバントは 1 件のリクエストに対してだけ使用されるということを知っているので，サーバントマネージャのメソッドに追加情報を提供できます。これによってサーバントマネージャの 1 組のメソッドは，PortableServer::ServantLocator サーバントマネージャとは異なる処理を実行できます。

22.17.1 インクルードファイル

このクラスを使用するときは，poa_c.hh ファイルをインクルードしてください。

22.17.2 PortableServer::ServantLocator のメソッド

- PortableServer::Servant preinvoke(
  const PortableServer::ObjectId& oid,
  PortableServer::POA_ptr adapter,
  const char* operation,Cookie& the_cookie);
  このメソッドは，POA が現在アクティブの状態にないオブジェクトへのリクエストを受け取ったときに呼び出されます。このメソッドは，NON_RETAIN ポリシーと USE_SERVANT_MANAGER ポリシーがあることを前提とします。
  ユーザが提供するサーバントマネージャは，oid に指定したオブジェクトに対応する適切なサーバントを探索，および作成する必要があります。
  - oid
    到着したリクエストに対応するオブジェクト ID
  - adapter
    オブジェクトを活性化する POA
  - operation
    サーバントが返されるときに POA が呼び出すオペレーションの名前
  - the_cookie
    サーバントマネージャがあとで postinvoke メソッドに設定して使用できる不特定の値

- void **postinvoke**(
  const PortableServer::ObjectId& **oid**,
  PortableServer::POA_ptr **adapter**,
  const char* **operation**,
  Cookie **the_cookie**,
  PortableServer::Servant **the_servant**);
  このメソッドは，サーバントがリクエストの実行を完了するときに呼び出されます。このメソッドは，POA に NON_RETAIN ポリシーと USE_SERVANT_MANAGER ポリシーがあることを前提とします。このメソッドは，オブジェクトに対するリクエストの一部とみなされます。つまり，メソッドが正常終了したのに postinvoke メソッドがシステム例外を出力した場合は，このメソッドの正常リターンはオーバーライドされ，リクエストは例外を発生して終了します。
  POA が認識しているサーバントをデストラクトした場合，結果は不定です。
  - oid
    到着したリクエストに対応するオブジェクト ID
  - adapter
    オブジェクトを活性化する POA
  - operation
    サーバントが返されるときに POA が呼び出すオペレーションの名前
  - the_cookie
    サーバントマネージャが preinvoke メソッドに設定して使用できる不特定の値
  - the_servant
    オブジェクトに対応するサーバント

22.18　PortableServer::ServantManager

```
class PortableServer::ServantManager
```

サーバントマネージャは，POA に対応づけられています。サーバントマネージャによって POA は，POA が「非アクティブ」状態のオブジェクトに対するリクエストを受信したときに，オンデマンドでオブジェクトを活性化できます。

PortableServer::ServantManager クラスはメソッドを持たないで，ほかの二つのクラス（PortableServer::ServantActivator と Portableserver::ServantLocator）のベースクラスとして使用します。詳細については，「22.15　PortableServer::ServantActivator」，および「22.17　PortableServer::ServantLocator」を参照してください。PortableServer::ServantActivator クラスを使用するには，POA に RETAIN ポリシーが必要です。Portableserver::ServantLocator クラスを使用するには，POA に NON_RETAIN ポリシーが必要です。

22.18.1　インクルードファイル

このクラスを使用するときは，poa_c.hh ファイルをインクルードしてください。

22.19 PortableServer::ForwardRequest

```
class PortableServer::ForwardRequest :public CORBA::UserException
```

この例外を使用して，別のオブジェクトにリクエストを転送するようにクライアントに指示できます。PortableServer::ServantLocator の preinvoke()メソッド，および PortableServer::ServantActivator の incarnate()メソッドで使用できます。

22.19.1 インクルードファイル

このクラスを使用するときは，poa_c.hh ファイルをインクルードしてください。

22.19.2 PortableServer::ForwardRequest のメソッド

- PortableServer::ForwardRequest(
  CORBA::Object_ptr ref);
  このメソッドは，指定されたプロパティを持つ ForwardRequest オブジェクトを生成します。
  - ref
    転送するオブジェクトリファレンス

22.20 SystemException

```
class CORBA::SystemException : public CORBA::Exception
```

SystemException クラスは，VisiBroker ORB またはオブジェクトインプリメンテーションに対して発生する標準システム例外を報告します。このクラスは，Exception クラスから派生し，出力ストリームに例外の名前，または記述を出力するメソッドを提供します。Exception クラスについては，「22.7 Exception」を参照してください。

SystemException オブジェクトには，例外の原因となるオペレーションが完了しているかを表す，完了ステータスが含まれます。また，SystemException オブジェクトは，設定，および参照できるマイナーコードも含んでいます。

22.20.1 インクルードファイル

このクラスを使用するときは，corba.h ファイルをインクルードしてください。

22.20.2 SystemException のメソッド

- CORBA::SystemException(
  CORBA::ULong minor = 0,
  CORBA::CompletionStatus status = CORBA::COMPLETED_NO);
  このメソッドは，指定されたプロパティを持つ SystemException オブジェクトを生成します。
  - minor
    マイナーコード
  - status
    次に示す完了ステータスのどれか一つ
    CORBA::COMPLETED_YES
    CORBA::COMPLETED_NO
    CORBA::COMPLETED_MAYBE
- CORBA::CompletionStatus completed() const;
  このメソッドは，オブジェクトの完了ステータスを返します。
- void completed(
  CORBA::CompletionStatus status);
  このメソッドは，オブジェクトの完了ステータスを設定します。
  - status
    次に示す完了ステータスのどれか一つ

CORBA::COMPLETED_YES
CORBA::COMPLETED_NO
CORBA::COMPLETED_MAYBE

- CORBA::ULong **minor**() const;
  このメソッドは，オブジェクトのマイナーコードを返します。
- void **minor**(
  CORBA::ULong **val**);
  このメソッドは，オブジェクトのマイナーコードを設定します。
  - val
    マイナーコード
- static CORBA::SystemException *_**downcast**(
  CORBA::Exception ***exc**);
  このメソッドは，指定された Exception ポインタを SystemException ポインタにダウンキャストしようとします。指定されたポインタが SystemException オブジェクトまたは SystemException から派生したオブジェクトを指す場合，オブジェクトを指すポインタが返されます。指定されたポインタが SystemException オブジェクトを指していない場合は，NULL ポインタが返されます。

  注
  このメソッドを使って，Exception オブジェクトのリファレンスカウントを増やすことはできません。

  - exc
    ダウンキャストされる Exception ポインタ

  システム例外一覧を次の表に示します。

表 22-1　システム例外一覧（C++）

例外	例外の内容
BAD_CONTEXT	サーバに無効コンテキストが渡されました。
BAD_INV_ORDER	オペレーション要求の前に，必要な前提条件オペレーションが呼び出されていません。
BAD_OPERATION	無効オペレーションが実行されました。
BAD_PARAM	無効なパラメタが引き渡されました。
BAD_TYPECODE	ORB が不正な TypeCode を検出しました。
CODESET_INCOMPATIBLE	クライアントとサーバのコードセットに互換がないため，通信に失敗しました。
COMM_FAILURE	通信障害が発生しました。
DATA_CONVERSION	データ変換エラーが発生しました。
FREE_MEM	メモリを解放できません。
IMP_LIMIT	インプリメンテーションの上限に違反しました。

例外	例外の内容
INITIALIZE	必要な初期化が実行されませんでした。
INTERNAL	内部エラーが発生しました。
INTF_REPOS	インタフェースリポジトリへのアクセスエラーが発生しました。
INV_FLAG	無効フラグが指定されました。
INV_IDENT	識別子の構文が無効です。
INV_OBJREF	無効なオブジェクトリファレンスが検出されました。
INV_POLICY	無効なポリシーの変更が検出されました。
INVALID_TRANSACTION	トランザクションコンテキストが不正です。
MARSHAL	マーシャルパラメタまたは結果が不当です。
NO_IMPLEMENT	オペレーションのインプリメンテーションが使用できません。
NO_MEMORY	動的メモリ割り当て障害が発生しました。
NO_PERMISSION	許可されていないオペレーションを実行しようとしました。
NO_RESOURCES	必要な資源を取得できませんでした。
NO_RESPONSE	クライアントが送信したリクエストの応答がまだありません。
OBJ_ADAPTER	オブジェクトアダプタが障害を検出しました。
OBJECT_NOT_EXIST	リクエストされたオブジェクトが存在していません。
PERSIST_STORE	パーシステントストレージ障害が発生しました。
REBIND	クライアントが，QoS ポリシーに矛盾する IOR を受信しました。
TIMEOUT	オペレーションがタイムアウトしました。
TRANSACTION_REQUIRED	リクエスト時に無効なトランザクションコンテキストがトランザクションサービスに渡されましたが，アクティブなトランザクションが必要です。
TRANSACTION_ROLLEDBACK	リクエストに対応するトランザクションがすでにロールバックされているか，またはロールバック用にマーキングされています。
TRANSIENT	通信エラーが検出されましたが，再接続できる場合があります。
UNKNOWN	未知の例外です。

22.21 UserException

```
class CORBA::UserException : public CORBA::Exception
```

UserException ベースクラスは，オブジェクトインプリメンテーションが発生させるユーザ例外を派生します。このクラスは，Exception クラスから派生しています。

22.22 VISPropertyManager

```
class VISPropertyManager
```

このクラスは，ユーザ作成のサーバエンジンプロパティの変更が必要な場合に使用します。詳細については，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「これらのプロパティはいつ使用するか」を参照してください。

22.22.1 インクルードファイル

このクラスを使用する場合には corba.h ファイルをインクルードしてください。

22.22.2 VISPropertyManager のメソッド

- void addProperty(
  const char* property, const char* value);
  プロパティを設定します。
  - property
    プロパティ名称を表す文字列
  - value
    プロパティの指定値を表す文字列
- void addProperty(
  const char* property, CORBA::Boolean value);
  プロパティを設定します。
  - property
    プロパティ名称を表す文字列
  - value
    プロパティの指定値を表す Boolean
- void addProperty(
  const char* property, CORBA::ULong value);
  プロパティを設定します。
  - property
    プロパティ名称を表す文字列
  - value
    プロパティの指定値を表す定数値

- char* **getString**(
  const char* **property**);
  文字列型を指定されるプロパティ値を取得します。存在しないプロパティや，指定値が文字列型ではないプロパティを引数に指定した場合，CORBA::BAD_PARAM 例外が発生します。
  - property
    プロパティ名称を表す文字列
- CORBA::Boolean **getBoolean**(
  const char* **property**);
  boolean 型を指定されるプロパティ値を取得します。存在しないプロパティや，指定値が boolean 型ではないプロパティを引数に指定した場合，CORBA::BAD_PARAM 例外が発生します。
  - property
    プロパティ名称を表す文字列
- CORBA::ULong **getULong**(
  const char* **property**);
  整数型を指定されるプロパティ値を取得します。存在しないプロパティや，指定値が整数型ではないプロパティを引数に指定した場合，CORBA::BAD_PARAM 例外が発生します。
  - property
    プロパティ名称を表す文字列

23 動的インタフェースとクラス（C++）

この章では，C++言語のクライアントアプリケーションが使用する動的起動インタフェース，およびオブジェクトサーバが使用する動的スケルトンインタフェースをサポートするクラスについて説明します。

23.1 Any

```
class CORBA::Any
```

このクラスは，値がタイプセーフで渡せるような IDL 型を表します。このクラスのオブジェクトは，オブジェクトの型を定義する TypeCode を指すポインタ，およびオブジェクトの値を指すポインタを持っています。オブジェクトを構築・コピーしたりデストラクトしたりするためのメソッド，オブジェクトの型や値を確認したり初期化したりするためのメソッド，およびオブジェクトをストリームに読み込んだり書き込んだりするためのストリーム演算子が，このクラスによって提供されます。コードサンプル 23-1 に Any の生成と使用方法を示します。

コードサンプル 23-1　Any オブジェクトの生成と使用

```
// create an any object
CORBA::Any anObject;
// use the typecode operator to specify that
// 'anObject' object can store long
anObject <<= CORBA::_tc_long;
```

23.1.1 インクルードファイル

このクラスを使用するときは，corba.h ファイルをインクルードしてください。

23.1.2 Any のメソッド

- CORBA::Any();
  これは，空の Any オブジェクトを生成するデフォルトのコンストラクタです。
- CORBA::Any(
  const CORBA::Any& val);
  これは，指定されたターゲットのコピーである Any オブジェクトを生成するコピーコンストラクタです。
  - val
    コピーされるオブジェクト
- CORBA::Any(
  CORBA::TypeCode_ptr tc, void *value,
  CORBA::Boolean release = 0);
  このコンストラクタは，指定された値，および TypeCode で初期化された Any オブジェクトを生成します。
  - tc
    この Any に含まれる値の TypeCode

- value
  この Any に含まれる値
- release
  TRUE を設定した場合，この Any オブジェクトがデストラクトされた際に，Any オブジェクトの値に対応するメモリが解放されます。

- static CORBA::Any_ptr _duplicate(
  CORBA::Any_ptr ptr);
  この静的メソッドは，指定されたオブジェクトのリファレンスカウントを増やし，そのポインタを返します。
  - ptr
    複製される Any
- static CORBA::Any_ptr _nil();
  この静的メソッドは，初期化の際に使用できる NULL ポインタを返します。
- static void _release(
  CORBA::Any_ptr ptr);
  この静的メソッドは，指定されたオブジェクトのリファレンスカウントを減らします。カウントが 0 になった場合，そのオブジェクトによって管理されていたすべてのメモリが解放され，そのオブジェクトは削除されます。
  - ptr
    解放される Any

23.1.3 初期化演算子

```
void operator<<=(CORBA::Short);
   void operator<<=(CORBA::UShort);
   void operator<<=(CORBA::Long);
   void operator<<=(CORBA::ULong);
   void operator<<=(CORBA::Float);
   void operator<<=(CORBA::Double);
   void operator<<=(const CORBA::Any&);
   void operator<<=(const char *);
   void operator<<=(CORBA::LongLong);
   void operator<<=(CORBA::ULongLong);
   void operator<<=(CORBA::LongDouble);
```

これらの演算子は，指定された値でこのオブジェクトを初期化します。これらの演算子によって，指定された値の TypeCode が自動的に設定されます。この Any オブジェクトの release フラグに TRUE が設定されている場合，新しい値が代入される前に，この Any オブジェクトに格納されていた値が解放されます。

- void operator<<=(
  CORBA::TypeCode_ptr tc);

このメソッドは，指定された値の TypeCode でこのオブジェクトを初期化します。

- tc

  この Any に対して設定する TypeCode

23.1.4 抽出演算子

```
CORBA::Boolean operator>>=(CORBA::Short&) const;
   CORBA::Boolean operator>>=(CORBA::UShort&) const;
   CORBA::Boolean operator>>=(CORBA::Long&) const;
   CORBA::Boolean operator>>=(CORBA::ULong&) const;
   CORBA::Boolean operator>>=(CORBA::Float&) const;
   CORBA::Boolean operator>>=(CORBA::Double&) const;
   CORBA::Boolean operator>>=(CORBA::Any&) const;
   CORBA::Boolean operator>>=(char *&) const;
   CORBA::Boolean operator>>=(CORBA::LongLong&) const;
   CORBA::Boolean operator>>=(CORBA::ULongLong&) const;
   CORBA::Boolean operator>>=(CORBA::LongDouble&) const;
```

これらの演算子は，このオブジェクトからの値を，指定されたターゲットに格納します。ターゲットの TypeCode が，格納されている値の TypeCode と一致しない場合，FALSE が返され，値は抽出されません。一致する場合は，ターゲットに格納されている値が代入され，TRUE が返されます。

- CORBA::Boolean operator>>=(
  CORBA::TypeCode_ptr& tc) const;

  このメソッドは，このオブジェクトに格納されている値の TypeCode を抽出します。

  - tc

    この Any の TypeCode が格納されているオブジェクト

23.2 ContextList

```
class CORBA::ContextList
```

このクラスには，オペレーション要求に対応する可能性のあるコンテキストのリストが含まれます。
「23.15 Request」も参照してください。

23.2.1 ContextList のメソッド

- CORBA::ContextList();
  このメソッドは，空の Context リストを構築します。
- void add(
  const char *ctx);
  このメソッドは，指定されたコンテキストをこのオブジェクトのリストに追加します。
  - ctx
    リストに追加されるコンテキスト
- void add_consume(
  char *ctx);
  このメソッドは，指定されたコンテキストコードをこのオブジェクトのリストに追加します。引数で指定されたコンテキストの所有権は，このリストによって引き受けられます。このメソッドを呼び出したあとに，コンテキストへアクセスしたり，コンテキストを解放したりすることはできません。
  - ctx
    リストに追加されるコンテキスト
- CORBA::ULong count() const;
  このメソッドは，現在リストに格納されている項目数を返します。
- const char *item(
  CORBA::Long index);
  このメソッドは，指定されたインデックスのリストに格納されているコンテキストを指すポインタを返します。そのインデックスが不当な場合，NULL ポインタが返されます。返されたコンテキストは解放できません。リストからコンテキストを削除するには，remove メソッドを使用します。
  - index
    返されるコンテキストのインデックス。インデックスは 0 から始まります。
- void remove(
  CORBA::long index);
  このメソッドは，指定されたインデックスを持つコンテキストをリストから削除します。インデックスが不当な場合は，削除されません。

- index
  削除されるコンテキストのインデックス。インデックスは 0 から始まります。

- static CORBA::ContextList_ptr _duplicate(
  CORBA::ContextList_ptr ptr);
  この静的メソッドは，オブジェクトのリファレンスカウントを増やし，そのオブジェクトを指すポインタを返します。
  - ptr
    複製されるオブジェクト
- static CORBA::ContextList_ptr _nil();
  この静的メソッドは，初期化の際に使用できる NULL ポインタを返します。
- static void _release(
  CORBA::ContextList *ptr);
  この静的メソッドは，このオブジェクトのリファレンスカウントを減らします。そのカウントが 0 になると，オブジェクトによって管理されていたすべてのメモリが解放され，そのオブジェクトが削除されます。
  - ptr
    解放されるオブジェクト

23.3 DynamicImplementation

```
class PortableServer::DynamicImplementation :
                        public virtual PortableServer::ServantBase
```

このベースクラスを使って，IDL コンパイラによって生成されたスケルトンクラスの代わりに，動的スケルトンインタフェースを使用するオブジェクトインプリメンテーションを派生します。このクラスから派生させるには，invoke メソッドと，_primary_interface()メソッドのインプリメンテーションを提供する必要があります。

23.3.1 DynamicImplementation のメソッド

- virtual void **invoke**(
  CORBA::ServerRequest_ptr **request**) = 0;
  このメソッドは，オブジェクトインプリメンテーションに対するクライアントオペレーション要求が受信された際に，必ず POA によって呼び出されます。このメソッドのインプリメンテーションには，ServerRequest オブジェクトの内容の有効化，リクエスト条件を満たすために必要な処理，およびクライアントに対する結果の送信が含まれます。ServerRequest クラスの詳細については，「23.16 ServerRequest」を参照してください。
  - request
    クライアントのオペレーション要求を表す ServerRequest オブジェクト
- virtual CORBA::RepositoryId **_primary_interface**(
  const PortableServer::ObjectId& **oid**,
  PortableServer::POA_ptr **poa**) const;
  このメソッドは，POA によってコールバックとして呼び出されます。 DynamicImplementation クラスから継承したサーバントは，このメソッドを実装する必要があります。このメソッドを直接呼び出し以外の状況で呼び出すと，予想できない結果が生じることがあります。_primary_interface メソッドは，入力パラメタとして ObjectId 値と POA_ptr を受け取り，その oid に対して最下位派生インタフェースを示す有効な RepositoryId を返します。

23.4 DynAny

```
class DynamicAny::DynAny : public CORBA::PseudoObject
```

コンパイル時にデータ型が定義されなかった場合，実行時にクライアントアプリケーション，またはサーバがこのクラスのオブジェクトを使用して，データ型の作成と解釈をします。DynAny は，基本型（boolean，int，float など），または複合型（struct または union）を格納できます。DynAny に含まれる型は作成時に定義され，オブジェクトの存在期間内は変更できません。

DynAny オブジェクトは，データ型を，値を持つ一つ以上のコンポーネントとして表すことがあります。next，seek，rewind，および current_component メソッドを使用して，各コンポーネント間を自由に行き来できます。

DynAnyFactory は ORB::resolve_initial_references("DynAnyFactory")の呼び出しによって生成されます。ファクトリは，基本型や複合型の生成に使用されます。DynAnyFactory は DynamicAny モジュールに属します。

基本型の DynAny オブジェクトは，DynAnyFactory::create_dyn_any_from_type_code メソッドを使用して作成します。また，DynAny オブジェクトは，DynAnyFactory::create_dyn_any メソッドを使用して Any オブジェクトから作成，および初期化することもできます。これらのメソッドについては，「22. コアインタフェースとクラス（C++）」を参照してください。

次のインタフェースは DynAny から派生したもので，動的に管理される構造化型のサポートを提供します。

- Array
  「23.6 DynArray」を参照してください。
- Enumeration
  「23.7 DynEnum」を参照してください。
- Sequence
  「23.8 DynSequence」を参照してください。
- Structure
  「23.9 DynStruct」を参照してください。
- Union
  「23.10 DynUnion」を参照してください。

23.4.1 インクルードファイル

このクラスを使用するときは，dynany.h ファイルをインクルードしてください。

23.4.2 注意事項

DynAny オブジェクトは，オペレーション要求，および DII リクエストのパラメタとして使用できません。また，ORB::object_to_string メソッドで外部化することもできません。ただし，DynAny::to_any メソッドを使用して DynAny オブジェクトを Any オブジェクトに変換すれば，パラメタとして使用できるようになります。

23.4.3 DynAny のメソッド

- void **assign**(
  DynamicAny::DynAny_ptr **dyn_any**);
  このメソッドは，指定した DynAny から，このオブジェクト内の値を初期化します。Any に含まれる型がこのオブジェクトに含まれる型と一致しなかった場合，TypeMismatch 例外が発生します。
- DynamicAny::DynAny_ptr **copy**();
  このメソッドは，このオブジェクトのコピーを返します。
- virtual CORBA::ULong **component_count**();
  このメソッドは，DynAny に格納された複合型コンポーネントの数を unsigned long として返します。
- virtual DynamicAny::DynAny_ptr **current_component**();
  このメソッドは，このオブジェクト内のカレントコンポーネントを返します。
- virtual void **destroy**();
  このメソッドは，このオブジェクトをデストラクトします。
- virtual CORBA::Boolean **equal**(
  const DynamicAny::DynAny_ptr **value**);
  このメソッドは，二つの DynAny 値が等しいかどうか比較します。等しい場合は TRUE を，等しくない場合は FALSE を返します。
- virtual void **from_any**(
  CORBA::Any& **value**);
  このメソッドは，指定した Any オブジェクトから，このオブジェクトのカレントコンポーネントを初期化します。
  Any に含まれる値の TypeCode が，このオブジェクトの作成時に定義した TypeCode と一致しない場合，TypeMismatch 例外が発生します。また，渡された value パラメタが不正の場合，InvalidValue 例外が発生します。
  - value
    このオブジェクトに設定する値を格納する Any オブジェクト
- virtual boolean **next**();
  このメソッドは，次のコンポーネントがあれば，制御をそこへ進め，TRUE を返します。次のコンポーネントがなければ FALSE を返します。

- virtual void rewind();
  このメソッドは，このオブジェクトのカレントコンポーネントを，DynAny で定義された先頭のコンポーネントに設定します。
  オブジェクトにコンポーネントが一つしかなかった場合，このメソッドは何もしません。
- virtual CORBA::Boolean seek(
  CORBA::Long index);
  このメソッドは，指定したインデックスのコンポーネントをカレントコンポーネントにします。指定したインデックスのコンポーネントがない場合，FALSE を返します。そうでない場合，TRUE を返します。
  - index
    目標コンポーネントのインデックス。インデックスは 0 から始まります。
- virtual CORBA::Any* to_any( );
  このメソッドは，DynAny オブジェクトを Any オブジェクトに変換し，Any オブジェクトを指すポインタを返します。
- CORBA::TypeCode_ptr type();
  このメソッドは，DynAny が格納する値の TypeCode を返します。

23.4.4 DynAny の抽出メソッド

抽出メソッドは，DynAny オブジェクトのカレントコンポーネントが格納する型を返すメソッドの集まりです。コードサンプル 23-2 に各抽出メソッドの名前を示します。

使用した抽出メソッドの返す型が，DynAny が格納する値と一致しない場合，TypeMismatch 例外が発生します。

コードサンプル 23-2　DynAny クラスが提供する抽出メソッド

```
virtual CORBA::Any* get_any();
virtual CORBA::Boolean get_boolean();
virtual CORBA::Char get_char();
virtual CORBA::Double get_double();
virtual DynamicAny::DynAny* get_dyn_any();
virtual CORBA::Float get_float();
virtual CORBA::Long get_LongLong();
virtual CORBA::Long get_longlong();
virtual CORBA::Octet get_octet();
virtual CORBA::Object_ptr get_reference();
virtual CORBA::Short get_short();
virtual char* get_string();
virtual CORBA::TypeCode_ptr get_typecode();
virtual CORBA::ULong get_ulong();
virtual CORBA::ULongLong get_ulonglong();
virtual CORBA::UShort get_ushort();
virtual CORBA::ValueBase* get_val();
virtual CORBA::WChar get_wchar();
```

```
virtual CORBA::WChar* get_wstring();

 (UNIX)
virtual CORBA::LongDouble get_longdouble();
```

23.4.5　DynAny の挿入メソッド

挿入メソッドは，特定の型の値を，DynAny オブジェクトのカレントコンポーネントにコピーするメソッドの集まりです。コードサンプル 23-3 に，さまざまな型の挿入例を示します。

挿入したオブジェクトの型が，DynAny オブジェクトの型と一致しない場合，InvalidValue 例外を発行します。

コードサンプル 23-3　DynAny クラスが提供する挿入メソッド

```
virtual void insert_any(const CORBA:Any& value);
virtual void insert_boolean(CORBA::Boolean value);
virtual void insert_char(CORBA::Char value);
virtual void insert_double(CORBA::Double value);
virtual void insert_dyn_any (DynamicAny::DynAny_ph_value);
virtual void insert_float(CORBA::Float value);
virtual void insert_long(CORBA::Long value);
virtual void insert_longlong(CORBA::LongLong value);
virtual void insert_octet(CORBA::Octet value);
virtual void insert_reference(CORBA:Object_ptr value);
virtual void insert_short(CORBA::Short value);
virtual void insert_string(const char* value);
virtual void insert_typecode(CORBA:TypeCode_ptr value);
virtual void insert_ulong(CORBA::ULong value);
virtual void insert_ulonglong(CORBA::ULongLong value);
virtual void insert_ushort(CORBA::UShort value);
virtual void insert_val(count CORBA::ValueBase& value);
virtual void insert_wchar(CORBA::WChar value);
virtual void insert_wstring(const CORBA::WChar* value);

 (UNIX)
virtual void insert_longdouble(CORBA::LongDouble value);
```

23.5 DynAnyFactory

```
class DynamicAny::DynAnyFactory : public CORBA::PseudoObject
```

DynAnyFactory オブジェクトは，新しい DynAny オブジェクトの生成に使用します。DynAnyFactory オブジェクトのリファレンスを取得するには，ORB::resolve_initial_references("DynAnyFactory")を呼び出します。

23.5.1 DynAnyFactory のメソッド

- DynamicAny::DynAny_ptr create_dyn_any(
  const CORBA::Any& value);
  このメソッドは，指定した値の DynAny オブジェクトを作成します。
  - value
    作成する DynAny オブジェクトの値
- DynamicAny::DynAny_ptr create_dyn_any_from_type_code(
  CORBA::TypeCode_ptr type);
  このメソッドは，指定した型の DynAny オブジェクトを作成します。
  - type
    作成する DynAny オブジェクトの型

23.6 DynArray

```
class DynamicAny::DynArray : public VISDynComplex
```

コンパイル時にデータ型が定義されなかった場合，実行時にクライアントアプリケーション，またはサーバがこのクラスのオブジェクトを使用して，配列データ型の作成と解釈をします。DynArray は，基本型（boolean，int，float など），または複合型（struct，union など）を格納できます。

DynAny に含まれる型は作成時に定義され，オブジェクトの存在期間内は変更できません。

DynAny から継承した next，rewind，seek，および current_component メソッドを使用して，各コンポーネント間を自由に行き来できます。

VISDynComplex クラスは helper クラスの一つで，このクラスによって VisiBroker ORB は複雑な DynAny の型を管理できます。

23.6.1 注意事項

DynAny オブジェクトは，オペレーション要求，および DII リクエストのパラメタとして使用できません。また，ORB::object_to_string メソッドで外部化することもできません。ただし，DynAny::to_any メソッドを使用して DynArray オブジェクトを Any オブジェクトのシーケンスに変換すれば，パラメタとして使用できるようになります。

23.6.2 DynArray のメソッド

- virtual void **destroy**();
  このメソッドは，このオブジェクトをデストラクトします。
- CORBA::AnySeq* **get_elements**();
  このメソッドは，このオブジェクトが格納する値を含む Any オブジェクトのシーケンスを返します。
- void **set_elements**(
  CORBA::AnySeq& **_value**);
  このメソッドは，value パラメタで指定したシーケンスの要素に，DynArray の要素を設定します。
- DynamicAny::DynAnySeq* **get_elements_as_dyn_any**();
  このメソッドは，DynAny の要素を DynAny シーケンスとして返します。
- void **set_elements_as_dyn_any**(
  const DynamicAny::DynAnySeq& **value**);
  このメソッドは，指定した DynAny シーケンスから，このオブジェクトに含まれる要素を設定します。

value 内の要素数がこの DynArray の要素数と一致しない場合，InvalidValue 例外が発生します。Any に含まれる値の TypeCode が，このオブジェクトの作成時に定義した TypeCode と一致しない場合，TypeMismatch 例外が発生します。

- value
  DynArray に値を設定する Any オブジェクトの配列

23.7 DynEnum

```
class DynamicAny::DynEnum : public DynamicAny::DynAny
```

コンパイル時に列挙体の値が定義されなかった場合，実行時にクライアントアプリケーション，またはサーバがこのクラスのオブジェクトを使用して，値の作成と解釈をします。

この型が含むコンポーネントは一つだけなので，DynEnum オブジェクトに対して DynAny::rewind メソッド，または DynAny::next メソッドを起動すると，常に FALSE を返します。

23.7.1 注意事項

DynEnum オブジェクトは，オペレーション要求，および DII リクエストのパラメタとして使用できません。また，ORB::object_to_string メソッドで外部化することもできません。ただし，to_any メソッドを使用して DynEnum オブジェクトを Any オブジェクトに変換すれば，パラメタとして使用できるようになります。

23.7.2 DynEnum のメソッド

- void **from_any**(
  const CORBA::Any& **value**);
  このメソッドは，指定した Any オブジェクトから，このオブジェクトの値を初期化します。
  Any に含まれる値の TypeCode が，このオブジェクトの作成時に定義した TypeCode と一致しない場合，Invalid 例外が発生します。
  - value
    Any オブジェクト
- CORBA::Any* **to_any**();
  カレントコンポーネントの値を格納する Any オブジェクトを返します。
- char* **get_as_string**();
  このメソッドは，DynEnum オブジェクトの値を文字列として返します。
- void **set_as_string**(
  const char* **value_as_string**);
  このメソッドは，指定した文字列に，DynEnum の値を設定します。
  - value_as_string
    この DynEnum に値を設定するために使用する文字列
- CORBA::ULong **get_as_ulong**();
  このメソッドは，DynEnum オブジェクトの値を格納する unsigned long を返します。

- void **set_as_ulong**(
  CORBA::ULong **value_as_ulong**);
  このメソッドは，指定した CORBA::Ulong に，DynEnum の値を設定します。
  - value_as_ulong
    DynEnum に値を設定するために使用する整数

23.8 DynSequence

```
class DynamicAny::DynSequence : public DynamicAny::DynArray
```

コンパイル時にシーケンスデータ型が定義されなかった場合，実行時にクライアントアプリケーション，またはサーバがこのクラスのオブジェクトを使用して，シーケンスデータ型の作成と解釈をします。DynSequence は，基本型（boolean，int，float など），または複合型（struct，union など）を格納できます。DynSequence に含まれる型は作成時に定義され，オブジェクトの存在期間内は変更できません。

next，rewind，seek，および current_component メソッドを使用して，各コンポーネント間を自由に行き来できます。

23.8.1 注意事項

DynSequence オブジェクトは，オペレーション要求，および DII リクエストのパラメタとして使用できません。また，ORB::object_to_string メソッドで外部化することもできません。ただし，to_any メソッドを使用して DynSequence オブジェクトを Any オブジェクトのシーケンスに変換すれば，パラメタとして使用できるようになります。

23.8.2 DynSequence のメソッド

- CORBA::ULong **get_length**();
  このメソッドは，この DynSequence が格納する要素数を返します。
- void **set_length**(
  CORBA::ULong **length**);
  このメソッドは，この DynSequence が格納できる要素数を設定します。
  カレント要素数より小さい値（length）を指定すると，指定した数までのシーケンスが格納されます。
  - length
    この DynSequence が格納するコンポーネントの数
- CORBA::AnySeq * **get_elements**();
  このメソッドは，このオブジェクトに格納された値を含む Any オブジェクトのシーケンスを返します。
- void **set_elements**(
  const AnySeq& **_value**);
  このメソッドは，指定された Any オブジェクトのシーケンスで，このオブジェクト内に要素を設定します。
- void **set_elements_as_dyn_any**(
  const DynamicAny::DynAnySeq& **value**);

このメソッドの詳細については，「23.6　DynArray」を参照してください。

- DynamicAny::DynAnySeq* get_elements_as_dyn_any();
  このメソッドの詳細については，「23.6　DynArray」を参照してください。

23.9 DynStruct

```
class DynamicAny::DynStruct : public VISDynComplex
```

コンパイル時に構造体が定義されなかった場合，実行時にクライアントアプリケーション，またはサーバがこのクラスのオブジェクトを使用して，作成と解釈をします。

next，rewind，seek，および current_component メソッドを使用して，各構造体メンバ間を自由に行き来できます。

DynStruct オブジェクトは，DynAnyFactory::create_dyn_any_from_type_code メソッドを起動することで作成されます。

23.9.1 注意事項

DynStruct オブジェクトは，オペレーション要求，および DII リクエストのパラメタとして使用できません。また，ORB::object_to_string メソッドで外部化することもできません。ただし，to_any メソッドを使用して DynStruct オブジェクトを Any オブジェクトに変換すれば，パラメタとして使用できるようになります。

23.9.2 DynStruct のメソッド

- void **destroy**();
  このメソッドは，このオブジェクトをデストラクトします。
- CORBA::FieldName **current_member_name**();
  このメソッドは，カレントコンポーネントのメンバ名を返します。
- CORBA::TCKind **current_member_kind**();
  このメソッドは，カレントコンポーネントに対応する TypeCode を返します。
- DynamicAny::NameValuePairSeq **get_members**();
  このメソッドは，この構造体のすべてのメンバを，NameValuePair オブジェクトのシーケンスとして返します。
- void **set_members**(
  const DynamicAny::NameValuePairSeq& **value**);
  このメソッドは，NameValuePair オブジェクトの配列から，構造体メンバを設定します。
- DynamicAny::Name DynAnyPairSeq **get_members_as_dyn_any**();
  このメソッドは，構造体メンバを，NameDynAnyPair シーケンスとして返します。
- void **set_members_as_dyn_any**(

const DynamicAny::nameDynAnyPairSeq value);

このメソッドは，NameDynAnyPair オブジェクトから，構造体メンバを設定します。

value のシーケンスの長さと，この DynStruct オブジェクトのメンバ数が一致しない場合，InvalidValue 例外が発生します。要素のタイプコードに構造体のタイプコードと一致しないものが一つでもある場合，TypeMismatch 例外が発生します。

23.10 DynUnion

```
class DynamicAny::DynUnion : public VISDynComplex
```

コンパイル時に union が定義されなかった場合，実行時にクライアントアプリケーション，またはサーバがこのインタフェースを使用して，union の作成と解釈をします。DynUnion には，union の識別子と実メンバの二つの要素のシーケンスがあります。

next，rewind，seek，および current_component メソッドを使用して，各コンポーネント間を自由に行き来できます。

DynUnion オブジェクトは，DynamicAny::DynAnyFactory::create_dyn_any_from_type_code メソッドを起動し，union 型を引数として渡すことで作成されます。

23.10.1 注意事項

DynUnion オブジェクトは，オペレーション要求，および DII リクエストのパラメタとして使用できません。また，ORB::object_to_string メソッドで外部化することもできません。ただし，DynAny::to_any メソッドを使用して DynUnion オブジェクトを Any オブジェクトに変換すれば，パラメタとして使用できるようになります。

23.10.2 DynUnion のメソッド

- DynamicAny::DynAny_ptr **get_discriminator**();
  このメソッドは，union の識別子を格納する DynAny オブジェクトを返します。
- CORBA::TCKind **discriminator_kind**();
  このメソッドは，union の識別子のタイプコードを返します。
- DynamicAny::DynAny_ptr **member**();
  このメソッドは，union のメンバであるカレントコンポーネントの DynAny オブジェクトを返します。
- CORBA::TCKind **member_kind**();
  このメソッドは，union のメンバであるカレントコンポーネントのタイプコードを返します。
- CORBA::FieldName **member_name**();
  このメソッドは，カレントコンポーネントのメンバ名を返します。
- void **set_discriminator**(
  DynamicAny::DynAny_ptr **value**);
  このメソッドは，DynUnion のディスクリミネータを，指定した値に設定します。
- void **set_to_default_member**();

このメソッドは，ディスクリミネータを，union のデフォルト値と一致する値に設定します。

- void set_to_no_active_member();
  このメソッドは，ディスクリミネータを，どの union のケースラベルにも対応しない値に設定します。
- boolean has_no_active_member();
  このメソッドは，union にアクティブなメンバがない場合，つまり，ディスクリミネータの値が明示的なケースラベルとしてリストされていないため，union の値がディスクリミネータだけで構成されている場合，TRUE を返します。

23.11 Environment

```
class CORBA::Environment
```

Environment クラスは，C++言語例外がサポートされていないプラットフォーム上でシステム例外，およびユーザ例外の両方にアクセスし，それらの例外を報告するために使います。ユーザ例外がオブジェクトのメソッドによって発生したものであるとインタフェースが指定する場合，Environment クラスがそのメソッドの明示的なパラメタとなります。インタフェースが例外を発生させない場合，Environment クラスは，暗黙的なパラメタとなり，システム例外を報告するだけの目的で使用されます。クライアントからスタブへ Environment オブジェクトが渡されない場合，各オブジェクトの Environment のデフォルトが使用されます。

マルチスレッドアプリケーションには，各スレッドに対してグローバル Environment オブジェクトが生成されます。マルチスレッドではないアプリケーションは，グローバル Environment オブジェクトが一つしかありません。

23.11.1 インクルードファイル

このクラスを使用するときは，corba.h ファイルをインクルードしてください。

23.11.2 Environment のメソッド

- void ORB::create_environment(
  CORBA::Environment_ptr& ptr);
  このメソッドは，新しい Environment オブジェクトの生成に使用できます。

  注

  このメソッドは CORBA で規定されているので提供されていますが，Environment クラス用のコンストラクタ，または C++言語の new 演算子を使用した方が便利な場合があります。

  - ptr
    新しく生成されたオブジェクトを指すように設定されたポインタ
- Environment();
  このメソッドは，Environment オブジェクトを生成するコンストラクタです。このメソッドの呼び出しは，ORB::create_environment メソッドの呼び出しに相当します。
- static CORBA::Environment& current_environment();
  この静的メソッドは，アプリケーションプロセスのグローバル Environment オブジェクトに対するリファレンスを返します。マルチスレッドのアプリケーションでは，スレッドに対するグローバル Environment オブジェクトが返されます。
- void exception(

CORBA::Exception *exp);

このメソッドは，引数として渡された Exception オブジェクトを記録します。指定されたオブジェクトが Exception オブジェクトの所有権を引き受け，Environment 自体が削除される際に，その所有権を破棄します。したがって，Exception オブジェクトは動的に割り当てられる必要があります。このメソッドに NULL ポインタを渡すことは，Environment で clear メソッドを呼び出すことに相当します。

- exp

  Environment で記録される Exception オブジェクトを指すポインタ。この Exception オブジェクトは動的に割り当てられます。

- CORBA::Exception *exception() const;

  このメソッドは，Environment に現在記録されている Exception を指すポインタを返します。この呼び出しによって返された Exception ポインタに対して delete を呼び出さないでください。記録されている Exception がない場合，NULL ポインタが返されます。

- void clear();

  このメソッドを使って，Environment が保持する Exception オブジェクトを Environment 自体に削除させます。オブジェクトが例外を持たない場合，このメソッドは有効にはなりません。

23.12 ExceptionList

```
class CORBA::ExceptionList
```

このクラスは，オペレーション要求によって発生する可能性のある例外を表すタイプコードのリストを含みます。「23.15 Request」を参照してください。

23.12.1 ExceptionList のメソッド

- CORBA::ExceptionList();
  このメソッドは，空の例外リストを生成するコンストラクタです。
- CORBA::ExceptionList(
  CORBA::ExceptionList& list);
  このメソッドはコピーコンストラクタです。
  - list
    コピーされるリスト
- ~CORBA::ExceptionList();
  このメソッドは，デフォルトデストラクタです。
- void add(
  CORBA::TypeCode_ptr tc);
  このメソッドは，オブジェクトのリストに指定された例外のタイプコードを追加します。
  - tc
    リストに追加される例外のタイプコード
- void add_consume(
  CORBA::TypeCode_ptr tc);
  このメソッドは，指定された例外のタイプコードをこのオブジェクトのリストに追加します。渡された引数の所有権は，ExceptionList が引き受けます。このメソッドを呼び出したあとに，渡された引数へのアクセス，およびその解放はしないでください。
  - tc
    リストに追加される例外のタイプコード
- CORBA::ULong count() const;
  このメソッドは，リストに現在格納されている項目の数を返します。
- CORBA::TypeCode_ptr item(
  CORBA::Long index);
  このメソッドは，指定されたインデックスのリストに格納されている TypeCode を指すポインタを返します。インデックスが不当な場合，NULL ポインタが返されます。このメソッドを呼び出したあと

に，渡された引数へアクセスしたり，その引数を解放したりできません。リストから TypeCode を削除するには，remove メソッドを使用します。

- index
  返されるタイプコードのインデックス。インデックスは 0 から始まります。

- void remove(
  CORBA::long index);
  このメソッドは，指定されたインデックスを持つ TypeCode をリストから削除します。インデックスが不当な場合は，削除されません。
  - index
    削除するタイプコードのインデックス。インデックスは 0 から始まります。

- static CORBA::ExceptionList_ptr _duplicate(
  CORBA::ExceptionList_ptr ptr);
  この静的メソッドは，指定されたオブジェクトのリファレンスカウントを増やし，そのオブジェクトを指すポインタを返します。
  - ptr
    複製されるオブジェクト

- static CORBA::ExceptionList_ptr _nil();
  この静的メソッドは，初期化に使用できる NULL ポインタを返します。

- static void _release(
  CORBA::ExceptionList *ptr);
  この静的メソッドは，指定されたオブジェクトのリファレンスカウントを減らします。そのカウントが 0 になると，オブジェクトによって管理されていたすべてのメモリが解放され，そのオブジェクトが削除されます。
  - ptr
    解放するオブジェクト

23.13 NamedValue

```
class CORBA::NamedValue
```

NamedValue クラスは，動的起動インタフェースリクエストのパラメタ，またはリターン値として使用される名前・値ペアを表すためのクラスです。このクラスのオブジェクトは，NVList にグループ化されます。名前・値ペアの値は，Any クラスを使って表します。NVList クラスの詳細については，「23.14 NVList」を，Request クラスの詳細については，「23.15 Request」を参照してください。

23.13.1 インクルードファイル

このクラスを使用するときは，corba.h ファイルをインクルードしてください。

23.13.2 NamedValue のメソッド

- CORBA::Flags **flags**() const;

  このメソッドは，名前・値ペアがどのように使用されるかを定義するフラグを返します。次のどれかが返されます。

  ARG_IN
  : in パラメタとして使用します。

  ARG_OUT
  : out パラメタとして使用します。

  ARG_INOUT
  : inout パラメタとして使用します。

  IN_COPY_VALUE
  : このフラグと，ARG_INOUT フラグを組み合わせると，ORB が out パラメタのコピーを作成するように指定できます。これによってクライアントアプリケーションのメモリに影響を及ぼすことなく，そのパラメタのメモリを解放できます。

- const char ***name**() const;

  このメソッドは，このオブジェクトの名前・値ペアの名前の部分だけを返します。返された引数がポイントしているメモリを絶対に解放しないでください。

- CORBA::Any ***value**() const;

  このメソッドは，このオブジェクトの名前・値ペアの値の部分だけを返します。返された引数がポイントしているメモリを絶対に解放しないでください。

- static CORBA::NamedValue_ptr **_duplicate**(
  CORBA::NamedValue_ptr **ptr**);

この静的メソッドは，指定されたオブジェクトのリファレンスカウントを増やし，そのオブジェクトを指すポインタを返します。

- ptr
  複製されるオブジェクト

- static CORBA::NamedValue_ptr _nil();
  この静的メソッドは，CORBA::NamedValue_ptr の初期化に使用できる NULL ポインタを返します。
- static void _release(
  CORBA::NamedValue *ptr);
  この静的メソッドは，指定されたオブジェクトのリファレンスカウントを減らします。そのカウントが 0 になると，オブジェクトによって管理されていたすべてのメモリが解放され，そのオブジェクトが削除されます。
  - ptr
    解放するオブジェクト

23.14 NVList

```
class CORBA::NVList
```

NVList クラスは，NamedValue オブジェクトのリストを含ませるために使用されます。また，動的起動インタフェースリクエストに対応するパラメタを渡すために使用されます。NamedValue オブジェクトについては，「23.13 NamedValue」を，Request クラスについては，「23.15 Request」を参照してください。

リストに項目を追加する際に使用できるメソッドが提供されています。返された引数がポイントしているメモリを絶対に解放しないでください。リストからの項目を削除する際には，必ず remove メソッドを使用してください。

23.14.1 インクルードファイル

このクラスを使用するときは，corba.h ファイルをインクルードしてください。

23.14.2 NVList のメソッド

- CORBA::NamedValue_ptr **add**(
  CORBA::Flags **flags**);
  このメソッドは，フラグだけを初期化し，NamedValue オブジェクトをリストに追加します。追加されたオブジェクトの名前，またはオブジェクトの値は初期化されません。NamedValue の名前および値の属性を初期化するために，返されたポインタを使用できます。返された引数がポイントしているメモリを絶対に解放しないでください。
  - flags
    NamedValue オブジェクトの用途を表すフラグ。このフラグは，ARG_IN，ARG_OUT，ARG_INOUT のどれかになります。
- CORBA::NamedValue_ptr **add_item**(
  const char ***name**, CORBA::Flags **flag**);
  このメソッドは，オブジェクトのフラグまたは名前の属性を初期化して，NamedValue オブジェクトをリストに追加します。NamedValue の値の属性を初期化するために，返されたポインタを使用できます。
  返された引数がポイントしているメモリを絶対に解放しないでください。
  - name
    名前
  - flag

NamedValue オブジェクトの用途を表すフラグ。このフラグは，ARG_IN，ARG_OUT，ARG_INOUT のどれかになります。

- NamedValue_ptr **add_item_consume**(
  char ***nm**, CORBA::Flags **flag**);
  nm が示すメモリの管理が NVList によって引き継がれるという点を除き，このメソッドは add_item メソッドと同じです。そのリストがすでに nm のコピーや解放をしている可能性があるので，このメソッドを呼び出したあとは，nm にアクセスできません。この項目を削除すると，自動的にそのメモリが解放されます。
  このメソッドのリターン値のメモリを絶対に解放しないでください。
  - nm
    名前
  - flag
    NamedValue オブジェクトの用途を表すフラグ。このフラグは，ARG_IN，ARG_OUT，ARG_INOUT のどれかになります。
- CORBA::NamedValue_ptr **add_value**(
  const char ***name**,
  const CORBA::Any ***value**,
  CORBA::Flags **flag**);
  このメソッドは，オブジェクトのフラグ，名前，および値を初期化して，リストに NamedValue を追加します。NamedValue オブジェクトを指すポインタが返されます。
  返された引数がポイントしているメモリを絶対に解放しないでください。
  - name
    名前
  - value
    値
  - flag
    NamedValue オブジェクトの用途を表すフラグ。このフラグは，ARG_IN，ARG_OUT，ARG_INOUT のどれかになります。
- NamedValue_ptr **add_value_consume**(
  char ***nm**,
  CORBA::Any ***value**,
  CORBA::Flags **flag**);
  このメソッドは，nm または value が示すメモリの管理を NVList が引き継ぐという点を除き，add_value メソッドと同じです。そのリストがすでに nm のコピーや解放をしている可能性があるので，このメソッドを呼び出したあとは，nm または value にアクセスできません。この項目を削除すると，自動的にそのメモリが解放されます。
  - nm

名前

- value
  値
- flag
  NamedValue オブジェクトの用途を表すフラグ。このフラグは，ARG_IN，ARG_OUT，ARG_INOUT のどれかになります。

- CORBA::Long **count**() const;
  このメソッドは，リスト内にある NamedValue オブジェクトの数を返します。
- static CORBA::Boolean **CORBA::is_nil**(
  NVList_ptr **obj**);
  指定された NamedValue ポインタが NULL の場合，このメソッドは TRUE を返します。
  - obj
    チェックされるオブジェクトを指すポインタ
- NamedValue_ptr **item**(
  CORBA::Long **index**);
  このメソッドは，指定されたインデックスを持つリストの NamedValue を返します。
  返された引数がポイントしているメモリを絶対に解放しないでください。
  - index
    NamedValue オブジェクトのインデックス。インデックスは 0 から始まります。
- static void **CORBA::release**(
  CORBA::NVList_ptr **obj**);
  この静的メソッドは指定されたオブジェクトを解放します。
  - obj
    解放されるオブジェクト
- Status **remove**(
  CORBA::Long **index**);
  このメソッドは，指定されたインデックスにあるリストから NamedValue オブジェクトを削除します。add_item_consume または add_value_consume メソッドを使用して追加されたリストの項目が格納されているメモリは，その項目が削除される前に解放されます。
  - index
    NamedValue オブジェクトのインデックス。インデックスは 0 から始まります。
- static CORBA::NVList_ptr **_duplicate**(
  CORBA::NVList_ptr **ptr**);
  この静的メソッドは，指定されたオブジェクトのリファレンスカウントを増やし，そのオブジェクトを指すポインタを返します。

- ptr
  複製されるオブジェクト

- static CORBA::NVList_ptr _nil();
  この静的メソッドは，NV_List ポインタの初期化に使用できる NULL ポインタを返します。例を次に示します。

```
CORBA::NV_List_ptr p = CORBA::NVList::_nil();
```

- static void _release(
  CORBA::NVList *ptr);
  この静的メソッドは，指定されたオブジェクトのリファレンスカウントを減らします。そのカウントが 0 になると，オブジェクトによって管理されていたすべてのメモリが解放され，そのオブジェクトが削除されます。

  - ptr
    解放されるオブジェクト

23.15 Request

```
class CORBA::Request
```

Request クラスは，動的起動インタフェースを使って ORB オブジェクトに対するオペレーションを呼び出す際に，クライアントアプリケーションによって使用されます。一つの Request オブジェクトに対して一つの ORB オブジェクトが対応づけられます。Request は，ORB オブジェクトに対して実行されるオペレーションを表します。Request には，渡される必要のある引数，Context，および Environment オブジェクトが含まれます。また，リクエストを呼び出したり，オブジェクトインプリメンテーションからの応答を受信したり，オペレーションの結果を取得したりするためのメソッドか提供されています。

「22.8.2 CORBA::Object のメソッド」の Object::_create_request を使って，Request オブジェクトを生成できます。

