HITACHI
Inspire the Next

# COBOL2002　Java プログラム呼び出し機能ガイド

手引・操作書

4010-1J-807

COBOL2002

# 前書き

## ■ 対象製品

P-2636-2354 COBOL2002 Net Developer 05-00（適用 OS：Windows 10(x64)，Windows 11，Windows Server 2019，Windows Server 2022）

P-2436-5354 COBOL2002 Net Server Runtime 05-00（適用 OS：Windows Server 2019，Windows Server 2022）

P-2636-3354 COBOL2002 Net Client Runtime 05-00（適用 OS：Windows 10(x64)，Windows 11）

P-2436-6354 COBOL2002 Net Server Suite 05-00（適用 OS：Windows Server 2019，Windows Server 2022）

P-2636-4354 COBOL2002 Net Client Suite 05-00（適用 OS：Windows 10(x64)，Windows 11）

P-2936-2354 COBOL2002 Net Developer(64) 05-00（適用 OS：Windows 10(x64)，Windows 11，Windows Server 2019，Windows Server 2022）

P-2936-5354 COBOL2002 Net Server Runtime(64) 05-00（適用 OS：Windows Server 2019，Windows Server 2022）

P-2936-6354 COBOL2002 Net Server Suite(64) 05-00（適用 OS：Windows Server 2019，Windows Server 2022）

P-2636-7354 COBOL2002 Developer Professional 05-00（適用 OS：Windows 10(x64)，Windows 11，Windows Server 2019，Windows Server 2022）

P-2936-7354 COBOL2002 Developer Professional(64) 05-00（適用 OS：Windows 10(x64)，Windows 11，Windows Server 2019，Windows Server 2022）

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



## ■ 発行

2025 年 2 月　4010-1J-807

## ■ 著作権

# はじめに

このマニュアルは，次のプログラムプロダクトの機能，およびプログラムの作成方法について説明したものです。

- P-2636-2354 COBOL2002 Net Developer
- P-2436-5354 COBOL2002 Net Server Runtime
- P-2636-3354 COBOL2002 Net Client Runtime
- P-2436-6354 COBOL2002 Net Server Suite
- P-2636-4354 COBOL2002 Net Client Suite
- P-2936-2354 COBOL2002 Net Developer(64)
- P-2936-5354 COBOL2002 Net Server Runtime(64)
- P-2936-6354 COBOL2002 Net Server Suite(64)
- P-2636-7354 COBOL2002 Developer Professional
- P-2936-7354 COBOL2002 Developer Professional(64)

前提ソフトウェアの詳細については，「リリースノート」を参照してください。

## ■ 対象読者

Java 言語と COBOL 言語について基本的な知識を持っていて，両言語の通信を簡単に実現したい方を対象としています。

## ■ このマニュアルで使用する記号

このマニュアルで使用する記号を次に示します。

| 記号 | 意味 |
|---|---|
| { } | この記号で囲まれている複数の項目のうちから一つを選択することを意味する。項目が縦に複数行にわたって記述されている場合は，そのうちの 1 行分を選択する。 |
| { } + | この記号で囲まれている複数の項目のうちから一つ以上を選択することを意味する。複数指定でき，同じ要素の繰り返しでも指定できる。例えば，{A \| B \| C} +と表記されている場合は，A・B・C をすべて選択できる。 |
| [ ] | 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する。<br>複数の項目が縦または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，記号 { } と同じく，どれか一つを選択する。 |
| … | 記述が省略されていることを意味する。 |

| 記号 | 意味 |
| --- | --- |
| | この記号の直前に示された項目を繰り返して複数個指定できる。 |
| 下線 | 括弧で囲まれた複数の項目のうち 1 項目に対して使用され，括弧内のすべてを省略したときにシステムがとる標準値を意味する。 |
| \| | 横に並べられた複数の項目に対して項目間の区切りを示し，「または」を意味する。 |
| *斜体文字* | ユーザが値を指定することを意味する。 |
| *斜体文字と下線* | Java プログラム呼び出し機能のプログラム作成支援ツールが決定する値を意味する。 |

# 目次

# 1 Java プログラム呼び出し機能の概要

この章では，COBOL で作成したユーザアプリケーションプログラム（以降，UAP と表記します）から Java プログラムを使用する場合の Java プログラム呼び出し機能の特長と構成について説明します。

## 1.1　Java プログラム呼び出し機能とは

Java プログラム呼び出し機能では，JNI（Java Native Interface）を使って，同一プロセスで Java のオブジェクトのフィールド（変数）にアクセスしたり，Java のメソッドを呼び出したりするための COBOL プログラム向けのサービスルーチンを提供します。

Java プログラム呼び出し機能のサービスルーチンの特長を次に示します。

- CALL 文での Java プログラムの呼び出しができます。
  Java クラスおよび Java インスタンスオブジェクトを使用または操作するためのサービスルーチンを提供します。
- Java の高速な実行が期待できます。
  COBOL プログラムと同一プロセス内での Java クラスの使用または操作を実装します。
- COBOL プログラムで使用する文字列と Java の String クラスのデータ変換を容易にします。

また，Java プログラム呼び出し機能を使用する COBOL プログラムの作成を容易にするツール（プログラム作成支援ツール）を提供します。プログラム作成支援ツールは COBOL2002 Developer Professional でだけ使用できます。

32bit 版 COBOL2002 で作成した COBOL プログラムからは 32bit Java プログラムだけを，64bit 版 COBOL2002 で作成した COBOL プログラムからは 64bit Java プログラムだけを呼び出せます。

**注意事項**

Java プログラム呼び出し機能を使用して呼び出した Java プログラムから，同じスレッド内で COBOL プログラムを呼び出すことはできません。COBOL プログラムを別プロセスで呼び出すなどの対処をしてください。

## 1.2　Java プログラム呼び出し機能の使用例

Java プログラム呼び出し機能の使用例を説明します。

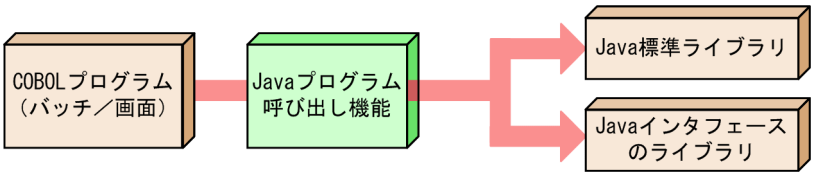
### 1.2.1　Java 標準ライブラリや Java インタフェースのパッケージをプログラム部品として使用する

Java プログラム呼び出し機能を使用すると，Java 標準ライブラリや Java インタフェースのパッケージをプログラム部品として使用できます。

COBOL プログラムから豊富な Java の部品を使用できます。また，Java で開発したプログラムを COBOL プログラムから呼び出すこともできます。

Java のパッケージをプログラム部品として使用する例を次の図に示します。

図 1-1　Java のパッケージをプログラム部品として使用する例



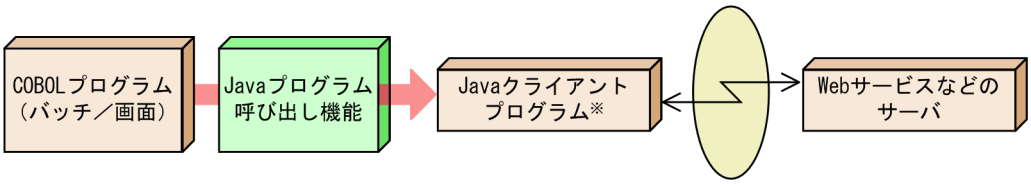

### 1.2.2　Web サービスなどと通信する Java クライアント部品を呼び出す

Java プログラム呼び出し機能を使用すると，Web サービスなどと通信する Java クライアント部品を呼び出すことができます。

COBOL で作成したバッチアプリケーションなどから，Java で作成された Web サービスを使用できます。

Web サービスなどと通信する Java クライアント部品を呼び出す例を次の図に示します。

図 1-2　Web サービスなどと通信する Java クライアント部品を呼び出す例



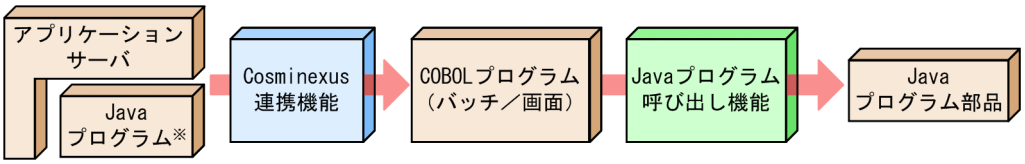

注※
ユーザプログラムや各社提供のパッケージ製品

### 1.2.3　Cosminexus 連携機能から呼び出した COBOL プログラムから Java プログラム部品を呼び出す

Cosminexus 連携機能を使用して，Java プログラムから COBOL プログラムを呼び出すことができます。さらに，Java プログラム呼び出し機能を使用して，Java プログラムから呼び出された COBOL プログラムから Java プログラム部品を呼び出せます。

Cosminexus 連携機能から呼び出した COBOL プログラムから Java プログラム部品を呼び出す例を次の図に示します。

図 1-3　Cosminexus 連携機能から呼び出した COBOL プログラムから Java プログラム部品を呼び出す例




注※
ServletやWebサービスなど

# 2 環境設定

この章では，Java プログラム呼び出し機能を使用するための環境設定について説明します。

## 2.1　前提ソフトウェア

Java プログラム呼び出し機能を使用するには，次に示すソフトウェアで提供される JDK が必要です。

- Cosminexus
- Oracle Java SE

**注意事項**

Cosminexus の実行環境製品で提供される JDK を Cosminexus 環境下のアプリケーションとは別プロセスで使用する場合は，Cosminexus のオプション製品である uCosminexus スタンドアロンプログラム実行機能を購入してください。

Java プログラム呼び出し機能は，Cosminexus および Oracle 社 Java の JDK に対応しています。Cosminexus または Oracle 社 Java の JDK を使用するように，Java の環境を設定してください。

このマニュアルでは，特に記載しないかぎり，Cosminexus で提供される JDK を前提として説明します。

## 2.2　JNI の使用を制限しているときの注意事項

COBOL プログラムを呼び出すソフトウェアや使用するライブラリで，JNI の使用を制限している場合，Java プログラム呼び出し機能を使用できません。

使用するソフトウェアおよびライブラリで，JNI の使用を制限していないことを確認してください。

## 2.3　実行時に設定が必要な環境変数

Java プログラム呼び出し機能を使用する場合に，あらかじめ設定を検討する必要がある環境変数を次に示します。

- CBLJRTERR

  実行時エラー情報ファイルの出力有無，および実行時エラー情報の出力先フォルダのパス名を指定します。

  実行時エラー情報ファイルについては，「5.3　実行時エラー情報の出力」を参照してください。

  環境変数 CBLJRTERR については，「6.2.2　実行時環境変数の詳細」の「(4)　CBLJRTERR」を参照してください。

- CBLJRTVMDEFAULTOPTIONS

  デフォルトの Java VM 起動オプションを指定します。

  Java VM 起動オプションについては，「4.2.2　Java VM 起動オプション」を参照してください。

  環境変数 CBLJRTVMDEFAULTOPTIONS については，「6.2.2　実行時環境変数の詳細」の「(5) CBLJRTVMDEFAULTOPTIONS」を参照してください。

- CBLJRTVMOPTLOG

  Java VM 起動オプション情報の出力先を変更する場合に，出力先フォルダのパス名を指定します。

  Java VM 起動オプション情報については，「5.4　Java VM 起動オプション情報の収集」を参照してください。

  環境変数 CBLJRTVMOPTLOG については，「6.2.2　実行時環境変数の詳細」の「(7) CBLJRTVMOPTLOG」を参照してください。

- CBLJRTVMOPTLOG_MAXSIZE

  Java VM 起動オプション情報の出力先ファイルの最大サイズを指定します。

  Java VM 起動オプション情報については，「5.4　Java VM 起動オプション情報の収集」を参照してください。

  環境変数 CBLJRTVMOPTLOG_MAXSIZE については，「6.2.2　実行時環境変数の詳細」の「(8) CBLJRTVMOPTLOG_MAXSIZE」を参照してください。

上記以外の実行時環境変数については，「6.2　実行時環境変数」を参照してください。

COBOL2002 の実行時環境変数については，マニュアル「COBOL2002　ユーザーズガイド」を参照してください。

## 2.4　日立以外の Java VM を使用する場合の起動オプション

Java プログラム呼び出し機能を使用する場合，プロセス単位に Java VM を起動する必要があります。

プロセス実行時に Java VM が起動していない場合，Java プログラム呼び出し機能が Java VM を起動します。デフォルトの起動オプションは，Cosminexus 製品に含まれる日立製 Java VM 固有の拡張オプションです。そのため，日立製以外の Java VM を使用する場合は，環境変数 CBLJRTVMDEFAULTOPTIONS を使用してデフォルトの起動オプションを取り消す必要があります。

環境変数 CBLJRTVMDEFAULTOPTIONS については，「6.2.2　実行時環境変数の詳細」の「(5) CBLJRTVMDEFAULTOPTIONS」を参照してください。

なお，Java VM の起動オプションには，障害発生時の調査のために，使用する Java VM が提供する詳細情報取得用のオプションを指定することをお勧めします。

Java VM 起動オプションについては，「4.2.2　Java VM 起動オプション」を参照してください。

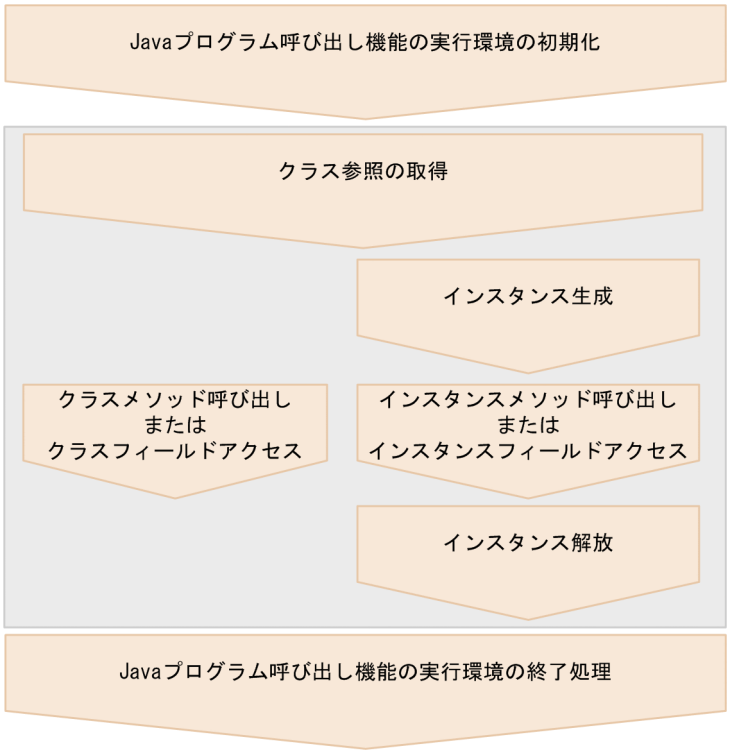
# 3 Java プログラムを使用する COBOL プログラムの開発

この章では，Java プログラム呼び出し機能を使用して COBOL の UAP を開発する手順について説明します。

## 3.1　処理の流れ

Java プログラム呼び出し機能を使用する COBOL プログラムの処理の流れを次の図に示します。

図 3-1　Java プログラム呼び出し機能を使用する COBOL プログラムの処理の流れ




Java プログラム呼び出し機能の実行環境の初期化については，「クラス参照の取得」で自動的に行われるため，明示的な初期化は不要です。

Java プログラム呼び出し機能の実行環境の終了処理については，すべての Java クラスの操作が終わったあとで実行してください。

## 3.2 プログラムの作成

ここでは，Java プログラム呼び出し機能を使用した COBOL プログラムの各処理のコーディングや，コーディング時の注意事項について説明します。

### 3.2.1 Java クラス操作のコーディング

ここでは，次に示す Java プログラムを操作する COBOL プログラムについて説明します。

Java プログラムのソースファイル例

```
public class SampleClass {
   public static String SampleStaticField;
   public           int SampleField;
   public static   void SampleStaticMethod( int a, double b ) { ... }
   public           int SampleMethod( String a ) { ... }
}
```

COBOL プログラムのソースファイル例

```
       IDENTIFICATION        DIVISION.
       PROGRAM-ID.           MAIN.
       ENVIRONMENT           DIVISION.
       CONFIGURATION         SECTION.
       SPECIAL-NAMES.
       DYNAMIC LENGTH STRUCTURE C-STRING IS C-STATIC-STRUCTURE.
       DATA                  DIVISION.
       WORKING-STORAGE       SECTION.
      *>-----------------------------------------------------------------
      *> CBLJENV集団項目の定義
      *>-----------------------------------------------------------------
       01  CBLJENV.
         02  CBLJENVCORE     USAGE POINTER        VALUE NULL.
         02  CBLJEXCEPTION   USAGE POINTER        VALUE NULL.
         02  CBLJFLAGS       PIC 1(32) USAGE BIT  VALUE ALL B'0'.
         02  CBLJSTRMAXLEN   PIC S9(9) USAGE COMP VALUE 256.
         02  CBLJVMOPTIONS.
           03  CBLJOPTCOUNT  PIC S9(9) USAGE COMP VALUE 1.
           03  CBLJOPTION-1  PIC X(256) VALUE '-Djava.class.path=java'.
      *>-----------------------------------------------------------------
      *> クラス名，フィールド名，メソッド名の定義
      *>-----------------------------------------------------------------
       01  SampleClass        PIC X DYNAMIC C-STRING
                              VALUE 'SampleClass'.
       01  SampleStaticField  PIC X DYNAMIC C-STRING
                              VALUE 'SampleStaticField'.
       01  SampleField        PIC X DYNAMIC C-STRING
                              VALUE 'SampleField'.
       01  SampleStaticMethod PIC X DYNAMIC C-STRING
                              VALUE 'SampleStaticMethod'.
       01  SampleMethod       PIC X DYNAMIC C-STRING
                              VALUE 'SampleMethod'.
      *>-----------------------------------------------------------------
```

```
*> クラス参照・オブジェクト参照の定義
*>-----------------------------------------------------------------
 01  CLASSREF          USAGE POINTER.
 01  OBJREF            USAGE POINTER.
*>-----------------------------------------------------------------
*> 引数リストの定義（NO-ARGは引数なしメソッド用の引数リスト）
*>-----------------------------------------------------------------
 01  NO-ARG.
   02  FILLER          USAGE POINTER VALUE NULL.
 01  ARG-LIST.
   02  ARGPTR          USAGE POINTER OCCURS 3 TIMES.
*>-----------------------------------------------------------------
*> パラメタ領域の定義(RTN-VOIDはvoid型メソッド用の返却項目指定)
*>-----------------------------------------------------------------
 01  ARG-INT.
   02  ARG-INT-TYPE    PIC X(1) VALUE 'I'.
   02  FILLER          PIC X(7) VALUE ALL LOW-VALUE.
   02  ARG-INT-AREA    PIC S9(9) USAGE COMP.
 01  ARG-DOUBLE.
   02  ARG-DBL-TYPE    PIC X(1) VALUE 'D'.
   02  FILLER          PIC X(7) VALUE ALL LOW-VALUE.
   02  ARG-DBL-AREA    USAGE COMP-2.
 01  ARG-STRING.
   02  ARG-STR-TYPE    PIC X(256) VALUE 'Ljava/lang/String;'.
   02  ARG-STR-AREA    USAGE POINTER.
 01  RTN-VOID.
   02  RTN-TYPE        PIC X(1) VALUE 'V'.
 01  RTN-INT.
   02  RTN-INT-TYPE    PIC X(1) VALUE 'I'.
   02  FILLER          PIC X(7) VALUE ALL LOW-VALUE.
   02  RTN-INT-AREA    PIC S9(9) USAGE COMP.
 01  RTN-STRING.
   02  RTN-STR-TYPE    PIC X(256) VALUE 'Ljava/lang/String;'.
   02  RTN-STR-AREA    USAGE POINTER.
*>-----------------------------------------------------------------
*> 作業領域
*>-----------------------------------------------------------------
 01  WK-STRING-LEN     PIC S9(9) USAGE COMP VALUE 256.
 01  WK-STRING         PIC X(256).

 PROCEDURE DIVISION.

*> JavaのSampleClassクラスのクラス参照を取得する

     CALL 'CBLJGETCLASS' USING CBLJENV SampleClass CLASSREF.

*> Javaのクラスフィールド(SampleStaticField)に値を設定する

     MOVE  'HITACHI' TO WK-STRING.
     CALL 'CBLJXTOSTRING' USING CBLJENV
             WK-STRING  WK-STRING-LEN  ARG-STR-AREA.

     CALL 'CBLJSETSTATICFIELD' USING CBLJENV
             CLASSREF SampleStaticField ARG-STRING.

*> Javaのクラスフィールド(SampleStaticField)から値を取り出す

     CALL 'CBLJGETSTATICFIELD' USING CBLJENV
```

```
              CLASSREF SampleStaticField RTN-STRING.

      *> Javaのクラスメソッド(SampleStaticMethod)を呼び出す

           COMPUTE ARG-INT-AREA = 10.
           COMPUTE ARG-DBL-AREA = 3.14.
           COMPUTE ARGPTR(1) = FUNCTION ADDR( ARG-INT ).
           COMPUTE ARGPTR(2) = FUNCTION ADDR( ARG-DOUBLE ).
           COMPUTE ARGPTR(3) = ZERO.
           CALL 'CBLJSTATICINVOKE' USING CBLJENV
                  CLASSREF SampleStaticMethod ARG-LIST  RTN-VOID.

      *> Javaのインスタンスを生成する

           CALL 'CBLJNEW' USING CBLJENV CLASSREF NO-ARG  OBJREF.

      *> Javaのインスタンスフィールド(SampleField)に値を設定する

           MOVE  100 TO ARG-INT-AREA.
           CALL 'CBLJSETFIELD' USING CBLJENV
                  OBJREF SampleField  ARG-INT.

      *> Javaのインスタンスフィールド(SampleField)から値を取り出す

           CALL 'CBLJGETFIELD' USING CBLJENV
                  OBJREF SampleField  RTN-INT.

      *> Javaのインスタンスメソッド(SampleMethod)を呼び出す

           COMPUTE ARG-STR-AREA = RTN-STR-AREA.
           COMPUTE ARGPTR(1) = FUNCTION ADDR( ARG-STRING ).
           COMPUTE ARGPTR(2) = ZERO.
           CALL 'CBLJINVOKE' USING CBLJENV
                  OBJREF SampleMethod ARG-LIST  RTN-INT.

      *> Javaのインスタンスを解放する

           CALL 'CBLJRELEASE'  USING CBLJENV OBJREF.

      *> Javaの実行環境を削除する

           CALL 'CBLJFINALIZE' USING CBLJENV.

       END PROGRAM MAIN.
```

## (1) パラメタ領域の定義

Java プログラム呼び出し機能のサービスルーチンを使用するには，受け渡しで使用するパラメタ領域を定義する必要があります。

### (a) CBLJENV 集団項目

CBLJENV 集団項目には，Java の実行環境を保持させるための領域を記述します。

Java VM 起動時のパラメタは，CBLJENV 集団項目に指定します。ただし，ほかのスレッドのプログラムですでに Java VM が起動されている場合は，CBLJENV 集団項目の CBLJVMOPTIONS 項目に指定したオプションは適用されません。また，環境変数 CBLJVMOPTIONS が指定されているときは，CBLJVMOPTIONS 項目の指定は無効となります。

CBLJENV 集団項目の CBLJSTRMAXLEN 項目には，Java プログラム呼び出し機能のサービスルーチンに渡す固定長英数字項目の長さを指定します。CBLJVMOPTIONS やパラメタ型集団項目の DATA-TYPE 項目では，CBLJSTRMAXLEN 項目に指定した長さの英数字項目を指定してください。

CBLJENV 集団項目の例を次に示します。

```
*>------------------------------------------------------------
*> CBLJENV集団項目の定義
*>------------------------------------------------------------
 01  CBLJENV.
   02  CBLJENVCORE     USAGE POINTER        VALUE NULL.
   02  CBLJEXCEPTION   USAGE POINTER        VALUE NULL.
   02  CBLJFLAGS       PIC 1(32) USAGE BIT  VALUE ALL B'0'.
   02  CBLJSTRMAXLEN   PIC S9(9) USAGE COMP VALUE 256.
   02  CBLJVMOPTIONS.
     03  CBLJOPTCOUNT  PIC S9(9) USAGE COMP VALUE 1.
     03  CBLJOPTION-1  PIC X(256) VALUE '-Djava.class.path=java'.
```

（凡例）

-Djava.class.path：.class ファイルの検索パスをソースプログラム中で指定する場合のオプションです。

CBLJENV 集団項目は，CBLJINITIALIZE サービスルーチンが明示的または暗黙的に呼び出される前に値を設定します。CBLJINITIALIZE サービスルーチンが呼び出されたあとは変更しないでください。

CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。

パラメタ型集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(3)　パラメタ型集団項目」を参照してください。

### (b)　クラス名，フィールド名，およびメソッド名

使用する Java 名称（クラス名，フィールド名，およびメソッド名）について，動的長基本項目を使用して次のように記述します。

```
*>------------------------------------------------------------
*> クラス名，フィールド名，メソッド名の定義
*>------------------------------------------------------------
 01  SampleClass        PIC X DYNAMIC C-STRING VALUE 'SampleClass'.
 01  SampleStaticField  PIC X DYNAMIC C-STRING
                        VALUE 'SampleStaticField'.
 01  SampleField        PIC X DYNAMIC C-STRING
                        VALUE 'SampleField'.
 01  SampleStaticMethod PIC X DYNAMIC C-STRING
                        VALUE 'SampleStaticMethod'.
```

```
       01  SampleMethod        PIC X DYNAMIC C-STRING
                               VALUE 'SampleMethod'.
```

パッケージ名付きのクラス名を記述する場合は，ピリオドではなく，スラントを使って区切ります。

例）

```
java/lang/String
```

項目名を Java 名称と同じにすると，読みやすいプログラムになります。

また，動的長基本項目を使用しない従来の構文を使用した名前型集団項目を使用して定義することもできます。名前型集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(2)　名前型集団項目」を参照してください。

**注意事項**

LIMIT 句を指定しない動的長基本項目は，プログラム中で値を変更できません。

一つのデータ項目に対して，プログラム中で名称を転記して使用したい場合には，LIMIT 句を指定した動的長基本項目か，名前型集団項目を定義して使用します。このとき，データ項目は使用する名称の最大長で定義する必要があります。

LIMIT 句を指定した動的長基本項目を使用する場合

```
       01  MethodName          PIC X DYNAMIC C-STRING LIMIT 256.

           MOVE 'SampleStaticMethod'     TO MethodName.
           CALL 'CBLJSTATICINVOKE' USING CBLJENV
                   CLASSREF MethodName ARG-LIST  RTN-VOID.
```

名前型集団項目を使用する場合

```
       01  MethodName
         02  NAMESTR          PIC X(256).
         02  FILLER           PIC X     VALUE LOW-VALUE.

           MOVE 'SampleStaticMethod'     TO NAMESTR OF MethodName.
           CALL 'CBLJSTATICINVOKE' USING CBLJENV
                   CLASSREF MethodName ARG-LIST  RTN-VOID.
```

### (c)　引数リストおよびパラメタ領域

Java のメソッドの呼び出しや Java のフィールドの出し入れで使用する領域を記述します。

引数リストおよびパラメタ領域の例を次に示します。

```
      *>-------------------------------------------------------------
      *> 引数リストの定義（NO-ARGは引数なしメソッド用の引数リスト）
      *>-------------------------------------------------------------
       01  NO-ARG.
```

```
      02  FILLER          USAGE POINTER VALUE NULL.
    01  ARG-LIST.
      02  ARGPTR          USAGE POINTER OCCURS 3 TIMES.
   *>-------------------------------------------------------------
   *> パラメタ領域の定義(RTN-VOIDはvoid型メソッド用の返却項目指定)
   *>-------------------------------------------------------------
    01  ARG-INT.
      02  ARG-INT-TYPE    PIC X(1) VALUE 'I'.
      02  FILLER          PIC X(7) VALUE ALL LOW-VALUE.
      02  ARG-INT-AREA    PIC S9(9) USAGE COMP.
    01  ARG-DOUBLE.
      02  ARG-DBL-TYPE    PIC X(1) VALUE 'D'.
      02  FILLER          PIC X(7) VALUE ALL LOW-VALUE.
      02  ARG-DBL-AREA    USAGE COMP-2.
    01  ARG-STRING.
      02  ARG-STR-TYPE    PIC X(256) VALUE 'Ljava/lang/String;'.
      02  ARG-STR-AREA    USAGE POINTER.
    01  RTN-VOID.
      02  RTN-TYPE        PIC X(1) VALUE 'V'.
    01  RTN-INT.
      02  RTN-INT-TYPE    PIC X(1) VALUE 'I'.
      02  FILLER          PIC X(7) VALUE ALL LOW-VALUE.
      02  RTN-INT-AREA    PIC S9(9) USAGE COMP.
    01  RTN-STRING.
      02  RTN-STR-TYPE    PIC X(256) VALUE 'Ljava/lang/String;'.
      02  RTN-STR-AREA    USAGE POINTER.
```

引数リスト型集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(4)　引数リスト型集団項目」を参照してください。

パラメタ型集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(3)　パラメタ型集団項目」を参照してください。

**注意事項**

- 引数の最大個数は 16 個です。
- オブジェクト型を扱う ARG-STR-TYPE や RTN-STR-TYPE の英数字項目のけた数は，CBLJENV 集団項目の CBLJSTRMAXLEN 項目に指定した値と同じにしてください。

## (2)　Java プログラム呼び出し機能の初期化

CBLJINITIALIZE サービスルーチンを呼び出して，Java プログラム呼び出し機能を初期化します。

CBLJINITIALIZE サービスルーチンを呼び出す例を次に示します。

```
           CALL 'CBLJINITIALIZE' USING CBLJENV.
```

この例では，最初の CBLJGETCLASS サービスルーチンの呼び出し時に，このサービスルーチンが自動で呼び出されるため，明示的に呼び出していません。

CBLJINITIALIZE サービスルーチンについては，「6.1.2　基本操作」の「(1)　CBLJINITIALIZE」を参照してください。

## (3)　クラス参照の取得

CBLJGETCLASS サービスルーチンを呼び出して，使用する Java クラスを初期化し，クラス参照を取得します。

CBLJGETCLASS サービスルーチンを呼び出して，Java の SampleClass クラスのクラス参照を取得する例を次に示します。

```
      *> JavaのSampleClassクラスのクラス参照を取得する

           CALL 'CBLJGETCLASS' USING CBLJENV SampleClass CLASSREF.
```

取得したクラス参照は次の操作で使用します。

- クラスメソッドの呼び出し
- クラスフィールドのアクセス
- クラスのインスタンスの生成
- インスタンスオブジェクトのクラスの判定

CBLJGETCLASS サービスルーチンについては，「6.1.2　基本操作」の「(2)　CBLJGETCLASS」を参照してください。

## (4)　クラスフィールドのアクセス

CBLJSETSTATICFIELD サービスルーチン（設定用）または CBLJGETSTATICFIELD サービスルーチン（取り出し用）を呼び出して，クラス参照とクラスフィールド名を使ってアクセスします。データの受け渡しの領域には，パラメタ型集団項目として定義したデータ項目を指定します。

クラスフィールドにアクセスする例を次に示します。

```
      *> Javaのクラスフィールド(SampleStaticField)に値を設定する

           MOVE  'HITACHI' TO WK-STRING.
           CALL 'CBLJXTOSTRING' USING CBLJENV
                   WK-STRING  WK-STRING-LEN  ARG-STR-AREA.

           CALL 'CBLJSETSTATICFIELD' USING CBLJENV
                   CLASSREF SampleStaticField  ARG-STRING.

      *> Javaのクラスフィールド(SampleStaticField)から値を取り出す

           CALL 'CBLJGETSTATICFIELD' USING CBLJENV
                   CLASSREF SampleStaticField
                   RTN-STRING.
```

CBLJSETSTATICFIELD サービスルーチンについては，「6.1.2　基本操作」の「(3) CBLJSETSTATICFIELD」を参照してください。

CBLJGETSTATICFIELD サービスルーチンについては，「6.1.2　基本操作」の「(4) CBLJGETSTATICFIELD」を参照してください。

## (5)　クラスメソッドの呼び出し

CBLJSTATICINVOKE サービスルーチンを呼び出して，クラス参照とクラスメソッド名を使って呼び出します。

メソッド呼び出しの引数には引数リストを，返却値にはパラメタ型集団項目として定義したデータ項目を指定します。

クラスメソッドを呼び出す例を次に示します。

```
      *> Javaのクラスメソッド(SampleStaticMethod)を呼び出す

           COMPUTE ARG-INT-AREA = 10.
           COMPUTE ARG-DBL-AREA = 3.14.
           COMPUTE ARGPTR(1) = FUNCTION ADDR( ARG-INT ).
           COMPUTE ARGPTR(2) = FUNCTION ADDR( ARG-DOUBLE ).
           COMPUTE ARGPTR(3) = ZERO.
           CALL 'CBLJSTATICINVOKE' USING CBLJENV
                   CLASSREF SampleStaticMethod ARG-LIST  RTN-VOID.
```

**注意事項**

- 引数なしのメソッドを呼び出す場合も引数なし用の引数リスト（NO-ARG）が必要です。
- void 型メソッドを呼び出す場合も void 型用パラメタ型集団項目（RTN-VOID）が必要です。

CBLJSTATICINVOKE サービスルーチンについては，「6.1.2　基本操作」の「(5) CBLJSTATICINVOKE」を参照してください。

## (6)　インスタンスの生成

CBLJNEW サービスルーチンを呼び出して，クラス参照を使ってコンストラクタでクラスインスタンスを生成し，オブジェクト参照を取得します。

コンストラクタの引数には引数リストを指定します。

インスタンスを生成する例を次に示します。

```
      *> Javaのインスタンスを生成する

           CALL 'CBLJNEW' USING CBLJENV CLASSREF NO-ARG  OBJREF.
```

**注意事項**

引数なしコンストラクタを呼び出す場合も引数なし用の引数リスト（NO-ARG）が必要です。

CBLJNEW サービスルーチンについては，「6.1.2　基本操作」の「(6)　CBLJNEW」を参照してください。

## (7)　インスタンスフィールドのアクセス

CBLJSETFIELD サービスルーチン（設定用）または CBLJGETFIELD サービスルーチン（取り出し用）で，オブジェクト参照およびインスタンスフィールド名を使ってアクセスします。

データの受け渡しの領域には，パラメタ型集団項目として定義したデータ項目を指定します。

インスタンスフィールドにアクセスする例を次に示します。

```
      *> Javaのインスタンスフィールド(SampleField)に値を設定する

           MOVE  100 TO ARG-INT-AREA.
           CALL 'CBLJSETFIELD' USING CBLJENV
                  OBJREF SampleField  ARG-INT.

      *> Javaのインスタンスフィールド(SampleField)から値を取り出す

           CALL 'CBLJGETFIELD' USING CBLJENV
                  OBJREF SampleField  RTN-INT.
```

CBLJSETFIELD サービスルーチンについては，「6.1.2　基本操作」の「(7)　CBLJSETFIELD」を参照してください。

## (8)　インスタンスメソッドの呼び出し

CBLJINVOKE サービスルーチンで，オブジェクト参照およびインスタンスメソッド名を使って呼び出します。メソッド呼び出しの引数には引数リストを，返却値はパラメタ型集団項目として定義したデータ項目を指定します。

インスタンスメソッドを呼び出す例を次に示します。

```
      *> Javaのインスタンスメソッド(SampleMethod)を呼び出す

           COMPUTE ARG-STR-AREA = RTN-STR-AREA.
           COMPUTE ARGPTR(1) = FUNCTION ADDR( ARG-STRING ).
           COMPUTE ARGPTR(2) = ZERO.
           CALL 'CBLJINVOKE' USING CBLJENV
                  OBJREF SampleMethod ARG-LIST  RTN-INT.
```

**注意事項**

- 引数なしのメソッドを呼び出す場合も引数なし用の引数リスト（NO-ARG）が必要です。
- void 型メソッドを呼び出す場合も void 型用パラメタ型集団項目（RTN-VOID）が必要です。

CBLJINVOKE サービスルーチンについては，「6.1.2　基本操作」の「(9)　CBLJINVOKE」を参照してください。

## (9)　インスタンスの解放

CBLJRELEASE サービスルーチンまたは CBLJSETNULL サービスルーチンを使ってインスタンスを解放します。

不要になったインスタンスは削除しないとメモリ圧迫の原因になります。

インスタンスを解放する例を次に示します。

例 1）

```
      *> Javaのインスタンスを解放する
           CALL 'CBLJRELEASE'  USING CBLJENV OBJREF.
```

例 2）

```
      *> Javaのインスタンスを解放する
           CALL 'CBLJSETNULL'  USING CBLJENV OBJREF.
```

CBLJRELEASE サービスルーチンについては，「6.1.2　基本操作」の「(10)　CBLJRELEASE」を参照してください。

CBLJSETNULL サービスルーチンについては，「6.1.4　オブジェクト操作」の「(6)　CBLJSETNULL」を参照してください。

## (10)　Java プログラム呼び出し機能の実行環境の終了処理

CBLJFINALIZE サービスルーチンを使って Java プログラム呼び出し機能の実行環境を削除します。

Java の実行環境は多くのシステムリソースを必要としますので，不要になったら削除する必要があります。

Java プログラム呼び出し機能の実行環境の終了例を次に示します。

```
      *> Javaの実行環境を削除する
           CALL 'CBLJFINALIZE' USING CBLJENV.
```

CBLJFINALIZE サービスルーチンについては，「6.1.2　基本操作」の「(11)　CBLJFINALIZE」を参照してください。

## 3.2.2　文字列操作のコーディング

COBOL プログラムと Java プログラムでは，文字列データとして内部的に処理する文字コードが異なります。また，Java プログラムの引数に直接 COBOL の英数字項目や日本語項目を指定することはできません。これらの文字列操作のために次の機能が用意されています。

- 英数字項目／日本語項目から Java に渡す String オブジェクト※を生成する機能
- Java から受け取った String オブジェクト※を英数字項目／日本語項目の値に変換する機能

注※

java.lang.String クラスのオブジェクトです。内部的な文字コードは UTF-16 です。

### (1)　Java プログラム呼び出し機能で使用する文字コード

Java プログラム呼び出し機能を使用する場合，COBOL プログラムで使用できる文字コードは，COBOL2002 で使用する文字コードと同じです。Unicode 機能を使用して COBOL が扱うデータを Unicode として扱うこともできます。取り扱う文字コードと Unicode 機能については，マニュアル「COBOL2002　ユーザーズガイド」の Unicode 機能の説明を参照してください。

Unicode 機能と同時に使用する場合，UCS-4 の範囲（UCS-2 の範囲を含む）の Unicode 文字を使用できます。

### (2)　英数字項目／日本語項目から Java に渡す String オブジェクトを生成する機能

次に示すサービスルーチンが用意されています。

- CBLJXTOSTRING サービスルーチン
  英数字項目の値から String オブジェクトを生成する。
- CBLJNTOSTRING サービスルーチン
  日本語項目の値から String オブジェクトを生成する。

英数字項目／日本語項目から String オブジェクトを生成する例を次に示します。

```
       01 X-ITEM     PIC X(256).
       01 X-ITEM-LEN PIC S9(9) USAGE COMP VALUE 256.
       01 N-ITEM     PIC N(256).
       01 N-ITEM-LEN PIC S9(9) USAGE COMP VALUE 256.
       01 X-STROBJ   USAGE POINTER.
       01 N-STROBJ   USAGE POINTER.
```

```
*> 英数字項目からStringオブジェクトを生成する。
     MOVE '日立製作所' TO X-ITEM.
     CALL 'CBLJXTOSTRING' USING CBLJENV
                                X-ITEM X-ITEM-LEN X-STROBJ.

*> 日本語項目からStringオブジェクトを生成する。
     MOVE N'日立製作所' TO N-ITEM.
     CALL 'CBLJNTOSTRING' USING CBLJENV
                                N-ITEM N-ITEM-LEN N-STROBJ.
```