Request オブジェクトがすべての返されたパラメタの所有権を保持することに注意してください。それらのパラメタを絶対に解放しないでください。

23.15.1 インクルードファイル

このクラスを使用するときは，corba.h ファイルをインクルードしてください。

23.15.2 Request のメソッド

- CORBA::Any& add_in_arg();
  このメソッドは，名前が付けられていない in 引数を Request に追加し，その Any オブジェクトのリファレンスを返すことによって，その in 引数の名前，型，および値を設定できるようにします。
- CORBA::Any& add_in_arg(
  const char *name);
  このメソッドは，名前が付けられている in 引数を Request に追加し，その Any オブジェクトのリファレンスを返すことによって，その in 引数の型，および値を設定できるようにします。
  このメソッドのリターン値のメモリを絶対に解放しないでください。
  - name
    追加される in 引数の名前
- CORBA::Any& add_inout_arg();
  このメソッドは，名前が付けられていない inout 引数を Request に追加し，その Any オブジェクトのリファレンスを返すことによって，その inout 引数の名前，型，および値を設定できるようにします。
- CORBA::Any& add_inout_arg(
  const char *name);

このメソッドは，名前が付けられている inout 引数を Request に追加し，その Any オブジェクトのリファレンスを返すことによって，その inout 引数の型，および値を設定できるようにします。

- name
  追加される inout 引数の名前

- CORBA::Any& **add_out_arg**();
  このメソッドは，名前が付けられていない out 引数を Request に追加し，その Any オブジェクトのリファレンスを返すことによって，その out 引数の名前，型，および値を設定できるようにします。
- CORBA::Any& **add_out_arg**(
  const char ***name**);
  このメソッドは，名前が付けられている out 引数を Request に追加し，その Any オブジェクトのリファレンスを返すことによって，その out 引数の型，および値を設定できるようにします。
  - name
    追加される out 引数の名前
- CORBA::NVList_ptr **arguments**();
  このメソッドは，このリクエストの引数を含んでいる NVList オブジェクトを指すポインタを返します。引数の値を設定したり取得したりする際に，このポインタを使えます。NVList の詳細については，「23.14　NVList」を参照してください。
  このメソッドのリターン値のメモリを絶対に解放しないでください。
- CORBA::ContextList_ptr **contexts**();
  このメソッドは，Request に対応するすべての Context オブジェクトのリストを指すポインタを返します。Context クラスの詳細については，「22.5　Context」を参照してください。
  このメソッドのリターン値のメモリを絶対に解放しないでください。
- CORBA::Context_ptr **ctx**() const;
  このメソッドは，このリクエストに対応する Context を指すポインタを返します。
- void **ctx**(
  CORBA::Context_ptr **ctx**);
  このメソッドは，このリクエストで使用される Context を設定します。Context クラスの詳細については，「22.5　Context」を参照してください。
  - ctx
    このリクエストに対応づけられる Context オブジェクト
- CORBA::Environment_ptr **env**();
  このメソッドは，このリクエストに対応する Environment を指すポインタを返します。Environment クラスの詳細については，「23.11　Environment」を参照してください。
- CORBA::ExceptionList_ptr **exceptions**();
  このメソッドは，このリクエストによって発生されるすべての例外のリストを指すポインタを返します。
  このメソッドのリターン値のメモリを絶対に解放しないでください。

- void **get_response**();
  オブジェクトインプリメンテーションからの応答を取得するために，send_deferred が呼び出されたあとに，このメソッドを使用します。応答がない場合，このメソッドは，クライアントアプリケーションが応答を受信するまで待ちます。
- void **invoke**();
  このメソッドは，このリクエストに対応する ORB オブジェクトに対して Request を呼び出します。このメソッドは，クライアントが，オブジェクトインプリメンテーションからの応答を受信するまで待ちます。このメソッドを呼び出す前に，Request をターゲットオブジェクト，オペレーション名，および引数で初期化します。
- const char* **operation**() const;
  このメソッドは，このリクエストが表すオペレーションの名前を返します。
- CORBA::Boolean **poll_response**();
  応答を待たないこのメソッドは，send_deferred メソッドが，応答を受信したかを判別したあとに呼び出されます。応答が受信されている場合，このメソッドは TRUE を返し，そうでない場合は FALSE を返します。
- CORBA::NamedValue_ptr **result**();
  このメソッドは，オペレーションのリターン値が格納される NamedValue オブジェクトを指すポインタを返します。このポインタは，オブジェクトインプリメンテーションによって，リクエストが処理されたあとの結果値を参照するのに使用できます。NamedValue クラスの詳細については，「23.13 NamedValue」を参照してください。
- CORBA::Any& **return_value**();
  このメソッドは，この Request オブジェクトのリターン値を表す Any オブジェクトのリファレンスを返します。
- void **set_return_type**(
  CORBA::TypeCode_ptr **tc**);
  このメソッドは，受け取る TypeCode のリターン値を設定します。invoke メソッドまたは send メソッドのどちらかを呼び出す前に，リターン値の型を設定する必要があります。
  - tc
    リターン値の型
- void **send_deferred**();
  このメソッドは，invoke メソッドと同様に，この Request をオブジェクトインプリメンテーションへ送信します。しかし，invoke メソッドとは異なり，このメソッドは応答を待ちません。クライアントアプリケーションは get_response メソッドを使って応答を取得できます。
- void **send_oneway**();
  このメソッドは，一方向オペレーションとしてこの Request を呼び出します。一方向オペレーションは，オブジェクトインプリメンテーションからクライアントアプリケーションへ送信される応答を待ちません。また，そのような応答を送信することもありません。

- CORBA::Object_ptr **target**() const;
  このメソッドは，このリクエストが実行されるターゲットオブジェクトのリファレンスを返します。
- static CORBA::Request_ptr **_duplicate**(
  CORBA::Request_ptr **ptr**);
  この静的メソッドは，指定されたオブジェクトのリファレンスカウントを増やし，そのオブジェクトを指すポインタを返します。
  - ptr
    複製されるポインタ
- static CORBA::Request_ptr **_nil**();
  この静的メソッドは，CORBA::Request_ptr オブジェクトの初期化に使用できる NULL ポインタを返します。
- static void **_release**(
  CORBA::Request ***ptr**);
  この静的メソッドは，指定されたオブジェクトのリファレンスカウントを減らします。そのカウントが 0 になると，オブジェクトによって管理されていたすべてのメモリが解放され，そのオブジェクトが削除されます。
  - ptr
    解放されるオブジェクト

23.16 ServerRequest

```
class CORBA::ServerRequest
```

ServerRequest クラスは，動的スケルトンインタフェースを使用するオブジェクトインプリメンテーションによって受信される，オペレーション要求を表します。POA は，クライアントオペレーション要求を受信すると，そのオブジェクトインプリメンテーションの invoke メソッドを呼び出し，この型のオブジェクトを渡します。

このクラスは，リクエストされたオペレーション，および引数を判別するオブジェクトインプリメンテーションにとって必要なメソッドを提供します。また，リターン値の設定，およびクライアントアプリケーションへの例外の反映のために必要なメソッドも提供します。

このクラスが返す値に対応するメモリを絶対に解放しないでください。

動的スケルトンの使用については，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「動的スケルトンインタフェースの使用」の記述を参照してください。

23.16.1 インクルードファイル

このクラスを使用するときは，corba.h ファイルをインクルードしてください。

23.16.2 ServerRequest のメソッド

- void **arguments**(
  CORBA::NVList_ptr **params**);
  このメソッドは，このリクエストのパラメタリストを設定します。
  - params
    項目を追加するパラメタリスト。このメソッドを呼び出す前に，適切な数の Any オブジェクトでこのリストを初期化し，それらの型とフラグの値を設定します。
- CORBA::Context_ptr **ctx**();
  このメソッドは，リクエストに対応する Context オブジェクトを返します。
  このメソッドのリターン値のメモリを絶対に解放しないでください。
- const char ***operation**() const;
  リクエストされたオペレーションの名前を返します。
- void **set_exception**(
  const CORBA::Any& **a**);
  このメソッドは，クライアントアプリケーションに反映される例外を設定します。

- a
  例外を表す Any オブジェクト

- void set_result(
  const CORBA::Any& a);
  このメソッドは，クライアントアプリケーションに反映される結果を設定します。
  - a
    リターン値を表す Any オブジェクト
- static CORBA::ServerRequest_ptr _duplicate(
  CORBA::ServerRequest_ptr ptr);
  この静的メソッドは，指定されたオブジェクトのリファレンスカウントを増やし，そのオブジェクトを指すポインタを返します。
  - ptr
    複製されるオブジェクト
- static CORBA::ServerRequest_ptr _nil();
  この静的メソッドは，初期化に使用できる NULL ポインタを返します。
- static void _release(
  CORBA::ServerRequest *ptr);
  この静的メソッドは，指定されたオブジェクトのリファレンスカウントを減らします。そのカウントが 0 になると，オブジェクトによって管理されていたすべてのメモリが解放され，そのオブジェクトが削除されます。
  - ptr
    解放されるオブジェクト

23.17 TCKind

```
enum TCKind
```

この列挙体は，TypeCode オブジェクトが表す，いろいろな型を記述しています。TypeCode オブジェクトについては，「23.18 TypeCode」を参照してください。その値を次の表に示します。

表 23–1 型の一覧（TCKind）（C++）

名前	意味
tk_abstract_interface	抽象インタフェース
tk_alias	エイリアス
tk_any	Any
tk_array	配列
tk_boolean	boolean
tk_char	文字
tk_double	double
tk_enum	enum
tk_except	例外
tk_fixed	fixed type
tk_float	float
tk_long	long
tk_longdouble	long double
tk_longlong	long long
tk_native	native type
tk_null	NULL
tk_objref	オブジェクトリファレンス
tk_octet	オクテット文字列
tk_Principal	Principal
tk_sequence	シーケンス
tk_short	short
tk_string	文字列
tk_struct	構造体
tk_TypeCode	TypeCode

名前	意味
tk_ulong	unsigned long
tk_ulonglong	unsigned long long
tk_union	union
tk_ushort	unsigned short
tk_value	value
tk_value_box	value box
tk_void	void
tk_wchar	Unicode 文字
tk_wstring	Unicode 文字列

23.18 TypeCode

```
class CORBA::TypeCode
```

TypeCode クラスは，IDL で定義できるさまざまな型を表します。タイプコードは，通常 Any オブジェクトに格納されている値の型を定義するために使います。詳細については，「23.1.2 Any のメソッド」の CORBA::Any();メソッドを参照してください。また，タイプコードはパラメタとしてメソッド呼び出しに渡すこともできます。

TypeCode オブジェクトは，CORBA::ORB::create_<type>_tc メソッド，またはここで提供されるコンストラクタを使って生成できます。CORBA::ORB::create_<type>_tc メソッドについては，「22.8 Object」を参照してください。

23.18.1 インクルードファイル

このクラスを使用するときは，corba.h ファイルをインクルードしてください。

23.18.2 TypeCode のコンストラクタ

- CORBA::TypeCode(
  CORBA::TCKind kind, CORBA::Boolean is_constant);
  このメソッドは，拡張パラメタを必要としない型の TypeCode オブジェクトを構築します。kind がこのコンストラクタに対して不当な型である場合，BAD_PARAM 例外が発生します。
  - kind
    表されるオブジェクトの型の記述。次のどれかになります。

```
CORBA::tk_null, CORBA::tk_void, CORBA::tk_short,
CORBA::tk_long, CORBA::tk_ushort, CORBA::tk_ulong,
CORBA::tk_float, CORBA::tk_double, CORBA::tk_boolean,
CORBA::tk_char, CORBA::tk_octet, CORBA::tk_any,
CORBA::tk_TypeCode, CORBA::tk_Principal, CORBA::tk_longlong,
CORBA::tk_ulonglong, CORBA::tk_longdouble, CORBA::tk_wchar,
CORBA::tk_fixed, CORBA::tk_value, CORBA::tk_value_box,
CORBA::native, CORBA::tk_abstract_interface.
```

  - is_constant
    TRUE である場合，表されている型が定数であると考えられます。TRUE でない場合，そのオブジェクトは定数ではありません。

23.18.3 TypeCode のメソッド

- CORBA::TypeCode_ptr content_type() const;

  このメソッドは，シーケンスまたは配列の要素の TypeCode を返します。また，エイリアスの型も返します。オブジェクトの種類が，CORBA::tk_sequence，CORBA::tk_array，または CORBA::tk_alias のどれでもない場合，BadKind 例外が発生します。

- CORBA::Long default_index() const;

  このメソッドは，union を表す TypeCode のデフォルトインデックスを返します。このオブジェクトの種類が CORBA::tk_union でない場合，BadKind 例外が発生します。

- CORBA::TypeCode_ptr discriminator_type() const;

  このメソッドは，union を表す TypeCode のディスクリミネータ型を返します。このオブジェクトの種類が，CORBA::tk_union でない場合，BadKind 例外が発生します。

- CORBA::Boolean equal(

  CORBA::TypeCode_ptr tc) const;

  このメソッドは，指定された TypeCode とこのオブジェクトを比較します。すべての面で一致する場合，TRUE が返されます。そうでない場合は，FALSE が返されます。

  - tc

    このオブジェクトと比較されるオブジェクト

- const char* id() const;

  このメソッドは，このオブジェクトによって表される型のリポジトリ ID を返します。その型がリポジトリ ID を持たない場合，BadKind 例外が発生します。リポジトリ ID を持つ型を次に示します。

```
CORBA::tk_alias
CORBA::tk_enum
CORBA::tk_except
CORBA::tk_objref
CORBA::tk_struct
CORBA::tk_union
```

- CORBA::TCKind kind() const;

  このメソッドは，オブジェクトの種類を返します。

- CORBA::ULong length() const;

  このメソッドは，このオブジェクトによって表される文字列，シーケンス，配列の長さを返します。オブジェクトの種類が CORBA::tk_string，CORBA::tk_sequence，または CORBA::tk_array のどれでもない場合，BadKind 例外が発生します。

- CORBA::ULong member_count() const;

  このメソッドは，このオブジェクトによって表される型のメンバカウントを返します。その型がメンバを持たない場合，BadKind 例外が発生します。メンバを持つ型を次に示します。

```
CORBA::tk_enum
CORBA::tk_except
```

```
CORBA::tk_struct
CORBA::tk_union
```

- CORBA::Any_ptr member_label(

  CORBA::ULong index) const;

  このメソッドは，union の TypeCode オブジェクトから，指定されたインデックスとともにメンバのラベルを返します。このオブジェクトの種類が CORBA::tk_union でない場合，BadKind 例外が発生します。また，インデックスが不当な場合，Bounds 例外が発生します。

  - index

    型が返される union メンバのラベル。インデックスは 0 から始まります。

- const char *member_name(

  CORBA::ULong index) const;

  このメソッドは，このオブジェクトによって表される型から，指定されたインデックスとともにメンバの名前を返します。その型がメンバを持たない場合，BadKind 例外が発生します。また，インデックスが不当な場合，Bounds 例外が発生します。メンバを持つ型を次に示します。

```
CORBA::tk_enum
CORBA::tk_except
CORBA::tk_struct
CORBA::tk_union
```

  - index

    名前が返されるメンバのインデックス。インデックスは 0 から始まります。

- CORBA::TypeCode_ptr member_type(

  CORBA::ULong index) const;

  このメソッドは，このオブジェクトによって表される型から，指定されたインデックスとともにメンバの型を返します。その型に型を持つメンバがない場合，BadKind 例外が発生します。また，インデックスが不当な場合，Bounds 例外が発生します。メンバを持つ型を次に示します。

```
CORBA::tk_except
CORBA::tk_union
```

  - index

    型が返されるメンバのインデックス。インデックスは 0 から始まります。

- const char *name() const;

  このメソッドは，このオブジェクトによって表される型の名前を返します。その型が名前を持たない場合，BadKind 例外が発生します。名前を持つ型を次に示します。

```
CORBA::tk_alias
CORBA::tk_enum
CORBA::tk_except
CORBA::tk_objref
CORBA::tk_struct
CORBA::tk_union
```

- static CORBA::TypeCode_ptr _duplicate(

CORBA::TypeCode_ptr obj);

この静的メソッドは，指定された TypeCode を複製します。

- obj
  複製されるオブジェクト

- static CORBA::TypeCode_ptr _nil();
  この静的メソッドは，初期化に使用できる NULL TypeCode ポインタを返します。
- static void _release(
  CORBA::TypeCode_ptr obj);
  この静的メソッドは，指定されたオブジェクトのリファレンスカウントを減らします。そのカウントが 0 になると，オブジェクトによって管理されていたすべてのメモリが解放され，そのオブジェクトが削除されます。
  - obj
    解放されるオブジェクト
- CORBA::Boolean equivalent(
  CORBA::TypeCode_ptr tc) const;
  equivalent オペレーションは，ORB が，IDL に格納された値の型等価を決定するときに使用します。
  - tc
    このオブジェクトと比較されるオブジェクト
- CORBA::TypeCode_ptr get_compact_typecode() const;
  get_compact_code オペレーションは，すべての任意の指定名フィールドとメンバ名フィールドを削除します。ただし，エイリアスのタイプコードは変更しません。
- virtual CORBA::Visibility member_visibility(
  CORBA::ULong index) const;
  このメソッドは，インデックスで識別される valuetype メンバの Visibility を返します。
  - index
    visibility のインデックス。インデックスは 0 から始まります。

  注

  member_visibility オペレーションは，valuebox（またはボックス化された value）ではなく，valuetype の TypeCode にだけ呼び出されます。
- virtual CORBA::ValueModifier type_modifier() const;
  type_modifier オペレーションは，ボックス化されていない valuetype の TypeCode にだけ呼び出されます。このメソッドは，ターゲット TypeCode が表す valuetype に適用される ValueModifier を返します。
- virtual CORBA::TypeCode_ptr concrete_base_type();
  concrete_base_type オペレーションは，ボックス化されていない valuetype の TypeCode にだけ呼び出されます。ターゲット TypeCode が表す値が concrete ベースの valuetype を持つ場合，このメ

ソッドは concrete ベースの TypeCode を返します。それ以外の場合，nil TypeCode リファレンスを返します。

24 インタフェースリポジトリインタフェースとクラス (C++)

この章では，C++言語でインタフェースリポジトリにアクセスするのに使用できるインタフェース，およびクラスについて説明します。インタフェースリポジトリは，モジュール，モジュールに含まれるインタフェース，オペレーション，属性，および定数についての情報を保持します。

24.1 AliasDef

```
class CORBA::AliasDef : public virtual CORBA::TypedefDef,
                        public virtual CORBA::Object
```

このクラスは，TypedefDef クラスからの派生であり，インタフェースリポジトリに格納されている typedef のエイリアスを表します。このクラスは，エイリアスが生成された元の typedef の IDLType を設定，および取得するメソッドを提供します。

TypedefDef クラスの詳細については「24.37　TypedefDef」を，IDLType クラスの詳細については「24.18　IDLType」を参照してください。

24.1.1 AliasDef のメソッド

- CORBA::IDLType original_type_def();
  このメソッドは，エイリアスであるこのオブジェクトが生成された元の typedef の IDLType を返します。
- void original_type_def(
  CORBA::IDLType_ptr val);
  このメソッドは，エイリアスであるこのオブジェクトが生成された元の typedef の IDLType を設定します。
  - val
    このエイリアスの IDLType を設定します。

24.2 ArrayDef

```
class CORBA::ArrayDef : public virtual CORBA::IDLType,
                        public virtual CORBA::Object
```

このクラスは，IDLType からの派生であり，インタフェースリポジトリに格納されている配列を表します。このクラスは，配列の要素の型および配列の長さを，設定および取得するメソッドを提供します。

24.2.1 ArrayDef のメソッド

- CORBA::TypeCode element_type();
  このメソッドは，配列の要素の TypeCode を返します。
- CORBA::IDLType_ptr element_type_def();
  このメソッドは，配列に格納されている要素の IDLType を返します。
- void element_type_def(
  CORBA::IDLType_ptr element_type_def);
  このメソッドは，配列に格納されている要素の IDLType を設定します。
  - element_type_def
    配列内の要素の IDLType
- CORBA::ULong length();
  このメソッドは，配列の要素数を返します。
- void length(
  CORBA::ULong length);
  このメソッドは，配列の要素数を設定します。
  - length
    配列の要素数

24.3 AttributeDef

```
class CORBA::AttributeDef : public virtual CORBA::Contained,
                            public virtual CORBA::Object
```

このクラスは，インタフェースリポジトリに格納されるインタフェース属性を表します。このクラスは，属性のモードである typedef の設定，および取得の際に使用するメソッドを提供します。また，属性の型を取得するメソッドも提供しています。

24.3.1 AttributeDef のメソッド

- CORBA::AttributeMode **mode**();

  このメソッドは，属性のモードを返します。戻り値は，CORBA::AttributeMode ATTR_READONLY（read-only 属性の場合），または CORBA::AttributeMode ATTR_NORMAL（read-write 属性の場合）のどちらかです。「24.5 AttributeMode」を参照してください。

- void **mode**(

  CORBA::AttributeMode **_val**);

  このメソッドは，属性のモードを設定します。

  - _val

    設定するモード

- CORBA::TypeCode_ptr **type**();

  このメソッドは，属性の型を表す TypeCode を返します。

- CORBA::IDLType_ptr **type_def**();

  このメソッドは，このオブジェクトの IDLType を返します。

- void **type_def**(

  CORBA::IDLType_ptr **type_def**);

  このメソッドは，このオブジェクトの IDLType を設定します。

  - type_def

    このオブジェクトの IDLType

24.4 AttributeDescription

```
struct CORBA::AttributeDescription
```

AttributeDescription 構造体は，インタフェースリポジトリに格納されている属性の情報を提供します。

24.4.1 AttributeDescription のメンバ

- CORBA::Identifier_var **name**
  属性の名前です。
- CORBA::RepositoryId_var **id**
  属性のリポジトリ ID です。
- CORBA::RepositoryId_var **defined_in**
  この属性が定義されるインタフェースのリポジトリ ID です。
- CORBA::String_var **version**
  属性のバージョンです。
- CORBA::TypeCode_var **type**
  属性の IDL 型です。
- CORBA::AttributeMode **mode**
  属性のモードです。

24.5 AttributeMode

```
enum CORBA::AttributeMode
```

この列挙体は，属性のモードが read-only（読み取り専用）または read-write（標準）のどちらであるかを表すために使用します。

24.5.1 AttributeMode の値

AttributeMode の値を次の表に示します。

表 24–1 AttributeMode の値（C++）

定数	説明
ATTR_NORMAL	read-write（標準）属性
ATTR_READONLY	read-only 属性

24.6 ConstantDef

```
class CORBA::ConstantDef : public virtual CORBA::Contained,
                           public virtual CORBA::Object
```

このクラスは，インタフェースリポジトリに格納されている定数定義を表します。このクラスは，定数の型，値，typedef を，設定および取得するメソッドを提供します。

24.6.1 ConstantDef のメソッド

- CORBA::TypeCode_ptr type();
  このメソッドは，オブジェクトの型を表す TypeCode を返します。
- CORBA::IDLType_ptr type_def();
  このメソッドは，このオブジェクトの IDLType を返します。
- void type_def(
  CORBA::IDLType_ptr type_def);
  このメソッドは，定数の IDLType を設定します。
  - type_def
    この定数の IDLType
- CORBA::Any *value();
  このメソッドは，このオブジェクトの値を表す Any オブジェクトを指すポインタを返します。
- void value(
  CORBA::Any& _val);
  このメソッドは，この定数の値を設定します。
  - _val
    このオブジェクトの値を表す Any オブジェクト

24.7 ConstantDescription

```
struct CORBA::ConstantDescription
```

ConstantDescription 構造体は，インタフェースリポジトリに格納されている定数の情報を提供します。

24.7.1 ConstantDescription のメンバ

- CORBA::Identifier_var **name**
  定数の名前です。
- CORBA::RepositoryId_var **id**
  定数のリポジトリ ID です。
- CORBA::RepositoryId_var **defined_in**
  定数が定義されたモジュールまたはインタフェースの名前です。
- CORBA::String_var **version**
  定数のバージョンです。
- CORBA::TypeCode_var **type**
  定数の IDL 型です。
- CORBA::Any **value**
  この定数の値です。

24.8 Contained

```
class CORBA::Contained : public virtual CORBA::IRObject,
                         public virtual CORBA::Object
```

Contained クラスは，それ自体がほかのインタフェースリポジトリオブジェクトに含まれている，すべてのインタフェースリポジトリオブジェクトを派生させるために使用します。このクラスは，次のメソッドを提供します。

- オブジェクトの名前とバージョンを設定および取得するメソッド
- このオブジェクトを含む Container を判別するメソッド
- オブジェクトの絶対名，リポジトリ，および記述を取得するメソッド
- Container 間でオブジェクトを移動させるメソッド

24.8.1 インクルードファイル

このクラスを使用するときは，corba.h ファイルをインクルードしてください。

```
interface Contained: IRObject {
    attribute RepositoryId id;
    attribute Identifier name;
    attribute String_var version;

    readonly attribute Container defined_in;
    readonly attribute ScopedName absolute_name;
    readonly attribute Repository containing_Repository;

    struct Description {
       DefinitionKind kind;
       any value;
    };
    Description describe();
    void move(
       in Container new_Container,
       in Identifier new_name,
       in String_var new_version
    );
};
```

24.8.2 Contained のメソッド

- char *absolute_name();

このメソッドは，Repository 内でこのオブジェクトを一意に識別できるようにするための絶対名を返します。オブジェクトの生成時に設定される defined_in 属性が Repository を参照する場合，その絶対名には，単に"::"という文字列をオブジェクト名の先頭に付けます。

- CORBA::Repository_ptr **containing_repository**();
  このオブジェクトを格納しているリポジトリを指すポインタを返します。
- CORBA::Container_ptr **defined_in**();
  このオブジェクトが定義されている Container を指すポインタを返します。
- Description* **describe**();
  オブジェクトの記述を返します。Description 構造体の詳細については，「24.11　Description」を参照してください。
- char ***id**();
  このオブジェクトのリポジトリ ID を返します。
- void **id**(
  const char ***id**);
  このオブジェクトを一意に識別するリポジトリ ID を設定します。
  - id
    このオブジェクトのリポジトリ ID
- char ***name**();
  このメソッドは，Container のスコープ内で一意に識別するオブジェクトの名前を返します。
- void **name**(
  const char * **name**);
  このメソッドは，含まれるオブジェクトの名前を設定します。
  - name
    オブジェクトの名前
- CORBA::String_var **version**();
  このメソッドは，同一の名前を持つほかのオブジェクトと見分けられるようにオブジェクトのバージョンを返します。
- void **version**(
  CORBA::String_var& **val**);
  このメソッドは，このオブジェクトのバージョンを設定します。
  - val
    オブジェクトのバージョン
- void **move**(
  CORBA::Container_ptr **new_container**,
  const char ***new_name**,

CORBA::String_var& new_version);

このメソッドは，このオブジェクトを現在の Container から new_container に移動します。

- new_container
  このオブジェクトの移動先の Container
- new_name
  オブジェクトの新しい名前
- new_version
  オブジェクトの新しいバージョン仕様

24.9 Container

```
class CORBA::Container : public virtual CORBA::IRObject,
                         public virtual CORBA::Object
```

Container クラスは，インタフェースリポジトリで包含階層を作成するために使用します。Container オブジェクトは，Contained クラスから派生したオブジェクト定義を保持します。また，Repository クラスを除いた，Container クラスから派生したすべてのオブジェクト定義は，Contained クラスを継承します。

Container は，orbtypes.h で定義される IDL 型の型を生成するメソッドを提供します。このとき，InterfaceDef，ModuleDef，および ConstantDef クラスは対象としますが，ValueMemberDef クラスは対象外です。生成された各定義の defined_in 属性は，このオブジェクトを示すように初期化されています。

24.9.1 インクルードファイル

このクラスを使用するときは，corba.h ファイルをインクルードしてください。

```
interface Container: IRObject {
    Contained lookup(in ScopedName search_name);
    ContainedSeq contents(
        in DefinitionKind limit_type,
        in boolean exclude_inherited
    );
    ContainedSeq lookup_name(
        in Identifier search_name,
        in long levels_to_search,
        in CORBA::DefinitionKind limit_type,
        in boolean exclude_inherited
    );
    struct Description {
        Contained Contained_object;
        DefinitionKind kind;
        any value;
    };
    typedef sequence<Description> DescriptionSeq;
    DescriptionSeq describe_contents(
        in DefinitionKind limit_type,
        in boolean exclude_inherited,
        in long max_returned_objs
    );
};
```

24.9.2 Container のメソッド

- CORBA::ContainedSeq * contents(

CORBA::DefinitionKind **limit_type**,

CORBA::Boolean **exclude_inherited**);

このメソッドは，直接 Container に含まれる，または Container へ継承される包含オブジェクト定義のリストを返します。このメソッドを使って Repository でのオブジェクト定義の階層を操作できます。Repository のモジュール群に含まれるすべてのオブジェクト定義が返され，次にそれら各モジュールに含まれるすべてのオブジェクト定義が返されます。

- limit_type
  返されるインタフェースオブジェクト型。dk_all を指定すると，すべての型のオブジェクトが返されます。
- exclude_inherited
  TRUE を設定した場合，継承されたオブジェクトは返されません。

- CORBA::AliasDef_ptr **create_alias**(

  const char * **id**,

  const char ***name**,

  const CORBA::String_var& **version**,

  CORBA::IDLType_ptr **original_type**);

  このメソッドは，指定された属性で AliasDef オブジェクトをこの Container 内に生成し，新しく生成されたそのオブジェクトを指すポインタを返します。

  - id
    エイリアスの ID
  - name
    エイリアスの名前
  - version
    エイリアスのバージョン
  - original_type
    エイリアスであるこのオブジェクトが生成された元のオブジェクトの型

- CORBA::ConstantDef_ptr **create_constant**(

  const char * **id**,

  const char ***name**,

  const CORBA::String_var& **version**,

  CORBA::IDLType_ptr **type**,

  const CORBA::Any& **value**);

  このメソッドは，指定された属性で ConstantDef オブジェクトをこの Container 内に生成し，新しく生成されたそのオブジェクトを指すポインタを返します。

  - id
    定数の ID

- name
  定数の名前
- version
  定数のバージョン
- type
  定数の値（次のパラメタで指定する）の型
- value
  定数の値

- CORBA::EnumDef_ptr **create_enum**(
  const char * **id**,
  const char ***name**,
  const CORBA::String_var& **version**,
  const CORBA::EnumMemberSeq& **members**);
  このメソッドは，指定された属性で EnumDef オブジェクトをこの Container 内に生成し，新しく生成されたそのオブジェクトを指すポインタを返します。
  - id
    列挙体の ID
  - name
    列挙体の名前
  - version
    列挙体のバージョン
  - members
    列挙体のフィールドのリスト
- CORBA::ExceptionDef_ptr **create_exception**(
  const char * **id**,
  const char ***name**,
  const CORBA::String_var& **version**,
  const CORBA::StructMemberSeq& **members**);
  このメソッドは，指定された属性で ExceptionDef オブジェクトをこの Container 内に生成し，新しく生成されたそのオブジェクトを指すポインタを返します。
  - id
    例外の ID
  - name
    例外の名前
  - version

例外のバージョン

- members
  構造体のフィールドのシーケンス

- CORBA::InterfaceDef_ptr **create_interface**(
  const char * **id**,
  const char ***name**,
  const CORBA::String_var& **version**,
  const CORBA::InterfaceDefSeq& **base_interfaces**);
  このメソッドは，指定された属性で InterfaceDef オブジェクトをこの Container 内に生成し，新しく生成されたそのオブジェクトを指すポインタを返します。
  - id
    インタフェースの ID
  - name
    インタフェースの名前
  - version
    インタフェースのバージョン
  - base_interfaces
    このインタフェースが継承したすべてのインタフェースのリスト
- CORBA::ModuleDef_ptr **create_module**(
  const char * **id**,
  const char ***name**,
  const CORBA::String_var& **version**);
  このメソッドは，指定された属性で ModuleDef オブジェクトをこの Container 内に生成し，新しく生成されたそのオブジェクトを指すポインタを返します。
  - id
    モジュールの ID
  - name
    モジュールの名前
  - version
    モジュールのバージョン
- CORBA::StructDef_ptr **create_struct**(
  const char * **id**,
  const char ***name**,
  const CORBA::String_var& **version**,
  const CORBA::StructMemberSeq& **members**);

このメソッドは，指定された属性で StructDef オブジェクトをこの Container 内に生成し，新しく生成されたそのオブジェクトを指すポインタを返します。

- id
  構造体の ID
- name
  構造体の名前
- version
  構造体のバージョン
- members
  構造体フィールドのシーケンス

- CORBA::UnionDef_ptr create_union(
  const char * id,
  const char *name,
  const CORBA::String_var& version,
  CORBA::IDLType_ptr discriminator_type,
  const CORBA::UnionMemberSeq& members);
  このメソッドは，指定された属性で UnionDef オブジェクトをこの Container 内に生成し，新しく生成されたそのオブジェクトを指すポインタを返します。
  - id
    union の ID
  - name
    union の名前
  - version
    union のバージョン
  - discriminator_type
    union の識別値の型
  - members
    union の各フィールドのシーケンス
- CORBA::Container::DescriptionSeq * describe_contents(
  CORBA::DefinitionKind limit_type,
  CORBA::Boolean exclude_inherited,
  CORBA::Long max_returned_objs);
  このメソッドは，この container に直接含まれているか，またはこの container に継承されているすべての定義に関する記述を返します。
  - limit_type

記述が返されるインタフェースオブジェクトの型。dk_all を指定すると，すべての型のオブジェクトの記述が返されます。

- exclude_inherited
  true を設定した場合，継承されたオブジェクトの記述は返されません。
- max_returned_objs
  返される記述の最大数。このパラメタに-1 を設定すると，すべてのオブジェクトが返されます。

- CORBA::Contained_ptr **lookup**(
  const char *search_name);
  このメソッドは，指定された範囲名で，この Container と相対的な定義を探します。先頭が"::"で始まる絶対範囲名を指定すれば，囲みリポジトリ内の定義を探せます。オブジェクトが見つからない場合，NULL 値が返されます
  - search_name
    オブジェクトのインタフェース名
- CORBA::ContainedSeq * **lookup_name**(
  const char *search_name,
  CORBA::Long levels_to_search,
  CORBA::DefinitionKind limit_type,
  CORBA::Boolean exclude_inherited);
  このメソッドは，ある特定のオブジェクト内でオブジェクトを名前で探します。検索対象の階層内のレベル数，オブジェクトの型，継承されたオブジェクトを返すかどうか，などで検索を制限できます。
  - search_name
    含まれているオブジェクトの名前
  - levels_to_search
    検索対象の階層内のレベル数。このパラメタに-1 を設定すると，すべてのレベルが検索対象となります。このパラメタに 1 を設定すると，このオブジェクトだけを検索します。
  - limit_type
    返されるインタフェースオブジェクトの型。dk_all を指定すると，すべての型のオブジェクトが返されます。
  - exclude_inherited
    true を設定した場合，継承されたオブジェクトは返されません。
- CORBA::ValueDef_ptr **create_value**(
  const char * id,
  const char *name,
  const char version,
  CORBA::boolean is_custom,
  CORBA::boolean is_abstract,

const CORBA::ValueDef_ptr _base_value,
CORBA::boolean is_truncatable,
const CORBA::ValueDefSeq& abstract_base_values,
const CORBA::InterfaceDefSeq& supported _interfaces,
const CORBA::InitializerSeq& initializers);
このメソッドは，指定された属性で Container 内に ValueDef オブジェクトを生成し，生成されたオブジェクトを指すポインタを返します。

- id
  構造体のリポジトリ ID
- name
  構造体の名前
- version
  構造体のバージョン
- is_custom
  true を設定した場合，custom 型の valuetype を生成します。
- is_abstract
  true を設定した場合，abstract 型の valuetype を生成します。
- _base_value
  サポートされているベース値のリスト
- is_truncatable
  true を設定した場合，切り捨てができる valuetype を生成します。
- abstract_base_values
  サポートされている abstract 型のベース値のリスト
- supported _interfaces
  サポートされているインタフェースのリスト
- initializers
  この valuetype がサポートするイニシャライザのリスト

- CORBA::ValueBoxDef_ptr create_value_box(
  const char* id,
  const char* name,
  const char* version,
  CORBA::IDLType_ptr original_type);
  このメソッドは，指定された属性で Container 内に ValueBoxDef オブジェクトを生成し，生成されたオブジェクトを指すポインタを返します。
  - id

構造体のリポジトリ ID

- name
  構造体の名前
- version
  構造体のバージョン
- original_type
  エイリアスである，このオブジェクトの元のオブジェクトの IDL 型

24.10 DefinitionKind

```
enum CORBA::DefinitionKind
```

DefinitionKind 列挙体の定数は，インタフェースリポジトリオブジェクトとして指定できる型を定義します。

24.10.1 DefinitionKind の列挙値

DefinitionKind の定数値を次の表に示します。

表 24–2 DefinitionKind の定数値（C++）

定数	意味
dk_all	すべての指定できる型（リポジトリ検索メソッドで使用）
dk_none	すべての型を除外（リポジトリ検索メソッドで使用）
dk_Alias	エイリアス
dk_Array	配列
dk_Attribute	属性
dk_Constant	定数
dk_Enum	Enum（列挙体）
dk_Exception	例外
dk_Fixed	Fixed
dk_Interface	インタフェース
dk_Module	module
dk_Native	Native
dk_Operation	インタフェースオペレーション
dk_Primitive	基本型（int，long など）
dk_Repository	リポジトリ
dk_Sequence	シーケンス
dk_String	文字列
dk_Struct	構造体
dk_Typedef	TypeDef
dk_Union	union
dk_Value	ValueType

定数	意味
dk_ValueBox	ValueBox
dk_ValueMember	ValueMember
dk_Wstring	Unicode 文字列

24.11 Description

```
struct CORBA::Container::Description
```

この構造体は，Contained クラスから派生したインタフェースリポジトリ内の項目の一般的な記述を提供します。

24.11.1 Description のメンバ

- CORBA::Contained_var contained_object
  この構造体にオブジェクトが含まれます。
- CORBA::DefinitionKind kind
  オブジェクトの種類です。
- CORBA::Any value
  オブジェクトの値です。

24.12 EnumDef

```
class CORBA::EnumDef : public virtual CORBA::TypedefDef,
                       public virtual CORBA::Object
```

このクラスは，インタフェースリポジトリに格納されている列挙体を記述します。このクラスは，列挙体のメンバリストを設定および取得するメソッドを提供します。

24.12.1 EnumDef のメソッド

- CORBA::EnumMemberSeq *members();
  このメソッドは，列挙体のメンバリストを返します。
- void members(
  CORBA::EnumMemberSeq members) ;
  このメソッドは，列挙体のメンバリストを設定します。
  - members
    メンバのリスト

24.13 ExceptionDef

```
class CORBA::ExceptionDef : public virtual CORBA::Contained,
                            public virtual CORBA::Container,
                            public virtual CORBA::Object
```

このクラスは，インタフェースリポジトリに格納されている例外を記述します。このクラスは，例外のメンバのリストを設定および取得するメソッドを提供します。また，例外の TypeCode を取得するメソッドも提供します。

24.13.1 ExceptionDef のメソッド

- CORBA::StructMemberSeq *members() ;
  このメソッドは，例外のメンバのリストを返します。
- void members(
  CORBA::StructMemberSeq& members) ;
  このメソッドは，例外のメンバのリストを設定します。
  - members
    メンバのリスト
- CORBA::TypeCode_ptr type() ;
  このメソッドは，例外の型を表す TypeCode を返します。

24.14 ExceptionDescription

```
struct CORBA::ExceptionDescription
```

この構造体は，インタフェースリポジトリに格納されている例外の情報を記述します。

24.14.1 ExceptionDescription のメンバ

- CORBA::String_var defined_in
  例外が定義されたモジュールまたはインタフェースのリポジトリ ID です。
- CORBA::String_var id
  例外のリポジトリ ID です。
- CORBA::String_var name
  例外の名前です。
- CORBA::TypeCode_var type
  例外の IDL 型です。
- CORBA::String_var version
  例外のバージョンです。

24.15 FixedDef

```
class CORBA::FixedDef public virtual CORBA::IDLType,
                      public virtual CORBA::Object
```

このクラスは，インタフェースリポジトリに格納されている fixed 型を記述するために使用します。

24.15.1 FixedDef のメソッド

- CORBA::UShort digits();
  このメソッドは，fixed 型のけた数を設定します。
- void digits(
  CORBA::UShort _digits);
  このメソッドは，fixed 型の属性を設定します。
- CORBA::Short scale();
  このメソッドは，fixed 型のスケールを設定します。
- void scale(
  CORBA::Short _scale);
  このメソッドは，fixed 型の属性を設定します。

24.16 FullInterfaceDescription

```
struct CORBA::FullInterfaceDescription
```

FullInterfaceDescription 構造体は，インタフェースリポジトリに格納されている定数を記述します。

24.16.1 FullInterfaceDescription のメンバ

- CORBA::String_var Name
  インタフェースの名前です。
- CORBA::String_var id
  インタフェースのリポジトリ ID です。
- CORBA::String_var defined_in
  インタフェースが定義された module またはインタフェースの名前です。
- CORBA::String_var version
  インタフェースのバージョンです。
- CORBA::OpDescriptionSeq operations
  このインタフェースがサポートするオペレーションのリストです。
- CORBA::AttrDescriptionSeq attributes
  このインタフェースに含まれる属性のリストです。
- CORBA::RepositoryIdSeq base_interfaces
  このインタフェースが継承するインタフェースのリストです。
- CORBA::RepositoryIdSeq derived_interfaces
  このインタフェースから派生したインタフェースです。
- CORBA::TypeCode_var type
  インタフェースの TypeCode です。
- CORBA::Boolean is_abstract
  このインタフェースが abstract 型かどうかを表します。

24.17 FullValueDescription

```
struct CORBA::FullValueDescription
```

この構造体は，インタフェースリポジトリに格納されている完全な値定義を表すために使用します。

24.17.1 FullValueDescription の変数

- CORBA::String_var **name**
  valuetype の名前を表します。
- CORBA::String_var **id**
  valuetype のリポジトリ ID を表します。
- CORBA::Boolean **is_abstract**
  この変数が true の場合，abstract 型の valuetype を指定します。
- CORBA::Boolean **is_custom**
  この変数が true の場合，valuetype に対して custom 型のマーシャリングを指定します。
- CORBA::String_var **defined_in**
  valuetype が定義されているモジュールのリポジトリ ID を表します。
- CORBA::String_var **version**
  valuetype のバージョンを表します。
- CORBA::OpDescriptionSeq **operations**
  valuetype が提供するオペレーションの一覧を表します。
- CORBA::AttrDescriptionSeq **attributes**
  valuetype のメンバ属性の，valuetype の一覧を表します。
- CORBA::ValueMemberSeq **members**
  値定義の配列を表します。
- CORBA::InitializerSeq **initializers**
  イニシャライザの配列を表します。
- CORBA::RepositoryIdSeq **supported_interfaces;**
  サポートされるインタフェースの一覧を表します。
- CORBA::RepositoryIdSeq **abstract_base_values;**
  この valuetype の継承しているすべての抽象 value 型の一覧を表します。
- CORBA::Boolean **is_truncatable;**
  この変数が true の場合，値をベースの valuetype に安全に切り捨てることができます。

- CORBA::String_var **base_values;**
  この valuetype が継承している valuetype を説明します。
- CORBA::TypeCode_var **type**
  valuetype の IDL TypeCode を表します。

24.18 IDLType

```
class CORBA::IDLType : public virtual CORBA::IRObject,
                       public virtual CORBA::Object
```

IDLType クラスは，IDL 型を表しているすべてのインタフェースリポジトリ定義によって，継承されている抽象インタフェースを提供します。このクラスは，オブジェクトの型を識別する，オブジェクトの TypeCode を返すためのメソッドを提供します。IDLType は個々の特有なものですが，TypeCode は個々の特有なものではありません。

24.18.1 インクルードファイル

このクラスを使用するときは，corba.h ファイルをインクルードしてください。

```
interface IDLType:IRObject {
    readonly attribute TypeCode type;
};
```

24.18.2 IDLType のメソッド

- CORBA::Typecode_ptr type();
  このメソッドは，現在の IRObject の TypeCode を返します。

24.19 InterfaceDef

```
class CORBA::InterfaceDef : public virtual CORBA::Container,
                            public virtual CORBA::Contained,
                            public virtual CORBA::IDLType,
                            public virtual CORBA::Object
```

InterfaceDef クラスは，インタフェースリポジトリに格納されている ORB オブジェクトのインタフェースを定義するために使用します。

詳細については，「24.8 Contained」，「24.9 Container」，「24.18 IDLType」を参照してください。

24.19.1 インクルードファイル

このクラスを使用するときは，corba.h ファイルをインクルードしてください。

```
interface InterfaceDef: Container, Contained, IDLType {
typedef sequence<RepositoryId> RepositoryIdSeq;
typedef sequence<OperationDescription> OpDescriptionSeq;
typedef sequence<AttributeDescription> AttrDescriptionSeq;
    attribute InterfaceDefSeq base_interfaces;
    attribute boolean is_abstract;
    readonly attribute InterfaceDefSeq
    derived_interfaces
    boolean is_a(in RepositoryId interface_id);
    struct FullInterfaceDescription {
Identifier name;
    RepositoryId id;
    RepositoryId defined_in;
    String_var version;
    OpDescriptionSeq operations;
    AttrDescriptionSeq attributes;
    RepositoryIdSeq base_interfaces;
        RepositoryIdSeq derived_interfaces;
    TypeCode type;
        boolean is_abstract;
    };

    FullInterfaceDescription describe_interface();

        AttributeDef create_attribute(
            in RepositoryId id,
            in Identifier name,
            in String_var version,
            in IDLType type,
            in CORBA::AttributeMode mode
        );
        OperationDef create_operation(
            in RepositoryId id,
            in Identifier name,
            in String_var version,
            in IDLType result,
```

```
            in OperationMode mode,
            in ParDescriptionSeq params,
            in ExceptionDefSeq exceptions,
            in ContextIdSeq contexts
        );
        struct InterfaceDescription {
            Identifier name;
            RepositoryId id;
            RepositoryId defined_in;
            String_var version;
            RepositoryIdSeq base_interfaces;
            boolean is_abstract;
        };
    };
```

24.19.2 InterfaceDef のメソッド

- CORBA::InterfaceDefSeq *base_interfaces();
  このメソッドは，このクラスが継承したインタフェースのリストを返します。
- void base_interfaces(
  const CORBA::InterfaceDefSeq& val);
  このメソッドは，このクラスが継承したインタフェースのリストを設定します。
  - val
    このインタフェースが継承したインタフェースのリスト
- CORBA::AttributeDef_ptr create_attribute(
  const char * id,
  const char * name,
  const CORBA::String_var& version,
  CORBA::IDLType_ptr type,
  CORBA::AttributeMode mode);
  このメソッドは，このオブジェクトに含まれている，新しく生成された AttributeDef を指すポインタを返します。id，name，version，type，および mode が指定された値に設定されます。
  - id
    使用するインタフェース ID
  - name
    使用するインタフェース名
  - version
    使用するインタフェースのバージョン
  - type
    使用する型

- mode
  インタフェースモード。指定できる値については，「24.5　AttributeMode」を参照してください。

- CORBA::OperationDef_ptr **create_operation**(
  const char *id,
  const char *name,
  CORBA::String_var& version,
  CORBA::IDLType_ptr result,
  CORBA::OperationMode mode,
  const CORBA::ParDescriptionSeq& params,
  const CORBA::ExceptionDefSeq& exceptions,
  const CORBA::ContextIdSeq& contexts);
  このメソッドは，指定されたパラメタを使って，このオブジェクトによって含まれる新規の OperationDef を生成します。その新しく生成された OperationDef の defined_in 属性は，この InterfaceDef を識別できるように設定します。
  - id
    オペレーションのインタフェース ID
  - name
    オペレーションの名前
  - version
    オペレーションのバージョン
  - result
    オペレーションによって返される IDL 型
  - mode
    オペレーションのモード（一方向モードまたは標準モード）
  - params
    オペレーションに渡すパラメタのリスト
  - exceptions
    オペレーションによって発生した例外のリスト
  - contexts
    コンテキストのリストは，コンテキスト内の値の名前であり，リクエストとともに渡されます。
- CORBA::InterfaceDef::FullInterfaceDescription
  *describe_interface();
  このメソッドは，このオブジェクトのインタフェースを記述する FullInterfaceDescription を返します。
- CORBA::Boolean **is_a**(
  const char * interface_id);

このインタフェースが，指定されたインタフェースと同一であるか，そのインタフェースから直接的または間接的に継承されたものである場合，このメソッドは true を返します。

- interface_id
  このインタフェースと比較してチェックされるインタフェースの ID

24.20 InterfaceDescription

```
struct CORBA::InterfaceDescription
```

この構造体は，インタフェースリポジトリに格納されているインタフェースを記述します。

24.20.1 InterfaceDescription のメンバ

- CORBA::String_var name
  インタフェースの名前です。
- CORBA::String_var id
  インタフェースのリポジトリ ID です。
- CORBA::String_var defined_in
  インタフェースが定義されたリポジトリ ID の名前です。
- CORBA::String_var version
  インタフェースのバージョンです。
- CORBA::RepositoryIdSeq base_interfaces
  インタフェースのベースインタフェースのリストです。
- CORBA::Boolean is_abstract
  このインタフェースが abstract 型かどうかを表します。
  インタフェースが abstract 型の場合，そのインタフェースを表すオブジェクトは実体化できません。

24.21 IRObject

```
class CORBA::IRObject : public virtual CORBA::Object
```

IRObject クラスは，インタフェースリポジトリオブジェクトの最も一般的なインタフェースを提供します。このクラスから，Container クラス，IDLType，Contained クラス，およびその他のクラスが派生します。

24.21.1 インクルードファイル

このクラスを使用するときは，corba.h ファイルをインクルードしてください。

```
interface IRObject {
    readonly attribute DefinitionKind def_kind;
    void destroy();
};
```

24.21.2 IRObject のメソッド

- CORBA::DefinitionKind **def_kind**();
  このメソッドは，このインタフェースリポジトリオブジェクトの型を返します。指定できる型については，「24.10 DefinitionKind」を参照してください。
- void **destroy**();
  このメソッドは，インタフェースリポジトリから，このオブジェクトを削除します。オブジェクトが Container である場合，その内容のすべてを削除します。オブジェクトがほかのオブジェクトによって含まれている場合でも，そのオブジェクトは削除されます。Repository オブジェクト，または PrimitiveDef オブジェクトに対して destroy メソッドが呼び出されると，Exception (CORBA::BAD_PARAM)が返されます。Repository クラスの詳細については，「24.32 Repository」を参照してください。

24.22 ModuleDef

```
class CORBA::ModuleDef : public virtual CORBA::Container,
                         public virtual CORBA::Contained,
                         public virtual CORBA::Object
```

このクラスは，インタフェースリポジトリ内の IDL モジュールを表すために使用します。

24.23 ModuleDescription

```
struct CORBA::ModuleDescription
```

ModuleDescription 構造体は，インタフェースリポジトリに格納されている module を記述します。

24.23.1 ModuleDescription のメンバ

- CORBA::String_var name
  module の名前です。
- CORBA::String_var id
  module のリポジトリ ID です。
- CORBA::String_var defined_in
  module が定義されたリポジトリ ID の名前です。
- CORBA::String_var version
  module のバージョンです。

24.24 NativeDef

```
class CORBA::NativeDef : public virtual CORBA::TypedefDef,
                         public virtual CORBA::Object
```

このクラスは，インタフェースリポジトリが格納するネイティブ定義を表すために使用します。

24.25 OperationDef

```
class CORBA::OperationDef : public virtual CORBA::Contained,
                            public virtual CORBA::Object
```

OperationDef クラスは，インタフェースリポジトリに格納されているインタフェースオペレーションの情報を含んでいます。このクラスは，Contained クラスから派生します。このクラスを継承した describe メソッドは，オペレーションについての詳細な情報を提供する OperationDescription 構造体を返します。Contained クラスについては，「24.8 Contained」を参照してください。

24.25.1 インクルードファイル

このクラスを使用するときは，corba.h ファイルをインクルードしてください。

```
interface OperationDef: Contained {
    typedef sequence<ParameterDescription> ParDescriptionSeq;
    typedef Identifier ContextIdentifier;
    typedef sequence<ContextIdentifier> ContextIdSeq;
    typedef sequence<ExceptionDef> ExceptionDefSeq;
    typedef sequence<ExceptionDescription> ExcDescriptionSeq;
    readonly attribute TypeCode result;
    attribute IDLType result_def;
    attribute ParDescriptionSeq params;
    attribute CORBA::OperationMode mode;
    attribute ContextIdSeq contexts;
    attribute ExceptionDefSeq exceptions;
readonly attribute OperationKind bind;
};
struct OperationDescription {
    Identifier name;
    RepositoryId id;
    RepositoryId defined_in;
    String_var version;
    TypeCode result;
    OperationMode mode;
    ContextIdSeq contexts;
    ParDescriptionSeq parameters;
    ExcDescriptionSeq exceptions;
};
```

24.25.2 OperationDef のメソッド

- CORBA::ContextIdSeq * **contexts**();
  このメソッドは，オペレーションに適用されるコンテキスト識別子のリストを返します。
- void **context**(
  const CORBA::ContextIdSeq& **val**);

このメソッドは，オペレーションに適用されるコンテキスト識別子のリストを設定します。

- val
  コンテキスト識別子のリスト

- CORBA::ExceptionDefSeq * **exceptions**();
  このメソッドは，このオペレーションによって発生する可能性のある例外型のリストを返します。
- void **exceptions**(
  const CORBA::ExceptionDefSeq& **val**);
  このメソッドは，このオペレーションによって発生する可能性のある例外型のリストを設定します。
  - val
    このオペレーションによって発生する可能性のある例外のリスト
- CORBA::OperationMode **mode**();
  このメソッドは，この OperationDef が示すオペレーションのモードを返します。モードは，標準または一方向のどちらかです。標準モードのオペレーションは，同期的であり，クライアントアプリケーションに値を返します。一方向オペレーションでは応答を待ちません。また，オブジェクトインプリメンテーションからクライアントへの応答は送信されません。
- void **mode**(
  CORBA::OperationMode **val**);
  このメソッドは，オペレーションのモードを設定します。
  - val
    このオペレーションの指定モード。OP_ONEWAY または OP_NORMAL のどちらかです。詳細については，「24.27　OperationMode」を参照してください。
- CORBA::ParDescriptionSeq * **params**();
  このメソッドは，この OperationDef に対するパラメタを記述する ParameterDescription 構造体のリストを指すポインタを返します。
- void **params**(
  const CORBA::ParDescriptionSeq& **val**);
  このメソッドは，この OperationDef の ParameterDescription 構造体のリストを設定します。リストする構造体の順序は，オペレーションの IDL 定義で定義された順序と同一の必要があります。
  - val
    ParameterDescription 構造体のリスト
- CORBA::TypeCode_ptr **result**();
  このメソッドは，このオペレーションによって返された値の型を表す TypeCode を指すポインタを返します。TypeCode は read-only 属性を持ちます。
- CORBA::IDLType_ptr **result_def**();
  このメソッドは，この OperationDef によって返された IDL 型の定義を指すポインタを返します。
- void **result_def**(

CORBA::IDLType_ptr val);

このメソッドは，この OperationDef によって返された型の定義を設定します。

- val
  使用する型定義を指すポインタ

24.26 OperationDescription

```
struct CORBA::OperationDescription
```

OperationDescription 構造体は，インタフェースリポジトリに格納されているオペレーションの情報を記述します。

24.26.1 OperationDescription のメンバ

- CORBA::String_var **name**
  オペレーションの名前です。
- CORBA::String_var **id**
  オペレーションのリポジトリ ID です。
- CORBA::String_var **defined_in**
  このオペレーションが定義されたインタフェース，または valuetype のリポジトリ ID です。
- CORBA::String_var **version**
  オペレーションのバージョンです。
- CORBA::TypeCode_var **result**
  オペレーションの結果です。
- CORBA::OperationMode **mode**
  オペレーションのモードです。
- CORBA::ContextIdSeq **contexts**
  オペレーションに対応するコンテキストリストです。
- CORBA::ParameterDescriptionSeq **parameters**
  オペレーションのパラメタです。
- CORBA::ExceptionDescriptionSeq **exceptions**
  オペレーションによって発生する可能性のある例外です。

24.27 OperationMode

```
enum CORBA::OperationMode
```

この列挙体は，オペレーションのモードを表す値を定義するために使用します。モードは，標準モードまたは一方向モードのどちらかです。標準オペレーション要求では，リクエストの結果を含む応答がオブジェクトインプリメンテーションによってクライアントへ送信されます。一方向オペレーション要求では，クライアントアプリケーションは応答を受信しません。

24.27.1 OperationMode の値

OperationMode の値を次の表に示します。

表 24-3 OperationMode の値（C++）

定数	意味
OP_NORMAL	標準モード（標準オペレーション要求）
OP_ONEWAY	一方向モード（一方向オペレーション要求）

24.28 ParameterDescription

```
struct CORBA::ParameterDescription
```

ParameterDescription 構造体は，インタフェースリポジトリに格納されているオペレーションのパラメタを記述します。

24.28.1 ParameterDescription のメンバ

- CORBA::String_var **name**
  パラメタの名前です。
- CORBA::TypeCode_var **type**
  パラメタの型です。
- CORBA::IDLType_var **type_def**
  パラメタの IDL 型です。
- CORBA::ParameterMode **mode**
  パラメタのモードです。

24.29 ParameterMode

```
enum CORBA::ParameterMode
```

オペレーションで使用できるパラメタのモードを表す値を定義します。

24.29.1 ParameterMode の値

ParameterMode の値を次の表に示します。

表 24-4 ParameterMode の値（C++）

定数	意味
PARAM_IN	クライアントからサーバへの入力に使用するパラメタ
PARAM_OUT	サーバからクライアントへの結果の出力に使用するパラメタ
PARAM_INOUT	クライアントからの入力，またはサーバからの出力の両方に使用できるパラメタ

24.30 PrimitiveDef

```
class CORBA::PrimitiveDef : public virtual CORBA::IDLType,
                            public virtual CORBA::Object
```

このクラスは，インタフェースリポジトリに格納されている基本型（int，long など）を記述するために使用します。このクラスは，基本型の種類を取得するメソッドを提供します。

24.30.1 PrimitiveDef のメソッド

- CORBA::PrimitiveKind kind();

  このメソッドは，このオブジェクトが表す基本型の種類を表します。

24.31 PrimitiveKind

```
enum CORBA::PrimitiveKind
```

PrimitiveKind 列挙体は，インタフェースリポジトリに格納できるオブジェクトの基本型を定義する定数を含んでいます。

24.31.1 PrimitiveKind の定数

PrimitiveKind の定数値を次の表に示します。

表 24-5 PrimitiveKind の定数値（C++）

定数	意味
pk_any	Any
pk_boolean	boolean
pk_char	文字
pk_double	double
pk_float	float
pk_long	long
pk_longdouble	long double
pk_longlong	long long
pk_null	NULL 値
pk_objref	オブジェクトリファレンス
pk_octet	オクテット
pk_Principal	Principal
pk_short	short
pk_string	文字列
pk_TypeCode	TypeCode
pk_ulong	unsigned long
pk_ulonglong	unsigned long long
pk_ushort	unsigned short
pk_void	void
pk_wchar	Unicode 文字
pk_wstring	Unicode 文字列

24.32 Repository

```
class CORBA::Repository : public virtual CORBA::Container,
                          public virtual CORBA::Object
```

Repository クラスは，インタフェースリポジトリへのアクセスを提供します。このクラスは Container クラスから派生します。Container クラスの詳細については，「24.9 Container」を参照してください。

このクラスを使用するには，ORB::resolve_initial_references("InterfaceRepository")で取得したオブジェクトをナロウして使用してください。

24.32.1 インクルードファイル

このクラスを使用するときは，corba.h ファイルをインクルードしてください。

```
interface Repository: Container {
    Contained lookup_id(in RepositoryId search_id);
    PrimitiveDef get_primitive(in CORBA::PrimitiveKind kind);
    StringDef create_string(in unsigned long bound);
    WStringDef create_wstring(in unsigned long bound);
    SequenceDef create_sequence(
        in unsigned long bound,
        in IDLType element_type
    );
    ArrayDef create_array(
        in unsigned long length,
        in IDLType element_type
    );
    FixedDef create_fixed(
       in unsigned short digits,
       in short scale
    );
};
```

24.32.2 Repository のメソッド

- CORBA::ArrayDef_ptr create_array(
  CORBA::ULong length,
  CORBA::IDLType_ptr element_type);
  このメソッドは，新規の ArrayDef を生成し，そのオブジェクトを指すポインタを返します。
  - length
    配列要素の最大数。この値には 0 より大きい数を設定します。
  - element_type

配列に格納されている要素の IDLType

- CORBA::SequenceDef_ptr **create_sequence**(
  CORBA::ULong **bound**,
  CORBA::IDLType_ptr **element_type**);
  このメソッドは，新規の SequenceDef オブジェクトを生成し，そのオブジェクトを指すポインタを返します。
  - bound
    シーケンス内の項目の最大数。この値には 0 より大きい数を設定します。
  - element_type
    シーケンスに格納されている項目の IDLType を指すポインタ
- CORBA::StringDef_ptr **create_string**(
  CORBA::Ulong **bound**);
  このメソッドは，新規の StringDef オブジェクトを生成し，そのオブジェクトを指すポインタを返します。
  - bound
    文字列の最大長。この値には 0 より大きい数を設定します。
- CORBA::WstringDef_ptr **create_wstring**(
  CORBA::Ulong **bound**);
  このメソッドは，新規の WstringDef オブジェクトを生成し，そのオブジェクトを指すポインタを返します。
  - bound
    文字列の最大長。この値には 0 より大きい数を設定します。
- CORBA::PrimitiveDef_ptr **get_primitive**(
  CORBA::PrimitiveKind **kind**);
  このメソッドは，PrimitiveKind のリファレンスを返します。
  - kind
    返されるリファレンス
- CORBA::Contained_ptr **lookup_id**(
  const char * **search_id**);
  このメソッドは，指定された検索対象の ID と一致するオブジェクトをインタフェースリポジトリ内で検索します。一致するオブジェクトが見つからない場合，NULL 値が返されます。
  - search_id
    検索対象となる識別子
- CORBA::FixedDef_ptr **create_fixed** (
  CORBA::UShort **digits**, CORBA::Short **scale**);

このメソッドは，fix 型のけた数とスケールを設定します。

- digits
  fix 型のけた数
- scale
  fix 型のスケール

24.33 SequenceDef

```
class CORBA::SequenceDef : public virtual CORBA::IDLType,
                           public virtual CORBA::Object
```

このクラスは，インタフェースリポジトリに格納されているシーケンスを表すために使用します。このクラスは，シーケンスのバウンドまたは要素型を設定したり取得したりするメソッドを提供します。

24.33.1 SequenceDef のメソッド

- CORBA::ULong **bound**();
  このメソッドは，シーケンスのバウンドを返します。
- void **bound**(
  CORBA::ULong **bound**);
  このメソッドは，シーケンスのバウンドを設定します。
  - bound
    シーケンスのバウンド
- CORBA::TypeCode_ptr **element_type**();
  このメソッドは，このシーケンスの要素の TypeCode を返します。
- CORBA::IDLType_ptr **element_type_def**();
  このメソッドは，このシーケンスに格納されている要素の IDL 型を返します。
- void **element_type_def**(
  CORBA::IDLType_ptr **element_type_def**);
  このメソッドは，このシーケンスに格納されている要素の IDL 型を設定します。
  - element_type_def
    要素を設定する IDL 型

24.34 StringDef

```
class CORBA::StringDef : public virtual CORBA::IDLType,
                         public virtual CORBA::Object
```

このクラスは，インタフェースリポジトリに格納されている String を記述するために使用します。このクラスは，文字列のバウンドを設定および取得するメソッドを提供します。

24.34.1 StringDef のメソッド

- CORBA::ULong **bound**() ;
  このメソッドは，String のバウンドを返します。
- void **bound**(
  CORBA::ULong **bound**) ;
  このメソッドは，String のバウンドを設定します。
  - bound
    新しく設定する String オブジェクトのバウンド

24.35 StructDef

```
class CORBA::StructDef : public virtual CORBA::TypedefDef,
                         public virtual CORBA::Container,
                         public virtual CORBA::Object
```

このクラスは，インタフェースリポジトリに格納されている構造体のメンバ一覧を設定および取得するためのメソッドを提供します。

24.35.1 StructDef のメソッド

- CORBA::StructMemberSeq *members() ;
  このメソッドは，構造体のメンバのリストを返します。
- void members(
  CORBA::StructMemberSeq& members) ;
  このメソッドは，構造体のメンバのリストを設定します。
  - members
    メンバのリスト

24.36 StructMember

```
struct CORBA::StructMember
```

この構造体は，struct のメンバを定義するために使用します。この構造体は，定義内の名前と型を変数として使用します。

24.36.1 StructMember のメンバ

- CORBA::String_var **name**
  型の名前です。
- CORBA::TypeCode_var **type**
  型の IDL 型です。
- CORBA::IDLType_var **type_def**
  IDL 型の IDL 型定義です。

24.37 TypedefDef

```
class CORBA::TypedefDef : public virtual CORBA::Contained,
                          public virtual CORBA::IDLType,
                          public virtual CORBA::Object
```

この抽象ベースクラスは，インタフェースリポジトリに格納されているユーザ定義構造体を表します。このクラスを継承するすべてのインタフェースを次に示します。

- AliasDef
  「24.1　AliasDef」を参照してください。
- EnumDef
  「24.12　EnumDef」を参照してください。
- ExceptionDef
  「24.13　ExceptionDef」を参照してください。
- StructDef
  「24.35　StructDef」を参照してください。
- UnionDef
  「24.39　UnionDef」を参照してください。
- WstringDef
  「24.44　WstringDef」を参照してください。

24.38 TypeDescription

```
struct CORBA::TypeDescription
```

TypeDescription 構造体は，インタフェースリポジトリに格納されているオペレーションの型を記述した情報を含んでいます。

24.38.1 TypeDescription のメンバ

- CORBA::String_var name
  型の名前です。
- CORBA::String_var id
  型のリポジトリ ID です。
- CORBA::String_var defined_in
  この型が定義されたモジュールまたはインタフェースの名前です。
- CORBA::String_var version
  型のバージョンです。
- CORBA::TypeCode_var type
  型の IDL 型です。

24.39 UnionDef

```
class CORBA::UnionDef : public virtual CORBA::TypedefDef,
                        public virtual CORBA::Container,
                        public virtual CORBA::Object
```

このクラスは，インタフェースリポジトリに格納されている union を表すために使用します。このクラスは，union のメンバのリスト，またはディスクリミネータ型を，設定および取得するメソッドを提供します。

24.39.1 UnionDef のメソッド

- CORBA::TypeCode_ptr **discriminator_type**() ;
  このメソッドは，union のディスクリミネータの TypeCode を返します。
- CORBA::IDLType_ptr **discriminator_type_def**() ;
  このメソッドは，union のディスクリミネータの IDL 型を返します。
- void **discriminator_type_def**(
  CORBA::IDLType_ptr **discriminator_type_def**) ;
  このメソッドは，union のディスクリミネータの IDL 型を設定します。
  - discriminator_type_def
    メンバのリスト
- CORBA::UnionMemberSeq ***members**() ;
  このメソッドは，union のメンバのリストを返します。
- void **members**(
  CORBA::UnionMemberSeq& **members**);
  このメソッドは，union のメンバのリストを設定します。
  - members
    メンバのリスト

24.40 UnionMember

```
struct CORBA::UnionMember
```

UnionMember 構造体は，インタフェースリポジトリに格納されている union の情報を含みます。

24.40.1 UnionMember のメンバ

- CORBA::String_var **name**
  union の名前です。
- CORBA::Any **label**
  union のラベルです。
- CORBA::TypeCode_var **type**
  union の TypeCode です。
- CORBA::IDLType_var **type_def**
  union の IDL 型です。

24.41 ValueBoxDef

```
class CORBA::ValueBoxDef public virtual CORBA::Contained,
                         public virtual CORBA::IDLType,
                         public virtual CORBA::Object
```

ValueBoxDef クラスは，任意の IDL 型の公開メンバを一つ格納する簡易な valuetype として使用します。このクラスは，次の valuetype を簡略化したものです。

```
public valuetype <IDLType> value;
```

この宣言はボックス型の<IDLType>とほとんど同じですが，ValueBoxDef クラスは簡易な ValueTypeDef インタフェースとは異なります。

24.41.1 ValueBoxDef のメソッド

- CORBA::IDLType_ptr original_type_def();
  このメソッドは，ボックス化されている型を識別します。
- void original_type_def(
  CORBA::IDLType_ptr original_type_def);
  このメソッドは，ボックス化する型を設定します。

24.42 ValueDef

```
class CORBA::ValueDef public virtual CORBA::Container,
                      public virtual CORBA::Contained,
                      public virtual CORBA::IDLType,
                      public virtual CORBA::Object
```

このインタフェースは，construct という IDL 値を記述します。このインタフェースが格納できるのは，定数，型定義，例外，オペレーション，および属性です。このインタフェースは，クラス型とよく似ていて，インタフェースリポジトリに格納されている値定義を表します。

24.42.1 ValueDef のメソッド

- CORBA::InterfaceDefSeq **supported_interfaces**( );
  このメソッドは，この valuetype がサポートするインタフェースの一覧を返します。
- void **supported_interfaces**(
  const CORBA::interfaceDefSeq& supported_interfaces);
  このメソッドは，サポートするインタフェースを設定します。
- CORBA::InitializerSeq* **initializers**( );
  このメソッドは，イニシャライザの一覧を返します。
- void **initializers**(
  const CORBA::InitializerSeq& **initializers**);
  このメソッドは，イニシャライザを設定します。
- CORBA::ValueDef_ptr **base_value**( );
  このメソッドは，この値の継承元 valuetype を返します。
- void **base_value**(
  CORBA::ValueDef_ptr **base_value**);
  このメソッドは，valuetype を設定します。
- CORBA::ValueDefSeq& **abstract_base_values**( );
  このメソッドは，この値が継承する abstract 型 valuetype の一覧を返します。
- void **abstract_base_values**(
  const CORBA::ValueDefSeq& abstract_base_values);
  このメソッドは，abstract 型 valuetype のベース値を定義します。
- CORBA::Boolean **is_abstract**( );
  値が abstract 型 valuetype の場合，このメソッドは true を返します。
- void **is_abstract**(

CORBA::Boolean is_abstract);
このメソッドは，valuetype を abstract 型 valuetype に設定します。

- CORBA::Boolean **is_custom**( );
  値が custom 型のマーシャリングを使用する場合，このメソッドは true を返します。
- void **is_custom**(
  CORBA::Boolean **is_custom**);
  このメソッドは，値に対して custom 型のマーシャリングを設定します。
- CORBA::Boolean **is_truncatable**( ):
  値をベース値から安全に切り捨てられる場合，このメソッドは true を返します。
- void **is_truncatable**(
  CORBA::Boolean is_truncatable);
  このメソッドは，この値に truncatable 属性を設定します。
- CORBA::Boolean **is_a**(
  const char* **value_id**);
  このメソッドの呼び出しに使用した値が，value_id パラメタで定義したインタフェースまたは値と同一であるか，直接的または間接的に継承されたものである場合，このメソッドは true を返し，そうでない場合は false を返します。
- CORBA::ValueDef _ptr FullValueDescription* **describe_value**();
  このメソッドは，値に対応する FullValueDescription オブジェクトを，オペレーションと属性を含めて返します。
- CORBA::ValueMemberDef_ptr **create_value_member**(
  const char* **id**,
  const char* **name**,
  const char***version**,
  CORBA::IDLType_ptr **type_def**,
  CORBA::Short **access**);
  このメソッドは，このメソッドの呼び出し対象の ValueDef オブジェクトが格納する，新しい ValueMemberDef オブジェクトを返します。
  - id
    型のリポジトリ ID
  - name
    型の名前
  - version
    オブジェクトのバージョン
  - type_def

IDL 型の値

- access
  アクセス値

- CORBA::AttributeDef_ptr **create_attribute**(
  const char* **id**,
  const char* **name**,
  const char* **version**,
  CORBA::IDLType_ptr **type**,
  CORBA::AttributeMode **mode**);
  このメソッドは，この valuetype に新規属性定義を生成し，その定義に対応する新規 AttributeDef オブジェクトを返します。
  - id
    型のリポジトリ ID
  - name
    型の名前
  - version
    オブジェクトのバージョン
  - type
    IDL 型の値
  - mode
    オブジェクトのモード
- CORBA::OperationDef_ptr **create_operation**(
  const char* **id**,
  const char* **name**,
  const char* **version**,
  CORBA::IDLType_ptr **result**,
  CORBA::OpeartionMode **mode**,
  const CORBA::ParDescriptionSeq& **params**,
  const CORBA::ExceptionDefSeq& **exceptions**,
  const CORBA::ContextIdSeq& **contexts**);
  このメソッドは，この valuetype の新規オペレーションを生成し，対応する OperationDef オブジェクトを返します。
  - id
    型のリポジトリ ID
  - name

型の名前

- version
  オブジェクトのバージョン
- result
  オペレーションの IDL 型
- mode
  オブジェクトのモード
- params
  オペレーションのパラメタの一覧
- exceptions
  オペレーションの例外の一覧
- contexts
  オペレーションのコンテキストの一覧

24.43 ValueDescription

```
struct CORBA::ValueDescription
```

この構造体は，インタフェースリポジトリが格納する valuetype の定義を表すために使用します。

24.43.1 ValueDescription の変数

- CORBA::String_var **name**
  型の名前を表します。
- CORBA::String_var **id**
  型のリポジトリ ID を表します。
- CORBA::Boolean **is_abstract**
  true を設定した場合，値は abstract 型 valuetype です。
- CORBA::Boolean **is_custom**
  true を設定した場合，valuetype に custom 型マーシャリングが実行されます。
- CORBA::String_var **defined_in**
  型が定義されているモジュールのリポジトリ ID を表します。
- CORBA::String_var **version**
  型のバージョンを表します。
- CORBA::RepositoryIdSeq& **supported_interfaces**
  valuetype がサポートするインタフェースの一覧を表します。
- CORBA::RepositoryIdSeq& **abstract_base_values**
  型の継承する abstract 型 valuetype の一覧を表します。
- CORBA::Boolean **is_truncatable**
  この変数が true の場合，valuetype は安全に切り捨てられ，ベース値にできます。
- CORBA::String_var **base_value**
  この値が継承する valuetype を表します。

24.44 WstringDef

```
class CORBA::WstringDef : public virtual CORBA::IDLType,
                          public virtual CORBA::Object
```

このクラスは，インタフェースリポジトリに格納されている Unicode 文字列を記述するために使用します。このクラスは，文字列のバウンドを設定および取得するメソッドを提供します。

24.44.1 WstringDef のメソッド

- CORBA::ULong **bound**() ;
  このメソッドは，Wstring のバウンドを返します。
- void **bound**(
  CORBA::ULong **bound**);
  このメソッドは，Wstring のバウンドを設定します。
  - bound
    新しく設定する Wstring オブジェクトのバウンド

25 活性化インタフェースとクラス（C++）

この章では，C++言語のオブジェクトインプリメンテーションの活性化で使用されるインタフェースおよびクラスについて説明します。

25.1 CreationImplDef

```
struct extension::CreationImplDef
```

CreationImplDef は，ある特定のオブジェクトインプリメンテーション用の属性の集合を提供する IDL 構造体です。これらの属性の値の問い合わせと，設定メソッドは，このインタフェースで提供します。該当する属性は，_args，_env，id（リファレンスデータ用），object_name，_path_name，_policy，および repository_id です。

オブジェクト活性化デーモンは，この構造体を使用してオブジェクトインプリメンテーションを一覧表示，登録，および登録解除します。コマンドラインパラメタは oadutil を使用したときに指定され，この構造体で定義した属性の設定に使用します。

25.1.1 インクルードファイル

この構造体を使用するときは，oad_c.hh ファイルをインクルードしてください。

25.1.2 C++のサンプル

次のサンプルでは，OAD の環境に適切な ReentrantServer 引数と LD_LIBRARY_PATH 環境変数を設定して，factory_r（ローカルディレクトリ内にある）という VisiBroker の C++アプリケーションを活性化しています。

なお，このサンプルは Solaris の場合です。HP-UX の場合は SHLIB_PATH 環境変数，AIX の場合は LIBPATH 環境変数を設定してください。

```
path_name = "/home/developer/Project1/factory_r"
args = ["ReentrantServer"]
env = ["LD_LIBRARY_PATH=/usr/ucblib:/usr/local/VisiCpp/lib"]
```

これは，OAD が次のコマンドを生成していることになります。

```
"/home/developer/Project1/factory_r ReentrantServer ¥
    -Dvbroker.orb.oadUID=<unique_id> ¥
    -Dvbroker.orb.activationIOR=<oad's ior>"
```

vbroker.orb.oadUID および vbroker.orb.activationIOR プロパティは，OAD が自動的に付加します。

また，次に示す環境変数が，OAD の環境から，生成されたサーバの環境に伝えられます。

25.1.3　環境変数

次の環境変数が，OAD の環境から，生成されたサーバの環境に伝えられるか，または OAD によって明示的に渡されます。ただし，環境変数がシェル内に設定されていない場合は，oadutil コマンドの-e オプションを使用して設定する必要があります。

- PATH
- CLASSPATH
- OSAGENT_PORT
- OSAGENT_ADDR
- VBROKER_ADM（または下位互換性のための ORBELINE）
- LD_LIBRARY_PATH，SHLIB_PATH または LIBPATH（CreationImplDef 内に設定）

そのほかの環境変数はすべて，CreationImplDef の env 属性を使って登録する必要があります。例えば，C++インプリメンテーションを生成する場合，生成された実行対象に共用ライブラリが必要なときは，CreationImplDef 環境に LD_LIBRARY_PATH（Solaris の場合），SHLIB_PATH（HP-UX の場合），または LIBPATH（AIX の場合）を明示的に登録する必要があります。

生成された C++アプリケーションでは，登録は次のコマンドにマッピングします。

```
exec-path { args1 ... argsN } -OAoad_uid=<unique_id>¥
    -OAactivateIOR=<oad's ior>
```

生成された環境には，インプリメンテーション定義から指定したすべての環境変数が含まれ，また，起動時に OAD 自身の環境から取った PATH，CLASSPATH，OSAGENT_PORT，および OSAGENT_ADDR の定義も含まれます。どの OA パラメタの場合も，OAD が追加したものは BOA_init の中に取られ，クライアントプログラムからは見えません。詳細については，「34.　コマンドラインオプション（C++）」を参照してください。

25.1.4　CreationImplDef のメンバ

- extension::Policy activation_policy;
  サーバの活性化ポリシーを設定します。
  活性化ポリシーには，SHARED_SERVER，UNSHARED_SERVER があります。
- CORBA::StringSequence args;
  サーバに渡すコマンドライン引数を設定します。
  最初の引数には必ず実行形式ファイル名を指定してください。
- CORBA::StringSequence env;
  サーバに渡す環境設定を設定します。
  env 属性の設定の詳細については，「25.1.3　環境変数」を参照してください。

- CORBA::OctetSequence id;
  インプリメンテーションのリファレンスデータ識別子を設定します。
- CORBA::String_var object_name;
  このメソッドは，インプリメンテーションのオブジェクト名を設定します。
- CORBA::String_var path_name;
  実行形式ファイルのパス名を指定する文字列を設定します。
- CORBA::String_var repository_id;
  インプリメンテーションのリポジトリ ID を指定する文字列を設定します。

25.2 ImplementationDef

```
class extension::ImplementationDef
```

ImplementationDef は，ImplementationDefs の型である ActivationImplDef と CreationImplDef 用の空のベースクラスです。ImplementationDef が使用できるのは，ActivationImplDef メソッド，または CreationImplDef メソッド内のシグニチャの中だけです。

25.2.1 インクルードファイル

このクラスを使用するときは，oad_c.hh ファイルをインクルードしてください。

25.3 ImplementationStatus

```
struct Activation::ImplementationStatus
```

ImplementationStatus は，OAD に登録されているサーバのために活性化状態を監視します。

```
module Activation
{
    ...
    struct ImplementationStatus {
      extension::CreationImplDef impl;
      ObjectStatusList status;
    };
    ...
};
```

25.3.1 インクルードファイル

この構造体を使用するときは，oad_c.hh ファイルをインクルードしてください。

25.3.2 ImplementationStatus のメンバ

- CreationImplDef **impl**;
  オブジェクトインプリメンテーションの CreationImplDef です。
- ObjectStatusList **status**;
  サーバが提供する各オブジェクトのステータス情報のリストを表します。ObjectStatusList クラスの詳細については，「25.6 ObjectStatusList」を参照してください。

25.4 OAD

```
class Activation::OAD : public virtual CORBA::Object
```

OAD クラスは，OAD へのアクセスを提供します。管理ツールは，オブジェクトのリスティング，登録，再登録をするときに OAD クラスを使用します。OAD のプログラム的管理のためにクライアントコードで使用することもできます。

コードサンプル 25-1　OAD IDL

```
module Activation {
    interface OAD {
      extension::CreationImplDef create_CreationImplDef();

      Object reg_implementation(
            in extension::CreationImplDef impl)
          raises(DuplicateEntry, InvalidPath);

      extension::CreationImplDef get_implementation(
            in CORBA::RepositoryId repId,
            in CORBA::RepositoryId repId,
            in string object_name)
          raises(NotRegistered);

      void change_implementation(
            in extension::CreationImplDef old_info,
            in extension::CreationImplDef new_info)
          raises(NotRegistered, InvalidPath, IsActive);

      attribute boolean destroy_on_unregister;

      void unreg_implementation(
            in CORBA::RepositoryId repId,
            in string object_name)
          raises(NotRegistered);

      void unreg_interface(in CORBA::RepositoryId repId)
          raises(NotRegistered);

      void unregister_all();

      ImplementationStatus get_status(
            in CORBA::RepositoryId repId,
            in string object_name)
          raises(NotRegistered);

      ImplStatusList get_status_interface(
            in CORBA::RepositoryId repId)
          raises(NotRegistered);

      ImplStatusList get_status_all();

      Object lookup_interface(
            in CORBA::RepositoryId repId,
            in long timeout)
```

```
            raises(NotRegistered, FailedToExecute,
                   NotResponding, Busy);

      Object lookup_implementation(
             in CORBA::RepositoryId repId,
             in string object_name, in long timeout)
            raises(NotRegistered, FailedToExecute,
                   NotResponding, Busy);

      extension::CreationImplDef boa_active_obj(
             in Object obj, in string repository_id,
             in long unique_id)
            raises(NotRegistered);

      void boa_deactive_obj(in Object obj,
             in string repository_id,
             in long unique_id)
            raises(NotRegistered);

    string generated_command(
              in extension::CreationImplDef impl);

    string generated_environment(
              in extension::CreationImplDef impl);

    };
};
```

25.4.1 インクルードファイル

このクラスを使用するときは，oad_c.hh ファイルをインクルードしてください。

25.4.2 OAD のメソッド

- void change_implementation(
  CORBA::CreationImplDef old_info,
  CORBA::CreationImplDef new_info);
  このメソッドは，オブジェクトのインプリメンテーションを動的に変更します。このメソッドを使って，登録の活性化ポリシー，パス名，引数設定，環境設定を変更できます。
  - old_info
    変更したい情報
  - new_info
    old_info と差し替えたい情報

  このメソッドでは次の例外が発生します。

NotRegistered

指定したオブジェクトは未登録です。登録済みオブジェクトを指定してください。

IsActive

オブジェクトインプリメンテーションは現在実行中です。オブジェクトを非活性化してから，その情報を変更してください。

注

現在活性状態のインプリメンテーションの情報は変更できません。このメソッドでオブジェクトのインプリメンテーション名やオブジェクト名を変更するときは，必ずこの注意を守ってください。そうすることで，クライアントアプリケーションがオブジェクトを古い名前で検索することを防げます。

- CreationImplDef_ptr **create_CreationImplDef**();

  このメソッドは，CreationImplDef オブジェクトのインスタンスを返し，その後，属性を設定できます。詳細については，「25.1　CreationImplDef」を参照してください。

- void **destroy_on_unregister**(

  CORBA::Boolean **val**);

  このメソッドは，OAD の destroy_on_unregister 属性を設定します。

  - val

    TRUE を設定すると，活性状態のインプリメンテーションが登録解除時にすべてシャットダウンされます。TRUE を設定しない場合は，登録解除時にシャットダウンされません。

- CORBA::Boolean **destroy_on_unregister**();

  このメソッドは，インプリメンテーションの destroy_on_unregister 属性の設定を返します。この属性に TRUE が設定されていると，活性状態のインプリメンテーションが，登録解除時にすべてシャットダウンされます。

- CORBA::CreationImplDef_ptr **get_implementation**(

  const char ***repId**,

  const char ***object_name**);

  このメソッドは，指定されたリポジトリ ID とオブジェクト名に対して登録されたインプリメンテーションに関する情報を検索します。

  - repId

    リポジトリ ID

  - object_name

    オブジェクト名

  このメソッドでは次の例外が発生します。

  NotRegistered

  指定したオブジェクトは未登録です。登録済みオブジェクトを指定してください。

- ImplementationStatus ***get_status**(

const char *repId,

const char *object_name);

このメソッドは，指定されたリポジトリ ID とオブジェクト名に対して登録されたインプリメンテーションに関するステータス情報を検索します。

- repId
  リポジトリ ID
- object_name
  オブジェクト名

- ImplStatusList *get_status_all();
  このメソッドは，すべてのインプリメンテーションに関するステータス情報を含む ImplStatusList を返します。
- ImplStatusList *get_status_interface(
  const char *repId);
  このメソッドは，指定されたリポジトリ ID に対して登録されたインプリメンテーションに関するステータス情報を取得します。
  - repId
    リポジトリ ID

  このメソッドでは次の例外が発生します。

  NotRegistered
  　指定したオブジェクトは未登録です。登録済みオブジェクトを指定してください。
- CORBA::Object reg_implementation(
  CORBA::CreationImplDef_ptr impl);
  このメソッドは，インプリメンテーションを OAD と Borland Enterprise Server VisiBroker ディレクトリサービスに登録します。
  - impl
    CreationImplDef のインスタンス

  このメソッドでは次の例外が発生します。

  DuplicateEntry
  　指定したオブジェクトは重複エントリです。未登録オブジェクトを指定してください。
- void unreg_implementation(
  const char *repId,const char *object_name);
  このメソッドは，リポジトリ ID とオブジェクト名でインプリメンテーションを登録解除します。destroy_on_unregister 属性に true が設定してあると，このメソッドは，指定されたリポジトリ ID とオブジェクト名を現在実装しているプロセスをすべて終了させます。
  - repId

リポジトリ ID

- object_name
  オブジェクト名

このメソッドでは次の例外が発生します。

NotRegistered
　指定したオブジェクトは未登録です。登録済みオブジェクトを指定してください。

- void **unreg_interface**(
  const char ***repId**);
  このメソッドは，リポジトリ ID に対応するすべてのインプリメンテーションを登録解除します。destroy_on_unregister 属性が true に設定してあると，このメソッドは，指定されたリポジトリ ID が現在実装されているプロセスを，すべて終了させます。
  - repId
    リポジトリ ID

  このメソッドでは次の例外が発生します。

  NotRegistered
　　指定したオブジェクトは未登録です。登録済みオブジェクトを指定してください。

- void **unregister_all**();
  このメソッドは，すべてのインプリメンテーションを登録解除します。destroy_on_unregister 属性に true を設定していないかぎり，活性状態のすべてのインプリメンテーションが実行を続けます。

25.5 ObjectStatus

```
struct Activation::ObjectStatus
```

この構造体は，OAD に登録されているオブジェクトインプリメンテーションによって提供される，特定のオブジェクトについての情報を表すために使います。この構造体は，ObjectStatusList クラスによって返されます。ObjectStatusList クラスについては，「25.6 ObjectStatusList」を参照してください。

```
module Activation
{
    ...
    struct ObjectStatus {
        long      unique_id;
        State     activation_state;
        Object    objRef;
    };
    ...
};
```

25.5.1 インクルードファイル

この構造体を使用するときは，oad_c.hh をインクルードしてください。

25.5.2 ObjectStatus のメンバ

- CORBA::Long **unique_id;**
  オブジェクト特有の識別子です。
- State **activation_state;**
  オブジェクトの現在の活性状態です。次のどれかの値となります。
  - ACTIVE
  - INACTIVE
  - WAITING_FOR_ACTIVATION
- CORBA::Object **objRef;**
  構造体で表される状態のオブジェクト

25.6　ObjectStatusList

```
class Activation::ObjectStatusList
```

このクラスは，ObjectStatus 構造体のリストを実装し，サーバが提供するオブジェクトの情報を表すために使用します。

「25.5　ObjectStatus」も参照してください。

25.6.1　インクルードファイル

このクラスを使用するときは，oad_c.hh をインクルードしてください。

25.6.2　ObjectStatusList のメソッド

- void **length**(
  CORBA::ULong **len**);
  リストの長さを設定します。
  - len
    リストの長さ
- CORBA::ULong **length**() const;
  リストの長さを返します。
- CORBA::ULong **maximum**() const;
  リストの最大長を返します。
- ObjectStatus& **operator[ ]**(
  CORBA::ULong **index**);
  指定されたインデックスを持つ ObjectStatus 構造体をリストから返します。
  - index
    返されるリスト項目のインデックス。このインデックスは 0 から始まります。

25.7 StringSequence

```
class CORBA::StringSequence
```

StringSequence クラスは，CreationImplDef オブジェクトに対応する引数，または環境変数のリストを含みます。CreationImplDef オブジェクトについては，「25.1 CreationImplDef」を参照してください。

25.7.1 インクルードファイル

このクラスを使用するときは，corba.h をインクルードしてください。

25.7.2 StringSequence のメソッド

- CORBA::StringSequence(
  CORBA::ULong max = 0);
  このメソッドは，指定された長さで，StringSequence オブジェクトを生成します。
  - max
    リスト内の引数の最大長。デフォルト長は 0 です。
- CORBA::StringSequence(
  CORBA::ULong max,
  CORBA::ULong length,
  char **data,
  CORBA::Boolean release = 0);
  このメソッドは，指定されたパラメタで，StringSequence オブジェクトを生成します。
  - max
    シーケンスの引数の最大数
  - length
    シーケンスの長さ
  - data
    シーケンスを構成する文字列
  - release
    1 を設定した場合，このオブジェクトがデストラクトされた際にリストに対応するすべてのメモリが解放されます。
- ~CORBA::StringSequence();
  このメソッドは，このオブジェクトをデストラクトします。

25.7.3 StringSequence に関連するメソッド

- static char **allocbuf(
  CORBA::ULong nelems);
  このメソッドは，指定されたリスト要素の数に合わせてメモリを割り当てます。
  - nelems
    リスト内の要素数
- CORBA::ULong compare(
  const CORBA::StringSequence& seq1,
  const CORBA::StringSequence& seq2);
  このメソッドは，二つの StringSequence オブジェクトを比較し，それらが同一である場合，0 を返します。そうでない場合は 0 以外の数字を返します。
  - seq1
    比較する最初のオブジェクト
  - seq2
    比較する 2 番目のオブジェクト
- static void freebuf(
  char **data);
  このメソッドは，指定されたポインタに対応するメモリを解放します。
  - data
    解放されるリストメモリ
- static void freebuf_elems(
  char **data, CORBA::ULong nelems);
  このメソッドは，指定されたリスト要素の数に合わせてメモリを解放します。また，シーケンスが使用したメモリ，および StringSequence が保持する文字列を解放します。

  注

  StringSequence に格納される文字列が，プログラムのほかから参照されている場合，その文字列が使用したメモリは解放しません。ただし，複数の文字列の配列である StringSequence は，削除され，メモリを解放します。
  - data
    解放されるリストメモリ
  - nelems
    要素の数
- CORBA::ULong hash(
  CORBA::StringSequence& seq);
  このメソッドは，指定されたオブジェクトのハッシュ値を返します。

- seq
  ハッシュ値が返される StringSequence

- CORBA::ULong **length**() const;
  このメソッドは，シーケンス内の要素数を返します。
- void **length**(
  CORBA::ULong **len**);
  このメソッドは，シーケンス内の要素数を設定します。
  - len
    新しい長さ
- CORBA::ULong **maximum**() const;
  このメソッドは，リスト内の引数の数を返します。
- CORBA::StringSequence& **operator=**(
  const CORBA::StringSequence& **seq**);
  この演算子は，代入による StringSequence のコピーを可能にします。
  - seq
    コピーするオブジェクト
- char ***operator[ ]**(
  CORBA::ULong **index**);
  この演算子は，インデックスを使って StringSequence 内の引数へのアクセスを可能にします。
  - index
    任意の文字列シーケンスの 0 から始まるインデックス
- static void **_release**(
  CORBA::StringSequence* **ptr**);
  このメソッドは，指定された StringSequence オブジェクトを解放します。
  - ptr
    解放する StringSequence オブジェクト

26 ネーミングサービスインタフェースとクラス（C++）

この章では，C++言語でプログラミングする場合に，Borland Enterprise Server VisiBroker のネーミングサービスで使用するインタフェースとクラスについて説明します。

26.1 NamingContext

```
class NamingContext : public virtual CORBA::Object
```

このオブジェクトを使用して，VisiBroker ORB オブジェクト，またはほかの NamingContext オブジェクトにバインドされるネームを登録したり，操作したりします。クライアントアプリケーションは，このインタフェースを使用して，コンテキスト内のネームを resolve または list によって処理します。オブジェクトインプリメンテーションは，このオブジェクトを使用して，オブジェクトインプリメンテーション，または NamingContext オブジェクトをネームにバインドします。NamingContext の IDL 仕様を IDL サンプル 26-1 に示します。

IDL サンプル 26-1　NamingContext インタフェースの IDL 仕様

```
module CosNaming {
    interface NamingContext {
        void bind(in Name n, in Object obj)
            raises(NotFound, CannotProceed, InvalidName,
                   AlreadyBound);
        void rebind(in Name n, in Object obj)
            raises(NotFound, CannotProceed, InvalidName);
        void bind_context(in Name n, in NamingContext nc)
            raises(NotFound, CannotProceed, InvalidName,
                   AlreadyBound);
        void rebind_context(in Name n, in NamingContext nc)
            raises(NotFound, CannotProceed, InvalidName);
        Object resolve(in Name n)
            raises(NotFound, CannotProceed, InvalidName);
        void unbind(in Name n)
            raises(NotFound, CannotProceed, InvalidName);
        NamingContext new_context( );
        NamingContext bind_new_context(in Name n)
            raises(NotFound, CannotProceed, InvalidName,
                   AlreadyBound);
        void destroy( )
            raises(NotEmpty);
        void list(in unsigned long how_many,
            out BindingList bl,
            out BindingIterator bi);
    };
};
```

26.1.1 NamingContext のメソッド

- virtual void **bind**(

  const Name& **_n**, CORBA::Object_ptr **_obj**) ;

  このメソッドは，指定された name を指定された Object にバインドします。このとき，最初の NameComponent に対応づけられたコンテキストを解決し，その後，次に示す Name を使用して新しいコンテキストにオブジェクトをバインドします。

```
Name[NameComponent2 ,...,NameComponent(n-1) ,NameComponentn ]
```

解決とバインドのこの再帰的なプロセスは，NameComponent$_{(n-1)}$に関連づけられたコンテキストが解決され，ネームとオブジェクトとの実際のバインドが格納されるまで続きます。パラメタ n がシンプルネームである場合には，obj は，この NamingContext 内の n にバインドされます。

- _n
  オブジェクトに指定するネームで初期化される Name 構造体
- _obj
  ネーミングされるオブジェクト

このメソッドでは，次の例外が発生します。

NotFound
: Name またはそのコンポーネントの一つが見つかりません。

CannotProceed
: シーケンスの NameComponent オブジェクトの一つが解決されていません。クライアントは，返されたネーミングコンテキストからオペレーションを継続できます。

InvalidName
: 指定された Name にはネームコンポーネントがありません。または ID フィールドに空文字列を指定したネームコンポーネントがあります。

AlreadyBound
: bind オペレーション，または bind_context オペレーションの Name は，NamingContext 内の別のオブジェクトにすでにバインドされています。

- virtual void **rebind**(
  const Name& **_n**, CORBA::Object_ptr **_obj**) ;
  このメソッドは，AlreadyBound 例外が発行されないという点を除いて，bind メソッドと同じです。指定された Name がすでに別のオブジェクトにバインドされている場合には，このバインドは，新しいバインドで置き換えられます。
  - _n
    オブジェクトに指定するネームで初期化される Name 構造体
  - _obj
    ネーミングされるオブジェクト

  このメソッドでは，次の例外が発生します。

  NotFound
  : Name またはそのコンポーネントの一つが見つかりません。

  CannotProceed
  : シーケンスの NameComponent オブジェクトの一つが解決されていません。クライアントは，返されたネーミングコンテキストからオペレーションを継続できます。

InvalidName

指定された Name にはネームコンポーネントがありません。または ID フィールドに空文字列を指定したネームコンポーネントがあります。

- virtual void **bind_context**(

const Name& _n, NamingContext_ptr _nc);

このメソッドは，指定された Name が，任意の VisiBroker ORB オブジェクトではなく，NamingContext に対応づけられるという点を除いて，bind メソッドと同じです。

- _n

  希望するネーミングコンテキスト名で初期化される Name 構造体。シーケンス内の最初の(n-1)個の NameComponent 構造体は，NamingContext を解決する必要があります。

- _nc

  バインドされる NamingContext オブジェクト

このメソッドでは，次の例外が発生します。

NotFound

Name またはそのコンポーネントの一つが見つかりません。

CannotProceed

シーケンスの NameComponent オブジェクトの一つが解決されていません。クライアントは，返されたネーミングコンテキストからオペレーションを継続できます。

InvalidName

指定された Name にネームコンポーネントがありません。または ID フィールドに空文字列を指定したネームコンポーネントがあります。

AlreadyBound

bind オペレーション，または bind_context オペレーションの Name は，NamingContext 内の別のオブジェクトにすでにバインドされています。

- virtual void **rebind_context**(

const Name& _n, NamingContext_ptr _nc);

このメソッドは，AlreadyBound 例外が発行されないという点を除いて，bind_context メソッドと同じです。指定された Name がすでに別のネーミングコンテキストにバインドされている場合には，このバインドは，新しいバインドで置き換えられます。

- _n

  オブジェクトに指定するネームで初期化される Name 構造体

- _nc

  リバインドされる NamingContext オブジェクト

このメソッドでは，次の例外が発生します。

NotFound

Name またはそのコンポーネントの一つが見つかりません。

CannotProceed

シーケンスの NameComponent オブジェクトの一つが解決されていません。クライアントは，返されたネーミングコンテキストからオペレーションを継続できます。

InvalidName

指定された Name にはネームコンポーネントがありません。または ID フィールドに空文字列を指定したネームコンポーネントがあります。

- virtual CORBA::Object _ptr **resolve**(

  const Name& _n);

  このメソッドは，指定された Name を解決し，オブジェクトリファレンスを返します。パラメタ n がシンプルネームである場合には，この NamingContext で解決されます。

  n がコンプレックスネームである場合には，最初の NameComponent に関連づけられたコンテキストを使用して解決されます。その後，新しいコンテキストを使用して次に示す Name を解決します。

  $$Name[NameComponent_2\ ,...,NameComponent_{(n-1)}\ ,NameComponent_n\ ]$$

  この再帰的なプロセスは，n 番目の NameComponent に関連づけられたオブジェクトが返されるまで続きます。

  - _n

    対象となるオブジェクトのネームで初期化される Name 構造体

  このメソッドでは，次の例外が発生します。

  NotFound

  Name またはそのコンポーネントの一つが見つかりません。

  CannotProceed

  シーケンスの NameComponent オブジェクトの一つが解決されていません。クライアントは，返されたネーミングコンテキストからオペレーションを継続できます。

  InvalidName

  指定された Name にはネームコンポーネントがありません。または ID フィールドに空文字列を指定したネームコンポーネントがあります。

- virtual void **unbind**(

  const Name& _n);

  このメソッドは，bind メソッドの逆で，指定された Name に対応しているバインドを削除します。

  - _n

    バインドの解除をしたいネームの Name 構造体

  このメソッドでは，次の例外が発生します。

  NotFound

  Name またはそのコンポーネントの一つが見つかりません。

CannotProceed

シーケンスの NameComponent オブジェクトの一つが解決されていません。クライアントは，返されたネーミングコンテキストからオペレーションを継続できます。

InvalidName

指定された Name にはネームコンポーネントがありません。または ID フィールドに空文字列を指定したネームコンポーネントがあります。

- virtual NamingContext_ptr new_context();

このメソッドは，新しいネーミングコンテキストを作成します。新しく作成されたコンテキストは，このオブジェクトと同じサーバ内で実装されます。新しいコンテキストは，初期状態ではどの Name にもバインドされていません。

- virtual NamingContext_ptr bind_new_context(

const Name& _n);

このメソッドは，新しいコンテキストを作成し，そのコンテキスト内で指定された Name にバインドします。

  - _n

    新しく作成された NamingContext オブジェクトに対して指定されたネームで初期化される Name 構造体

このメソッドでは，次の例外が発生します。

NotFound

Name またはそのコンポーネントの一つが見つかりません。

CannotProceed

シーケンスの NameComponent オブジェクトの一つが解決されていません。クライアントは，返されたネーミングコンテキストからオペレーションを継続できます。

InvalidName

指定された Name にはネームコンポーネントがありません。または ID フィールドに空文字列を指定したネームコンポーネントがあります。

AlreadyBound

bind オペレーション，または bind_context オペレーションの Name は，NamingContext 内の別のオブジェクトにすでにバインドされています。

- virtual void destroy();

このメソッドは，現在のネーミングコンテキストを非活性化します。このオブジェクトでオペレーションを呼び出そうとすると，CORBA::OBJECT_NOT_EXIST ランタイム例外が発生します。

このメソッドを使用する前に，unbind メソッドを使用して，NamingContext オブジェクトに対応してバインドされているすべての Name オブジェクトをバインド解除しておく必要があります。空でない NamingContext オブジェクトを破棄しようとすると，NotEmpty 例外が発生します。

- virtual void list(

CORBA::ULong _how_many,

BindingList_out _bl,

BindingIterator_out _bi);

このメソッドは，現在のコンテキストに含まれているすべてのバインドを返します。how_many パラメタで指定した数までの Name が，BindingList によって返されます。残りのバインドは，BindingIterator によって返されます。返された BindingList と BindingIterator を使用すると，ネームのリストを参照できます。BindingList の詳細については，「26.3　Binding と BindingList」を参照してください。

- _how_many
  返される Name の最大数
- _bl
  呼び出しプログラムに返される Name のリスト。リストのネームの数は how_many の値を超えません。
- _bi
  残りの Name を参照するためのイタレータ

26.2 NamingContextExt

```
class NamingContextExt :
                        public virtual NamingContext,
                        public virtual CORBA::Object
```

このクラスは，NamingContext インタフェースを継承したもので，文字列化した名前と URL を使用するときに必要なオペレーションを提供します。

IDL サンプル 26-2　NamingContextExt インタフェースの IDL 仕様

```
module CosNaming {
    interface NamingContextExt{
        typedef string StringName;
        typedef string Address;
        typedef string URLString;

        StringName to_string(in Name n)
            raises(InvalidName);
        Name to_name(in StringName sn)
            raises(InvalidName);

        exception InvalidAddress {};

        URLString to_url(in Address addr, in StringName sn)
            raises(InvalidAddress, InvalidName);
        Object resolve_str(in StringName n)
            raises(NotFound, CannotProceed,
                   InvalidName);
    };
};
```