CBLJXTOSTRING サービスルーチンについては，「6.1.5　文字列オブジェクト操作」の「(1) CBLJXTOSTRING」を参照してください。

CBLJNTOSTRING サービスルーチンについては，「6.1.5　文字列オブジェクト操作」の「(2) CBLJNTOSTRING」を参照してください。

## (3)　Java から受け取った String オブジェクトを英数字項目／日本語項目の値に変換する機能

次に示すサービスルーチンが用意されています。

- CBLJSTRINGTOX サービスルーチン

  String オブジェクトの文字列を COBOL で使用する文字コードに変換して，英数字項目に格納する。
- CBLJSTRINGTON サービスルーチン

  String オブジェクトの文字列を COBOL で使用する文字コードに変換して，日本語項目に格納する(Unicode 機能と同時に使用する場合は，データを変換しない)。

**注意事項**

CBLJSTRINGTON サービスルーチンを呼び出して String オブジェクトの文字列を変換するとき，変換後に 1 バイトになる文字が含まれないようにしてください。1 バイトになる文字が含まれている場合の動作は保証されません。

String オブジェクトを英数字項目／日本語項目の値に変換して格納する例を次に示します。

```
 01 X-STROBJ   USAGE POINTER.
 01 X-ITEM     PIC X(256).
 01 X-ITEM-LEN PIC S9(9) USAGE COMP VALUE 256.
 01 N-STROBJ   USAGE POINTER.
 01 N-ITEM     PIC N(256).
 01 N-ITEM-LEN PIC S9(9) USAGE COMP VALUE 256.

*> Stringオブジェクトの文字列を英数字項目に格納する。
     CALL 'CBLJSTRINGTOX' USING CBLJENV STROBJ X-ITEM X-ITEM-LEN.

*> Stringオブジェクトの文字列を日本語項目に格納する。
     CALL 'CBLJSTRINGTON' USING CBLJENV STROBJ N-ITEM N-ITEM-LEN.
```

String オブジェクトは，CBLJINVOKE サービスルーチンを使って String オブジェクトのメソッドを呼び出すことで，直接操作することもできます。

CBLJSTRINGTOX サービスルーチンについては，「6.1.5　文字列オブジェクト操作」の「(3) CBLJSTRINGTOX」を参照してください。

CBLJSTRINGTON サービスルーチンについては，「6.1.5　文字列オブジェクト操作」の「(4) CBLJSTRINGTON」を参照してください。

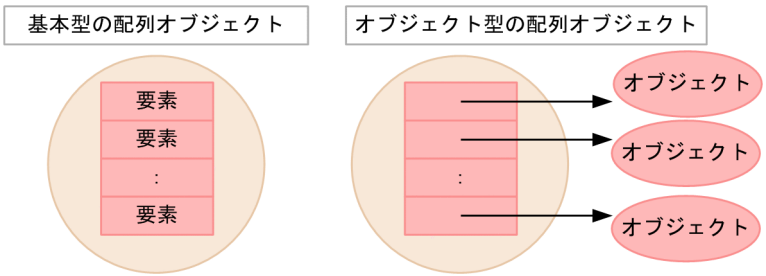
## 3.2.3　配列操作のコーディング

Java コンポーネントの配列オブジェクトを操作する手順を説明します。

### (1)　配列オブジェクトの構成と生成のコーディング

Java の配列オブジェクトの持つ配列は 1 次元配列だけです。多次元配列は，オブジェクト型の配列オブジェクトと，基本型またはオブジェクト型の配列オブジェクトとの組み合わせで表現されます。多次元配列については，「(4)　多次元配列を構成する配列オブジェクトの生成と操作のコーディング」を参照してください。

1 次元配列について次に示します。

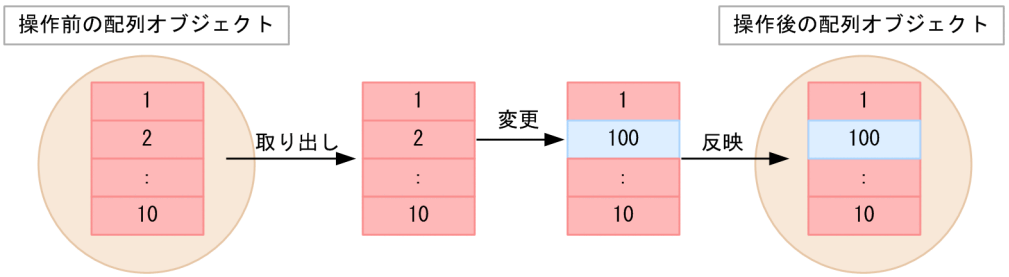



配列オブジェクトを生成する例を次に示します。

```
       01 INT-ARRAY-TYPE  PIC X(256) VALUE '[I'.
       01 INT-ARRAY-LEN   PIC S9(9) USAGE COMP VALUE 10.
       01 INT-ARRAYOBJ    USAGE POINTER.
       01 STR-ARRAY-TYPE  PIC X(256) VALUE '[Ljava/lang/String;'.
       01 STR-ARRAY-LEN   PIC S9(9) USAGE COMP VALUE 20.
       01 STR-ARRAYOBJ    USAGE POINTER.

      *> int型の配列オブジェクトを生成する。
           CALL 'CBLJNEWARRAY' USING CBLJENV INT-ARRAY-TYPE
                                             INT-ARRAY-LEN
                                             INT-ARRAYOBJ.

      *> String型の配列オブジェクトを生成する。
           CALL 'CBLJNEWARRAY' USING CBLJENV STR-ARRAY-TYPE
                                             STR-ARRAY-LEN
                                             STR-ARRAYOBJ.
```

## (2)　基本型の配列オブジェクト操作のコーディング

基本型の配列オブジェクトを操作する場合，いったん配列を配列オブジェクトから取り出して操作し，操作終了後に配列オブジェクトに戻す手順になります。




配列オブジェクトから配列を出し入れするために，CBLJGETARRAYADDR サービスルーチンとCBLJRELEASEARRAY サービスルーチンが用意されています。

基本型の配列オブジェクトを操作する例を次に示します。

```
       01 ARRAYOBJ USAGE POINTER.
       01 ARRAYLEN PIC S9(9)  USAGE COMP.
       01 ARRAYADR USAGE POINTER.
       01 INTARRAY ADDRESSED BY P.
         02 ELEMENTS PIC S9(9) USAGE COMP
                     OCCURS m TIMES DEPENDING ON ARRAYLEN.

      *> 配列オブジェクトから配列アドレスを取り出し，
      *> 反復項目にマッピングする
           CALL 'CBLJGETARRAYADDR' USING CBLJENV ARRAYOBJ ARRAYADR.
           CALL 'CBLJARRAYLENGTH'  USING CBLJENV ARRAYOBJ ARRAYLEN.
           COMPUTE P = ARRAYADR.
      *> 配列の3番目の要素の参照
           DISPLAY ELEMENTS(3).
      *> 配列の3番目の要素の更新
           MOVE 100 TO ELEMENTS(3).
      *> 配列要素の配列オブジェクトへの書き戻し
           CALL 'CBLJRELEASEARRAY' USING CBLJENV ARRAYOBJ ARRAYADR.
```

反復回数（*m*）には想定される配列要素数の上限を指定してください。

### 注意事項

- INTARRAY による集団項目操作はしないでください。OCCURS 句で指定した反復回数が配列オブジェクトの要素数を超える場合，不当領域参照の要因になります。
- Java の配列オブジェクトでは要素の添字が 0 から始まりますが，COBOL の繰り返し項目では要素の添字は 1 から指定します。添字 n で参照する基本型配列オブジェクトの要素を，COBOL の繰り返し項目で参照する場合は添字 n ＋ 1 で参照します。

CBLJGETARRAYADDR サービスルーチンについては，「6.1.6　配列オブジェクト操作」の「(5) CBLJGETARRAYADDR」を参照してください。

CBLJRELEASEARRAY サービスルーチンについては，「6.1.6　配列オブジェクト操作」の「(6) CBLJRELEASEARRAY」を参照してください。

## (3)　オブジェクト型の配列オブジェクト操作のコーディング

オブジェクト型の配列オブジェクトの場合，CBLJSETOBJARRAY サービスルーチン（格納用）と CBLJGETOBJARRAY サービスルーチン（取出用）を使って要素のオブジェクトの出し入れをします。

オブジェクト型の配列オブジェクトを操作する例を次に示します。

```
      *> 配列オブジェクトのJavaオブジェクト参照
       01 ARRAYOBJ USAGE POINTER.
      *> 要素のJavaオブジェクト参照
       01 ELEMENT  USAGE POINTER.
      *> 要素の添字
       01 IDX      PIC S9(9) USAGE COMP.

      *> オブジェクト型の配列オブジェクトから3番目の要素（オブジェクト）を取り出す。
           MOVE 2 TO IDX.
           CALL 'CBLJGETOBJARRAY' USING CBLJENV ARRAYOBJ IDX ELEMENT.

      *> オブジェクト型の配列オブジェクトに3番目の要素（オブジェクト）を格納する。
           MOVE 2 TO IDX.
           CALL 'CBLJSETOBJARRAY' USING CBLJENV ARRAYOBJ IDX ELEMENT.
```

**注意事項**

Java の配列オブジェクトでは要素の添字が 0 から始まります。

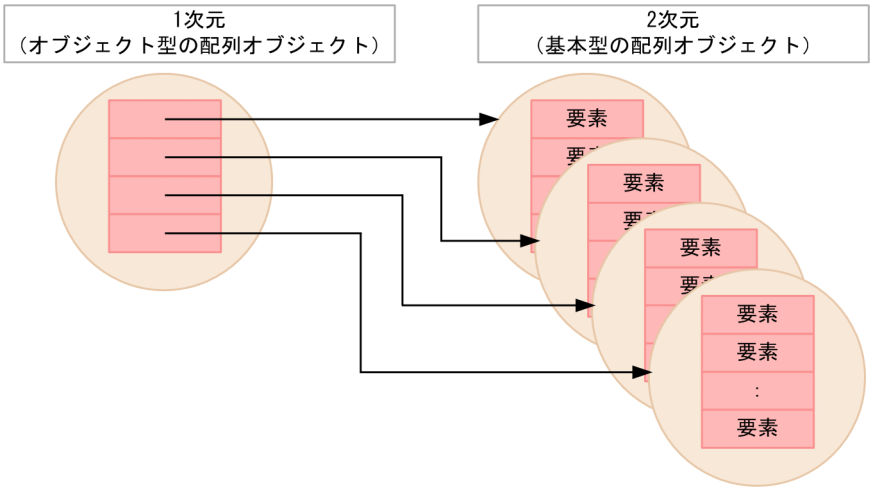
CBLJSETOBJARRAY サービスルーチンについては，「6.1.6　配列オブジェクト操作」の「(3) CBLJSETOBJARRAY」を参照してください。

CBLJGETOBJARRAY サービスルーチンについては，「6.1.6　配列オブジェクト操作」の「(4) CBLJGETOBJARRAY」を参照してください。

## (4)　多次元配列を構成する配列オブジェクトの生成と操作のコーディング

多次元配列は，オブジェクト型の配列オブジェクトと，基本型またはオブジェクト型の配列オブジェクト（最下位次元だけ）との組み合わせで表現されます。

基本型の 2 次元配列について次に示します。




int 型の 3×4 の 2 次元配列を生成する例を次に示します。

```
       01 D1-ARRAY-TYPE  PIC X(256) VALUE '[[I'.
       01 D1-ARRAYOBJ    USAGE POINTER.
       01 D1-ARRAYLEN    PIC S9(9)  USAGE COMP VALUE 3.
       01 D2-ARRAY-TYPE  PIC X(256) VALUE '[I'.
       01 D2-ARRAYOBJ    USAGE POINTER.
       01 D2-ARRAYLEN    PIC S9(9)  USAGE COMP VALUE 4.
       01 I              PIC S9(9)  USAGE COMP.
      *> 1次元の配列オブジェクトを生成する。
           CALL 'CBLJNEWARRAY' USING CBLJENV D1-ARRAY-TYPE D1-ARRAYLEN
                                     D1-ARRAYOBJ.
           PERFORM VARYING I FROM 0 BY 1 UNTIL I > D1-ARRAYLEN - 1
      *> 2次元の配列オブジェクトを生成する。
             CALL 'CBLJNEWARRAY'    USING CBLJENV D2-ARRAY-TYPE
                                          D2-ARRAYLEN
                                          D2-ARRAYOBJ
             CALL 'CBLJSETOBJARRAY' USING CBLJENV D1-ARRAYOBJ
                                          I
                                          D2-ARRAYOBJ
           END-PERFORM.
```

int 型の 2×3×4 の 3 次元配列の要素の値を合計する例を次に示します。

```
      *> 1次元目の配列オブジェクト参照用ワーク
       01 D1-ARRAYOBJ    USAGE POINTER.
       01 LEN-1          PIC S9(9) USAGE COMP.
      *> 2次元目の配列オブジェクト参照用ワーク
       01 D2-ARRAYOBJ    USAGE POINTER.
       01 LEN-2          PIC S9(9) USAGE COMP.
      *> 3次元目の配列オブジェクト参照用ワーク
       01 D3-ARRAYOBJ    USAGE POINTER.
       01 LEN-3          PIC S9(9) USAGE COMP.
      *> 配列オブジェクト参照用ワーク
       01 ARRAYADDR      USAGE POINTER.
       01 ARRAYLEN       PIC S9(9)  USAGE COMP.
       01 INTARRAY       ADDRESSED BY P.
         02 ELEMENTS     PIC S9(9) USAGE COMP
                         OCCURS 24 TIMES DEPENDING ON ARRAYLEN.
```

```
 01 I              PIC S9(9) USAGE COMP.
 01 J              PIC S9(9) USAGE COMP.
 01 K              PIC S9(9) USAGE COMP.
 01 TOTAL          PIC S9(9) USAGE COMP.

     MOVE 0 TO TOTAL.
     CALL 'CBLJARRAYLENGTH'  USING CBLJENV D1-ARRAYOBJ LEN-1

     PERFORM VARYING I FROM 0 BY 1 UNTIL I > LEN-1 - 1

*> 1次元の配列オブジェクトから2次元の配列オブジェクトを取り出す。
       CALL 'CBLJGETOBJARRAY'  USING CBLJENV
                                     D1-ARRAYOBJ I D2-ARRAYOBJ
       CALL 'CBLJARRAYLENGTH'  USING CBLJENV D2-ARRAYOBJ LEN-2

       PERFORM VARYING J FROM 0 BY 1 UNTIL J > LEN-2 - 1

*> 2次元の配列オブジェクトから3次元の配列オブジェクトを取り出す。
         CALL 'CBLJGETOBJARRAY'  USING CBLJENV
                                       D2-ARRAYOBJ J D3-ARRAYOBJ
         CALL 'CBLJARRAYLENGTH'  USING CBLJENV D3-ARRAYOBJ LEN-3

*> 3次元の配列オブジェクトから基本型の配列を取り出す。
         CALL 'CBLJGETARRAYADDR' USING CBLJENV
                                       D3-ARRAYOBJ ARRAYADDR
*> 基本型の配列をCOBOLの反復項目にマッピングする。
         COMPUTE P = ARRAYADDR

*> 基本型の配列の要素の値を加算する。（COBOLでの添字は1から）
         PERFORM VARYING K FROM 1 BY 1 UNTIL K > LEN-3
           COMPUTE TOTAL = TOTAL + ELEMENTS(K)
         END-PERFORM

*> 基本型の配列をオブジェクト参照に反映させて解放する。
         CALL 'CBLJRELEASEARRAY' USING CBLJENV
                                       D3-ARRAYOBJ ARRAYADDR

*> 取り出した3次元の配列オブジェクトのオブジェクト参照を解放する。
         CALL 'CBLJRELEASE'      USING CBLJENV D3-ARRAYOBJ

       END-PERFORM

*> 取り出した2次元の配列オブジェクトのオブジェクト参照を解放する。
       CALL 'CBLJRELEASE'      USING CBLJENV D2-ARRAYOBJ

     END-PERFORM.

*> 結果を表示する。
     DISPLAY 'TOTAL=' TOTAL.
```

### 注意事項

- Java の配列オブジェクトでは要素の添字が 0 から始まりますが，COBOL の繰り返し項目では要素の添字は 1 から指定します。添字 n で参照する基本型配列オブジェクトの要素を，COBOL の繰り返し項目で参照する場合は添字 n ＋ 1 で参照します。

- 配列オブジェクトから取り出した配列オブジェクトのオブジェクト参照は必ず解放する必要があります。ただし，CBLJGETOBJARRAY サービスルーチン呼び出しの受け取り項目に指定されている場合，格納されていた配列オブジェクトのオブジェクト参照は自動的に解放されます。

## 3.2.4　例外処理のコーディング

ここでは，サービスルーチンで検知する Java 例外とサービスルーチンで検知しない Java 例外について説明します。

### (1)　サービスルーチンで検知する Java 例外の処理

次のサービスルーチンでは，Java プログラムから例外がスローされたことで，サービスルーチンの呼び出しが失敗で終了することがあります。

- CBLJSTATICINVOKE サービスルーチン
- CBLJNEW サービスルーチン
- CBLJINVOKE サービスルーチン

Java プログラムから例外がスローされた場合，CBLJENV 集団項目の CBLJEXCEPTION 項目にスローされた例外オブジェクトを設定し，RETURN-CODE 特殊レジスタに 1 を返します。

RETURN-CODE 特殊レジスタが 0 でない場合は，CBLJEXCEPTION 項目の例外オブジェクトを使って例外処理をしてください。

インスタンスメソッドを呼び出す例について説明します。

```
           COMPUTE ARGPTR(1) = FUNCTION ADDR( ARG-STRING ).
           COMPUTE ARGPTR(2) = ZERO.

           CALL 'CBLJINVOKE' USING CBLJENV
                   OBJREF SampleMethod ARG-LIST  RTN-INT.

           IF RETURN-CODE NOT = 0

       *>      CBLJEXCEPTIONを使った例外処理をここに記述する。

               CALL 'CBLJFINALIZE' USING CBLJENV                    *> ※1
               MOVE  nnnn TO ABN-CODE
               CALL 'CBLABN' USING ABN-CODE                         *> ※2
           END-IF.
```

注※1

Java プログラム呼び出し機能の実行環境の終了処理の呼び出しを示します。

注※2

COBOL プログラムを異常終了（終了コードは*nnnn*）させるサービスルーチンの呼び出しを示します。

CBLJEXCEPTION を使った例外処理では，CBLJEXCEPTION 項目を使って例外オブジェクトの種別判定や例外オブジェクトのメソッド呼び出しまたはフィールドアクセスを実行します。

この例では※1，※2 の呼び出しによって COBOL プログラムを終了していますが，続行できる例外ならば続行させてもかまいません。

なお，例外処理中にメソッドを呼び出す場合は，CBLJCOPY サービスルーチンを使って，CBLJEXCEPTION 項目の例外オブジェクトのオブジェクト参照を退避してください。メソッド呼び出しのサービスルーチンは，CBLJEXCEPTION 項目を更新します。

```
       01 EXP-OBJ           USAGE POINTER.
       01 EXP-CLS           USAGE POINTER VALUE NULL.
       01 EXP-TOSTRING      PIC X DYNAMIC C-STRING VALUE 'toString'.
       01 EXP-STRING.
         02  EXP-STR-TYPE    PIC X(256) VALUE 'Ljava/lang/String;'.
         02  EXP-STR-AREA    USAGE POINTER.

      *>      例外オブジェクトのメソッドtoString()を呼び出して
      *>      情報を取得する
              CALL 'CBLJCOPY'   USING CBLJENV
                                      CBLJEXCEPTION EXP-OBJ EXP-CLS.
              CALL 'CBLJINVOKE' USING CBLJENV EXP-OBJ
                                      EXP-TOSTRING NO-ARG EXP-STRING.
```

CBLJEXCEPTION 項目を使って発生した例外オブジェクトの種別を判定する例を次に示します。

- 発生した例外が，ある特定のクラスまたはそのサブクラスであるかの判定

```
       01  MyException       PIC X DYNAMIC C-STRING
                             VALUE 'MyException'.
       01 MY-EXCEPTION       USAGE POINTER.

          CALL 'CBLJGETCLASS'   USING CBLJENV MyException   MY-EXCEPTION
          CALL 'CBLJINSTANCEOF' USING CBLJENV CBLJEXCEPTION MY-EXCEPTION
          IF RETURN-CODE NOT = 0

        *>        MyException例外の処理をここに記述する。

          END-IF
```

- 発生した例外が，ある特定のクラスであるかの判定

```
       01  EXCEPTNAME-LEN PIC 9(9) USAGE COMP VALUE 256.
       01  EXCEPTNAME     PIC X(256).

           CALL 'CBLJCLASSNAME' USING CBLJENV CBLJEXCEPTION
                                      EXCEPTNAME EXCEPTNAME-LEN
           EVALUATE EXCEPTNAME
             WHEN '例外クラス名1'

        *>          例外クラス名1の処理をここに記述する。
```

```
            WHEN ’例外クラス名2’
*>              例外クラス名2の処理をここに記述する。
            WHEN ’例外クラス名3’
*>              例外クラス名3の処理をここに記述する。
            WHEN OTHER
*>              その他の例外の処理をここに記述する。
        END-EVALUATE
```

**注意事項**

CBLJCLASSNAME サービスルーチンが返す名前は，Java プログラム呼び出し機能のクラス名（パッケージとクラス名をスラントで区切る）ではなく，Java のクラス名（パッケージとクラス名をピリオドで区切る）です。

例）

```
java.lang.Exception
```

### (2)　サービスルーチンで検知しない Java 例外

サービスルーチンで検知しない Java 例外の場合，Java プログラムで例外が発生したとき，COBOL プログラムのスレッドまたはプロセスが終了します。

この場合の対処については，「5.　プログラムのデバッグとトラブルシュート」を参照してください。

## 3.2.5　Java オブジェクト参照の使用ガイドライン

Java プログラム呼び出し機能のサービスルーチンから取得した Java オブジェクト参照は，必ず解放する必要があります。

解放しない場合，Java VM のガーベジコレクションでオブジェクトを解放できないため，メモリリークの原因になります。

COBOL プログラムで Java オブジェクト参照を扱う場合は，次のガイドラインに従ってください。

1. Java オブジェクト参照用の領域はポインタ項目(USAGE POINTER)として記述します。
   パラメタ型集団項目の中に記述する場合を除いて，できるだけ 01 レベルの項目（01 または 77）としてください。
   従属項目として記述された場合，上位の集団項目の操作でポインタ項目に不当な値が設定され，不正な動作となることがあります。

2. 一つの Java オブジェクト参照は必ず一つのポインタ項目に格納してください。

   Java オブジェクト参照を別のポインタ項目に転記したい場合は，COMPUTE 文や SET 文を使って転記しないで，必ず CBLJCOPY サービスルーチンで複製を作成して転記してください。そして，不要になったら必ず両方の Java オブジェクト参照を解放してください。

   COMPUTE 文や SET 文を使って転記すると，多重解放の原因になります。

3. 不要になった Java オブジェクト参照は，CBLJRELEASE サービスルーチンまたは CBLJSETNULL サービスルーチンで必ず解放してください。

   なお，次の Java オブジェクト参照は，これらのサービスルーチンを使用しなくても自動的に解放されます。ただし，解放漏れを防止するために，これらのサービスルーチンで明示的に解放することをお勧めします。

   - 次のサービスルーチン呼び出しの受け取りのポインタ項目に格納されている Java オブジェクト参照
     - CBLJGETSTATICFIELD サービスルーチン
     - CBLJSTATICINVOKE サービスルーチン
     - CBLJNEW サービスルーチン
     - CBLJGETFIELD サービスルーチン
     - CBLJINVOKE サービスルーチン
     - CBLJXTOSTRING サービスルーチン
     - CBLJNTOSTRING サービスルーチン
     - CBLJNEWARRAY サービスルーチン
     - CBLJGETOBJARRAY サービスルーチン
   - CBLJFINALIZE サービスルーチンを呼び出した場合の未解放のすべての Java オブジェクト参照

4. 手続き部で Java オブジェクト参照用のポインタ項目を NULL で初期化する場合は，SET 文または INITIALIZE 文を使用しないで，必ず CBLJSETNULL サービスルーチンを使用してください。

   SET 文または INITIALIZE 文を使用した場合，格納されていた Java オブジェクト参照を解放できなくなります。

   CBLJSETNULL サービスルーチンは，ポインタ項目に Java オブジェクト参照が入っていれば，解放してから NULL を設定します。

5. Java オブジェクト参照用のポインタ項目を引数に指定する場合は，BY CONTENT や BY VALUE ではなく，必ず BY REFERENCE にしてください。

6. Java オブジェクト参照用のポインタ項目は，RETURNING 指定には指定しないでください。

7. Java オブジェクト参照は，C 言語などの他言語プログラムには渡さないでください。

8. Java オブジェクト参照用のポインタ項目を，局所場所節または初期化プログラムや CANCEL 文でキャンセルされるプログラムの作業場所節に指定した場合は，そのプログラムがリターンするまでに必ず解放してください。

9. CBLJFINALIZE サービスルーチンを呼び出すと，未解放のすべての Java オブジェクト参照は解放されます。

CBLJFINALIZE サービスルーチンを呼び出したあとは，Java オブジェクト参照用のポインタ項目のアドレスは解放された Java オブジェクト参照の領域を指しています。このため，これを使ってサービスルーチンを呼び出さないでください。アクセス違反や実行時エラーとなります。

### 3.2.6　コーディング時の注意事項

コーディング時の注意事項を次に示します。

- 引数なしのメソッド，コンストラクタ，および void 型のメソッドを呼び出す場合は，次の項目を必ず指定してください。指定がない場合は，アプリケーションエラー（アクセス違反）の原因となります。
  - 引数なしを示す引数リスト(NO-ARG)

    NO-ARG の使用例については，「3.2.1　Java クラス操作のコーディング」の「(6)　インスタンスの生成」を参照してください。
  - void 型を示すパラメタ型集団項目(RTN-VOID)

    RTN-VOID の使用例については，「3.2.1　Java クラス操作のコーディング」の「(5)　クラスメソッドの呼び出し」を参照してください。
- Java オブジェクト参照の取り扱いは，Java オブジェクト参照の使用ガイドラインに従ってください。ガイドラインに従わない場合は，メモリリークの原因になります。Java オブジェクト参照の使用ガイドラインについては，「3.2.5　Java オブジェクト参照の使用ガイドライン」を参照してください。
- 次のようなケースは，それぞれの Java コンポーネントを個別に呼び出すと，Java プログラム呼び出し機能が使用する JNI 呼び出しのオーバヘッドが影響するため，Java から呼び出した場合に比べて実行性能が大幅に遅くなるおそれがあります。ロジックを Java のクラスメソッドとして実装し，COBOL からはそのクラスメソッドを呼び出すだけにすることをお勧めします。
  - オブジェクト間のリンクが複雑なオブジェクトを扱うロジック
  - 多くのメソッド呼び出しやフィールド呼び出しを必要とするロジックで，かつクリティカルな性能が要求されているケース

# 4 プログラムのコンパイルと実行

この章では，Java プログラム呼び出し機能を使用した COBOL の UAP のコンパイルと実行について説明します。

## 4.1　プログラムのコンパイル

Java プログラムを使用する COBOL プログラムのコンパイルでは，Java プログラム呼び出し機能の実行時ライブラリ（インポートライブラリ cbl2kjrt.lib）を指定してリンクさせます。

実行時ライブラリの指定例を次に示します。

```
ccbl2002 -Main,System COBOLプログラム.cbl cbl2kjrt.lib
```

COBOL2002 のコンパイルの方法や注意事項については，マニュアル「COBOL2002　ユーザーズガイド」を参照してください。

### 4.1.1　COBOL2002 のコンパイラオプションを指定するときの注意事項

Java プログラム呼び出し機能を使用する COBOL プログラムのコンパイル時に，コンパイラオプションを指定する場合の注意事項について説明します。

COBOL2002 のコンパイラオプションについては，マニュアル「COBOL2002　ユーザーズガイド」を参照してください。

#### (1)　-BigEndian コンパイラオプション（バイナリデータをビッグエンディアン形式で取り扱う）

-BigEndian コンパイラオプション※を指定してコンパイルすると，COBOL プログラムの 2 進形式の数字項目および内部浮動小数点項目をビッグエンディアン形式で取り扱うことができます。

注※

-BigEndian コンパイラオプションは，システムのエンディアンがリトルエンディアン形式の場合に有効です。

Java プログラム呼び出し機能を使用するプログラムで，-BigEndian コンパイラオプションを指定するときの注意事項を次に示します。

サービスルーチンの引数

- -BigEndian,Bin コンパイラオプションを指定したとき，サービスルーチンの引数に直接指定する Java のデータと対応しない 4 バイト 2 進形式の数字項目の定義では，USAGE 句に COMP ではなく，COMP-5 を指定してください。

  対象の数字項目については，各サービスルーチンの注意事項を確認してください。

  なお，USAGE 句に COMP-5 を指定する場合は，-Comp5 コンパイラオプションを指定する必要があります。
- Java の型が char 型のとき，対応する COBOL データ項目を日本語項目 1 文字（PIC N）として定義した場合も，データの内容はビッグエンディアン形式となります。

実行時に指定する環境変数

-BigEndian コンパイラオプションの指定に従って，環境変数 CBLJRTBIGENDIAN を指定してください。指定しない場合，システムのエンディアンの形式で扱われます。

環境変数 CBLJRTBIGENDIAN については，「6.2.2　実行時環境変数の詳細」の「(1) CBLJRTBIGENDIAN」を参照してください。

## (2)　-DynamicLink コンパイラオプション（動的なリンクを使用する）

-DynamicLink,Call コンパイラオプションを指定してコンパイルすることで，定数指定の CALL 文の呼び出し先を動的にリンクできます。

Java プログラム呼び出し機能のサービスルーチンを呼び出す COBOL プログラムに-DynamicLink,Call コンパイラオプションを指定してコンパイルしたとき，Java プログラム呼び出し機能の実行時ライブラリを，動的にロードするライブラリの検索対象に指定する必要があります。

COBOL2002 の実行時環境変数の指定例を次に示します。

```
CBLLDLL=cbl2kjrt.dll
```

## (3)　-JPN コンパイラオプション（日本語項目の扱いを指定する）

-JPN,Alnum または-JPN,V3JPN コンパイラオプションを指定してコンパイルすると，COBOL プログラムの日本語項目の空白の扱いを指定できます。

Java プログラム呼び出し機能の文字列操作では，英数字項目と日本語項目とでは，残りの領域を補う空白の扱いが異なります。Java プログラム呼び出し機能での空白の扱いを次の表に示します。

### 表 4–1　Java プログラム呼び出し機能での空白の扱い

| サービスルーチンの引数 | 領域を補う空白の扱い |
|---|---|
| 英数字項目として扱う場合 | 半角空白(X'20') |
| 日本語項目として扱う場合 | 全角空白(X'8140') |

-JPN コンパイラオプションを指定する場合の注意事項を次に示します。

日本語項目から Java に渡す String オブジェクトを生成する場合

- -JPN,Alnum または-JPN,V3JPN コンパイラオプションを指定するときは，CBLJXTOSTRING サービスルーチンを呼び出して String オブジェクトを生成してください。このとき，長さを指定する引数には，日本語項目の文字数ではなく，英数字項目として扱ったときの長さ（バイト数）を指定してください。
- -JPN,V3JPNSpace コンパイラオプションを指定するときは，CBLJNTOSTRING サービスルーチンを呼び出して String オブジェクトを生成してください。

String オブジェクトを日本語項目の値に変換する場合

- -JPN,Alnum または-JPN,V3JPN コンパイラオプションを指定するときは，CBLJSTRINGTOX サービスルーチンを呼び出して値を変換してください。このとき，長さを指定する引数には，日本語項目の文字数ではなく，英数字項目として扱ったときの長さ（バイト数）を指定してください。
- -JPN,V3JPNSpace コンパイラオプションを指定するときは，CBLJSTRINGTON サービスルーチンを呼び出して値を変換してください。

## 4.2　プログラムの実行

Java プログラム，および COBOL プログラムのための環境設定をしたあとに，プログラムを実行します。

COBOL2002 の実行時の環境設定については，マニュアル「COBOL2002　ユーザーズガイド」を参照してください。

Java プログラム呼び出し機能の環境変数については，「6.2　実行時環境変数」を参照してください。

### 4.2.1　プログラム実行時の注意事項

Java プログラム呼び出し機能のサービスルーチンを呼び出す COBOL プログラムのコンパイルで-DynamicLink,Call コンパイラオプションを指定する場合，動的なリンクの設定が必要です。

動的なリンクの設定については，「4.1.1　COBOL2002 のコンパイラオプションを指定するときの注意事項」の「(2)　-DynamicLink コンパイラオプション（動的なリンクを使用する）」を参照してください。

### 4.2.2　Java VM 起動オプション

Java プログラム呼び出し機能を使用する場合，プロセス単位に Java VM を起動する必要があります。ただし，CBLJINITIALIZE サービスルーチンが呼び出される前に Java VM を起動していないとき，Java プログラム呼び出し機能が Java VM を起動します。ここでは，Java プログラム呼び出し機能が Java VM を起動する場合の起動オプションについて説明します。

なお，Java プログラム呼び出し機能が Java VM を起動する場合，Java VM 起動オプション情報を収集します。Java VM 起動オプション情報については，「5.4　Java VM 起動オプション情報の収集」を参照してください。

#### (1)　プロセス単位の起動オプション

プロセス単位の起動オプションは，環境変数 CBLJRTVMOPTIONS または CBLJENV 集団項目の CBLJVMOPTIONS 項目で指定します。プロセス単位の起動オプションは，Java プログラム呼び出し機能が Java VM を起動するときだけ有効です。

プロセス単位の起動オプションの優先順位を次に示します。

1. 環境変数 CBLJRTVMOPTIONS に指定したファイルに登録した内容
2. CBLJINITIALIZE サービスルーチンの引数に指定した CBLJENV 集団項目の CBLJVMOPTIONS 項目の内容

環境変数 CBLJRTVMOPTIONS の指定方法については，「6.2.2　実行時環境変数の詳細」の「(6) CBLJRTVMOPTIONS」を参照してください。

CBLJENV 集団項目の CBLJVMOPTIONS 項目の指定方法については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。

## (2)　デフォルトの起動オプション

Java プログラム呼び出し機能が Java VM を起動するとき，プロセス単位の起動オプションに加えて，調査用資料の採取のための起動オプションを指定します。これらのオプションは，主に Java VM での障害発生時の調査に役立つ資料を採取するために指定します。

デフォルトの起動オプションを次に示します。

調査用の資料採取と出力先を指定する起動オプション

```
-XX:-HitachiThreadDumpToStdout
-XX:+HitachiOutputMilliTime
-XX:+HitachiVerboseGC
-XX:+HitachiOutOfMemoryStackTrace
-XX:+HitachiJavaClassLibTrace
-XX:+HitachiLocalsInStackTrace
-XX:+HitachiLocalsSimpleFormat
-XX:+HitachiTrueTypeInLocals

-XX:HitachiJavaLog:<出力先フォルダ>/CBLJRTJAVALOG
-XX:HitachiJavaLogFileSize=256k
-XX:HitachiJavaLogNumberOfFile=4
```

<出力先フォルダ>は次の優先順位で決定します。

1. 環境変数 TEMP に指定したフォルダ
2. 環境変数 TMP に指定したフォルダ
3. カレントフォルダ

その他の目的で指定する起動オプション

```
-XX:+HitachiOutOfMemoryAbort
```

このオプションを指定すると，Java が OutOfMemory 例外を検知したときに，ダンプを出力して強制終了します。メモリ不足のままプロセスが動作し続けることを防止します。

デフォルトの起動オプションは，環境変数 CBLJRTVMDEFAULTOPTIONS で指定を変更できます。

環境変数 CBLJRTVMDEFAULTOPTIONS の指定方法については，「6.2.2　実行時環境変数の詳細」の「(5)　CBLJRTVMDEFAULTOPTIONS」を参照してください。

### 注意事項

- Java プログラム呼び出し機能のデフォルトの起動オプションは，Cosminexus 製品に含まれる日立製 Java VM 固有の拡張オプションです。オプションについては，Cosminexus 製品のマニュアルを参照してください。

- 日立製以外の Java VM を使用する場合は，環境変数 CBLJRTVMDEFAULTOPTIONS を使用してデフォルトの起動オプションを取り消してください。このとき，障害発生時の調査のために，使用する Java VM が提供する詳細情報取得用のオプションを指定しておくことをお勧めします。
- Cosminexus 環境下のアプリケーションとは別のプロセスで日立製 Java VM を使用する場合，環境変数 CBLJRTVMDEFAULTOPTIONS を使用して指定を変更するとき，調査用の資料採取と出力先を指定する起動オプションを指定することをお勧めします。この起動オプションを指定しない場合は，Java VM での障害発生時に調査が困難になるときがあります。

# 5 プログラムのデバッグとトラブルシュート

この章では，Java プログラムを呼び出す COBOL の UAP のデバッグとトラブルシュートについて説明します。

## 5.1　Java プログラム呼び出し機能のデバッグとトラブルシュートの概要

Java プログラム呼び出し機能のサービスルーチンの引数が不正（引数不足や構造不正）の場合，アプリケーションエラー（アクセス違反）となることがあります。これらが発生した場合は，引数が正しいかを確認してください。

また，プログラムの実行中に次に示す要因で Java プログラム呼び出し機能の実行が失敗した場合，実行時エラーが出力され，スレッドまたはプロセスが終了します。

- Java VM や JNI の環境の起動に失敗した。
- OS，Java VM または JNI の環境がリソース不足などで不安定な状態になっている。
- サービスルーチンに不正な引数が渡された。
- Java VM や JNI で例外が発生した。

出力される COBOL 実行時メッセージには，Java プログラム呼び出し機能が管理する詳細メッセージ番号と詳細メッセージが出力されます。これらを基に対処することができます。

実行時メッセージの出力例を次に示します。

```
KCCC7904R-S
Javaプログラム呼び出し機能で実行時エラーが発生しました。
プログラム名＝MAIN
行番号／欄＝012000/12
エラー情報＝[0010] Java VMが起動されていません。（CBLJGETSTATICFIELD）
```

出力されるメッセージ，要因および対処については，「付録 B　Java プログラム呼び出し機能の詳細メッセージ」を参照してください。

Java プログラム呼び出し機能のデバッグとトラブルシュートのために，次の機能を使用できます。

- デバッグ情報出力機能

  Java プログラム呼び出し機能を使用した COBOL プログラムの実行時のデバッグ情報を出力します。
- 実行時エラー情報出力機能

  Java プログラム呼び出し機能で実行時エラーが発生したとき，デバッグ情報出力機能を使用していなくても，エラーの詳細情報を取得できます。

  Cosminexus 環境下のアプリケーションとは別のプロセスで Java プログラム呼び出し機能を使用する場合は，Java VM での障害調査用の資料を採取する設定にすることをお勧めします。Java VM 起動オプションについては，「4.2.2　Java VM 起動オプション」の「(2)　デフォルトの起動オプション」を参照してください。
- Java VM 起動オプション情報収集

  Java VM を起動するときに指定した起動オプションの情報を収集します。

COBOL プログラムのデバッグについては，マニュアル「COBOL2002　ユーザーズガイド」を参照してください。

## 5.2　デバッグ情報の出力

Java プログラム呼び出し機能では，Java プログラム呼び出し機能を使用した COBOL プログラムのテスト時やエラー発生時に，指定したファイル（デバッグ情報ファイル）に調査のためのデバッグ情報を出力できます。

出力されたデバッグ情報を確認することで，Java プログラム呼び出し機能のサービスルーチンの呼び出し順序，引数の内容やエラーの詳細情報などを確認できます。

デバッグ情報ファイルに出力される情報は次のとおりです。

- Java プログラム呼び出し機能のサービスルーチンで使用する情報
- オブジェクト解放漏れやエラー発生時の異常状態の情報

この節で説明する CBLJENV 集団項目などの Java プログラム呼び出し機能で使用するデータ項目については，「6.1.1　サービスルーチンで使用する引数」を参照してください。