26.2.1 NamingContextExt のメソッド

- virtual char* to_string(

  const Name& _n);

  このオペレーションは，指定した Name の文字列化表現を返します。

  - _n

    対象となるオブジェクトの名前で初期化される Name 構造体

  このメソッドでは，次の例外が発生します。

  InvalidName

  指定された Name にはネームコンポーネントがありません。または ID フィールドに空文字列を指定したネームコンポーネントがあります。

- virtual Name* to_name(

  const char* _sn);

このオペレーションは，指定した文字列化された名前の Name オブジェクトを返します。

- _sn
  オブジェクトの文字列化された名前

このメソッドでは，次の例外が発生します。

InvalidName
  指定された Name にはネームコンポーネントがありません。または ID フィールドに空文字列を指定したネームコンポーネントがあります。

- virtual char* to_url(
  const char* _addr, const char* _sn);
  このオペレーションは，_addr に指定した URL コンポーネントと，_sn に指定した文字列化名から，完全な URL 文字列を返します。
  - _addr
    「myhost.inprise.com:800」の形式の URL コンポーネント。このパラメタの指定を省略すると，自ホストを仮定します。
  - _sn
    文字列化したオブジェクト名

  このメソッドでは，次の例外が発生します。

  InvalidAddress
    addr パラメタに指定したアドレスが不正です。

  InvalidName
    指定された Name にはネームコンポーネントがありません。または ID フィールドに空文字列を指定したネームコンポーネントがあります。

- virtual CORBA::Object _ptr resolve_str(
  const char* _n);
  このオペレーションは，指定した文字列名の Name オブジェクトを返します。
  - _n
    文字列化したオブジェクト名

  このメソッドでは，次の例外が発生します。

  NotFound
    Name またはそのコンポーネントの一つが見つかりません。

  CannotProceed
    シーケンスの NameComponent オブジェクトの一つが解決されていません。クライアントは，返された NamingContext からオペレーションを継続できます。

InvalidName

指定された Name にはネームコンポーネントがありません。または ID フィールドに空文字列を指定したネームコンポーネントがあります。

26.3 Binding と BindingList

```
struct Binding
    class BindingList : private VISResource
```

Binding 構造体，BindingList クラス，および BindingIterator クラスは，NamingContext オブジェクトの中に「名前とオブジェクトのバインド」を定義するために使用します。Binding 構造体は，名前とオブジェクトのペアを一組カプセル化します。binding_name フィールドは Name を表し，binding_type フィールドは VisiBroker ORB オブジェクトと NamingContext オブジェクトのどちらに Name をバインドするかを表します。

BindingList は，NamingContext オブジェクトに含まれる Binding 構造体のシーケンスです。BindingList を使用するプログラムの例については，サンプルとして提供されている examples/vbe/ins/pluggable_adapter ディレクトリを参照してください。

IDL サンプル 26-3　Binding 構造体の IDL 仕様

```
module CosNaming {
    enum BindingType {
        nobject,
        ncontext
    }
    struct Binding {
        Name binding_name;
        BindingType binding_type;
    };
    typedef sequence<Binding>BindingList;
};
```

26.4 BindingIterator

```
class BindingIterator : public virtual CORBA::Object
```

このオブジェクトは，クライアントアプリケーションが，NamingContext の list メソッドによって返されるこのオブジェクトを使用して，オブジェクトとネームのバインドを，数を指定しないで反復的に参照するために使用します。詳細については，「26.1.1 NamingContext のメソッド」の「virtual void list(CORBA::ULong _how_many, BindingList_out _bl, BindingIterator_out _bi)」を参照してください。BindingIterator を使用するプログラムの例については，サンプルとして提供されている examples/vbe/ins/pluggable_adapter ディレクトリを参照してください。

IDL サンプル 26-4 Binding 構造体の IDL 仕様

```
module CosNaming {
    interface BindingIterator {
        boolean next_one(out Binding b);
        boolean next_n(in unsigned long how_many,
                       out BindingList b);
        void destroy( );
    };
};
```

26.4.1 BindingIterator のメソッド

- virtual CORBA::Boolean next_one(
  Binding_out b_);
  このメソッドは，リストから次の Binding を返します。リストに次の Binding がない場合には CORBA::FALSE が返ります。それ以外の場合には CORBA::TRUE が返ります。
  - b_
    リストの次の Binding オブジェクト
- virtual CORBA::Boolean next_n(
  CORBA::ULong _how_many, BindingList_out _b);
  このメソッドは，リストから要求された Binding オブジェクトの数を含む BindingList を返します。リストに次の Binding がない場合には，返されるバインドの数が，要求された数よりも少ない場合があります。リストに次の Binding がない場合には CORBA::FALSE が返ります。それ以外の場合には CORBA::TRUE が返ります。
  - _how_many
    要求される Binding オブジェクトの最大数
  - _b
    要求された Binding オブジェクト数以下のオブジェクトが登録された BindingList
- virtual void destroy();

このメソッドは，オブジェクトを破棄し，オブジェクトに対応するメモリを解放します。このメソッドの呼び出しに失敗すると，メモリの使用量が増大します。

26.5　NamingContextFactory

```
class NamingContextFactory :
                        public virtual CORBA::Object
```

このクラスは，最初の NamingContext を実体化します。クライアントは，このオブジェクトにバインドし，create_context メソッドを使用して最初のコンテキストを作成できます。最初のコンテキストを作成したら，new_context メソッドを使用して，ほかのコンテキストを作成できます。new_context メソッドについては，「26.1.1　NamingContext のメソッド」の「virtual NamingContext_ptr new_context();」を参照してください。

ネーミングサービスが起動するときに，このネーミングコンテキストファクトリのインスタンスが作成されます。詳細については，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「ネーミングコンテキストファクトリ」の記述およびサンプルとして提供されている examples/vbe/ins/cluster_failover ディレクトリを参照してください。

一つのルートコンテキストを自動的に作成する NamingContextFactory の作成方法については，「26.6 ExtendedNamingContextFactory」を参照してください。

IDL サンプル 26-5　NamingContextFactory の IDL 仕様

```
module CosNaming {
    interface NamingContextFactory {
        NamingContext create_context( );
        oneway void shutdown( );
    };
};
```

26.5.1　NamingContextFactory のメソッド

- virtual CosNaming::NamingContextExt_ptr **create_context**();
  このメソッドによって，クライアントはネーミングコンテキストを作成します。ネーミングコンテキストにはルートコンテキストが指定されていません。そのため，NamingContextFactory を実体化しただけではネーミングコンテキストは作成されません。
- virtual ClusterManager_ptr **get_cluster_manager**();
  このメソッドは，クラスタを返します。
- virtual NamingContextList* **list_all_roots**(
  const char* _password);
  このメソッドは，ルートコンテキストの一覧を返します。
- virtual void **remove_stale_contexts** (
  const char* _password);

このメソッドは，クラスタの存在期間中に，クライアントがクラスタからメンバを削除できるようにします。

- virtual void **shutdown**();

  このメソッドによって，クライアントは安全にネーミングサービスを終了できます。サービスを同じログファイルで再開した場合，ファクトリは，前回終了した状態です。

26.6 ExtendedNamingContextFactory

```
class ExtendedNamingContextFactory :
                        public virtual NamingContextFactory,
                        public virtual CORBA::Object
```

このクラスは，NamingContextFactory インタフェースを拡張し，拡張ネーミングサービスの起動時にファクトリ内でデフォルトルートを作成できるようにします。詳細については，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「ネーミングサービスの使用」の記述を参照してください。

IDL サンプル 26-6　ExtendedNamingContextFactory の IDL 仕様

```
module CosNaming {
    interface ExtendedNamingContextFactory
                            :NamingContextFactory{
        NamingContext root_context( );
    };
};
```

26.6.1 ExtendedNamingContextFactory のメソッド

- virtual CosNaming::NamingContextExt_ptr root_context();
  このメソッドは，該当するオブジェクトが実体化したときに，自動的に生成されたルートネーミングコンテキストを返します。

27 ポータブルインタセプタインタフェースとクラス（C++）

この章では，OMG 標準規格で定義されたポータブルインタセプタのインタフェースとクラスの，Borland Enterprise Server VisiBroker でのインプリメンテーションについて，C++言語でのインタフェースを説明します。これらのインタフェースとクラスの詳細については，「OMG Final Adopted 仕様」を参照してください。また，この章で説明するインタフェースを使用する前に，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「ポータブルインタセプタの使用」の記述を参照してください。

27.1 概要

VisiBroker ORB は，拡張機能をプラグインするためのインタセプタである API を提供します。例えば，トランザクションとセキュリティのサポートなどが拡張機能の一例です。インタセプタは，VisiBroker ORB の内部にフックされています。これによって，VisiBroker ORB サービスは，VisiBroker ORB の通常の実行の流れを受け取れます。Borland Enterprise Server VisiBroker がサポートしているインタセプタには，次の 2 種類があります。

- ポータブルインタセプタ

  OMG が標準化したインタセプタです。これによって，異なるベンダの ORB 間で使用できる，ポータブルなインタセプタのコードを記述できます。

- VisiBroker 4.x のインタセプタ

  VisiBroker 4.x で定義された，Borland Enterprise Server VisiBroker 独自のインタセプタです。

VisiBroker 4.x インタセプタの詳細については，マニュアル「Borland Enterprise Server VisiBroker デベロッパーズガイド」の「VisiBroker 4.x インタセプタの使用」の記述を参照してください。また，「28. VisiBroker 4.x インタセプタおよびオブジェクトラッパーのインタフェースとクラス（C++）」を参照してください。

ポータブルインタセプタには，次の 2 種類があります。

- リクエストインタセプタ

  リクエストインタセプタを使用すると，VisiBroker ORB サービスはクライアントとサーバの間でコンテキスト情報の受け渡しができるようになります。リクエストインタセプタには，クライアントリクエストインタセプタとサーバリクエストインタセプタの 2 種類があります。

- IOR インタセプタ

  IOR インタセプタを使用すると，VisiBroker ORB サービスは，サーバまたはオブジェクトの ORB サービス関連の機能を定義する IOR に情報を登録できるようになります。例えば，SSL などのセキュリティサービスは，自身のタグ付きコンポーネントを IOR に登録できるようになります。これによって，そのコンポーネントを認識するクライアントは，そのコンポーネントの情報に基づいて，サーバとの間にコネクションを確立できます。

ポータブルインタセプタの詳細については，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「ポータブルインタセプタの使用」の記述を参照してください。

27.2 ClientRequestInfo

```
class PortableInterceptor::ClientRequestInfo :
                        public virtual RequestInfo,
                        public CORBA::LocalObject
```

このクラスは，RequestInfo から派生したクラスで，クライアント側のインタセプトポイントに渡されます。ClientRequestInfo のメソッドは，一部のインタセプトポイントでは有効ではありません。

次の表に，属性またはメソッドの有効性を示します。無効の属性またはメソッドにアクセスすると，標準マイナーコード 14 の BAD_INV_ORDER 例外が発生します。

表 27-1 ClientRequestInfo の有効性（C++）

	send_request	send_poll	receive_reply	receive_exception	receive_other
request_id	○	○	○	○	○
operation	○	○	○	○	○
arguments	○※1	×	○	×	×
exceptions	○	×	○	○	○
contexts	○	×	○	○	○
operation_context	○	×	○	○	○
result	×	×	○	×	×
response_expected	○	○	○	○	○
sync_scope	○	×	○	○	○
reply_status	×	×	○	○	○
forward_reference	×	×	×	×	○※2
get_slot	○	○	○	○	○
get_request_service_context	○	×	○	○	○
get_reply_service_context	×	×	○	○	○
target	○	○	○	○	○
effective_target	○	○	○	○	○
effective_profile	○	○	○	○	○
received_exception	×	×	×	○	×
received_exception_id	×	×	×	○	×
get_effective_component	○	×	○	○	○

	send_request	send_poll	receive_reply	receive_exception	receive_other
get_effective_components	○	×	○	○	○
get_request_policy	○	×	○	○	○
add_request_service_context	○	×	×	×	×

（凡例）

○：有効，×：無効

注※1

ClientRequestInfo が send_request()に渡される場合，in，inout，または out の各引数のリストにエントリがありますが，使用できるのは inout 引数と out 引数だけです。

注※2

reply_status()が LOCATION_FORWARD を返さない場合，この属性にアクセスすると，標準マイナーコード 14 の BAD_INV_ORDER 例外が発生します。

27.2.1　インクルードファイル

このクラスを使用するときは，PortableInterceptor_c.hh ファイルをインクルードしてください。

27.2.2　ClientRequestInfo のメソッド

- virtual CORBA::Object_ptr **target**() = 0;

  このメソッドは，クライアントがオペレーションを実行するために呼び出したオブジェクトを返します。effective_target()メソッドも参照してください。

- virtual CORBA::Object_ptr **effective_target** () = 0;

  このメソッドは，実際のオペレーション呼び出し元オブジェクトを返します。reply_status()が LOCATION_FORWARD を返す場合，後続のリクエストに対して target が返す内容は変わりませんが，effective_target は，フォワードされた IOR を返します。

- virtual IOP::TaggedProfile* **effective_profile**() = 0;

  このメソッドは，IOP::TaggedProfile の形式でプロファイルを返します。返されたプロファイルは，リクエストの送信に使用されます。このオペレーションのオブジェクトにロケーションフォワードが発生し，そのオブジェクトのプロファイルが変更された場合，このプロファイルが探索結果のプロファイルとなります。

- virtual CORBA::Any* **received_exception**() = 0;

  このメソッドは，クライアントに返される例外を格納するデータを CORBA::Any の形式で返します。

  CORBA::Any に挿入できないユーザ例外の場合，例えば，未知の例外やバインディングで TypeCode が提供されない場合などは，この属性は，標準マイナーコード 1 の UNKNOWN システム例外を格納

する CORBA::Any となります。ただし，この例外の RepositoryId は，received_exception_id 属性に使用できます。

- virtual char* **received_exception_id**() = 0;
  このメソッドは，クライアントに返される received_exception の ID を返します。
- virtual IOP::TaggedComponent* **get_effective_component**(
  CORBA::ULong **_id**) = 0;
  このメソッドは，リクエストに対して選択されたプロファイルに指定された ID を持つ IOP::TaggedComponent を返します。
  指定された ID のコンポーネントが複数ある場合に，どのコンポーネントを返すかは定義されていません。指定された ID のコンポーネントが複数ある場合は，このメソッドの代わりに get_effective_components()が呼び出されます。指定された ID のコンポーネントがない場合は，標準マイナーコード 28 の BAD_PARAM 例外が発生します。
  - _id
    返されるコンポーネントの ID
- virtual IOP::TaggedComponentSeq* **get_effective_components**(
  CORBA::ULong **_id**) = 0;
  このメソッドは，リクエストに対して選択されたプロファイルに指定された ID を持つタグ付きコンポーネントをすべて返します。IOP::TaggedComponentSeq の形式で返されます。指定された ID のコンポーネントがない場合は，標準マイナーコード 28 の BAD_PARAM 例外が発生します。
  - _id
    返されるコンポーネントの ID
- virtual CORBA::Policy_ptr **get_request_policy**(
  CORBA::ULong **_type**) = 0;
  このメソッドは，オペレーションに対して有効な所定のポリシーを返します。指定した型に ORB が対応していない，または指定した型のポリシーオブジェクトがこのオブジェクトに対応づけられていないために，ポリシーの型が無効となる場合は，標準マイナーコード 2 の INV_POLICY 例外が発生します。
  - _type
    返されるポリシーを指定するポリシーの型
- virtual void **add_request_service_context**(
  const IOP::ServiceContext& **_service_context**,
  CORBA::Boolean **_replace**) = 0;
  このメソッドを使用すると，インタセプタで一つ以上のサービスコンテキストをリクエストに登録できます。サービスコンテキストの順序の宣言はありません。登録された順序でサービスコンテキストが表示される場合も，されない場合もあります。
  - _service_context
    リクエストに登録する IOP::ServiceContext
  - _replace

指定した ID のサービスコンテキストがすでに存在する場合のメソッドの動作。false の場合は，標準マイナーコード 15 の BAD_INV_ORDER 例外が発生します。true の場合は，既存のコンテキストが新しいコンテキストに置き換えられます。

27.3 ClientRequestInterceptor

```
class PortableInterceptor::ClientRequestInterceptor :
                        public virtual Interceptor,
                        public virtual CORBA::LocalObject
```

このクラスは，ユーザ定義クライアント側インタセプタを継承するときに使用します。
ClientRequestInterceptor インスタンスは，VisiBroker ORB に登録されます。詳細については，「27.14 ORBInitializer」を参照してください。

27.3.1 インクルードファイル

このクラスを使用するときは，PortableInterceptor_c.hh ファイルをインクルードしてください。

27.3.2 ClientRequestInterceptor のメソッド

- virtual void send_request(
  ClientRequestInfo_ptr _ri) = 0;
  このインタセプトポイントを使用すると，リクエストがサーバに送信される前に，リクエスト情報の照会とサーバコンテキストの修正が，インタセプタでできます。
  このインタセプトポイントでは，システム例外が発生する場合があります。システム例外が発生した場合，ほかのインタセプタの send_request()インタセプトポイントは呼び出されません。フロースタックからインタセプタが取り出され，そのインタセプタのインタセプトポイントが呼び出されます。
  このインタセプトポイントでは，ForwardRequest 例外を発生させることができます。インタセプトでこの例外が発生した場合，ほかのインタセプタの send_request メソッドは呼び出されません。フロースタックからインタセプタが取り出され，そのインタセプタの receive_other()インタセプトポイントが呼び出されます。ForwardRequest 例外については，「27.9 ForwardRequest」を参照してください。
  - _ri
    インタセプタが使用する ClientRequestInfo インスタンス
- virtual void send_poll(
  ClientRequestInfo_ptr _ri) = 0;
  このインタセプトポイントを使用すると，時間非依存呼び出し（TII）によるポーリング get reply シーケンス中に，インタセプタで情報を照会できます。
  ただし，VisiBroker ORB は TII に対応していないため，この send_poll()インタセプトポイントが呼び出されることはありません。
  - _ri
    インタセプタが使用する ClientRequestInfo インスタンス

- virtual void **receive_reply**(
  ClientRequestInfo_ptr _ri) = 0;
  このインタセプトポイントを使用すると，サーバから応答が返されてから，制御がクライアントに戻るまでの間に，インタセプタで応答の情報を照会できます。
  このインタセプトポイントでは，システム例外が発生する場合があります。システム例外が発生した場合，ほかのインタセプタの receive_reply()メソッドは呼び出されません。フロースタックからインタセプタが取り出され，そのインタセプタの receive_exception()インタセプトポイントが呼び出されます。
  - _ri
    インタセプタが使用する ClientRequestInfo インスタンス
- virtual void **receive_exception**(
  ClientRequestInfo_ptr _ri) = 0;
  このインタセプトポイントは例外が発生したときに呼び出されます。これによって，例外の情報がクライアントに通知される前にインタセプタで例外の情報を照会できます。
  このインタセプトポイントでは，システム例外が発生する場合があります。システム例外が発生した場合は，フロースタックから取り出された一連のインタセプタが，receive_exception()呼び出し時に受け取る例外が変更されます。クライアントに通知される例外は，インタセプタが通知する最後の例外です。ほかのインタセプタが例外を変更しなければ，元の例外が通知されます。
  また，このインタセプトポイントでは，ForwardRequest 例外を発生させることができます。インタセプタでこの例外が発生した場合，ほかのインタセプタの receive_exception()インタセプトポイントは呼び出されません。フロースタックからインタセプタが取り出され，そのインタセプタの receive_other()インタセプトポイントが呼び出されます。ForwardRequest 例外については，「27.9 ForwardRequest」を参照してください。
  - _ri
    インタセプタが使用する ClientRequestInfo インスタンス
- virtual void **receive_other**(
  ClientRequestInfo_ptr _ri) = 0;
  このインタセプトポイントを使用すると，リクエストの結果が，正常な応答でも例外でもない場合に使用できる情報をインタセプタで照会できます。それは，リクエストがリトライになる場合（例えば，LOCATION_FORWARD 状態で GIOP Reply を受信した場合）や，非同期呼び出し時にリクエストの直後に応答が返されないで，制御がクライアントに戻って終了インタセプトポイントが呼び出される場合などです。
  リクエストがリトライになる場合は，適用されているポリシーによって，リトライ指示直後に新しいリクエストが発行されるときと発行されないときがあります。新しいリクエストが発行されるときは，このリクエストが新しいリクエストである間は，インタセプタに関して，元のリクエストとリトライとの間に相関性があります。制御がクライアントに戻らないため，リクエストをスコープとする PortableInterceptor::Current は，元のリクエストの場合もリトライリクエストの場合も同一です。詳細については，「27.6　Current」を参照してください。このインタセプトポイントでは，システム例外が発生する場合があります。システム例外が発生した場合，ほかのインタセプタの receive_other()

インタセプトポイントは呼び出されません。フロースタックからインタセプタが取り出され，そのインタセプタの receive_exception()インタセプトポイントが呼び出されます。

また，このインタセプトポイントでは，ForwardRequest 例外を発生させることができます。インタセプタでこの例外が発生した場合，ForwardRequest 例外が提供する新しい情報で一連のインタセプタの receive_other()メソッドが呼び出されます。ForwardRequest 例外については，「27.9 ForwardRequest」を参照してください。

- _ri
  インタセプタが使用する ClientRequestInfo インスタンス

27.4 Codec

```
class IOP::Codec
```

ORB サービスが使用する IOR コンポーネントとサービスコンテキストデータの形式は，IDL データ型を持つ CDR カプセル化エンコーディングインスタンスとして定義される場合があります。Codec は，IDL データ型と CDR カプセル化表現との間でコンポーネントを受け渡すための機能を持っています。

Codec は CodecFactory から取得します。CodecFactory は，
ORB::resolve_initial_references("CodecFactory")の呼び出しで取得できます。

27.4.1 インクルードファイル

このクラスを使用するときは，IOP_c.hh ファイルをインクルードしてください。

27.4.2 Codec のメンバ

- class Codec::**InvalidTypeForEncoding** : public CORBA::UserException
  この例外は，エンコーディングに不正な型が指定されている場合に，encode()または encode_value()が出力します。
- class Codec::**FormatMismatch** : public CORBA::UserException
  この例外は，オクテットシーケンスのデータを CORBA::Any にデコードできない場合に，decode()または decode_value()が出力します。
- class Codec::**TypeMismatch** : public CORBA::UserException
  この例外は，指定した TypeCode とオクテットシーケンスが適合しない場合に，decode_value()が出力します。

27.4.3 Codec のメソッド

- virtual CORBA::OctetSequence* **encode**(
  const CORBA::Any& **_data**) = 0;
  このメソッドは，この Codec に適用されているエンコーディング形式に基づいて，CORBA::Any の形式で指定したデータをオクテットシーケンスに変換します。このオクテットシーケンスには，TypeCode と型のデータの両方が格納されます。このオペレーションでは，InvalidTypeForEncoding 例外が発生する場合があります。
  - _data
    オクテットシーケンスに変換する，CORBA::Any 形式のデータ

- virtual CORBA::Any* **decode**(
  const CORBA::OctetSequence& **_data**) = 0;
  このメソッドは，この Codec に適用されているエンコーディング形式に基づいて，指定したオクテットシーケンスを CORBA::Any オブジェクトにデコードします。
  オクテットシーケンスを CORBA::Any にデコードできない場合，このメソッドは，FormatMismatch 例外を出力します。
  - _data
    CORBA::Any に変換する，オクテットシーケンス形式のデータ
- virtual CORBA::OctetSequence* **encode_value**(
  const CORBA::Any& **_data**) = 0;
  このメソッドは，この Codec に適用されているエンコーディング形式に基づいて，指定した CORBA::Any オブジェクトをオクテットシーケンスに変換します。CORBA::Any のデータだけをエンコードします。TypeCode のデータはエンコードしません。
  このオペレーションでは，InvalidTypeForEncoding 例外が発生する場合があります。
  - _data
    エンコード済み CORBA::Any のデータを格納するオクテットシーケンス
- virtual CORBA::Any* **decode_value**(
  const CORBA::OctetSequence& **_data**,
  CORBA::TypeCode_ptr **_tc**) = 0;
  このメソッドは，指定した TypeCode と，この Codec に適用されているエンコーディング形式に基づいて，指定したオクテットシーケンスを CORBA::Any にデコードします。
  オクテットシーケンスを CORBA::Any にデコードできない場合，このメソッドは，FormatMismatch 例外を出力します。
  - _data
    CORBA::Any にデコードする，オクテットシーケンス形式のデータ
  - _tc
    データのデコードに使用する TypeCode

27.5 CodecFactory

```
class IOP::CodecFactory
```

このクラスは，Codec を取得するときに使用します。CodecFactory は，ORB::resolve_initial_references("CodecFactory")の呼び出しで取得できます。

27.5.1 インクルードファイル

このクラスを使用するときは，IOP_c.hh ファイルをインクルードしてください。

27.5.2 CodecFactory のメンバ

- class CodecFactory::**UnknownEncoding** : public CORBA::UserException

  この例外は，CodecFactory で Codec を生成できない場合に発生します。「27.5.3 CodecFactory のメソッド」の create_codec()メソッドを参照してください。

27.5.3 CodecFactory のメソッド

- virtual Codec_ptr **create_codec**(

  const Encoding& **_enc**) = 0;

  このメソッドは，指定したエンコーディング形式の Codec を生成します。

  指定したエンコーディング形式の Codec をファクトリが生成できない場合は，UnknownEncoding 例外を出力します。

  - _enc

    Codec の生成に使用するエンコーディング形式

27.6 Current

```
class PortableInterceptor::Current:
                        public virtual CORBA::Current,
                        public virtual CORBA::LocalObject
```

Current クラスは，単なるスロットテーブルです。このテーブルのスロットは，スロットのコンテキストと，リクエストまたは応答のサービスコンテキストとの間で，各サービスがコンテキストデータを受け渡すために使用されます。

Current を使用する各サービスは，初期化時に確保したスロットを，リクエストと応答の処理中に使用します。「27.15.3 ORBInitInfo のメソッド」の allocate_slot_id()メソッドを参照してください。

Current は，ORB::resolve_initial_references("PICurrent")の呼び出しで取得されます。

RequestInfo オブジェクトの get_slot()メソッドを使用すると，インタセプトポイントの内側から，スレッドスコープからリクエストスコープへ移動した Current のデータを使用できます。

27.6.1 インクルードファイル

このクラスを使用するときは，PortableInterceptor_c.hh ファイルをインクルードしてください。

27.6.2 Current のメソッド

- virtual CORBA::Any* **get_slot**(
  CORBA::ULong _id);
  このメソッドを使用すると，サービスは，PICurrent に設定したスロットデータを CORBA::Any 形式で取得できます。
  設定されていないスロットを指定した場合は，tk_null の TCKind 値を持つタイプコードを格納する CORBA::Any を返します。
  割り当てられていないスロットに対して get_slot()を呼び出すと，InvalidSlot 例外が発生します。
  ORB イニシャライザの内側から get_slot()を呼び出すと，マイナーコード 14 の BAD_INV_ORDER 例外が発生します。ORB イニシャライザについては，「27.14 ORBInitializer」を参照してください。
  - _id
    データを取り出すスロットの SlotId
- virtual void **set_slot**(
  CORBA::ULong _id, const CORBA::Any& _data);
  サービスは，このメソッドを使用して，スロットにデータを CORBA::Any オブジェクトの形式で設定します。

スロットにすでにデータが設定されている場合，既存データは上書きされます。

割り当てられていないスロットに対して set_slot()を呼び出すと，InvalidSlot 例外が発生します。

ORB イニシャライザの内側から set_slot()を呼び出すと，マイナーコード 14 の BAD_INV_ORDER 例外が発生します。ORB イニシャライザについては，「27.14　ORBInitializer」を参照してください。

- _id
  データを設定するスロットの SlotId
- _data
  指定したスロットに設定する，CORBA::Any オブジェクト形式のデータ

27.7 Encoding

```
struct IOP::Encoding
```

この構造体は，Codec のエンコーディング形式を定義します。CDR カプセル化エンコーディングなどのエンコード形式，メジャーバージョン，およびマイナーバージョンを定義します。

次のエンコード形式に対応しています。

- ENCODING_CDR_ENCAPS バージョン 1.0
- ENCODING_CDR_ENCAPS バージョン 1.1
- ENCODING_CDR_ENCAPS バージョン 1.2
- GIOP の将来のバージョンすべてに対応する ENCODING_CDR_ENCAPS

27.7.1 インクルードファイル

この構造体を使用するときは，IOP_c.hh ファイルをインクルードしてください。

27.7.2 Encoding のメンバ

- CORBA::Short format;
  このメンバは，Codec のエンコーディング形式を保持します。
- CORBA::Octet major_version;
  このメンバは，Codec のメジャーバージョン番号を保持します。
- CORBA::Octet minor_version;
  このメンバは，Codec のマイナーバージョン番号を保持します。

27.8 ExceptionList

```
class Dynamic::ExceptionList
```

このクラスは，RequestInfo クラスの exceptions()メソッドが返す例外情報を保持するために使用する，CORBA::TypeCode 型の可変長配列のインプリメンテーションです。ExceptionList の長さは，実行時に使用できます。

詳細については，「27.19.2 RequestInfo のメソッド」の exceptions()メソッドを参照してください。

27.8.1 インクルードファイル

このクラスを使用するときは，Dynamic_c.hh ファイルをインクルードしてください。

27.9 ForwardRequest

```
class PortableInterceptor::ForwardRequest : public CORBA::UserException
```

インタセプタは，ForwardRequest 例外を使用して，新規オブジェクトを指定し，リクエストのリトライを ORB に指示できます。インタセプタからの ForwardRequest 例外を ORB が受信した場合だけ，リトライが指示されます。それ以外の場合に ForwardRequest 例外が発生すると，その例外は，ユーザ例外と同様に ORB を介して渡されます。

インタセプタの呼び出しに対してインタセプタから ForwardRequest 例外が出力された場合，そのインタセプトポイントに，ほかのインタセプタは呼び出されません。フロースタックに蓄積されたインタセプタに対応する終了インタセプトポイント（クライアントの receive_other，またはサーバの send_other()）が呼び出されます。receive_other()および send_other()の中では，reply_status()は LOCATION_FORWARD を返します。

27.9.1 インクルードファイル

このクラスを使用するときは，PortableInterceptor_c.hh ファイルをインクルードしてください。

27.10 Interceptor

```
class PortableInterceptor::Interceptor : public virtual CORBA::LocalObject
```

Interceptor は，すべてのインタセプタの派生元となるベースクラスです。

27.10.1 インクルードファイル

このクラスを使用するときは，PortableInterceptor_c.hh ファイルをインクルードしてください。

27.10.2 Interceptor のメソッド

- virtual char* name() = 0;

  このメソッドは，インタセプタの名前を返します。個々のインタセプタには，インタセプタを並べ替えるための名前を付けられます。インタセプタ型ごとに，指定した名前を持つ唯一のインタセプタを VisiBroker ORB に登録できます。空文字列を名前に設定することで，インタセプタを匿名にすることもできます。

  VisiBroker ORB には，幾つでも匿名のインタセプタを登録できます。

- virtual void destroy() = 0;

  このメソッドは，ORB::destroy()の実行中に呼び出されます。アプリケーションによって ORB::destroy()が呼び出されると，VisiBroker ORB は次のように処理します。

  1. 処理中のリクエストがすべて完了するまで待ちます。
  2. Interceptor::destroy()メソッドをインタセプタごとに呼び出します。
  3. VisiBroker ORB のデストラクトを完了します。

  デストラクト中の VisiBroker ORB に実装されているオブジェクトのオブジェクトリファレンスに，Interceptor::destroy()の中からメソッド呼び出しをした場合の動作は保証できません。ただし，デストラクト中ではない VisiBroker ORB に，実装されているオブジェクトのメソッド呼び出しはできます。つまり，デストラクト中の VisiBroker ORB は，クライアントとしては使用できますが，サーバとしては使用できません。

27.11 IORInfo

```
class PortableInterceptor::IORInfo : public virtual CORBA::LocalObject
```

IORInfo クラスによって，サーバ側の ORB サービスはコンポーネントを追加したり，IOR 構築中に適用可能なポリシーへアクセスしたりできるようになります。

ORB は，このクラスの ORB のインプリメンテーションのインスタンスを IORInterceptor::establish_components()にパラメタとして渡します。

次の表に，IORInterceptor に定義されたメソッドの，IORInfo での属性またはメソッドの有効性を示します。IORInfo の属性またはメソッドに不当な呼び出しをすると，標準マイナーコード 14 の BAD_INV_ORDER 例外が発生します。

表 27-2 IORInfo の有効性（C++）

	establish_components	components_established
get_effective_policy	○	○
add_ior_component	○	×
add_ior_component_to_profile	○	×
manager_id	○	○
state	○	○
adapter_template	×	○
current_factory	×	○

（凡例）

○：有効，×：無効

27.11.1 インクルードファイル

このクラスを使用するときは，PortableInterceptor_c.hh ファイルをインクルードしてください。

27.11.2 IORInfo のメソッド

- virtual CORBA::Policy_ptr **get_effective_policy**(
  CORBA::ULong _type) = 0;
  ORB サービスのインプリメンテーションでは，get_effective_policy()メソッドを呼び出すことで，特定の型のどのサーバ側ポリシーが構築中の IOR に適用されているかを調べられます。構築中の IOR が，POA を使用して実装されたオブジェクトの IOR である場合，その POA を生成した

PortableServer::POA::create_POA()呼び出しで渡された Policy オブジェクトはすべて，get_effective_policy でアクセスできます。

指定した型のポリシーを ORB が認識していない場合，このメソッドは標準マイナーコード 3 の INV_POLICY 例外を出力します。

- _type
  取得するポリシーの型を指定した CORBA::PolicyType

- virtual void **add_ior_component**(
  const IOP::TaggedComponent& **_a_component**) = 0;
  このメソッドは，IOR 構築時にインクルードされるタグ付きコンポーネントのセットにメンバを追加するときに，establish_components()から呼び出されます。すべてのプロファイルにコンポーネントのセットがインクルードされます。
  同じコンポーネント ID のコンポーネントを複数存在させることもできます。
  - _a_component
    追加する IOP::TaggedComponent
- virtual void **add_ior_component_to_profile**(
  const IOP::TaggedComponent&**_a_component**,
  CORBA::ULong **_profile_id**) = 0;
  このメソッドは，IOR 構築時にインクルードされるタグ付きコンポーネントのセットにメンバを追加するときに，establish_components()から呼び出されます。指定したプロファイルにコンポーネントのセットがインクルードされます。
  指定したプロファイル ID が既存のプロファイルを定義していない場合，およびプロファイルにコンポーネントを追加できない場合，標準マイナーコード 27 の BAD_PARAM 例外が発生します。
  - _a_component
    追加する IOP::TaggedComponent
  - _profile_id
    コンポーネントを追加するプロファイルの IOP::ProfileId
- virtual CORBA::Long **manager_id**() = 0;
  このメソッドは，アダプタのマネージャへの不透明なハンドルを提供する属性を返します。このメソッドは，同じアダプタマネージャに管理されているアダプタの状態変更を通知するために使用します。
- virtual CORBA::Short **state**() = 0;
  このメソッドは，アダプタの現在の状態を返します。状態として，HOLDING，ACTIVE，DISCARDING，INACTIVE，または NON_EXISTENT のどれかを返します。
- virtual ObjectReferenceTemplate_ptr **adapter_template**() = 0;
  IOR インタセプタが呼び出されたときは常に，このメソッドがオブジェクトリファレンスのテンプレートを取得するための属性を返します。オブジェクトリファレンスのテンプレートを直接作成する方法は標準では提供されません。adapter_template()が返す値は，add_component()と add_component_to_profile()の IOR インタセプタ呼び出しのために作成されるテンプレート，およ

びアダプタポリシーです。adapter_template()が返す値は，オブジェクトアダプタが存続している間は変更されません。

- virtual ObjectReferenceFactory_ptr current_factory() = 0;

  このメソッドが返す属性を使用して，アダプタがオブジェクトリファレンスを作成するときに使用するファクトリにアクセスできます。current_factory()が返す初期値は adapter_template 属性と同じ値ですが，current_factory にほかのファクトリを設定することで変更できます。オブジェクトアダプタが作成するオブジェクトリファレンスは，すべて current_factory の make_object()メソッドを呼び出して作成する必要があります。

- virtual void current_factory(

  ObjectReferenceFactory_ptr _current_factory) = 0;

  このメソッドでは，current_factory 属性を設定します。アダプタが使用する current_factory 属性の値を設定できるのは，components_established メソッドの呼び出し時だけです。

  - _current_factory

    設定対象の current_factory オブジェクト

27.12 IORInfoExt

```
class PortableInterceptorExt::IORInfoExt:
                              public virtual PortableInterceptor::IORInfo,
                              public virtual CORBA::LocalObject
```

このクラスでは，Borland Enterprise Server VisiBroker のポータブルインタセプタを拡張して，POA をスコープとするサーバリクエストインタセプタをインストールします。IORInfoExt クラスは，IORInfo インタフェースから継承します。IORInfoExt クラスは，POA をスコープとするサーバリクエストインタセプタに対応するための拡張メソッドを提供します。

27.12.1 インクルードファイル

このクラスを使用するときは，PortableInterceptorExt_c.hh ファイルをインクルードしてください。

27.12.2 IORInfoExt のメソッド

- virtual void add_server_request_interceptor(
  ServerRequestInterceptor_ptr _interceptor) = 0;
  このメソッドは，POA をスコープとするサーバ側リクエストインタセプタをサーバに追加するときに使用します。
  - _interceptor
    追加する ServerRequestInterceptor
- virtual char* full_poa_name();
  このメソッドは，POA のフルネームを返します。

27.13 IORInterceptor

```
class PortableInterceptor::IORInterceptor : public virtual Interceptor,
                                            public virtual CORBA::LocalObject
```

ポータブル ORB サービスインプリメンテーションでは，クライアントの ORB サービスインプリメンテーションが正しく機能するように，必要に応じてサーバまたはオブジェクトの ORB サービスに関連する機能の定義情報をオブジェクトリファレンスに追加する必要があります。それは，IORInterceptor クラスと IORInfo クラスが提供しています。IOR インタセプタは，IOR にあるプロファイルにタグ付きコンポーネントを設定するために使用します。

27.13.1 インクルードファイル

このクラスを使用するときは，PortableInterceptor_c.hh ファイルをインクルードしてください。

27.13.2 IORInterceptor のメソッド

- virtual void establish_components(
  IORInfo_ptr _info) = 0;
  サーバ側 ORB は，特定のオブジェクトリファレンスの一つ以上のプロファイルにインクルードするコンポーネントの一覧を生成するときに，登録済み IORInterceptor インスタンスすべての establish_components()メソッドを呼び出します。このメソッドは，オブジェクトリファレンスごとに呼び出す必要はありません。POA の場合は，POA::create_POA()呼び出し時に毎回呼び出されます。ほかのアダプタの場合は，通常，アダプタの初期化時に呼び出されます。
  この段階では，アダプタテンプレートは使用できません。それは，アダプタテンプレートに必要な情報（コンポーネント）がまだ作成されていないためです。
  - _info
    適用可能なポリシーを照会し，生成された IOR にインクルードするコンポーネントを追加するときに，ORB サービスが使用する IORInfo インスタンス

27.14 ORBInitializer

```
class PortableInterceptor::ORBInitializer : public virtual CORBA::LocalObject
```

ORBInitializer クラスをインプリメントする ORBInitializer オブジェクトを登録すると，対応するインタセプタが登録されます。ORB は，初期化時に登録済み ORBInitializer をすべて呼び出し，それぞれに ORBInitInfo オブジェクトを渡します。この ORBInitInfo オブジェクトは，ORBInitializer のインタセプタの登録に使用されます。

27.14.1 インクルードファイル

このクラスを使用するときは，PortableInterceptor_c.hh ファイルをインクルードしてください。

27.14.2 ORBInitializer のメソッド

- virtual void pre_init(
  ORBInitInfo_ptr _info) = 0;
  このメソッドは，ORB の初期化中に呼び出されます。インタセプタによって登録された初期サービスが，ほかのインタセプタに使用される場合は，事前に ORBInitInfo::register_initial_reference()呼び出しで，その初期サービスが登録されます。
  - _info
    インタセプタを登録するための初期化属性とメソッド
- virtual void post_init(
  ORBInitInfo_ptr _info) = 0;
  このメソッドは，ORB の初期化中に呼び出されます。サービスが初期化の一部として初期リファレンスを解決する必要がある場合は，サービスは，この段階ですべての初期リファレンスを使用できるとみなします。
  post_init()メソッド呼び出しは，ORB の初期化の最後の処理ではありません。post_init()のあとに，登録済みインタセプタの一覧を ORB にアタッチするのが最後の処理です。したがって，post_init()呼び出し中は，ORB にはインタセプタが含まれていません。ORB を介する呼び出しが post_init()の内部で実行された場合，その呼び出しに対するリクエストインタセプタは呼び出されません。
  同様に，IOR を作成するメソッドが実行された場合も，IOR インタセプタは呼び出されません。
  - _info
    インタセプタを登録するための初期化属性とメソッド

27.15 ORBInitInfo

```
class PortableInterceptor::ORBInitInfo : public virtual CORBA::LocalObject
```

このクラスは，インタセプタを登録するために，ORBInitializer オブジェクトに渡されます。

27.15.1 インクルードファイル

このクラスを使用するときは，PortableInterceptor_c.hh ファイルをインクルードしてください。

27.15.2 ORBInitInfo のメンバ

- class DuplicateName : public CORBA::UserException;
  インタセプタ型ごとに，指定した名前を持つ唯一のインタセプタを ORB に登録できます。既存のインタセプタと同じ名前でインタセプタを登録しようとすると，DuplicateName 例外が発生します。
  空文字列を名前に設定することで，インタセプタを匿名にすることもできます。ORB には，幾つでも匿名のインタセプタを登録できます。このため，匿名のインタセプタを登録する場合は，DuplicateName 例外は発生しません。
- class InvalidName : public CORBA::UserException
  この例外は，register_initial_reference()と resolve_initial_references()で発生します。
  register_initial_reference()で InvalidName 例外が発生するのは次の場合です。
  - このメソッドが空文字列 ID で呼び出された場合
  - このメソッドがすでに登録されている ID で呼び出された場合（OMG 定義済みの既定の名前で呼び出された場合も含みます）

  resolve_initial_references()では，解決する名前が不正な場合に InvalidName 例外が発生します。

27.15.3 ORBInitInfo のメソッド

- virtual CORBA::StringSequence* arguments() = 0;
  このメソッドは，ORB_init()に渡された引数を返します。この引数には，ORB の引数が含まれる場合も，含まれない場合もあります。
- virtual char* orb_id() = 0;
  このメソッドは，初期化中の ORB の ID を返します。
- virtual IOP::CodecFactory_ptr codec_factory() = 0;
  このメソッドは，IOP::CodecFactory を返します。通常，CodecFactory は，ORB::resolve_initial_references("CodecFactory")を呼び出して取得しますが，その時点では，ORB

はまだ使用できません。しかし，サービスコンテキスト処理時などに，インタセプタが Codec を必要とするため，Codec を取得する手段が ORB の初期化時に必要となります。

- virtual void **register_initial_reference**(
  const char* **_id**, CORBA::Object_ptr **_obj**) = 0;
  例えば，このメソッドが ID「Y」とオブジェクト「YY」で呼び出されたあとに，register_initial_reference メソッドが再び呼び出されると，オブジェクト「YY」が返されます。
  このメソッドの機能は，ORB::register_initial_reference()と同一です。このメソッドは，ORB が完全に初期化されていないためにまだ使用できない場合に，インタセプタの登録の一部として初期リファレンスが必要となるときに使用します。なお，このメソッドと ORB::register_initial_reference()との違いは，ORB のメソッドのバージョンは PIDL（CORBA::ORB::ObjectId と CORBA::ORB::InvalidName）を使用し，インタフェースのバージョンは，インタフェースに定義された IDL を使用する点です。構文に違いはありません。
  このメソッドが空文字列 ID で呼び出された場合，またはこのメソッドがすでに登録されている ID で呼び出された場合（OMG 定義済みの既定の名前で呼び出された場合も含みます），register_initial_reference()で InvalidName 例外が発生します。
  - _id
    初期リファレンスを認識するための ID
  - _obj
    初期リファレンス
- virtual CORBA::Object_ptr **resolve_initial_references**(
  const char* **_id**) = 0;
  このメソッドは，post_init()呼び出し時にだけ有効です。このメソッドの機能は ORB::resolve_initial_references()と同一です。このメソッドは，ORB が完全に初期化されていないためにまだ使用できないときに，インタセプタの登録の一部として初期リファレンスが必要となる場合に使用します。
  - _id
    初期リファレンスを認識するための ID

  解決する名前が不正な場合，resolve_initial_references()では InvalidName 例外が発生します。
- virtual void **add_client_request_interceptor**(
  ClientRequestInterceptor_ptr **_interceptor**) = 0;
  このメソッドは，クライアント側リクエストインタセプタの一覧に項目を追加するときに使用します。同じ名前のクライアント側リクエストインタセプタがすでに存在している場合は，DuplicateName 例外が発生します。
  - _interceptor
    追加する ClientRequestInterceptor
- virtual void **add_server_request_interceptor**(
  ServerRequestInterceptor_ptr **_interceptor**) = 0;

このメソッドは，サーバ側リクエストインタセプタの一覧に項目を追加するときに使用します。同じ名前のサーバ側リクエストインタセプタがすでに存在している場合は，DuplicateName 例外が発生します。

- _interceptor
  追加する ServerRequestInterceptor

- virtual void **add_ior_interceptor**(
  IORInterceptor_ptr **_interceptor**) = 0;
  このメソッドは，IOR インタセプタの一覧に項目を追加するときに使用します。同じ名前の IOR インタセプタがすでに存在している場合は，DuplicateName 例外が発生します。
  - _interceptor
    追加する IORInterceptor
- virtual CORBA::ULong **allocate_slot_id**() = 0;
  このメソッドは，割り当て済みスロットのインデックスを返します。
  サービスは，allocate_slot_id を呼び出して PortableInterceptor::Current のスロットを割り当てます。
  注
  スロット ID の割り当ては，ORB イニシャライザの内部でできますが，スロット自体は初期化できません。ORB イニシャライザの内部で Current の set_slot()または get_slot()を呼び出すと，マイナーコード 14 の BAD_INV_ORDER 例外が発生します。詳細については，「27.6　Current」を参照してください。
- virtual void **register_policy_factory**(
  CORBA::ULong **_type**, PolicyFactory_ptr **_policy_factory**) = 0;
  このメソッドは，指定した PolicyType の PolicyFactory を登録します。
  指定した PolicyType の PolicyFactory がすでに登録されている場合は，標準マイナーコード 16 の BAD_INV_ORDER 例外が発生します。
  - _type
    指定した PolicyFactory の CORBA::PolicyType
  - _policy_factory
    指定した CORBA::PolicyType のファクトリ

27.16　Parameter

```
struct Dynamic::Parameter
```

この構造体は，パラメタ情報を保持します。また，ParameterList に使用される要素です。詳細については，「27.17　ParameterList」を参照してください。

27.16.1　インクルードファイル

この構造体を使用するときは，Dynamic_c.hh ファイルをインクルードしてください。

27.16.2　Parameter のメンバ

- CORBA::Any argument;
  このメンバは，CORBA::Any 形式でパラメタデータを格納します。
- CORBA::ParameterMode mode;
  このメンバは，パラメタのモードを指定します。PARAM_IN，PARAM_OUT，または PARAM_INOUT のどれかの enum 値がこのメンバの値となります。

27.17　ParameterList

```
class Dynamic::ParameterList
```

このクラスは，RequestInfo クラスの arguments()メソッドが返すパラメタ情報の受け渡しで使用する，Parameter 型の可変長配列のインプリメンテーションです。ParameterList の長さは，実行時に使用できます。

詳細については，「27.19.2　RequestInfo のメソッド」の arguments()メソッドを参照してください。

27.17.1　インクルードファイル

このクラスを使用するときは，Dynamic_c.hh ファイルをインクルードしてください。

27.18 PolicyFactory

```
class PortableInterface::PolicyFactory : public virtual CORBA::LocalObject
```

ポータブル ORB サービスインプリメンテーションは，ORB 初期化時に，PolicyFactory クラスのインスタンスを登録します。これは，CORBA::ORB::create_policy()を使用してポリシーの型を構成できるようにするためです。POA は，この方法で ORBInitInfo に登録されたポリシーを保存するために必要です。詳細については，「27.15.3 ORBInitInfo のメソッド」の register_policy_factory()メソッドを参照してください。

27.18.1 インクルードファイル

このクラスを使用するときは，PortableInterceptor_c.hh ファイルをインクルードしてください。

27.18.2 PolicyFactory のメソッド

- virtual CORBA::Policy_ptr **create_policy**(
  CORBA::ULong **_type**, const CORBA::Any& **_value**) = 0;
  登録済み PolicyFactory の PolicyType に CORBA::ORB::create_policy()が呼び出されると，ORB は，登録済み PolicyFactory インスタンスの create_policy()を呼び出します。create_policy()メソッドは，次に，指定した CORBA::Any に対応する値を持つ CORBA::Policy から派生したインタフェースのインスタンスを返します。失敗した場合は，CORBA::ORB::create_policy()で示す例外が発生します。
  - _type
    作成するポリシーの型を指定している CORBA::PolicyType
  - _value
    CORBA::Policy を構成するためのデータを格納している CORBA::Any

27.19 RequestInfo

```
class PortableInterceptor::RequestInfo : public virtual CORBA::LocalObject
```

このクラスは，ClientRequestInfo と ServerRequestInfo の派生元となるベースクラスです。各インタセプトポイントに与えられるオブジェクトを通じて，インタセプタはリクエスト情報にアクセスできます。クライアント側とサーバ側のインタセプトポイントでは扱う情報が異なるため，情報オブジェクトは二つあります。ClientRequestInfo はクライアント側インタセプトポイントに渡され，ServerRequestInfo は，サーバ側インタセプトポイントに渡されます。しかし，両方に共通する情報があるため，どちらも共通インタフェースの RequestInfo を継承しています。

27.19.1 インクルードファイル

このクラスを使用するときは，PortableInterceptor_c.hh ファイルをインクルードしてください。

27.19.2 RequestInfo のメソッド

- virtual CORBA::ULong request_id() = 0;

  このメソッドは，アクティブなリクエストまたは応答シーケンスを一意に識別する ID を返します。リクエストまたは応答シーケンスが解決された場合は，ID が再使用されることもあります。

  注

  この ID は GIOP の request_id とは異なります。GIOP がトランスポート機能として使用されている場合，この ID と GIOP の request_id は必ずしも一致するとは限りません。また，これらが一致している必要はありません。

- virtual char* operation() = 0;

  このメソッドは，呼び出されているオペレーションの名前を返します。

- virtual Dynamic::ParameterList* arguments() = 0;

  このメソッドは，呼び出されているオペレーションの引数を格納する，Dynamic::ParameterList を返します。引数がない場合，この属性は長さが 0 のシーケンスとなります。

- virtual Dynamic::ExceptionList* exceptions() = 0;

  このメソッドは，オペレーション呼び出しで発生するユーザ例外の TypeCode を定義する Dynamic::ExceptionList を返します。ユーザ例外がない場合，この属性は長さが 0 のシーケンスとなります。

- virtual CORBA::StringSequence* contexts() = 0;

  このメソッドは，オペレーション呼び出しで渡される可能性のあるコンテキストを定義する CORBA::StringSequence を返します。コンテキストがない場合，この属性は長さが 0 のシーケンスとなります。

- virtual CORBA::StringSequence* operation_context() = 0;

  このメソッドは，リクエストで送信されるコンテキストを定義する CORBA::StringSequence を返します。

- virtual CORBA::Any* result() = 0;

  このメソッドは，オペレーション呼び出しの結果を CORBA::Any 形式で返します。オペレーションのリターン型が void の場合，この属性は，TCKind 値が tk_void または値なしのタイプコードを格納する CORBA::Any となります。

- virtual CORBA::Boolean response_expected() = 0;

  このメソッドは，応答が期待されているかどうかを示すブール値を返します。

  クライアントでは，response_expected が false の場合，応答が返らないため，receive_reply()を呼び出せません。例外が発生した場合以外は，receive_other()を呼び出します。例外が発生した場合は，receive_exception()を呼び出します。

- virtual CORBA::Short sync_scope() = 0;

  このメソッドは，Messaging 仕様で定義されている属性を返します。この属性は，response_expected が false のときだけ有効です。response_expected が true の場合，sync_scope()の値は不定です。この属性は，クライアントに制御が戻る前のリクエストの進捗状況を表します。この属性の値を次に示します。

  - Messaging::SYNC_NONE
  - Messaging::SYNC_WITH_TRANSPORT
  - Messaging::SYNC_WITH_SERVER
  - Messaging::SYNC_WITH_TARGET

  サーバでは，すべてのスコープで，ターゲットオペレーション呼び出しの戻り値から応答が生成されますが，その応答はクライアントには返されません。クライアントに返されなくても応答は生成されているので，通常のサーバ側インタセプトポイント，つまり，receive_request_service_contexts()，receive_request()，send_reply()，または send_exception()に従います。

  SYNC_WITH_SERVER と SYNC_WITH_TARGET に関しては，ターゲットオペレーションが呼び出される前に，サーバは空の応答をクライアントに返却します。サーバ側インタセプタはこの応答を受け取りません。

- virtual CORBA::Short reply_status() = 0;

  このメソッドは，オペレーションの呼び出し結果の状態を表す属性を返します。この属性の値を次に示します。

  - PortableInterceptor::SUCCESSFUL = 0
  - PortableInterceptor::SYSTEM_EXCEPTION = 1
  - PortableInterceptor::USER_EXCEPTION = 2
  - PortableInterceptor::LOCATION_FORWARD = 3
  - PortableInterceptor::TRANSPORT_RETRY = 4

クライアント側では，この属性の値は次のようになります。

- インタセプトポイント receive_reply の中では，この属性の値は SUCCESSFUL だけです。
- インタセプトポイント receive_exception の中では，この属性の値は SYSTEM_EXCEPTION または USER_EXCEPTION です。
- インタセプトポイント receive_other の中では，この属性の値は SUCCESSFUL，LOCATION_FORWARD，または TRANSPORT_RETRY のどれかです。SUCCESSFUL は，非同期リクエストが正常にリターンしたことを意味します。LOCATION_FORWARD は，LOCATION_FORWARD 状態が応答で返却されたことを意味します。TRANSPORT_RETRY は，トランスポート機能がリトライを指示したことを意味します。例えば，NEEDS_ADDRESSING_MODE 状態の GIOP 応答が該当します。

サーバ側では，この属性の値は次のようになります。

- インタセプトポイント send_reply の中では，この属性の値は SUCCESSFUL だけです。
- インタセプトポイント send_exception の中では，この属性の値は SYSTEM_EXCEPTION または USER_EXCEPTION です。
- インタセプトポイント send_other の中では，この属性の値は SUCCESSFUL または LOCATION_FORWARD です。SUCCESSFUL は，非同期リクエストが正常にリターンしたことを意味します。LOCATION_FORWARD は，LOCATION_FORWARD 状態が応答で返却されたことを意味します。

- virtual CORBA::Object_ptr **forward_reference**() = 0;

  reply_status()が LOCATION_FORWARD を返す場合，このメソッドは，リクエストフォワード先オブジェクトを返します。フォワードされるリクエストが実際に発生するかどうかはわかりません。

- virtual CORBA::Any* **get_slot**(

  CORBA::ULong _id) = 0;

  このメソッドは，リクエストのスコープにある Current の指定スロットからデータを CORBA::Any 形式で返します。

  指定したスロットが設定されていない場合，TCKind 値が tk_null のタイプコードを格納する CORBA::Any が返されます。

  ID が割り当てられていないスロットを表す場合は，InvalidSlot 例外が発生します。

  スロットと PortableInterceptor::Current の詳細については，「27.6　Current」を参照してください。

  - _id

    取得するスロットの SlotId

- virtual IOP::ServiceContext* **get_request_service_context**(

  CORBA::ULong _id) = 0;

  このメソッドは，リクエストに対応づけられたサービスコンテキストのうち，指定した ID のサービスコンテキストのコピーを返します。

  リクエストのサービスコンテキストに，指定した ID のエントリが含まれていない場合は，標準マイナーコード 26 の BAD_PARAM 例外が発生します。

- _id
  取得するスロットの IOP::ServiceContext

- virtual IOP::ServiceContext* get_reply_service_context(
  CORBA::ULong _id) = 0;
  このメソッドは，応答に対応づけられたサービスコンテキストのうち，指定した ID のサービスコンテキストのコピーを返します。
  応答のサービスコンテキストに，指定した ID のエントリが含まれていない場合は，標準マイナーコード 26 の BAD_PARAM 例外が発生します。
  - _id
    取得するスロットの IOP::ServiceContext

27.20 ServerRequestInfo

```
class PortableInterceptor::ServerRequestInfo :
                        public virtual RequestInfo,
                        public virtual CORBA::LocalObject
```

このクラスは RequestInfo から派生したクラスで，サーバ側インタセプトポイントに渡されます。ServerRequestInfo のメソッドは，一部のインタセプトポイントでは有効ではありません。

次の表に，属性またはメソッドの有効性を示します。無効の属性またはメソッドにアクセスすると，標準マイナーコード 14 の BAD_INV_ORDER 例外が発生します。

表 27-3 ServerRequestInfo の有効性（C++）

	receive_request_service_contexts	receive_request	send_reply	send_exception	send_other
request_id	○	○	○	○	○
operation	○	○	○	○	○
arguments	×	○※1	○	×	×
exceptions	×	○	○	○	○
contexts	×	○	○	○	○
operation_context	×	○	○	×	×
result	×	×	○	×	×
response_expected	○	○	○	○	○
sync_scope	○	○	○	○	○
reply_status	×	×	○	○	○
forward_reference	×	×	×	×	○※2
get_slot	○	○	○	○	○
get_request_service_context	○	○	○	○	○
get_reply_service_context	×	×	○	○	○
sending_exception	×	×	×	○	×
object_id	×	○	○	○※3	○※3
adapter_id	×	○	○	○※3	○※3
target_most_derived_interface	×	○	×※4	×※4	×※4
get_server_policy	○	○	○	○	○
set_slot	○	○	○	○	○

	receive_request_service_contexts	receive_request	send_reply	send_exception	send_other
target_is_a	×	○	×※4	×※4	×※4
add_reply_service_context	○	○	○	○	○

注※1

ServerRequestInfo が receive_request()に渡される場合，in，inout，または out の各引数のリストにエントリがありますが，使用できるのは in 引数と inout 引数だけです。

注※2

reply_status()が LOCATION_FORWARD を返さない場合，この属性にアクセスすると標準マイナーコード 14 の BAD_INV_ORDER 例外が発生します。

注※3

Servant Locator でロケーションのフォワードや，例外が発生した場合，この属性またはメソッドは，当該インタセプトポイントで使用できないことがあります。使用できない場合は，標準マイナーコード 1 の NO_RESOURCES 例外が発生します。

注※4

このメソッドは，当該インタセプトポイントで使用できません。

それは，必要な情報を取得するためにはターゲットオブジェクトのサーバントにアクセスする必要がありますが，この時点で ORB はこのサーバントを使用できないためです。例えば，ServantLocator を使用する POA がオブジェクトのアダプタである場合，ORB は，ServantLocator::postinvoke()を呼び出したあとでインタセプトポイントを呼び出します。

27.20.1　インクルードファイル

このクラスを使用するときは，PortableInterceptor_c.hh ファイルをインクルードしてください。

27.20.2　ServerRequestInfo のメソッド

- virtual CORBA::Any* **sending_exception**() = 0;

  このメソッドは，クライアントに返される例外を格納するデータを CORBA::Any の形式で返します。

  CORBA::Any に挿入できないユーザ例外の場合，例えば，未知の例外やバインディングで TypeCode が提供されない場合などは，この属性は，標準マイナーコード 1 の UNKNOWN システム例外を格納する CORBA::Any となります。

- virtual CORBA::OctetSequence* **object_id**() = 0;

  このメソッドは，オペレーション呼び出しのターゲットを表す不透明な object_id を CORBA::OctetSequence の形式で返します。

- virtual CORBA::OctetSequence* **adapter_id**() = 0;
  このメソッドは，オブジェクトアダプタの不透明な識別子を CORBA::OctetSequence の形式で返します。
- virtual char* **target_most_derived_interface**() = 0;
  このメソッドは，サーバントの派生したインタフェースの RepositoryID を返します。
- virtual CORBA::Policy_ptr **get_server_policy**(
  CORBA::ULong **_type**) = 0;
  このメソッドは，オペレーションに対して有効なポリシーのうち，指定したポリシー型を持つポリシーを返します。register_policy_factory で登録された型を持つポリシーだけが，CORBA::Policy オブジェクトとして返されます。
  指定した型のポリシーが register_policy_factory で登録されていない場合は，標準マイナーコード 3 の INV_POLICY 例外が発生します。詳細については，「27.15.3　ORBInitInfo のメソッド」の register_policy_factory()メソッドを参照してください。
  - _type
    取得するポリシーを指定する CORBA::PolicyType
- virtual void **set_slot**(
  CORBA::ULong **_id**, const CORBA::Any& **_data**) = 0;
  このメソッドを使用すると，インタセプタでリクエストのスコープ内の PortableInterceptor::Current のスロットにデータを設定できます。スロットにすでにデータが設定されている場合，既存データは上書きされます。割り当てられていないスロットを表す ID を指定した場合は，InvalidSlot 例外が発生します。スロットと PortableInterceptor::Current の詳細については，「27.6　Current」を参照してください。
  - _id
    スロットの SlotId
  - _data
    指定したスロットに設定する，CORBA::Any オブジェクト形式のデータ
- virtual CORBA::Boolean **target_is_a**(
  const char* **_id**) = 0;
  このメソッドは，指定した RepositoryId がサーバントの場合は true を返し，そうでない場合は false を返します。
  - _id
    サーバントがこの CORBA::RepositoryId であるかどうかを呼び出し元が調べます。
- virtual void **add_reply_service_context**(
  const IOP::ServiceContext&**_service_context**,
  CORBA::Boolean **_replace**) = 0;
  このメソッドを使用すると，インタセプタでサービスコンテキストを応答に登録できます。

サービスコンテキストの順序の宣言はありません。登録された順序でサービスコンテキストが表示される場合も，されない場合もあります。

- _service_context
  応答に登録する IOP::ServiceContext
- _replace
  指定した ID のサービスコンテキストがすでに存在する場合のメソッドの動作。false の場合は，標準マイナーコード 15 の BAD_INV_ORDER 例外が発生します。true の場合は，既存のコンテキストが新しいコンテキストに置き換えられます。

27.21 ServerRequestInterceptor

```
class PortableInterceptor::ServerRequestInterceptor :
                        public virtual Interceptor,
                        public virtual CORBA::LocalObject
```

このクラスは，ユーザ定義サーバ側インタセプタを継承するときに使用します。ServerRequestInterceptor インスタンスは，VisiBroker ORB に登録されます。詳細については，「27.14 ORBInitializer」を参照してください。

27.21.1 インクルードファイル

このクラスを使用するときは，PortableInterceptor_c.hh ファイルをインクルードしてください。

27.21.2 ServerRequestInterceptor のメソッド

- virtual void receive_request_service_contexts(
  ServerRequestInfo_ptr _ri) = 0;
  このインタセプトポイントでは，インタセプタは受信したリクエストからサービスコンテキスト情報を取得し，PortableInterceptor::Current のスロットに転送する必要があります。
  このインタセプトポイントは，サーバントマネージャよりも先に呼び出されます。オペレーションのパラメタはまだこの時点では使用できません。このインタセプトポイントは，ターゲットオペレーションの呼び出しと同じスレッドで実行される場合も，実行されない場合もあります。
  このインタセプトポイントでは，システム例外が発生する場合があります。システム例外が発生した場合，ほかのインタセプタの receive_request_service_contexts()インタセプトポイントは呼び出されません。フロースタックからインタセプタが取り出され，そのインタセプタの send_exception()インタセプトポイントが呼び出されます。
  このインタセプトポイントでは，ForwardRequest 例外を発生させることができます。インタセプトでこの例外が発生した場合，ほかのインタセプタの receive_request_service_contexts()メソッドは呼び出されません。フロースタックからインタセプタが取り出され，そのインタセプタの send_other インタセプトポイントが呼び出されます。ForwardRequest 例外については，「27.9 ForwardRequest」を参照してください。
  - _ri
    インタセプタが使用する ServerRequestInfo インスタンス
- virtual void receive_request(
  ServerRequestInfo_ptr _ri) = 0;

このインタセプトポイントを使用すると，メソッドのパラメタを含むすべての情報が使用できる状態になったあとで，リクエスト情報をインタセプタで照会できます。このインタセプトポイントは，ターゲットオペレーションの呼び出しと同じスレッドで実行されます。

DSI モデルでは，ユーザコードが arguments()を呼び出したときに最初にパラメタが使用できるようになるため，arguments()の内側から receive_request()が呼び出されます。DSI モデルでは，arguments()を呼び出さないようにすることもできます。

ターゲットオペレーションは arguments()より先に set_exception()を呼び出すことがあります。ORB は，arguments()と set_exception()のどちらが呼び出されても，receive_request()は 1 回だけ呼び出されることを保障します。

set_exception()で receive_request()が呼び出された場合，arguments()のリクエストは，標準マイナーコード 1 の NO_RESOURCES 例外となります。

このインタセプトポイントでは，システム例外が発生する場合があります。システム例外が発生した場合，ほかのインタセプタの receive_request()メソッドは呼び出されません。フロースタックからインタセプタが取り出され，そのインタセプタの send_exception()インタセプトポイントが呼び出されます。

このインタセプトポイントでは，ForwardRequest 例外を発生させることができます。インタセプトでこの例外が発生した場合，ほかのインタセプタの receive_request()メソッドは呼び出されません。フロースタックからインタセプタが取り出され，そのインタセプタの send_other インタセプトポイントが呼び出されます。ForwardRequest 例外については，「27.9　ForwardRequest」を参照してください。

- _ri
  インタセプタが使用する ServerRequestInfo インスタンス

- virtual void **send_reply**(
  ServerRequestInfo_ptr _ri) = 0;
  このインタセプトポイントを使用すると，ターゲットオペレーションが呼び出されてから応答がクライアントに返されるまでの間に，インタセプタで応答情報の照会と応答サービスコンテキストの修正ができます。このインタセプトポイントは，ターゲットオペレーションと同じスレッドで実行されます。
  このインタセプトポイントでは，システム例外が発生する場合があります。システム例外が発生した場合，ほかのインタセプタの send_reply()インタセプトポイントは呼び出されません。フロースタックにあるインタセプタの send_exception()インタセプトポイントが呼び出されます。
  - _ri
    インタセプタが使用する ServerRequestInfo インスタンス
- virtual void **send_exception**(
  ServerRequestInfo_ptr _ri) = 0;
  このインタセプトポイントは，例外が発生したときに呼び出されます。このインタセプトポイントを使用すると，例外がクライアントに通知される前にインタセプタでリクエスト情報の照会と応答サーバコンテキストの修正ができます。このインタセプトポイントは，ターゲットオペレーションと同じスレッドで実行されます。このインタセプトポイントでは，システム例外が発生する場合があります。システム例外が発生した場合は，フロースタックから取り出された一連のインタセプタが send_exception()

呼び出し時に受け取る例外が変更されます。クライアントに通知される例外は，インタセプタが通知する最後の例外です。ほかのインタセプタが例外を変更しなければ，元の例外が通知されます。

このインタセプトポイントでは，ForwardRequest 例外を発生させることができます。インタセプタがこの例外を出力した場合，ほかのインタセプタの send_exception()インタセプトポイントは呼び出されません。フロースタックにあるインタセプタの send_other インタセプトポイントが呼び出されます。ForwardRequest 例外については，「27.9　ForwardRequest」を参照してください。

- _ri
  インタセプタが使用する ServerRequestInfo インスタンス

- virtual void send_other(
  ServerRequestInfo_ptr _ri) = 0;
  このインタセプトポイントを使用すると，リクエストの結果が正常な応答でも例外でもない場合に使用できる情報をインタセプタで照会できます。例えば，リクエストがリトライになる場合（例えば，LOCATION_FORWARD 状態で GIOP Reply を受信した場合）などです。このインタセプトポイントは，ターゲットオペレーションと同じスレッドで実行されます。

  このインタセプトポイントでは，システム例外が発生する場合があります。システム例外が発生した場合，ほかのインタセプタの send_other()メソッドは呼び出されません。フロースタックにあるインタセプタの send_exception()インタセプトポイントが呼び出されます。

  このインタセプトポイントでは，ForwardRequest 例外を発生させることができます。インタセプタがこの例外を出力した場合，ForwardRequest 例外が提供する新しい情報で一連のインタセプタの send_other()メソッドが呼び出されます。ForwardRequest 例外については，「27.9 ForwardRequest」を参照してください。

  - _ri
    インタセプタが使用する ServerRequestInfo インスタンス

28 VisiBroker 4.x インタセプタおよびオブジェクトラッパーのインタフェースとクラス（C++）

この章では，VisiBroker 4.x インタセプタとオブジェクトラッパーに対して使用するインタフェースとクラスについて，C++言語でのインタフェースを説明します。VisiBroker 4.x インタセプタおよびオブジェクトラッパーの生成方法と使用方法については，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「VisiBroker 4.x インタセプタの使用」，「オブジェクトラッパーの使用」，および「イベントリスナー」の記述を参照してください。

28.1 概要

VisiBroker 4.x インタセプタは，VisiBroker 4.x に定義され，インプリメントされているインタセプタです。ポータブルインタセプタと同様に，Borland Enterprise Server VisiBroker の ORB サービスに ORB の実行フローを受け取る機能を提供します。VisiBroker 4.x インタセプタには次の 4 種類があります。

- クライアントインタセプタ

  システムレベルのインタセプタです。トランザクションやセキュリティなどの ORB サービスのフックを提供し，クライアントの ORB に処理させるために使用できます。

- サーバインタセプタ

  システムレベルのインタセプタです。トランザクションやセキュリティなどの ORB サービスのフックをサーバの ORB に処理させるために使用できます。

- オブジェクトラッパー

  ユーザレベルのインタセプタです。簡易なトレースとデータキャッシュができるような，スタブとスケルトンの呼び出しをインタセプトするための簡易な機構を提供します。

- イベントキュー

  イベントキュー機能について説明します。サーバが対象とするイベントタイプに基づいてリスナーをイベントキューに登録できるので，サーバが必要なときにこのイベントを処理できます。

VisiBroker 4.x インタセプタ，およびオブジェクトラッパーの使用方法の詳細については，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「VisiBroker 4.x インタセプタの使用」，「オブジェクトラッパーの使用」，「イベントキュー」，および「イベントリスナー」の記述を参照してください。

28.2 インタセプタマネージャ

インタセプタのインストールと管理は，インタセプタマネージャで実行します。InterceptorManager インタフェースは，すべてのグローバルインタセプタを管理するためのグローバルインタセプタマネージャです。

InterceptorManager は各インタセプタに対応します。InterceptorManager は特定の種類のインタセプタのリスト，またはチェーンを保持します。これらのインタセプタは同じスコープを持ち，同時に起動する必要があります。そのため，POALifeCycleInterceptor や BindInterceptor のようなグローバルインタセプタは，グローバル InterceptorManager を持ちますが，スコープ付きのインタセプタは，POA およびオブジェクトごとのインタセプタも，各スコープの InterceptorManager を持ちます。各スコープは，グローバル，POA，およびオブジェクトのどの場合も複数のインタセプタ型を保持できます。InterceptorManagerControl から，特定のインタセプタに対する正しい種類のマネージャを取得します。

グローバルインタセプタには，ローカライズされたインタセプタをインストールするために，拡張インタセプタマネージャが渡されることがあります。例えば，各 POA のインタセプタは POALifeCycleInterceptorManager を使用します。

グローバルインタセプタマネージャのインスタンスである InterceptorManager は，ORB::resolve_initial_references を呼び出して，文字列 InterceptorManager を引数として渡すことで，取得できます。この値は，POALifeCycleInterceptor や BindInterceptor のようなグローバルインタセプタのインストールのために，ORB が管理モードのとき，つまり ORB の初期化中にだけ使用できます。

POALifeCycleInterceptorManager は各 POA のマネージャで，POALifeCycleInterceptor の create 呼び出し中にだけ使用できます。POALifeCycleInterceptor は，生成用の呼び出し時に，ほかのすべてのサーバ側インタセプタをセットアップできます。BindInterceptorManager は各オブジェクトのマネージャで，BindInterceptor の bind_succeeded 呼び出し時にだけ使用できます。BindInterceptor は呼び出し中に ClientRequestInterceptor を設定できます。

28.3 IOR テンプレート

POALifeCycleInterceptor::create()の呼び出し中に，インタセプタに加えて，IOR テンプレートを直接 POAInterceptorManager インタフェースで修正できます。IOR テンプレートは完全な IOR 値ですが，type_id が未設定で，GIOP::ProfileBodyValue のオブジェクトキーはすべて不完全なものです。IORCreationInterceptor の呼び出し前に，POA は type_id を設定してテンプレートのオブジェクトキーを記述します。

28.4 InterceptorManager

```
class interceptor::InterceptorManager :
                        public virtual VISPseudoInterface
```

ほかのすべてのインタセプタマネージャは，このクラスから継承します。インタセプタマネージャは，インタセプタのインストールと削除を管理するために使用するクラスです。

28.5 InterceptorManagerControl

```
class interceptor::InterceptorManagerControl :
                        public virtual CORBA::PseudoObject
```

このクラスは，関連のあるインタセプタマネージャを一括管理するために使用します。このクラスは使用できるすべてのマネージャを保持し，各マネージャは管理対象のインタセプタの型に対応する文字列で識別されます。スコープごとに一つの InterceptorManagerControl があります。

28.5.1 インクルードファイル

このクラスを使用するときは，interceptor_c.hh ファイルをインクルードしてください。

28.5.2 InterceptorManagerControl のメソッド

- virtual InterceptorManager_ptr **get_manager**(
  const char ***name**);
  このメソッドは，InterceptorManager のインスタンスを返します。InterceptorManager は，マネージャを識別する文字列を返します。
  - name
    インタセプタの名前

28.6 BindInterceptor

```
class interceptor::BindInterceptor : public virtual VISPseudoInterface
```

このクラスは，クライアント側のバインドイベントとリバインドイベント時に呼び出されます。ユーザはこのクラスを派生させてユーザ処理を実装させてください。

バインドインタセプタは，バインドの前後にクライアント側で呼び出すグローバルインタセプタです。

バインド中に CORBA 例外が発生した場合は，チェーンに登録されている残りのインタセプタは呼び出さないで，チェーンから削除します。bind_succeeded または bind_failed 中に発生した CORBA 例外は無視されます。

28.6.1 インクルードファイル

このクラスを使用するときは，interceptor_c.hh ファイルをインクルードしてください。

28.6.2 BindInterceptor のメソッド

- virtual IOP::IORValue_ptr bind(
  IOP::IORValue_ptr ior,
  CORBA::Object_ptr obj,
  CORBA::Boolean rebind,
  VISClosure& closure);
  このメソッドは，すべての ORB バインドオペレーション中に呼び出されます。
  新規 IOR を使用してバインドオペレーションを継続する場合，新規 IOR を返します。それ以外の場合は null を返し，元の IOR を使用してバインドを続行します。渡されたパラメタと同じ IOR を返すことは不正なので，バインド時に例外が発生します。
  - ior
    クライアントがバインドしているサーバオブジェクトの IOR
  - obj
    サーバにバインドしているクライアントオブジェクト。このオブジェクトはこの時点では適切に初期化されていないので，オブジェクトのオペレーションはしないでください。しかし，このオブジェクトをデータ構造体に保存してバインド完了後に使用できます。
  - rebind
    サーバのリバインドを試行します。QoS によっては，bind()が失敗したあとにリバインドが試行されることがあります。
  - closure

バインドオペレーション用の新しい closure オブジェクト。closure は，bind_failed，bind_succeeded，または exception_occurred に対応した呼び出しで使用します。

- virtual IOP::IORValue_ptr **bind_failed**(
  IOP::IORValue_ptr **ior**,
  CORBA::Object_ptr **obj**,
  VISClosure& **closure**);
  このメソッドは，バインドオペレーションが失敗したときに呼び出されます。
  この IOR に対してリバインドを試行する場合，新規 IOR を返します。それ以外の場合は null を返し，リバインドは試行されません。
  - ior
    バインドオペレーションが失敗したサーバオブジェクトの IOR
  - obj
    サーバにバインドしているクライアントオブジェクト
  - closure
    bind メソッドでデータを格納した場合，格納データを取得できます。
- virtual void **bind_succeeded**(
  IOP::IORValue_ptr **ior**,
  CORBA::Object_ptr **obj**,
  CORBA::Long **profileIndex**,
  InterceptorManagerControl_ptr **interceptorControl**,
  VISClosure& **closure**);
  このメソッドは，バインドオペレーションが成功すると呼び出されます。
  - ior
    バインドオペレーションが成功したサーバオブジェクトの IOR
  - obj
    サーバにバインドしているクライアントオブジェクト
  - profileIndex
    コネクションプロトコルを識別します。
  - interceptorControl
    このマネージャがマネージャ種別一覧を提供します。
  - closure
    bind メソッドでデータを格納した場合，格納データを取得できます。
- void **exception_occurred**(
  IOP::IORValue_ptr **ior**,
  CORBA::Object_ptr **obj**,

CORBA::Environment_ptr env,
VISClosure& closure)
このメソッドは，bind で例外が発生すると呼び出されます。

- ior
  バインドオペレーションが成功したサーバオブジェクトの IOR
- target
  サーバにバインドしているクライアントオブジェクト
- env
  発生した例外の情報
- closure
  bind メソッドでデータを格納した場合，格納データを取得できます。

28.7 BindInterceptorManager

```
class interceptor::BindInterceptorManager :
                        public virtual InterceptorManager,
                        public virtual VISPseudoInterface
```

このクラスは，すべてのグローバルバインドインタセプタを管理します。パブリックメソッドを一つ持ち，インタセプタを登録するために使います。

BindInterceptorManager は必ず ORB_init()で使用してください。orb が初期化されたあとは，BindInterceptorManager は無効です。そのため，VISInit から継承するローダクラスの ORB_init()の中でだけ使用してください。

ORB_init()内で ORB を使用して resolve_initial_references("VisiBrokerInterceptorControl")を実行して InterceptorControlManager を取得後，識別文字列 Bind を指定した InterceptorManagerControl::get_manager()を使って，InterceptorManagerControl から BindInterceptorManager を取得します。

28.7.1 インクルードファイル

このクラスを使用するときは，interceptor_c.hh ファイルをインクルードしてください。

28.7.2 BindInterceptorManager のメソッド

- void **add**(
  BindInterceptor_ptr **interceptor**);
  このメソッドを使用して，バインド時に起動するインタセプタのリストに，BindInterceptor を追加します。
  - interceptor
    追加するインタセプタ

28.8 ClientRequestInterceptor

```
class interceptor::ClientRequestInterceptor :
                        public virtual VISPseudoInterface
```

このクラスは，ユーザ独自のクライアントインタセプタを派生させるために使用します。

クライアントリクエストインタセプタは，バインドインタセプタの BindInterceptor::bind_succeeded 呼び出し中に ClientRequestInterceptorManager を使用してインストールできます。また，コネクションが確立されている間はアクティブ状態を維持します。ユーザが派生したクラスに定義しているメソッドは，オペレーション要求の準備時，送信時，返信メッセージの受信時，または例外発生時に ORB によって呼び出されます。

28.8.1 インクルードファイル

このクラスを使用するときは，interceptor_c.hh ファイルをインクルードしてください。

28.8.2 ClientRequestInterceptor のメソッド

- virtual void **preinvoke_premarshal**(
  CORBA::Object_ptr **target**,
  const char* **operation**,
  IOP::ServiceContextList& **service_contexts**,
  VisClosure& **closure**);
  このメソッドは，リクエストごとに，それらがマーシャリングされる前に ORB に呼び出されます。このインタセプタで例外が発生した場合は，即時にリクエストは終了させられます。インタセプタのチェーンでは，処理済みのインタセプタだけがチェーンに残ります。その例外を起こしたリクエストは送信されないで，exception_occurred()がインタセプタのチェーン全体に対して呼び出されます。
  - target
    サーバへのバインドを試みたクライアントオブジェクト
  - operation
    呼び出すオペレーションの名前を識別します。
  - service_contexts
    ORB が割り当てたサービス。このサービスは OMG に登録されているタグで識別されます。
  - closure
    あるインタセプタメソッドが保存したデータを格納する場合があります。その場合，このデータをほかのインタセプタメソッドがあとで取得できます。

- virtual void **preinvoke_postmarshal**(
  CORBA::Object_ptr **target**,
  CORBA::MarshalOutBuffer& **payload**,
  VISClosure& **closure**);
  このメソッドは，リクエストごとに，リクエストのマーシャリングが済んでから送信されるまでの間に呼び出されます。このメソッドで例外が発生した場合は，残りのチェーンは呼び出されません。また，該当するリクエストはサーバに送信されません。その後，exception_occurred()はインタセプタのチェーン全体に呼び出されます。
  - target
    サーバへのバインドを試みたクライアントオブジェクト
  - payload
    マーシャリング済みバッファ
  - closure
    あるインタセプタメソッドが保存したデータを格納する場合があります。その場合，このデータをほかのインタセプタメソッドがあとで取得できます。
- virtual void **postinvoke**(
  CORBA::Object_ptr **target**,
  const IOP::ServiceContextList& **service_contexts**,
  CORBA::MarshalInBuffer& **payload**,
  CORBA::Environment_ptr **env**,
  VISClosure& **closure**);
  リクエストが正常に，または例外を発生させて完了したあと，このメソッドが呼び出されます。このメソッドは ServantLocator の起動後に呼び出されます。チェーン内のインタセプタで例外が発生すると，そのインタセプタは exception_occurred()を呼び出し，チェーン内の残りのインタセプタは postinvoke()ではなく exception_occurred()を呼び出します。
  - target
    サーバへのバインドを試みたクライアントオブジェクト
  - service_contexts
    ORB が割り当てたサービスを識別します。このサービスは OMG に規定されているものです。
  - payload
    マーシャリング済みバッファ
  - env
    発生した例外についての情報を格納します。
  - closure
    あるインタセプタメソッドが保存したデータを格納する場合があります。その場合，このデータをほかのインタセプタメソッドがあとで取得できます。

- virtual void **exception_occurred**(
  CORBA::Object_ptr **target**,
  CORBA::Environment_ptr **env**,
  VISClosure& **closure**);
  このメソッドは，呼び出し前に例外が発生した場合に ORB が呼び出します。呼び出し後に発生した例外はすべて，postinvoke メソッドの Environment パラメタに収集されます。
  - target
    サーバへのバインドを試みたクライアントオブジェクト
  - env
    発生した例外についての情報を格納します。
  - closure
    あるインタセプタメソッドが保存したデータを格納する場合があります。その場合，このデータをほかのインタセプタメソッドがあとで取得できます。

28.9 ClientRequestInterceptorManager

```
class interceptor::ClientRequestInterceptorManager :
                        public virtual InterceptorManager,
                        public virtual VISPseudoInterface
```

このクラスは，カレントオブジェクトの ClientRequestInterceptor のチェーンを保持します。ClientRequestInterceptor は，BindInterceptor::bind_succeeded()メソッドの引数として渡される InterceptorManagerControl を使用して InterceptorManagerControl::get_manager("ClientRequest")を呼び出すことによって返却されます。

28.9.1 インクルードファイル

このクラスを使用するときは，interceptor_c.hh ファイルをインクルードしてください。

28.9.2 ClientRequestInterceptorManager のメソッド

- virtual void add (
  ClientRequestInterceptor_ptr interceptor);
  このメソッドを呼び出して ClientRequestInterceptor を，ローカルチェーンに追加します。
  - interceptor
    追加するインタセプタ
- virtual void remove (
  ClientRequestInterceptor_ptr interceptor);
  このメソッドは，ClientRequestInterceptorManager を削除します。
  - interceptor
    追加するインタセプタ

28.10 POALifeCycleInterceptor

```
class PortableServerExt::POALifeCycleInterceptor :
                        public virtual VISPseudoInterface
```

このクラスは，POA が生成またはデストラクトされるたびに呼び出されるグローバルインタセプタです。ほかのサーバ側インタセプタはすべてグローバルインタセプタとして，または特定の POA に対してインストールできます。POALifeCycleInterceptor は，POALifeCycleInterceptorManager クラスを使用してインストールします。POALifeCycleInterceptorManager クラスについては，「28.11 POALifeCycleInterceptorManager」を参照してください。POALifeCycleInterceptor は，POA の生成時またはデストラクト時に呼び出されます。

28.10.1 インクルードファイル

このクラスを使用するときは，PortableServerExt_c.hh ファイルをインクルードしてください。

28.10.2 POALifeCycleInterceptor のメソッド

- virtual void **create**(
  PortableServer::POA_ptr **poa**,
  CORBA::PolicyList& **policies**,
  IOP::IORValue*& **iorTemplate**,
  interceptor::InterceptorManagerControl_ptr **poaAdmin**);
  このメソッドは，新規 POA が create_POA の呼び出しによって明示的に生成されたとき，または AdapterActivator によって生成されたときに，呼び出されます。AdapterActivator の場合，インタセプタは，unknown_adapter メソッドが AdapterActivator から正常にリターンしたあとにだけ呼び出されます。create メソッドは，最近生成された POA のリファレンス，またはその POA インスタンスの POAInterceptorManager のリファレンスとして呼び出されます。
  - poa
    生成されたカレント POA に対応する ID
  - policies
    生成された POA のポリシー
  - iorTemplate
    IOR テンプレートは，type_id が未設定の完全 IOR 値で，すべての GIOP::ProfileBodyValue のオブジェクトキーが不完全になります。
  - poaAdmin

生成された POA の制御。詳細については，「28.5　InterceptorManagerControl」を参照してください。

- virtual void **destroy**(
  PortableServer::POA_ptr **poa**);
  このメソッドは，そのすべてのオブジェクトがエーテライズされている場合に，POA のデストラクト前に呼び出されます。このメソッドは，create が同じ名前の POA に対して再び呼び出される前に，destroy がすべてのインタセプタに対して必ず呼び出されるようにします。destroy オペレーションにシステム例外が発生してもそのシステム例外は無視され，残りのインタセプタは引き続き呼び出されます。
  - poa
    デストラクトされる POA

28.11 POALifeCycleInterceptorManager

```
class PortableServerExt::POALifeCycleInterceptorManager :
                    public virtual interceptor::InterceptorManager,
                    public virtual VISPseudoInterface
```

このクラスは，すべての POALifeCycle グローバルインタセプタを管理します。ORB で定義された POALifeCycleInterceptorManager にはインスタンスが一つあります。また，このクラスのスコープはグローバルスコープか，または ORB ごとのスコープです。このクラスは，VISInit から継承するローダクラスの ORB_init()時にだけアクティブです。

ORB_init()内で ORB を使用して resolve_initial_references("VisiBrokerInterceptorControl")を実行して InterceptorControlManager を取得後，InterceptorManagerControl::get.manager("POALifeCycle")を使用して InterceptorManagerControl から POALifeCycleInterceptorManager を取得します。

28.11.1 インクルードファイル

このクラスを使用するときは，PortableServerExt_c.hh ファイルをインクルードしてください。

28.11.2 POALifeCycleInterceptorManager のメソッド

- virtual void add(
  POALifeCycleInterceptor_ptr interceptor);
  このメソッドを呼び出して，POALifeCycleInterceptor のグローバルチェーンに，POALifeCycleInterceptor を追加します。
  - interceptor
    追加するインタセプタ

28.12 ActiveObjectLifeCycleInterceptor

```
class PortableServerExt::ActiveObjectLifeCycleInterceptor :
                        public virtual VISPseudoInterface
```

このクラスは，オブジェクトをアクティブオブジェクトマップに追加するときと，アクティブオブジェクトマップからオブジェクトを削除するときに呼び出されます。POA に RETAIN ポリシーがあるときだけ使用できます。また，このクラスは，POA 生成時に POALifeCycleInterceptor が POA ごとに ActiveObjectLifeCycleInterceptorManager を使用してインストールできるインタセプタです。

28.12.1 インクルードファイル

このクラスを使用するときは，PortableServerExt_c.hh ファイルをインクルードしてください。

28.12.2 ActiveObjectLifeCycleInterceptor のメソッド

- virtual void **create**(
  const PortableServer::ObjectId& **oid**,
  PortableServer::ServantBase* **servant**,
  PortableServer::POA_ptr **adapter**);
  このメソッドは，直接 API または ServantActivator を使用して実行した（明示的または暗黙的な）呼び出しによってオブジェクトがアクティブオブジェクトマップに追加されたあとに呼び出されます。オブジェクトリファレンスと新規アクティブオブジェクトの POA はパラメタとして渡されます。
  - oid
    活性化しているサーバントのオブジェクト ID
  - servant
    対応するサーバント
  - adapter
    生成中，またはデストラクト中の POA
- virtual void **destroy**(
  const PortableServer::ObjectId& **oid**,
  PortableServer::ServantBase* **servant**,
  PortableServer::POA_ptr **adapter**);
  このメソッドは，オブジェクトが非活性化されエーテライズされたあとに呼び出されます。該当するオブジェクトのオブジェクトリファレンスと POA はパラメタとして渡されます。
  - oid
    非活性化されたオブジェクトのオブジェクト ID

- servant
  対応するサーバント
- adapter
  生成中，またはデストラクト中の POA

28.13 ActiveObjectLifeCycleInterceptorManager

```
class PortableServerExt::ActiveObjectLifeCycleInterceptorManager :
                      public virtual interceptor::InterceptorManager,
                      public virtual VISPseudoInterface
```

このクラスは，カレントオブジェクトの ActiveObjectLifeCycleInterceptor のチェーンを保持します。各 POA には ActiveObjectLifeCycleInterceptorManager が一つずつあります。ActiveObjectLifeCycleInterceptor は，POALifeCycleInterceptor::create()メソッドの引数として渡される InterceptorManagerControl を使用して InterceptorManagerControl::get_manager("ActiveObjectLifeCycle")を呼び出すことによって返却されます。

28.13.1 インクルードファイル

このクラスを使用するときは，PortableServer_c.hh ファイルをインクルードしてください。

28.13.2 ActiveObjectLifeCycleInterceptorManager のメソッド

- virtual void add(
  ActiveObjectLifeCycleInterceptor
  interceptor_ptr interceptor);
  このメソッドを呼び出して ActiveObjectLifeCycleInterceptor をチェーンに追加します。
  - interceptor
    追加するインタセプタ

28.14 ForwardRequestException

```
class interceptor::ForwardRequestException :
                        public CORBA::UserException
```

この例外は ServerRequestInterceptor の preinvoke メソッドで発生させることができます。preinvoke メソッドはこの例外を発生させて，リクエストをほかのオブジェクトに転送できます。

28.14.1 ForwardRequestException の変数

- CORBA::Boolean is_permanent
  ロケーションの転送が恒久的であるかどうかを指定します。
- CORBA::Object_var forward_reference
  リクエストの転送先のオブジェクトのリファレンスを提供します。

28.15 ServerRequestInterceptor

```
class interceptor::ServerRequestInterceptor :
                        public virtual VISPseudoInterface
```

このクラスは，POALifeCycleInterceptor が POA 生成時に ServerInterceptorManager を使用してインストールできる，POA スコープのインタセプタです。このクラスを使用して，アクセス制御，サービスコンテキストの検査と挿入，およびリクエストの応答状況の変更ができます。

28.15.1 インクルードファイル

このクラスを使用するときは，interceptor_c.hh ファイルをインクルードしてください。

28.15.2 ServerRequestInterceptor のメソッド

- virtual void **preinvoke**(
  CORBA::Object_ptr **_target**,
  const char* **operation**,
  const IOP::ServiceContextList& **service_contexts**,
  CORBA::MarshalInBuffer& **payload**,
  VISClosure& **closure**);
  リクエストがアンマーシャルされる前に，ORB がこのメソッドを呼び出します。このインタセプタで例外が発生した場合，リクエストはすぐに終了させられます。このメソッドは，Servant Locator の呼び出し前に呼び出されるため，このメソッドの実行中は，サーバントを利用できないことがあります。
  - _target
    サーバにバインドされているクライアントオブジェクト
  - operation
    呼び出すオペレーションの名前を識別します。
  - service_contexts
    ORB が割り当てたサービスをすべて識別します。このサービスは OMG に規定されているものです。
  - payload
    マーシャリング済みバッファ
  - closure
    あるインタセプタメソッドが保存したデータを格納する場合があります。このデータは，ほかのインタセプタメソッドがあとで取得できます。
- virtual void **postinvoke_premarshal**(

CORBA::Object_ptr **target**,
IOP::ServiceContextList& **ServiceContextList**,
CORBA::Environment_ptr **env**,
VISClosure& **closure**);
このメソッドは，リクエストがサーバントに送信されてから，応答がマーシャリングされるまでの間に呼び出されます。ここで発生した例外には，チェーンを中断することによって対処します。この場合，リクエストはサーバに送信されないで，exception_occurred()がチェーンのすべてのインタセプタに対して呼び出されます。

- target
  サーバにバインドされているクライアントオブジェクト
- ServiceContextList
  ORB が割り当てたサービスを識別します。このサービスは OMG に規定されているものです。
- env
  発生した例外の情報を格納します。
- closure
  あるインタセプタメソッドが保存したデータを格納する場合があります。このデータは，ほかのインタセプタメソッドがあとで取得できます。

- virtual void **postinvoke_postmarshal**(
  CORBA::Object_ptr **_target**,
  CORBA::MarshalOutBuffer& **_payload**,
  VISClosure& **_closure**);
  このメソッドは，応答がマーシャリングされてから，クライアントに送信されるまでの間に呼び出されます。ここで発生した例外は無視されます。チェーン全体が呼び出されるように保証されます。
  - _target
    アプリケーションがバインドしようとしたオブジェクト
  - _payload
    マーシャリング済みバッファ
  - _closure
    あるインタセプタメソッドが保存したデータを格納する場合があります。このデータは，ほかのインタセプタメソッドがあとで取得できます。
- virtual void **exception_occurred**(
  CORBA::Object_ptr **_target**,
  CORBA::Environment_ptr **_env**,
  VISClosure& **_closure**);

インタセプタの一つで例外が発生したあとに，チェーンに残っているすべてのインタセプタに対して exception_occurred インタセプタが呼び出されると，ORB はこのメソッドを呼び出します。この呼び出し中に発生した例外によって，該当する環境の既存の例外が置き換えられます。

- _target
  サーバにバインドされているクライアントオブジェクト
- _env
  発生した例外の情報を格納します。
- _closure
  あるインタセプタメソッドが保存したデータを格納する場合があります。このデータは，ほかのインタセプタメソッドがあとで取得できます。

28.16 ServerRequestInterceptorManager

```
class interceptor::ServerRequestInterceptorManager :
                        public virtual InterceptorManager,
                        public virtual VISPseudoInterface
```

このクラスは，カレントオブジェクトの ServerRequestInterceptor のチェーンを保持します。各 POA には ServerRequestInterceptorManager が一つずつあります。

ServerRequestInterceptor は，POALifeCycleInterceptor::create()メソッドの引数として渡される InterceptorManagerControl を使用して InterceptorManagerControl::get_manager("ServerRequest")を呼び出すことによって返却されます。

28.16.1 インクルードファイル

このクラスを使用するときは，interceptor_c.hh ファイルをインクルードしてください。

28.16.2 ServerRequestInterceptorManager のメソッド

- virtual void **add**(
  ServerRequestInterceptor_ptr **interceptor**);
  このメソッドを呼び出して，ServerRequestInterceptor をチェーンに追加します。
  - interceptor
    追加するインタセプタ

28.17 IORCreationInterceptor

```
class PortableServerExt::IORCreationInterceptor :
                        public virtual VISPseudoInterface
```

このクラスは，POA 生成時に POALifeCycleInterceptor が POA ごとに IORCreationInterceptorManager を使用してインストールできるインタセプタです。このインタセプタを使用して，拡張プロファイルや拡張コンポーネントを追加することによって，IOR を修正できます。このクラスは通常，トランザクションやファイアウォールなどのサービスをサポートするために使用します。

このインタセプタは，開発時に名前とアイデンティティがわからない POA の，ある特定のクラスの IOR テンプレートを変更するために使用します。トランザクションやファイアウォールのサービスがこれに該当します。

注

POA が生成した IOR を変更するには，その POA の IOR テンプレートだけを変更してください。変更内容は，新規生成された IOR にだけ適用され，既存の IOR には適用されません。IOR を根本的に変更することはお勧めしません。

28.17.1 インクルードファイル

このクラスを使用するときは，PortableServerExt_c.hh ファイルをインクルードしてください。

28.17.2 IORCreationInterceptor のメソッド

- virtual void create(
  PortableServer::POA poa, IOP::IORValue*& ior);
  このメソッドは，POA がオブジェクトリファレンスを生成する必要がある場合にいつでも呼び出せます。このメソッドは，POA とリファレンス用の IORValue を引数として取得します。インタセプタは，拡張プロファイルや拡張コンポーネントを追加したり，既存のプロファイルやコンポーネントに変更を加えることによって，IORValue を修正できます。
  - poa
    生成対象の POA に対応する ID
  - ior
    クライアントがバインドしているサーバの IOR

28.18 IORCreationInterceptorManager

```
class PortableServerExt::IORCreationInterceptorManager :
                        public virtual interceptor::InterceptorManager,
                        public virtual VISPseudoInterface
```

このクラスは，カレントオブジェクトの IORCreationInterceptor のチェーンを保持します。各 POA には IORCreationInterceptorManager が一つずつあります。IORCreationInterceptor は，POALifeCycleInterceptor::create()メソッドの引数として渡される InterceptorManagerControl を使用して InterceptorManagerControl::get_manager("IORCreation")を呼び出すことによって返却されます。

28.18.1 インクルードファイル

このクラスを使用するときは，PortableServerExt_c.hh ファイルをインクルードしてください。

28.18.2 IORCreationInterceptorManager のメソッド

- virtual void add(
  IORCreationInterceptor_ptr _interceptor);
  このメソッドを呼び出して，IORCreationInterceptor をローカルチェーンに追加できます。
  - interceptor
    追加するインタセプタ

28.19 ExtendedClosure

```
class ExtendedClosure : public VISClosure {
public:
   interceptor::RequestInfo reqInfo;
   CORBA::MarshalInBuffer_ptr payload;
};
```

このクラスは VISClosure の派生クラスであり，read-only 属性の RequestInfo を格納しています。

IDL サンプル 28-1　RequestInfo

```
struct RequestInfo {
   CORBA::Boolean response_expected;
   CORBA::ULong request_id;
};
```

ServerRequestInterceptor に渡された Closure オブジェクトをキャストし，ClientRequestInterceptor をそのサブクラスである ExtendedClosure にキャストできます。ExtendedClosure を使用して RequestInfo を抽出し，また，その RequestInfo から request_id と response_expected を抽出できます。request_id は，リクエストに割り当てられた一意の識別子です。response_expected フラグは，リクエストが一方向呼び出しであるかどうかを識別します。

```
CORBA::Boolean response_expected =
            ((ExtendedClosure)
                        closure).reqInfo.response_expected;
CORBA::ULong request_id =
            ((ExtendedClosure)closure).reqInfo.request_id;
```

詳細については，examples/interceptor/client_server にある例を参照してください。

28.20 VISClosure

```
class VISClosure
```

このクラスは，異なるインタセプタメソッドの呼び出し間で共有できるようにデータを格納するために使用します。格納データはアンタイプドデータで，オペレーション要求，またはバインドリクエストや探索リクエストに関連する状態情報を表します。このデータは VISClosureData クラスとともに使用します。

28.20.1 インクルードファイル

このクラスを使用するときは，vclosure.h ファイルをインクルードしてください。

28.20.2 VISClosure のメンバ

- CORBA::ULong id

  複数の VISClosure オブジェクトを使用する場合，このデータメンバを使用して該当するオブジェクトを一意に識別できます。

- void *data

  このデータメンバは，インタセプタメソッドが格納したりアクセスしたりできるアンタイプドデータを指します。

- VISClosureData *managedData

  このデータメンバは，実データを表す VISClosureData クラスを指します。管理データをこの型にキャストできます。

28.21 VISClosureData

```
class VISClosureData
```

このクラスは，異なるインタセプタメソッド間で共有できる管理データを表すベースクラスです。

28.22 ChainUntypedObjectWrapperFactory

```
class VISObjectWrapper::ChainUntypedObjectWrapperFactory :
                        public UntypedObjectWrapperFactory
```

クライアントまたはサーバのアプリケーションが，UntypedObjectWrapperFactory オブジェクトを追加または削除するために，このクラスを使用します。UntypedObjectWrapperFactory は，クライアントアプリケーションがバインドするオブジェクトごとに，またはサーバアプリケーションが生成するオブジェクトインプリメンテーションごとに UntypedObjectWrapper を生成するために使用します。オブジェクトラッパーの詳細については，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「オブジェクトラッパーの使用」の記述を参照してください。

28.22.1 インクルードファイル

このクラスを使用するときは，vobjwrap.h ファイルをインクルードしてください。

28.22.2 ChainUntypedObjectWrapperFactory のメソッド

- void add(
  UntypedObjectWrapperFactory_ptr factory,Location loc);
  このメソッドは，指定したアンタイプドオブジェクトラッパーファクトリを，クライアントアプリケーション，サーバアプリケーション，またはクライアントとサーバの機能を持つアプリケーションに追加します。
  アプリケーションがクライアント，およびサーバのアプリケーションとして動作している場合，つまりクライアントとサーバの機能を持つアプリケーションである場合は，アンタイプドオブジェクトラッパーファクトリをインストールできます。このため，バインドしたオブジェクト，およびオブジェクトインプリメンテーションが受信したオペレーション要求の呼び出しに対してオブジェクトラッパーのメソッドが呼び出せます。つまり，これらのメソッドはアプリケーションのクライアント部分とサーバ部分の両方で呼び出せます。

  注

  クライアント側では，オブジェクトをバインドする前にアンタイプドオブジェクトラッパーファクトリをインストールしておいてください。サーバ側では，インプリメンテーションオブジェクトの呼び出しを受信する前にアンタイプドオブジェクトラッパーファクトリをインストールしておいてください。

  - factory
    登録するファクトリを指すポインタ
  - loc
    追加するファクトリの位置。次のどれかの値になります。

VISObjectWrapper::Client
VISObjectWrapper::Server
VISObjectWrapper::Both

- void remove(
  UntypedObjectWrapperFactory_ptr factory, Location loc);
  このメソッドは，指定したアンタイプドオブジェクトラッパーファクトリを指定した位置から削除します。
  アプリケーションがクライアントとサーバの両方で動作している場合は，オブジェクトラッパーファクトリを，クライアント側オブジェクト，サーバ側オブジェクト，またはその両方から削除できます。

  注

  オブジェクトラッパーファクトリをクライアントから削除しても，すでにクライアントからバインドされている同じクラスのオブジェクトには影響しません。しかし，削除したあとにバインドしたオブジェクトには影響します。オブジェクトラッパーファクトリをサーバから削除しても，すでにリクエストにサービスされているオブジェクトインプリメンテーションには影響しません。しかし，削除したあとに生成したオブジェクトインプリメンテーションには影響します。

  - factory
    登録するファクトリを指すポインタ
  - loc
    削除されるファクトリの位置。次のどれかの値になります。
    VISObjectWrapper::Client
    VISObjectWrapper::Server
    VISObjectWrapper::Both
- static CORBA::ULong count(
  Location loc);
  この静的メソッドは，指定位置にインストールされたアンタイプドオブジェクトラッパーファクトリの数を返します。
  - loc
    ファクトリの位置。次のどれかの値になります。
    VISObjectWrapper::Client
    VISObjectWrapper::Server
    VISObjectWrapper::Both
- static ChainUntypedObjectWrapperFactory* instance(
  CORBA::Boolean doCreate=1);
  このメソッドは，プロセスでユニークな ChainUntypedObjectWrapperFactory のインスタンスを返却します。
  - doCreate