### 5.2.1　デバッグ情報出力の指定方法

デバッグ情報を出力するには，Java プログラム呼び出し機能を使用する COBOL プログラムの実行時に，次に示す環境変数を指定します。

- CBLJRTDUMP：デバッグ情報ファイルのパス名を指定します。
- CBLJRTDUMP_MAXSIZE：デバッグ情報ファイルの最大サイズを指定します。

環境変数 CBLJRTDUMP については，「6.2.2　実行時環境変数の詳細」の「(2)　CBLJRTDUMP」を参照してください。

環境変数 CBLJRTDUMP_MAXSIZE については，「6.2.2　実行時環境変数の詳細」の「(3)　CBLJRTDUMP_MAXSIZE」を参照してください。

### 5.2.2　デバッグ情報の種類と出力フォーマット

Java プログラム呼び出し機能で出力するデバッグ情報の種類と出力フォーマットについて説明します。

ここでは，出力フォーマットに次の表記を使用します。

- △：半角空白（X'20'）
- ▽：タブ文字（X'09'）

## (1) デバッグ情報ファイルの出力例

デバッグ情報ファイルには，出力日時とデバッグ情報を 1 レコードとするデバッグ情報レコードが出力されます。

Java プログラムを呼び出す COBOL プログラムとデバッグ情報ファイルの出力例を次に示します。

Java プログラムを呼び出す COBOL プログラムの例

```
      *>  Javaのsampleクラスのクラス参照を取得する ----------------------

           CALL 'CBLJGETCLASS' USING CBLJENV sample CLASSREF.

      *>  Javaのsampleクラスのインスタンスを生成する ------------------

           CALL 'CBLJNEW' USING CBLJENV CLASSREF NO-ARG OBJREF.

      *>  SampleFieldに値を設定する ------------------------------------

           MOVE 333  TO ARG1-01-AREA.

           CALL 'CBLJSETFIELD' USING CBLJENV
                               OBJREF SampleField ARG1-01.

      *>  SampleMethodメソッドを呼び出す ------------------------------

           MOVE 4    TO ARG1-01-AREA.

           MOVE 'サンプル' TO WORK.
           CALL 'CBLJXTOSTRING' USING CBLJENV WORK WORK-L ARG1-02-AREA.

           COMPUTE ARGPTR(1) = FUNCTION ADDR( ARG1-01 ).
           COMPUTE ARGPTR(2) = FUNCTION ADDR( ARG1-02 ).
           COMPUTE ARGPTR(3) = ZERO.

           CALL 'CBLJINVOKE' USING CBLJENV
                             OBJREF SampleMethod ARG-LIST RTN1.

           CALL 'CBLJDISPLAY' USING CBLJENV RTN1-AREA.

      *>  Javaのインスタンスを解放する --------------------------------

           CALL 'CBLJRELEASE'  USING CBLJENV OBJREF.

      *>  Javaの実行環境を削除する ------------------------------------

           CALL 'CBLJFINALIZE' USING CBLJENV.
```

デバッグ情報ファイルの出力例

```
2014-08-11 23:00:00.000 1000 100 CBLJINITIALIZE # MAIN (******/**)
2014-08-11 23:00:00.000 1000 100 CBLJINITIALIZE argument-1(CBLJENV):
2014-08-11 23:00:00.000 1000 100 CBLJINITIALIZE   00223530: 9c1a3500 00000000 000000c0
64000000 '..5.........d...'
2014-08-11 23:00:00.000 1000 100 CBLJINITIALIZE   00223540: 01000000 2d446a61
'....-Dja'
2014-08-11 23:00:00.000 1000 100 CBLJINITIALIZE  option-1: -Djava.class.path=java
2014-08-11 23:00:00.000 1000 100 CBLJINITIALIZE Environment: CBLJRTDUMP=D:¥temp¥CBLJRTDUMP.log
2014-08-11 23:00:00.150 1000 100 CBLJINITIALIZE # return(0)
2014-08-11 23:00:00.150 1000 100 CBLJINITIALIZE argument-1(CBLJENV):
2014-08-11 23:00:00.150 1000 100 CBLJINITIALIZE   00223530: 9c1a3500 00000000 000000c0
64000000 '..5.........d...'
2014-08-11 23:00:00.150 1000 100 CBLJINITIALIZE   00223540: 01000000 2d446a61
'....-Dja'
2014-08-11 23:00:00.150 1000 100 CBLJINITIALIZE  option-1: -Djava.class.path=java
2014-08-11 23:00:00.150 1000 100 CBLJGETCLASS # MAIN (******/**)
2014-08-11 23:00:00.150 1000 100 CBLJGETCLASS argument-1(CBLJENV):
2014-08-11 23:00:00.150 1000 100 CBLJGETCLASS   00223530: 9c1a3500 00000000 000000c0 64000000
'..5.........d...'
2014-08-11 23:00:00.150 1000 100 CBLJGETCLASS   00223540: 01000000 2d446a61
'....-Dja'
2014-08-11 23:00:00.150 1000 100 CBLJGETCLASS argument-2(CBLJNAME): 'sample'
2014-08-11 23:00:00.150 1000 100 CBLJGETCLASS argument-3(CLASSREF): (null)
2014-08-11 23:00:00.150 1000 100 CBLJGETCLASS # return(0)
2014-08-11 23:00:00.150 1000 100 CBLJGETCLASS argument-3(CLASSREF): 004F89CC[sample]
2014-08-11 23:00:00.150 1000 100 CBLJNEW # MAIN (******/**)
2014-08-11 23:00:00.150 1000 100 CBLJNEW argument-1(CBLJENV):
2014-08-11 23:00:00.150 1000 100 CBLJNEW   00223530: 9c1a3500 00000000 000000c0 64000000
'..5.........d...'
2014-08-11 23:00:00.150 1000 100 CBLJNEW   00223540: 01000000 2d446a61
'....-Dja'
2014-08-11 23:00:00.150 1000 100 CBLJNEW argument-2(CLASSREF): 004F89CC[sample]
2014-08-11 23:00:00.150 1000 100 CBLJNEW argument-3(CBLJLIST): No parameter.
2014-08-11 23:00:00.150 1000 100 CBLJNEW argument-4(OBJECTREF): (null)
2014-08-11 23:00:00.150 1000 100 CBLJNEW # return(0)
2014-08-11 23:00:00.150 1000 100 CBLJNEW argument-4(OBJECTREF): 003580B0[sample]
2014-08-11 23:00:00.150 1000 100 CBLJSETFIELD # MAIN (******/**)
2014-08-11 23:00:00.150 1000 100 CBLJSETFIELD argument-1(CBLJENV):
2014-08-11 23:00:00.150 1000 100 CBLJSETFIELD   00223530: 9c1a3500 00000000 000000c0 64000000
'..5.........d...'
2014-08-11 23:00:00.150 1000 100 CBLJSETFIELD   00223540: 01000000 2d446a61
'....-Dja'
2014-08-11 23:00:00.150 1000 100 CBLJSETFIELD argument-2(OBJECTREF): 003580B0[sample]
2014-08-11 23:00:00.150 1000 100 CBLJSETFIELD argument-3(CBLJNAME): 'SampleField'
2014-08-11 23:00:00.150 1000 100 CBLJSETFIELD argument-4(CBLJPARAM): I: 0000014d (333)
2014-08-11 23:00:00.150 1000 100 CBLJSETFIELD # return(0)
2014-08-11 23:00:00.150 1000 100 CBLJXTOSTRING # MAIN (******/**)
2014-08-11 23:00:00.150 1000 100 CBLJXTOSTRING argument-1(CBLJENV):
2014-08-11 23:00:00.150 1000 100 CBLJXTOSTRING   00223530: 9c1a3500 00000000 000000c0 64000000
'..5.........d...'
2014-08-11 23:00:00.150 1000 100 CBLJXTOSTRING   00223540: 01000000 2d446a61
'….-Dja'
2014-08-11 23:00:00.150 1000 100 CBLJXTOSTRING argument-2(ALNUM):
2014-08-11 23:00:00.150 1000 100 CBLJXTOSTRING   002236E0: 83548393 8376838b 20202020 20202020
'.T...v..        '
2014-08-11 23:00:00.150 1000 100 CBLJXTOSTRING   002236F0: 20202020 20202020 20202020 20202020
'                '
2014-08-11 23:00:00.150 1000 100 CBLJXTOSTRING   00223700: 20202020 20202020 20202020 20202020
'                '
2014-08-11 23:00:00.150 1000 100 CBLJXTOSTRING   00223710: 20202020 20202020 20202020 20202020
'                '
2014-08-11 23:00:00.150 1000 100 CBLJXTOSTRING   00223720: 20202020 20202020 20202020 20202020
'                '
```

```
2014-08-11 23:00:00.150 1000 100 CBLJXTOSTRING   00223730: 20202020 20202020 20202020 20202020
'                '
2014-08-11 23:00:00.150 1000 100 CBLJXTOSTRING   00223740: 20202020
'    '
2014-08-11 23:00:00.150 1000 100 CBLJXTOSTRING argument-3(BIN4): 00000064 (100)
2014-08-11 23:00:00.150 1000 100 CBLJXTOSTRING argument-4(OBJECTREF): (null)
2014-08-11 23:00:00.150 1000 100 CBLJXTOSTRING # return(0)
2014-08-11 23:00:00.150 1000 100 CBLJXTOSTRING argument-4(OBJECTREF): 00358090[java/lang/
String]
2014-08-11 23:00:00.150 1000 100 CBLJINVOKE # MAIN (******/**)
2014-08-11 23:00:00.150 1000 100 CBLJINVOKE argument-1(CBLJENV):
2014-08-11 23:00:00.150 1000 100 CBLJINVOKE   00223530: 9c1a3500 00000000 000000c0 64000000
'..5.........d...'
2014-08-11 23:00:00.150 1000 100 CBLJINVOKE   00223540: 01000000 2d446a61
'....-Dja'
2014-08-11 23:00:00.150 1000 100 CBLJINVOKE argument-2(OBJECTREF): 003580B0[sample]
2014-08-11 23:00:00.150 1000 100 CBLJINVOKE argument-3(CBLJNAME): 'SampleMethod'
2014-08-11 23:00:00.150 1000 100 CBLJINVOKE argument-4(CBLJLIST): [01] I: 00000004 (4)
2014-08-11 23:00:00.150 1000 100 CBLJINVOKE argument-4(CBLJLIST): [02] Ljava/lang/String;:
00358090[java/lang/String]
2014-08-11 23:00:00.150 1000 100 CBLJINVOKE argument-5(CBLJPARAM): Ljava/lang/String;: (null)
2014-08-11 23:00:00.150 1000 100 CBLJINVOKE # return(0)
2014-08-11 23:00:00.150 1000 100 CBLJINVOKE argument-5(CBLJPARAM): Ljava/lang/String;:
003580D0[java/lang/String]
2014-08-11 23:00:00.150 1000 100 CBLJDISPLAY # MAIN (******/**)
2014-08-11 23:00:00.150 1000 100 CBLJDISPLAY argument-1(CBLJENV):
2014-08-11 23:00:00.150 1000 100 CBLJDISPLAY   00223530: 9c1a3500 00000000 000000c0 64000000
'..5.........d...'
2014-08-11 23:00:00.150 1000 100 CBLJDISPLAY   00223540: 01000000 2d446a61
'....-Dja'
2014-08-11 23:00:00.150 1000 100 CBLJDISPLAY argument-2(OBJECTREF): 003580D0[java/lang/String]
2014-08-11 23:00:00.150 1000 100 CBLJDISPLAY # return(0)
2014-08-11 23:00:00.150 1000 100 CBLJRELEASE # MAIN (******/**)
2014-08-11 23:00:00.150 1000 100 CBLJRELEASE argument-1(CBLJENV):
2014-08-11 23:00:00.150 1000 100 CBLJRELEASE   00223530: 9c1a3500 00000000 000000c0 64000000
'..5.........d...'
2014-08-11 23:00:00.150 1000 100 CBLJRELEASE   00223540: 01000000 2d446a61
'....-Dja'
2014-08-11 23:00:00.150 1000 100 CBLJRELEASE argument-2(OBJECTREF): 003580B0[sample]
2014-08-11 23:00:00.150 1000 100 CBLJRELEASE # return(0)
2014-08-11 23:00:00.150 1000 100 CBLJRELEASE argument-2(OBJECTREF): (null)
2014-08-11 23:00:00.150 1000 100 CBLJFINALIZE # MAIN (******/**)
2014-08-11 23:00:00.150 1000 100 CBLJFINALIZE argument-1(CBLJENV):
2014-08-11 23:00:00.150 1000 100 CBLJFINALIZE   00223530: 9c1a3500 00000000 000000c0 64000000
'..5.........d...'
2014-08-11 23:00:00.150 1000 100 CBLJFINALIZE   00223540: 01000000 2d446a61
'....-Dja'
2014-08-11 23:00:00.150 1000 100 CBLJFINALIZE Unreleased object reference 003580D0
2014-08-11 23:00:00.150 1000 100 CBLJFINALIZE Unreleased object reference 00358090
2014-08-11 23:00:00.150 1000 100 CBLJFINALIZE # return(0)
```

## (2) デバッグ情報の種類と出力フォーマット

ここでは，デバッグ情報ファイルに出力されるデバッグ情報の種類と出力フォーマットについて説明します。

デバッグ情報ファイルには，デバッグ情報がデバッグ情報レコードとして出力されます。

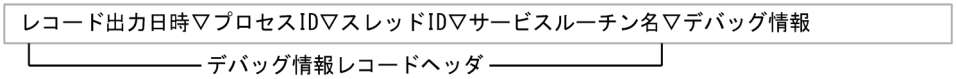
出力されるデバッグ情報を次の表に示します。

表 5-1　出力されるデバッグ情報

| 項番 | デバッグ情報 | 出力情報 |
|---|---|---|
| 1 | サービスルーチン呼び出し情報 | サービスルーチンを呼び出した COBOL プログラムの情報と指定した引数の内容が出力されます。 |
| 2 | サービスルーチン返却値情報 | サービスルーチンの戻り値と更新した引数の内容が出力されます。 |
| 3 | オブジェクト参照解放漏れ情報 | オブジェクト参照の解放漏れが通知されます。 |
| 4 | Java 例外情報 | Java 例外発生時の詳細情報が出力されます。 |
| 5 | 実行時エラー情報 | 実行時エラー発生時の詳細情報が出力されます。 |
| 6 | ユーザデバッグ情報（文字列） | プログラムで指定したデバッグ情報（文字列）が出力されます。 |
| 7 | ユーザデバッグ情報（メモリ） | プログラムで指定したデバッグ情報（メモリ領域の内容）が出力されます。 |
| 8 | 実行環境情報 | Java プログラム呼び出し機能の実行環境情報が出力されます。 |

デバッグ情報レコードのフォーマットを次の図に示します。

図 5-1　デバッグ情報レコードのフォーマット




デバッグ情報レコードヘッダとして出力される情報を次に示します。

- レコード出力日時：「YYYY-MM-DD△hh:mm:ss.sss」の形式のレコード出力日時が出力されます。
- プロセス ID：プログラムを実行しているプロセス ID が出力されます。
- スレッド ID：プログラムを実行しているスレッド ID が出力されます。
- サービスルーチン名：処理中の Java プログラム呼び出し機能のサービスルーチン名が出力されます。

それぞれのデバッグ情報で出力される情報について説明します。

各項に示すフォーマットは，デバッグ情報レコードのデバッグ情報部分だけを示します。一つのデバッグ情報で 2 レコード以上のレコードが出力される場合があります。この場合，それぞれの行にデバッグ情報レコードヘッダが出力されます。

また，デバッグ情報にはメモリ領域の内容が領域のダンプリストのフォーマットで出力されることがあります。出力フォーマットについては，「(3)　データの種類と出力フォーマット」の「(l)　領域のダンプリストの出力フォーマット」を参照してください。

## (a)　サービスルーチン呼び出し情報

サービスルーチンが呼び出されたとき，サービスルーチンを呼び出した COBOL プログラムの情報と指定した引数の内容が出力されます。

フォーマット

```
#△COBOLプログラム名△(行番号/カラム位置)
argument-1(引数1のデータ型):△引数1の内容
      :
argument-n(引数nのデータ型):△引数nの内容
```

COBOL プログラム名

サービスルーチンを呼び出した COBOL プログラム名が出力されます。

行番号/カラム位置

サービスルーチンを呼び出した CALL 文の行番号とカラム位置が出力されます。-DebugInf コンパイラオプションを指定してコンパイルしたプログラムの場合，行番号は 6 けたの数値，カラム位置は 2 けたの数値で出力されます（指定していない場合は，'******/**'が出力されます）。

引数 n のデータ型/内容

n 番目の引数のデータ型を示す文字列と内容が出力されます。内容の表示形式はデータ型によって異なります。表示形式については，「(3)　データの種類と出力フォーマット」を参照してください。

領域のダンプリストなど，引数 n の内容が複数レコードになる場合は，「:」のあとで改行され，次のレコードから引数 n の内容が出力されます。

出力例

```
2014 … CBLJGETCLASS # MAIN (******/**)
2014 … CBLJGETCLASS argument-1(CBLJENV):
2014 … CBLJGETCLASS   00404540: 3c5ea800 00000000 000000e0 64000000 '<^.........d...'
2014 … CBLJGETCLASS   00404550: 02000000 2d766572                     '....-ver'
2014 … CBLJGETCLASS argument-2(CBLJNAME): 'sample'
2014 … CBLJGETCLASS argument-3(CLASSREF): (null)
```

注意事項

引数の CBLJENV 集団項目の指定が誤っている場合など，サービスルーチンの処理が継続できないときは，この情報は出力されないことがあります。

## (b)　サービスルーチン返却値情報

サービスルーチンの処理が終了するとき，サービスルーチンの戻り値とサービスルーチンが更新した引数の内容が出力されます。

フォーマット

```
#△return(戻り値)
argument-x(引数xのデータ型):△引数xの内容
      :
```

戻り値

サービスルーチンの戻り値が出力されます。

引数 x のデータ型と引数 x の内容

サービスルーチンが更新した x 番目の引数のデータ型を示す文字列と内容が出力されます。内容の表示形式はデータ型によって異なります。表示形式については，「(3)　データの種類と出力フォーマット」を参照してください。サービスルーチンが更新する引数がない場合は出力されません。

領域のダンプリストなど，引数 x の内容が複数レコードになる場合は，「:」のあとで改行され，次のレコードから引数 x の内容が出力されます。

**出力例**

```
2014-08-11 23:00:00.000 1000 100 CBLJGETCLASS # return(0)
2014-08-11 23:00:00.000 1000 100 CBLJGETCLASS argument-3(CLASSREF): 00A8AB14[sample]
```

## (c)　オブジェクト参照解放漏れ情報

CBLJFINALIZE サービスルーチン実行時，オブジェクト参照の解放漏れがある場合，解放されていないオブジェクト参照のアドレスが通知されます。解放漏れの個数だけレコードが出力されます。

**フォーマット**

```
Unreleased△object△reference△アドレス
```

アドレス

解放されていないオブジェクト参照のアドレス（16 進形式）が出力されます。

**出力例**

```
2014-08-11 23:00:00.000 1000 100 CBLJFINALIZE Unreleased object reference 00A8B610
```

## (d)　Java 例外情報

Java 例外発生時の詳細情報が出力されます。

**フォーマット**

```
Handled△Java△Exception: CBLJEXCEPTION=例外オブジェクトのアドレス
Java例外の説明
JNI関数名(引数情報)
```

例外オブジェクトのアドレス

CBLJENV 集団項目の CBLJEXCEPTION 項目に設定する例外オブジェクトのアドレス（16 進形式）が出力されます。Java プログラムから例外がスローされた場合に設定されます。デバッグ情報出力機能では，JNI の例外で CBLJEXCEPTION 項目を更新しない場合は，このレコードは出力されません。例外処理については，「3.2.4　例外処理のコーディング」を参照してください。

Java 例外の説明

JNI 関数の例外およびスタックのバックトレースが出力されます。

JNI 関数名

最後に呼び出した JNI 関数名が出力されます。

引数情報

JNI 関数に渡した引数の値の並びが出力されます。引数が複数ある場合は，コンマ（,）で区切って出力されます。

出力例

```
2014 … CBLJINVOKE Free temporary memory: address=00A8B698
2014 … CBLJINVOKE Handled Java Exception: CBLJEXCEPTION=00A8B650
2014 … CBLJINVOKE java.lang.ArithmeticException: / by zero
2014 … CBLJINVOKE sample.SampleMethod(sample.java:8)
2014 … CBLJINVOKE CallObjectMethodA(689b4203,ececb600,a2154303,1cfe1200)
```

## (e) 実行時エラー情報

実行時エラー発生時の詳細情報が出力されます。

フォーマット

```
KCCCnnnnR-x△詳細メッセージ
```

KCCCnnnnR-x

実行時のメッセージ ID が出力されます。

詳細メッセージ

実行時のメッセージのエラー情報に出力される詳細メッセージが出力されます。

出力例

```
2014 … CBLJNEW KCCC7904R-S [0061] <init> に対してJNIで例外が発生しました。（CBLJNEW／･･･）
```

## (f) ユーザデバッグ情報（文字列）

CBLJDEBUGSTRING サービスルーチンを呼び出したとき，ユーザデバッグ情報として，指定された文字列がデバッグ情報ファイルに出力されます。

フォーマット

```
#△COBOLプログラム名△(行番号/カラム位置)
ユーザデバッグ情報
```

COBOL プログラム名

CBLJDEBUGSTRING サービスルーチンを呼び出した COBOL プログラム名が出力されます。

行番号/カラム位置

CBLJDEBUGSTRING サービスルーチンを呼び出した CALL 文の行番号とカラム位置が出力されます。-DebugInf コンパイラオプションを指定してコンパイルしたプログラムの場合，行番号は 6 けたの数値，カラム位置は 2 けたの数値で出力されます（指定していない場合は，'******/**'が出力されます）。

ユーザデバッグ情報

CBLJDEBUGSTRING サービスルーチンの引数 2 に指定した英数字項目の値が出力されます。引数 3 に指定した長さだけ出力されます。

**注意事項**

ユーザデバッグ情報に，NULL 終端文字（X'00'）が含まれる場合は，それ以降の文字列は出力されません。

## (g) ユーザデバッグ情報（メモリ）

CBLJMEMDUMP サービスルーチンを呼び出したとき，指定されたメモリ領域のダンプリストがデバッグ情報ファイルに出力されます。

**フォーマット**

```
#△COBOLプログラム名△(行番号/カラム位置)
領域のダンプリスト
```

COBOL プログラム名

CBLJMEMDUMP サービスルーチンを呼び出した COBOL プログラム名が出力されます。

行番号/カラム位置

CBLJMEMDUMP サービスルーチンを呼び出した CALL 文の行番号とカラム位置が出力されます。-DebugInf コンパイラオプションを指定してコンパイルしたプログラムの場合，行番号は 6 けたの数値，カラム位置は 2 けたの数値で出力されます（指定していない場合は，'******/**'が出力されます）。

領域のダンプリスト

CBLJMEMDUMP サービスルーチンの引数 2 に指定したポインタ項目が指すメモリ領域のダンプリスト（16 進形式）が出力されます。引数 3 に指定した長さだけ出力されます。

## (h) 実行環境情報

Java プログラム呼び出し機能の実行環境情報が出力されます。CBLJINITIALIZE サービスルーチンで実行環境が構築されたあとに出力されます。

**フォーマット**

```
Environment:△環境変数名=環境変数の値
     :
```

環境変数名

Java プログラム呼び出し機能で使用する環境変数の名称が出力されます。実行時環境変数が指定されているとき，このレコードは繰り返して出力されます。実行時環境変数については，「6.2 実行時環境変数」を参照してください。

環境変数の値

環境変数に指定された値が出力されます。ただし，値の長さが 2,048 バイトを超えるとき，先頭から 2,048 バイトまでが出力されます。

出力例

```
2014-08-11 23:00:00.000 1000 100 CBLJINITIALIZE Environment: CBLJRTDUMP=D:¥temp¥CBLJRTDUMP.log
```

## (3) データの種類と出力フォーマット

ここでは，デバッグ情報で出力される引数などのデータの種類の出力フォーマットについて説明します。

### (a) 引数の内容の出力フォーマット

データの内容は，出力されるデータの種類（データ型）によって出力フォーマットが異なります。

デバッグ情報に出力されるデータの種類を次の表に示します。

表 5-2 デバッグ情報に出力されるデータの種類

| 項番 | データの種類（データ型） | 「データ型」として出力される文字列 |
|---|---|---|
| 1 | CBLJENV 集団項目 | CBLJENV |
| 2 | 動的長基本項目／名前型集団項目 | CBLJNAME |
| 3 | パラメタ型集団項目 | CBLJPARAM |
| 4 | 引数リスト型集団項目 | CBLJLIST |
| 5 | クラス参照 | CLASSREF |
| 6 | オブジェクト参照 | OBJECTREF |
| 7 | 英数字項目 | ALNUM |
| 8 | 日本語項目 | JPN |
| 9 | 数字項目（4 バイト 2 進形式） | BIN4 |
| 10 | Java 型名 | JAVATYPE |
| 11 | ポインタ項目（配列アドレス） | ADDR |

### (b) CBLJENV 集団項目

CBLJENV 集団項目は次のフォーマットで出力されます。

フォーマット

```
領域のダンプリスト
▽option-1:△Java VM起動オプション1
    :
▽option-n:△Java VM起動オプションn
```

領域のダンプリスト

CBLJENV 集団項目領域のダンプリスト（16 進形式）が出力されます。

Java VM 起動オプション n

n 番目に指定した Java VM 起動オプション文字列が出力されます。出力長は CBLJENV 集団項目中の CBLJSTRMAXLEN に指定した値です。

この情報は，CBLJINITIALIZE サービスルーチンでのデバッグ情報出力時だけ出力されます。

### (c)　動的長基本項目／名前型集団項目

名前型集団項目は次のフォーマットで出力されます。

フォーマット

```
'名前文字列'
```

名前文字列

名前型集団項目に指定した文字列が出力されます。NULL 終端文字（X'00'）までが出力されます。

### (d)　パラメタ型集団項目

パラメタ型集団項目は，パラメタ型集団項目中の Java の型を表す文字に従ってデータの内容が出力されます。

フォーマット

```
Javaの型:△データの内容
```

Java の型

パラメタ型集団項目中の Java の型を表す文字列が出力されます。パラメタ型集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(3)　パラメタ型集団項目」を参照してください。

パラメタ型集団項目が未構築の場合は，次に示す内容が出力されます。

```
Not△yet△constructed.
```

Java の型を表す文字列に該当しない値の場合は，次に示す内容が出力されます。

```
X'xx'△is△invalid△parameter△type.
```

xx：パラメタ型集団項目中の Java の型を表す文字列の先頭 1 バイトの値（16 進形式）を示します。

データの内容

次の表に示す内容が出力されます。

表 5-3　パラメタ型集団項目中の Java の型とデータの内容

| 項番 | Java の型を表す文字と Java の型 | 「データの内容」に出力される内容 |
|---|---|---|
| 1 | V (void) | (出力しない) |
| 2 | B (byte) | 領域（1 バイト）の内容（16 進形式）△(値) |

| 項番 | Java の型を表す文字と Java の型 | 「データの内容」に出力される内容 |
|---|---|---|
| 3 | C (char) | 領域（2 バイト）の内容（16 進形式）△(値) |
| 4 | S (short) | 領域（2 バイト）の内容（16 進形式）△(値) |
| 5 | I (int) | 領域（4 バイト）の内容（16 進形式）△(値) |
| 6 | J (long) | 領域（8 バイト）の内容（16 進形式）△(値) |
| 7 | F (float) | 領域（4 バイト）の内容（16 進形式）△(値)※ |
| 8 | D (double) | 領域（8 バイト）の内容（16 進形式）△(値)※ |
| 9 | Z (boolean) | 領域（1 バイト）の内容（16 進形式）△(値) |
| 10 | T (クラス参照) | 「(f)　クラス参照のポインタ項目」を参照してください。 |
| 11 | L (オブジェクト参照) | 「(g)　オブジェクト参照のポインタ項目」を参照してください。 |
| 12 | [ (配列オブジェクト) | |

注※

値の表示精度は，Java プログラム呼び出し機能が使用している C コンパイラの仕様に従うため，COBOL または Java プログラムで設定した値と異なることがあります。

## (e)　引数リスト型集団項目

引数リスト型集団項目は，パラメタ型集団項目を指すアドレス値の配列です。配列の終端まで出現順序で配列の要素（パラメタ型集団項目）のデータの内容が出力されます。

### フォーマット

```
[出現順序]△パラメタ型集団項目で出力する内容
```

出現順序

パラメタ型集団項目を指すアドレス値の出現順序（1 から昇順）が出力されます。

配列の要素がない（配列の先頭が NULL 値）の場合は，次に示す内容が出力されます。

```
No△parameter.
```

## (f)　クラス参照のポインタ項目

クラス参照を示すポインタ項目は，アドレスとクラス名が出力されます。

### フォーマット

```
アドレス[クラス名]
```

アドレス

ポインタ項目の値

クラス名

ポインタ項目が示すクラス参照に保持しているクラス名が出力されます。クラス参照が未構築の場合は「(null)」が出力されます。

### (g) オブジェクト参照のポインタ項目

オブジェクト参照を示すポインタ項目は，アドレスが出力されます。

フォーマット

```
アドレス[クラス名]
```

アドレス

ポインタ項目の値

クラス名

ポインタ項目が示すオブジェクト参照が指すクラス参照に保持しているクラス名が出力されます。オブジェクト参照が未構築，またはクラス参照のアドレスが NULL 値の場合は「(null)」が出力されます。

### (h) 英数字項目／日本語項目

英数字項目および日本語項目は，サービスルーチンに同時に指定した文字数（数字項目の値）分の領域のダンプリストが出力されます。

フォーマット

```
領域のダンプリスト
```

領域のダンプリスト

英数字項目または日本語項目の領域のダンプリストが出力されます。出力される領域サイズ（バイト数）を次に示します。

- 英数字項目の場合，指定した文字数
- 日本語項目の場合，指定した文字数×2

### (i) 数字項目（2 進形式）

数字項目（2 進形式）は，指定した数字項目の値とメモリ領域の内容（16 進形式）が出力されます。

フォーマット

```
領域の内容（16進形式）△(値)
```

領域の内容（16 進形式）

数字項目のメモリ領域の内容（16 進形式）が出力されます。数字項目に該当するバイト数だけ出力されます。

値

数字項目の値（10 進形式）が出力されます。

## (j)　Java 型名

Java 型名は，指定された Java 型名が出力されます。フォーマットについては，「(d)　パラメタ型集団項目」の Java の型を参照してください。

## (k)　ポインタ項目

ポインタ項目は，アドレス値が出力されます。

### フォーマット

```
アドレス
```

アドレス

ポインタ項目に設定されているアドレス値が出力されます。

## (l)　領域のダンプリストの出力フォーマット

領域のダンプリストを出力する場合，該当するデータが示すメモリ領域の内容が 16 進形式で出力されます。

### フォーマット

```
△△アドレス:△メモリ領域の内容（16進形式）△'メモリの内容（文字列）'
```

アドレス

領域の先頭アドレスが出力されます。

メモリ領域の内容（16 進形式）

領域の先頭アドレスから 16 バイト分のメモリ領域の内容（16 進形式）が出力されます。

- 区切り文字として，4 バイトごとに半角空白が出力されます。
- a～f は小文字で出力されます。

メモリ領域の内容（文字列）

領域の先頭アドレスから 16 バイト分のメモリ領域の内容を，1 バイトずつ文字に変換した文字列が出力されます。ASCII コードの制御コードおよび文字が割り当てられていないコードに該当する場合は，「.（ピリオド）」が出力されます。

メモリ領域の内容は 1 レコードに 16 バイトずつ出力されます。出力されるサイズが 16 バイトを超える領域の場合は，16 バイトごとに複数のレコードが出力されます。

また，16 バイトに満たない場合，メモリ領域の内容（16 進形式）の残りの領域には半角空白が出力され，メモリ領域の内容（文字列）には領域の終端の位置に「'」が出力されます。

CBLJGETCLASS サービスルーチンの引数（CBLJENV 集団項目）の出力例

```
2014 … CBLJGETCLASS argument-1(CBLJENV):
2014 … CBLJGETCLASS   00404540: 3c5ea800 00000000 000000e0 64000000 '<^...........d...'
2014 … CBLJGETCLASS   00404550: 02000000 2d766572                     '....-ver'
```

## 5.2.3　ユーザデバッグ情報を出力するサービスルーチン

サービスルーチンを使用して，実行中の COBOL プログラムからデバッグ情報ファイルにデバッグ情報を出力できます。

ユーザデバッグ情報を出力するためのサービスルーチンを次に示します。

- CBLJDEBUGSTRING：ユーザデバッグ情報（文字列）を出力します。
- CBLJMEMDUMP：ユーザデバッグ情報（メモリ領域の内容）を出力します。

ここでは，CBLJDEBUGSTRING サービスルーチンと CBLJMEMDUMP サービスルーチンの使用方法について説明します。

サービスルーチンについては，「6.1.7　デバッグ操作」の「(1)　CBLJDEBUGSTRING」および「(2)　CBLJMEMDUMP」を参照してください。

### (1)　ユーザデバッグ情報（文字列）の出力（CBLJDEBUGSTRING サービスルーチン）

CBLJDEBUGSTRING サービスルーチンを呼び出して，デバッグ情報ファイルに文字列情報を出力できます。

出力フォーマットについては，「5.2.2　デバッグ情報の種類と出力フォーマット」の「(2)　デバッグ情報の種類と出力フォーマット」の「(f)　ユーザデバッグ情報（文字列）」を参照してください。

使用方法と出力例

COBOL プログラム（ユーザデバッグ情報（メモリ）を出力する）

```
01 D-ALPH            PIC X(20)
                     VALUE 'Hallo COBOL2002.'.
01 D-ALPH-LEN        PIC S9(9) USAGE COMP VALUE 20.

  CALL 'CBLJDEBUGSTRING' USING CBLJENV D-ALPH D-ALPH-LEN.
```

デバッグ情報ファイル

```
2014-08-11 00:00:00.000 1000 100 CBLJDEBUGSTRING # MAIN (******/**)
2014-08-11 00:00:00.000 1000 100 CBLJDEBUGSTRING 'Hallo COBOL2002.     '
```

## (2) ユーザデバッグ情報（メモリ領域の内容）の出力（CBLJMEMDUMP サービスルーチン）

CBLJMEMDUMP サービスルーチンを呼び出して，デバッグ情報ファイルにメモリ領域の内容を出力できます。

出力フォーマットについては，「5.2.2　デバッグ情報の種類と出力フォーマット」の「(2)　デバッグ情報の種類と出力フォーマット」の「(g)　ユーザデバッグ情報（メモリ）」を参照してください。

使用方法と出力例

COBOL プログラム（ユーザデバッグ情報（メモリ）を出力する）

```
01 D-ALPH           PIC X(20)
                    VALUE 'Hallo COBOL2002.'.
01 D-ALPH-LEN       PIC S9(9) USAGE COMP VALUE 20.
01 D-ALPH-PTR       USAGE POINTER.

  COMPUTE D-ALPH-PTR = FUNCTION ADDR(D-ALPH).
  CALL 'CBLJMEMDUMP' USING CBLJENV D-ALPH-PTR D-ALPH-LEN.
```

デバッグ情報ファイル

```
2014 … CBLJMEMDUMP # MAIN (******/**)
2014 … CBLJMEMDUMP   00534c63: 48616c6c 6f20434f 424f4c32 3030322e 'Hallo COBOL2002.'
2014 … CBLJMEMDUMP   00534c73: 20202020                          '    '
```

## 5.3　実行時エラー情報の出力

Java プログラム呼び出し機能では，Java プログラム呼び出し機能を使用した COBOL プログラムで実行時エラーが発生したときの状態をファイル（実行時エラー情報ファイル）に出力できます。

特別な設定をしなくても実行時エラー発生時に常に情報が出力されるため，運用時の障害調査などに使用できます。

### 5.3.1　実行時エラー情報ファイルに出力される情報

実行時エラー情報ファイルに出力される情報は次のとおりです。

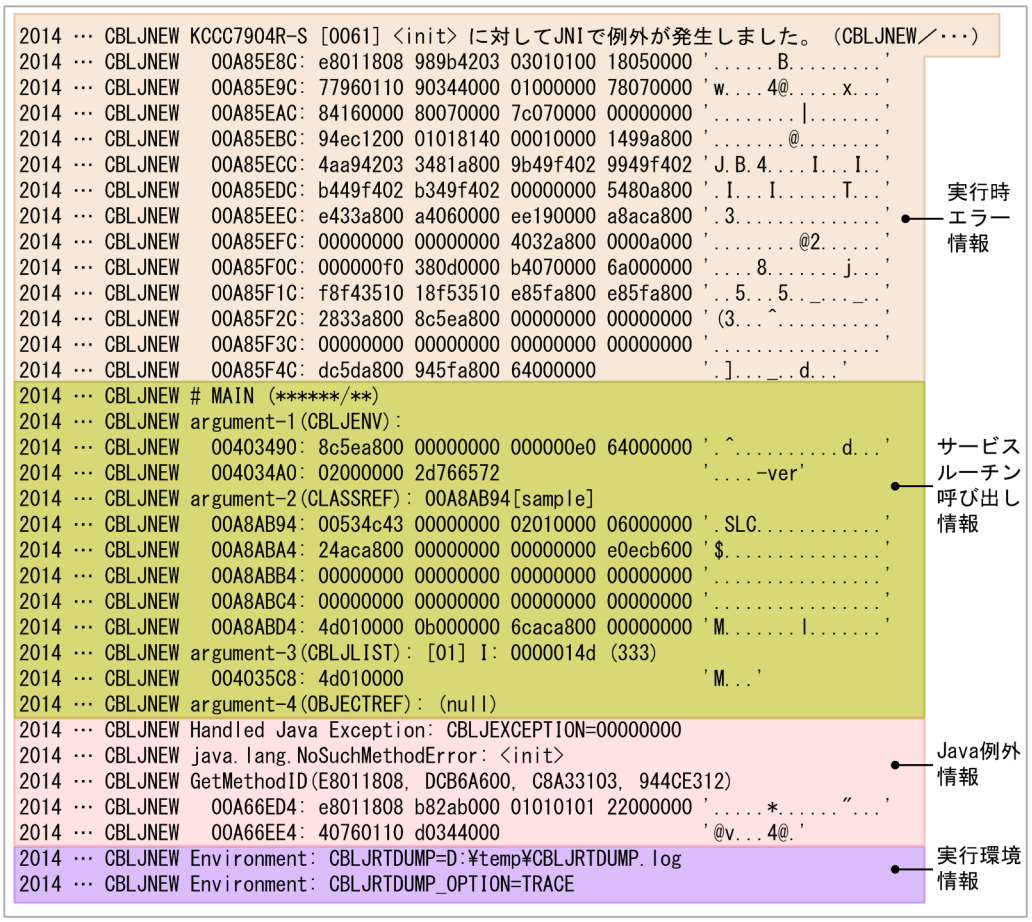
- 実行時エラー情報
  実行時エラー発生時の詳細情報です。
  - 実行時エラー番号と詳細メッセージ
  - Java プログラム呼び出し機能の管理領域のダンプリスト（16 進形式）
- サービスルーチン呼び出し情報
  最後に呼び出したサービスルーチンの呼び出し情報です。
  - 呼び出した COBOL プログラムの情報
  - 指定した引数の内容とメモリ領域のダンプリスト（16 進形式）
- Java 例外情報
  Java 例外発生時の詳細情報です（ただし，JNI 関数または使用する Java プログラムで例外が発生したときだけ出力されます）。
  - Java 例外情報とバックトレース
  - 最後に呼び出した JNI 関数と引数情報
  - Java プログラム呼び出し機能の管理領域のダンプリスト（16 進形式）
- 実行環境情報
  Java プログラム呼び出し機能の実行環境情報です。