1 を指定した場合，ChainUntypedObjectWrapperFactory のインスタンスが存在しないとき，新しくインスタンスを作成して返却します。

0 を指定した場合，ChainUntypedObjectWrapperFactory のインスタンスが存在しないとき，NULL リファレンスを返却します。

デフォルトは 1 です。

28.23 UntypedObjectWrapper

```
class VISObjectWrapper::UntypedObjectWrapper :
                        public virtual VISResource
```

このクラスは，アンタイプドオブジェクトラッパーをクライアントアプリケーション，サーバアプリケーション，または同一プロセスにあるアプリケーションに派生させ，実装するために使用します。このクラスを使用して，アンタイプドオブジェクトラッパーを派生させる場合，クライアントアプリケーションがオペレーション要求を発行する前，またはサーバ側のオブジェクトインプリメンテーションがそのオペレーション要求を処理する前に，呼び出される pre_method メソッドを，定義します。また，サーバ側のオブジェクトインプリメンテーションがオペレーション要求を処理したあと，またはクライアントアプリケーションが応答を受信したあとに，呼び出される post_method メソッドも定義します。

また，アンタイプドオブジェクトラッパーを生成するファクトリクラスも派生させてください。詳細については，「28.24 UntypedObjectWrapperFactory」を参照してください。

オブジェクトラッパーの使用方法の詳細については，マニュアル「Borland Enterprise Server VisiBroker デベロッパーズガイド」の「オブジェクトラッパーの使用」の記述を参照してください。

28.23.1 インクルードファイル

このクラスを使用するときは，vobjwrap.h ファイルをインクルードしてください。

28.23.2 UntypedObjectWrapper のメソッド

- virtual void **pre_method**(
  const char* **operation**,
  CORBA::Object_ptr **target**,
  VISClosure& **closure**);
  このメソッドは，クライアント側でオペレーション要求が送信される前，またはサーバ側のオブジェクトインプリメンテーションがそのオペレーション要求を処理する前に呼び出されるメソッドです。
  - operation
    リクエストされているオペレーションの名前
  - target
    リクエストのターゲットオブジェクト
  - closure
    オブジェクトラッパーメソッド間のデータ受け渡しに使用できる closure オブジェクト
- virtual void **post_method**(

const char* operation,
CORBA::Object_ptr target,
CORBA::Environment& env,
VISClosure& closure);
このメソッドは，サーバ側のオブジェクトインプリメンテーションがオペレーション要求を処理したあと，またはクライアント側のスタブが応答メッセージを処理する前に呼び出されるメソッドです。

- operation
  リクエストされているオペレーションの名前
- target
  リクエストのターゲットオブジェクト
- env
  オペレーション要求の処理中に発生した可能性のある例外を反映するときに使用される Environment オブジェクト
- closure
  オブジェクトラッパーメソッド間のデータ受け渡しに使用できる closure オブジェクト

28.24 UntypedObjectWrapperFactory

```
class VISObjectWrapper::UntypedObjectWrapperFactory
```

このクラスは，ユーザのアンタイプドオブジェクトラッパーファクトリを派生させるときに使用します。

新規オブジェクトのバインド時，またはオブジェクトインプリメンテーションがリクエストにサービスするときは，ユーザのファクトリを使用して，アプリケーション用のユーザのアンタイプドオブジェクトラッパーのインスタンスを生成します。

28.24.1 インクルードファイル

このクラスを使用するときは，vobjwrap.h ファイルをインクルードしてください。

28.24.2 UntypedObjectWrapperFactory のコンストラクタ

- UntypedObjectWrapperFactory(
  Location loc, CORBA::Boolean doAdd=1);
  指定された位置にアンタイプドオブジェクトラッパーファクトリを生成し，デフォルトではそれを ChainUntypedObjectWrapperFactory に登録します。アプリケーションがクライアントアプリケーションとサーバアプリケーションの両方として動作している場合は，アンタイプドオブジェクトラッパーファクトリをインストールして，バインドしたオブジェクト，およびオブジェクトインプリメンテーションが受信したオペレーション要求の，両方の呼び出しに対してオブジェクトラッパーのメソッドが呼び出せます。
  デフォルトパラメタを使わない場合，doAdd を実行しないことを指定できます。ただし，アンタイプドオブジェクトラッパーを生成するには，ChainUntypedObjectWrapper::add を呼び出す必要があります。
  - loc
    追加するファクトリの位置。次の値のどれかになります。
    VISObjectWrapper::Client
    VISObjectWrapper::Server
    VISObjectWrapper::Both
  - doAdd
    ファクトリを登録するかどうかを指定するフラグ

28.24.3 UntypedObjectWrapperFactory のメソッド

- virtual UntypedObjectWrapper_ptr **create**(
  CORBA::Object_ptr **target**,Location **loc**);
  このメソッドは，ユーザ任意の型で UntypedObjectWrapper のインスタンスを生成するときに呼び出します。このメソッドを実装すると，バインドされたオブジェクトまたはオブジェクトインプリメンテーションの型を検査し，そのオブジェクトラッパーの生成が必要かどうかを判定します。loc パラメタを使用して，create リクエストを呼び出し，クライアントオブジェクト，またはサーバインプリメンテーションを，ラッピングするかどうかを指定してください。
  - target
    アンタイプドオブジェクトラッパーを生成中のクライアントアプリケーションが，バインドしているオブジェクト。このメソッドがサーバ側で呼び出された場合，このパラメタは生成されるオブジェクトインプリメンテーションを表します。
  - loc
    追加するファクトリの位置

28.25 EventQueueManager

```
class EventQueue::EventQueueManager :
                        public virtual InterceptorManager,
                        public virtual VISPseudoInterface
```

このクラスは，カレントオブジェクトの EventQueueManager のチェーンを保持します。EventQueueManager は，必ず ORB_init()で使用してください。ORB が初期化されたあとは，無効になります。

VISInit から継承するローダクラスの ORB_init()の中で使用してください。ORB_init()内で ORB を使用して resolve_initial_references("VisiBrokerInterceptorControl")を呼び出して InterceptorControlManager を取得後，InterceptorManagerControl::get_manager("EventQueueManager")を呼び出すことによって返却されます。

28.25.1 インクルードファイル

このクラスを使用するときは，interceptor_c.hh ファイルおよび EventQueue_c.hh ファイルをインクルードしてください。

28.25.2 EventQueueManager のメソッド

- void **register_listener**(
  EventListener_ptr _**listener**,EventType _**type**);
  このオペレーションは，指定のイベントタイプのイベントリスナーの登録用に用意されています。
- void **unregister_listener**(
  EventListener_ptr _listener,EventType _**type**);
  このオペレーションは，事前に登録された指定のタイプのリスナーを削除します。
- EventListeners* **get_listeners**(
  EventType _**type**);
  このオペレーションは，指定のタイプの登録済みイベントリスナーのリストを返します。

28.26 ConnEventListeners

```
class EventQueue::ConnEventListeners :public virtual EventListener,
                        public virtual VISPseudoInterface
```

28.26.1 インクルードファイル

このクラスを使用するときは，interceptor_c.hh ファイルおよび EventQueue_c.hh ファイルをインクルードしてください。

28.26.2 ConnEventListeners のメソッド

- void conn_established(
  const ConnInfo& _info)
  このオペレーションは VisiBroker ORB によってコールバックされ，コネクション設定イベントをプッシュします。VisiBroker ORB は in ConnInfo info パラメタにクライアントコネクション情報を与えて，この値をコールバックオペレーションに渡します。
- void conn_closed(
  const ConnInfo& _info)
  このオペレーションは VisiBroker ORB によってコールバックされ，コネクションクローズイベントをプッシュします。VisiBroker ORB は in ConnInfo info パラメタにクライアントコネクション情報を与えて，この値をコールバックオペレーションに渡します。
  サーバ側アプリケーションは，リスナーにプッシュされているイベントの処理と同様に，ConnEventListeners インタフェースのインプリメンテーションにも責任があります。

28.27 ConnInfo

```
struct ConnInfo
```

28.27.1 インクルードファイル

この構造体を使用するときは，interceptor_c.hh ファイルおよび EventQueue_c.hh ファイルをインクルードしてください。

28.27.2 ConnInfo のメンバ

- CORBA::String_var ipaddress;
  "xxx.xxx.xxx.xxx"のフォーマットで通信相手の IP アドレスを格納します。
- CORBA::Long port;
  通信相手のポート番号を格納します。
- CORBA::Long connID;
  このクライアントコネクションのサーバごとの一意の識別子を格納します。

29 QoS インタフェースとクラス（C++）

この章では，Borland Enterprise Server VisiBroker の C++言語での QoS インタフェースとクラスについて説明します。

ポリシーの生成の詳細については，「22.11　PortableServer::POA」を参照してください。

29.1 概要

QoS（Quality of Service）API の Borland Enterprise Server VisiBroker でのインプリメンテーションについて説明します。QoS API によって，ポリシーを使用して，ユーザのクライアントアプリケーションとサーバとの間のコネクションを定義したり管理したりできます。ポリシーの作成方法については，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「QoS インタフェース」の記述を参照してください。

QoS は，VisiBroker ORB レベルのポリシー，スレッドレベルのポリシー，およびオブジェクトレベルのポリシーを管理するために，それぞれ次のクラスを提供しています。

- VisiBroker ORB レベルのポリシー

  VisiBroker ORB レベルのポリシーは，ローカルの PolicyManager によって適用できます。PolicyManager によって，ポリシーを設定したり，現在のポリシーのオーバーライドを参照できます。システムデフォルトをオーバーライドする VisiBroker ORB レベルでのポリシーです。

- スレッドレベルのポリシー

  スレッドレベルのポリシーは，PolicyCurrent によって適用できます。PolicyCurrent は，スレッドレベルでのポリシーのオーバーライドを参照したり設定したりするための各種オペレーションを格納しています。システムデフォルトと VisiBroker ORB レベルで設定した値をオーバーライドするスレッドレベルでのポリシーです。

- オブジェクトレベルのポリシー

  オブジェクトレベルのポリシーは，継承元 Object インタフェースの QoS オペレーションにアクセスすることによって適用できます。システムデフォルト，VisiBroker ORB レベル，およびスレッドレベルで設定した値をオーバーライドするオブジェクトレベルでのポリシーです。

29.2 CORBA::PolicyManager

```
class CORBA::PolicyManager
```

このクラスは，VisiBroker ORB レベルでのポリシーオーバーライドの設定とアクセスのために使用します。VisiBroker ORB レベルで定義されているポリシーは，システムデフォルトをオーバーライドします。マネージャスレッドに属しているインスタンスは，resolve_initial_references("ORBPolicyManager")を使用して CORBA::PolicyManager にナロウイングすることでアクセスできます。

29.2.1 IDL の定義

```
module CORBA {
      interface PolicyManager {
      PolicyList get_policy_overrides(in PolicyTypeSeq ts);
      void set_policy_overrides(in PolicyList policies,
                                      in SetOverrideType set_add)
          raises (InvalidPolicies);
   };
};
```

29.2.2 CORBA::PolicyManager のメソッド

- CORBA::PolicyList *get_policy_overrides(
  const CORBA::PolicyTypeSeq& ts);
  このメソッドは，要求された Policy 型のポリシーを含むリストを返します。空シーケンスを指定した場合，つまりリストの長さが 0 の場合，該当するスコープのポリシーをすべて返します。要求された Policy 型がターゲット PolicyManager に設定されていない場合，空シーケンスを返します。
- void set_policy_overrides(
  const CORBA::PolicyList& policies,
  CORBA::SetOverrideType set_add);
  このメソッドは，要求されたポリシーオーバーライドのリストを使用して現在の一連のポリシーを更新します。PolicyManager からオーバーライドをすべて削除するには，ポリシーに空シーケンス，および処理モードに SET_OVERRIDE を指定して set_policy_overrides を呼び出してください。
  このオペレーションを使用してオーバーライドできるポリシーは，クライアント側でのオペレーションの呼び出しに関係するポリシーだけです。それ以外のポリシーをオーバーライドしようとすると，CORBA::NO_PERMISSION 例外が発生します。このメソッドの呼び出しが原因で，ターゲット PolicyManager の一連のオーバーライドポリシーに矛盾が生じる場合は，ポリシーの変更も追加もされないで，InvalidPolicies 例外が発生します。ほかの PolicyManager に設定されたポリシーとの互換性はチェックされません。
  - policies

Policy オブジェクトのリファレンスのシーケンス

- set_add

  ポリシーを，PolicyManager にすでに存在するほかのオーバーライドに追加（ADD_OVERRIDE）するか，またはオーバーライドがない PolicyManager に追加（SET_OVERRIDE）するかを示します。

29.3 CORBA::PolicyCurrent

```
class CORBA::PolicyCurrent : public CORBA::Current,
                             public CORBA::PolicyManager
```

このクラスは，スレッドレベルでオーバーライドされたポリシーへのアクセスを提供するものです。このクラスは，QoS 値を問い合わせてスレッドに適用するためのオペレーションを使用して定義されます。スレッドレベルで定義されたポリシーは，任意のシステムデフォルト値，または VisiBroker ORB レベルで設定された値をオーバーライドできます。ただし，オブジェクトレベルの値はオーバーライドできません。カレントスレッドに属するインスタンスにアクセスするには，resolve_initial_references("PolicyCurrent") を使って，CORBA::PolicyCurrent にナロウしてください。

29.3.1 IDL の定義

```
#pragma prefix ”omg.org”
module CORBA {
    interface PolicyCurrent : PolicyManager, Current {
    };
};
```

29.4 CORBA::Object

```
class CORBA::Object
```

Borland Enterprise Server VisiBroker の QoS API のインプリメンテーションは，オブジェクト，スレッド，および VisiBroker ORB に，ポリシーを割り当てられるようにします。オブジェクトに割り当てられたポリシーは，ほかのポリシーをすべてオーバーライドします。

29.4.1 IDL の定義

```
#pragma prefix "omg.org"
module CORBA {
   interface Object {
      Policy get_client_policy(in PolicyType type);
      Policy get_policy(in PolicyType type);
      PolicyList get_policy_overrides(in PolicyTypeSeq types);
      Object set_policy_overrides(in PolicyList policies,
                                  in SetOverrideType set_add)
         raises (InvalidPolicies);
      boolean validate_connection(out PolicyList
                  inconsistent_policies);
   };
};
```

29.4.2 CORBA::Object のメソッド

- CORBA::Policy_ptr **get_client_policy**(
  CORBA::PolicyType **type**);
  このメソッドは，オブジェクトリファレンスの有効なオーバーライドポリシーを返します。まず，Object スコープに指定した PolicyType のオーバーライドがあるかどうかをチェックし，次に Current スコープ，最後に VisiBroker ORB スコープをチェックすると取得できます。要求された PolicyType のオーバーライドがない場合，その PolicyType のシステム依存のデフォルト値が使用されます。デフォルトのポリシー値は指定されていないので，ポータブルアプリケーションは，VisiBroker ORB スコープで必要なデフォルトを設定してください。
- CORBA::Policy_ptr **get_policy**(
  CORBA::PolicyType **type**);
  このメソッドは，オブジェクトリファレンスの有効ポリシーを返します。有効ポリシーは，リクエストの発行時に使用するポリシーです。まず，get_client_policy が返す PolicyType の有効オーバーライドを取得します。次に，IOR 指定のポリシーと有効オーバーライドを比較します。有効ポリシーは，有効オーバーライドと IOR 指定のポリシーが許容する値の共通部分です。共通部分が空の場合は，INV_POLICY システム例外が発生します。空でない場合は，共通部分に正しく入っているポリシーを有効ポリシーとして返します。IOR にポリシー値の指定がない場合は，正しい任意の値を使用できま

す。有効なポリシーを確実に返すには，get_policy メソッドを呼び出す前に，オブジェクトリファレンスに対して_non_existent メソッド，または validate_connection メソッドを呼び出してください。オブジェクトリファレンスをバインドする前に get_policy を呼び出すと，実装に依存した有効なポリシーが返されます。この状況では，仕様に準拠したインプリメンテーションは次のどれかの動作をする可能性があります。

- CORBA::BAD_INV_ORDER 例外を発生する
- バインドの実行後に変更される可能性のある PolicyType の値を返す
- バインドを試みてから有効ポリシーを返す

RebindPolicy の値が TRANSPARENT である場合は，透過的なリバインドによって，有効ポリシーは呼び出しごとに変わる可能性があります。

注

Borland Enterprise Server VisiBroker インプリメンテーションでは，このメソッドはオブジェクト，スレッド，および VisiBroker ORB に割り当てられたポリシーを取得します。

- CORBA::PolicyList *get_policy_overrides(
  const CORBA::PolicyTypeSeq& ts);
  このメソッドは，要求された Policy 型のポリシーを含むリストを返します。空シーケンスを指定した場合，つまりリストの長さが 0 の場合，該当するスコープのポリシーをすべて返します。要求された Policy 型がターゲット PolicyManager に設定されていない場合，空シーケンスを返します。
- CORBA::Object_ptr set_policy_overrides(
  const PolicyList& _policies, CORBA::SetOverrideType _set_add);
  このメソッドは，指定されたポリシーのリストを反映した CORBA::Object の複製を返します。
  - policies
    Policy オブジェクトのリファレンスのシーケンス
  - set_add
    ポリシーを，Object にすでに存在するほかのオーバーライドに追加（ADD_OVERRIDE）するか，またはオーバーライドがない Object に追加（SET_OVERRIDE）するかを示します。
- CORBA::Boolean validate_connection(
  CORBA::PolicyList& inconsistent_policies);
  オブジェクトの現在の有効ポリシーで呼び出せる場合，このメソッドは TRUE を返します。オブジェクトリファレンスがバインド済みでない場合，オペレーションの一環としてバインディングをします。オブジェクトリファレンスがバインド済みの場合でも，現在のポリシーオーバーライドが変更されていたり，バインディングが無効になっている場合は，RebindPolicy オーバーライドの設定に関係なくリバインドされます。現在の有効 RebindPolicy が暗黙的リバインドを許可していない場合，そのようなリバインドを強制的にできるのは validate_connection オペレーションだけです。バインドやリバインドを試みると，VisiBroker ORB によって GIOP の LocateRequests 処理がされます。
  現在の有効ポリシーで呼び出すと INV_POLICY システム例外が発生する場合，このメソッドは FALSE を返します。現在の有効ポリシーに不具合がある場合，不具合を生じさせるポリシーを，out パラメタ

の inconsistent_policies に格納します。この返されたポリシーのリストは，すべての原因を網羅しているわけではありません。ポリシーのオーバーライドに関係ない原因によってバインディングが失敗した場合は，その原因に応じたシステム例外が発生します。

29.5 Messaging::RebindPolicy

```
class Messaging::RebindPolicy : public virtual CORBA::Policy,
                                public virtual CORBA::Object
```

Borland Enterprise Server VisiBroker の RebindPolicy は，CORBA 2.5 の仕様に従って実装されています。また，Borland Enterprise Server VisiBroker では，オブジェクト障害後，osagent を使用したほかのオブジェクトの呼び出しをサポートするために拡張しています。

VisiBroker ORB の RebindPolicy は，GIOP のロケーションフォワードメッセージ，およびオブジェクト障害を，VisiBroker ORB がどのように処理するかを決定します。VisiBroker ORB は，CORBA::Object インスタンスにある有効ポリシーを参照して，オブジェクト障害後，osagent を使用したほかのオブジェクトの呼び出しやリバインドを処理します。

CORBA::Policy から派生した OMG インプリメンテーションは，ターゲットサーバに正常にバインドされた場合，VisiBroker ORB が透過的にリバインドをするかどうかを決定します。この継承インプリメンテーションは，ターゲットオブジェクト，スレッド，または VisiBroker ORB に正常にバインドされた場合，VisiBroker ORB が透過的にオブジェクト障害後，osagent を使用したほかのオブジェクトの呼び出しをするかどうかを決定します。

29.5.1 IDL の定義

```
#pragma prefix "omg.org"
module Messaging {
   typedef short RebindMode;
   const CORBA::PolicyType REBIND_POLICY_TYPE = 23;
   interface RebindPolicy CORBA::Policy {
      readonly attribute RebindMode rebind_mode;
   };
};
```

29.5.2 ポリシーの値

ポリシーは，正常にバインドされたあとにだけ使用できます。

リバインドポリシーとして設定できる OMG ポリシー値を次に示します。

- TRANSPARENT

  このポリシーは，リモートリクエスト実行時に，オブジェクトの転送と必要な再接続を VisiBroker ORB が暗黙的に行うことを許可します。これは，最も制限を加えない値です。
- NO_REBIND

このポリシーは，リモートリクエスト実行時に，クローズしたコネクションの再開を VisiBroker ORB が暗黙的に行うことを許可します。ただし，クライアント側の有効 QoS ポリシーに変更を加えるような，透過的なオブジェクトの転送は許可しません。

- NO_RECONNECT

  このポリシーは，オブジェクト転送とクローズしたコネクションの再開を，VisiBroker ORB が暗黙的に行うことを禁止します。これは，最も制限を加える値です。

リバインドポリシーとして設定できる，Borland Enterprise Server VisiBroker 固有の値を次に示します。

- VB_TRANSPARENT

  このポリシーは，TRANSPARENT 動作を継承します。これはデフォルトポリシーです。

  このポリシーを設定した場合，サーバオブジェクトのダウンが原因でリモート呼び出しが失敗すると，VisiBroker ORB は osagent を使用してほかのサーバとの再接続を試みます。VisiBroker ORB は通信障害をマスクし，呼び出し元のクライアントで例外を発生させません。

- VB_NOTIFY_REBIND

  このポリシーは，VB_TRANSPARENT とほぼ同様の動作をしますが，通信障害を検知した場合に例外を発生させます。再度呼び出しをすると，このポリシーは，ほかのオブジェクトとの透過的な再接続を試みます。

- VB_NO_REBIND

  このポリシーは，オブジェクト障害後，osagent を使用してほかのオブジェクトを呼び出すことを有効にしません。また，オブジェクトの転送も許可しません。クライアント VisiBroker ORB が，同じサーバに対してクローズしたコネクションを再開することだけを許可します。

29.6 QoSExt::DeferBindPolicy

```
class QoSExt::DeferBindPolicy : public virtual CORBA::Policy,
                                public virtual CORBA::Object
```

DeferBindPolicy は，リモートオブジェクトが最初に生成されたときに，VisiBroker ORB がオブジェクトに接続する契機を決定します。

FALSE が設定された場合は，bind()の発行時，またはリクエストが呼び出されたときに接続します。TRUE が設定された場合は，bind()の発行時には接続しないで，最初にリクエストが呼び出されたときまで接続を延期します。

デフォルトは FALSE です。

29.6.1 IDL の定義

```
#pragma prefix ”inprise.com”
module QoSExt {
   const CORBA::PolicyType DEFER_BIND_POLICY_TYPE = 0x56495305;
   interface DeferBindPolicy :CORBA::Policy {
      readonly attribute boolean value;
   };
};
```

29.7 QoSExt::ExclusiveConnectionPolicy

```
class QoSExt::ExclusiveConnectionPolicy : public virtual CORBA::Policy,
                                          public virtual CORBA::Object
```

ExclusiveConnectionPolicy は，指定したサーバオブジェクトとの排他接続（共有でない接続）を確立するための，VisiBroker 固有のポリシーです。このポリシーは，true または false のブール値を持ちます。true が設定された場合は，サーバオブジェクトへの排他接続をオープンします。false が設定された場合で既存のコネクションを再使用できるときは，既存のコネクションを再使用します。既存のコネクションを再使用できないときは，新しいコネクションをオープンします。デフォルトは false です。

このポリシーは，VisiBroker 3.x の CORBA::Object::_clone()と同じ動作をします。

このポリシーが有効になるのは次のような場合です。

- _bind()前に ORB または PolicyCurrent に対して Policy を設定している場合
- string_to_object で作成した Object を呼び出す前に，ORB または PolicyCurrent に対して Policy を設定している場合
- string_to_object や_bind で作成した Object に対して Policy を設定し，Policy を設定された Object に対してリクエストを行う場合

注

このポリシーは，プロセス内通信には適用されません。

29.7.1 IDL の定義

```
module QoSExt {
  const CORBA::PolicyType EXCLUSIVE_CONNECTION_POLICY_TYPE = 0x56495320;
  interface ExclusiveConnectionPolicy :CORBA::Policy {
    /** Returns the current setting of */
    /** the ExclusiveConnectionPolicy  */
    readonly attribute boolean value;
  };
};
```

29.8 QoSExt::RelativeConnectionTimeoutPolicy

```
class QoSExt::RelativeConnectionTimeoutPolicy :
                        public virtual CORBA::Policy,
                        public virtual CORBA::Object
```

RelativeConnectionTimeoutPolicy には，利用できるサーバのオブジェクトへの接続タイムアウト値を指定します。

値は 100 ナノ秒単位で指定します（1 秒の場合 10000000 と指定）。

このタイムアウト値が示す時間を過ぎても接続できないときは，接続を中止します。タイムアウト値は VisiBroker ORB が接続しようとするすべてのサーバに適用されます。したがって，複数のコネクションを試みた場合，タイムアウト値は設定値の倍数となります。

このポリシーの領域は，CORBA::ULongLong で表されます。

デフォルトは 0 で，OS のデフォルトタイムアウト値が設定されます。

注

このポリシーは，プロセス内通信には適用されません。

29.8.1 IDL の定義

```
module QoSExt {
   const CORBA::PolicyType RELATIVE_CONN_TIMEOUT_POLICY_TYPE
                                                 = 0x56495304;
   interface RelativeConnectionTimeoutPolicy : CORBA::Policy {
      readonly attribute TimeBase::TimeT relative_expiry;
   };
};
```

29.9 Messaging::RelativeRequestTimeoutPolicy

```
class Messaging::RelativeRequestTimeoutPolicy :
                        public virtual CORBA::Policy,
                        public virtual CORBA::Object
```

RelativeRequestTimeoutPolicy には，クライアントがリクエストを送信完了待ちする最大時間を指定します。

値は 100 ナノ秒単位で指定します（1 秒の場合 10000000 と指定）。

リクエストがタイムアウトした場合，CORBA::COMM_FAILURE, CORBA::TRANSIENT, または CORBA::TIMEOUT 例外が発生し，サーバへのコネクションは破棄される場合があります。

このポリシーの領域は，CORBA::ULongLong で表されます。

デフォルトは 0 で，クライアントはシステムおよびネットワークが許す限り，リクエストの送信が完了するのを待ち続けることを意味します。

注

- このポリシーはプロセス内通信には適用されません。
- 「29.10　Messaging::RelativeRoundtripTimeoutPolicy」の注も参照してください。

29.9.1 IDL の定義

```
module Messaging {
    const CORBA::PolicyType RELATIVE_REQ_TIMEOUT_POLICY_TYPE = 31;
    interface RelativeRequestTimeoutPolicy : CORBA::Policy {
        readonly attribute TimeBase::TimeT relative_expiry;
    };
};
```

29.10 Messaging::RelativeRoundtripTimeoutPolicy

```
class Messaging::RelativeRoundtripTimeoutPolicy :
                        public virtual CORBA::Policy,
                        public virtual CORBA::Object
```

RelativeRoundtripTimeoutPolicy には，クライアントがリクエストの送信および受信の完了を待機する最大時間を指定します。

値は 100 ナノ秒単位で指定します（1 秒の場合 10000000 と指定）。

リクエストがタイムアウトした場合，CORBA::COMM_FAILURE, CORBA::TRANSIENT，または CORBA::TIMEOUT 例外が発生し，サーバへのコネクションは破棄される場合があります。

このポリシーの領域は，CORBA::ULongLong で表されます。

デフォルトは 0 で，クライアントはシステムおよびネットワークが許す限り，リクエストの送信および受信の完了を待ち続けることを意味します。

注

- このポリシーはプロセス内通信には適用されません。
- RelativeRequestTimeoutPolicy ＜ RelativeRoundtripTimeoutPolicy の関係でタイムアウト値を指定した場合，リクエストの要求完了待機には RelativeRequestTimeoutPolicy が使用され，リクエストの受信完了待機には（RelativeRoundtripTimeoutPolicy－リクエストの要求完了まで待機した時間）が使用されます。ただし，RelativeRequestTimeoutPolicy に 0 を指定するか，または値を指定しなかった場合，リクエストの要求完了待機には RelativeRoundtripTimeoutPolicy が使用され，リクエストの受信完了待機には（RelativeRoundtripTimeoutPolicy－リクエストの要求完了まで待機した時間）が使用されます。
- RelativeRequestTimeoutPolicy ≧ RelativeRoundtripTimeoutPolicy の関係でタイムアウト値を指定した場合，リクエストの要求完了待機には RelativeRoundtripTimeoutPolicy が使用され，リクエストの受信完了待機には（RelativeRoundtripTimeoutPolicy－リクエストの要求完了まで待機した時間）が使用されます。ただし，RelativeRoundtripTimeoutPolicy に 0 を指定するか，または値を指定しなかった場合，リクエストの要求完了待機には RelativeRequestTimeoutPolicy が使用され，システムおよびネットワークが許す限り，リクエストの受信完了を待機し続けます。

  この動作は，今後のバージョンで変更になる可能性がありますので，RelativeRequestTimeoutPolicy ＜ RelativeRoundtripTimeoutPolicy の関係でタイムアウト値を指定することをお勧めします。

29.10.1　IDL の定義

```
module Messaging {
    const CORBA::PolicyType RELATIVE_RT_TIMEOUT_POLICY_TYPE = 32;
    interface RelativeRoundtripTimeoutPolicy : CORBA::Policy {
        readonly attribute TimeBase::TimeT relative_expiry;
    };
};
```

29.11 Messaging::SyncScopePolicy

```
class Messaging::SyncScopePolicy: public virtual CORBA::Policy,
```

```
public virtual CORBA::Object
```

SyncScopePolicy は，ターゲットに関するリクエストの同期レベルを定義します。

SyncScope 型の値は，一方向オペレーションの動作を制御するために，SyncScopePolicy とともに使用されます。

SyncScopePolicy のデフォルトは，SYNC_WITH_TRANSPORT です。

アプリケーションは，VisiBroker ORB インプリメンテーションのポータビリティを確保するために，明示的に VisiBroker ORB レベルの SyncScopePolicy を設定する必要があります。

SyncScopePolicy のインスタンスが作成される場合，Messaging::SyncScope 型の値は CORBA::ORB::create_policy に渡されます。

注

このポリシーはクライアント側の変更だけ適用できます。

29.11.1 IDL の定義

```
module Messaging {
    const CORBA::PolicyType SYNC_SCOPE_POLICY_TYPE = 24;
    interface SyncScopePolicy : CORBA::Policy {
        readonly attribute SyncScope synchronization;
    };
};
```

29.11.2 SyncScope ポリシーの値

SyncScope ポリシーの値と動作を次の表に示します。

表 29-1 SyncScope のポリシー値（C++）

ポリシー値	説明
SYNC_NONE	VisiBroker ORB は，リクエストメッセージをトランスポートプロトコルに渡す前に（例えば，メソッド呼び出しから），クライアントに制御を返します。クライアントは，動作を抑止しないことが保障されます。サーバから応答はないため，このレベルの同期ではロケーションフォワードは実行されません。

ポリシー値	説明
SYNC_WITH_SERVER	サーバ側 VisiBroker ORB は，対象インプリメンテーションを呼び出す前に応答を送信します。NO_EXCEPTION の応答が送信された場合，必要なロケーションフォワードがすでに発生しています。この応答受信時に，クライアント側 VisiBroker ORB は，制御をクライアントアプリケーションに返します。クライアントは，ロケーションフォワードがすべて完了するまで動作を抑止します。POA を使用しているサーバの場合，応答は，ServantManager が呼び出されてから対象サーバントにリクエストが渡されるまでの間に送信されます。
SYNC_WITH_TARGET	CORBA 2.2 の，同期型非一方向オペレーションと同等です。サーバ側 VisiBroker ORB は，オペレーション呼び出し先でオペレーションが完了したあとに，応答メッセージの送信だけを実行します。LOCATION_FORWARD の応答は，オペレーション呼び出し前に送信済みです。SYSTEM_EXCEPTION の応答は，例外の構文に従った任意の時間に送信できる状態となります。一方向と定義されている場合でも，実際は，オペレーションは同期オペレーションと同様に動作します。この形式の同期では，対象オペレーションがリクエストを解釈し，そのリクエストに従って実行したことをクライアントが認識していることが保障されます。CORBA 2.2 と同様，この最高レベルの同期では，OTS だけを使用できます。これより低いレベルの同期で呼び出されたオペレーションでは，対象はクライアントのカレントトランザクションに入れません。
SYNC_WITH_TRANSPORT	VisiBroker ORB は，リクエストメッセージをトランスポートプロトコルに渡したあとに，クライアントに制御を返します。 サーバから応答はないため，このレベルの同期ではロケーションフォワードは実行されません。

29.12 QoS 例外

QoS で発生する例外について，次の表に示します。

表 29–2 QoS で発生する例外（C++）

例外	説明
CORBA::INV_POLICY	ポリシーオーバーライド間に互換性がない場合に発生します。
CORBA::REBIND	RebindPolicy に NO_REBIND，NO_RECONNECT，または VB_NOTIFY_REBIND を設定し，バインドされたオブジェクトリファレンスの呼び出しがオブジェクトフォワードメッセージまたはロケーションフォワードメッセージとなった場合に発生します。
CORBA::PolicyError	指定したポリシーがサポートされていない場合に発生します。

30

IOP および IIOP のインタフェースとクラス（C++）

この章では，General Inter-ORB Protocol の主なインタフェース，および CORBA の仕様で定義された，その他の構造体の Borland Enterprise Server VisiBroker でのインプリメンテーションについて，C++言語でのインタフェースを説明します。

30.1 GIOP::MessageHeader

```
struct MessageHeader
```

この構造体は，GIOP メッセージの情報を表すために使用します。

30.1.1 MessageHeader のメンバ

- CORBA::Char magic[4]
  この文字列は常に"GIOP"を格納します。
- Version GIOP_version
  使用されているプロトコルのバージョンを示します。この構造体は，次に示すように，メジャーバージョン番号およびマイナーバージョン番号が格納されます。メジャーバージョンは 1，マイナーバージョンは 2 を設定します。ただし，VisiBroker 3.x などの古いバージョンを使用している場合は，マイナーバージョンは 0 を設定します。

```
struct Version {
      CORBA::Octet major;
      CORBA::Octet minor;
};
```

- CORBA::Boolean byte_order
  TRUE を設定すると，メッセージにリトルエンディアンのバイトオーダが使用されます。FALSE を設定すると，メッセージにビッグエンディアンのバイトオーダが使用されます。
- CORBA::Octet message_type
  ヘッダに続くメッセージの型を示します。次のどれかの値を持ちます。

```
enum MsgType {
      Request,
      Reply,
      CancelRequest,
      LocateRequest,
      LocateReply,
      CloseConnection,
      MessageError,
      Fragment
    };
```

- CORBA::ULong message_size
  ヘッダに続くメッセージの長さを示します。

30.2 GIOP::CancelRequestHeader

```
struct CancelRequestHeader
```

この構造体は，キャンセルするリクエストのメッセージヘッダの情報を表すために使用します。

30.2.1 CancelRequestHeader のメンバ

- CORBA::ULong request_id
  このデータメンバは，キャンセルされるリクエスト識別子を表します。

30.3 GIOP::LocateReplyHeader

```
struct LocateReplyHeader
```

この構造体は，ロケートリクエストメッセージへの応答として送信されるメッセージを表すために使用します。locate_status に OBJECT_FORWARD を設定すると，このヘッダのあとに拡張データが続きます。

30.3.1 LocateReplyHeader のメンバ

- CORBA::ULong request_id
  元のリクエストのリクエスト識別子です。
- LocateStatusType locate_status
  ロケートリクエストの性質を表します。次の値のどれかとなります。
  - UNKNOWN_OBJECT
    リクエストされたオブジェクトが見つからなかったことを示します。このメッセージに対応するデータはほかにありません。
  - OBJECT_HERE
    オブジェクトがこのサーバによって実装されることを示します。このメッセージに対応するデータはほかにありません。
  - OBJECT_FORWARD
    応答に含まれるオブジェクトリファレンス（IOR）が，ロケートリクエストメッセージに指定されたオブジェクトへのリクエストのターゲットとして使用できることを示します。オブジェクトはほかのサーバによって実装され，そのサーバの IOR がヘッダのあとに続きます。
  - OBJECT_FORWARD_PERM
    応答に含まれるオブジェクトリファレンス（IOR）が，ロケートリクエストメッセージに指定された，オブジェクトへのリクエストのターゲットとして使用できることを示します。
  - LOC_SYSTEM_EXCEPTION
    マーシャリング済みの GIOP::System ExceptionReplyBody が例外に格納されていることを示します。
  - LOC_NEEDS_ADDRESSING_MODE
    ロケートリクエストを再送するときに，リクエストされたアドレッシングモードを使用することを示します。

30.4 GIOP::LocateRequestHeader

```
struct LocateRequestHeader
```

この構造体は，オブジェクトを探索するためのリクエストを含むメッセージを表します。

30.4.1 LocateRequestHeader のメンバ

- CORBA::ULong request_id
  このメッセージのリクエスト識別子を表し，複数のアウトスタンディングメッセージを識別するために使用します。
- GIOP::TargetAddress target
  探索対象のオブジェクトを表します。ターゲットとなるのはオブジェクトキー，プロファイル，および IOR の三つから構成される union です。

30.5 GIOP::ReplyHeader

```
struct ReplyHeader
```

この構造体は，リクエストメッセージへの応答として，クライアントに送信される応答メッセージの応答ヘッダを表します。

30.5.1 インクルードファイル

この構造体を使用するときは，giop_c.hh ファイルをインクルードしてください。このファイルは，インストール/インクルードディレクトリの corba.h にすでにインクルードされています。

30.5.2 ReplyHeader のメンバ

- CORBA::ULong request_id
  この応答が対応しているリクエストメッセージと同一の request_id を設定します。
- ReplyStatusType reply_status
  応答の状態を示します。次の enum 値のどれかを設定します。
  - NO_EXCEPTION
  - USER_EXCEPTION
  - SYSTEM_EXCEPTION
  - LOCATION_FORWARD
  - LOCATION_FORWARD_PERM
  - NEEDS_ADDRESSING_MODE
- IOP::ServiceContextList service_info
  サーバからクライアントに渡せるサービスコンテキスト情報のリスト

30.6 GIOP::RequestHeader

```
struct RequestHeader
```

この構造体は，オブジェクトインプリメンテーションに送信されるリクエストメッセージのリクエストヘッダを表します。

30.6.1 インクルードファイル

この構造体を使用するときは，giop_c.hh ファイルをインクルードしてください。このファイルは，インストール/インクルードディレクトリの corba.h にすでにインクルードされています。

30.6.2 RequestHeader のメンバ

- CORBA::ULong request_id
  応答メッセージと特定のリクエストメッセージを対応づけるために使用する一意の識別子です。
- CORBA::Boolean response_expected
  リクエストが応答を受信することがない一方向オペレーションである場合，このメンバは FALSE に設定されます。オペレーション要求およびその他のリクエストが応答を受信する場合は，このメンバは TRUE に設定されます。
- GIOP::TargetAddress _target
  リクエストのターゲットであるオブジェクトです。ターゲットとなるのはオブジェクトキー，プロファイル，および IOR の union です。オブジェクトキーはベンダ固有のフォーマットで格納され，IOR が生成された際に生成されます。
- CORBA::String_var operation
  ターゲットオブジェクトに対してリクエストされているオペレーションを識別します。このメンバが管理型であるという以外は，operator メンバとまったく同じです。
- const char *operation
  ターゲットオブジェクトに対してリクエストされているオペレーションを識別します。このメンバが管理型ではないという以外は，oper メンバとまったく同じです。
- IOP::ServiceContextList service_context
  クライアントからサーバに渡せるサービスコンテキスト情報のリストです。

30.7 IIOP::ProfileBody

```
struct ProfileBody
```

この構造体は，オブジェクトによってサポートされているプロトコルの情報を格納します。

```
module IIOP {
    . . .
    struct ProfileBody {
      Version iiop_version;
      string host;
      unsigned short port;
      sequence<octet> object_key;
      sequence<IOP::TaggedComponent> components;
    };
};
```

30.7.1 ProfileBody のメンバ

- Version iiop_version
  サポートされている IIOP のバージョンです。
- CORBA::String_var host
  オブジェクトのホストになるサーバを実行中のホスト名です。
- CORBA::UShort port
  オブジェクトへのコネクションを確立するために使用するポート番号です。
- CORBA::OctetSequence object_key
  オブジェクトキーはベンダ固有のフォーマットで格納され，IOR が生成された際に生成されます。
- IIOP::MultiComponentProfile components
  サポートしているプロトコルに関する情報を格納する TaggedComponent のシーケンスです。

30.8 IOP::IOR

```
struct IOR
```

この構造体は，IOR を表し，オブジェクトリファレンスの情報を提供するために使用します。クライアントアプリケーションは ORB::object_to_string メソッドを呼び出して文字列化 IOR を生成できます。詳細については，「22.9.2 CORBA::ORB のメソッド」を参照してください。

30.8.1 インクルードファイル

この構造体を使用するときは，giop_c.hh ファイルをインクルードしてください。

30.8.2 IOR のメンバ

- CORBA::String_var type_id
  このデータメンバは，この IOR が表すオブジェクトリファレンスの型を記述します。
- TaggedProfileSeq profiles
  このデータメンバは，一つ以上の TaggedProfile 構造体のシーケンスを表します。その構造体は，サポートされているプロトコルについての情報を格納します。

30.9 IOP::TaggedProfile

```
struct TaggedProfile
```

この構造体は，IOR によってサポートされている，特定のプロトコルを表します。

30.9.1 TaggedProfile のメンバ

- ProfileID tag
  このデータメンバは，プロファイルデータの内容を表します。次の値のどれかになります。
  - TAG_INTERNET_IOP
    プロトコルが IIOP であることを示します。
  - TAG_MULTIPLE_COMPONENTS
    プロファイルデータがそのプロトコルで使用できる VisiBroker ORB サービスのリストを含んでいることを示します。
  - TAG_VB_LOCATOR
    IOR は，osagent が本当の IOR を受信するまで使用される暫定の擬似オブジェクトであることを示します。
  - TAG_VSGN_LIOP
    プロトコルがローカル IPC メカニズムに適用されている IOP であることを示します。
- CORBA::OctetSequence profile_data
  このデータメンバは，IOR に対するオペレーションを起動するために必要なすべてのプロトコル情報をカプセル化します。

31 マーシャルバッファインタフェースとクラス（C++）

この章では，C++言語でオペレーション要求または応答メッセージを生成するときに，データをバッファにマーシャリングするために使用するバッファクラスについて説明します。また，受信したオペレーション要求または応答メッセージからデータを抽出するために使用するバッファクラスについても説明します。

31.1 CORBA::MarshalInBuffer

```
class CORBA::MarshalInBuffer : public VISistream
```

このクラスは，バッファから IDL 型の読み込みができるようにするストリームバッファを表し，実装するインタセプタメソッドによって使用されます。インタセプタインタフェースの詳細については，「27. ポータブルインタセプタインタフェースとクラス（C++）」を参照してください。

CORBA::MarshalInBuffer クラスは，応答メッセージに対応するデータを抽出するために，クライアント側で使用します。サーバ側では，オペレーション要求に対応するデータを抽出するために使います。このクラスは，バッファからさまざまな型のデータを取り出すメソッドを数多く提供しています。

このクラスは，CORBA::MarshalInBuffer ポインタをテストおよび操作する静的メソッドを提供します。また，包含オブジェクトを自動的に管理するオブジェクトラッパーを提供する，
CORBA::MarshalInBuffer_var クラスも提供します。

クラスの説明で使用する用語について次に記載します。

- 開始オフセットとは，確保されたバッファの先頭アドレスからマーシャルバッファが管理する領域が始まるアドレスまでのバイト数を示す値です。
- シークポジションとは，バッファ内の現在の操作位置です。

31.1.1 インクルードファイル

このクラスを使用するときは，mbuf.h ファイルをインクルードしてください。このファイルは corba.h にインクルード済みです。別途 mbuf.h ファイルをインクルードする必要はありません。

31.1.2 CORBA::MarshalInBuffer のコンストラクタとデストラクタ

- CORBA::MarshalInBuffer(
  char *read_buffer,
  CORBA::ULong length,
  CORBA::Boolean release_flag=0,
  CORBA::ULong start_offset =0,
  CORBA::Boolean byte_order = CORBA::ByteOrder);
  これはデフォルトのコンストラクタです。
  - read_buffer
    マーシャリングされたデータが実際に書き込まれたバッファ
  - length

read_buffer に書き込まれた最大バイト数

- release_flag
  true を設定すると，このオブジェクトがデストラクトされた際に，read_buffer に対応するメモリが解放されます。デフォルト値は false です。
- start_offset
  read_buffer からデータを読み込む際の開始オフセットを設定します。
  デフォルト値は 0 です。
- byte_order
  リトルエンディアンバイトオーダが使用されていることを示す場合は，true を設定し，ビッグエンディアンバイトオーダが使用されていることを示す場合は false を設定します。

- virtual ~CORBA::MarshalInBuffer();
  これは，デフォルトのデストラクタです。release_flag に true を設定した場合，このオブジェクトに対応するバッファメモリが解放されます。オブジェクトを生成するときに release_flag を設定するか，または release_flag メソッドを呼び出すことによって，release_flag を設定できます。release_flag メソッドについては，「31.1.3　CORBA::MarshalInBuffer のメソッド」の「void release_flag(CORBA::Boolean val);」を参照してください。

31.1.3　CORBA::MarshalInBuffer のメソッド

- char *buffer() const;
  このオブジェクトに対応するバッファを指すポインタを返します。
- void byte_order(
  CORBA::Boolean val) const;
  このメッセージバッファのバイトオーダを設定します。
  - val
    リトルエンディアンバイトオーダが使用されていることを示す場合は，true を設定し，ビッグエンディアンバイトオーダが使用されていることを示す場合は false を設定します。
- CORBA::Boolean byte_order() const;
  バッファのバイトオーダにリトルエンディアンバイトオーダを使用している場合は true が返され，ビッグエンディアンバイトオーダを使用している場合は false が返されます。
- CORBA::ULong curoff() const;
  このオブジェクトに対応するバッファ内のシークポジションを返します。
- virtual VISistream& get(char& data);
  virtual VISistream& get(unsigned char& data);
  これらのメソッドを使用して，シークポジションから一つの文字を引数 data に読み込み，一文字分だけシークポジションを進めます。このメソッドは，このオブジェクト自身を返します。

- data
  読み込んだデータが書き込まれる変数

- virtual VISistream& **get**(
  <data_type> **data**, unsigned **size**);
  このメソッドを使用して，シークポジションからデータのシーケンスを引数 data が指す領域に読み込み，次のデータの先頭位置までシークポジションを進めます。次に示すそれぞれのターゲットデータ型に対して別々のメソッドが提供されています。このメソッドは，このオブジェクト自身を返します。
  - data
    読み込んだデータが書き込まれる領域。サポートされるターゲットデータ型を次に示します。
    char*，unsigned char*，short*，unsigned short*，int*，unsigned int*，long*，unsigned long*，float*，double*，long double*，VISLongLong*，VISULongLong*，wchar_t*
  - size
    読み込むデータ型の数

- virtual VISistream& **getCString**(
  char* **data**, unsigned **maxlen**);
  このメソッドを使用して，シークポジションから文字列を引数 data が指す領域に読み込み，次のデータの先頭位置までシークポジションを進めます。
  このメソッドは，このオブジェクト自身を返します。
  - data
    読み込んだ文字列が書き込まれる領域
  - maxlen
    読み込む最大文字数

- virtual const CORBA::WChar *__getWString__(
  CORBA::ULong& **len**);
  このメソッドは，シークポジションから wstring のデータを指すポインタを返し，次のデータの先頭位置までシークポジションを進めます。VisiBroker では，ワイド文字のサイズを 2 バイトとして扱います。
  - len
    読み込んだワイド文字列の文字数が設定されます。設定される値は GIOP のバージョンによって異なります。
    GIOP1.2 の場合，読み込んだワイド文字列のバイト数が設定されます。
    GIOP1.1/1.0 の場合，ワイド文字列の長さ+1 が設定されます。

- virtual int **is_available**(
  unsigned long **size**);
  指定された size が，このオブジェクトに対応するバッファのサイズ以下である場合，1 を返します。それ以外の場合，0 を返します。
  - size

このバッファから読み取る必要のあるバイト数

- virtual CORBA::ULong **length**() const;
  現在のバッファ長を返します。
- virtual void **new_encapsulation**() const;
  バッファ内の開始オフセットを 0 にリセットします。
- void **release_flag**(
  CORBA::Boolean **val**);
  このオブジェクトがデストラクトされた際の，バッファメモリの自動解放を有効または無効にします。
  - val
    val に true を設定すると，このオブジェクトがデストラクトされた際に，このオブジェクトのバッファメモリが解放されます。val に false を設定すると，このオブジェクトがデストラクトされた際にバッファは解放されません。
- CORBA::Boolean **release_flag**() const;
  オブジェクトのバッファメモリの自動解放が有効である場合，true を返します。そうでない場合は，false を返します。
- void **reset**();
  開始オフセット，およびシークポジションを 0 にリセットします。
- void **rewind**();
  開始オフセットはコンストラクタで設定した値にリセットされ，シークポジションは 0 にリセットされます。
- CORBA::ULong **seekpos**(
  CORBA::ULong **pos**);
  pos の値で，シークポジションを任意に設定できます。pos で指定された値がバッファサイズより大きい場合，CORBA::BAD_PARAM 例外が発生します。
- static CORBA::MarshalInBuffer *_**duplicate**(
  CORBA::MarshalInBuffer_ptr **ptr**);
  ptr が指すこのオブジェクトの複製ポインタを返し，このオブジェクトのリファレンスカウントを増やします。
- static CORBA::MarshalInBuffer *_**nil**();
  CORBA::MarshalInBuffer 型の NULL ポインタを返します。
- static void _**release**(
  CORBA::MarshalInBuffer_ptr **ptr**);
  ptr が指すオブジェクトのリファレンスカウントを減らします。リファレンスカウントが 0 になると，そのオブジェクトはデストラクトされます。構築された際にオブジェクトの release_flag に true を設定した場合，オブジェクトに対応するバッファが解放されます。

31.1.4 CORBA::MarshalInBuffer の演算子

- virtual VISistream& operator>>(<data_type> data);

  このストリーム演算子を使用して，引数に指定された data_type 型のデータをシークポジションから引数 data に読み込み，次のデータの先頭位置までシークポジションを進めます。

  このメソッドは，このオブジェクト自身を返します。

  - data

    読み込んだデータが書き込まれる領域。サポートされるソースデータ型は次のとおりです。

    char*&，char&，unsigned char&，short&，unsigned short&，int&，unsigned int&，long&，unsigned long&，float&，double&，long double&，wchar_t*&，wchar_t&

31.2 CORBA::MarshalOutBuffer

```
class CORBA::MarshalOutBuffer : public VISostream
```

このクラスは，バッファからの IDL 型を書き込めるようにするストリームバッファを表し，実装するインタセプタメソッドによって使用されます。インタセプタインタフェースの詳細については，「27. ポータブルインタセプタインタフェースとクラス（C++）」を参照してください。

このクラスは，オペレーション要求に対応するデータをマーシャリングするために，クライアント側で使用します。サーバ側では，応答メッセージに対応するデータをマーシャリングするために使用します。このクラスは，バッファへさまざまな型のデータを追加したり，バッファに書き込まれたデータを取り出したりするメソッドを数多く提供しています。

このクラスは，CORBA::MarshalOutBuffer ポインタをテストおよび操作する静的メソッドを提供します。また，包含オブジェクトを自動的に管理するオブジェクトラッパーを提供する，CORBA::MarshalOutBuffer_var クラスも提供します。

クラスの説明で使用する用語について次に記載します。

- 開始オフセットとは，確保されたバッファの先頭アドレスからマーシャルバッファが管理する領域が始まるアドレスまでのバイト数を示す値です。
- シークポジションとは，バッファ内の現在の操作位置です。

31.2.1 インクルードファイル

このクラスを使用するときは，mbuf.h ファイルをインクルードしてください。このファイルは corba.h にインクルード済みです。別途 mbuf.h ファイルをインクルードする必要はありません。

31.2.2 CORBA::MarshalOutBuffer のコンストラクタとデストラクタ

- CORBA::MarshalOutBuffer(
  CORBA::ULong initial_size = 255,
  CORBA::Boolean release_flag = 0,
  CORBA::ULong start_offset=0);
  initial_size に指定されたサイズの MarshalOutBuffer を生成します。MarshalOutBuffer には，put オペレーション時に自分自身のサイズを変更する機能があります。書き込まれたすべての内容を保持するスペースがバッファに不足している場合，バッファのサイズが倍になります。
  - initial_size
    このオブジェクトに対応するバッファの初期サイズ。デフォルトサイズは，255 バイトです。

- release_flag
  true を設定すると，このオブジェクトがデストラクトされた場合にバッファに対応するメモリが解放されます。デフォルト値は false です。
- start_offset
  バッファにデータを書き込む際の開始オフセットを設定します。
  デフォルト値は 0 です。

- CORBA::MarshalOutBuffer(
  char *buffer,
  CORBA::ULong len,
  CORBA::Boolean release_flag=0,
  CORBA::ULong start_offset=0);
  指定されたバッファ，バッファ長，解放フラグ値で，オブジェクトを生成します。
  - buffer
    マーシャリングされたデータが実際に書き込まれるバッファ
  - len
    buffer に書き込むことができる最大バイト数
  - release_flag
    true を設定すると，このオブジェクトがデストラクトされた場合に buffer に対応するメモリが解放されます。デフォルト値は false です。
  - start_offset
    buffer からデータを追加する際の開始オフセットを設定します。
    デフォルト値は 0 です。
- virtual ~CORBA::MarshalOutBuffer();
  これはデフォルトのデストラクタです。release_flag に true を設定した場合，このオブジェクトに対応するバッファメモリが解放されます。オブジェクトを生成するときに release_flag を設定するか，または release_flag メソッドを呼び出すことによって release_flag を設定できます。release_flag メソッドについては，「31.2.3　CORBA::MarshalOutBuffer のメソッド」の「void release_flag(CORBA::Boolean val);」を参照してください。

31.2.3　CORBA::MarshalOutBuffer のメソッド

- char *buffer() const;
  このオブジェクトに対応するバッファを指すポインタを返します。
- CORBA::ULong curoff() const;
  このオブジェクトに対応するバッファ内のシークポジションを返します。
- virtual CORBA::ULong length() const;

現在のバッファ長を返します。

- virtual void **new_encapsulation**() const;
  バッファ内の開始オフセットを 0 にリセットします。
- virtual VISostream& **put**(
  char **data**);
  シークポジションに一つの文字を書き込み，一文字分だけシークポジションを進めます。
  このメソッドは，このオブジェクト自身を返します。
  - data
    シークポジションに書き込むデータ
- virtual VISostream& **put**(
  const <data_type> **data**, unsigned **size**);
  このメソッドを使用して，シークポジションにデータのシーケンスを書き込み，次のデータの先頭位置までシークポジションを進めます。
  このメソッドは，このオブジェクト自身を返します。
  - data
    シークポジションに書き込むデータ。サポートされるソースデータ型を次に示します。
    char*，unsigned char*，short*，unsigned short*，int*，unsigned int*，long*，unsigned long*，float*，double*，long double*，VISLongLong*，VISULongLong*，wchar_t*
  - size
    書き込むデータ型の数
- virtual VISostream& **putCString**(
  const char* **data**);
  このメソッドを使って，シークポジションに文字列を書き込み，書き込んだサイズ分だけシークポジションを進めます。
  このメソッドは，このオブジェクト自身を返します。
  - data
    書き込む文字列
- void **release_flag**(
  CORBA::Boolean **val**);
  このオブジェクトがデストラクトされた際の，バッファメモリの自動解放を有効または無効にします。
  - val
    val に true を設定すると，このオブジェクトがデストラクトされた際に，このオブジェクトのバッファメモリが解放されます。val に false を設定すると，このオブジェクトがデストラクトされた際に，バッファは解放されません。
- CORBA::Boolean **release_flag**() const;

このオブジェクトがデストラクトされたとき，バッファメモリが自動的に解放される場合は，true を返します。そうでない場合は，false を返します。

- void reset();
  開始オフセット，およびシークポジションを 0 にリセットします。
- void rewind();
  開始オフセットはコンストラクタで設定した値にリセットされ，シークポジションは 0 にリセットします。
- CORBA::ULong seekpos(
  CORBA::ULong pos);
  pos の値で，シークポジションを任意に設定できます。pos で指定された値がバッファサイズより大きい場合，CORBA::BAD_PARAM 例外が発生します。
- static CORBA::MarshalOutBuffer *_duplicate(
  CORBA::MarshalOutBuffer_ptr ptr);
  ptr が指すこのオブジェクトの複製ポインタを返し，このオブジェクトのリファレンスカウントを増やします。
- static CORBA::MarshalOutBuffer *_nil();
  CORBA::MarshalOutBuffer 型の NULL ポインタを返します。
- static void _release(
  CORBA::MarshalOutBuffer_ptr ptr);
  ptr が指すオブジェクトのリファレンスカウントを減らします。リファレンスカウントが 0 になると，そのオブジェクトはデストラクトされます。構築された際にオブジェクトの release_flag に true を設定した場合，オブジェクトに対応するバッファが解放されます。

31.2.4 CORBA::MarshalOutBuffer の演算子

- virtual VISostream& operator<<(<data_type> data);
  このストリーム演算子を使用して，引数に指定された data_type 型の data をシークポジションに書き込み，次のデータの先頭位置までシークポジションを進めます。
  このメソッドは，このオブジェクト自身を返します。
  - data
    バッファに書き込まれるデータ。サポートされるデータ型は次のとおりです。
    const char*，char，unsigned char，short，unsigned short，int，unsigned int，long，unsigned long，float，double，long double，VISLongLong，VISULongLong，const wchar_t *，wchar_t

32 ロケーションサービスインタフェースとクラス（C++）

この章では，スマートエージェントが管理するネットワーク上にあるオブジェクトインスタンスの検索に使用できる，C++言語のインタフェースについて説明します。ロケーションサービスの詳細については，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「ロケーションサービスの使用」の記述を参照してください。

32.1 Agent

```
class Agent : public CORBA::Object
```

このクラスで提供されるメソッドを使って，特定のオブジェクトのすべてのインスタンスを，スマートエージェントのネットワーク上で検索できます。このクラスで提供されるメソッドは，次に示す二つのカテゴリに分けられます。このオブジェクトを取得するためには，ORB::resolve_initial_references("LocationService")で求めたオブジェクトを ObjLocation::Agent でナロウしてください。

- オブジェクトのデータについてスマートエージェントに問い合わせるメソッド
- トリガーを扱うメソッド

クライアントアプリケーションは，インタフェースリポジトリ ID だけ，またはインタフェースリポジトリ ID とインスタンス名の両方に基づいてオブジェクト情報を得ることができます。

トリガーを使って，一つ以上のオブジェクトインスタンスが使用できるかどうかなどの，状況の変化をクライアントアプリケーションに通知できます。

ロケーションサービスを使用するアプリケーションのコマンドラインオプションについては，「34.2 ロケーションサービスオプション」を参照してください。

32.1.1 IDL の定義

```
module ObjLocation {
  interface Agent {
    HostnameSeq all_agent_locations()
         raises (Fail);
    RepositoryIdSeq all_repository_ids()
         raises (Fail);
    ObjSeqSeq all_available()
         raises (Fail);
    ObjSeq all_instances (in string repository_id)
         raises (Fail);
    ObjSeq all_replica (in string repository_id, in string instance_name)
         raises (Fail);
    DescSeqSeq all_available_descs()
         raises (Fail);
    DescSeq all_instances_descs (in string repository_id)
         raises (Fail);
    DescSeq all_replica_descs (in string repository_id,
         in string instance_name)
         raises (Fail);
      void reg_trigger(in TriggerDesc desc,
         in TriggerHandler handler)
         raises (Fail);
      void unreg_trigger(in TriggerDesc desc,
         in TriggerHandler handler)
```

```
        raises (Fail);
    attribute boolean willRefreshOADs;
};
};
```

32.1.2　インクルードファイル

このクラスを使用するときは，locate_c.hh ファイルをインクルードしてください。

32.1.3　Agent のメソッド

- ObjLocation::HostnameSeq_ptr **all_agent_locations**();

  現在 osagent プロセスが実行されているホストを表すホスト名のシーケンスを返します。「32.6 <type>Seq」も参照してください。

  このメソッドでは，次の例外が発生します。

  Fail

  FailReason 値は次のとおりです。

  ・NO_AGENT_AVAILABLE

  ・NO_SUCH_TRIGGER

  ・AGENT_ERROR

  Fail クラスの詳細については，「32.3　Fail」を参照してください。

- ObjLocation::ObjSeq_ptr **all_available**();

  ネットワーク上のスマートエージェントに現在登録されているすべてのオブジェクトの，オブジェクトリファレンスのシーケンスを返します。「32.6　<type>Seq」も参照してください。

  このメソッドでは，次の例外が発生します。

  Fail

  FailReason 値は次のとおりです。

  ・NO_AGENT_AVAILABLE

  ・NO_SUCH_TRIGGER

  ・AGENT_ERROR

  Fail クラスの詳細については，「32.3　Fail」を参照してください。

- ObjLocation::DescSeqSeq_ptr **all_available_descs**();

  ネットワーク上のスマートエージェントに現在登録されているすべてのオブジェクトの記述を返します。返される記述情報はリポジトリ ID によって編成されます。「32.7　<type>SeqSeq」も参照してください。

  このメソッドでは，次の例外が発生します。

Fail

FailReason 値は次のとおりです。

・NO_AGENT_AVAILABLE

・NO_SUCH_TRIGGER

・AGENT_ERROR

Fail クラスの詳細については，「32.3　Fail」を参照してください。

- ObjLocation::ObjSeq_ptr **all_instances**(
  const char ***repository_id**);
  指定された repository_id を持つすべてのインスタンスのオブジェクトリファレンスのシーケンスを返します。「32.6　<type>Seq」も参照してください。
  引数の意味を次に示します。
  - repository_id
    取得するオブジェクトリファレンスのリポジトリ ID

  このメソッドでは，次の例外が発生します。

  Fail

  NO_SUCH_TRIGGER 以外の FailReason 値。Fail クラスの詳細については，「32.3　Fail」を参照してください。

- ObjLocation::DescSeq_ptr **all_instances_descs**(
  const char ***repository_id**);
  指定された repository_id を持つすべてのオブジェクトインスタンスの記述情報を返します。「32.6　<type>Seq」も参照してください。
  引数の意味を次に示します。
  - repository_id
    取得するオブジェクト記述のリポジトリ ID

  このメソッドでは，次の例外が発生します。

  Fail

  NO_SUCH_TRIGGER 以外の FailReason 値。Fail クラスの詳細については，「32.3　Fail」を参照してください。

- ObjLocation::ObjSeq_ptr **all_replica**(
  const char ***repository_id**,
  const char ***instance_name**);
  指定された repository_id と instance_name を持つオブジェクトのオブジェクトリファレンスのシーケンスを返します。「32.6　<type>Seq」も参照してください。
  引数の意味を次に示します。
  - repository_id
    検索するオブジェクトリファレンスのリポジトリ ID

- instance_name
  返されるオブジェクトリファレンスのインスタンス名

このメソッドでは，次の例外が発生します。

Fail

NO_SUCH_TRIGGER 以外の FailReason 値。Fail クラスの詳細については，「32.3　Fail」を参照してください。

- ObjLocation::DescSeq_ptr **all_replica_descs**(
  const char ***repository_id**,
  const char ***instance_name**);
  指定された repository_id と instance_name を持つすべてのオブジェクトインスタンスの記述情報のシーケンスを返します。「32.6　<type>Seq」も参照してください。
  引数の意味を次に示します。
  - repository_id
    取得するオブジェクト記述のリポジトリ ID
  - instance_name
    取得するオブジェクト記述のインスタンス名

  このメソッドでは，次の例外が発生します。

  Fail

  NO_SUCH_TRIGGER 以外の FailReason 値。Fail クラスの詳細については，「32.3　Fail」を参照してください。

- CORBA::StringSequence* **all_repository_ids**();
  このメソッドは，任意の osagent によって認識されているインタフェースをすべて取得します。
  このメソッドでは，次の例外が発生します。

  Fail

  リポジトリ ID が不正です。

- void **reg_trigger**(
  const ObjLocation::TriggerDesc& **desc**,
  ObjLocation::TriggerHandler_ptr **hdlr**);
  desc で指定されている記述情報と一致するオブジェクトインスタンスのトリガーハンドラ hdlr を登録します。

  注

  トリガーの記述に合うオブジェクトが使用できるようになるたびに，TriggerHandler が呼び出されます。オブジェクトの最初のインスタンスを使用できるようにするには，最初の通知を受信したあとに，unreg_trigger メソッドを使ってトリガーを削除してください。

  引数の意味を次に示します。

- desc
  次の情報を組み合わせたオブジェクトインスタンス記述情報
  ・リポジトリ ID
  ・インスタンス名
  ・ホスト名
  情報を提供して，監視するオブジェクトインスタンスのナロウイングおよびワイドニングができます。
- hdlr
  登録されるトリガーハンドラオブジェクト

このメソッドでは，次の例外が発生します。

Fail
NO_SUCH_TRIGGER 以外の FailReason 値。Fail クラスの詳細については，「32.3　Fail」を参照してください。

- void unreg_trigger(
  const ObjLocation::TriggerDesc& desc,
  ObjLocation::TriggerHandler_ptr hdlr);
  desc で指定されている記述情報と一致する，オブジェクトインスタンスのトリガーハンドラ hdlr を，登録解除します。
  引数の意味を次に示します。
  - desc
    オブジェクト記述情報
  - hdlr
    登録解除されるトリガーハンドラオブジェクト

  このメソッドでは，次の例外が発生します。

  Fail
  NO_SUCH_TRIGGER 以外の FailReason 値。Fail クラスの詳細については，「32.3　Fail」を参照してください。
- CORBA::Boolean willRefreshOADs();
  このクラスが提供するメソッドが呼び出されるたびに，OAD（オブジェクト活性化デーモン）群が更新される場合は，TRUE を返します。そうでない場合は，FALSE を返します。それぞれの呼び出しでキャッシュがリフレッシュされないと，次のようになります。
  - すべてのオブジェクトが報告されていますが，それらの記述子の activable フラグが不当な場合があります。
  - 前回の OAD キャッシュのリフレッシュ後に起動された，OAD に登録されているオブジェクトの存在を検証しようとすると，それらのオブジェクトが OAD によって活性化されてしまいます。
- void willRefreshOADs(
  CORBA::Boolean val);

このクラスは，OAD 群を保持します。このクラスによって提供されるメソッドは，その OAD 群に含まれる OAD の自動リフレッシュを有効または無効にします。

引数の意味を次に示します。

- val

  TRUE の場合，このクラスで提供されているメソッドが呼び出されるたびに，OAD 群がリフレッシュされます。

32.2 Desc

```
struct Desc
```

この構造体は，オブジェクトの特性を記述するために使用する情報を格納します。この章で説明する，ロケーションサービスメソッドの幾つかに，この構造体を引数として渡します。幾つかのロケーションサービスメソッドによって Desc 構造体，またはそのシーケンスが返されます。

「32.6 <type>Seq」も参照してください。

32.2.1 IDL の定義

```
module ObjLocation {
  struct Desc {
    Object ref;
    IIOP::ProfileBody iiop_locator;
    string repository_id;
    string instance_name;
    boolean activable;
    string agent_hostname;
  };
. . .
};
```

32.2.2 Desc のメンバ

- CORBA::Object_var **ref**
  記述されるオブジェクトのリファレンスです。
- IIOP::ProfileBody **iiop_locator**
  オブジェクトのプロファイルデータを表します。詳細については，「30.7 IIOP::ProfileBody」を参照してください。
- CORBA::String_var **repository_id**
  オブジェクトのリポジトリ ID です。
- CORBA::String_var **instance_name**
  オブジェクトのインスタンス名です。
- CORBA::Boolean **activable**
  このオブジェクトが，オブジェクト活性化デーモンに登録されていることを示す場合は TRUE を設定します。オブジェクトが手動で起動され，osagent に登録されていることを示す場合は FALSE を設定します。
- CORBA::String_var **agent_hostname**

このオブジェクトが登録されているスマートエージェントを実行しているホストの名前です。

32.3 Fail

```
class Fail : public CORBA::UserException
```

さまざまなエラーを示すために，Agent クラスはこの例外クラスを発生させます。FailReason というデータメンバは，障害の性質を示します。

32.3.1 Fail のメンバ

- FailReason reason

  このメンバは，障害の性質を示す次の値のどれかを設定します。

```
enum FailReason {
    NO_AGENT_AVAILABLE,
    INVALID_REPOSITORY_ID,
    INVALID_OBJECT_NAME,
    NO_SUCH_TRIGGER,
    AGENT_ERROR
};
```

32.4 TriggerDesc

```
struct TriggerDesc
```

この構造体は，TriggerHandler を登録する一つ以上のオブジェクトの特性を記述するために使用する情報を格納します。TriggerHandler については，「32.5 TriggerHandler」を参照してください。広範囲のオブジェクトを監視するには，host_name メンバ，および instance_name メンバに NULL を設定します。指定する情報が多いほど，対象オブジェクトを限定できます。

32.4.1 IDL の定義

```
module ObjLocation {
    . . .
         struct TriggerDesc {
            string repository_id;
            string instance_name;
            string host_name;
};
 . . .
};
```

32.4.2 TriggerDesc のメンバ

- CORBA::String_var repository_id

  TriggerHandler によって監視されるオブジェクトのリポジトリ ID です。すべてのリポジトリ ID を含めるには，NULL を設定します。

- CORBA::String_var instance_name

  このメンバは，TriggerHandler によって監視されるオブジェクトのインスタンス名を表します。すべてのインスタンス名を含めるには，NULL を設定します。

- CORBA::String_var host_name;

  このメンバは，TriggerHandler によって監視されるオブジェクトのホスト名を表します。ネットワーク上のすべてのホストを含めるには，NULL を設定します。

32.5 TriggerHandler

```
class TriggerHandler : public virtual CORBA::Object
```

オブジェクトが使用できるたびに，または使用できなくなるたびに呼び出す，独自のコールバックオブジェクトを派生させるために，このベースクラスを使用します。対象とするオブジェクトの基準を指定します。Agent クラスの reg_trigger メソッドを使用して，TriggerHandler オブジェクトを登録します。reg_trigger メソッドについては，「32.1.3 Agent のメソッド」の「void reg_trigger(const ObjLocation::TriggerDesc& desc, ObjLocation::TriggerHandler_ptr hdlr);」を参照してください。

impl_is_ready メソッドと impl_is_down メソッドのインプリメンテーションを提供する必要があります。

32.5.1 IDL の定義

```
interface TriggerHandler {
    void impl_is_ready(in Desc desc);
    void impl_is_down(in Desc desc);
};
```

32.5.2 インクルードファイル

このクラスを使用するときは，locate_c.hh ファイルをインクルードしてください。

32.5.3 TriggerHandler のメソッド

- virtual void **impl_is_ready**(
  const Desc& **desc**);
  このメソッドは，desc で指定された基準に一致するオブジェクトインスタンスがアクセスできるようになった場合に，ロケーションサービスによって呼び出されます。
  - desc
    オブジェクト記述情報
- virtual void **impl_is_down**(
  const Desc& **desc**);
  このメソッドは，desc で指定された基準に一致するオブジェクトインスタンスがアクセスできなくなった場合に，ロケーションサービスによって呼び出されます。
  - desc
    オブジェクト記述情報

32.6 <type>Seq

これは，ロケーションサービスによって使用される，次に示すシーケンスクラスについて記述した，汎用クラスです。

- DescSeq
  Desc 構造体のシーケンス
- HostnameSeq
  ホスト名のシーケンス
- ObjSeq
  オブジェクトリファレンスのシーケンス
- RepositoryIdSeq
  リポジトリ ID のシーケンス

各クラスは，<type>のある特定のシーケンスを表しています。ロケーションサービスは，これらのクラスのどれかにマッピングされたシーケンスの形で，クライアントアプリケーションに情報のリストを返します。

C++配列の場合と同様に，各クラスはシーケンス内の項目をインデックス付けする演算子を提供しています。その配列の長さを取得するメソッド，および配列の長さを設定するメソッドも提供しています。

コードサンプル 32-1 に，Agent::all_agent_locations メソッドから返された HostnameSeq をインデックス付けする正しい方法を示します。

コードサンプル 32-1　HostnameSeq_var クラスのインデックス付け

```
. . .
ObjLocation::HostnameSeq_var hostnames(
  myAgent->all_agent_locations());

for (CORBA::ULong i=0; i < hostnames->length(); i++) {
  cout << "Agent host #" << i+1 << ": " << hostnames[i] << endl;
}
. . .
```

「32.7　<type>SeqSeq」も参照してください。

32.6.1 <type>Seq のメソッド

- <type>& operator[ ](
  CORBA::ULong index) const;
  このメソッドは，index で識別されているシーケンス内の要素のリファレンスを返します。

注

インデックスに対しては CORBA::ULong 型を使用しなければなりません。int 型を使用すると予想できない結果が起こります。

引数の意味を次に示します。

- index

  要素が返されるメンバのインデックス。インデックスは 0 から始まります。

このメソッドでは，次の例外が発生します。

CORBA::BAD_PARAM

指定したインデックスが，0 より小さいか，またはシーケンスのサイズを超えています。

- CORBA::ULong **length**() const;
  シーケンス内の要素数を返します。
- void **length**(
  CORBA::ULong **len**);
  シーケンスの最大長を len に含まれる値に設定します。
  引数の意味を次に示します。
  - len

    シーケンスの新しい長さ

32.7 <type>SeqSeq

これは，ロケーションサービスによって使用される，次に示すクラスについて記述した，汎用クラスです。

- DescSeqSeq
  DescSeq オブジェクトのシーケンス
- ObjSeqSeq
  ObjSeq オブジェクトのシーケンス

各クラスは，<type>Seq のある特定のシーケンスを表しています。幾つかのロケーションサービスメソッドは，それらのクラスのどれかにマッピングされたシーケンスの形で，クライアントアプリケーションに情報のリストを返します。

C++配列の場合と同様に，各クラスはシーケンス内の項目をインデックス付けするための演算子を提供しています。その配列の長さを取得するメソッドおよび配列の長さの設定をするメソッドも提供しています。

「32.6 <type>Seq」も参照してください。

32.7.1 <type>SeqSeq のメソッド

- <type>Seq& operator[ ](
  CORBA::ULong index) const;
  このメソッドは，index で識別されているシーケンス内の要素のリファレンスを返します。このリファレンスは，一次元シーケンスのリファレンスです。一次元シーケンスについては，「32.6 <type>Seq」を参照してください。

  注

  インデックスに対しては CORBA::ULong 型を使用しなければなりません。int 型を使用すると予想できない結果が起こります。

  引数の意味を次に示します。

  - index
    要素が返されるメンバのインデックス。インデックスは 0 から始まります。

  このメソッドでは，次の例外が発生します。

  CORBA::BAD_PARAM

  指定したインデックスが，0 より小さいか，またはシーケンスのサイズを超えています。
- CORBA::ULong length() const;
  シーケンス内の要素数を返します。
- void length(
  CORBA::ULong len);

シーケンスの最大長を len に含まれる値に設定します。
引数の意味を次に示します。

- len
  シーケンスの新しい長さ

33 初期化インタフェースとクラス（C++）

この章では，インタセプタなどの VisiBroker ORB サービスを，静的に初期化するために提供されている，C++言語のインタフェースとクラスについて説明します。

33.1 VISInit

```
class VISInit
```

この抽象ベースクラスは，VisiBroker ORB の初期化後，サービスクラスを静的に初期化します。VISInit からサービスクラスを派生させ，そのクラスを静的に宣言することで，そのサービスクラスインスタンスを正しく初期化できます。

アプリケーションが CORBA::ORB_init メソッドを呼び出した際，VisiBroker ORB は VISInit::ORB_init および VISInit::ORB_initialized を呼び出します。これらのメソッドのインプリメンテーションを提供することによって，ユーザ独自のサービスに必要な初期化を追加できます。

33.1.1 インクルードファイル

このクラスを使用するときは，vinit.h ファイルをインクルードしてください。

33.1.2 VISInit のメソッド

- VISInit();
  これはデフォルトのコンストラクタです。
- VISInit(
  CORBA::Long init_priority);
  このコンストラクタは，指定した優先度で VISInit から派生したオブジェクトを生成します。優先度の指定によって，このオブジェクトをほかの VISInit 派生オブジェクトより先に初期化したり，あとに初期化したりできます。
  ユーザ定義クラスより先に初期化する必要のある Borland Enterprise Server VisiBroker 内部クラスの優先度には，負の値が使用されています。現在，Borland Enterprise Server VisiBroker 内部クラスの優先度に使用されている値のうちで最も小さい値は-10 です。

  注

  ユーザ定義クラスを Borland Enterprise Server VisiBroker 内部クラスより先に初期化したい場合は，優先度に-10 より小さい値を設定する必要があります。ただし，その時点では VisiBroker は初期化されていないため，VisiBroker の機能は使用できません。

  デフォルトコンストラクタを使用した場合の優先度は 0 です。

  - init_priority
    このオブジェクトの初期化優先度。負の値を指定すると優先度が高くなり，正の値を指定すると優先度が低くなります。
- virtual void ORB_init(

int& **argc**, char * const ***argv**, CORBA::ORB_ptr **orb**);

このメソッドは，VisiBroker ORB が初期化された際に呼び出されます。使用するクライアント側インタセプタファクトリの初期化に応じて，実装する必要があります。

- argc
  引数のカウント
- argv
  引数ポインタの配列
- orb
  初期化される VisiBroker ORB

- virtual void **ORB_initialized**(
  CORBA::ORB_ptr **orb**);

  このメソッドは，VisiBroker ORB が初期化されたあとに呼び出されます。使用するクライアント側インタセプタファクトリの初期化に応じて，実装する必要があります。
  - orb
    初期化された VisiBroker ORB
- virtual void **ORB_shutdown**()

  このメソッドは，VisiBroker ORB がシャットダウンされる際に呼び出されます。

34 コマンドラインオプション（C++）

この章では，C++言語でプログラミングするときに，オブジェクトリクエストブローカー（ORB）およびロケーションサービスで設定するオプションについて説明します。

34.1　ORB_init()メソッド

クライアントプログラムは，使用するスマートエージェントの IP アドレスや，ポート番号などのオプションを設定する際に，ORB_init()メソッドを使用します。これらのパラメタは，クライアントプログラムのプロセスが起動された際に，引数としてそのプロセスに渡されます。

```
prompt> client -ORBagentAddr 199.99.129.33 -ORBagentPort 19000
```

ORB_init()メソッドの定義およびこのメソッドが受け入れる引数について，コードサンプル 34-1 に示します。ORB_init()に渡された argc と argv パラメタは，クライアントプログラムのメインルーチンに渡された引数と同じものです。ORB_init()メソッドが認識できない引数は無視されます。

コードサンプル 34-1　ORB_init()メソッド定義

```
class CORBA {
    ...
    static ORB_ptr ORB_init(int& argc, char *const *argv,
                            const char *orb_id = (char *)NULL);
    ...
};
```

このメソッドを呼び出すと，認識された VisiBroker ORB 引数はすべて元のパラメタリストから削除されます。これは，クライアントプログラムが要求するほかの引数処理を妨げないためです。

34.1.1　ORB オプション

一つのオプションを除いて，その他すべての ORB オプションは型・値ペアの形式になっています。ORB_init()オプションについて，次の表に示します。

表 34-1　クライアントプログラムが使用する ORB_init オプション（C++）

オプション	説明
-ORBagent <0|1>	VisiBroker の_bind()時に，スマートエージェントがサーバを探索するかどうかを指定します。 0 を設定した場合，探索しません。 1 を設定した場合，探索します。 デフォルトは 1 です。
-ORBagentAddr <hostname|ip_address>	このクライアントが使用するスマートエージェントを実行しているホストのホスト名または IP アドレスを指定します。 hostname を設定した場合，ホスト名を指定します。 ip_address を設定した場合，IP アドレスを指定します。 このオプションの指定を省略した場合や，自ホストまたは agentaddr ファイルで指定したホストからスマートエージェントを探索できない場合は，ブロードキャストメッセージでスマートエージェントを見つけます。

オプション	説明
-ORBagentPort <port_number>	スマートエージェントのポート番号を指定します。複数の VisiBroker ORB ドメインが必要な場合に有用です。 指定を省略した場合，14000 番で動作します。
-ORBconnectionMax <#>	コネクションの最大数を指定します。 指定を省略した場合，無制限にコネクションを許可します。
-ORBconnectionMaxIdle <#>	コネクションが非アクティブな状態の最大監視時間を秒単位で指定します。最大監視時間を経過してもコネクションが非アクティブなままの場合，VisiBroker がコネクションを終了します。 このオプションはインターネットアプリケーションで設定します。 0 を設定した場合，監視しません。 デフォルトは 0 です。
-ORBDefaultInitRef	デフォルトの初期リファレンスを指定します。
-ORBInitRef	初期リファレンスを指定します。
-ORBir_ior <ior_string>	Object::_get_interface()メソッドがオブジェクトインプリメンテーションで呼び出されるとき，アクセスされるインタフェースリポジトリの IOR を指定します。
-ORBir_name <ir_name>	Object::_get_interface()メソッドがオブジェクトインプリメンテーションで呼び出されるとき，アクセスされるインタフェースリポジトリの名前を指定します。
-ORBnullstring <0|1>	1 を設定した場合，VisiBroker ORB が C++ NULL 文字をストリームします。NULL 文字列は長さが 0 の文字列としてマーシャルされます。これは，長さが 1 の文字列としてマーシャルされる空の文字列（""）とは異なり，単独の文字（"¥0"）です。 0 を設定した場合，NULL 文字列をマーシャルしたときは，CORBA::BAD_PARAM となります。NULL 文字列をアンマーシャルしたときは，CORBA::MARSHAL となります。 デフォルトは 0 です。
-ORBrcvbufsize <buffer_size>	応答を受信するために使用する TCP バッファのサイズをバイト単位で指定します。指定値は，性能やベンチマークの結果に著しく影響します。 指定を省略した場合，デフォルト値が設定されます。 デフォルトは OS よって異なるため，各 OS のマニュアルを参照してください。
-ORBsendbufsize <buffer_size>	クライアント要求を送信するために使用する TCP バッファのサイズをバイト単位で指定します。指定値は，性能やベンチマークの結果に著しく影響します。 指定を省略した場合，デフォルト値が設定されます。 デフォルトは OS よって異なるため，各 OS のマニュアルを参照してください。
(Windows) -ORBshmsize <size>	共用メモリ内の送信セグメントおよび受信セグメントのサイズをバイト単位でを指定します。クライアントプログラムやオブジェクトインプリメンテーションが共用メモリを介して通信する場合，この

オプション	説明
	オプションを使うと，パフォーマンスが向上します。ただし，このオプションをサポートしているのは Windows だけです。
-ORBtcpNoDelay <0|1>	ソケットが要求を送信する契機を指定します。指定値は，性能やベンチマークの結果に著しく影響します。 1 を設定した場合，すべてのソケットが即時に要求を送信します。 0 を設定した場合，ソケットはバッファが満杯になった時点で要求を一括して送信します。 デフォルトは 0 です。

34.2　ロケーションサービスオプション

VisiBroker 4.x では，コマンドラインオプションの LOCdebug，LOCtimeout，および LOCverify をサポートしていません。これらのオプションの代わりに使用できるプロパティとデフォルトの詳細については，「35.4　ロケーションサービスプロパティ」を参照してください。

35 Borland Enterprise Server VisiBroker プロパティ (C++)

この章では，C++言語でプログラミングする場合に，Borland Enterprise Server VisiBroker で設定できるプロパティについて説明します。

これらのパラメタは，プロセスが起動された際に引数としてプロセスに渡せます。

```
prompt> client -Dvbroker.agent.port=14000
```

ORB_init()メソッドの定義およびこのメソッドが受け入れる引数について，コードサンプル 35-1 に示します。ORB_init()に渡された argc と argv パラメタは，クライアントプログラムのメインルーチンに渡された引数と同じものです。

ORB_init()メソッドが認識できない引数は無視されます。

コードサンプル 35-1　ORB_init()メソッド定義

```
class CORBA {
    ...
    static ORB_ptr ORB_init(int& argc, char *const *argv,
                            const char *orb_id = (char *)NULL);
    ...
};
```

このメソッドを呼び出すと，認識された VisiBroker ORB 引数はすべて元のパラメタリストから削除されます。これは，クライアントプログラムが要求するほかの引数処理を妨げないためです。

35.2 osagent（スマートエージェント）プロパティ

osagent（スマートエージェント）のプロパティを次の表に示します。

表 35–1 osagent（スマートエージェント）プロパティ（C++）

プロパティ	デフォルト	旧プロパティ	説明
vbroker.agent.addr	null	-ORBagentAddr	osagent が動作しているホストの IP アドレスまたはホスト名を，VisiBroker アプリケーションに指定します。デフォルトの null に設定すると，VisiBroker アプリケーションは OSAGENT_ADDR 環境変数の値を使用します。 OSAGENT_ADDR 環境変数が未設定のときは，osagent がローカルホストで動作していると仮定します。 このプロパティは osagent には指定できません。
vbroker.agent.addrFile	null	該当しません。	agentaddr ファイルを指定します。agentaddr ファイルについては，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「agentaddr ファイルによるホストの指定」または「ポイントツーポイント通信の使用」を参照してください。 このプロパティは osagent とアプリケーションのどちらでも指定する事ができます。
vbroker.agent.clientHandlerPort	0	該当しません。	osagent がクライアントアプリケーションと通信するために使用するポートを指定します。デフォルトの 0 に設定すると，osagent はシステムに割り当てられたポート番号を使用します。 このプロパティはアプリケーションには指定できません。 注 0 以外を指定する場合，有効範囲は 5001～65535 の範囲の整数値です。それ以外の値を指定した場合，動作の保証はできません。
vbroker.agent.debug	false	該当しません。	true を設定すると，システムは VisiBroker アプリケーションと osagent との間での通信のデバッグ情報を表示します。 このプロパティは osagent には指定できません。
vbroker.agent.enableLocator	true	-ORBagent	false を設定すると，VisiBroker アプリケーションは osagent と通信を行いません。

プロパティ	デフォルト	旧プロパティ	説明
			このプロパティは osagent には指定できません。
vbroker.agent.localFile	null	該当しません。	マルチホームマシン上で使用するネットワークインタフェースを指定します。これは，以前は OSAGENT_LOCAL_FILE 環境変数でした。 このプロパティは osagent には指定できません。
vbroker.agent.port	14000	-ORBagentPort	ネットワーク上の ORB ドメインを定義するポート番号を指定します。すべての VisiBroker アプリケーションと osagent に同一のポート番号を設定すると，それらを連携させることができます。このプロパティは OSAGENT_PORT 環境変数と同じ働きをします。 このプロパティは osagent とアプリケーションのどちらでも指定する事ができます。

35.3 ORB プロパティ

C++言語に対応する Borland Enterprise Server VisiBroker の ORB のプロパティを次の表に示します。

表 35–2 ORB プロパティ（C++）

プロパティ	デフォルト	旧プロパティ	説明
vbroker.orb.propStorage	null	該当しません。	プロパティ値を含むプロパティファイルを指定します。
vbroker.orb.nullstring	false	-ORBnullString	true を設定すると，ORB によって C++の NULL 文字列が送信されます。NULL 文字列は，長さが 0 である文字列としてマーシャリングされます（これに対し，空文字列（""）は，'¥0' という単一文字を持つ長さ 1 の文字列としてマーシャリングされます）。このオプションに false を設定して NULL 文字列をマーシャリングすると，CORBA::BAD_PARAM となります。NULL 文字列をアンマーシャルすると，CORBA::MARSHAL となります。
vbroker.orb.admDir	<環境変数 TPDIR の設定値>/adm	該当しません。	さまざまなシステムファイルが存在する管理ディレクトリを指定します。このプロパティは VBROKER_ADM 環境変数を使って設定できます。環境変数 TPDIR については，マニュアル「TPBroker ユーザーズガイド」を参照してください。
vbroker.orb.isNTService	false	該当しません。	true を指定すると，このプロパティはアプリケーションを Windows のサービスとして実行できるようにして，カレントユーザがログアウトした場合でも終了しないようにします。
vbroker.orb.obv.debug	false	該当しません。	true を指定すると，このプロパティは ORB のオブジェクト値をインプリメンテーションで渡せるようにして，デバッグ情報を表示します。
vbroker.orb.dynamicLibs	null	該当しません。	ORB が明示的にロードするライブラリ（DLL）を指定できます。
vbroker.orb.input.maxBuffers	16	該当しません。	プールに保持されている入力バッファの最大数を指定します。
vbroker.orb.input.buffSize	255	該当しません。	入力バッファのサイズを指定します。
vbroker.orb.output.maxBuffers	16	該当しません。	プールに保持されている出力バッファの最大数を指定します。
vbroker.orb.output.buffSize	255	該当しません。	出力バッファのサイズを指定します。

プロパティ	デフォルト	旧プロパティ	説明
vbroker.orb.initRef	null	該当しません。	初期リファレンスを指定します。
vbroker.orb.defaultInitRef	null	該当しません。	デフォルトの初期リファレンスを指定します。
vbroker.orb.alwaysProxy	false	該当しません。	true を設定すると，リクエストするときに，クライアントは常にゲートキーパーを使用してサーバに接続しなければならないことを指定します。
vbroker.orb.gatekeeper.ior	null	該当しません。	指定した IOR のゲートキーパーを使用して，常にクライアントアプリケーションをサーバに強制的に接続します。
vbroker.locator.ior	null	該当しません。	osagent のプロキシとして使用するゲートキーパーの IOR を指定します。このプロパティを設定しない場合は，vbroker.orb.gatekeeper.ior プロパティで指定したゲートキーパーが使用されます。
vbroker.orb.proxyPassthru	false	該当しません。	アプリケーションスコープ全体にファイアウォールモード「PASSTHROUGH」を適用します。オブジェクトまたは orb ごとにファイアウォールモードを選択する場合は，コードに QoSExt::ProxyModePolicy を使用します。
vbroker.orb.bids.critical	inprocess	該当しません。	vbroker.orb.bidOrder と vbroker.orb.bids.critical に同時に優先順位が設定されている場合は，vbroker.orb.bids.critical の設定が有効となります。vbroker.orb.bids.critical で複数の値が設定されている場合は，vbroker.orb.bidOrder に基づいて相対的な重要度が決まります。
vbroker.orb.bidOrder	inprocess:liop:iiop:proxy:locator	該当しません。	各トランスポートの相対的な重要度を指定します。トランスポートの優先順位は，次のように割り当てられています。 1. inprocess 2. liop 3. iiop 4. proxy 5. locator 例えば，IOR に LIOP と IIOP の両プロファイルが含まれている場合は，LIOP が優先されます。IIOP が使用されるのは，LIOP が失敗したときだけです。ただし，vbroker.orb.bidOrder と

プロパティ	デフォルト	旧プロパティ	説明
			vbroker.orb.bids.critical に同時に優先順位が設定されている場合は，vbroker.orb.bids.critical の設定が有効となります。
vbroker.orb.cacheDSQuery	true	-ORBcacheDSQuery	true を設定すると，VisiBroker アプリケーションは osagent を使用した通信でリファレンスをキャッシュします。リファレンスをキャッシュした場合は，同一サーバに対する 2 度目以降の_bind() 処理では osagent に問い合わせをしないで，プロセス内にキャッシュしたリファレンスを使用します。

35.4 ロケーションサービスプロパティ

C++言語に対応する Borland Enterprise Server VisiBroker のロケーションサービスのプロパティを次の表に示します。

表 35–3 ロケーションサービスプロパティ（C++）

プロパティ	デフォルト	説明
vbroker.locationservice.debug	false	true を設定すると，このプロパティはロケーションサービスがデバッグ情報を表示できるようにします。
vbroker.locationservice.verify	false	true を設定すると，このプロパティはロケーションサービスが，osagent から送られたオブジェクトリファレンスで参照されるオブジェクトがあるかどうかをチェックできるようにします。
vbroker.locationservice.timeout	1	ロケーションサービスとの通信時の接続，受信，および送信のタイムアウトを指定します。

35.5 OAD プロパティ

C++言語に対応する Borland Enterprise Server VisiBroker の OAD のプロパティを次の表に示します。

表 35–4 認定可能な OAD プロパティ（C++）

プロパティ	デフォルト	説明
vbroker.oad.spawnTimeOut	20	このプロパティは，OAD が実行形式ファイルを生成したあと，オブジェクトからコールバックが何秒間来なければ対象オブジェクトを起動するかを指定します。
vbroker.oad.verbose	false	OAD がオペレーションの詳細情報を出力できるようにします。
vbroker.oad.readOnly	false	true を設定すると，OAD インプリメンテーションの登録，登録解除，および変更ができなくなります。
vbroker.oad.iorFile	oadj.ior	OAD の文字列化 IOR のファイル名を指定します。
vbroker.oad.quoteSpaces	false	コマンドを引用するかどうかを指定します。
vbroker.oad.killOnUnregister	false	生成したサーバの登録を解除した場合に，それらを kill するかどうかを指定します。
vbroker.oad.verifyRegistration	false	オブジェクトの登録を確認するかどうかを指定します。

プロパティファイルにオーバーライドできない Borland Enterprise Server VisiBroker の ORB プロパティを次の表に示します。なお，このプロパティは，環境変数またはコマンドラインでオーバーライドできます。

表 35–5 プロパティファイルにオーバーライドできない OAD プロパティ（C++）

プロパティ	デフォルト	説明
vbroker.oad.implName	impl_rep	インプリメンテーションリポジトリのファイル名を指定します。
vbroker.oad.implPath	null	インプリメンテーションリポジトリを格納するディレクトリを指定します。
vbroker.oad.path	null	OAD のディレクトリを指定します。
vbroker.oad.systemRoot	null	ルートディレクトリを指定します。
vbroker.oad.windir	null	Windows ディレクトリを指定します。

35.6　インタフェースリポジトリリゾルバのプロパティ

C++言語に対応する Borland Enterprise Server VisiBroker のインタフェースリポジトリリゾルバのプロパティを次の表に示します。

表 35-6　インタフェースリポジトリリゾルバのプロパティ（C++）

プロパティ	デフォルト	説明
vbroker.ir.debug	false	true を設定すると，IR リゾルバはデバッグ情報を表示できるようになります。
vbroker.ir.ior	null	vbroker.ir.name プロパティにデフォルト値の null が設定されている場合，VisiBroker ORB はこのプロパティを使用して IR を捜し出します。
vbroker.ir.name	null	VisiBroker ORB が IR を探すときに使用する名前を指定します。

35.7　ネーミングサービスプロパティ

ネーミングサービスプロパティについては，マニュアル「Borland Enterprise Server VisiBroker　デベロッパーズガイド」の「ネーミングサービスプロパティ」の記述を参照してください。

35.8 TypeCode のプロパティ

C++言語に対応する Borland Enterprise Server VisiBroker の TypeCode のプロパティを次の表に示します。

表 35-7 TypeCode のプロパティ（C++）

プロパティ	デフォルト	説明
vbroker.typecode.debug	false	true を設定すると，このプロパティは typecode コードがデバッグ情報を表示できるようにします。
vbroker.typecode.noIndirection	false	true を設定すると，このプロパティは再帰的 typecode コードの書き込み時にインディレクションの使用を禁止します。

35.9 クライアント側 IIOP のコネクションプロパティ

C++言語に対応する Borland Enterprise Server VisiBroker のクライアント側 IIOP コネクションのプロパティを次の表に示します。

表 35–8 クライアント側 IIOP のコネクションプロパティ（C++）

プロパティ	デフォルト	説明
vbroker.ce.iiop.ccm.connectionCacheMax	5	1 クライアントがキャッシュできるコネクション数の上限を指定します。コネクションはクライアントが解放したときにキャッシュされます。したがって，次回クライアントが新規コネクションを必要とするときは，クライアントは新規コネクションを生成する前に，キャッシュの中に解放したコネクションが残っていないかどうかを確認し，あればそれを使用します。
vbroker.ce.iiop.ccm.connectionMax	0	1 クライアントの合計コネクション数の上限を指定します。この値は，アクティブなコネクション数とキャッシュされたコネクション数の合計です。デフォルトの 0 を指定すると，クライアントは以前のアクティブなまたはキャッシュされたコネクションをクローズしません。
vbroker.ce.iiop.ccm.connectionMaxIdle	0	クライアントがキャッシュされているコネクションをクローズするかどうかを判定するための基準の時間を，秒単位で指定します。クライアントで使用するオブジェクトリファレンスを解放するときに，キャッシュされたコネクションがこの時間より長く非アクティブ状態になっていると，クライアントはそのコネクションをクローズします。デフォルトの 0 を指定すると，キャッシュされているコネクションをクローズしません。
vbroker.ce.iiop.ccm.type	Pool	クライアントが使用するクライアントコネクション管理の種類を指定します。デフォルト値の Pool はコネクションプールの意味です。
vbroker.ce.iiop.connection.rcvBufSize	0	受信ソケットバッファのサイズを指定します。デフォルト値の 0 は，システムの設定値に依存します。
vbroker.ce.iiop.connection.sendBufSize	0	送信ソケットバッファのサイズを指定します。デフォルト値の 0 は，システムの設定値に依存します。
vbroker.ce.iiop.connection.tcpNoDelay	false	true に設定すると，ORB はサーバへの送信データをコネクションへ書き込むと同時に送信するように，TCP/IP の設定を変更します。
vbroker.ce.iiop.connection.socketLinger	0	VisiBroker が使用する TCP/IP コネクションの設定をします。設定値については，各プラットフォームの setsockopt(2)を参照してください。
vbroker.ce.iiop.connection.keepAlive	true	VisiBroker が使用する TCP/IP コネクションの設定をします。設定値については，各プラットフォームの setsockopt(2)を参照してください。

35.10　サーバ側エンジンプロパティ

サーバ側のサーバエンジンのプロパティを次の表に示します。

表 35–9　サーバ側エンジンプロパティ（C++）

プロパティ	デフォルト	説明
vbroker.se.default	iiop_tp	デフォルトサーバエンジンを指定します。

35.11　サーバ側スレッドセッションコネクションのプロパティ

C++言語に対応する Borland Enterprise Server VisiBroker のサーバ側スレッドセッションコネクションのプロパティを次の表に示します。

xxx はサーバエンジン，yyy はサーバコネクションマネージャです。

- iiop_ts
  このデフォルトサーバエンジンプロパティは，デフォルト IIOP リスナーと一緒に ThreadSession の変種で使用します。

表 35–10　サーバ側スレッドセッションコネクションのプロパティ（C++）

プロパティ	デフォルト	説明
vbroker.se.xxx.host	null	該当するサーバエンジンが使用するホスト名です。デフォルト値の null は，システムからホスト名を使用することを指定します。
vbroker.se.xxx.proxyHost	null	IOR 文字列にホスト名で出力する場合にはこのプロパティにホスト名を指定してください。このプロパティの指定を省略した場合は，vbroker.se.xxx.host の指定に従います。
vbroker.se.xxx.scms	xxx=iiop_ts, iiop_ts	サーバコネクションマネージャ名の一覧です。
vbroker.se.xxx.scm.yyy.manager.type	Socket	サーバコネクションマネージャの種別を指定します。
vbroker.se.xxx.scm.yyy.manager.connectionMax	0	サーバが許可するコネクション数の最大値を指定します。デフォルト値の 0 は，コネクション数を制限しないことを表します。
vbroker.se.xxx.scm.yyy.manager.connectionMaxIdle	0	非アクティブなコネクションをクローズするかどうかをサーバが判定するためのタイムアウトを秒数で指定します。
vbroker.se.xxx.scm.yyy.listener.type	IIOP	リスナーが使用するプロトコルの種別を指定します。
vbroker.se.xxx.scm.yyy.listener.port	0	ホスト名プロパティに使用するポート番号を指定します。デフォルト値の 0 は，システムはポート番号を無作為に選択することを表します。
vbroker.se.xxx.scm.yyy.listener.proxyPort	0	プロキシホスト名プロパティに使用するプロキシポート番号を指定します。デフォルト値の 0 は，システムはポート番号を無作為に選択することを表します。
vbroker.se.xxx.scm.yyy.listener.rcvBufSize	0	受信ソケットバッファのサイズを指定します。デフォルト値の 0 は，システムの設定値に依存します。
vbroker.se.xxx.scm.yyy.listener.sendBufSize	0	送信ソケットバッファのサイズを指定します。デフォルト値の 0 は，システムの設定値に依存します。
vbroker.se.xxx.scm.yyy.listener.socketLinger	0	VisiBroker が使用する TCP/IP コネクションの設定をします。設定値については，各プラットフォームの setsockopt(2)を参照してください。

プロパティ	デフォルト	説明
vbroker.se.xxx.scm.yyy.listener.keepAlive	true	VisiBroker が使用する TCP/IP コネクションの設定をします。設定値については，各プラットフォームの setsockopt(2)を参照してください。
vbroker.se.xxx.scm.yyy.dispatcher.type	ThreadSession	サーバコネクションマネージャに使用するスレッドディスパッチャの種別を指定します。
vbroker.se.xxx.scm.yyy.dispatcher.threadStackSize	デフォルトでは，ORB は設定しません。	スレッドスタックのサイズです。

35.12　サーバ側スレッドプールコネクションのプロパティ

C++言語に対応する Borland Enterprise Server VisiBroker のサーバ側スレッドプールコネクションのプロパティを次の表に示します。

xxx はサーバエンジン，yyy はサーバコネクションマネージャです。

- iiop_tp
  このデフォルトサーバエンジンプロパティは，POA の作成時に使用されます。

表 35–11　サーバ側スレッドプールコネクションのプロパティ（C++）

プロパティ	デフォルト	説明
vbroker.se.xxx.host	null	該当するサーバエンジンが使用するホスト名です。デフォルト値の null は，システムからホスト名を使用することを指定します。
vbroker.se.xxx.proxyHost	null	IOR 文字列にホスト名で出力する場合にはこのプロパティにホスト名を指定してください。このプロパティの指定を省略した場合は，vbroker.se.xxx.host の指定に従います。
vbroker.se.xxx.scms	xxx=iiop_tp, iiop_tp	サーバコネクションマネージャ名の一覧です。
vbroker.se.xxx.scm.yyy.manager.type	Socket	サーバコネクションマネージャの種別を指定します。
vbroker.se.xxx.scm.yyy.manager.connectionMax	0	サーバが受け付けるキャッシュコネクション数の最大値を指定します。デフォルト値の 0 は，コネクション数を制限しないことを表します。
vbroker.se.xxx.scm.yyy.manager.connectionMaxIdle	0	非アクティブなコネクションをクローズするかどうかをサーバが判定するためのタイムアウトを秒数で指定します。
vbroker.se.xxx.scm.yyy.manager.garbageCollectTimer	30	接続用のガーベッジコレクトタイマ（秒）です。
vbroker.se.xxx.scm.yyy.listener.type	IIOP	リスナーが使用するプロトコルの種別です。
vbroker.se.xxx.scm.yyy.listener.port	0	ホスト名プロパティに使用するポート番号です。デフォルト値の 0 は，システムはポート番号を無作為に選択することを表します。
vbroker.se.xxx.scm.yyy.listener.proxyPort	0	プロキシホスト名プロパティに使用するプロキシポート番号です。デフォルト値の 0 は，システムはポート番号を無作為に選択することを表します。
vbroker.se.xxx.scm.yyy.listener.rcvBufSize	0	受信ソケットバッファのサイズを指定します。デフォルト値の 0 は，システムの設定値に依存します。
vbroker.se.xxx.scm.yyy.listener.sendBufSize	0	送信ソケットバッファのサイズを指定します。デフォルト値の 0 は，システムの設定値に依存します。