出力されるフォーマットは，デバッグ情報の出力フォーマットと同じです。

なお，デバッグ情報で出力されるデータのほかに，当社保守員が調査するときに使用するデータが追加で出力されます。

出力例を次に示します。

出力例




## 5.3.2　実行時エラー情報の出力先

実行時エラー情報ファイルは，次に示すファイル名で出力されます。

```
CBLJRTERR_YYYYMMDD_hhmmss_ppppp_ttttt
```

YYYYMMDD_hhmmss

　実行時エラー情報出力時のタイムスタンプが出力されます。

ppppp

　プロセス ID（10 進形式）が出力されます。

ttttt

　スレッド ID（10 進形式）が出力されます。

実行時エラー情報ファイルの出力先フォルダの優先順位を次に示します。

1. 環境変数 CBLJRTERR に指定したフォルダ
2. 環境変数 TEMP に指定したフォルダ

3. 環境変数 TMP に指定したフォルダ
4. カレントフォルダ

注意事項

- 実行時エラー情報ファイルを出力するときに，ファイルの出力先フォルダは作成しません。出力先はあらかじめ存在している必要があります。環境変数に存在しないフォルダのパスを指定している場合は，優先順位は次の順位に移動します。
- ファイルの書き込みでエラーが発生した場合，実行時エラー情報ファイルは出力されません。実行時エラー情報ファイルが出力されない場合の要因を次に示します。確認して対処してください。
  ・ファイルの出力先フォルダまたは指定したファイルに書き込み権限がない。
  ・出力するファイル名を加えたパス名の長さがシステムの制限を超える。
  ・ディスク容量が不足している。

環境変数 CBLJRTERR については，「6.2.2　実行時環境変数の詳細」の「(4)　CBLJRTERR」を参照してください。

## 5.3.3　実行時エラー情報の出力時の注意事項

実行時エラー情報の出力時の注意事項を次に示します。

- Java プログラム呼び出し機能では，実行時エラー情報ファイルを削除しません。ディスク容量不足の要因となることがあるため，定期的に不要なファイルを削除してください。
- 障害発生時の調査情報として有効のため，運用時には環境変数 CBLJRTERR に"<SUPPRESS>"を指定しないでください。

## 5.4　Java VM 起動オプション情報の収集

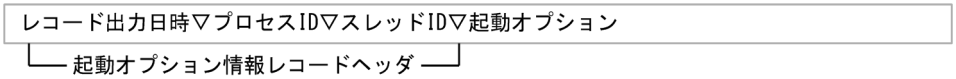
起動オプション情報とは，Java VM で異常終了する場合など，Java プログラム呼び出し機能でエラーの情報が取得できないときの調査に必要な情報です。

Java プログラム呼び出し機能が Java VM を起動するとき，起動オプション情報を収集します。

### 5.4.1　起動オプション情報の収集

Java プログラム呼び出し機能が Java VM を起動するとき，起動オプション情報が出力先ファイル（CBLJRTVMOPT.log）に追加書きで出力されます。起動オプションを複数指定している場合は，指定した数だけレコードが出力されます。

1 レコードの出力フォーマットを次に示します。




ファイルの書き込みでエラーが発生した場合，標準エラー出力にデータが出力されます。

### 5.4.2　起動オプション情報の出力先

起動オプション情報は，CBLJRTVMOPT.log ファイルに出力されます。

出力先フォルダは，次に示す優先順位で決定されます。

1. 環境変数 CBLJRTVMOPTLOG に指定したフォルダ
2. 環境変数 TEMP に指定したフォルダ
3. 環境変数 TMP に指定したフォルダ
4. カレントフォルダ

起動オプション情報の出力先フォルダは，環境変数 CBLJRTVMOPTLOG で変更できます。

環境変数 CBLJRTVMOPTLOG については，「6.2.2　実行時環境変数の詳細」の「(7) CBLJRTVMOPTLOG」を参照してください。

### 5.4.3　起動オプション情報の出力先の最大サイズ

起動オプション情報の出力先ファイルの既定の最大サイズは，2MB です。

最大サイズのチェックと出力処理を次に示します。

- 起動オプション情報の書き込みで出力先ファイルが最大サイズを超えない場合，起動オプション情報の出力先ファイルに追加書きします。
- 起動オプション情報の書き込みで出力先ファイルが最大サイズを超える場合，書き込み前の出力先ファイルをバックアップしたあと，出力先ファイルを新規で作成します。バックアップファイルの名称は，出力先ファイルのファイル名の後ろ 1 バイトをアンダーバー（_）に置き換えた名称です。すでに同じ名称のファイルが存在する場合は，上書きします。
- 出力先ファイルのバックアップが失敗したとき，最大サイズを超えて，出力先ファイルに追加書きします。出力時にすでに最大サイズを超えている場合も追加書きします。

環境変数 CBLJRTVMOPTLOG_MAXSIZE を指定して最大サイズを変更できます。

環境変数 CBLJRTVMOPTLOG_MAXSIZE については，「6.2.2　実行時環境変数の詳細」の「(8) CBLJRTVMOPTLOG_MAXSIZE」を参照してください。

# 6 API リファレンス

この章では，Java プログラム呼び出し機能が提供するサービスルーチン，および実行時環境変数について説明します。

# 6.1　サービスルーチン

ここでは，Java プログラム呼び出し機能が提供するサービスルーチンについて，基本操作，クラス操作，オブジェクト操作，文字列オブジェクト操作，配列オブジェクト操作，デバッグ操作別に説明します。

Java プログラム呼び出し機能が提供するサービスルーチンの一覧を次の表に示します。

## 表 6-1　サービスルーチンの一覧

| 項番 | サービスルーチン | 機能 |
|---|---|---|
| 基本操作 | | |
| 1 | CBLJINITIALIZE | Java プログラム呼び出し機能を初期化する。 |
| 2 | CBLJGETCLASS | Java クラス参照を返す。 |
| 3 | CBLJSETSTATICFIELD | クラスフィールドに値を設定する。 |
| 4 | CBLJGETSTATICFIELD | クラスフィールドの値を取り出す。 |
| 5 | CBLJSTATICINVOKE | クラスメソッドを呼び出す。 |
| 6 | CBLJNEW | Java クラスのインスタンスを生成して，Java オブジェクト参照を返す。 |
| 7 | CBLJSETFIELD | インスタンスフィールドに値を設定する。 |
| 8 | CBLJGETFIELD | インスタンスフィールドから値を取り出す。 |
| 9 | CBLJINVOKE | インスタンスメソッドを呼び出す。 |
| 10 | CBLJRELEASE | Java オブジェクト参照を解放する。 |
| 11 | CBLJFINALIZE | Java プログラム呼び出し機能の実行環境を削除する。 |
| クラス操作 | | |
| 12 | CBLJGETOBJCLASS | インスタンスの属する Java クラスの Java クラス参照を返す。 |
| 13 | CBLJGETNAME | Java クラス参照の示す Java クラスの名前を返す。 |
| 14 | CBLJGETSUPERCLASS | Java クラス参照の示す Java クラスのスーパークラスの Java クラス参照を返す。 |
| オブジェクト操作 | | |
| 15 | CBLJCLASSNAME | インスタンスの属する Java クラスの名前を返す。 |
| 16 | CBLJINSTANCEOF | インスタンスが Java クラス参照で指定されたクラスのインスタンスまたはそのサブクラスのインスタンスかを判定する。 |
| 17 | CBLJSAMEOBJECT | 二つの Java オブジェクト参照が同じ Java インスタンスを指すかどうかを判定する。 |
| 18 | CBLJEQUAL | 二つの Java インスタンスが同じ値を持つかどうかを判定する。 |
| 19 | CBLJCOPY | Java オブジェクト参照の複製を作成して転記する。 |
| 20 | CBLJSETNULL | Java オブジェクト参照用のポインタ項目に NULL を設定する。 |
| 文字列オブジェクト操作 | | |

| 項番 | サービスルーチン | 機能 |
|---|---|---|
| 21 | CBLJXTOSTRING | 英数字項目から String オブジェクトを生成する。 |
| 22 | CBLJNTOSTRING | 日本語項目から String オブジェクトを生成する。 |
| 23 | CBLJSTRINGTOX | String オブジェクトの文字列を英数字項目に格納する。 |
| 24 | CBLJSTRINGTON | String オブジェクトの文字列を日本語項目に格納する。 |
| 25 | CBLJSTRLENGTH | String オブジェクトの長さ（文字数）を返す。 |
| 26 | CBLJDISPLAY | String オブジェクトの内容を標準出力に出力する。 |
| 配列オブジェクト操作 | | |
| 27 | CBLJNEWARRAY | 配列オブジェクトのインスタンスを生成する。 |
| 28 | CBLJARRAYLENGTH | 配列オブジェクトの要素数を返す。 |
| 29 | CBLJSETOBJARRAY | オブジェクト型の配列オブジェクトのインデクスが示す要素にオブジェクトを格納する。 |
| 30 | CBLJGETOBJARRAY | オブジェクト型の配列オブジェクトのインデクスの示す要素からオブジェクトを取り出す。 |
| 31 | CBLJGETARRAYADDR | 基本型の配列オブジェクトから基本型配列のアドレスを取得する。 |
| 32 | CBLJRELEASEARRAY | 基本型配列の内容変更を配列オブジェクトに反映させて，基本型配列の領域を解放する。 |
| デバッグ操作 | | |
| 33 | CBLJDEBUGSTRING | ユーザデバッグ情報として文字列情報をデバッグ情報ファイルに出力する。 |
| 34 | CBLJMEMDUMP | ユーザデバッグ情報としてメモリ領域の内容をデバッグ情報ファイルに出力する。 |

## 6.1.1　サービスルーチンで使用する引数

ここでは，Java プログラム呼び出し機能のサービスルーチンで使用する引数について説明します。

## (1)　CBLJENV 集団項目

Java プログラム呼び出し機能のサービスルーチンの第 1 引数には，CBLJENV 集団項目を指定します。

CBLJENV 集団項目は，スレッドごとに一つの領域を確保する必要があります。また，一つの Java プログラム呼び出し機能の実行単位（CBLJINITIALIZE サービスルーチンで初期化してから CBLJFINALIZE サービスルーチンで終了するまでの間）に，複数の CBLJENV 集団項目を使用することはできません。

CBLJENV 集団項目は，作業場所節に 01 レベルの集団項目として記述してください。

CBLJENV 集団項目を次に示します。

```
      01  CBLJENV.
       02  CBLJENVCORE    USAGE POINTER         VALUE NULL.              *> JNI/Java VM情報
       02  CBLJEXCEPTION  USAGE POINTER         VALUE NULL.              *> 例外情報
       02  CBLJFLAGS      PIC 1(32) USAGE BIT   VALUE ALL B'0'.          *> オプションフラグ
```

```
       02  CBLJSTRMAXLEN  PIC S9(9) USAGE COMP VALUE m.                *> 引数で使う文字列長
       02  CBLJVMOPTIONS.                                             *> Java VM起動オプ
ション
         03 CBLJOPTCOUNT  PIC S9(9) USAGE COMP VALUE n.                *> オプション数
         03 CBLJOPTION-1  PIC X(m)  VALUE 'オプション文字列'.
         03 CBLJOPTION-2  PIC X(m)  VALUE 'オプション文字列'.
            :
         03 CBLJOPTION-n  PIC X(m)  VALUE 'オプション文字列'.
```

複数のプログラムで一つの実行単位を構成する場合にも，一つの CBLJENV 集団項目を使用する必要があります。あるプログラムの作業場所節で CBLJENV 集団項目を定義し，ほかのプログラムでは参照渡し（BY REFERENCE）の引数として受け渡した CBLJENV 集団項目を使用します。

複数のプログラムで一つの CBLJENV 集団項目を使用する場合を次に示します。

```
       IDENTIFICATION  DIVISION.
       PROGRAM-ID.     MAIN.
       DATA            DIVISION.
       WORKING-STORAGE SECTION.
       01  CBLJENV.
        02  CBLJENVCORE    USAGE POINTER        VALUE NULL.
             :
       PROCEDURE       DIVISION.
           CALL 'SUB1' USING CBLJENV.
             :

       IDENTIFICATION  DIVISION.
       PROGRAM
       PROGRAM-ID.     SUB1.
       DATA            DIVISION.
       LINKAGE         SECTION.
       01  CBLJENV.
        02  CBLJENVCORE    USAGE POINTER.
             :
       PROCEDURE       DIVISION USING CBLJENV.
           CALL 'CBLJGETCLASS' USING CBLJENV SampleClass CLASSREF.
```

CBLJENV 集団項目の内容を次の表に示します。

### 表 6-2　CBLJENV 集団項目の内容

| 項番 | 項目名 | 設定値 |
|---|---|---|
| 1 | CBLJENVCORE | CBLJINITIALIZE サービスルーチンで作成された Java の実行環境（JNI および Java VM）の管理情報が格納されます。<br>必ず NULL で初期化してください。 |
| 2 | CBLJEXCEPTION | Java のコンストラクタやメソッドの呼び出しでスローされた例外オブジェクトが格納されます。<br>必ず NULL で初期化してください。 |
| 3 | CBLJFLAGS | Java プログラム呼び出し機能の初期化時オプションが格納されます。<br>必ず ALL B'0'で初期化してください。 |

| 項番 | 項目名 | 設定値 |
|---|---|---|
| 4 | CBLJSTRMAXLEN | サービスルーチンへの引数で使う英数字項目または日本語項目の最大のサイズ（バイト数）を指定します。1～1,024 の範囲で指定します。 |
| 5 | CBLJVMOPTIONS | Java VM の起動オプションを英数字項目で指定します。<br>項目のサイズは CBLJSTRMAXLEN に指定した値です。<br>なお，オプション文字列の値の先頭および末尾の連続した半角空白は無視されます。<br>また，ほかのスレッドで Java VM がすでに起動されている場合，ここで指定したオプションは適用されません。 |

CBLJVMOPTIONS の CBLJOPTION-(1～n)項目に指定できるオプションの例を次に示します。指定するプロパティ名や値などについては，使用する Java VM のドキュメントを参照してください。

Java VM 起動オプションの例 1

オプション文字列：-D プロパティ名=値

説明：システムプロパティを設定します。

例

```
-Djava.compiler=NONE      …JITの無効化設定
-Djava.class.path=パス    …クラスのパス設定
-Djava.library.path=パス …ライブラリのパス設定
```

Java VM 起動オプションの例 2

オプション文字列：-verbose:｛gc ｜ class ｜ jni｝

説明：指定メッセージ種別の詳細メッセージ出力を有効にします。

gc：ガーベジコレクション関連メッセージ

class：クラスロード関連メッセージ

jni：JNI 関連メッセージ

**注意事項**

CBLJENV 集団項目の初期値は，CBLJINITIALIZE サービスルーチンが明示的または暗黙的に呼び出される前に設定してください。CBLJINITIALIZE サービスルーチンが呼び出されたあとで変更しないでください。

## (2)　名前型集団項目

Java のクラス名，メソッド名，およびフィールド名を引数として渡す場合に，動的長基本項目の代わりに名前型集団項目を使用できます。名前型集団項目は，最後の領域が NULL 終端文字となる LOW-VALUE となるように定義します。

名前型集団項目の例を次に示します。

```
01  名前.
  02  NAMESTR  PIC X(16) VALUE '名前の文字列'.                    *> ※1
  02  FILLER   PIC X     VALUE LOW-VALUE.                        *> ※2
```

注※1

項目の名前は NAMESTR 以外を設定できます。参照しない場合は，FILLER を設定できます。

注※2

NULL 終端文字を示します。

名前の文字列にパッケージ付きのクラス名やインタフェース名を記述する場合は，ピリオドの代わりにスラントを使って区切ります。

例）

```
java/lang/String
```

NAMESTR 項目のサイズは，名前の文字列のサイズより大きく宣言してもよく，名前のあとに連続する空白文字が挿入されていてもかまいません。ただし，Java プログラム呼び出し機能のサービスルーチンに渡した場合，末尾の連続する空白文字は LOW-VALUE 文字に置き換えられます。

## (3) パラメタ型集団項目

Java のメソッドやフィールドとデータの受け渡しをするデータ項目は次のように記述されている必要があります。

基本データ型の場合

```
01  データ項目名.
  02  DATA-TYPE  PIC X.                                        *> ※1
  02  FILLER     PIC X(7) VALUE ALL LOW-VALUE.                 *> ※2
  02  DATA-AREA  受け渡し領域のデータ項目のデータ記述項.          *> ※3
```

オブジェクト系の型（クラスオブジェクト，オブジェクト，および配列オブジェクト）の場合

```
01  データ項目名.
  02  DATA-TYPE  PIC X(m).                                     *> ※1
  02  DATA-AREA  受け渡し領域のデータ項目のデータ記述項.          *> ※3
```

注※1

DATA-TYPE 項目には，Java の型を表す文字列（シグニチャ）を指定します。

基本データ型の場合は，1 文字の英数字項目を指定します。

オブジェクト系の型の場合は，CBLJENV 集団項目の CBLJSTRMAXLEN に指定したサイズの英数字項目を指定します。CBLJENV 集団項目の CBLJSTRMAXLEN に指定したサイズ以外で定義すると，アクセス違反などの不正な動作となります。

注※2

この値は必ず LOW-VALUE となるように定義してください。

注※3

DATA-AREA 項目には，Java の型に対応する COBOL のデータ項目を指定します。

Java の型，Java の型を表す文字列（シグニチャ），Java の型に対応する COBOL のデータ項目の関係を次の表に示します。

## 表 6-3　Java の型と対応する COBOL の型

| 項番 | Java の型 | Java の型を表す文字列（シグニチャ） | Java の型に対応する COBOL の型 | |
|---|---|---|---|---|
| 1 | void | V | 該当しない（DATA-TYPE 項目の宣言だけでよい） | |
| 2 | byte | B | 英数字項目（1 けた） | PIC X |
| | | | 符号付き 2 進項目（1 バイト）※1 | PIC S9(1) USAGE COMP ～<br>PIC S9(2) USAGE COMP |
| 3 | char | C | 日本語項目（1 けた） | PIC N |
| | | | 符号なし 2 進項目（2 バイト）※2 | PIC 9(1) USAGE COMP ～<br>PIC 9(4) USAGE COMP<br>PIC 9(3) USAGE COMP-X ～<br>PIC 9(4) USAGE COMP-X |
| 4 | short | S | 符号付き 2 進項目（2 バイト）※2 | PIC S9(1) USAGE COMP ～<br>PIC S9(4) USAGE COMP |
| 5 | int | I | 符号付き 2 進項目（4 バイト） | PIC S9(5) USAGE COMP ～<br>PIC S9(9) USAGE COMP |
| 6 | long | J | 符号付き 2 進項目（8 バイト） | PIC S9(10) USAGE COMP ～<br>PIC S9(18) USAGE COMP |
| 7 | float | F | 単精度内部浮動小数点項目 | USAGE COMP-1 |
| 8 | double | D | 倍精度内部浮動小数点項目 | USAGE COMP-2 |
| 9 | boolean | Z | 英数字項目（1 けた） | PIC X |
| | | | 符号なし 2 進項目（1 バイト） | PIC 9(1) USAGE COMP-X ～<br>PIC 9(2) USAGE COMP-X |
| 10 | クラスオブジェクト | T クラス完全修飾名； | ポインタ項目 | USAGE POINTER |
| 11 | オブジェクト | L クラス完全修飾名； | ポインタ項目 | USAGE POINTER |
| 12 | 配列オブジェクト | [要素の型の文字列 | ポインタ項目 | USAGE POINTER |

注※1

1 バイト 2 進機能（-Bin1Byte コンパイラオプションを指定）を使用する場合です。

注※2

1 バイト 2 進機能を使用する場合，対応する COBOL の数字項目のけた数の範囲がこの表で示す値と異なる個所があります。1 バイト 2 進機能については，マニュアル「COBOL2002　言語　拡張仕様編」を参照してください。

COBOL プログラムでのパラメタ型集団項目の記述例を次に示します。

（例 1）int 型の場合

```
01 ARG.
   02  DATA-TYPE   PIC X(1)   VALUE 'I'.
   02  FILLER      PIC X(7)   VALUE ALL LOW-VALUE.
   02  DATA-AREA   PIC S9(9)  USAGE COMP.
```

（例 2）String 型のオブジェクトの場合

```
01 ARG.
   02  DATA-TYPE   PIC X(256) VALUE 'Ljava/lang/String;'.
   02  DATA-AREA   USAGE POINTER.
```

（例 3）String 型の配列オブジェクトの場合

```
01 ARG.
   02  DATA-TYPE   PIC X(256) VALUE '[Ljava/lang/String;'.
   02  DATA-AREA   USAGE POINTER.
```

基本データ型では，DATA-AREA 項目のデータ型を判定しないで，Java の型に対応するバイナリデータをそのまま受け渡します。対応する COBOL のデータ型は，その用途によって適切な型を選択してください。

- Java の char 型に対応する DATA-AREA 項目を日本語項目（PIC N）で定義したときも内容をそのまま受け渡します。COBOL で取り扱う文字コードとの変換はしません。
- boolean 型と英数字項目の値の対応を次の表に示します。

表 6-4　boolean 型と英数字項目の値

| boolean の値 | 英数字項目の値 |
|---|---|
| true | X'01' |
| false | X'00' |

## (4)　引数リスト型集団項目

Java のメソッドを呼び出す場合は，メソッドに渡す引数を引数リスト型集団項目で指定します。

引数リスト型集団項目は，パラメタ型集団項目を示すポインタ項目の繰り返し項目です。

終端の要素には NULL 値を設定する必要があります。このため，実際の引数の個数＋ 1 以上の繰り返し要素数を記述してください。

引数リスト型集団項目の例を次に示します。

引数なしのメソッドを呼び出す場合

```
      *>
      *> 引数リスト型（引数なし）の宣言
      *>
       01 NO-ARG.
          02 FILLER         USAGE POINTER VALUE NULL.

           CALL 'CBLJSTATICINVOKE'
               USING CBLJENV CLSREF sample NO-ARG RTN-1.
```

引数の個数が３のメソッドを呼び出す場合

```
      *>
      *> 引数リスト型（引数の個数が3）の宣言
      *>
       01 ARG-LIST.
          02  ARGPTR        USAGE POINTER OCCURS 4 TIMES.             *> 個数＋1
      *>
      *> 引数の宣言
      *>
       01 ARG-1.
          02  DATA-TYPE-1   PIC X(1)   VALUE 'I'.
          02  FILLER        PIC X(7)   VALUE ALL LOW-VALUE.
          02  DATA-AREA-1   PIC S9(9)  USAGE COMP.
       01 ARG-2.
          02  DATA-TYPE-2   PIC X(256) VALUE 'LString;'.
          02  DATA-AREA-2   USAGE POINTER.
       01 ARG-3.
          02  DATA-TYPE-3   PIC X(1)   VALUE 'Z'.
          02  FILLER        PIC X(7)   VALUE ALL LOW-VALUE.
          02  DATA-AREA-3   PIC X.

           COMPUTE ARGPTR(1) = FUNCTION ADDR( ARG-1 ).
           COMPUTE ARGPTR(2) = FUNCTION ADDR( ARG-2 ).
           COMPUTE ARGPTR(3) = FUNCTION ADDR( ARG-3 ).
           COMPUTE ARGPTR(4) = ZERO.                                    *> 引数リスト型の終
端

           CALL 'CBLJSTATICINVOKE'
               USING CBLJENV CLSREF sample ARG-LIST RTN-1.
```

## 6.1.2　基本操作

ここでは，Java プログラム呼び出し機能が提供する基本操作のサービスルーチンについて説明します。

## (1)　CBLJINITIALIZE

CBLJINITIALIZE サービスルーチンは，Java プログラム呼び出し機能を初期化します。

CBLJINITIALIZE サービスルーチンの動作を次に示します。

- Java VM をロードして Java 実行環境を作成します。

ほかのスレッドですでにJava実行環境が作成されている場合，その環境が利用できるようにします。

作成したJava実行環境の情報は，CBLJENV集団項目のCBLJENVCORE項目に設定されます。絶対に変更してはいけません。

- JNIへのアクセスを最小限に抑えるために，キャッシングの準備をします。

  Javaプログラム呼び出し機能自身が使用するクラスとそのフィールドおよびメソッドは，この初期化のタイミングで情報を取得します。

  また，一度使用したクラス，フィールド，メソッドの情報をキャッシュするためのハッシュを準備します。

Javaプログラム呼び出し機能の初期化は，各スレッドでJavaプログラム呼び出し機能を使用する前に実行する必要があります。

ただし，次に示すサービスルーチンは，CBLJENV集団項目のCBLJENVCORE項目がNULLの場合に，CBLJINITIALIZEサービスルーチンを自動的に呼び出します。このため，これらのサービスルーチンをスレッドの最初に呼び出す場合は，CBLJINITIALIZEサービスルーチンを明示的に呼び出す必要はありません。

- CBLJGETCLASSサービスルーチン
- CBLJXTOSTRINGサービスルーチン
- CBLJNTOSTRINGサービスルーチン
- CBLJNEWARRAYサービスルーチン

また，CBLJINITIALIZEサービスルーチンは，CBLJENV集団項目のCBLJENVCORE項目がNULLでない場合は何もしないで終了するため，何度呼び出してもかまいません。

CBLJINITIALIZEサービスルーチンが明示的または暗黙的に呼び出されるまでに，CBLJENV集団項目のCBLJVMOPTIONS項目に，VALUE句またはMOVE文でJava VMの起動オプションを設定してください。

VALUE句で*n*個の起動オプションを設定するには，次のように記述します。

```
02  CBLJVMOPTIONS.
  03 CBLJOPTCOUNT  PIC S9(9) USAGE COMP VALUE n.
  03 CBLJOPTION-1  PIC X(m)  VALUE 'オプション文字列'.
  03 CBLJOPTION-2  PIC X(m)  VALUE 'オプション文字列'.
     :
  03 CBLJOPTION-n  PIC X(m)  VALUE 'オプション文字列'.
```

CBLJVMOPTIONS項目のCBLJOPTION-n項目に指定できるオプションについては，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV集団項目」を参照してください。ただし，ほかのスレッドのCOBOL以外のプログラムで，すでにJava VMが起動されている場合は，CBLJVMOPTIONS項目に指定されたオプションは適用されません。

### 形式

```
CALL ’CBLJINITIALIZE’ USING  引数1.
```

### 引数

引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。

### 戻り値

RETURN-CODE 特殊レジスタには常に 0 を返します。

初期化に失敗した場合は実行時エラーとなります。

### 注意事項

- Java プログラム呼び出し機能は，32bit 版 COBOL2002 の場合，JDK1.5 以上を，64bit 版 COBOL2002 の場合，JDK1.6 以上を前提とします。

  64bit 版 COBOL2002 では前提バージョンより古い JDK を使用した場合，明示的または暗黙的にこのサービスルーチンが呼び出されたときに，実行時エラー（詳細メッセージ番号 0110）となります。32bit 版 COBOL2002 では前提バージョンより古い JDK を使用した場合でも実行時エラーとなりません。

- 同じプロセスで作成できる Java 実行環境は一度だけです。すでに Java 実行環境を削除しているプロセスでは，このサービスルーチンで Java 実行環境を作成できないため，実行時エラー（詳細メッセージ番号 0111）となります。

## (2)　CBLJGETCLASS

CBLJGETCLASS サービスルーチンは，Java クラスを Java 実行環境（Java VM）上にロードして使用できるようにし，その Java クラス参照を返します。すでにロードされている場合は，Java クラス参照だけを返します。

### 形式

```
CALL ’CBLJGETCLASS’ USING  引数1  引数2  引数3.
```

### 引数

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。
- 引数 2 には，クラス名を持つ動的長基本項目または名前型集団項目を指定します。名前型集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(2)　名前型集団項目」を参照してください。
- 引数 3 には，Java クラス参照を受け取るポインタ項目を指定します。

### 戻り値

RETURN-CODE 特殊レジスタには常に 0 を返します。

クラスのロードに失敗した場合は実行時エラーとなります。

## (3) CBLJSETSTATICFIELD

CBLJSETSTATICFIELD サービスルーチンは，Java クラス参照の指す Java クラスのクラスフィールドに値を設定します。

### 形式

```
CALL ’CBLJSETSTATICFIELD’ USING 引数1 引数2 引数3 引数4.
```

### 引数

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1 サービスルーチンで使用する引数」の「(1) CBLJENV 集団項目」を参照してください。
- 引数 2 には，Java クラス参照を持つポインタ項目を指定します。
- 引数 3 には，クラスフィールド名を持つ動的長基本項目または名前型集団項目を指定します。名前型集団項目については，「6.1.1 サービスルーチンで使用する引数」の「(2) 名前型集団項目」を参照してください。
- 引数 4 には，設定値を持つパラメタ型集団項目を指定します。パラメタ型集団項目については，「6.1.1 サービスルーチンで使用する引数」の「(3) パラメタ型集団項目」を参照してください。

### 戻り値

RETURN-CODE 特殊レジスタには常に 0 を返します。

失敗した場合は実行時エラーとなります。

## (4) CBLJGETSTATICFIELD

CBLJGETSTATICFIELD サービスルーチンは，Java クラス参照の指す Java クラスのクラスフィールドの値を取り出します。

### 形式

```
CALL ’CBLJGETSTATICFIELD’ USING 引数1 引数2 引数3 引数4.
```

### 引数

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1 サービスルーチンで使用する引数」の「(1) CBLJENV 集団項目」を参照してください。
- 引数 2 には，Java クラス参照を持つポインタ項目を指定します。
- 引数 3 には，クラスフィールド名を持つ動的長基本項目または名前型集団項目を指定します。名前型集団項目については，「6.1.1 サービスルーチンで使用する引数」の「(2) 名前型集団項目」を参照してください。
- 引数 4 には，値を受け取るパラメタ型集団項目を指定します。パラメタ型集団項目については，「6.1.1 サービスルーチンで使用する引数」の「(3) パラメタ型集団項目」を参照してください。

### 戻り値

RETURN-CODE 特殊レジスタには常に 0 を返します。

失敗した場合は実行時エラーとなります。

### 注意事項

クラスフィールドの型が Class インスタンスの場合，引数 4 の Java の型には java/lang/Class ではなく，実体のクラス名を指定してください。指定した Java の型と実体のクラス名が一致しない場合，実行時エラーとなります。

## (5) CBLJSTATICINVOKE

CBLJSTATICINVOKE サービスルーチンは，Java クラス参照を使って，Java のクラスメソッドを呼び出します。

### 形式

```
CALL ’CBLJSTATICINVOKE’ USING  引数1  引数2  引数3  引数4  引数5.
```

### 引数

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。
- 引数 2 には，Java クラス参照を持つポインタ項目を指定します。
- 引数 3 には，クラスメソッド名を持つ動的長基本項目または名前型集団項目を指定します。名前型集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(2)　名前型集団項目」を参照してください。
- 引数 4 には，クラスメソッドに渡す引数リストを指定します。引数なしのクラスメソッドを呼び出す場合も指定が必要です。引数リストについては，「6.1.1　サービスルーチンで使用する引数」の「(4)　引数リスト型集団項目」を参照してください。
- 引数 5 には，返却値を受け取る取るパラメタ型集団項目を指定します。void 型のクラスメソッドを呼び出す場合も指定が必要です。パラメタ型集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(3)　パラメタ型集団項目」を参照してください。

### 戻り値

RETURN-CODE 特殊レジスタに 0（正常リターン）または 1（例外リターン）を返します。

呼び出しができなかった場合や致命的な例外（java/lang/Error のサブクラス）がスローされた場合は実行時エラーとなります。

### 備考

RETURN-CODE が 1 の場合，CBLJENV 引数の CBLJEXCEPTION 項目に Java プログラムからスローされた例外オブジェクトが格納されています。これを使ってエラーの詳細を確認できます。

### 注意事項

返却値の型が Class インスタンスの場合，引数 5 の Java の型には java/lang/Class ではなく，実体のクラス名を指定してください。指定した Java の型と実体のクラス名が一致しない場合，実行時エラーとなります。

## (6) CBLJNEW

CBLJNEW サービスルーチンは，Java クラス参照の指す Java クラスのインスタンスを生成して，コンストラクタを呼び出し，Java インスタンスのオブジェクト参照（Java オブジェクト参照）を返します。

取得した Java オブジェクト参照の扱いは，必ず「3.2.5　Java オブジェクト参照の使用ガイドライン」に従ってください。

### 形式

```
CALL 'CBLJNEW' USING  引数1  引数2  引数3  引数4.
```

### 引数

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。
- 引数 2 には，Java クラス参照を持つポインタ項目を指定します。
- 引数 3 には，コンストラクタに渡す引数リストを指定します。引数なしのコンストラクタを呼び出す場合も指定が必要です。引数リストについては，「6.1.1　サービスルーチンで使用する引数」の「(4)　引数リスト型集団項目」を参照してください。
- 引数 4 には，Java オブジェクト参照を受け取るポインタ項目を指定します。

### 戻り値

RETURN-CODE 特殊レジスタに 0（成功）または 1（例外発生）を返します。

コンストラクタの呼び出しができなかった場合や致命的な例外（java/lang/Error のサブクラス）がスローされた場合は，実行時エラーとなります。

### 備考

RETURN-CODE が 1 の場合，CBLJENV 引数の CBLJEXCEPTION 項目に Java プログラムからスローされた例外オブジェクトが格納されています。これを使ってエラーの詳細を確認できます。

### 注意事項

- 取得した Java オブジェクト参照は COMPUTE 文や SET 文では転記しないでください。
- 不要になった Java オブジェクト参照は，必ず CBLJRELEASE または CBLJSETNULL サービスルーチンで解放してください。

## (7) CBLJSETFIELD

CBLJSETFIELD サービスルーチンは，Java オブジェクト参照の指す Java インスタンスのインスタンスフィールドに値を設定します。

### 形式

```
CALL 'CBLJSETFIELD' USING  引数1  引数2  引数3  引数4.
```

### 引数

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。
- 引数 2 には，Java オブジェクト参照を持つポインタ項目を指定します。
- 引数 3 には，インスタンスフィールド名を持つ動的長基本項目または名前型集団項目を指定します。名前型集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(2)　名前型集団項目」を参照してください。
- 引数 4 には，設定値を持つパラメタ型集団項目を指定します。パラメタ型集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(3)　パラメタ型集団項目」を参照してください。

### 戻り値

RETURN-CODE 特殊レジスタには常に 0 を返します。

失敗した場合は実行時エラーとなります。

## (8)　CBLJGETFIELD

CBLJGETFIELD サービスルーチンは，Java オブジェクト参照の指す Java インスタンスのインスタンスフィールドの値を取り出します。

### 形式

```
CALL ’CBLJGETFIELD’ USING　引数1　引数2　引数3　引数4.
```

### 引数

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。
- 引数 2 には，Java オブジェクト参照を持つポインタ項目を指定します。
- 引数 3 には，インスタンスフィールド名を持つ動的長基本項目または名前型集団項目を指定します。名前型集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(2)　名前型集団項目」を参照してください。
- 引数 4 には，値を受け取るパラメタ型集団項目を指定します。パラメタ型集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(3)　パラメタ型集団項目」を参照してください。

### 戻り値

RETURN-CODE 特殊レジスタには常に 0 を返します。

失敗した場合は実行時エラーとなります。

### 注意事項

クラスフィールドの型が Class インスタンスの場合，引数 4 の Java の型には java/lang/Class ではなく，実体のクラス名を指定してください。指定した Java の型と実体のクラス名が一致しない場合，実行時エラーとなります。

## (9)　CBLJINVOKE

CBLJINVOKE サービスルーチンは，Java オブジェクト参照を使って，Java のインスタンスメソッドを呼び出します。

### 形式

```
CALL ’CBLJINVOKE’ USING　引数1　引数2　引数3　引数4　引数5.
```

### 引数

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。
- 引数 2 には，Java オブジェクト参照を持つポインタ項目を指定します。
- 引数 3 には，インスタンスメソッド名を持つ動的長基本項目または名前型集団項目を指定します。名前型集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(2)　名前型集団項目」を参照してください。
- 引数 4 には，インスタンスメソッドに渡す引数リストを指定します。引数なしのインスタンスメソッドを呼び出す場合も指定が必要です。引数リストについては，「6.1.1　サービスルーチンで使用する引数」の「(4)　引数リスト型集団項目」を参照してください。
- 引数 5 には，返却値を受け取る取るパラメタ型集団項目を指定します。void 型のインスタンスメソッドを呼び出す場合も指定が必要です。パラメタ型集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(3)　パラメタ型集団項目」を参照してください。

### 戻り値

RETURN-CODE 特殊レジスタに 0（正常リターン）または 1（例外リターン）を返します。

呼び出しができなかった場合や致命的な例外（java/lang/Error のサブクラス）がスローされた場合は実行時エラーとなります。

### 備考

RETURN-CODE が 1 の場合，引数 1 に指定した CBLJENV 集団項目の CBLJEXCEPTION 項目に Java プログラムからスローされた例外オブジェクトが格納されます。これを使ってエラーの詳細を確認できます。

### 注意事項

返却値の型が Class インスタンスの場合，引数 5 の Java の型には java/lang/Class ではなく，実体のクラス名を指定してください。指定した Java の型と実体のクラス名が一致しない場合，実行時エラーとなります。

## (10)　CBLJRELEASE

CBLJRELEASE サービスルーチンは，Java オブジェクト参照を解放します。

不要になった Java オブジェクト参照は，CBLJRELEASE サービスルーチンまたは CBLJSETNULL サービスルーチンで解放する必要があります。

形式

```
CALL ’CBLJRELEASE’ USING  引数1  引数2.
```

引数

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。
- 引数 2 には，Java オブジェクト参照を持つポインタ項目を指定します。

戻り値

RETURN-CODE 特殊レジスタには常に 0 を返します。

注意事項

- 引数 2 のポインタ項目に NULL を指定しないでください。
- Java クラス参照は解放できません。

## (11)　CBLJFINALIZE

CBLJFINALIZE サービスルーチンは，Java プログラム呼び出し機能の実行環境を削除します。

形式

```
CALL ’CBLJFINALIZE’ USING  引数1.
```

引数

引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。

戻り値

RETURN-CODE 特殊レジスタには常に 0 を返します。

環境の削除に失敗した場合は実行時エラーとなります。

注意事項

このサービスルーチンを実行したあとに，Java クラス参照や Java オブジェクト参照を使って，ほかのサービスルーチンを呼び出さないでください。

# 6.1.3　クラス操作

ここでは，Java プログラム呼び出し機能が提供するクラス操作のサービスルーチンについて説明します。

## (1)　CBLJGETOBJCLASS

CBLJGETOBJCLASS サービスルーチンは，Java のインスタンスの属する Java クラスの Java クラス参照を返します。

形式

```
CALL ’CBLJGETOBJCLASS’ USING　引数1　引数2　引数3.
```

引数

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。
- 引数 2 には，Java オブジェクト参照を持つポインタ項目を指定します。
- 引数 3 には，Java クラス参照を受け取るポインタ項目を指定します。

戻り値

RETURN-CODE 特殊レジスタには常に 0 を返します。

Java クラス参照の取得に失敗した場合は実行時エラーとなります。

## (2)　CBLJGETNAME

CBLJGETNAME サービスルーチンは，java/lang/Class.getName メソッドを呼び出して，Java クラス参照の示す Java クラスの名前（パッケージ名とクラス名の区切りはピリオド）を返します。

形式

```
CALL ’CBLJGETNAME’ USING　引数1　引数2　引数3　引数4.
```

引数

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。
- 引数 2 には，Java クラス参照を持つポインタ項目を指定します。
- 引数 3 には，クラス名を受け取る英数字項目を指定します。
- 引数 4 には，クラス名を受け取る英数字項目のサイズ（文字数）を持つ，4 バイト 2 進形式の数字項目を指定します。

戻り値

RETURN-CODE 特殊レジスタに 0（正常）または 1（サイズが不足）を返します。

名前の取得に失敗した場合は実行時エラーとなります。

注意事項

RETURN-CODE が 1 の場合，引数 3 に指定した項目に，引数 4 に指定した長さまでの Java クラスの名前を設定します。

## (3)　CBLJGETSUPERCLASS

CBLJGETSUPERCLASS サービスルーチンは，Java クラス参照の示す Java クラスのスーパークラスの Java クラス参照を返します。

形式

```
CALL ’CBLJGETSUPERCLASS’ USING　引数1　引数2　引数3.
```

引数

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。
- 引数 2 には，Java クラス参照を持つポインタ項目を指定します。
- 引数 3 には，スーパークラスの Java クラス参照を受け取るポインタ項目を指定します。