プロパティ	デフォルト	説明
vbroker.se.xxx.scm.yyy.listener.socketLinger	0	VisiBroker が使用する TCP/IP コネクションの設定をします。設定値については，各プラットフォームの setsockopt(2)を参照してください。
vbroker.se.xxx.scm.yyy.listener.keepAlive	true	VisiBroker が使用する TCP/IP コネクションの設定をします。設定値については，各プラットフォームの setsockopt(2)を参照してください。
vbroker.se.xxx.scm.yyy.dispatcher.type	ThreadPool	サーバコネクションマネージャに使用するスレッドディスパッチャの種別です。
vbroker.se.xxx.scm.yyy.dispatcher.threadMin	0	サーバコネクションマネージャが生成できるスレッド数の下限です。
vbroker.se.xxx.scm.yyy.dispatcher.threadMax	0	サーバコネクションマネージャが生成できるスレッド数の上限です。デフォルト値の 0 は，生成できるスレッド数に上限がないことを表します。
vbroker.se.xxx.scm.yyy.dispatcher.threadMaxIdle	300	アイドルなスレッドをデストラクトするまでのタイムアウトを秒数で指定します。0 を指定した場合は，デフォルトの 300 秒となります。
vbroker.se.xxx.scm.yyy.dispatcher.threadStackSize	デフォルトでは，ORB は設定しません。	スレッドスタックのサイズです。

35.13　双方向通信をサポートするプロパティ

双方向通信をサポートしているプロパティを次の表に示します。

注

これらのプロパティが評価されるのは，SCM が生成されるときの一度だけです。SCM の exportBiDir プロパティと importBiDir プロパティには，enableBiDir プロパティで優先順位が設定されます。つまり，これらのプロパティに矛盾する値が設定された場合は，SCM 固有のプロパティが有効になります。そのため，enableBiDir プロパティをグローバルに設定して，個々の SCM で双方向性を無効にできます。

表 35-12　双方向通信をサポートするプロパティ（C++）

プロパティ	デフォルト	説明
vbroker.orb.enableBiDir	none	双方向コネクションを選択して確立できます。クライアントに vbroker.orb.enableBiDir=client を設定し，サーバに vbroker.orb.enableBiDir=server を設定した場合は，ゲートキーパーの vbroker.orb.enableBiDir の値でコネクションの状態が決まります。このプロパティの値は，server，client，both，または none です。
vbroker.se.<se>.scm.<scm>.manager.exportBiDir	デフォルトでは，ORB は設定しません。	クライアント側プロパティです。true を設定すると，指定したサーバエンジンの双方向コールバック POA を生成できるようになります。false を設定すると，指定したサーバエンジンの双方向コールバック POA を生成できなくなります。
vbroker.se.<se>.scm.<scm>.manager.importBiDir	デフォルトでは，ORB は設定しません。	サーバ側プロパティです。true を設定すると，リクエストをクライアントに送信するために，クライアントによってすでに確立されたコネクションをサーバ側で再使用できます。false を設定した場合は，再使用できません。

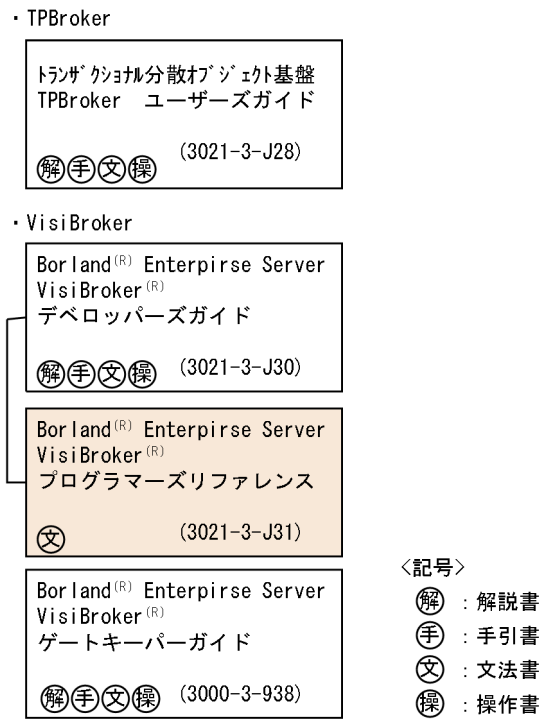
付録

付録 A　このマニュアルの参考情報

このマニュアルを読むに当たっての参考情報を示します。

付録 A.1　関連マニュアル

関連マニュアルを次に示します。必要に応じてお読みください。



なお，CORBA の仕様の詳細については，「The Common Object Request Broker: Architecture and Specification」を参照してください。

付録 A.2　このマニュアルでの表記

このマニュアルでは，製品名を次のように表記しています。

表記	製品名
AIX	AIX V7.1
	AIX V7.2
Borland Enterprise Server VisiBroker	Borland(R) Enterprise Server VisiBroker(R)
HP-UX	HP-UX 11i V2

表記			製品名
			HP-UX 11i V3 (IPF)
IPF			Itanium(R) Processor Family
Java			Java(TM)
JavaVM			Java(TM) Virtual Machine
Linux			Red Hat Enterprise Linux 7.1 (AMD/Intel 64)
			Red Hat Enterprise Linux 8.1 (AMD/Intel 64)
Solaris			Solaris 9
			Solaris 10
VisiBroker			Borland(R) Enterprise Server VisiBroker(R)
VisiBroker 3.x			VisiBroker Version 3.0（x は 0 以上の整数）
VisiBroker 4.x			VisiBroker Version 4.0（x は 0 以上の整数）
VisiBroker 5.x			VisiBroker Version 5.0（x は 0 以上の整数）
Windows	Windows Server 2016	Windows Server 2016 Standard	Windows Server(R) 2016 Standard 日本語版
		Windows Server 2016 Datacenter	Windows Server(R) 2016 Datacenter 日本語版
	Windows Server 2019	Windows Server 2019 Standard	Windows Server(R) 2019 Standard 日本語版
		Windows Server 2019 Datacenter	Windows Server(R) 2019 Datacenter 日本語版
	Windows 10	Windows 10 x64	Windows(R) 10 Enterprise 日本語版 (64 ビット版)

AIX，HP-UX，Linux および Solaris を総称して UNIX と表記しています。

このマニュアルでは，ご使用になるプラットフォームごとに説明が異なる場合，製品名称を次のように表記しています。

表記	意味
(UNIX)	AIX，HP-UX，Linux および Solaris の UNIX プラットフォームに該当
(Windows)	Windows に該当

付録 A.3　英略語

このマニュアルで使用する英略語を次に示します。

英略語	英字での表記
API	Application Programming Interface
BOA	Basic Object Adapter
CORBA	Common Object Request Broker Architecture
DII	Dynamic Invocation Interface
DSI	Dynamic Skeleton Interface
EJB	Enterprise Java Beans(TM)
GIOP	General Inter -ORB Protocol
GUI	Graphical User Interface
HTTP	Hyper Text Transport Protocol
IDL	Interface Definition Language
IEEE	Institute of Electrical and Electronics Engineers
IIOP	Internet Inter-ORB Protocol
IOR	Interoperable Object Reference
IP	Internet Protocol
IPC	Inter Processor Communication
IR	Interface Repository
J2EE	Java(TM) 2 Platform, Enterprise Edition
JAAS	Java(TM) Authentication and Authorization Service
JDK	Java(TM) Development Kit
JSSE	Java Secure Socket Extension
LIOP	Local Inter-ORB Protocol
NDI	Netscape Digital Identity
OAD	Object Activation Daemon
OBV	Object by Value
OMG	Object Management Group
ORB	Object Request Broker
OS	Operating System
POA	Portable Object Adapter

英略語	英字での表記
QoP	Quality of Protection
QoS	Quality of Service
RMI	Remote Method Invocation
SSL	Secure Socket Layer
TCP/IP	Transmission Control Protocol /Internet Protocol
TII	Time-Independent Invocation
URL	Uniform Resource Locator
VM	Virtual Machine

索引

記号

_add_ref 568, 571
_clone 542
_create_request 116, 539
_default_POA 571
_desc 542
_duplicate 535, 541, 584, 587, 607, 608, 612, 617, 619, 624, 839, 844
_forany 511
_get_client_policy 412
_get_interface 541, 571
_get_policy 411
_get_policy_overrides 412
_hash 117, 541
_init クラス例 514
_interface_name 542
_is_a 117, 541, 571
_is_bound 119, 542
_is_equivalent 118, 541
_is_local 119, 542
_is_persistent 119, 543
_is_remote 119, 543
_nil 535, 549, 584, 587, 607, 609, 613, 617, 619, 625, 839, 844
_non_existent 118, 541
_primary_interface 588
_ref_count 543
_release 536, 543, 584, 587, 607, 609, 613, 617, 619, 625, 708, 839, 844
_remove_ref 568, 571
_repository_id 119, 543
_request 118, 542
_resolve_reference 119, 542
_set_policy_override 411
_slice 型 510
_slice 型の生成 510
_target 831
_tie 526
_validate_connection 412
_var 527
<interface_name> 521
<type>SeqSeq のメソッド 859
<type>Seq のメソッド 857
-ORBagent 865
-ORBagentAddr 865
-ORBagentPort 866
-ORBconnectionMax 866
-ORBconnectionMaxIdle 866
-ORBDefaultInitRef 866
-ORBInitRef 866
-ORBir_ior 866
-ORBir_name 866
-ORBnullstring 866
-ORBrcvbufsize 866
-ORBsendbufsize 866
-ORBshmsize 866
-ORBtcpNoDelay 867

A

absolute_name 218, 635
abstract_base_values 281, 687
AbstractInterfaceDef 206
AbstractInterfaceDef のメソッド 206
abstract インタフェース 86
access 287
activable 852
activate 150, 291, 565
activate_object 139, 554
activate_object_with_id 139, 554
activation_policy 296, 695
activation_state 704
ActivationImplDef 290
ActivationImplDef のメソッド 290
Activator 291

activator_obj 290
Activator のメソッド 291
ACTIVE 151, 567
ActiveObjectLifeCycleInterceptor 388, 783
ActiveObjectLifeCycleInterceptorManager 390, 785
ActiveObjectLifeCycleInterceptorManager のメソッド 390, 785
ActiveObjectLifeCycleInterceptor のメソッド 388, 783
adapter_id 366, 761
adapter_manager_state_changed 352
adapter_state_changed 352
adapter_template 348, 744
add 165, 182, 188, 380, 384, 387, 390, 395, 397, 401, 586, 606, 610, 775, 779, 782, 785, 790, 792, 796
add_client_request_interceptor 355, 750
add_consume 586, 606
add_in_arg 193, 614
add_inout_arg 193, 614
add_ior_component 348, 744
add_ior_component_to_profile 348, 744
add_ior_interceptor 356, 751
add_item 188, 610
add_item_consume 611
add_named_in_arg 193
add_named_inout_arg 193
add_named_out_arg 193
add_out_arg 193, 615
add_reply_service_context 366, 761
add_request_service_context 333, 729
add_server_request_interceptor 350, 356, 746, 750
add_value 189, 611
add_value_consume 611
addClientObjectWrapperClass 102
addServerObjectWrapperClass 102
Agent 440, 846
agent_hostname 852
Agent のメソッド 440, 847
AliasDef 209, 628
AliasDef のメソッド 209, 628
all_agent_locations 440, 847
all_available 847
all_available_descs 847
all_instances 441, 848
all_instances_descs 441, 848
all_replica 441, 848
all_replica_descs 442, 849
all_repository_ids 442, 849
allocate_slot_id 356, 751
allocbuf 707
Any 159, 583
Any 型のマッピング 93
Any の挿入メソッド 161
Any の抽出メソッド 160
Any のメソッド 159, 583
ARG_IN 162
ARG_INOUT 163
ARG_INOUT の変数 163
ARG_IN の変数 162
ARG_OUT 164
ARG_OUT の変数 164
args 296, 695
argument 358, 752
arguments 193, 196, 354, 360, 615, 618, 749, 755
array 82
array_slice 型 510
ArrayDef 210, 629
ArrayDef のメソッド 210, 629
assign 168, 590
AttributeDef 211, 630
AttributeDef のメソッド 211, 630
AttributeDescription 212, 631
AttributeDescription の変数 212
AttributeDescription のメソッド 212
AttributeDescription のメンバ 631

AttributeMode 214, 632
AttributeMode の値 632
AttributeMode の要素 214

B

BAD_CONTEXT 326
BAD_INV_ORDER 326
BAD_OPERATION 326
BAD_PARAM 326
BAD_TYPECODE 326
base_interfaces 206, 245, 658
base_value 281, 687
bind 101, 132, 308, 377, 552, 710, 772
bind_context 310, 712
bind_failed 378, 773
bind_new_context 313, 714
bind_succeeded 378, 773
Binding 318, 719
BindingIterator 319, 720
BindingIterator のメソッド 319, 720
BindingList 318, 719
BindInterceptor 377, 772
BindInterceptorManager 380, 775
BindInterceptorManager のメソッド 380, 775
BindInterceptor のメソッド 377, 772
boolean 70
Borland Enterprise Server VisiBroker プロパティ (C++) 869
Borland Enterprise Server VisiBroker プロパティ (Java) 458
bound 270, 271, 678, 679, 692
buffer 837, 842
byte_order 826, 837

C

C++での IDL 定数の値の生成 497
CancelRequestHeader 827
CancelRequestHeader のメンバ 827
ChainUntypedObjectWrapperFactory 401, 796
ChainUntypedObjectWrapperFactory のメソッド 401, 796
change_implementation 301, 700
char 70
CLASSPATH 294, 695
clear 181, 605
ClientRequestInfo 331, 727
ClientRequestInfo のメソッド 332, 728
ClientRequestInterceptor 335, 381, 731, 776
ClientRequestInterceptorManager 384, 779
ClientRequestInterceptorManager のメソッド 384, 779
ClientRequestInterceptor のメソッド 335, 381, 731, 776
Closure 399
Codec 338, 734
codec_factory 354, 749
CodecFactory 340, 736
CodecFactory のメソッド 340, 736
CodecFactory のメンバ 340, 736
Codec のメソッド 338, 734
Codec のメンバ 338, 734
CODESET_INCOMPATIBLE 326
com.inprise.vbroker.CORBA.Object のメソッド 412
COMM_FAILURE 326
compare 707
completed 576
CompletionStatus 110, 531
CompletionStatus のメソッド 110
CompletionStatus のメンバ 110, 531
component_count 590
components 832
components_established 351
concrete_base_type 625
conn_closed 804
conn_established 804
connect 435
connection_count 551
connection_max 551

ConnEventListeners 804
connID 805
ConnInfo 805
ConnInfo のメンバ 805
ConstantDef 215, 633
ConstantDef のメソッド 215, 633
ConstantDescription 216, 634
ConstantDescription の変数 216
ConstantDescription のメソッド 216
ConstantDescription のメンバ 634
Contained 218, 635
contained_object 648
ContainedPackage.Description 221
ContainedPackage.Description の変数 221
ContainedPackage.Description のメソッド 221
Contained のメソッド 218, 635
Container 222, 638
ContainerPackage.Description 232
ContainerPackage.Description の変数 232
ContainerPackage.Description のメソッド 232
Container のメソッド 224, 638
containing_repository 219, 636
content_type 200, 623
contents 224, 638
Context 111, 534
context_name 112, 534
ContextList 165, 586
ContextList のメソッド 165, 586
contexts 193, 257, 361, 615, 666, 755
Context のメソッド 112, 534
copy 168, 428, 590
CORBA 532
CORBA::Any 583
CORBA::ContextList 586
CORBA::ExceptionList 606
CORBA::INV_POLICY 824
CORBA::is_nil 612
CORBA::MarshalInBuffer 836
CORBA::MarshalInBuffer のコンストラクタとデストラクタ 836
CORBA::MarshalInBuffer のメソッド 837
CORBA::MarshalOutBuffer 841
CORBA::MarshalOutBuffer のコンストラクタとデストラクタ 841
CORBA::MarshalOutBuffer のメソッド 842
CORBA::Object 811
CORBA::Object に対する VisiBroker の拡張機能 542
CORBA::Object のメソッド 539, 811
CORBA::ORB に対する VisiBroker の拡張機能 550
CORBA::ORB のメソッド 544
CORBA::PolicyCurrent 810
CORBA::PolicyError 824
CORBA::PolicyManager 808
CORBA::PolicyManager のメソッド 808
CORBA::REBIND 824
CORBA::release 612
CORBA::StringSequence 706
CORBA のメソッド 532
count 165, 182, 189, 402, 586, 606, 612, 797
create 385, 388, 396, 405, 780, 783, 791, 802
create_abstract_interface 206, 224
create_alias 224, 639
create_alias_tc 124, 544
create_any 124
create_array 268, 675
create_array_tc 124, 544
create_attribute 207, 245, 283, 658, 689
create_child 112, 534
create_codec 340, 736
create_constant 225, 639
create_context 321, 722
create_context_list 124
create_CreationImplDef 302, 701
create_dyn_any 124, 172, 593
create_dyn_any_from_type_code 172, 593
create_enum 225, 640

create_enum_tc 125, 545
create_environment 125, 545
create_exception 226, 640
create_exception_tc 125, 545
create_fixed 269
create_fixed 676
create_id_assignment_policy 141, 556
create_id_uniqueness_policy 141, 556
create_implicit_activation_policy 140, 555
create_input_stream 133, 159
create_interface 226, 641
create_interface_tc 125, 545
create_lifespan_policy 141, 556
create_list 126, 546
create_module 227, 641
create_named_value 126, 546
create_native 227
create_operation 207, 245, 283, 659, 689
create_operation_list 126, 546
create_output_stream 133, 159
create_POA 141, 557
create_policy 126, 359, 550, 754
create_recursive_sequence_tc 127, 546
create_reference 140, 555
create_reference_with_id 140, 555
create_request_processing_policy 142, 557
create_sequence 268, 676
create_sequence_tc 127, 546
create_servant_retention_policy 143, 558
create_string 268, 676
create_string_tc 127, 546
create_struct 227, 641
create_struct_tc 127, 547
create_thread_policy 143, 558
create_union 228, 642
create_union_tc 127, 547
create_value 229, 643
create_value_box 230, 644
create_value_member 282, 688
create_wstring 269, 676
create_wstring_tc 128, 551
CreationImplDef 293, 694
CreationImplDef 属性から実行コマンドへの変換例 294
CreationImplDef のメソッド 296
CreationImplDef のメンバ 695
ctx 193, 197, 615, 618
curoff 837, 842
Current 341, 737
current_component 168, 590
current_environment 604
current_factory 349, 745
current_member_kind 176, 600
current_member_name 176, 600
Current のメソッド 341, 737

D

data 794
DATA_CONVERSION 326
deactivate 150, 291, 565
deactivate_object 143, 558
decode 339, 735
decode_value 339, 735
def_kind 251, 662
default_index 200, 623
DeferBindPolicy 420
defined_in 219, 636
defined_in 属性 638
DefinitionKind 233, 646
DefinitionKind の定数値（C++） 646
DefinitionKind の定数値（Java） 233
DefinitionKind のメソッド 233
DefinitionKind の列挙値 233, 646
delete_values 112, 534
Desc 444, 852
describe 219, 636
describe_contents 228, 642
describe_interface 208, 246, 659

describe_value 282, 688
Description 221, 232, 648
Description のメンバ 648
Desc のコンストラクタ 444
Desc の変数 444
Desc のメソッド 445
Desc のメンバ 852
destroy 144, 168, 251, 313, 320, 346, 385, 388, 559, 590, 594, 600, 662, 714, 720, 742, 781, 783
destroy_on_unregister 302, 701
digits 239, 652
discard_requests 150, 565
DISCARDING 151, 567
discriminator_kind 178, 602
discriminator_type 200, 277, 623, 684
discriminator_type_def 277, 684
DuplicateName 354, 749
DynamicImplementation 180, 588
DynamicImplementation のメソッド 180, 588
DynAny 167, 589
DynAnyFactory 172, 593
DynAnyFactory のメソッド 172, 593
DynAny の挿入メソッド 169, 592
DynAny の抽出メソッド 169, 591
DynAny のメソッド 168, 590
DynArray 171, 594
DynArray のメソッド 171, 594
DynEnum 173, 596
DynEnum のメソッド 173, 596
DynSequence 174, 598
DynSequence のメソッド 174, 598
DynStruct 176, 600
DynStruct のメソッド 176, 600
DynUnion 178, 602
DynUnion のメソッド 178, 602

E

effective_profile 332, 728
effective_target 332, 728
element_type 210, 270, 629, 678
element_type_def 209, 210, 270, 629, 678
encode 338, 734
encode_value 339, 735
Encoding 343, 739
Encoding のメンバ 343, 739
enum 76
EnumDef 235, 649
EnumDef のメソッド 235, 649
enum の Holder クラス 77
enum の Java にマッピングされた IDL 77
env 194, 297, 615, 695
Environment 181, 604
Environment のメソッド 181, 604
equal 159, 200, 590, 623
equivalent 625
establish_components 351, 747
etherealize 152, 569
EventQueueManager 803
EventQueueManager のメソッド 803
except 197
exception 181, 604
Exception 538
exception_occurred 378, 382, 393, 773, 778, 788
ExceptionDef 236, 650
ExceptionDef のメソッド 236, 650
ExceptionDescription 237, 651
ExceptionDescription の変数 237
ExceptionDescription のメソッド 237
ExceptionDescription のメンバ 651
ExceptionList 182, 606, 740
ExceptionList のメソッド 182, 606
exceptions 194, 257, 360, 615, 667, 755
ExclusiveConnectionPolicy 421
exportObject 434
ExtendedClosure 400, 793
ExtendedNamingContextFactory 323, 724

ExtendedNamingContextFactory のメソッド 323, 724
extract 100

F

Fail 446, 854
FailReason 854
Fail のメンバ 854
Fail 変数 446
find_POA 144, 559
FixedDef 239, 652
FixedDef のメソッド 239, 652
flags 186, 608
force_register_url 437
format 343, 739
FormatMismatch 338, 734
forward_reference 362, 757, 786
ForwardRequest 157, 344, 741
ForwardRequestException 391, 786
ForwardRequestException の変数 391, 786
ForwardRequest の変数 157, 344
ForwardRequest のメソッド 157, 344
FREE_MEM 326
freebuf 707
freebuf_elems 707
from_any 168, 590, 596
from_int 198
full_poa_name 350, 746
FullInterfaceDescription 247, 653
FullInterfaceDescription のメンバ 653
FullValueDescription 240, 241, 654
FullValueDescription の変数 240, 654
FullValueDescription のメソッド 241

G

generated_command 302
generated_environment 302
get 837
get_as_string 173, 596
get_as_ulong 173, 596
get_client_policy 811
get_cluster_manager 321, 722
get_compact_typecode 625
get_default_context 128, 547
get_discriminator 178, 602
get_effective_component 333, 729
get_effective_policy 347, 743
get_elements 171, 174, 594, 598
get_elements_as_dyn_any 594, 599
get_implementation 302, 701
get_length 174, 598
get_listeners 803
get_manager 376, 771
get_members 176, 600
get_members_as_dyn_any 177, 600
get_next_response 128, 547
get_object_id 138, 537
get_POA 138, 537
get_policy 811
get_policy_overrides 408, 808, 812
get_primitive 269, 676
get_primitive_tc 129
get_reply_service_context 363, 758
get_request_policy 333, 729
get_request_service_context 363, 757
get_response 194, 616
get_servant 144, 559
get_servant_manager 145, 560
get_server_policy 366, 761
get_slot 341, 363, 737, 757
get_state 150, 566
get_status 303, 701
get_status_all 303, 702
get_status_interface 303, 702
get_values 112, 535
getCString 838
getPropertyManager 551
getWString 838

GIOP_version 826

H

has_no_active_member 179, 603
hash 707
Helper 62, 100
Helper クラス 72, 97
Helper のメソッド 100
hold_requests 150, 566
Holder 62, 104, 105
Holder クラス 66, 98
Holder のメソッド 105
HOLDING 151, 567
host 832
host_name 855

I

id 100, 200, 219, 290, 297, 623, 636, 696
id_to_reference 145, 560
id_to_servant 145, 560
idl2cpp 485
idl2ir 37, 489
idl2java 40
idl2java コンパイラで生成されるクラス 96
IDL array 82
IDL enum 76
IDL sequence 81
IDL struct 78
IDLType 243, 656
IDLType のメソッド 243, 656
IDL union 79
IDL union の C++クラスへのマッピング 505
IDL 型 Any 93
IDL 型拡張 66
IDL 可変長シーケンス 507
IDL 可変長シーケンスの C++クラスへのマッピング 507
IDL から C++言語へのマッピング 492
IDL から Java へのマッピング 60
IDL 基本型マッピング（C++） 493
IDL コンパイラで生成されるクラス 519
IDL 内の可変長構造体定義 504
IDL 内の固定長構造体定義 503
IDL の型定義 499
IDL の型定義の C++へのマッピング 499
IDL のトップレベル定義 496
IDL 配列定義 510
IDL 配列の C++配列へのマッピング 510
IDL モジュール 64
IDL モジュール定義 501
iiop_locator 852
iiop_version 832
IIOP.ProfileBody 426
IIOP.ProfileBody のコンストラクタ 427
IIOP.ProfileBody の変数 426
IMP_LIMIT 327
impl 698
impl_is_down 449, 856
impl_is_ready 449, 856
ImplementationDef 299, 697
ImplementationStatus 301, 698
ImplementationStatus のメンバ 698
INACTIVE 151, 567
incarnate 153, 570
init 129, 134
INITIALIZE 327
initializers 281, 687
InputStream 184
InputStream からデータを読み出すためのメソッド 184
InputStream のメソッド 184
insert 100
instance 797
instance_name 852, 855
Interceptor 346, 742
InterceptorManager 375, 770
InterceptorManagerControl 376, 771
InterceptorManagerControl のメソッド 376, 771

Interceptor のメソッド 346, 742
interface 84
InterfaceDef 244, 657
InterfaceDefPackage.FullInterfaceDescription 247
InterfaceDefPackage.FullInterfaceDescription の変数 247
InterfaceDefPackage.FullInterfaceDescription のメソッド 247
InterfaceDef のメソッド 245, 658
InterfaceDescription 249, 661
InterfaceDescription の変数 249
InterfaceDescription のメソッド 249
InterfaceDescription のメンバ 661
interface の Holder クラス 84
INTERNAL 327
INTF_REPOS 327
INV_FLAG 327
INV_IDENT 327
INV_OBJREF 327
INV_POLICY 327
INVALID_TRANSACTION 327
InvalidName 114, 354, 749
InvalidTypeForEncoding 338, 734
invoke 180, 194, 588, 616
in パラメタのマッピング 87
IOP.IORValue 428
IOP.IORValue の変数 428
IOP.IORValue のメソッド 428
IOP.ServiceContext 430
IOP.ServiceContext のコンストラクタ 430
IOP.ServiceContext の変数 430
IOP.TaggedProfile 431
IOP.TaggedProfile のコンストラクタ 431
IOP.TaggedProfile の変数 431
IOP および IIOP のインタフェースとクラス（C++） 825
IOP および IIOP のインタフェースとクラス（Java） 425
IOR 833
IORCreationInterceptor 396, 791
IORCreationInterceptorManager 397, 792
IORCreationInterceptorManager のメソッド 397, 792
IORCreationInterceptor のメソッド 396, 791
IORInfo 347, 743
IORInfoExt 350, 746
IORInfoExt のメソッド 350, 746
IORInfo のメソッド 347, 743
IORInterceptor 351, 747
IORInterceptor のメソッド 351, 747
IOR インタセプタ 330, 726
IOR テンプレート 374, 769
IOR のメンバ 833
ipaddress 805
ir2idl 39, 491
IRObject 251, 662
IRObject のメソッド 251, 662
is_a 208, 246, 282, 659, 688
is_abstract 208, 281, 687
is_available 838
is_custom 282, 688
is_permanent 786
is_truncatable 282, 688
item 165, 182, 189, 586, 606, 612

J

java2idl 44
java2iiop 47
Java null 70
Java VM の指定 57
Java 言語の予約キーワード 63
JDK での ORB の定義 121
JDK での ORB メソッド 123

K

kind 201, 265, 623, 673

L

LD_LIBRARY_PATH 294, 695
length 201, 210, 623, 629, 705, 708, 839, 842, 858, 859
LIBPATH 294, 695
list 314, 714
list_all_roots 321, 722
list_initial_services 129, 548
LocalInterfaceDef 252
locate 437
locate_status 828
LocateReplyHeader 828
LocateReplyHeader のメンバ 828
LocateRequestHeader 829
LocateRequestHeader のメンバ 829
Location 398
Location のメンバ 398
lookup 229, 643
lookup_id 269, 676
lookup_implementation 304
lookup_interface 304
lookup_name 229, 643
lookup_value_factory 550

M

magic 826
major_version 343, 739
managedData 794
manager_id 348, 744
MARSHAL 327
matchesTemplate 428
maximum 705, 708
member 178, 602
member_count 201, 623
member_kind 179, 602
member_label 201, 624
member_name 179, 201, 602, 624
member_type 202, 624
member_visibility 625
members 235, 236, 272, 277, 649, 650, 680, 684
message_size 826
message_type 826
MessageHeader 826
MessageHeader のメンバ 826
Messaging::RebindPolicy 814
Messaging::RelativeRequestTimeoutPolicy 819
Messaging::RelativeRoundtripTimeoutPolicy 820
Messaging::SyncScopePolicy 822
Messaging.RelativeRequestTimeoutPolicy 418
Messaging.RelativeRoundtripTimeoutPolicy 419
minor_version 343, 739
mode 211, 257, 358, 630, 667, 752
ModuleDef 253, 663
ModuleDescription 254, 664
ModuleDescription の変数 254
ModuleDescription のメソッド 254
ModuleDescription のメンバ 664
module 句 501
move 219, 636

N

name 186, 202, 219, 346, 608, 624, 636, 742
NamedValue 186, 608
NamedValue のメソッド 186, 608
NameValuePair 187
NameValuePair のコンストラクタ 187
NameValuePair の変数 187
NamingContext 308, 710
NamingContextExt 315, 716
NamingContextExt のメソッド 315, 716
NamingContextFactory 321, 722
NamingContextFactory のメソッド 321, 722
NamingContext のメソッド 308, 710
narrow 101, 435
NativeDef 256, 665
new_context 313, 714
new_encapsulation 839, 843

next 168, 590
next_n 319, 720
next_one 319, 720
NO_IMPLEMENT 327
NO_MEMORY 327
NO_PERMISSION 327
NO_REBIND 414, 814
NO_RECONNECT 414, 815
NO_RESOURCES 327
NO_RESPONSE 327
NVList 188, 610
NVList のメソッド 188, 610

O

OAD 300, 699
OAD のメソッド 301, 700
OAD プロパティ 472, 877
OBJ_ADAPTER 327
Object 115, 411, 539
object_id 365, 760
object_key 832
object_name 297, 696
OBJECT_NOT_EXIST 327
object_to_string 129, 548
ObjectId_to_string 530
ObjectStatus 704
ObjectStatusList 705
ObjectStatusList のメソッド 705
ObjectStatus のメンバ 704
ObjectWrapper 523
Object の継承 119
Object のメソッドの継承 119
objRef 704
octet 70
OMG による ORB の定義 132
op_name 197
operation 194, 196, 360, 616, 618, 755, 831
operation_context 361, 756
OperationDef 257, 666
OperationDef のメソッド 257, 666
OperationDescription 259, 669
OperationDescription の変数 259
OperationDescription のメソッド 259
OperationDescription のメンバ 669
OperationMode 261, 670
OperationMode の値 670
Operations 62, 99
Operations クラス 97
operator 705, 708, 857, 859
ORB 121, 544
orb_id 354, 749
ORB_init 532, 862
ORB_init()メソッド 865
ORB_init()メソッド定義 870
ORB_initialized 863
ORB_init オプション（C++） 865
ORB_shutdown 863
ORB::create_environment 604
ORB.init()の使用例 453
ORB.init( )メソッド 453
ORB.init オプション（Java） 453
ORBagentAddr 453
ORBagentAddrFile 453
ORBagentNoFailOver 453
ORBagentPort 454
ORBalwaysProxy 454
ORBalwaysTunnel 454
ORBconnectionMax 454
ORBconnectionMaxIdle 454
ORBdebug 454
ORBDefaultInitRef 455
ORBdisableAgentCache 455
ORBdisableGatekeeperCallbacks 455
ORBdisableLocator 455
ORBELINE 695
ORBgatekeeperIOR 455
ORBgcTimeout 455
ORBInitializer 353, 748

ORBInitializer のメソッド 353, 748
ORBInitInfo 354, 749
ORBInitInfo のメソッド 354, 749
ORBInitInfo のメンバ 354
ORBInitRef 455
ORBmbufSize 455
ORBnullString 455
ORBwarn 455
ORB オプションの設定方法 451
ORB クラスのクライアント使用例 121
ORB の継承 132
ORB の初期化と終了 134
ORB プロパティ 464, 873
org.omg.CORBA.Object の定義 115
org.omg.CORBA.Object のメソッド 116, 411
original_type_def 209, 280, 628, 686
OSAGENT_ADDR 294, 695
OSAGENT_ADDR_FILE 294
OSAGENT_CLIENT_HANDLER_PORT 296
OSAGENT_LOCAL_FILE 294
OSAGENT_PORT 294, 695
osagent（スマートエージェント）プロパティ 462, 871
OutputStream 190
OutputStream に特定の型を書き込むためのメソッド 190
OutputStream のメソッド 190
out パラメタおよび inout パラメタの Holder 87

P

Package 62
Parameter 358, 752
ParameterDescription 262, 671
ParameterDescription の変数 262
ParameterDescription のメソッド 262
ParameterDescription のメンバ 671
ParameterList 753
ParameterMode 264, 672
ParameterMode の値 672
Parameter のメンバ 358, 752
params 197, 258, 667
parent 113, 535
PATH 294, 695
path_name 297, 696
perform_work 129, 548
PERSIST_STORE 327
POA 62, 107, 525
POALifeCycleInterceptor 385, 780
POALifeCycleInterceptorManager 387, 782
POALifeCycleInterceptorManager のメソッド 387, 782
POALifeCycleInterceptor のメソッド 385, 780
POATie 62, 108
POATie クラス 98
POA クラス 98
POA プロパティ 469
POA マネージャの状態遷移 149
Policy 136
PolicyCurrent 410
PolicyFactory 359, 754
PolicyFactory のメソッド 359, 754
PolicyManager 408
PolicyManager のメソッド 408
poll_next_response 130, 548
poll_response 194, 616
port 805, 832
PortableRemoteObject 434
PortableRemoteObject のコンストラクタ 434
PortableRemoteObject のメソッド 434
PortableServer 530
PortableServer::AdapterActivator 529
PortableServer::AdapterActivator のメソッド 529
PortableServer::Current 537
PortableServer::Current のメソッド 537
PortableServer::ForwardRequest 575
PortableServer::ForwardRequest のメソッド 575
PortableServer::POA 554
PortableServer::POAManager 564

PortableServer::POAManager::State 567
PortableServer::POAManager::State のメソッド 567
PortableServer::POAManager のメソッド 565
PortableServer::POA のメソッド 554
PortableServer::RefCountServantBase 568
PortableServer::RefCountServantBase のメソッド 568
PortableServer::ServantActivator 569
PortableServer::ServantActivator のメソッド 569
PortableServer::ServantBase 571
PortableServer::ServantBase のメソッド 571
PortableServer::ServantLocator 572
PortableServer::ServantLocator のメソッド 572
PortableServer::ServantManager 574
PortableServer.AdapterActivator 137
PortableServer.AdapterActivator のメソッド 137
PortableServer.Current 138
PortableServer.Current のメソッド 138
PortableServer.ForwardRequest 157
PortableServer.POA 139
PortableServer.POAManager 149
PortableServer.POAManagerPackage.State 151
PortableServer.POAManagerPackage.State のメンバ 151
PortableServer.POAManager のメソッド 150
PortableServer.POA のメソッド 139
PortableServer.ServantActivator 152
PortableServer.ServantActivator のメソッド 152
PortableServer.ServantLocator 154
PortableServer.ServantLocator のメソッド 154
PortableServer.ServantManager 156
PortableServer のメソッド 530
post_init 353, 748
post_method 404, 799
postinvoke 155, 382, 573, 777
postinvoke_postmarshal 393, 788
postinvoke_premarshal 392, 787
pre_init 353, 748
pre_method 403, 799
preinvoke 154, 392, 572, 787
preinvoke_postmarshal 382, 777
preinvoke_premarshal 381, 776
PrimitiveDef 265, 673
PrimitiveDef のメソッド 265, 673
PrimitiveKind 266, 674
PrimitiveKind の定数値（C++） 674
PrimitiveKind の定数値（Java） 266
PrimitiveKind のメソッド 266
Principal 512
profile_data 834
ProfileBody 427, 832
ProfileBody のメンバ 832
profiles 833
property 580
put 843
putCString 843

Q

QoSExt::DeferBindPolicy 816
QoSExt::ExclusiveConnectionPolicy 817
QoSExt::RelativeConnectionTimeoutPolicy 818
QoS インタフェースとクラス（C++） 806
QoS インタフェースとクラス（Java） 406
QoS 例外 424, 824

R

read 100
read_value 159
reason 854
rebind 309, 711
REBIND 327
rebind_context 311, 712
RebindForwardPolicy 416
RebindPolicy 413
receive_exception 336, 732
receive_other 336, 732
receive_reply 336, 732

receive_request 368, 763
receive_request_service_contexts 368, 763
received_exception 332, 728
received_exception_id 333, 729
ref 852
reference_to_id 146, 561
reference_to_servant 145, 560
reg_implementation 305, 702
reg_trigger 442, 849
register_initial_reference 355, 750
register_listener 803
register_policy_factory 356, 751
register_url 438
register_value_factory 549
RelativeConnectionTimeoutPolicy 417
release_flag 839, 843
remove 165, 183, 189, 402, 586, 607, 612, 779, 797
remove_stale_contexts 321, 722
removeClientObjectWrapperClass 102
removeServerObjectWrapperClass 103
reply_status 362, 756, 830
ReplyHeader 830
ReplyHeader のメンバ 830
Repository 268, 675
repository_id 297, 696, 852, 855
Repository のメソッド 268, 675
Request 192, 614
request_id 360, 755
RequestHeader 831
RequestHeader のメンバ 831
RequestInfo 360, 400, 755
RequestInfo のメソッド 360, 755
Request のメソッド 193, 614
reset 839, 844
resolve 311, 713
resolve_initial_references 130, 548, 750
resolve_str 317, 717
Resolver 437
Resolver のメソッド 437
response_expected 361, 756, 831
result 194, 197, 258, 361, 616, 667, 756
result_def 258, 667
return_value 194, 616
rewind 168, 591, 839, 844
RMI インタフェースとクラス（Java） 433
RMI-IIOP プロパティ 461
root_context 323, 724
run 131, 549

S

scale 239, 652
seek 168, 591
seekpos 839, 844
send_deferred 195, 616
send_exception 369, 764
send_multiple_requests_deferred 131, 549
send_multiple_requests_oneway 131, 549
send_oneway 195, 616
send_other 370, 765
send_poll 731
send_reply 369, 764
send_request 335, 731
sending_exception 365, 760
Seq 857
SeqSeq 859
sequence 81
Sequence 507
SequenceDef 270, 678
SequenceDef のメソッド 270, 678
sequence の Holder クラス 81
servant_to_id 146, 561
servant_to_reference 147, 561
SERVER_PER_METHOD 293
ServerRequest 196, 618
ServerRequestInfo 364, 759
ServerRequestInfo のメソッド 365, 760
ServerRequestInterceptor 368, 392, 763, 787

ServerRequestInterceptorManager 395, 790
ServerRequestInterceptorManager のメソッド 395, 790
ServerRequestInterceptor のメソッド 368, 392, 763, 787
ServerRequest のメソッド 196, 618
service_context 831
service_name 290
ServiceContext 430
set_as_string 173, 596
set_as_ulong 173, 597
set_discriminator 178, 602
set_elements 171, 174, 594, 598
set_elements_as_dyn_any 594, 598
set_exception 197, 618
set_length 174, 598
set_members 176, 600
set_members_as_dyn_any 177, 600
set_one_value 113, 536
set_policy_overrides 408, 808, 812
set_result 196, 619
set_return_type 195, 616
set_servant 147, 562
set_servant_manager 147, 562
set_slot 341, 366, 737, 761
set_to_default_member 179, 602
set_to_no_active_member 179, 603
set_values 113, 536
SHARED_SERVER 293
SHLIB_PATH 294, 695
shutdown 321, 550, 723
Signature クラス 97
state 348, 744
status 698
string 70
string_alloc 532
string_free 532
string_to_object 131, 549
string_to_ObjectId 530
String_var クラス 494
StringDef 271, 679
StringDef のメソッド 271, 679
StringSequence 706
StringSequence に関連するメソッド 707
StringSequence のメソッド 706
struct 78
StructDef 272, 680
StructDef のメソッド 272, 680
StructMember 273, 681
StructMember の変数 273
StructMember のメソッド 273
StructMember のメンバ 681
struct の Holder クラス 78
struct を含む IDL union 505
Stub 106
supported_interfaces 281, 687
SYNC_NONE 422, 822
sync_scope 361, 756
SYNC_WITH_SERVER 423, 823
SYNC_WITH_TARGET 423, 823
SYNC_WITH_TRANSPORT 423, 823
SyncScopePolicy 422
SyncScope ポリシーの値 422, 822
SystemException 576
SystemException の属性 326
SystemException のメソッド 576

T

tag 834
TaggedProfile 431, 834
TaggedProfile のメンバ 834
target 195, 332, 617, 728, 829
target_is_a 366, 761
target_most_derived_interface 366, 761
TCKind 198, 620
TCKind のメソッド 198
the_activator 148, 562
the_name 148, 563

the_parent 148, 563
the_POAManager 148, 563
the_policies 148
tie クラス 526
tie のメソッド 108
TIMOUT 327
to_any 169, 591, 596
to_name 316, 716
to_string 315, 716
to_url 316, 717
toIOR 428
toString 445, 448
toStub 434
TPDIR 294
TRANSACTION_REQUIRED 327
TRANSACTION_ROLLEDBACK 327
TRANSIENT 327
TRANSPARENT 414, 814
TriggerDesc 447, 855
TriggerDesc のコンストラクタ 447
TriggerDesc の変数 447
TriggerDesc のメソッド 448
TriggerDesc のメンバ 855
TriggerHandler 449, 856
TriggerHandler のメソッド 449, 856
type 100, 160, 169, 211, 215, 236, 243, 287, 591, 630, 633, 650, 656
type_def 211, 215, 287, 630, 633
type_id 833
type_modifier 625
TypeCode 199, 622
TypeCode のコンストラクタ 622
TypeCode のプロパティ 880
TypeCode のメソッド 200, 623
TypedefDef 274, 682
typedef のマッピング 95
typedef 列の Java Helper クラスへのマッピング 73
TypeDescription 275, 683
TypeDescription の変数 275
TypeDescription のメソッド 275
TypeDescription のメンバ 683
TypeMismatch 338, 734

U

ULong id 794
unbind 312, 713
unexportObject 434
Unicode 文字列 692
union 79, 505, 684
UnionDef 277, 684
UnionDef のメソッド 277, 684
UnionMember 278, 685
UnionMember の変数 278
UnionMember のメソッド 278
UnionMember のメンバ 685
union でのメモリ管理 506
union の Holder クラス 80
union の管理型 506
union の情報 685
unique_id 704
UNKNOWN 327
unknown_adapter 137, 529
UnknownEncoding 340, 736
UnknownUserException 204
unreg_implementation 305, 702
unreg_interface 305, 703
unreg_trigger 443, 850
unregister_all 306, 703
unregister_listener 803
unregister_value_factory 550
UNSHARED_SERVER 293
UntypedObjectWrapper 403, 799
UntypedObjectWrapperFactory 405, 801
UntypedObjectWrapperFactory のコンストラクタ 801
UntypedObjectWrapperFactory のメソッド 405, 802
UntypedObjectWrapper のメソッド 403, 799

URL ネーミングインタフェースとクラス（Java） 436
URL ネーミングプロパティ 474
UserException 579
UserException のコンストラクタ 328

V

validate_connection 812
value 186, 198, 215, 233, 266, 580, 608, 633
Valuebox 516
ValueBoxDef 280, 686
ValueBoxDef のメソッド 280, 686
ValueDef 281, 687
ValueDef のメソッド 281, 687
ValueDescription 285, 691
ValueDescription の変数 285, 691
ValueDescription のメソッド 285
ValueMemberDef 287
ValueMemberDef のメソッド 287
Valuetype 513
var クラス 527
VB_NO_REBIND 415, 815
VB_NOTIFY_REBIND 415, 815
VB_TRANSPARENT 414, 815
vbj 52, 451
vbjc 54
VBROKER_ADM 294, 695
version 219, 636
VISClosure 794
VISClosureData 795
VISClosure のメンバ 794
VisiBroker 4.x インタセプタおよびオブジェクトラッパーのインタフェースとクラス（C++） 766
VisiBroker 4.x インタセプタおよびオブジェクトラッパーのインタフェースとクラス（Java） 371
VisiBroker 4.x のインタセプタ 330, 726
VisiBroker ORB レベルのポリシー 407
VISInit 862
VISInit のメソッド 862
VISPropertyManager 580
VISPropertyManager のメソッド 580

W

willRefreshOADs 850
work_pending 131, 544
write 100
write_value 160
wstring 71
wstring_alloc 533
wstring_free 533
WstringDef 288, 692
WstringDef のメソッド 692

あ

アクセッサメソッド 506
アクティブ 149, 564
アドオンサービスの詳細 130
暗黙的に渡される環境変数 295

い

一方向モード 670
一般オプション 36, 483
インタセプタマネージャ 373, 768
インタフェース型定義の C++へのマッピング 499
インタフェーススコープ 90
インタフェース属性 630
インタフェース内にない定数 74
インタフェース内の定数 74
インタフェースの IDL typedef 499
インタフェースリポジトリインタフェースとクラス（C++） 627
インタフェースリポジトリインタフェースとクラス（Java） 205
インタフェースリポジトリオブジェクト 662
インタフェースリポジトリプロパティ 473
インタフェースリポジトリリゾルバのプロパティ 878

え

エイリアス 628
演算子 840, 844

お

応答ヘッダ　830
オブジェクトリファレンスの情報　833
オブジェクトレベルのポリシー　407
オペレーションの型　683
オペレーションの情報　669
オペレーションのモード　670

か

型定義　499
型の一覧（TCKind）（C++）　620
活性化インタフェースとクラス（C++）　693
活性化インタフェースとクラス（Java）　289
活性化ポリシー　293
可変長構造体　504
可変長構造体でのメモリ管理　504
可変長構造体の C++へのマッピング　504
環境変数　295, 695
完了ステータス　576

き

基本 Java 型　65
基本型　65, 673
基本型の Holder クラス　67
基本型マッピング（Java）　65
基本型を定義する定数　674
基本データ型　493
キャンセルされるリクエスト識別子　827

く

クライアント側 IIOP のコネクションプロパティ　881
クライアント側コネクションプロパティ　475
クライアント側プロセス内コネクションプロパティ　476
クライアントスタブ　485
クラスパスの指定　55

け

継承によるサーバインプリメンテーション　88

こ

コアインタフェースとクラス（C++）　528
コアインタフェースとクラス（Java）　109
構造型　76
構造体　503, 680
コールバックオブジェクト　856
固定長 IDL 構造体の C++へのマッピング　503
固定長構造体　503
コマンドラインオプション（C++）　864
コマンドラインオプション（Java）　450
コンパイラ　485

さ

サーバ側エンジンプロパティ　477, 882
サーバ側スレッドセッション IIOP_TS プロパティ，および IIOP_TS コネクションプロパティ　478
サーバ側スレッドセッションコネクションのプロパティ　883
サーバ側スレッドプール IIOP_TP プロパティ，および IIOP_TP コネクションプロパティ　479
サーバ側スレッドプールコネクションのプロパティ　885
サーバスケルトン　485

し

シーケンス　678
シーケンス型定義の C++へのマッピング　500
シーケンスでのメモリ管理　509
シーケンスの IDL typedef　499
シーケンスの管理型　508
システム例外　92, 326, 576
システム例外一覧（C++）　577
システム例外一覧（Java）　326
初期化インタフェースとクラス（C++）　861
初期化演算子　584
シンプル IDL 型　95
シンプル Java 型にマッピングされる IDL 型　95

す

スタブクラス　98, 521

スマートエージェントのポート番号　866
スレッドレベルのポリシー　407

せ

整数型　71
生成されるインタフェースとクラス（C++）　519
生成されるインタフェースとクラス（Java）　96
静的メソッド　72

そ

双方向通信をサポートするプロパティ　481, 887
属性の情報　631
属性のモード　632
その他のツール　58

た

待機　149, 564
タイプセーフ配列　511

ち

抽出演算子　585
抽象インタフェース　517
抽象インタフェースの C++マッピング　517

て

定数　74, 496, 653
定数定義　633
定数の C++コード　496
定数の情報　634
定数を含む特別なケース　496
デリゲーションを使用したサーバインプリメンテーション　89

と

動的インタフェースとクラス（C++）　582
動的インタフェースとクラス（Java）　158
トリガー　846

な

名前　61
名前付き型の Java Helper クラスへのマッピング　72

ね

ネーミングサービスインタフェースとクラス（C++）　709
ネーミングサービスインタフェースとクラス（Java）　307
ネーミングサービスプロパティ　471, 879
ネストされた型のマッピング　94
ネストされたスコープの Java パッケージ名　62

は

配列　510, 629
配列スライス　510
配列でのメモリ管理　512
配列の Holder クラス　82
配列の管理型　510
配列のマッピング　82
破棄　149, 564
ハッシュ値　541, 707
パラメタ　671
パラメタの受け渡し　87
パラメタのモード　672

ひ

非アクティブ　149, 564
標準 IDL システム例外　92
標準 IDL 例外の Java クラス名　92
標準モード　670

ふ

複合 idl typedef のマッピング　95
複合 IDL 型　95
複合データ型　502
複次元配列の IDL 定義　510
浮動小数点型　71
プログラマツール（C++）　482
プログラマツール（Java）　35
プログラマツールの動作環境　483
プロトコルの情報　832

プロパティの設定方法　870

ほ

ポータビリティスケルトンインタフェース　98
ポータビリティスタブインタフェース　98
ポータブルインタセプタ　330, 726
ポータブルインタセプタインタフェースとクラス（C++）　725
ポータブルインタセプタインタフェースとクラス（Java）　329
ポリシーの値　414, 814

ま

マーシャルバッファインタフェースとクラス（C++）　835
マイナーバージョン　826
マッピング　60, 492

め

明示的に渡される環境変数　295
メジャーバージョン　826
メッセージの情報　826
メモリ管理　527

も

モジュール　64, 663
モジュール内にない IDL 宣言　64
文字列　494, 679
文字列を動的に割り当てる　494

ゆ

ユーザ定義型に生成された Java コード　72
ユーザ定義型の Holder クラス　69
ユーザ定義構造体　682
ユーザ定義例外　91
ユーザ定義例外のマッピング　91
ユーザ例外　328, 579

よ

予約語　63
予約名　62

り

リクエストインタセプタ　330, 726
リクエストヘッダ　831
リポジトリへのアクセス　675

れ

例外　650
例外クラス（Java）　324
例外の情報　651
例外のマッピング　91
列挙体　498, 649
列挙体の C++enum へのダイレクトマッピング　498
列挙体の IDL 定義　498

ろ

ローカルインタフェース　86
ロケーションサービスインタフェースとクラス（C++）　845
ロケーションサービスインタフェースとクラス（Java）　439
ロケーションサービスオプション　457, 868
ロケーションサービスプロパティ　470, 876
ロケートリクエストメッセージ　828