戻り値

RETURN-CODE 特殊レジスタには常に 0 を返します。

スーパークラスの取得に失敗した場合は実行時エラーとなります。

## 6.1.4　オブジェクト操作

ここでは，Java プログラム呼び出し機能が提供するオブジェクト操作のサービスルーチンについて説明します。

### (1)　CBLJCLASSNAME

CBLJCLASSNAME サービスルーチンは，Java のインスタンスの属する Java クラスの名前（パッケージ名とクラス名の区切りはピリオド）を返します。

形式

```
CALL ’CBLJCLASSNAME’ USING　引数1　引数2　引数3　引数4.
```

引数

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。
- 引数 2 には，Java オブジェクト参照を持つポインタ項目を指定します。
- 引数 3 には，クラス名を受け取る英数字項目を指定します。
- 引数 4 には，クラス名を受け取る英数字項目のサイズ（文字数）を持つ，4 バイト 2 進形式の数字項目を指定します。

戻り値

RETURN-CODE 特殊レジスタに 0（正常）または 1（サイズが不足）を返します。

名前の取得に失敗した場合は実行時エラーとなります。

注意事項

RETURN-CODE が 1 の場合，引数 3 に指定した項目に，引数 4 に指定した長さまでの Java クラスの名前を設定します。

## (2) CBLJINSTANCEOF

CBLJINSTANCEOF サービスルーチンは，Java のインスタンスが Java クラス参照で指定されたクラスのインスタンスまたはそのサブクラスのインスタンスである場合，1 を返します。そうでない場合，0 を返します。

### 形式

```
CALL ’CBLJINSTANCEOF’ USING　引数1　引数2　引数3.
```

### 引数

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。
- 引数 2 には，Java オブジェクト参照を持つポインタ項目を指定します。
- 引数 3 には，Java クラス参照を持つポインタ項目を指定します。

### 戻り値

RETURN-CODE 特殊レジスタに 1（真）または 0（偽）を返します。

判定自体に失敗した場合は実行時エラーとなります。

## (3) CBLJSAMEOBJECT

CBLJSAMEOBJECT サービスルーチンは，二つの Java オブジェクト参照が同じ Java インスタンスを指す場合，1 を返します。そうでない場合，0 を返します。

### 形式

```
CALL ’CBLJSAMEOBJECT’ USING　引数1　引数2　引数3.
```

### 引数

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。
- 引数 2 には，Java オブジェクト参照を持つポインタ項目を指定します。
- 引数 3 には，Java オブジェクト参照を持つポインタ項目を指定します。

### 戻り値

RETURN-CODE 特殊レジスタに 1（真）または 0（偽）を返します。

判定自体に失敗した場合は実行時エラーとなります。

## (4) CBLJEQUAL

CBLJEQUAL サービスルーチンは，二つの Java オブジェクト参照の指す Java インスタンスが同じ値を持つ場合，1 を返します。そうでない場合，0 を返します。

二つの文字列オブジェクトが同じ文字列を持つかをチェックする場合などで使用します。

形式

```
CALL ’CBLJEQUAL’ USING  引数1  引数2  引数3.
```

引数

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。
- 引数 2 には，Java オブジェクト参照を持つポインタ項目を指定します。
- 引数 3 には，Java オブジェクト参照を持つポインタ項目を指定します。

戻り値

RETURN-CODE 特殊レジスタに 1（真）または 0（偽）を返します。

判定自体に失敗した場合は実行時エラーとなります。

注意事項

Java インスタンスが同じ値を持つかどうかの判定は，Java インスタンスの equals メソッドの結果に従います。

## (5)　CBLJCOPY

CBLJCOPY サービスルーチンは，引数 2 の Java オブジェクト参照の複製を作成して，引数 3 に格納します。

その際，引数 4 が NULL ポインタでなければ，引数 2 のインスタンスが引数 4 の型（クラスまたはインタフェース）にキャストできるかどうかをチェックします。キャストできない場合，複製は作成しないで，引数 3 に NULL ポインタを設定します。

引数 4 が NULL ポインタなら，無条件に引数 2 の Java オブジェクト参照の複製を引数 3 に格納します。

引数 3 に Java オブジェクト参照が格納されている場合は，解放してから格納します。

形式

```
CALL ’CBLJCOPY’ USING  引数1  引数2  引数3  引数4.
```

引数

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。
- 引数 2 には，Java オブジェクト参照を持つポインタ項目を指定します。
- 引数 3 には，Java オブジェクト参照を受け取るポインタ項目を指定します。
- 引数 4 には，Java クラス参照を持つポインタ項目を指定します。引数 3 に格納可能なオブジェクトの型（クラスまたはインタフェース）を Java クラス参照で指定します。

戻り値

RETURN-CODE 特殊レジスタに 0（コピー成功）または 1（コピー失敗）を返します。

キャストできるかどうかのチェック自体ができなかった場合は，実行時エラーとなります。

**注意事項**

Java オブジェクト参照は，COMPUTE 文や SET 文では転記しないで，必ず CBLJCOPY サービスルーチンで複製を作成して転記してください。

Java オブジェクト参照の扱いは，必ず Java オブジェクト参照の使用ガイドラインに従ってください。Java オブジェクト参照の使用ガイドラインについては，「3.2.5　Java オブジェクト参照の使用ガイドライン」を参照してください。

## (6)　CBLJSETNULL

CBLJSETNULL サービスルーチンは，Java オブジェクト参照用のポインタ項目に NULL を設定します。Java オブジェクト参照が格納されている場合は，解放してから NULL を設定します。CBLJSETNULL サービスルーチンは，ポインタ項目に NULL や不正な Java オブジェクト参照が入っていてもエラーとならないことを除けば，動作は CBLJRELEASE サービスルーチンと同じです。

**形式**

```
CALL ’CBLJSETNULL’ USING　引数1　引数2.
```

**引数**

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。
- 引数 2 には，NULL を設定する Java オブジェクト参照用のポインタ項目を指定します。ポインタ項目に NULL が入っていてもかまいません。

**戻り値**

RETURN-CODE 特殊レジスタには常に 0 を返します。

**注意事項**

手続き部で Java オブジェクト参照用のポインタ項目に NULL を設定する場合は，COMPUTE 文，SET 文または INITIALIZE 文を使用しないで，必ず CBLJSETNULL サービスルーチンを使用してください。

Java オブジェクト参照の扱いは，必ず Java オブジェクト参照の使用ガイドラインに従ってください。Java オブジェクト参照の使用ガイドラインについては，「3.2.5　Java オブジェクト参照の使用ガイドライン」を参照してください。

# 6.1.5　文字列オブジェクト操作

ここでは，Java プログラム呼び出し機能が提供する文字列オブジェクト操作のサービスルーチンについて説明します。

## (1) CBLJXTOSTRING

CBLJXTOSTRING サービスルーチンは，英数字項目のデータを変換して String オブジェクト（文字コードは UTF-16）を生成して，その Java オブジェクト参照を返します。

取得した String オブジェクトの Java オブジェクト参照の扱いは，必ず Java オブジェクト参照の使用ガイドラインに従ってください。Java オブジェクト参照の使用ガイドラインについては，「3.2.5 Java オブジェクト参照の使用ガイドライン」を参照してください。

英数字項目のデータの文字コードは，COBOL2002 で実行時に使用する文字コードに従います。Unicode 機能と同時に使用する場合，英数字項目のデータの文字コードは UTF-8 として扱います。COBOL2002 の実行時環境変数 CBLLANG の値と英数字項目のデータの文字コードの関係を次の表に示します。

### 表 6-5 環境変数 CBLLANG の値と英数字項目のデータの文字コード

| 環境変数 CBLLANG の値 | 英数字項目のデータの文字コード |
|---|---|
| 指定なし，または UNICODE 以外 | シフト JIS |
| UNICODE | UTF-8 |

COBOL で使用する文字集合については，マニュアル「COBOL2002 ユーザーズガイド」を参照してください。

#### 形式

```
CALL 'CBLJXTOSTRING' USING 引数1 引数2 引数3 引数4.
```

#### 引数

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1 サービスルーチンで使用する引数」の「(1) CBLJENV 集団項目」を参照してください。
- 引数 2 には，英数字項目を指定します。
- 引数 3 には，英数字項目のサイズ（バイト数）を持つ 4 バイト 2 進形式の数字項目を指定します。
- 引数 4 には，Java オブジェクト参照を受け取るポインタ項目を指定します。

#### 戻り値

RETURN-CODE 特殊レジスタには常に 0 を返します。

String オブジェクトの生成に失敗した場合は実行時エラーとなります。

#### 規則

- 英数字データの終わりの連続した半角空白文字は無視されます。
- 英数字項目のデータの末尾が 2 バイト文字の 1 バイト目である場合は無視されます。

## (2) CBLJNTOSTRING

CBLJNTOSTRING サービスルーチンは，日本語項目のデータを変換して String オブジェクト（文字コードは UTF-16）を生成し，その Java オブジェクト参照を返します。

取得した String オブジェクトの Java オブジェクト参照の扱いは，必ず Java オブジェクト参照の使用ガイドラインに従ってください。Java オブジェクト参照の使用ガイドラインについては，「3.2.5 Java オブジェクト参照の使用ガイドライン」を参照してください。

日本語項目のデータの文字コードは，COBOL2002 で実行時に使用する文字コードに従います。Unicode 機能と同時に使用する場合，日本語項目のデータの文字コードは UTF-16 として扱います。COBOL2002 の実行時環境変数 CBLLANG の値と日本語項目のデータの文字コードの関係を次の表に示します。

### 表 6-6 環境変数 CBLLANG の値と日本語項目のデータの文字コード

| 環境変数 CBLLANG の値 | 日本語項目のデータの文字コード |
|---|---|
| 指定なし，または UNICODE 以外 | シフト JIS |
| UNICODE | UTF-16 |

COBOL で使用する文字集合については，マニュアル「COBOL2002 ユーザーズガイド」を参照してください。

#### 形式

```
CALL ’CBLJNTOSTRING’ USING 引数1 引数2 引数3 引数4.
```

#### 引数

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1 サービスルーチンで使用する引数」の「(1) CBLJENV 集団項目」を参照してください。
- 引数 2 には，日本語項目を指定します。
- 引数 3 には，日本語項目のサイズ（文字数）を持つ 4 バイト 2 進形式の数字項目を指定します。
- 引数 4 には，Java オブジェクト参照を受け取るポインタ項目を指定します。

#### 戻り値

RETURN-CODE 特殊レジスタには常に 0 を返します。

String オブジェクトの生成に失敗した場合は実行時エラーとなります。

#### 規則

日本語データの終わりの連続した全角空白文字は無視されます。

## (3) CBLJSTRINGTOX

CBLJSTRINGTOX サービスルーチンは，String オブジェクトの文字列（文字コードは UTF-16）を変換して英数字項目に格納します。

String オブジェクトの文字列は，COBOL2002 で実行時に使用する文字コードに従って変換されます。
Unicode 機能と同時に使用する場合，英数字項目のデータの文字コードは UTF-8 となります。
COBOL2002 の実行時環境変数 CBLLANG の値と英数字項目のデータの文字コードの関係を次の表に示します。

表 6-7　環境変数 CBLLANG の値と英数字項目のデータの文字コード

| 環境変数 CBLLANG の値 | 英数字項目のデータの文字コード |
|---|---|
| 指定なし，または UNICODE 以外 | シフト JIS |
| UNICODE | UTF-8 |

COBOL で使用する文字集合については，マニュアル「COBOL2002　ユーザーズガイド」を参照してください。

**形式**

```
CALL ’CBLJSTRINGTOX’ USING　引数1　引数2　引数3　引数4.
```

**引数**

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。
- 引数 2 には，String オブジェクトの Java オブジェクト参照を持つポインタ項目を指定します。
- 引数 3 には，英数字データを受け取る英数字項目を指定します。
- 引数 4 には，英数字項目のサイズ（バイト数）を持つ 4 バイト 2 進形式の数字項目を指定します。

**戻り値**

RETURN-CODE 特殊レジスタには常に 0 を返します。
データ変換に失敗した場合は実行時エラーとなります。

**規則**

- 英数字項目の余った部分には半角空白文字が挿入されます。
- 英数字項目の末尾の文字が 2 バイト文字の 1 バイト目になる場合は半角空白が設定されます。

## (4)　CBLJSTRINGTON

CBLJSTRINGTON サービスルーチンは，String オブジェクトの文字列（文字コードは UTF-16）を変換して日本語項目に格納します。

String オブジェクトの文字列は，COBOL2002 で実行時に使用する文字コードに従って変換されます。
Unicode 機能と同時に使用する場合，日本語項目のデータの文字コードは UTF-16 となります。
COBOL2002 の実行時環境変数 CBLLANG の値と日本語項目のデータの文字コードの関係を次の表に示します。

表 6-8　環境変数 CBLLANG の値と日本語項目のデータの文字コード

| 環境変数 CBLLANG の値 | 日本語項目のデータの文字コード |
|---|---|
| 指定なし，または UNICODE 以外 | シフト JIS |
| UNICODE | UTF-16 |

COBOL で使用する文字集合については，マニュアル「COBOL2002　ユーザーズガイド」を参照してください。

**形式**

```
CALL ’CBLJSTRINGTON’ USING　引数1　引数2　引数3　引数4.
```

**引数**

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。
- 引数 2 には，String オブジェクトの Java オブジェクト参照を持つポインタ項目を指定します。
- 引数 3 には，日本語データを受け取る日本語項目を指定します。
- 引数 4 には，日本語項目のサイズ（文字数）を持つ 4 バイト 2 進形式の数字項目を指定します。

**戻り値**

RETURN-CODE 特殊レジスタには常に 0 を返します。

データ変換に失敗した場合は実行時エラーとなります。

**規則**

日本語項目の余った部分には全角空白文字が挿入されます。

**注意事項**

Unicode 機能を使用していない場合，Java プログラムから渡された String オブジェクトのデータの内容を変換したときに，1 バイトとなる文字が含まれているときの動作は保証しません。

## (5)　CBLJSTRLENGTH

CBLJSTRLENGTH サービスルーチンは，String オブジェクトの文字列の文字数を返します。

**形式**

```
CALL ’CBLJSTRLENGTH’ USING　引数1　引数2　引数3.
```

**引数**

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。
- 引数 2 には，String オブジェクトの Java オブジェクト参照を持つポインタ項目を指定します。
- 引数 3 には，文字数を受け取る 4 バイト 2 進形式の数字項目を指定します。

戻り値

RETURN-CODE 特殊レジスタには常に 0 を返します。

長さの取得に失敗した場合は実行時エラーとなります。

## (6) CBLJDISPLAY

CBLJDISPLAY サービスルーチンは，String オブジェクトの内容を Java プログラムの標準出力に出力します。

形式

```
CALL ’CBLJDISPLAY’ USING  引数1  引数2.
```

引数

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1 サービスルーチンで使用する引数」の「(1) CBLJENV 集団項目」を参照してください。
- 引数 2 には，String オブジェクトの Java オブジェクト参照を持つポインタ項目を指定します。

戻り値

RETURN-CODE 特殊レジスタには常に 0 を返します。

内容の出力に失敗した場合は実行時エラーとなります。

注意事項

CBLJDISPLAY サービスルーチンは，System.out.println で Java の標準出力にデータを出力します。COBOL プログラムの出力先に String オブジェクトの内容を出力したい場合は，英数字項目または日本語項目にデータを取得したあと，DISPLAY 文を使用して出力してください。

# 6.1.6 配列オブジェクト操作

ここでは，Java プログラム呼び出し機能が提供する配列オブジェクト操作のサービスルーチンについて説明します。

## (1) CBLJNEWARRAY

CBLJNEWARRAY サービスルーチンは，配列オブジェクトのインスタンスを生成して，その Java オブジェクト参照を返します。

取得した Java オブジェクト参照の扱いは，必ず Java オブジェクト参照の使用ガイドラインに従ってください。Java オブジェクト参照の使用ガイドラインについては，「3.2.5 Java オブジェクト参照の使用ガイドライン」を参照してください。

形式

```
CALL ’CBLJNEWARRAY’ USING  引数1  引数2  引数3  引数4.
```

### 引数

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。
- 引数 2 には，作成する配列の型を示す文字列を英数字項目で指定します。英数字項目のサイズは，CBLJENV 集団項目の CBLJSTRMAXLEN 項目に指定した値と同じ値を指定してください。

  配列の型と配列の型を示す文字列を次の表に示します。

  表 6-9　配列の型と配列の型を示す文字列

| 項番 | 配列の型 | 配列の型を示す文字列 |
|---|---|---|
| 1 | byte 配列 | [B |
| 2 | char 配列 | [C |
| 3 | short 配列 | [S |
| 4 | int 配列 | [I |
| 5 | long 配列 | [J |
| 6 | float 配列 | [F |
| 7 | double 配列 | [D |
| 8 | boolean 配列 | [Z |
| 9 | オブジェクト配列 | [L クラス完全修飾名； |
| 10 | 配列オブジェクトの配列 | [配列の型の文字列 |

  2 次元文字列配列の指定例を次に示します。

  例）

```
      [[Ljava/lang/String;
```

- 引数 3 には，作成する配列の要素数を持つ 4 バイト 2 進形式の数字項目を指定します。
- 引数 4 には，配列オブジェクトの Java オブジェクト参照を受け取るポインタ項目を指定します。

### 戻り値

RETURN-CODE 特殊レジスタには常に 0 を返します。

配列オブジェクトの生成に失敗した場合は実行時エラーとなります。

## (2)　CBLJARRAYLENGTH

CBLJARRAYLENGTH サービスルーチンは，配列オブジェクトの要素数を返します。

### 形式

```
CALL ’CBLJARRAYLENGTH’ USING　引数1　引数2　引数3.
```

### 引数

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。
- 引数 2 には，配列オブジェクトの Java オブジェクト参照を持つポインタ項目を指定します。
- 引数 3 には，要素数を受け取る 4 バイト 2 進形式の数字項目を指定します。

### 戻り値

RETURN-CODE 特殊レジスタには常に 0 を返します。

長さの取得に失敗した場合は実行時エラーとなります。

## (3)　CBLJSETOBJARRAY

CBLJSETOBJARRAY サービスルーチンは，オブジェクト型の配列オブジェクトのインデクスが示す要素に，オブジェクトを格納します。

### 形式

```
CALL ’CBLJSETOBJARRAY’ USING　引数1　引数2　引数3　引数4.
```

### 引数

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。
- 引数 2 には，配列オブジェクトの Java オブジェクト参照を持つポインタ項目を指定します。
- 引数 3 には，インデクスを持つ 4 バイト 2 進形式の数字項目を指定します。
- 引数 4 には，格納する Java オブジェクト参照を持つポインタ項目を指定します。

### 戻り値

RETURN-CODE 特殊レジスタには常に 0 を返します。

設定に失敗した場合は実行時エラーとなります。

### 注意事項

引数 3 には，Java の要素のインデクス（0～（配列の要素の数-1））を指定する必要があります。

## (4)　CBLJGETOBJARRAY

CBLJGETOBJARRAY サービスルーチンは，オブジェクト型の配列オブジェクトのインデクスの示す要素からオブジェクトを取り出します。

取り出した Java オブジェクト参照の扱いは，必ず Java オブジェクト参照の使用ガイドラインに従ってください。Java オブジェクト参照の使用ガイドラインについては，「3.2.5　Java オブジェクト参照の使用ガイドライン」を参照してください。

形式

```
CALL ’CBLJGETOBJARRAY’ USING　引数1　引数2　引数3　引数4.
```

引数

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。
- 引数 2 には，配列オブジェクトの Java オブジェクト参照を持つポインタ項目を指定します。
- 引数 3 には，インデクスを持つ 4 バイト 2 進形式の数字項目を指定します。
- 引数 4 には，Java オブジェクト参照を受け取るポインタ項目を指定します。

戻り値

RETURN-CODE 特殊レジスタには常に 0 を返します。

取り出しに失敗した場合は実行時エラーとなります。

注意事項

引数 3 には，Java の要素のインデクス（0～（配列の要素の数-1））を指定する必要があります。

## (5)　CBLJGETARRAYADDR

CBLJGETARRAYADDR サービスルーチンは，基本型の配列オブジェクトから基本型配列のアドレスを取得します。

操作が完了した基本型配列の内容は CBLJRELEASEARRAY サービスルーチンで配列オブジェクトに反映します。

形式

```
CALL ’CBLJGETARRAYADDR’ USING　引数1　引数2　引数3.
```

引数

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。
- 引数 2 には，配列オブジェクトの Java オブジェクト参照を持つポインタ項目を指定します。
- 引数 3 には，基本型配列のアドレスを受け取るポインタ項目を指定します。

戻り値

RETURN-CODE 特殊レジスタには常に 0 を返します。

取得に失敗した場合は実行時エラーとなります。

注意事項

配列の値を変更しない場合も，取得した配列のアドレスが不要になったときに CBLJRELEASEARRAY サービスルーチンを呼び出してください。呼び出さない場合は，領域が解放されないでメモリ不足の要因となることがあります。

## (6) CBLJRELEASEARRAY

CBLJRELEASEARRAY サービスルーチンは，CBLJGETARRAYADDR で取得した基本型配列の内容変更を配列オブジェクトに反映させて，基本型配列の領域を解放します。

### 形式

```
CALL ’CBLJRELEASEARRAY’ USING　引数1　引数2　引数3.
```

### 引数

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。
- 引数 2 には，配列オブジェクトの Java オブジェクト参照を持つポインタ項目を指定します。
- 引数 3 には，基本型配列のアドレスを持つポインタ項目を指定します。

### 戻り値

RETURN-CODE 特殊レジスタには常に 0 を返します。

反映に失敗した場合は実行時エラーとなります。

## 6.1.7 デバッグ操作

ここでは，Java プログラム呼び出し機能が提供するデバッグ操作のサービスルーチンについて説明します。

## (1) CBLJDEBUGSTRING

CBLJDEBUGSTRING サービスルーチンは，ユーザデバッグ情報として，文字列情報をデバッグ情報ファイルに出力します。

### 形式

```
CALL ’CBLJDEBUGSTRING’ USING　引数1　引数2　引数3.
```

### 引数

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。
- 引数 2 には，英数字項目を指定します。
- 引数 3 には，引数 2 の英数字項目の長さを持つ 4 バイト 2 進形式の数字項目を指定します。

### 戻り値

RETURN-CODE 特殊レジスタには常に 0 を返します。

### 規則

- 環境変数 CBLJRTDUMP に指定したデバッグ情報ファイルに，引数 2 と引数 3 で指定した文字列を出力します。環境変数 CBLJRTDUMP を指定していない場合は出力しません。デバッグ情報ファイルの指定については，「5.2.1　デバッグ情報出力の指定方法」を参照してください。
- 引数 2 に指定した英数字項目の値を，引数 3 の長さだけ出力します。ただし，引数 2 の値に NULL 終端文字（X'00'）が含まれる場合は，それ以降の文字列を出力しません。
- 引数 3 に，引数 2 の英数字項目の長さを超えた値を指定した場合の動作は保証しません。
- CBLJINITIALIZE サービスルーチンを呼び出す前に，CBLJDEBUGSTRING サービスルーチンを呼び出す場合，引数 1 に指定する CBLJENV 集団項目は初期化されている必要があります。

### 注意事項

CBLJDEBUGSTRING サービスルーチンは，引数 2 と引数 3 で指定された値をそのまま出力します。値の末尾に不完全な多バイト文字が格納されていてもそのまま出力します。

## (2)　CBLJMEMDUMP

CBLJMEMDUMP サービスルーチンは，ユーザデバッグ情報として，メモリ領域の内容をデバッグ情報ファイルに出力します。

### 形式

```
CALL ’CBLJMEMDUMP’ USING 引数1　引数2　引数3.
```

### 引数

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。
- 引数 2 には，領域の先頭アドレスを示すポインタ項目を指定します。
- 引数 3 には，出力する領域のサイズ（バイト数）を持つ 4 バイト 2 進形式の数字項目を指定します。

### 戻り値

RETURN-CODE 特殊レジスタには常に 0 を返します。

### 規則

- 環境変数 CBLJRTDUMP に指定したデバッグ情報ファイルに，引数 2 と引数 3 で指定したメモリ領域の内容を出力します。環境変数 CBLJRTDUMP を指定していない場合は出力しません。デバッグ情報ファイルの指定については，「5.2.1　デバッグ情報出力の指定方法」を参照してください。
- CBLJMEMDUMP サービスルーチンは，引数 2 に指定したアドレス位置から引数 3 に指定したサイズ（バイト数）だけ，16 進形式の領域ダンプリストを出力します。領域ダンプリストの形式については，「5.2.2　デバッグ情報の種類と出力フォーマット」の「(3)　データの種類と出力フォーマット」の「(l)　領域のダンプリストの出力フォーマット」を参照してください。
- CBLJINITIALIZE サービスルーチンを呼び出す前に，CBLJMEMDUMP サービスルーチンを呼び出す場合，引数 1 に指定する CBLJENV 集団項目は初期化されている必要があります。

### 注意事項

引数 2 には，作業場所節のデータ項目など，実行中の COBOL プログラムで参照できる領域の先頭アドレスとサイズを指定してください。サイズが不明な領域は指定しないでください。また，参照できるサイズより大きなサイズを指定すると，バッファオーバーランが発生します。参照できるサイズを指定してください。

CBLJENV 集団項目の従属項目のポインタ項目やオブジェクト参照のポインタ項目など，Java プログラム呼び出し機能が実行時に確保する領域のアドレスも指定できません。これらのポインタ項目を指定した場合の動作は保証しません。

## 6.2　実行時環境変数

ここでは，実行時環境変数について説明します。

環境変数は次に示す形式で説明します。

| 環境変数名=環境変数の値 |
|---|

環境変数の値に関する注意事項

- 環境変数の値が固定値の場合，大文字と小文字は等価とみなします。
- 空白を含むパス名など，環境変数の値の一部に空白を指定する場合，引用符（"）で囲まないでください。引用符を指定した場合，環境変数の値の一部として取り扱います。

実行時環境変数の一覧を次の表に示します。

### 表 6-10　実行時環境変数の一覧

| 項番 | 実行時環境変数 | 機能 |
|---|---|---|
| 1 | CBLJRTBIGENDIAN | ビッグエンディアン形式で取り扱うバイナリデータの種類を指定する。 |
| 2 | CBLJRTDUMP | デバッグ情報を出力するデバッグ情報ファイルのパス名を指定する。 |
| 3 | CBLJRTDUMP_MAXSIZE | デバッグ情報を出力するデバッグ情報ファイルの最大サイズを指定する。 |
| 4 | CBLJRTERR | 実行時エラー情報ファイルの出力有無，および実行時エラー情報の出力先フォルダのパス名を指定する。 |
| 5 | CBLJRTVMDEFAULTOPTIONS | デフォルトの Java VM 起動オプションを指定する。 |
| 6 | CBLJRTVMOPTIONS | プロセス単位の Java VM 起動オプションを指定する。 |
| 7 | CBLJRTVMOPTLOG | Java VM 起動オプション情報の出力先を変更する場合に，出力先フォルダのパス名を指定する。 |
| 8 | CBLJRTVMOPTLOG_MAXSIZE | Java VM 起動オプション情報の出力先ファイルの最大サイズを指定する。 |

また，Java プログラム呼び出し機能で使用する COBOL2002 実行時環境変数の一覧を次の表に示します。COBOL2002 実行時環境変数については，マニュアル「COBOL2002　ユーザーズガイド」を参照してください。

### 表 6-11　COBOL2002 実行時環境変数の一覧

| 項番 | 環境変数名 | 指定する内容 |
|---|---|---|
| 1 | CBLLANG | 動作する言語環境（文字コード）を指定する。 |
| 2 | CBLUNIENDIAN | 用途が NATIONAL の項目に対する Unicode のバイトオーダを指定する。 |

## 6.2.1　実行時環境変数の設定

Java プログラム呼び出し機能の実行時の環境を実行時環境変数で指定します。

なお，Java プログラム呼び出し機能が実行時に使用する環境変数では，プロセス実行前に指定した環境変数だけが有効となります。実行支援および COBOL プログラムの環境変数へのアクセスで設定した環境変数は有効となりません。

COBOL プログラムの実行環境の設定方法については，マニュアル「COBOL2002　ユーザーズガイド」を参照してください。

Cosminexus 環境下で Cosminexus 連携機能と併用する場合，Cosminexus 連携機能の設定が必要です。Cosminexus 連携機能については，マニュアル「COBOL2002　Cosminexus 連携機能ガイド」を参照してください。

## 6.2.2　実行時環境変数の詳細

実行時環境変数について説明します。

### (1)　CBLJRTBIGENDIAN

ビッグエンディアン形式で取り扱うバイナリデータの種類を指定します。指定しない場合，システムのエンディアンの形式で取り扱われます。

この環境変数は，システムのエンディアンがリトルエンディアン形式の場合に有効です。

**形式**

```
CBLJRTBIGENDIAN=データの種類
```

データの種類

BIN，FLOAT のどちらか，または両方を指定します。両方指定する場合は，コロン（:）で区切ります。

例）

```
CBLJRTBIGENDIAN=BIN:FLOAT
```

**規則**

この環境変数を指定した場合，指定したデータの種類の Java データに対応する COBOL データ項目はビッグエンディアン形式として取り扱われます。

表 6-12　Java データ型との対応

| 指定値 | Java データ型 |
|---|---|
| BIN | char／short／int／long |

| 指定値 | Java データ型 |
|---|---|
| FLOAT | float／double |

注意事項

データの種類は，COBOL プログラムの-BigEndian コンパイラオプションの指定と合わせて指定する必要があります。-BigEndian コンパイラオプションの指定と矛盾している場合の動作は保証しません。

表 6-13　コンパイラオプションとの対応

| 指定値 | コンパイラオプション |
|---|---|
| BIN | -BigEndian,Bin |
| FLOAT | -BigEndian,Float |

## (2)　CBLJRTDUMP

デバッグ情報を出力するデバッグ情報ファイルのパス名を指定します。

形式

```
CBLJRTDUMP=ファイルパス
```

ファイルパス

デバッグ情報ファイルのパス名を絶対パスで指定します。デバッグ情報ファイルのファイル名にプロセス ID を付けない場合は，最後にセミコロン（;）を付けて指定します。

規則

デバッグ情報は指定したファイルに追加されます。

実際に書き込むファイル名は，拡張子の直前にアンダーバーとプロセス ID（_プロセス ID）を付けた名称です。プロセス ID を付けたくない場合は，指定するファイルパス名の最後にセミコロン（;）を付けて指定してください。

デバッグ情報の出力ファイル名の例（プロセス ID が 1000 の場合）を次の表に示します。

表 6-14　デバッグ情報の出力ファイル名の例（プロセス ID が 1000 の場合）

| 項番 | 環境変数の値 | 出力するファイル名 |
|---|---|---|
| 1 | C:¥tmp¥…¥sample.log | C:¥tmp¥…¥sample_1000.log |
| 2 | C:¥tmp¥…¥sample | C:¥tmp¥…¥sample_1000 |
| 3 | C:¥tmp¥…¥sample.log; | C:¥tmp¥…¥sample.log |
| 4 | C:¥tmp¥…¥sample; | C:¥tmp¥…¥sample |

注意事項

- プロセス ID を付けない出力ファイル名を指定する場合，別のプロセスですでに同じファイルを使用しているとき，同時にファイルを開くことができないため，デバッグ情報を出力できません。こ

の場合は，プロセスごとにローカライズされる環境変数の指定方法で，それぞれ異なるファイル名を使用するように出力ファイル名を指定してください。

- ファイルの書き込みでエラーが発生した場合，デバッグ情報は出力されません。このとき，実行時エラーは発生しません。要因を確認して対処してください。

  ファイルの書き込みでエラーが発生する要因を次に示します。

  ・ファイルの出力先フォルダがない。

  ・ファイルの出力先フォルダまたは指定したファイルに書き込み権限がない。

  ・別のプロセスで，すでに同じ名称のファイルが開かれている。

  ・ディスク容量が不足している。

- ファイルの出力先フォルダは作成されません。存在しているフォルダを出力先フォルダとして指定してください。

デバッグ情報の出力については，「5.2　デバッグ情報の出力」を参照してください。

## (3)　CBLJRTDUMP_MAXSIZE

デバッグ情報を出力するデバッグ情報ファイルの最大サイズを指定します。

### 形式

```
CBLJRTDUMP_MAXSIZE=最大サイズ
```

最大サイズ

デバッグ情報ファイルの最大サイズを示す数値を 0～2,000 の範囲（MB 単位）で指定します。

### 規則

- 環境変数 CBLJRTDUMP を指定していない場合，この環境変数は無効です。
- この環境変数を指定しない場合，または無効な値を指定している場合の既定値は，10 です。

### 注意事項

この環境変数に 0 を指定した場合，ファイルの最大サイズのチェックをしないで，デバッグ情報をファイルへ書き込みます。0 を指定する場合は，ディスクの残容量に注意してください。

この環境変数に 0 以外の値を指定した場合の動作を次に示します。

- デバッグ情報の書き込みで指定した最大サイズを超えない場合，指定したデバッグ情報ファイルに追加書きします。
- デバッグ情報の書き込みで，デバッグ情報ファイルが指定した最大サイズを超える場合，書き込み前のデバッグ情報ファイルをバックアップしたあと，デバッグ情報ファイルを新規作成します。バックアップファイルの名称は，デバッグ情報ファイルのファイル名の後ろ 1 バイトをアンダーバー（_）に置き換えた名称です。すでに同じ名称のファイルが存在する場合は，上書きします。
- デバッグ情報ファイルのバックアップが失敗した場合，指定した最大サイズを超えて，指定したデバッグ情報ファイルに追加書きします。書き込み時にすでに指定した最大サイズを超えている場合も追加書きします。

デバッグ情報の出力については，「5.2　デバッグ情報の出力」を参照してください。

## (4)　CBLJRTERR

実行時エラー情報ファイルの出力有無，および実行時エラー情報の出力先フォルダのパス名を指定します。

### 形式

```
CBLJRTERR={フォルダパス|<SUPPRESS>}
```

フォルダパス

　実行時エラー情報の出力先フォルダのパス名を絶対パスで指定します。

<SUPPRESS>

　実行時エラー情報ファイルを出力しません。

### 注意事項

- ファイルの出力先フォルダは作成されません。存在しているフォルダを出力先フォルダとして指定してください。
- ファイルの書き込みでエラーが発生した場合，実行時エラー情報ファイルは出力されません。実行時エラー情報ファイルが出力されない場合の要因を次に示します。確認して対処してください。
  ・ファイルの出力先フォルダまたは指定したファイルに書き込み権限がない。
  ・出力するファイル名を加えたパス名の長さがシステムの制限を超える。
  ・ディスク容量が不足している。
- Java プログラム呼び出し機能では，実行時エラー情報ファイルを削除しません。ディスク容量不足の要因となることがあるため，定期的に不要なファイルを削除してください。
- 障害発生時の調査情報として有効のため，運用時には環境変数 CBLJRTERR に"<SUPPRESS>"を指定しないでください。
- コマンドラインで環境変数の値に<SUPPRESS>を指定する場合，引用符で囲む必要があります。

実行時エラー情報の出力については，「5.3　実行時エラー情報の出力」を参照してください。

## (5)　CBLJRTVMDEFAULTOPTIONS

デフォルトの Java VM 起動オプションを指定します。この環境変数を指定した場合，Java プログラム呼び出し機能が Java VM を起動するときに指定するデフォルトの起動オプションを変更します。

### 形式

```
CBLJRTVMDEFAULTOPTIONS=ファイルパス名
```

ファイルパス名

　デフォルトの Java VM 起動オプションを指定したファイルのパス名を絶対パスで指定します。

### 規則

- この環境変数は，Java プログラム呼び出し機能が Java VM を起動する場合にだけ有効です。CBLJINITIALIZE サービスルーチン呼び出し前に，すでに同じプロセスで Java VM が起動されている場合は有効になりません。
- この環境変数を指定した場合，指定したファイルの内容は，Java プログラム呼び出し機能が起動するすべての Java VM に起動オプションとして引き渡されます。
- ファイルの終端の行も改行で終わっている必要があります。改行コードがない場合は，終端の行に記述したオプションが正しく Java VM に引き渡されません。

### 注意事項

この環境変数の値に指定したパス名のファイルを読み込めない場合，環境変数を指定したと見なして，Java プログラム呼び出し機能が保持しているデフォルトの起動オプションを取り消します。

### ファイルの形式と規則

デフォルトの Java VM 起動オプションを指定したファイルの形式，規則，および指定例を次に示します。

ファイルの形式

テキストファイル形式（使用する文字コードは，シフト JIS）のファイルを指定してください。

ファイルの規則

- 1 行の長さは，0～1,024 バイトの範囲で指定してください。それ以上の長さを指定すると無視されます。
- オプションは，1 行ごとに一つのオプションを記述してください。1 行に複数のオプションを記述した場合の動作は保証しません。
- 改行だけ，または空白文字（X'20'）だけの行（空白行），行の 1 バイト目が「#」（X'23'）の行（コメント行）を無視します。

ファイルの指定例

```
# Java 情報取得の設定
-XX:+PrintGCDetails
-XX:HeapDumpPath=javalogs
-XX:-HeapDumpOnOutOfMemoryError

# システムで使用するclasspath
-Djava.class.path=systemlibs
```

## (6) CBLJRTVMOPTIONS

プロセス単位の Java VM 起動オプションを指定します。この環境変数を指定した場合，Java VM を起動する COBOL プログラムで指定済みの CBLJENV 集団項目の CBLJVMOPTIONS 項目が無効になります。

### 形式

```
CBLJRTVMOPTIONS=ファイルパス名
```

ファイルパス名

Java VM の起動オプションを指定したファイルのパス名を絶対パスで指定します。

**規則**

- この環境変数は，Java プログラム呼び出し機能が Java VM を起動する場合にだけ有効です。CBLJINITIALIZE サービスルーチン呼び出し前に，すでに同じプロセスで Java VM が起動されているときは有効になりません。
- この環境変数を指定した場合，Java VM を起動する COBOL プログラムですでに設定している CBLJENV 集団項目の CBLJVMOPTIONS 項目は無効になります。必要なオプションをすべて指定してください。

**注意事項**

この環境変数の値に指定したパス名のファイルを読み込めない場合，環境変数を指定したと見なして，CBLJENV 集団項目の CBLJVMOPTIONS 項目の指定を無効とします。CBLJENV 集団項目の CBLJVMOPTIONS 項目の指定を有効とするには，環境変数の登録を削除してください。

**ファイルの形式と規則**

ファイルの形式は，環境変数 CBLJRTVMDEFAULTOPTIONS と同じです。ファイルの形式については，「(5)　CBLJRTVMDEFAULTOPTIONS」を参照してください。

## (7)　CBLJRTVMOPTLOG

Java VM 起動オプション情報の出力先を変更する場合に，出力先フォルダのパス名を指定します。

**形式**

```
CBLJRTVMOPTLOG=フォルダパス
```

フォルダパス

出力先フォルダのパス名を絶対パスで指定します。

**規則**

- Java VM 起動オプション情報は，CBLJRTVMOPT.log ファイルに出力されます。
- この環境変数を指定しない場合，または無効な値を指定した場合，既定の出力先フォルダに Java VM 起動オプション情報が出力されます。既定の出力先フォルダは，次に示す優先順位で決定されます。

  1.環境変数 TEMP に指定したフォルダ

  2.環境変数 TMP に指定したフォルダ

  3.カレントフォルダ

Java VM 起動オプション情報については，「5.4　Java VM 起動オプション情報の収集」を参照してください。

## (8) CBLJRTVMOPTLOG_MAXSIZE

Java VM 起動オプション情報の出力先ファイルの最大サイズを指定します。

### 形式

```
CBLJRTVMOPTLOG_MAXSIZE=最大サイズ
```

最大サイズ

変更する最大サイズを示す数値を 0～2,000 の範囲（MB 単位）で指定します。

### 規則

この環境変数を指定しない場合，または無効な値を指定した場合の既定値は，2 です。

### 注意事項

この環境変数に 0 を指定した場合，ファイルの最大サイズのチェックをしないで，起動オプション情報をファイルへ書き込みます。0 を指定する場合は，ディスクの残容量に注意してください。

この環境変数に 0 以外の値を指定した場合の動作を次に示します。

- 起動オプション情報の書き込みで指定した最大サイズを超えない場合，指定した起動オプション情報の出力先ファイルに追加書きします。
- 起動オプション情報の書き込みで出力先ファイルが指定した最大サイズを超える場合，書き込み前の出力先ファイルをバックアップしたあと，出力先ファイルを新規で作成します。バックアップファイルの名称は，出力先ファイルのファイル名の後ろ 1 バイトをアンダーバー（_）に置き換えた名称です。すでに同じ名称のファイルが存在する場合は，上書きします。
- 出力先ファイルのバックアップが失敗した場合，指定した最大サイズを超えて，指定した出力先ファイルに追加書きします。書き込み時にすでに指定した最大サイズを超えている場合も追加書きします。

Java VM 起動オプション情報については，「5.4　Java VM 起動オプション情報の収集」を参照してください。

# 7 プログラム作成支援ツール

この章では，Java プログラム呼び出し機能が提供するプログラム作成支援ツールについて説明します。

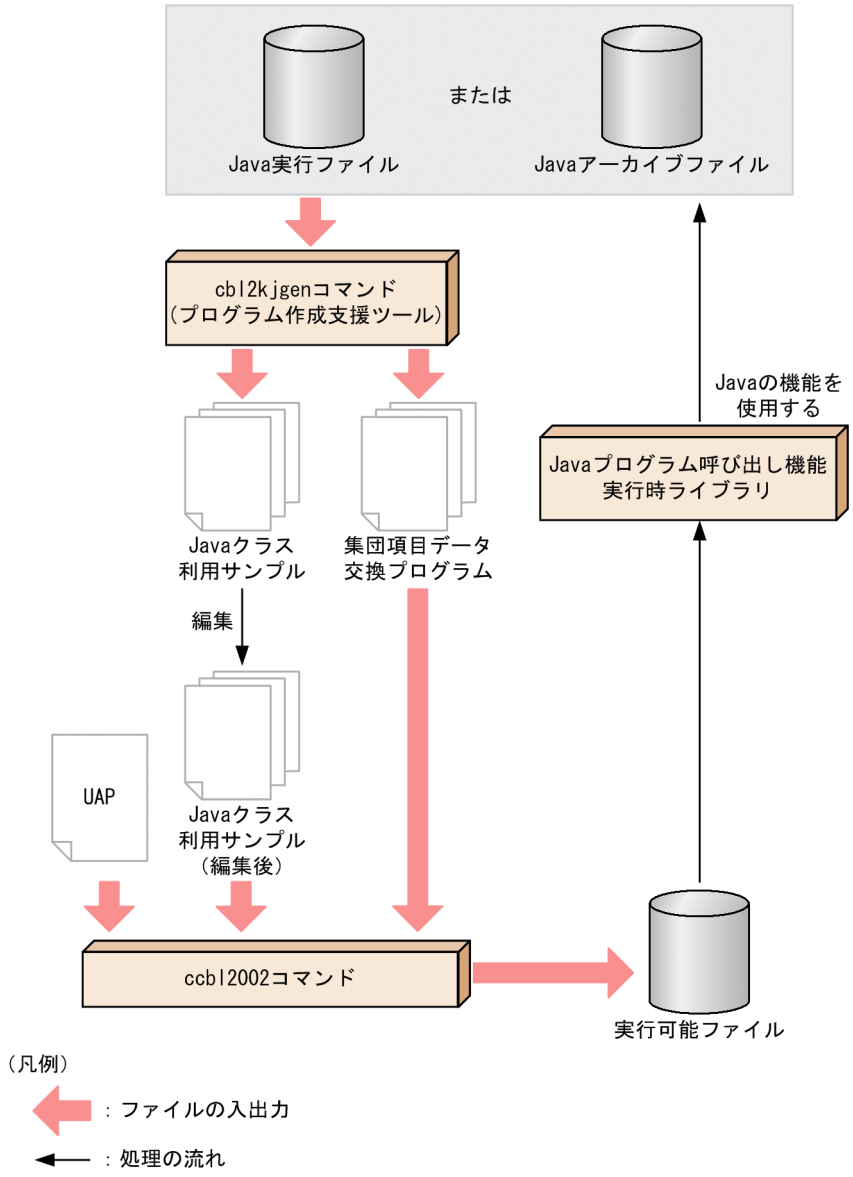
なお，プログラム作成支援ツールは，COBOL2002 Developer Professional でだけ提供するツールです。

# 7.1　プログラム作成支援ツールの概要

プログラム作成支援ツールは，Java プログラム呼び出し機能を使用する COBOL コードを生成するツールです。このツールを使用することで，使用する Java プログラムに合わせた COBOL プログラムの作成が容易になります。

プログラム作成支援ツールの概要を次の図に示します。

図 7-1　プログラム作成支援ツールの概要




プログラム作成支援ツールは，Java 実行ファイルまたは Java アーカイブファイルで定義されている Java の機能を使用する COBOL プログラムを生成します。生成できる COBOL プログラムは次に示す 2 種類です。

- Java クラス利用サンプル

使用する Java プログラムから，Java のフィールドやメソッドを利用する COBOL コードのサンプルを生成します。

- 集団項目データ交換プログラム

  使用する Java プログラムから，Java のフィールドに対応するデータ項目を含む COBOL 集団項目を定義した登録集原文と，一括でデータの取得や設定をする COBOL プログラムを生成します。

生成した COBOL プログラムは，編集後，COBOL2002 コンパイラでコンパイルし，Java プログラム呼び出し機能の実行時ライブラリを使用して実行します。

### 7.1.1 プログラム作成支援ツールの特長

プログラム作成支援ツールを使用すると，Java プログラム呼び出し機能を使用して Java プログラムを使用する COBOL プログラムを自動で生成できます。生成された COBOL プログラムは編集できるため，必要な定義や処理を選択して使用することで，Java プログラム呼び出し機能を使用する際のコーディング量を削減でき，Java のフィールドやメソッドの使用が容易になります。

Java プログラム呼び出し機能を使用して Java プログラムを使用するには，少なくとも次に示すコードを COBOL プログラムに記述する必要があります。

- Java プログラム呼び出し機能の初期化
- オブジェクト参照およびクラス参照の生成
- Java のフィールドの参照，設定，およびメソッドの呼び出し
- Java プログラム呼び出し機能の終了

sample.java の javaMethod メソッドを Java プログラム呼び出し機能で呼び出す場合の COBOL プログラムの例を次に示します。

sample.java

```
public class SampleClass {
   public static String classID = "SampleClass";
   public String javaMethod(int i, String str) {
       return str.substring(i);
   }
}
```

javaMethod メソッドを呼び出す COBOL プログラム

```
000000*  Javaの実行環境を作成する(Java VMロード) --------------------------------
000000     CALL 'CBLJINITIALIZE' USING CBLJENV.
000000
000000*  JavaのSampleClassクラスのクラス参照を取得する ------------------------
000000     CALL 'CBLJGETCLASS' USING CBLJENV sample CLASSREF.
000000
000000*  Javaのsampleクラスのインスタンスを生成する ----------------------------
000000     CALL 'CBLJNEW' USING CBLJENV CLASSREF NO-ARGS  OBJREF.
000000
000000*  javaMethod メソッドを呼び出す -----------------------------------------
000000     COMPUTE ARGPTR(1) = FUNCTION ADDR( ARG-01 ).
000000     COMPUTE ARGPTR(2) = FUNCTION ADDR( ARG-02 ).
000000     COMPUTE ARGPTR(3) = ZERO.
000000     CALL 'CBLJINVOKE' USING CBLJENV
000000                       OBJREF   javaMethod  ARG-LIST  RTN.
000000
000000*  Javaのインスタンスを解放する -------------------------------------------
000000     CALL 'CBLJRELEASE'  USING CBLJENV  OBJREF.
000000
000000*  Javaの実行環境を削除する (Java VMアンロード) ---------------------------
000000     CALL 'CBLJFINALIZE' USING CBLJENV.
```

javaMethod メソッドを呼び出す COBOL プログラムのように，一つのメソッドを呼ぶ場合でも次に示す課題があります。

- 1 回のメソッド呼び出しや変数参照だけで 10～20 ステップのコーディングが必要となる。
- サービスルーチンに渡す引数リストの準備など，Java プログラム呼び出し機能の作法的なコーディングに時間を取られる。

プログラム作成支援ツールを使用して COBOL プログラムを自動で生成することで，これらの課題を解決できます。

## 7.1.2　プログラム作成支援ツールの前提条件

プログラム作成支援ツールは，32bit 版 COBOL2002 Developer Professional と 64bit 版 COBOL2002 Developer Professional で使用できます。

このマニュアルの記述は，特に記載しないかぎり，32bit 版 COBOL2002 Developer Professional と 64bit 版 COBOL2002 Developer Professional のプログラム作成支援ツールで共通です。

## 7.1.3　生成例で使用する Java プログラム

プログラム作成支援ツールを使用したときの生成例の入力プログラムとして，次に示す Java プログラムを使用して説明します。

ファイル名：SampleClass.java

```
public class SampleClass {
   /** クラスID */
   public static String classID = "SampleClass";

   /** 実行種別 */
   private       int    execType;
   /** 実行データ */
   private       String execData;

   /* メンバID一覧 */
   private       int[][]  memberList;

   /** constructor */
   public SampleClass() {
      execType   = 0;
      execData   = "";
      memberList = new int[][]
      {
        { 1001, 1002, 1003, 1004 },
        { 2001, 2002, 2003, 2004 },
      };
   }

   /**
    * executeProcess
    * @return 実行結果
    */
   public boolean executeProcess(int flag) {
//           :
```

```
        return true;
    }
    /**
     * 実行種別を取得する
     * @return 実行種別
     */
    public int getExecType() {
        return execType;
    }
    /**
     * 実行種別を設定する
     */
    public void setExecType(int type) {
        execType = type;
    }
    /**
     * 実行データを取得する
     * @return 実行データ
     */
    public String getExecData() {
        return execData;
    }
    /**
     * 実行データを設定する
     */
    public void setExecData(String data) {
        execData = data;
    }

    /**
     * メンバID一覧を取得する
     * @return メンバID一覧
     */
    public int[][] getMemberList()
    {
        return memberList;
    }

    /**
     * メンバID一覧を設定する
     */
    public void setMemberList(int[][] list)
    {
        memberList = list;
    }
}
```

## 7.2　プログラム作成支援ツールの機能

プログラム作成支援ツールは cbl2kjgen コマンドで実行します。

cbl2kjgen コマンドの形式を次に示します。

### 形式

```
cbl2kjgen -Type  {Sample | GroupMapper} ＋
          {-Jar Javaアーカイブファイル名 | -Class Javaクラス名〔:内部名〕〔,Javaクラス名〔
:内部名〕〕…} ＋
          〔-ClassPath クラスパス〕
          〔-OutDir フォルダパス〕
          〔-StrMaxLen 最大長〕
          〔-MaxArrayLength 最大値〕
          〔-Format  {fixed | free} 〕
```

### 規則

- オプション名とオプションに指定する選択値では，英大文字と英小文字を区別しません。
- 同じオプションを指定した場合，後ろに指定したオプションが有効となります。
- ここで説明されていない文字列を cbl2kjgen コマンドに指定した場合，エラーメッセージを出力し，その指定を無視します。

### オプション

cbl2kjgen コマンドに指定できるオプションについて説明します。

-Type　{Sample ｜ GroupMapper} ＋

- このオプションは省略できません。
- このオプションで生成物の種別を指定します。-Type オプションの値を次の表に示します。

表 7-1　cbl2kjgen コマンドの-Type オプションの値

| 値 | 生成物の種別 |
|---|---|
| Sample | Java クラス利用サンプル |
| GroupMapper | 集団項目データ交換プログラム |

- Sample と GroupMapper は同時に指定できます（順不同）。同時に指定する場合はコンマ（,）で区切ります。
- Sample または GroupMapper 以外を指定した場合，エラーメッセージを出力し，処理を中止します。
- このオプションを省略した場合，エラーメッセージを出力し，処理を中止します。

-Jar Java アーカイブファイル名

- このオプションは省略できます。ただし，このオプションと-Class オプションをどちらも指定しない場合，エラーメッセージを出力し，処理を中止します。

- 指定した Java アーカイブファイルを解析対象とします。
- 次に示す場合はエラーメッセージを出力し，処理を中止します。
  ・Java アーカイブファイル名が指定されていない場合
  ・指定した Java アーカイブファイルが存在しない場合
  ・指定した Java アーカイブファイルの読み込みに失敗した場合
- このオプションと-Class オプションの組み合わせによって，プログラム作成支援ツールが解析するファイルが異なります。オプションの組み合わせと解析ファイルの関係については，「7.2.1 プログラム作成支援ツールの機能範囲」の「(1) Java プログラムの解析」の「(b) オプションの組み合わせによる解析ファイルの違い」を参照してください。

-Class Java クラス名〔:内部名〕〔,Java クラス名〔:内部名〕〕…

- このオプションは省略できます。ただし，このオプションと-Jar オプションをどちらも指定しない場合，エラーメッセージを出力し，処理を中止します。
- このオプションと-Jar オプションを同時に指定するかどうかで，プログラム作成支援ツールが解析するファイルが異なります。オプションの組み合わせと解析ファイルの関係については，「7.2.1 プログラム作成支援ツールの機能範囲」の「(1) Java プログラムの解析」の「(b) オプションの組み合わせによる解析ファイルの違い」を参照してください。
- Java クラス名には対応する内部名を指定できます。-Class オプションの値を次の表に示します。

表 7-2 cbl2kjgen コマンドの-Class オプションの値

| 値 | 説明 |
|---|---|
| Java クラス名 | • プログラム作成支援ツールは，このオプションで指定した Java クラスだけを解析対象とします。<br>• Java クラス名を指定しない場合，エラーメッセージを出力し，処理を中止します。<br>• 複数の Java クラス名を指定する場合，コンマ（,）で区切ります。<br>• 指定した Java クラスが解析対象のファイルに定義されていない場合，その旨を警告メッセージに出力し，処理を続行します。解析対象のファイルについては，「7.2.1 プログラム作成支援ツールの機能範囲」の「(1) Java プログラムの解析」の「(b) オプションの組み合わせによる解析ファイルの違い」を参照してください。 |
| 内部名 | • 内部名は省略できます。<br>• Java クラス名に内部名を指定すると，プログラム生成ツールが生成する COBOL プログラムで，Java クラスに対応するデータ項目名を変更します。<br>• 内部名は 1～31 文字で指定します。この範囲で指定しない場合，エラーメッセージを出力し，処理を中止します。 |

-ClassPath クラスパス

- このオプションは省略できます。
- クラスパスが指定されていない場合，エラーメッセージを出力し，処理を中止します。
- クラスパスに指定できるパスを次に示します。
  ・Java 実行ファイルが格納されているフォルダのパス
  ・Java アーカイブファイル（.jar）のパス

- フォルダおよびJavaアーカイブファイル（.jar）以外のクラスパスは無視します。
- クラスパスを複数指定する場合はセミコロン（;）で区切ります。複数指定した場合の検索順序については，「7.2.1　プログラム作成支援ツールの機能範囲」の「(5)　使用するファイルとファイルの入出力」の「(b)　ファイルの入力」を参照してください。
- このオプションを指定した場合，環境変数CLASSPATHは使用しません。Java実行ファイルまたはJavaアーカイブファイルの検索パスについては，「7.2.1　プログラム作成支援ツールの機能範囲」の「(5)　使用するファイルとファイルの入出力」の「(b)　ファイルの入力」を参照してください。

-OutDir フォルダパス

- このオプションは省略できます。
- このオプションを指定した場合，プログラム作成支援ツールが生成するCOBOLプログラムの出力先を指定したフォルダパスに変更します。
- 次に示すどれかの条件に当てはまる場合，エラーメッセージを出力して処理を中止します。
  ・フォルダパスが指定されていない場合
  ・指定したフォルダが存在しない場合
  ・指定したフォルダへ書き込む権限がない場合
  ・指定したフォルダへの書き込みに失敗した場合
- このオプションを省略した場合，プログラム作成支援ツールはカレントフォルダにCOBOLプログラムを生成します。

-StrMaxLen 最大長

- このオプションは省略できます。
- このオプションで指定した最大長は，プログラム作成支援ツールが生成するCOBOLプログラム中の次の値に使用されます。
  ・CBLJSTRMAXLEN項目の値
  ・Javaプログラムの String型データと対応づけるCOBOLプログラムの英数字項目の長さ
- このオプションを省略した場合，上記の値に256が仮定されます。
- 最大長には1～1,024の整数を指定できます。それ以外の値を指定した場合，または値を指定しなかった場合は，エラーメッセージを出力し，処理を中止します。

-MaxArrayLength 最大値

- このオプションは省略できます。
- このオプションで指定した最大値は，プログラム作成支援ツールが生成するCOBOLプログラム中の次の値に使用されます。
  ・配列オブジェクトと対応づけるCOBOLの可変長繰り返し項目の最大反復回数
- このオプションを省略した場合，上記の値に256が仮定されます。
- 最大値には1～16,777,215の整数を指定できます。それ以外の値を指定した場合，または値を指定しなかった場合は，エラーメッセージを出力し，処理を中止します。

-Format {fixed | free}

- このオプションは省略できます。
- このオプションで出力ファイル（COBOL プログラム）の正書法を指定できます。-Format オプションの値を次の表に示します。

表 7-3 cbl2kjgen コマンドの-Format オプションの値

| 値 | 正書法 |
|---|---|
| fixed | 固定形式正書法 |
| free | 自由形式正書法 |

- このオプションに値を指定しなかった場合，エラーメッセージを出力し，処理を中止します。
- fixed または free 以外の値を引数に指定した場合，エラーメッセージを出力し，処理を中止します。
- このオプションを省略した場合，正書法として固定形式正書法が仮定されます。

戻り値

プログラム作成支援ツールの戻り値を次の表に示します。

表 7-4 プログラム作成支援ツールの戻り値

| 戻り値 | 内容 |
|---|---|
| 0 | 正常終了しました。 |
| 1 | 正常終了しました。<br>ただし，警告メッセージが出力されています。 |
| 2 | エラーで処理を中止しました。 |

コマンドのヘルプ

オプションを指定しないで cbl2kjgen コマンドだけを入力した場合，または次に示すコマンドを指定した場合，cbl2kjgen コマンドのヘルプが出力されて，正常終了します。

形式

```
cbl2kjgen  {-Help|-?}
```

注意

-Help または-?を指定した場合，ほかのオプションは無視されます。

## 7.2.1 プログラム作成支援ツールの機能範囲

プログラム作成支援ツールの機能範囲について説明します。

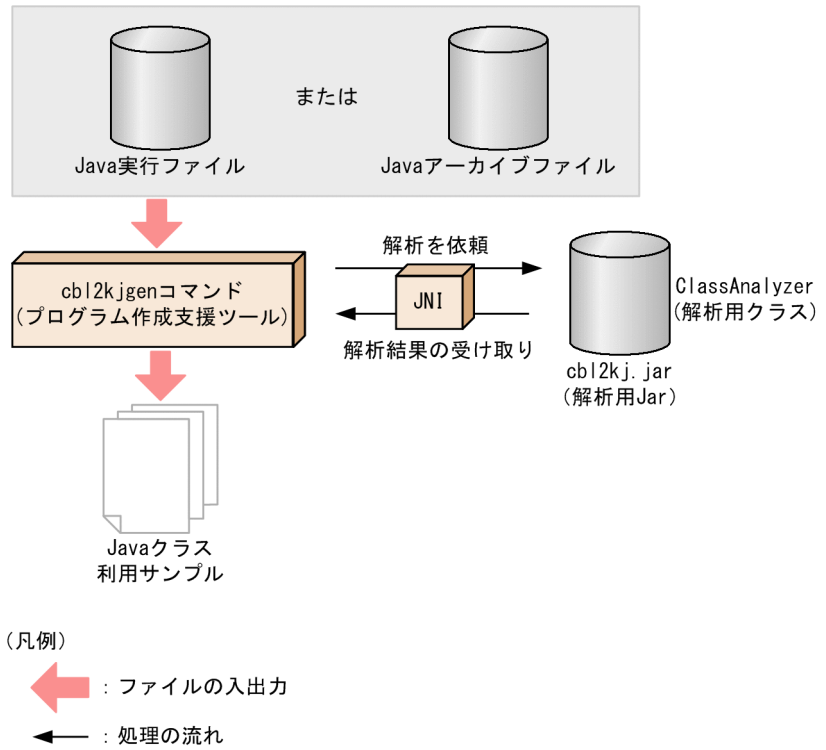
## (1)　Java プログラムの解析

プログラム作成支援ツールは，COBOL プログラムを作成する準備として Java 実行ファイルおよび Java アーカイブファイルを解析し，クラス，メソッドおよびフィールドを抽出します。解析結果は，プログラム作成支援ツールが生成する COBOL プログラムに反映されます。

Java 実行ファイルおよび Java アーカイブファイルの解析は，プログラム作成支援ツール専用の解析用 Java クラス（以降，解析用クラスと表記します）で行います。解析用クラスは，Java アーカイブファイルとして提供します。

解析用 Java アーカイブファイル（以降，解析用 Jar と表記します）および解析用クラスの概要を次の図に示します。解析用クラスへの解析依頼および解析結果の受け取りは JNI 経由で行います。

図 7-2　Java プログラムの解析の概要




プログラム作成支援ツールの解析対象は，解析する Java クラスやメソッドなどの公開範囲のほか，プログラム作成支援ツールのコマンドオプションの指定によって変わります。また，解析する Java クラスのほかに関連する Java クラスが必要となる場合があります。解析対象の条件を次に示します。

### (a)　プログラム作成支援ツールの解析対象

プログラム作成支援ツールの解析対象を次の表に示します。

表 7-5　プログラム作成支援ツールの解析対象

| 項番 | 対象 | 属性 | | |
|---|---|---|---|---|
| | | public | protected | private |
| 1 | クラス | ○ | × | × |
| 2 | コンストラクタ | ○ | × | × |
| 3 | メソッド※1 | ○ | × | × |
| 4 | フィールド※1※2 | ○ | × | × |
| 5 | インタフェース | × | × | × |

（凡例）

○：解析対象とする。

×：解析対象としない。

注※1

abstract 修飾子が指定された抽象クラスの場合，インスタンスメソッドおよびインスタンスフィールドは対象としません。

注※2

先祖クラスから継承されているフィールドは，条件付きで解析対象とします。フィールドの条件については，「(3)　フィールドの設定と取得」を参照してください。

## (b)　オプションの組み合わせによる解析ファイルの違い

-Jar オプションと-Class オプションの組み合わせで，解析するファイルが異なります。

-Jar オプションと-Class オプションの組み合わせと解析するファイルを次の表に示します。

表 7-6　-Jar オプションと-Class オプションの組み合わせと解析するファイル

| -Jar オプション | -Class オプション | |
|---|---|---|
| | あり | なし |
| あり | -Jar オプションに指定した Java アーカイブファイルの中から，-Class オプションで指定したクラスが定義されている Java 実行ファイルが解析対象の候補となる。 | -Jar オプションに指定した Java アーカイブファイルに含まれるすべての Java 実行ファイルが解析対象の候補となる。 |
| なし | 検索パス※に含まれるすべての Java 実行ファイルおよび Java アーカイブファイルの中から，-Class オプションで指定したクラスが定義されているファイルが解析対象の候補となる。 | エラーメッセージを出力し，処理を中断する。 |

注※

Java 実行ファイルおよび Java アーカイブファイルの検索パスについては，「(5)　使用するファイルとファイルの入出力」の「(b)　ファイルの入力」を参照してください。

## (c)　解析に必要となるクラス

プログラム作成支援ツールが Java プログラムを解析するとき，すべてのフィールド，メソッド，およびコンストラクタの情報を抽出します。そのため，解析対象のクラス以外も参照できる必要があります。

参照できる必要があるクラスを次に示します。

- 解析対象のクラス
- 解析対象のクラスのスーパークラス
- 解析対象のクラスが実装しているインタフェース
- 解析対象のクラスが import しているクラス

プログラム作成支援ツールが Java プログラムを解析する流れを次に示します。

1. 入力ファイル（Java 実行ファイルまたは Java アーカイブファイル）の各クラスからフィールド，メソッド，およびコンストラクタの情報を抽出する。
2. 抽出した情報（フィールド，コンストラクタ・メソッドの引数，およびメソッドの戻り値）のデータ型を解析する。

**注意事項**

- 解析に必要なクラスをすべて参照できるように，クラスの検索パスを指定してください。

  例えば，クラス A がクラス S を継承している場合，クラス A を解析するにはクラス S が参照できるように検索パスを設定する必要があります。

  検索パスについては，「(5)　使用するファイルとファイルの入出力」の「(b)　ファイルの入力」を参照してください。
- 解析に必要なクラスが参照できない場合，エラーメッセージを出力し，次の解析対象を解析します。

## (2)　データ項目のマッピング（共通）

Java プログラム呼び出し機能でクラス，メソッドまたはフィールドを操作する場合，値を保持するために COBOL 側でデータ項目を定義する必要があります。

Java プログラム呼び出し機能で使用するために適切なデータ型で定義し，Java のどのデータがどのデータ項目に設定されているかを一意に識別できる名前付けをします。

定義する必要があるデータ項目を次に示します。

- クラス名，メソッド名，およびフィールド名
- クラス参照とオブジェクト参照
- フィールド値
- コンストラクタおよびメソッドの引数
- メソッドの戻り値
- Java プログラム呼び出し機能のサービスルーチンの引数リスト
- String と英数字項目の変換で使用する作業用領域

- 配列オブジェクトと可変長繰り返し項目の変換で使用する作業用領域

プログラム作成支援ツールは，これらのデータ項目の定義を自動生成します。プログラム作成支援ツールが自動生成するデータ項目名と定義を次に示します。

データ項目名

クラス名，メソッド名，フィールド名を COBOL のデータ項目名に使用する場合，原則的に Java プログラムに定義されている名前をそのまま使用します。ただし，パッケージ名はデータ項目名に反映しません。

また，-Class オプションでクラス名とともに内部名を指定した場合，データ項目名をクラス名ではなく内部名に置き換えます。

**注意事項**

- クラス名，メソッド名，フィールド名を COBOL のデータ項目名に使用する場合，データ項目名が 31 文字を超えるときは警告メッセージとして該当する内部名と行番号を出力します。このとき，生成物は出力されますが，31 文字を超える語は COBOL2002 コンパイラでコンパイルエラーとなります。そのため，出力された警告メッセージを基に生成物を修正する必要があります。
- -Class オプションで指定した内部名が 1～31 文字でない場合，エラーメッセージを出力し，処理を中止します。
- 同じ名前が複数存在する場合，次に示すように二つ目以降の同じ名前のデータ項目（メソッド名やフィールド名を設定するデータ項目）の定義をコメントとして出力します。

```
     :
 01  wait              PIC X DYNAMIC C-STRING VALUE 'wait'.
*01  wait              PIC X DYNAMIC C-STRING VALUE 'wait'.
*01  wait              PIC X DYNAMIC C-STRING VALUE 'wait'.
     :
```

コメントとして出力する。

データ項目の定義（共通）

プログラム作成支援ツールが定義するデータ項目を次の表に示します。

### 表 7-7　プログラム作成支援ツールが定義するデータ項目

| 項番 | 用途 | データ項目※ |
|---|---|---|
| 1 | 引数がないコンストラクタおよびメソッドの引数リストを設定する。 | `01  NO-ARG.`<br>`  02  FILLER  USAGE POINTER VALUE NULL.` |
| 2 | 戻り値がないメソッドの型を設定する。 | `01  RTN-VOID.`<br>`  02  RTN-TYPE  PIC X(1) VALUE 'V'.` |
| 3 | String と英数字項目の変換で使用する領域とサイズを設定する。 | `01  WK-ALNUM  PIC X(`*STRMAXLEN*`).`<br>`01  WK-ALNUM-LEN  PIC S9(9) USAGE COMP VALUE `*STRMAXLEN*`.` |
| 4 | *m* 次元の COBOL の繰り返し項目の添字を設定する。 | `01  ARRAY-INDEX-`*m*`  PIC S9(9) USAGE COMP.` |

| 項番 | 用途 | データ項目※ |
|---|---|---|
| 5 | *m* 次元の配列オブジェクトのポインタを設定する。 | 01 ARRAY-ADDR-*m* USAGE POINTER. |
| 6 | *m* 次元の配列オブジェクトの添字を設定する。 | 01 ARRAY-INDEX-J-*m* PIC S9(9) USAGE COMP. |
| 7 | 配列の最下層の要素を設定する可変長繰り返し項目のアドレスを設定する。<br>CBLJGETARRAYADDR サービスルーチンや CBLJRELEASEARRAY サービスルーチンの第 3 引数に指定する。 | 01 TEMP-ARRAY-ADDR USAGE POINTER. |

注※

- "*STRMAXLEN*"の既定値は 256 です。この値はプログラム作成支援ツールの-StrMaxLen オプションで変更できます。
- "*m*"は配列オブジェクトの次元数を示します。

Java クラス利用サンプルの生成と集団項目データ交換プログラムの生成では，これらのデータ項目に加えて，それぞれの定義で必要なデータ項目を定義します。

Java クラス利用サンプルの生成でのデータ項目のマッピングについては，「7.2.2　Java クラス利用サンプルの生成」の「(2)　データ項目のマッピング」を参照してください。

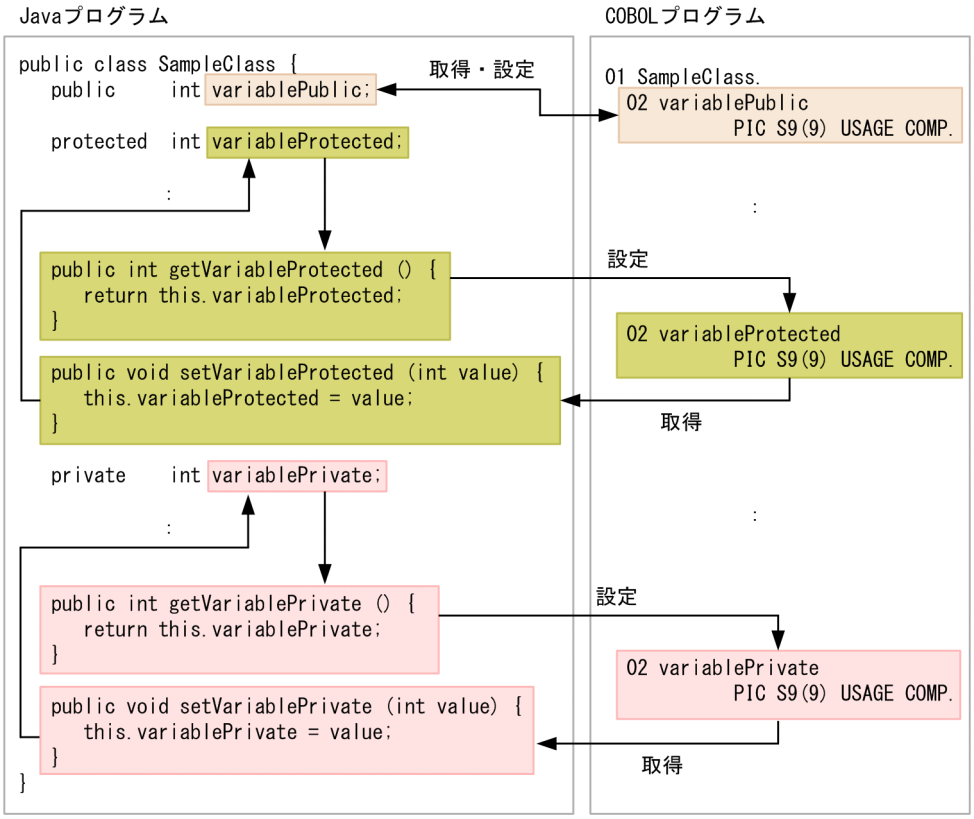
集団項目データ交換プログラムの生成でのデータ項目のマッピングについては，「7.2.3　集団項目データ交換プログラムの生成」の「(3)　データ項目のマッピング」を参照してください。

## (3)　フィールドの設定と取得

プログラム作成支援ツールは，Java のフィールドのアクセスレベルに応じた取得・設定方法で COBOL プログラムを定義します。

Java のフィールドのアクセスレベルと取得・設定方法を次の図に示します。

図 7-3　Java のフィールドのアクセスレベルと取得・設定方法




プログラム作成支援ツールが生成する COBOL プログラムで取得・設定できるフィールドの条件を次に示します。条件のどちらかに当てはまる場合，取得・設定できます。ただし，対象のフィールドに final 修飾子が付与されている場合，集団項目データ交換プログラム生成機能で生成するフィールドを一括設定するプログラムには，これらのフィールドに値を設定するコードを生成しません。

取得・設定できるフィールドの条件

- フィールドのアクセスレベルが public であること。

  メソッドを使用しないで，フィールドに直接アクセスして取得・設定します。getter メソッドまたは setter メソッドが定義されていても，該当のフィールドを取得・設定するときに，それらのメソッドを使用しません。

- フィールドのアクセスレベルが private または protected で，getter メソッドまたは setter メソッドが定義されていること。

  フィールドのアクセスレベルが private または protected の場合，getter メソッドまたは setter メソッドを使用してフィールドを取得・設定します。getter メソッドまたは setter メソッドが定義されていない場合，プログラム作成支援ツールが生成する COBOL プログラムでは，private または protected フィールドを取得・設定できません。

  プログラム作成支援ツールが，getter メソッドまたは setter メソッドが定義されているとみなす条件を次に示します。

getter メソッドまたは setter メソッドが定義されているとみなす条件※

- 該当のフィールドと同じクラスに定義されていること。

- メソッド名が次の命名規則にあてはまること。
  ・getter の命名規則："get"＋フィールド名（先頭は英大文字）
  ・setter の命名規則："set"＋フィールド名（先頭は英大文字）
  （例：フィールド名が variableInt の場合 → getVariableInt，setVariableInt）
- メソッドのアクセスレベルが public であること。
- （この条件は setter メソッドだけ適用）引数が一つで，該当のフィールドと同じ型であること。
- （この条件は setter メソッドだけ適用）戻り値がないこと。
- （この条件は getter メソッドだけ適用）引数がないこと。
- （この条件は getter メソッドだけ適用）戻り値の型が該当のフィールドと同じであること。
- （この条件は private フィールドだけ適用）対象のクラスでフィールドが定義されていること。
- （この条件は protected フィールドだけ適用）対象のクラスおよび，その先祖クラスでフィールドが定義されていること。

注※

プログラム作成支援ツールは，これらの条件をすべて満たしている場合だけ，getter メソッドまたは setter メソッドが定義されていると見なします。getter メソッドまたは setter メソッドとして定義されているメソッドがある場合も，満たさない条件がある場合，getter メソッドまたは setter メソッドが定義されているとは見なしません。

## (4)　配列オブジェクトのマッピング

解析対象のクラスで，次に示す場所に配列が使用されている場合，プログラム作成支援ツールは Java の配列と COBOL の繰り返し項目をマッピングするプログラムを生成します。

- フィールド（マッピングによって設定・取得が可能）※
- public コンストラクタまたは public メソッドの引数（マッピングによる設定だけ可能）
- public メソッドの戻り値（マッピングによる取得だけ可能）

注※

対象となるフィールドについては，「(3)　フィールドの設定と取得」を参照してください。

**注意事項**

プログラム作成支援ツールが COBOL のデータ項目にマッピングできる配列オブジェクトの次元数の上限は 7 次元です。7 次元を超える次元の配列要素は展開しないで，ポインタ項目のままとします。

### (a)　マッピングによる Java の配列の取得・設定の流れ

1. 配列オブジェクトのオブジェクト参照を取得する

配列が Java プログラムのどの場所に定義されているかでオブジェクト参照の取得方法が異なります。

フィールドとして定義されている場合

フィールドのオブジェクト参照を取得します。ただし，集団項目データ交換プログラムで配列フィールドを設定する場合は，CBLJNEW サービスルーチンで新規作成した配列オブジェクトを使用します。対象となるフィールドについては，「(3)　フィールドの設定と取得」を参照してください。

public コンストラクタまたは public メソッドの引数として定義されている場合

引数用に用意した COBOL のデータ項目に新しい配列オブジェクトを作成します。ただし，すでに他の処理で引数用に用意した COBOL のデータ項目に配列オブジェクトが作成されている場合は，新しく作成しないで流用します。

public メソッドの戻り値として定義されている場合

戻り値用に用意した COBOL のデータ項目からオブジェクト参照を取得します。

2. 配列の要素のオブジェクト参照を取得する

CBLJGETOBJARRAY サービスルーチンで，1.で取得したオブジェクト参照から配列要素のオブジェクト参照を取得します。このとき，PERFORM 文で配列の上層の要素から順次取り出すように添字を設定します。COBOL の繰り返し項目の添字と Java の配列の添字は別のデータ項目として保持します。

Java の配列の定義が 2 次元配列の例を次に示します。

例）

Java の配列の定義が 2 次元配列 a[x][y]の場合（x, y は 1 以上の整数）

```
CALL  'CBLJARRAYLENGTH' USING CBLJENV 配列J-1 配列長-1.

PERFORM VARYING 添字C-1 FROM 1 BY 1
 UNTIL 添字C-1 > 配列長-1

  COMPUTE 添字J-1 =添字C-1
  CALL  'CBLJGETOBJARRAY'  USING CBLJENV 配列J-1 添字J-1 配列J-2
  CALL  'CBLJARRAYLENGTH' USING CBLJENV 配列J-2 配列長-2
   （この部分は後述）
  PERFORM VARYING 添字C-2 FROM 1 BY 1
   UNTIL 添字C-2 > 配列長-2

   （この部分は後述）

  END-PERFORM
   （この部分は後述）
END-PERFORM.
```

```
添字C-1：COBOLの繰り返し項目の添字。添字xに相当する。値は1～x。
添字C-2：COBOLの繰り返し項目の添字。添字yに相当する。値は1～y。
添字J-1：Javaの配列の添字。添字xに相当する。値は0～x-1。
配列J-1：Javaの配列a[x]に相当する2次元配列オブジェクト。
配列J-2：Javaの配列a[][y]に相当する1次元配列オブジェクト。
配列長-1：配列-1の長さ。
配列長-2：配列-2の長さ。
```

配列 J-1（最上位次元の配列オブジェクト）は，1.で取得した配列オブジェクトを使用します。

多次元配列の場合，PERFORM 文を入れ子にすることで各要素を取り出します。例えば，Java の配列の定義が a[2][2]の場合，a[0][0]→a[0][1]→a[1][0]→a[1][1]の順でアクセスします。添字を設定するデータ項目は各次元に一つ定義します。

3. 配列の要素を取得・設定する

配列要素の取得・設定は配列が基本型配列かオブジェクト型配列かによって方法が異なります。

- 基本型配列の場合

  基本型配列の取得・設定は，Java プログラム呼び出し機能の CBLJGETARRAYADDR サービスルーチンおよび CBLJRELEASEARRAY サービスルーチンで Java の配列と COBOL の繰り返し項目を対応づけて，配列要素を取得・設定します。これらのサービスルーチンには繰り返し項目を持つ 01 レベルの集団項目のアドレスを渡す必要があるため，次に示すような 1 次元の可変長繰り返し項目を作業用に定義します。

```
01  TEMP-MAP-LEN    PIC S9(9) USAGE COMP VALUE ARRAYMAXLEN.
01  TEMP-MAP ADDRESSED BY TEMP-MAP-P.
  02  TEMP-MAP-ELEM TYPE-DEF  OCCURS ARRAYMAXLEN TIMES
                      DEPENDING ON TEMP-MAP-LEN.
```

  ・"*TYPE-DEF*"は Java の型に対応する COBOL の型を示します。Java の型に対応する COBOL の型については，「6.1.1　サービスルーチンで使用する引数」の「(3)　パラメタ型集団項目」を参照してください。

  ・"*ARRAYMAXLEN*"の既定値は 256 です。この値はプログラム作成支援ツールの-MaxArrayLength オプションで変更できます。

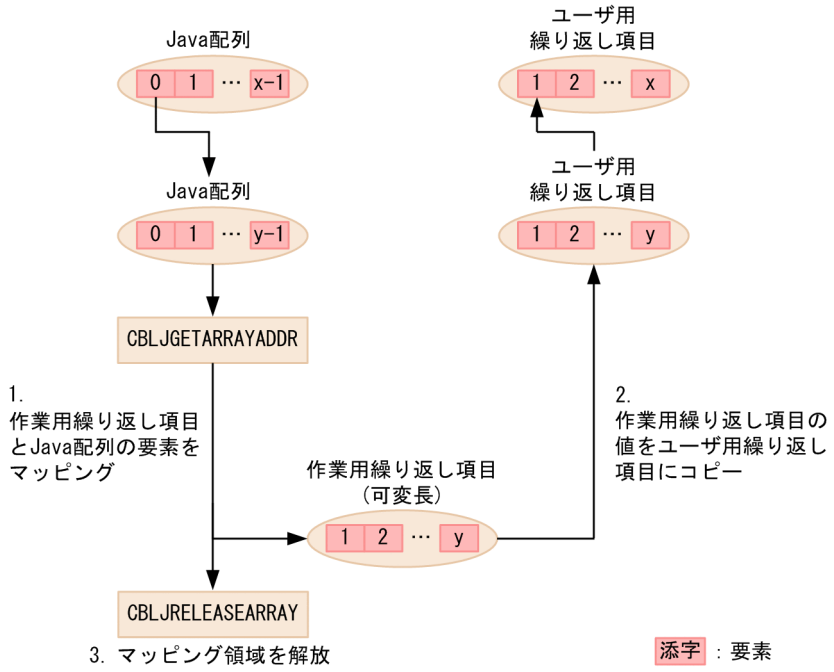
- オブジェクト型配列の場合

  オブジェクト型配列の取得・設定は，Java プログラム呼び出し機能の CBLJGETOBJARRAY サービスルーチンおよび CBLJSETOBJARRAY サービスルーチンで Java の配列と COBOL の繰り返し項目を対応付けることで実現します。基本型配列の場合に定義した作業用の一次元可変長繰り返し項目は使用しないため，定義しません。

### (b)　マッピングによる要素の取得の処理フロー，および COBOL プログラム

配列オブジェクトのマッピングによる要素の取得の処理フロー，および擬似的な COBOL プログラムを次に示します。

基本型配列の要素を取得する場合



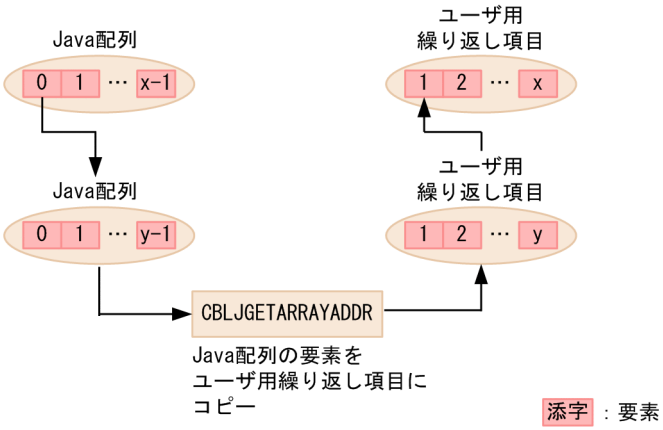

```
CALL 'CBLJARRAYLENGTH' USING CBLJENV 配列J-1 配列長-1.

PERFORM VARYING 添字C-1 FROM 1 BY 1 UNTIL 添字C-1 > 配列長-1

  COMPUTE 添字J-1 = 添字C-1 - 1
  CALL 'CBLJGETOBJARRAY' USING CBLJENV 配列J-1 添字J-1 配列J-2
  CALL 'CBLJARRAYLENGTH' USING CBLJENV 配列J-2 配列長-2
  MOVE 配列長-2 TO 作業用繰り返し項目の長さ
  CALL 'CBLJGETARRAYADDR' USING CBLJENV 配列J-2 TEMP-ARRAY-ADDR
  COMPUTE 作業用繰り返し項目のアドレス名 = TEMP-ARRAY-ADDR

  PERFORM VARYING 添字C-2 FROM 1 BY 1 UNTIL 添字C-2 > 配列長-2

     COMPUTE ユーザ用繰り返し項目(添字C-1,添字C-2) =
             作業用繰り返し項目(添字C-2)                    } ※

  END-PERFORM
  CALL 'CBLJRELEASEARRAY' USING CBLJENV 配列J-2 TEMP-ARRAY-ADDR
END-PERFORM.
```

添字C-1：COBOLの繰り返し項目の添字。添字xに相当する。値は1～x。
添字C-2：COBOLの繰り返し項目の添字。添字yに相当する。値は1～y。
添字J-1：Javaの配列の添字。添字xに相当する。値は0～x-1。
配列J-1：Javaの配列a[x]に相当する2次元配列オブジェクト。
配列J-2：Javaの配列a[][y]に相当する1次元配列オブジェクト。
配列長-1：配列-1の長さ。
配列長-2：配列-2の長さ。
作業用繰り返し項目の長さ：DEPENDING ONで指定した作業用繰り返し項目の長さを設定するデータ項目。

注※
byte型配列およびboolean型配列の場合，次に示すMOVE文に置き換わります。

```
MOVE 作業用繰り返し項目(添字C-2) TO
     ユーザ用繰り返し項目(添字C-1,添字C-2)
```

オブジェクト型配列の要素を取得する場合




```
CALL 'CBLJARRAYLENGTH' USING CBLJENV 配列J-1 配列長-1.

PERFORM VARYING 添字C-1 FROM 1 BY 1 UNTIL 添字C-1 > 配列長-1

  COMPUTE 添字J-1 = 添字C-1 - 1
  CALL 'CBLJGETOBJARRAY' USING CBLJENV 配列J-1 添字J-1 配列J-2
  CALL 'CBLJARRAYLENGTH' USING CBLJENV 配列J-2 配列長-2

  PERFORM VARYING 添字C-2 FROM 1 BY 1 UNTIL 添字C-2 > 配列長-2

    COMPUTE 添字J-2 = 添字C-2 - 1
    CALL 'CBLJGETOBJARRAY' USING CBLJENV 配列J-2 添字J-2
            ユーザ用繰り返し項目(添字C-1,添字C-2)                ※
  END-PERFORM
END-PERFORM.
```

添字C-1：COBOLの繰り返し項目の添字。添字xに相当する。値は1～x。
添字C-2：COBOLの繰り返し項目の添字。添字yに相当する。値は1～y。
添字J-1：Javaの配列の添字。添字xに相当する。値は0～x-1。
添字J-2：Javaの配列の添字。添字yに相当する。値は0～y-1。
配列J-1：Javaの配列a[x]に相当する2次元配列オブジェクト。
配列J-2：Javaの配列a[][y]に相当する1次元配列オブジェクト。
配列長-1：配列-1の長さ。
配列長-2：配列-2の長さ。

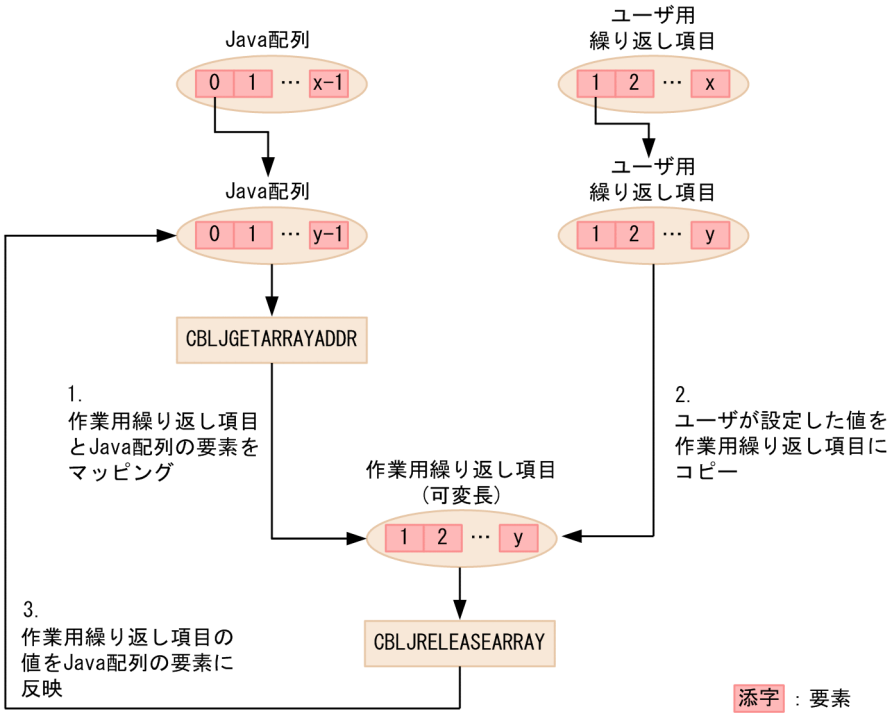
注※
集団項目データ交換プログラムの場合，String型配列を取得するとき，
次に示すようにString→英数字項目に変換するCALL文に置き換わります。

```
CALL 'CBLJGETOBJARRAY' USING CBLJENV 配列J-2 添字J-2
         TEMP-ARRAY-ADDR
CALL 'CBLJSTRINGTOX' USING CBLJENV TEMP-ARRAY-ADDR
         ユーザ用繰り返し項目(添字C-1,添字C-2)
         英数字項目の長さを格納するデータ項目
```

## (c)　マッピングによる要素の設定の処理フロー，および COBOL プログラム

配列オブジェクトのマッピングによる要素の設定の処理フロー，および擬似的な COBOL プログラムを次に示します。

基本型配列の要素を設定する場合



```
CALL 'CBLJARRAYLENGTH' USING CBLJENV 配列J-1 配列長-1.          } ※1

PERFORM VARYING 添字C-1 FROM 1 BY 1 UNTIL 添字C-1 > 配列長-1

  COMPUTE 添字J-1 = 添字C-1 - 1                                  }
  CALL 'CBLJGETOBJARRAY' USING CBLJENV 配列J-1 添字J-1 配列J-2   } ※2
  CALL 'CBLJARRAYLENGTH' USING CBLJENV 配列J-2 配列長-2          }
  MOVE 配列長-2 TO 作業用繰り返し項目の長さ
  CALL 'CBLJGETARRAYADDR' USING CBLJENV 配列J-2 TEMP-ARRAY-ADDR
  COMPUTE 作業用繰り返し項目のアドレス名 = TEMP-ARRAY-ADDR

  PERFORM VARYING 添字C-2 FROM 1 BY 1 UNTIL 添字C-2 > 配列長-2

     COMPUTE 作業用繰り返し項目(添字C-2) =                      }
             ユーザ用繰り返し項目(添字C-1,添字C-2)               } ※3

  END-PERFORM

  CALL 'CBLJRELEASEARRAY' USING CBLJENV 配列J-2 TEMP-ARRAY-ADDR
  CALL 'CBLJSETOBJARRAY'  USING CBLJENV 配列J-1 添字J-1 配列J-2
END-PERFORM.
```

添字C-1：COBOLの繰り返し項目の添字。添字xに相当する。値は1～x。
添字C-2：COBOLの繰り返し項目の添字。添字yに相当する。値は1～y。
添字J-1：Javaの配列の添字。添字xに相当する。値は0～x-1。
配列J-1：Javaの配列a[x]に相当する2次元配列オブジェクト。
配列J-2：Javaの配列a[][y]に相当する1次元配列オブジェクト。
配列長-1：配列-1の長さ。
配列長-2：配列-2の長さ。
作業用繰り返し項目の長さ：DEPENDING ONで指定した作業用繰り返し項目の長さを設定するデータ項目。

注※1
集団項目データ交換プログラムの場合，次に示すように配列を新規作成するCALL文に置き換わります。

```
CALL 'CBLJNEWARRAY' USING CBLJENV 配列J-1の型 配列長-1 配列J-1.
```

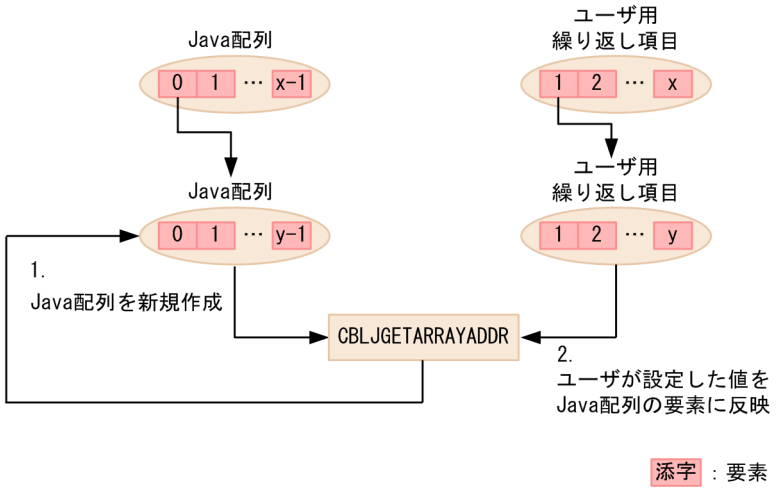
注※2
集団項目データ交換プログラムの場合，次に示すように配列を新規作成するCALL文に置き換わります。

```
CALL 'CBLJNEWARRAY' USING CBLJENV 配列J-2の型 配列長-2 配列J-2
```

注※3
byte型配列およびboolean型配列の場合，次に示すMOVE文に置き換わります。

```
MOVE ユーザ用繰り返し項目(添字C-1,添字C-2) TO
     作業用繰り返し項目(添字C-2)
```

オブジェクト型配列の要素を設定する場合



```
CALL 'CBLJARRAYLENGTH' USING CBLJENV 配列J-1 配列長-1.          } ※1

PERFORM VARYING 添字C-1 FROM 1 BY 1 UNTIL 添字C-1 > 配列長-1

  COMPUTE 添字J-1 = 添字C-1 - 1
  CALL 'CBLJGETOBJARRAY' USING CBLJENV 配列J-1 添字J-1 配列J-2  } ※2
  CALL 'CBLJARRAYLENGTH' USING CBLJENV 配列J-2 配列長-2

  PERFORM VARYING 添字C-2 FROM 1 BY 1 UNTIL 添字C-2 > 配列長-2

    COMPUTE 添字J-2 = 添字C-2 - 1
    CALL 'CBLJSETOBJARRAY' USING CBLJENV 配列J-2 添字J-2        } ※3
            ユーザ用繰り返し項目(添字C-1,添字C-2)

  END-PERFORM

  CALL 'CBLJSETOBJARRAY'  USING CBLJENV 配列J-1 添字J-1 配列J-2
END-PERFORM.
```

添字C-1：COBOLの繰り返し項目の添字。添字xに相当する。値は1～x。
添字C-2：COBOLの繰り返し項目の添字。添字yに相当する。値は1～y。
添字J-1：Javaの配列の添字。添字xに相当する。値は0～x-1。
添字J-2：Javaの配列の添字。添字yに相当する。値は0～y-1。
配列J-1：Javaの配列a[x]に相当する2次元配列オブジェクト。
配列J-2：Javaの配列a[][y]に相当する1次元配列オブジェクト。
配列長-1：配列-1の長さ。
配列長-2：配列-2の長さ。

注※1
集団項目データ交換プログラムの場合，次に示すように配列を新規作成するCALL文に置き換わります。

```
CALL 'CBLJNEWARRAY' USING CBLJENV 配列J-1の型 配列長-1 配列J-1.
```

注※2
集団項目データ交換プログラムの場合，次に示すように配列を新規作成するCALL文に置き換わります。

```
CALL 'CBLJNEWARRAY' USING CBLJENV 配列J-2の型 配列長-2 配列J-2
```

注※3
集団項目データ交換プログラムの場合，String型配列を設定するとき，次に示すように英数字項目→Stringに変換するCALL文に置き換わります。

```
CALL 'CBLJXTOSTRING' USING CBLJENV
          ユーザ用繰り返し項目(添字C-1,添字C-2)
          英数字項目の長さを格納するデータ項目
          TEMP-ARRAY-ADDR
CALL 'CBLJSETOBJARRAY' USING CBLJENV 配列J-2 添字J-2
          TEMP-ARRAY-ADDR
```

## (5)　使用するファイルとファイルの入出力

プログラム作成支援ツールが使用するファイルとファイルの入出力について説明します。

### (a)　使用するファイル

プログラム作成支援ツールが使用するファイルを次の表に示します。

表 7-8　プログラム作成支援ツールが使用するファイル

| 項番 | 種別 | | 拡張子 |
|---|---|---|---|
| 1 | 入力ファイル（Java プログラム） | 実行ファイル | .class |

| 項番 | 種別 | | 拡張子 |
|---|---|---|---|
| 2 | | アーカイブファイル | .jar |
| 3 | 出力ファイル（COBOL プログラム） | 固定形式正書法 | .cbl |
| 4 | | 自由形式正書法 | .cbf |

## (b) ファイルの入力

プログラム作成支援ツールが入力ファイル（Java 実行ファイルおよび Java アーカイブファイル）を検索するフォルダと条件を次の表に示します。

### 表 7-9 Java 実行ファイルおよび Java アーカイブファイルを検索するフォルダと条件

| 項番 | 検索パス | 条件 |
|---|---|---|
| 1 | -ClassPath オプションに指定したフォルダ | -ClassPath オプションが指定されている。 |
| 2 | 環境変数 CLASSPATH に指定したフォルダ | -ClassPath オプションが指定されていない。 |
| 3 | カレントフォルダ | -ClassPath オプションが指定されていないで，かつ環境変数 CLASSPATH が定義されていない。 |

-ClassPath オプションや環境変数 CLASSPATH に複数のフォルダを指定した場合，先頭のフォルダから順に検索します。

フォルダ A→フォルダ B の順にクラスを検索する例を次に示します。

例）

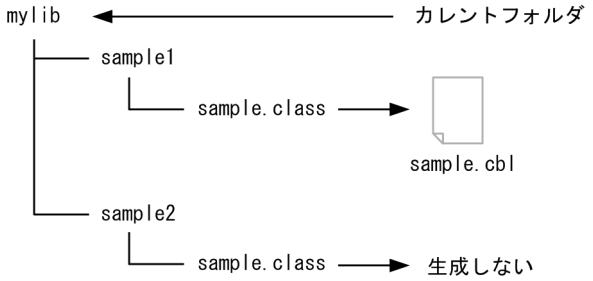
- -ClassPath オプションにフォルダ A とフォルダ B を指定した。
- 環境変数 CLASSPATH にフォルダ C を指定した。

注意事項

-Class オプションで指定したクラス名が完全修飾クラス名でない場合，検索パスの中にパッケージ名が異なる同名のクラスが存在するときは，最初に見つかったクラスだけが解析対象となります。

検索パスの中にパッケージ名が異なる同名のクラスが存在するときの例を次の図に示します。

### 図 7-4 検索パスの中にパッケージ名が異なる同名のクラスが存在するとき




上記の図では，sample1 の下の sample.class が先に見つかるため，sample2 の下の sample.class は検索結果に含まれません。

-Class オプションに完全修飾クラス名を指定するか，重複するクラス名がないように検索パスを再設定してから，ファイルを再生成してください。

### (c) ファイルの出力

ファイルの出力先の標準値はカレントフォルダです。出力先はプログラム作成支援ツールの-OutDir オプションで変更できます。

出力ファイルのファイル名は，解析対象のクラス名を基に設定します。出力ファイル名の命名規則を次に示します。

出力ファイル名の命名規則

- Java クラス利用サンプルの生成で生成する出力ファイルの名前（拡張子除く）は，解析対象のクラス名をそのまま使用します。
- 集団項目データ変換プログラムの生成で生成する出力ファイルの名前（拡張子除く）は，次に示すとおりです。

集団項目データ交換用データ定義

*解析対象のクラス名*_Map_COPY

集団項目データ交換プログラム

*解析対象のクラス名*_Map

- 出力ファイルの正書法によって拡張子を次のとおりに変更します。
  固定形式正書法：.cbl
  自由形式正書法：.cbf

出力ファイル名の例を次に示します。

#### 表 7-10　クラス名と出力ファイル名の対応例

| 項番 | クラス名 | 出力ファイル名 | 備考 |
|---|---|---|---|
| 1 | sample.sampleClass | sample.sampleClass.cbl | Java クラス利用サンプルを生成する場合の出力ファイルです。 |
| 2 | | sample.sampleClass_Map_COPY.cbl | 集団項目データ交換プログラムを生成する場合の出力ファイル（集団項目データ交換用データ定義）です。 |
| 3 | | sample.sampleClass_Map.cbl | 集団項目データ交換プログラムを生成する場合の出力ファイル（集団項目データ交換プログラム）です。 |
| 4 | SampleClass<br>（-Format オプションに free を指定する） | SampleClass.cbf | 正書法に自由形式正書法を指定した場合，拡張子を.cbf とします。 |

**注意事項**

COBOL コード生成中にエラーが発生した場合，出力途中の COBOL ファイルは削除します。

## 7.2.2　Java クラス利用サンプルの生成

プログラム作成支援ツールの-Type オプションに Sample を指定すると，Java プログラム呼び出し機能によって Java のフィールドやメソッドを利用する COBOL コードのサンプル（以降，Java クラス利用サンプルと表記します）を作成します。

```
cbl2kjgen -Type Sample 〔オプション〕 Javaプログラム指定オプション
```

Java プログラム指定オプションは，次の 2 種類の組み合わせで指定します。

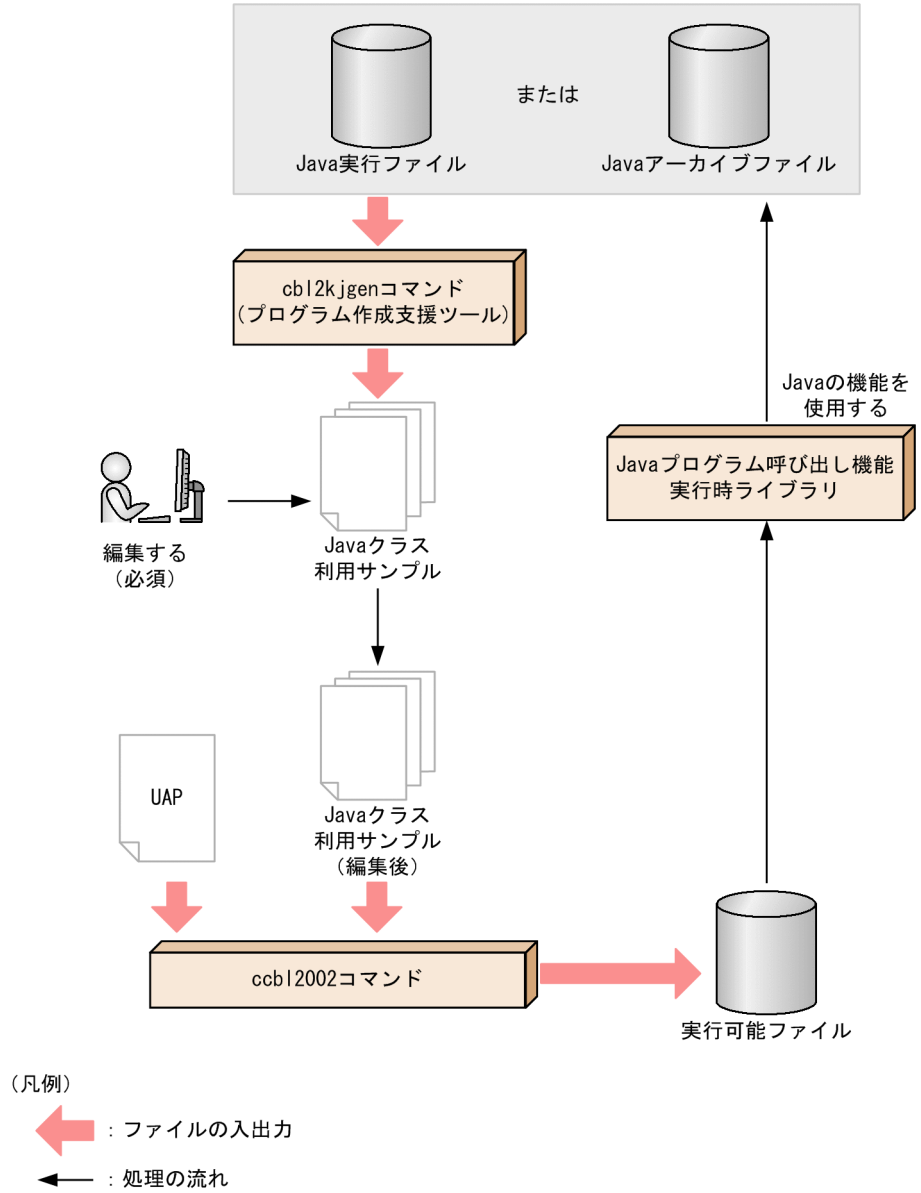
- -Class Java クラス名
- -Jar Java アーカイブファイル名

-Class オプションに指定した Java クラスを Java 実行ファイル（.class）または Java アーカイブファイル（.jar）から検索して解析対象とします。-Jar オプションだけを指定すると，アーカイブファイルの中からプログラム作成支援ツールが解析対象とするすべてのクラスを解析します。

-Class オプションおよび-Jar オプションについては，「7.2　プログラム作成支援ツールの機能」および「7.2.1　プログラム作成支援ツールの機能範囲」の「(5)　使用するファイルとファイルの入出力」を参照してください。

Java クラス利用サンプルの生成の概要を次の図に示します。

図 7-5　Java クラス利用サンプルの生成の概要




作成した Java クラス利用サンプルには，Java プログラムに含まれるフィールドやメソッドを，Java プログラム呼び出し機能を使用して呼び出す COBOL プログラムが記述されます。

Java クラス利用サンプルを使用する場合は，作成されたプログラムを編集し，必要な定義や処理を選択して，Java クラスを利用する COBOL プログラムを完成させます。

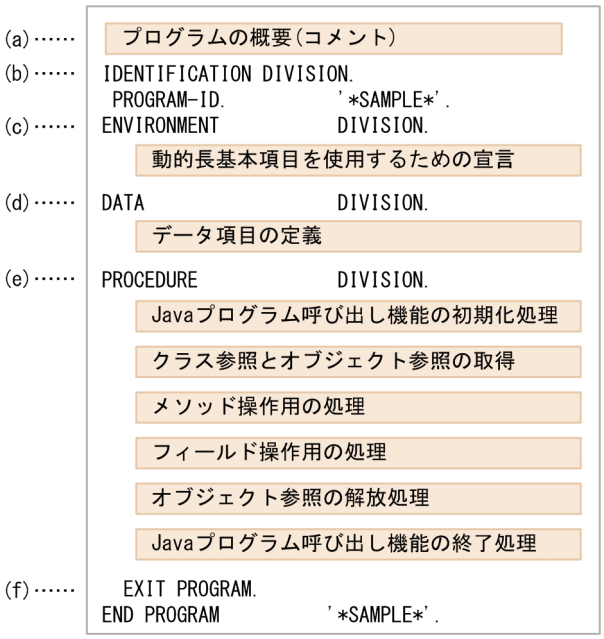
**注意事項**

プログラム作成支援ツールで生成した Java クラス利用サンプルは，未修正のまま COBOL2002 コンパイラでコンパイルするとコンパイルエラーとなります。プログラム名や Java のフィールド値などを設定してください。

## (1)　Java クラス利用サンプルの構造

Java クラス利用サンプルの構造を次の図に示します。

図 7-6　Java クラス利用サンプルの構造




Java クラス利用サンプルとして，COBOL ソースファイルが Java クラス単位に一つずつ作成されます。

Java クラス利用サンプルにはプログラムが一つあり，その中に Java プログラム呼び出し機能を使用するための処理が作成されます。また，各処理で使用するデータ項目の定義も自動生成されます。

不要な処理を削除したり，データ名を変更したりするなど，生成されたプログラムを編集して，目的のプログラムを完成させてください。

図中の(a)～(f)の処理について説明します。なお，(d)～(f)の処理は一つの処理につき，次に示す内容を生成します。

- 処理の概要（コメント）
- Java プログラム呼び出し機能による処理

## (a)　プログラムの概要（コメント）

プログラムの概要を先頭のコメントとして記載します。記載する内容を次に示します。

| 項番 | 項目 | 内容（引用符（"）で囲んだ文字列は固定） |
|---|---|---|
| 1 | タイトル | 次の内容を記載します。<br>"COBOL2002 Java プログラム呼び出し機能 クラス利用サンプルソース" |
| 2 | 利用クラス | 対象のクラス名を記載します。 |
| 3 | cbl2kjgen のバージョン | cbl2kjgen のバージョンを記載します。<br>例："03-02" |
| 4 | オプション | cbl2kjgen に指定されたオプションを記載します。1 行につき 1 オプションを記載します。 |

| 項番 | 項目 | 内容（引用符（"）で囲んだ文字列は固定） |
|---|---|---|
| 5 | 生成日時 | Java クラス利用サンプルの生成日時を記載します。<br>形式は YYYY/mm/dd HH:MM:SS です。 |

### (b)　見出し部

プログラム名に"*SAMPLE*"を指定します。※

注※

"*SAMPLE*"は COBOL で使用できないプログラム名です。コンパイルするために，プログラム名を変更してください。

### (c)　環境部

動的長基本項目を使用するため，次に示すように構成節を記載します。

```
ENVIRONMENT           DIVISION.
CONFIGURATION         SECTION.
 SPECIAL-NAMES.
  DYNAMIC LENGTH STRUCTURE C-STRING IS C-STATIC-STRUCTURE.
```

### (d)　データ部

作業場所節に次に示す順番でデータ項目を定義します。ただし，Java クラス利用サンプル中で使用しないデータ項目は定義しません。

1. CBLJENV 集団項目
2. クラス参照とオブジェクト参照
3. クラス名，フィールド名，およびメソッド名
4. パラメタ
5. 引数リスト
6. String⇔英数字項目変換領域
7. 配列オブジェクト⇔可変長繰り返し項目変換領域

各データ項目について説明します。

1. CBLJENV 集団項目

   CBLJENV 集団項目を定義します。CBLJOPTION-1 には初期値として'Java VM オプション文字列'が設定されています。この値を Java VM に渡すオプション文字列（例：'-Djava.class.path=java'）に変更する必要があります。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。

```
*>------------------------------------------------------------------
*> CBLJENV集団項目の定義
*>------------------------------------------------------------------
01  CBLJENV.
 02  CBLJENVCORE     USAGE POINTER VALUE NULL.
 02  CBLJEXCEPTION   USAGE POINTER VALUE NULL.
 02  CBLJFLAGS       PIC 1(32) USAGE BIT VALUE ALL B'0'.
 02  CBLJSTRMAXLEN   PIC S9(9) USAGE COMP VALUE 256.
 02  CBLJVMOPTIONS.
   03  CBLJOPTCOUNT  PIC S9(9) USAGE COMP VALUE 1.
   03  CBLJOPTION-1  PIC X(256) VALUE 'Java VMオプション文字列'.
```

2. クラス参照とオブジェクト参照

クラス参照とオブジェクト参照を定義します。クラス参照とオブジェクト参照は 1 クラスにつき一つずつ定義します。

```
*>------------------------------------------------------------------
*> クラス参照とオブジェクト参照
*>------------------------------------------------------------------
01  CLASSREF          USAGE POINTER.
01  OBJREF            USAGE POINTER.
```

3. クラス名，フィールド名，およびメソッド名

クラス名，フィールド名，およびメソッド名を設定する英数字の動的長基本項目を定義します。初期値には該当するクラス名，フィールド名またはメソッド名を設定します。

```
*>------------------------------------------------------------------
*> クラス名，フィールド名，メソッド名の定義
*>------------------------------------------------------------------
 01  SampleClass       PIC X DYNAMIC C-STRING VALUE 'SampleClass'.
 01  classID           PIC X DYNAMIC C-STRING VALUE 'classID'.
 01  execData          PIC X DYNAMIC C-STRING VALUE 'execData'.
 01  execType          PIC X DYNAMIC C-STRING VALUE 'execType'.
 01  memberList        PIC X DYNAMIC C-STRING VALUE 'memberList'.
 01  equals            PIC X DYNAMIC C-STRING VALUE 'equals'.
 01  executeProcess    PIC X DYNAMIC C-STRING VALUE 'executeProc'-
                       'ess'.
 01  getClass          PIC X DYNAMIC C-STRING VALUE 'getClass'.
 01  getExecData       PIC X DYNAMIC C-STRING VALUE 'getExecData'.
 01  getExecType       PIC X DYNAMIC C-STRING VALUE 'getExecType'.
 01  getMemberList     PIC X DYNAMIC C-STRING VALUE 'getMemberLi'-
                       'st'.
```

4. パラメタ

次に示す値を設定するデータ項目を定義します。

- フィールド※
- コンストラクタおよびメソッドに渡す各引数
- メソッドの戻り値

注※

対象となるフィールドについては，「7.2.1　プログラム作成支援ツールの機能範囲」の「(1)　Java プログラムの解析」の「(a)　プログラム作成支援ツールの解析対象」を参照してください。

```
*>-----------------------------------------------------------------
*> パラメタの定義
*>-----------------------------------------------------------------
 01  FIELD1.
   02  FIELD1-TYPE     PIC X(256) VALUE 'Ljava/lang/String;'.
   02  FIELD1-AREA     USAGE POINTER.
01  FIELD2.
   02  FIELD2-TYPE     PIC X(256) VALUE 'Ljava/lang/String;'.
   02  FIELD2-AREA     USAGE POINTER.
 01  FIELD3.
   02  FIELD3-TYPE     PIC X(1) VALUE 'I'.
   02  FILLER          PIC X(7) VALUE ALL LOW-VALUE.
   02  FIELD3-AREA     PIC S9(9) USAGE COMP.
 01  FIELD4.
   02  FIELD4-TYPE     PIC X(256) VALUE '[[I'.
   02  FIELD4-AREA     USAGE POINTER.
 01  ARG2-1.
   02  ARG2-1-TYPE     PIC X(256) VALUE 'Ljava/lang/Object;'.
   02  ARG2-1-AREA     USAGE POINTER.
 01  ARG3-1.
   02  ARG3-1-TYPE     PIC X(1) VALUE 'I'.
   02  FILLER          PIC X(7) VALUE ALL LOW-VALUE.
   02  ARG3-1-AREA     PIC S9(9) USAGE COMP.
 01  ARG11-1.
   02  ARG11-1-TYPE    PIC X(256) VALUE 'Ljava/lang/String;'.
   02  ARG11-1-AREA    USAGE POINTER.
```

5. 引数リスト

Java プログラム呼び出し機能のサービスルーチンに渡す引数リストを定義します。

```
*>-----------------------------------------------------------------
*> 引数リストの定義
*>-----------------------------------------------------------------
 01  ARG-LIST.
   02  ARGPTR          USAGE POINTER OCCURS 3 TIMES.
 01  NO-ARG.
   02  FILLER          USAGE POINTER VALUE NULL.
```

コンストラクタおよびメソッドに引数が一つ以上ある場合は，ARG-LIST を使用します。ARGPTR はクラス内のコンストラクタおよびメソッドが持つ引数の最大個数＋ 1 個を定義します。

コンストラクタおよびメソッドに引数がない場合は NO-ARG を使用します。

6. String⇔英数字項目変換領域

Java プログラム呼び出し機能のサービスルーチンで String と英数字項目の変換を行う際，変換結果を受け取るための領域として使用します。

```
*>-----------------------------------------------------------------
*> String <=> 英数字項目変換領域
*>-----------------------------------------------------------------
 01  WK-ALNUM          PIC X(256).
 01  WK-ALNUM-LEN      PIC S9(9) USAGE COMP VALUE 256.
```

7. 配列オブジェクト⇔可変長繰り返し項目変換領域

Java プログラム呼び出し機能のサービスルーチンで配列オブジェクトと可変長繰り返し項目の変換を行う際，変換結果を受け取るための領域として使用します。

```
*>-----------------------------------------------------------------
*> 配列オブジェクト <=> 可変長繰り返し項目変換領域
*>-----------------------------------------------------------------
 01  TEMP-ARRAY-ADDR   USAGE POINTER.
 01  ARRAY-INDEX-1     PIC S9(9) USAGE COMP.
 01  ARRAY-INDEX-2     PIC S9(9) USAGE COMP.
 01  ARRAY-ADDR-2      USAGE POINTER.
 01  ARRAY-INDEX-J-1   PIC S9(9) USAGE COMP.
 01  FIELD4-TEMP-MAP-LEN PIC S9(9) USAGE COMP VALUE 256.
 01  FIELD4-TEMP-MAP   ADDRESSED BY FIELD4-TEMP-MAP-P.
   02  FIELD4-TEMP-MAP-ELEM PIC S9(9) USAGE COMP OCCURS 256 TIMES
                      DEPENDING ON FIELD4-TEMP-MAP-LEN.
 01  FIELD4-1-MAP-LEN  PIC S9(9) USAGE COMP VALUE 256.
 01  FIELD4-2-MAP-LEN  PIC S9(9) USAGE COMP VALUE 256.
 01  FIELD4-MAP.
   02  FIELD4-1        OCCURS 256 TIMES
                       DEPENDING ON FIELD4-1-MAP-LEN.
     03  FIELD4-2      OCCURS 256 TIMES
                       DEPENDING ON FIELD4-2-MAP-LEN.
       04  FIELD4-ELEM PIC S9(9) USAGE COMP.
```

## (e) 手続き部

手続き部の内容について説明します。

Java プログラム呼び出し機能の初期化処理

CBLJENV 集団項目を初期化する処理です。

```
*>-----------------------------------------------------------------
*> Javaプログラム呼び出し機能の実行環境を初期化する
*>-----------------------------------------------------------------

   CALL 'CBLJINITIALIZE' USING CBLJENV.
```

} CBLJENV集団項目の初期化

クラス参照とオブジェクト参照の取得

クラスおよびそのインスタンス（オブジェクト参照）を取得する処理です。

```
*>-----------------------------------------------------------------
*> SampleClass クラスのクラス参照を取得する
*>
*> インスタンスを生成するか，クラスフィールド，クラスメソッドを
*> 利用する前に実行しなければならない
*>-----------------------------------------------------------------

   CALL 'CBLJGETCLASS' USING CBLJENV SampleClass CLASSREF.
```

} クラス参照の取得

```
*>-----------------------------------------------------------------
*> SampleClass クラスのインスタンスを生成する
*>
*> new SampleClass()
*>
*> インスタンスフィールド，インスタンスメソッドを利用する前に
*> 実行しなければならない
*>-----------------------------------------------------------------

   CALL 'CBLJNEW' USING CBLJENV CLASSREF NO-ARG OBJREF.

   IF RETURN-CODE NOT = 0 THEN
*>       ここに CBLJEXCEPTION 項目を使った例外処理を記述する

*>       処理を続行できないならば，
*>       CBLJFINALIZE サービスルーチンを実行して終了する
       GO TO END-PROC
   END-IF.
```

} オブジェクト参照の取得

} ユーザが例外処理を記載する部分

メソッド操作用の処理

クラスメソッドおよびインスタンスメソッドを呼び出す処理です。対象となるメソッドについては，「7.2.1　プログラム作成支援ツールの機能範囲」の「(1)　Java プログラムの解析」の「(a)　プログラム作成支援ツールの解析対象」を参照してください。

```
*>------------------------------------------------------------
*> SampleClass.javaMethod インスタンスメソッドを呼び出す
*>
*> RTN1 = OBJREF.javaMethod(ARG1-1, ARG1-2)
*>
*> 引数:
*>   ARG1-1            PIC S9(9) USAGE COMP
*>   ARG1-2            PIC X(256)
*> 戻り値:
*>   RTN1              PIC X(256)
*>------------------------------------------------------------

   MOVE ???????????? TO ARG1-1-AREA.
   MOVE '??????????' TO WK-ALNUM.
   CALL 'CBLJXTOSTRING' USING CBLJENV
                        WK-ALNUM WK-ALNUM-LEN ARG1-2-AREA.

   COMPUTE ARGPTR(1) = FUNCTION ADDR(ARG1-1).
   COMPUTE ARGPTR(2) = FUNCTION ADDR(ARG1-2).
   COMPUTE ARGPTR(3) = ZERO.

   CALL 'CBLJINVOKE' USING CBLJENV OBJREF javaMethod
                        ARG-LIST RTN1.

   IF RETURN-CODE NOT = 0 THEN
*>       ここに CBLJEXCEPTION 項目を使った例外処理を記述する
   END-IF.
```

「??…??」はユーザが編集する部分

メソッド呼び出し処理

ユーザが例外処理を記載する部分

フィールド操作用の処理

クラスフィールドおよびインスタンスフィールドにアクセス（取得・設定）する処理です。対象となるフィールドについては，「7.2.1　プログラム作成支援ツールの機能範囲」の「(1)　Java プログラムの解析」の「(a)　プログラム作成支援ツールの解析対象」を参照してください。

```
*>-------------------------------------------------------------
*> SampleClass.classID スタティックフィールドの値を取得する
*>
*> FIELD1 = SampleClass.classID
*>
*>   FIELD1               USAGE POINTER(java/lang/String)
*>-------------------------------------------------------------

   CALL 'CBLJGETSTATICFIELD' USING CBLJENV CLASSREF classID
                    FIELD1.

   CALL 'CBLJSTRINGTOX' USING CBLJENV FIELD1-AREA WK-ALNUM
                    WK-ALNUM-LEN.

*>-------------------------------------------------------------
*> SampleClass.classID スタティックフィールドの値を設定する
*>
*> SampleClass.classID = FIELD1
*>
*>   FIELD1               USAGE POINTER(java/lang/String)
*>-------------------------------------------------------------

   MOVE '??????????' TO WK-ALNUM.
   CALL 'CBLJXTOSTRING' USING CBLJENV
                    WK-ALNUM WK-ALNUM-LEN FIELD1-AREA.

   CALL 'CBLJSETSTATICFIELD' USING CBLJENV
                    CLASSREF classID FIELD1.
```

フィールドの値の取得

フィールドの値の設定（「??･･･??」はユーザが編集する部分）

オブジェクト参照の解放処理

「クラス参照とオブジェクト参照の取得」で取得したオブジェクト参照（インスタンス）を解放する処理です。

```
*>-------------------------------------------------------------
*> SampleClass クラスのインスタンスを解放する
*>
*> 不要になったクラスインスタンスは解放しなければならない
*>-------------------------------------------------------------

   CALL 'CBLJRELEASE' USING CBLJENV OBJREF.
```

Java プログラム呼び出し機能の終了処理

Java プログラム呼び出し機能の実行環境の終了処理です。手続き END-PROC の中に含みます。

```
END-PROC.
*>-------------------------------------------------------------
*> Javaプログラム呼び出し機能の実行環境を終了する
*>-------------------------------------------------------------

   CALL 'CBLJFINALIZE' USING CBLJENV.
```

## (f)　プログラムの終了

プログラムを終了します。

```
    EXIT PROGRAM.

END PROGRAM          '*SAMPLE*'.
```

注意

"*SAMPLE*"は COBOL で使用できないプログラム名です。見出し部で指定したプログラム名に変更してください。見出し部については，「(b)　見出し部」を参照してください。

## (2)　データ項目のマッピング

Java クラス利用サンプル中に各データ項目を定義する際に使用するルールを示します。

JNI に関するデータ項目

JNI に関するデータ項目を次に示します。

| 用途 | データ項目※ |
|---|---|
| Java VM に渡すオプションを 1 個定義する CBLJENV 集団項目を定義する。 | 01 CBLJENV GLOBAL.<br>02 CBLJENVCORE USAGE POINTER VALUE NULL.<br>02 CBLJEXCEPTION USAGE POINTER VALUE NULL.<br>02 CBLJFLAGS PIC 1(32) USAGE BIT VALUE ALL B'0'.<br>02 CBLJSTRMAXLEN PIC S9(9) USAGE COMP VALUE *STRMAXLEN*.<br>02 CBLJVMOPTIONS.<br>03 CBLJOPTCOUNT PIC S9(9) USAGE COMP VALUE 1.<br>03 CBLJOPTION-1 PIC X(*STRMAXLEN*) VALUE '*VMOPT-1*'. |

注※

- "*STRMAXLEN*"の既定値は 256 です。この値はプログラム作成支援ツールの-StrMaxLen オプションで変更できます。
- "*VMOPT-1*"には文字列'Java VM オプション文字列'が入ります。

クラスやオブジェクトに関するデータ項目

クラスやオブジェクトに関するデータ項目を次に示します。

| 項番 | 用途 | データ項目※ |
|---|---|---|
| 1 | クラス名を設定する。 | 01 *classname* PIC X DYNAMIC C-STRING VALUE '*classnameOrg*'. |
| 2 | クラス参照のアドレスを設定する。 | 01 CLASSREF USAGE POINTER. |
| 3 | オブジェクト参照のアドレスを設定する。 | 01 OBJREF USAGE POINTER. |

注※

- "*classname*"はクラス名または-Class オプションで指定した内部名を示します。クラス名または-Class オプションについては，「7.2.1　プログラム作成支援ツールの機能範囲」の「(2)　データ項目のマッピング（共通）」を参照してください。
- "*classnameOrg*"はクラス名を示します。クラス名に含まれるピリオドはスラントに置換します。

メソッド（コンストラクタ，クラスメソッド，インスタンスメソッド）に関するデータ項目

メソッド（コンストラクタ，クラスメソッド，インスタンスメソッド）に関するデータ項目を次に示します。

| 項番 | 用途 | データ項目※ |
|---|---|---|
| 1 | メソッド名を設定する。 | 01 *methodname* PIC X DYNAMIC C-STRING VALUE '*methodname*'. |
| 2 | *n* 番目のメソッドの第*m* 引数を設定するための集団項目を定義する。 | (基本型の場合)<br>01 ARG*n*-*m*.<br>02 ARG*n*-*m*-TYPE PIC X(1) VALUE '*SIGNATURE*'.<br>02 FILLER PIC X(7) VALUE ALL LOW-VALUE.<br>02 ARG*n*-*m*-AREA *TYPE-DEF*.<br>(基本型以外の場合)<br>01 ARG*n*-*m*.<br>02 ARG*n*-*m*-TYPE PIC X(*STRMAXLEN*) VALUE '*SIGNATURE*'.<br>02 ARG*n*-*m*-AREA *TYPE-DEF*. |
| 3 | *n* 番目のメソッドの第*m* 引数が配列オブジェクトの場合，配列の全要素とCOBOLの繰り返し項目をマッピングするための集団項目およびサイズを定義する。 | 01 ARG*n*-*m*-1-MAP-LEN PIC S9(9) USAGE COMP VALUE *ARRAYMAXLEN*.<br>:<br>01 ARG*n*-*m*-*k*-MAP-LEN PIC S9(9) USAGE COMP VALUE *ARRAYMAXLEN*.<br>01 ARG*n*-*m*-MAP.<br>02 ARG*n*-*m*-1 OCCURS *ARRAYMAXLEN* TIMES<br>DEPENDING ON ARG*n*-*m*-1- MAP-LEN.<br>:<br>08 ARG*n*-*m*-*k* OCCURS *ARRAYMAXLEN* TIMES<br>DEPENDING ON ARG*n*-*m*-*k*-MAP-LEN.<br>09 ARG*n*-*m*-ELEM *TYPE-DEF*. |
| 4 | *n* 番目のメソッドの第*m* 引数が配列オブジェクトの場合，配列の最下層の要素とCOBOLの繰り返し項目をマッピングするための作業用集団項目およびサイズを定義する。 | 01 ARG*n*-*m*-TEMP-MAP-LEN PIC S9(9) USAGE COMP<br>VALUE *ARRAYMAXLEN*.<br>01 ARG*n*-*m*-TEMP-MAP ADDRESSED BY ARG*n*-*m*-TEMP-MAP-P.<br>02 ARG*n*-*m*-TEMP-MAP-ELEM TYPE-DEF OCCURS *ARRAYMAXLEN* TIMES<br>DEPENDING ON ARG*n*-*m*-TEMP-MAP-LEN. |
| 5 | *n* 番目のメソッドの戻り値を設定するための集団項目を定義する。 | (基本型の場合)<br>01 RTN*n*.<br>02 RTN*n*-TYPE PIC X(1) VALUE '*SIGNATURE*'.<br>02 FILLER PIC X(7) VALUE ALL LOW-VALUE.<br>02 RTN*n*-AREA *TYPE-DEF*.<br>(基本型以外の場合)<br>01 RTN*n*.<br>02 RTN*n*-TYPE PIC X(*STRMAXLEN*) VALUE '*SIGNATURE*'.<br>02 RTN*n*-AREA *TYPE-DEF*. |
| 6 | *n* 番目のメソッドの戻り値が配列オブジェクトの場合，配列の全要素とCOBOLの繰り返し項目をマッピングするための集団項目およびサイズを定義する。 | 01 RTN*n*-1-MAP-LEN PIC S9(9) USAGE COMP VALUE *ARRAYMAXLEN*.<br>:<br>01 RTN*n*-*k*-MAP-LEN PIC S9(9) USAGE COMP VALUE *ARRAYMAXLEN*.<br>01 RTN*n*-MAP.<br>02 RTN*n*-1 OCCURS *ARRAYMAXLEN* TIMES<br>DEPENDING ON RTN*n*-1-MAP-LEN.<br>:<br>08 RTN*n*-*k* OCCURS *ARRAYMAXLEN* TIMES |

| 項番 | 用途 | データ項目※ |
|---|---|---|
| | | 　　DEPENDING ON RTN*n*-*k*-MAP-LEN.<br>　09　RTN*n*-ELEM　　*TYPE-DEF*. |
| 7 | *n* 番目のメソッドの戻り値が配列オブジェクトの場合，配列の最下層の要素と COBOL の繰り返し項目をマッピングするための作業用集団項目およびサイズを定義する。 | 01　RTN*n*-TEMP-MAP-LEN　PIC S9(9) USAGE COMP<br>　　VALUE *ARRAYMAXLEN*.<br>01　RTN*n*-TEMP-MAP　　ADDRESSED BY RTN*n*-TEMP-MAP-P.<br>　02　RTN*n*-TEMP-MAP-ELEM *TYPE-DEF*　OCCURS *ARRAYMAXLEN* TIMES<br>　　DEPENDING ON RTN*n*-TEMP-MAP-LEN. |

注※

- "*n*"はクラス名およびメソッド名を設定するデータ項目を定義する順番と一致します。
- "*m*"はメソッドの引数の順番と一致します。
- "*k*"は配列オブジェクトの次元数を示します。
- "*methodname*"はメソッド名を示します。メソッド名については，「7.2.1　プログラム作成支援ツールの機能範囲」の「(2)　データ項目のマッピング（共通）」を参照してください。
- "*SIGNATURE*"は Java の型を表す文字列を示します。ただし，引数の型が Class インスタンスの場合，型には java/lang/Class ではなく，実体のクラス名となるように修正してください。指定した Java の型と実体のクラス名が一致しない場合，実行時エラーとなることがあります。Java の型を表す文字列については，「6.1.1　サービスルーチンで使用する引数」の「(3)　パラメタ型集団項目」を参照してください。
- "*TYPE-DEF*"は Java の型に対応する COBOL の型を示します。Java の型に対応する COBOL の型については，「6.1.1　サービスルーチンで使用する引数」の「(3)　パラメタ型集団項目」を参照してください。
- "*STRMAXLEN*"の既定値は 256 です。この値はプログラム作成支援ツールの-StrMaxLen オプションで変更できます。

フィールド（クラスフィールド，インスタンスフィールド）に関するデータ項目

フィールド（クラスフィールド，インスタンスフィールド）に関するデータ項目を次に示します。

| 項番 | 用途 | データ項目※ |
|---|---|---|
| 1 | フィールド名を設定する。 | 01　*fieldname* PIC X DYNAMIC C-STRING VALUE '*fieldname*'. |
| 2 | *n* 番目のフィールドの値を設定するための集団項目を定義する。 | (基本型の場合)<br>01　FIELD*n*.<br>　02　FIELD*n*-TYPE PIC X(1) VALUE '*SIGNATURE*'.<br>　02　FILLER　　PIC X(7) VALUE ALL LOW-VALUE.<br>　02　FIELD*n*-AREA *TYPE-DEF*.<br>(基本型以外の場合)<br>01　FIELD*n*.<br>　02　FIELD*n*-TYPE PIC X(*STRMAXLEN*) VALUE '*SIGNATURE*'.<br>　02　FIELD*n*-AREA *TYPE-DEF*. |

| 項番 | 用途 | データ項目※ |
|---|---|---|
| 3 | *n* 番目のフィールドが配列オブジェクトの場合，配列の全要素とCOBOL の繰り返し項目をマッピングするための集団項目およびサイズを定義する。 | 01 FIELD*n*-1-MAP-LEN PIC S9(9) USAGE COMP VALUE *ARRAYMAXLEN*.<br>:<br>01 FIELD*n*-*m*-MAP-LEN PIC S9(9) USAGE COMP VALUE *ARRAYMAXLEN*.<br>01 FIELD*n*-MAP.<br>02 FIELD*n*-1 OCCURS *ARRAYMAXLEN* TIMES<br>DEPENDING ON FIELD*n*-1-MAP-LEN.<br>:<br>08 FIELD*n*-*m* OCCURS *ARRAYMAXLEN* TIMES<br>DEPENDING ON FIELD*n*-*m*-MAP-LEN.<br>09 FIELD*n*-ELEM *TYPE-DEF*. |
| 4 | *n* 番目のフィールドが配列オブジェクトの場合，配列の最下層の要素とCOBOL の繰り返し項目をマッピングするための作業用集団項目およびサイズを定義する。 | 01 FIELD*n*-TEMP-MAP-LEN PIC S9(9) USAGE COMP VALUE *ARRAYMAXLEN*.<br>01 FIELD*n*-TEMP-MAP ADDRESSED BY FIELD*n*-TEMP-MAP-P.<br>02 FIELD*n*-TEMP-MAP-ELEM *TYPE-DEF* OCCURS *ARRAYMAXLEN* TIMES<br>DEPENDING ON FIELD*n*-TEMP-MAP-LEN. |

注※

- "*n*"はフィールド名を設定するデータ項目を定義する順番と一致します。
- "*m*"は配列オブジェクトの次元数を示します。
- "*fieldname*"はフィールド名を示します。フィールド名については，「7.2.1　プログラム作成支援ツールの機能範囲」の「(2)　データ項目のマッピング（共通）」を参照してください。
- "*SIGNATURE*"は Java の型を表す文字列を示します。ただし，引数の型が Class インスタンスの場合，型には java/lang/Class ではなく，実体のクラス名となるように修正してください。指定した Java の型と実体のクラス名が一致しない場合，実行時エラーとなることがあります。Java の型を表す文字列については，「6.1.1　サービスルーチンで使用する引数」の「(3)　パラメタ型集団項目」を参照してください。
- "*TYPE-DEF*"は Java の型に対応する COBOL の型を示します。Java の型に対応する COBOL の型については，「6.1.1　サービスルーチンで使用する引数」の「(3)　パラメタ型集団項目」を参照してください。
- "*STRMAXLEN*"の既定値は 256 です。この値はプログラム作成支援ツールの-StrMaxLen オプションで変更できます。
- "*ARRAYMAXLEN*"の既定値は 256 です。この値はプログラム作成支援ツールの-MaxArrayLength オプションで変更できます。

## (3)　生成後の使用方法

作成した Java クラス利用サンプルは，そのままコンパイルして使用できません。少なくとも次の項目を修正する必要があります。

- プログラム名（*SAMPLE*）の変更

- 任意の定数やデータ項目を記述する必要がある個所（???…?）の変更
- 不要な処理（フィールド操作，メソッド呼び出しなど）の選択と削除

Java クラス利用サンプルの例を次に示します。

コマンド

```
cbl2kjgen -Type Sample -StrMaxLen 256 -Class mylib.sample.SampleClass
```

mylib¥sample¥SampleClass.java の内容

```
package mylib.sample;
public class SampleClass {
   /** クラスID */
   public static String classID = "SampleClass";

   /** 実行種別 */
   private       int    execType;
   /** 実行データ */
   private       String execData;

   /** constructor */
   public SampleClass() {
      execType = 0;
      execData = "";
   }
   /**
    * executeProcess
    * @return 実行結果
    */
   public boolean executeProcess(int flag) {
            :
      return true;
   }
   /**
    * 実行種別を取得する
    * @return 実行種別
    */
   public int getExecType() {
      return execType;
   }
   /**
    * 実行種別を設定する
    */
   public void setExecType(int type) {
      execType = type;
   }
   /**
    * 実行データを取得する
    * @return 実行データ
    */
   public String getExecData() {
      return execData;
   }
   /**
    * 実行データを設定する
    */
   public void setExecData(String data) {
      execData = data;
   }
}
```

mylib.sample.SampleClass.cbl の内容

```
*>-----------------------------------------------------------------------
*> COBOL2002 Javaプログラム呼び出し機能 クラス利用サンプルソース
*>
*> 利用クラス: mylib.sample.SampleClass
*> cbl2kjgen バージョン: 03-05
*>           オプション: -Type Sample
*>                       -Class mylib.sample.SampleClass
*>                       -StrMaxLen 256
*>
*> 生成日時: 2016/01/29 13:00:00
*>-----------------------------------------------------------------------
 IDENTIFICATION        DIVISION.
  PROGRAM-ID.          '*SAMPLE*'.
 ENVIRONMENT           DIVISION.
 CONFIGURATION         SECTION.
  SPECIAL-NAMES.
   DYNAMIC LENGTH STRUCTURE C-STRING IS C-STATIC-STRUCTURE.
 DATA                  DIVISION.
 WORKING-STORAGE       SECTION.

*>-----------------------------------------------------------------------
*> CBLJENV集団項目の定義
*>-----------------------------------------------------------------------
 01  CBLJENV.
   02  CBLJENVCORE     USAGE POINTER VALUE NULL.
   02  CBLJEXCEPTION   USAGE POINTER VALUE NULL.
   02  CBLJFLAGS       PIC 1(32) USAGE BIT VALUE ALL B'0'.
   02  CBLJSTRMAXLEN   PIC S9(9) USAGE COMP VALUE 256.
   02  CBLJVMOPTIONS.
     03  CBLJOPTCOUNT  PIC S9(9) USAGE COMP VALUE 1.
     03  CBLJOPTION-1  PIC X(256) VALUE 'Java VMオプション文字列'.

*>-----------------------------------------------------------------------
*> クラス参照とオブジェクト参照
*>-----------------------------------------------------------------------
 01  CLASSREF          USAGE POINTER.
 01  OBJREF            USAGE POINTER.

*>-----------------------------------------------------------------------
*> クラス名，フィールド名，メソッド名の定義
*>-----------------------------------------------------------------------
 01  SampleClass       PIC X DYNAMIC C-STRING VALUE 'mylib/sampl'-
                       'e/SampleClass'.
 01  classID           PIC X DYNAMIC C-STRING VALUE 'classID'.
 01  execData          PIC X DYNAMIC C-STRING VALUE 'execData'.
 01  execType          PIC X DYNAMIC C-STRING VALUE 'execType'.
 01  equals            PIC X DYNAMIC C-STRING VALUE 'equals'.
 01  executeProcess    PIC X DYNAMIC C-STRING VALUE 'executeProc'-
                       'ess'.
 01  getClass          PIC X DYNAMIC C-STRING VALUE 'getClass'.
 01  getExecData       PIC X DYNAMIC C-STRING VALUE 'getExecData'.
 01  getExecType       PIC X DYNAMIC C-STRING VALUE 'getExecType'.
 01  hashCode          PIC X DYNAMIC C-STRING VALUE 'hashCode'.
 01  notify            PIC X DYNAMIC C-STRING VALUE 'notify'.
 01  notifyAll         PIC X DYNAMIC C-STRING VALUE 'notifyAll'.
 01  setExecData       PIC X DYNAMIC C-STRING VALUE 'setExecData'.
 01  setExecType       PIC X DYNAMIC C-STRING VALUE 'setExecType'.
 01  toString          PIC X DYNAMIC C-STRING VALUE 'toString'.
 01  wait              PIC X DYNAMIC C-STRING VALUE 'wait'.
*> 01 wait             PIC X DYNAMIC C-STRING VALUE 'wait'.
*> 01 wait             PIC X DYNAMIC C-STRING VALUE 'wait'.
```

```
*>-------------------------------------------------------------
*> パラメタの定義
*>-------------------------------------------------------------
 01  FIELD1.
   02  FIELD1-TYPE     PIC X(256) VALUE 'Ljava/lang/String;'.
   02  FIELD1-AREA     USAGE POINTER.
 01  FIELD2.
   02  FIELD2-TYPE     PIC X(256) VALUE 'Ljava/lang/String;'.
   02  FIELD2-AREA     USAGE POINTER.
 01  FIELD3.
   02  FIELD3-TYPE     PIC X(1) VALUE 'I'.
   02  FILLER          PIC X(7) VALUE ALL LOW-VALUE.
   02  FIELD3-AREA     PIC S9(9) USAGE COMP.
 01  ARG2-1.
   02  ARG2-1-TYPE     PIC X(256) VALUE 'Ljava/lang/Object;'.
   02  ARG2-1-AREA     USAGE POINTER.
 01  ARG3-1.
   02  ARG3-1-TYPE     PIC X(1) VALUE 'I'.
   02  FILLER          PIC X(7) VALUE ALL LOW-VALUE.
   02  ARG3-1-AREA     PIC S9(9) USAGE COMP.
 01  ARG10-1.
   02  ARG10-1-TYPE    PIC X(256) VALUE 'Ljava/lang/String;'.
   02  ARG10-1-AREA    USAGE POINTER.
 01  ARG11-1.
   02  ARG11-1-TYPE    PIC X(1) VALUE 'I'.
   02  FILLER          PIC X(7) VALUE ALL LOW-VALUE.
   02  ARG11-1-AREA    PIC S9(9) USAGE COMP.
 01  ARG14-1.
   02  ARG14-1-TYPE    PIC X(1) VALUE 'J'.
   02  FILLER          PIC X(7) VALUE ALL LOW-VALUE.
   02  ARG14-1-AREA    PIC S9(18) USAGE COMP.
 01  ARG15-1.
   02  ARG15-1-TYPE    PIC X(1) VALUE 'J'.
   02  FILLER          PIC X(7) VALUE ALL LOW-VALUE.
   02  ARG15-1-AREA    PIC S9(18) USAGE COMP.
 01  ARG15-2.
   02  ARG15-2-TYPE    PIC X(1) VALUE 'I'.
   02  FILLER          PIC X(7) VALUE ALL LOW-VALUE.
   02  ARG15-2-AREA    PIC S9(9) USAGE COMP.
```

```
 01  RTN2.
   02  RTN2-TYPE       PIC X(1) VALUE 'Z'.
   02  FILLER          PIC X(7) VALUE ALL LOW-VALUE.
   02  RTN2-AREA       PIC X(1).
 01  RTN3.
   02  RTN3-TYPE       PIC X(1) VALUE 'Z'.
   02  FILLER          PIC X(7) VALUE ALL LOW-VALUE.
   02  RTN3-AREA       PIC X(1).
 01  RTN4.
   02  RTN4-TYPE       PIC X(256) VALUE 'Ljava/lang/Class;'.
   02  RTN4-AREA       USAGE POINTER.
 01  RTN5.
   02  RTN5-TYPE       PIC X(256) VALUE 'Ljava/lang/String;'.
   02  RTN5-AREA       USAGE POINTER.
 01  RTN6.
   02  RTN6-TYPE       PIC X(1) VALUE 'I'.
   02  FILLER          PIC X(7) VALUE ALL LOW-VALUE.
   02  RTN6-AREA       PIC S9(9) USAGE COMP.
 01  RTN7.
   02  RTN7-TYPE       PIC X(1) VALUE 'I'.
   02  FILLER          PIC X(7) VALUE ALL LOW-VALUE.
   02  RTN7-AREA       PIC S9(9) USAGE COMP.
 01  RTN12.
   02  RTN12-TYPE      PIC X(256) VALUE 'Ljava/lang/String;'.
   02  RTN12-AREA      USAGE POINTER.
 01  RTN-VOID.
   02  RTN-TYPE        PIC X(1) VALUE 'V'.

*>-----------------------------------------------------------------
*> 引数リストの定義
*>-----------------------------------------------------------------
 01  ARG-LIST.
   02  ARGPTR          USAGE POINTER OCCURS 3 TIMES.
 01  NO-ARG.
   02  FILLER          USAGE POINTER VALUE NULL.

*>-----------------------------------------------------------------
*> String <=> 英数字項目変換領域
*>-----------------------------------------------------------------
 01  WK-ALNUM          PIC X(256).
 01  WK-ALNUM-LEN      PIC S9(9) USAGE COMP VALUE 256.
 PROCEDURE            DIVISION.

*>-----------------------------------------------------------------
*> Javaプログラム呼び出し機能の実行環境を初期化する
*>-----------------------------------------------------------------

     CALL 'CBLJINITIALIZE' USING CBLJENV.

*>-----------------------------------------------------------------
*> mylib.sample.SampleClass クラスのクラス参照を取得する
*>
*> インスタンスを生成するか，クラスフィールド，クラスメソッドを
*> 利用する前に実行しなければならない
*>-----------------------------------------------------------------

     CALL 'CBLJGETCLASS' USING CBLJENV SampleClass CLASSREF.
```

```
*>-----------------------------------------------------------------
*> SampleClass クラスのインスタンスを生成する
*>
*> new SampleClass()
*>
*>
*> インスタンスフィールド，インスタンスメソッドを利用する前に
*> 実行しなければならない
*>-----------------------------------------------------------------

     CALL 'CBLJNEW' USING CBLJENV CLASSREF NO-ARG OBJREF.

     IF RETURN-CODE NOT = 0  THEN
*>       ここに CBLJEXCEPTION 項目を使った例外処理を記述する

*>       処理を続行できないならば，
*>       CBLJFINALIZE サービスルーチンを実行して終了する
          GO TO END-PROC
     END-IF.

*>-----------------------------------------------------------------
*> SampleClass.equals インスタンスメソッドを呼び出す
*>
*> RTN2 = OBJREF.equals(ARG2-1)
*>
*> 引数：
*>   ARG2-1            USAGE POINTER
*>
*> 戻り値：
*>     RTN2              PIC X(1)
*>
*>-----------------------------------------------------------------


     COMPUTE ARGPTR(1) = FUNCTION ADDR(ARG2-1).
     COMPUTE ARGPTR(2) = ZERO.

     CALL 'CBLJINVOKE' USING CBLJENV OBJREF equals ARG-LIST RTN2.

     IF RETURN-CODE NOT = 0 THEN
*>       ここに CBLJEXCEPTION 項目を使った例外処理を記述する
     END-IF.
```

```
*>-----------------------------------------------------------------
*> SampleClass.executeProcess インスタンスメソッドを呼び出す
*>
*> RTN3 = OBJREF.executeProcess(ARG3-1)
*>
*> 引数：
*>   ARG3-1            PIC S9(9) USAGE COMP
*>
*> 戻り値：
*>     RTN3              PIC X(1)
*>
*>-----------------------------------------------------------------

     MOVE ???????????? TO ARG3-1-AREA.

     COMPUTE ARGPTR(1) = FUNCTION ADDR(ARG3-1).
     COMPUTE ARGPTR(2) = ZERO.

     CALL 'CBLJINVOKE' USING CBLJENV OBJREF executeProcess
                       ARG-LIST RTN3.

     IF RETURN-CODE NOT = 0 THEN
*>       ここに CBLJEXCEPTION 項目を使った例外処理を記述する
     END-IF.

*>-----------------------------------------------------------------
*> SampleClass.getClass インスタンスメソッドを呼び出す
*>
*> RTN4 = OBJREF.getClass()
*>
*>
*> 戻り値：
*>     RTN4              USAGE POINTER
*>
*>-----------------------------------------------------------------

     CALL 'CBLJINVOKE' USING CBLJENV OBJREF getClass NO-ARG RTN4.

     IF RETURN-CODE NOT = 0 THEN
*>       ここに CBLJEXCEPTION 項目を使った例外処理を記述する
     END-IF.

*>-----------------------------------------------------------------
*> SampleClass.getExecData インスタンスメソッドを呼び出す
*>
*> RTN5 = OBJREF.getExecData()
*>
*>
*> 戻り値：
*>     RTN5              PIC X(256)
*>
*>-----------------------------------------------------------------

     CALL 'CBLJINVOKE' USING CBLJENV OBJREF getExecData NO-ARG
                       RTN5.

     IF RETURN-CODE NOT = 0 THEN
*>       ここに CBLJEXCEPTION 項目を使った例外処理を記述する
     END-IF.
```

```
*>-----------------------------------------------------------------
*> SampleClass.getExecType インスタンスメソッドを呼び出す
*>
*> RTN6 = OBJREF.getExecType()
*>
*>
*> 戻り値：
*>     RTN6              PIC S9(9) USAGE COMP
*>
*>-----------------------------------------------------------------

     CALL 'CBLJINVOKE' USING CBLJENV OBJREF getExecType NO-ARG
                       RTN6.

     IF RETURN-CODE NOT = 0 THEN
*>       ここに CBLJEXCEPTION 項目を使った例外処理を記述する
     END-IF.

*>-----------------------------------------------------------------
*> SampleClass.hashCode インスタンスメソッドを呼び出す
*>
*> RTN7 = OBJREF.hashCode()
*>
*>
*> 戻り値：
*>     RTN7              PIC S9(9) USAGE COMP
*>
*>-----------------------------------------------------------------

     CALL 'CBLJINVOKE' USING CBLJENV OBJREF hashCode NO-ARG RTN7.

     IF RETURN-CODE NOT = 0 THEN
*>       ここに CBLJEXCEPTION 項目を使った例外処理を記述する
     END-IF.

*>-----------------------------------------------------------------
*> SampleClass.notify インスタンスメソッドを呼び出す
*>
*> OBJREF.notify()
*>
*>
*>
*>-----------------------------------------------------------------

     CALL 'CBLJINVOKE' USING CBLJENV OBJREF notify NO-ARG
                        RTN-VOID.

     IF RETURN-CODE NOT = 0 THEN
*>       ここに CBLJEXCEPTION 項目を使った例外処理を記述する
     END-IF.
```

```
*>-----------------------------------------------------------------
*> SampleClass.notifyAll インスタンスメソッドを呼び出す
*>
*> OBJREF.notifyAll()
*>
*>
*>
*>-----------------------------------------------------------------

     CALL 'CBLJINVOKE' USING CBLJENV OBJREF notifyAll NO-ARG
                       RTN-VOID.

     IF RETURN-CODE NOT = 0 THEN
*>       ここに CBLJEXCEPTION 項目を使った例外処理を記述する
     END-IF.

*>-----------------------------------------------------------------
*> SampleClass.setExecData インスタンスメソッドを呼び出す
*>
*> OBJREF.setExecData(ARG1O-1)
*>
*> 引数：
*>   ARG1O-1           PIC X(256)
*>
*>
*>-----------------------------------------------------------------

     MOVE '??????????' TO WK-ALNUM.
     CALL 'CBLJXTOSTRING' USING CBLJENV WK-ALNUM WK-ALNUM-LEN
                       ARG1O-1-AREA.

     COMPUTE ARGPTR(1) = FUNCTION ADDR(ARG1O-1).
     COMPUTE ARGPTR(2) = ZERO.

     CALL 'CBLJINVOKE' USING CBLJENV OBJREF setExecData ARG-LIST
                       RTN-VOID.

     IF RETURN-CODE NOT = 0 THEN
*>       ここに CBLJEXCEPTION 項目を使った例外処理を記述する
     END-IF.

*>-----------------------------------------------------------------
*> SampleClass.setExecType インスタンスメソッドを呼び出す
*>
*> OBJREF.setExecType(ARG11-1)
*>
*> 引数：
*>   ARG11-1           PIC S9(9) USAGE COMP
*>
*>
*>-----------------------------------------------------------------

     MOVE ???????????? TO ARG11-1-AREA.

     COMPUTE ARGPTR(1) = FUNCTION ADDR(ARG11-1).
     COMPUTE ARGPTR(2) = ZERO.

     CALL 'CBLJINVOKE' USING CBLJENV OBJREF setExecType ARG-LIST
                       RTN-VOID.

     IF RETURN-CODE NOT = 0 THEN
*>       ここに CBLJEXCEPTION 項目を使った例外処理を記述する
     END-IF.
```

```
*>-----------------------------------------------------------------
*> SampleClass.toString インスタンスメソッドを呼び出す
*>
*> RTN12 = OBJREF.toString()
*>
*>
*> 戻り値：
*>     RTN12             PIC X(256)
*>
*>-----------------------------------------------------------------

     CALL 'CBLJINVOKE' USING CBLJENV OBJREF toString NO-ARG RTN12.

     IF RETURN-CODE NOT = 0 THEN
*>       ここに CBLJEXCEPTION 項目を使った例外処理を記述する
     END-IF.

*>-----------------------------------------------------------------
*> SampleClass.wait インスタンスメソッドを呼び出す
*>
*> OBJREF.wait()
*>
*>
*>
*>-----------------------------------------------------------------

     CALL 'CBLJINVOKE' USING CBLJENV OBJREF wait NO-ARG RTN-VOID.

     IF RETURN-CODE NOT = 0 THEN
*>       ここに CBLJEXCEPTION 項目を使った例外処理を記述する
     END-IF.

*>-----------------------------------------------------------------
*> SampleClass.wait インスタンスメソッドを呼び出す
*>
*> OBJREF.wait(ARG14-1)
*>
*> 引数：
*>   ARG14-1           PIC S9(18) USAGE COMP
*>
*>
*>-----------------------------------------------------------------

     MOVE ???????????? TO ARG14-1-AREA.

     COMPUTE ARGPTR(1) = FUNCTION ADDR(ARG14-1).
     COMPUTE ARGPTR(2) = ZERO.

     CALL 'CBLJINVOKE' USING CBLJENV OBJREF wait ARG-LIST
                       RTN-VOID.

     IF RETURN-CODE NOT = 0 THEN
*>       ここに CBLJEXCEPTION 項目を使った例外処理を記述する
     END-IF.
```

```
*>-----------------------------------------------------------------
*> SampleClass.wait インスタンスメソッドを呼び出す
*>
*> OBJREF.wait(ARG15-1, ARG15-2)
*>
*> 引数：
*>   ARG15-1           PIC S9(18) USAGE COMP
*>   ARG15-2           PIC S9(9) USAGE COMP
*>
*>
*>-----------------------------------------------------------------

     MOVE ???????????? TO ARG15-1-AREA.
     MOVE ???????????? TO ARG15-2-AREA.

     COMPUTE ARGPTR(1) = FUNCTION ADDR(ARG15-1).
     COMPUTE ARGPTR(2) = FUNCTION ADDR(ARG15-2).
     COMPUTE ARGPTR(3) = ZERO.

     CALL 'CBLJINVOKE' USING CBLJENV OBJREF wait ARG-LIST
                      RTN-VOID.

     IF RETURN-CODE NOT = 0 THEN
*>       ここに CBLJEXCEPTION 項目を使った例外処理を記述する
     END-IF.

*>-----------------------------------------------------------------
*> SampleClass.classID スタティックフィールドの値を取得する
*>
*> FIELD1 = SampleClass.classID
*>
*>   FIELD1               USAGE POINTER(java/lang/String)
*>-----------------------------------------------------------------

     CALL 'CBLJGETSTATICFIELD' USING CBLJENV CLASSREF classID
                       FIELD1.

     CALL 'CBLJSTRINGTOX' USING CBLJENV FIELD1-AREA WK-ALNUM
                       WK-ALNUM-LEN.

*>-----------------------------------------------------------------
*> SampleClass.classID スタティックフィールドの値を設定する
*>
*> SampleClass.classID = FIELD1
*>
*>   FIELD1               USAGE POINTER(java/lang/String)
*>-----------------------------------------------------------------

     MOVE '??????????' TO WK-ALNUM.
     CALL 'CBLJXTOSTRING' USING CBLJENV WK-ALNUM WK-ALNUM-LEN
                       FIELD1-AREA.

     CALL 'CBLJSETSTATICFIELD' USING CBLJENV CLASSREF classID
                       FIELD1.
```

```
*>-----------------------------------------------------------------
*> メソッド SampleClass.getExecData でフィールド execData の値を取得する
*>
*> FIELD2 = OBJREF.getExecData()
*>
*> 戻り値：
*>   FIELD2               USAGE POINTER(java/lang/String)
*>-----------------------------------------------------------------

     CALL 'CBLJINVOKE' USING CBLJENV OBJREF getExecData NO-ARG
                       FIELD2.

     IF RETURN-CODE NOT = 0 THEN
*>       ここに CBLJEXCEPTION 項目を使った例外処理を記述する
     END-IF.

*>-----------------------------------------------------------------
*> メソッド SampleClass.setExecData でフィールド execData の値を設定する
*>
*> OBJREF.setExecData(FIELD2)
*>
*> 引数：
*>   FIELD2               USAGE POINTER(java/lang/String)
*>-----------------------------------------------------------------

     MOVE '??????????' TO WK-ALNUM.
     CALL 'CBLJXTOSTRING' USING CBLJENV WK-ALNUM WK-ALNUM-LEN
                       FIELD2-AREA.

     COMPUTE ARGPTR(1) = FUNCTION ADDR(FIELD2).
     COMPUTE ARGPTR(2) = ZERO.

     CALL 'CBLJINVOKE' USING CBLJENV OBJREF setExecData ARG-LIST
                       RTN-VOID.

     IF RETURN-CODE NOT = 0 THEN
*>       ここに CBLJEXCEPTION 項目を使った例外処理を記述する
     END-IF.

*>-----------------------------------------------------------------
*> メソッド SampleClass.getExecType でフィールド execType の値を取得する
*>
*> FIELD3 = OBJREF.getExecType()
*>
*> 戻り値：
*>   FIELD3               PIC S9(9) USAGE COMP(int)
*>-----------------------------------------------------------------

     CALL 'CBLJINVOKE' USING CBLJENV OBJREF getExecType NO-ARG
                       FIELD3.

     IF RETURN-CODE NOT = 0 THEN
*>       ここに CBLJEXCEPTION 項目を使った例外処理を記述する
     END-IF.
```

```
*>------------------------------------------------------------------
*> メソッド SampleClass.setExecType でフィールド execType の値を設定する
*>
*> OBJREF.setExecType(FIELD3)
*>
*> 引数：
*>   FIELD3               PIC S9(9) USAGE COMP(int)
*>------------------------------------------------------------------

     MOVE ???????????? TO FIELD3-AREA.

     COMPUTE ARGPTR(1) = FUNCTION ADDR(FIELD3).
     COMPUTE ARGPTR(2) = ZERO.

     CALL 'CBLJINVOKE' USING CBLJENV OBJREF setExecType ARG-LIST
                       RTN-VOID.

     IF RETURN-CODE NOT = 0 THEN
*>       ここに CBLJEXCEPTION 項目を使った例外処理を記述する
     END-IF.

*>------------------------------------------------------------------
*> SampleClass クラスのインスタンスを解放する
*>
*> 不要になったクラスインスタンスは解放しなければならない
*>------------------------------------------------------------------

     CALL 'CBLJRELEASE' USING CBLJENV OBJREF.

 END-PROC.
*>------------------------------------------------------------------
*> Javaプログラム呼び出し機能の実行環境を終了する
*>------------------------------------------------------------------

     CALL 'CBLJFINALIZE' USING CBLJENV.

     EXIT PROGRAM.

 END PROGRAM           '*SAMPLE*'.
```

## 7.2.3　集団項目データ交換プログラムの生成

プログラム作成支援ツールの-Type オプションに GroupMapper を指定すると，次に示す二つの COBOL ソースファイルを Java クラスごとに作成します。

- 集団項目データ交換用データ定義

  データ取得設定クラスフィールドおよびインスタンスフィールドに対応する COBOL 集団項目を定義した登録集原文です。

- 集団項目データ交換プログラム

  Java のフィールドデータを一括して取得・設定する COBOL プログラムです。

生成された COBOL プログラムを呼び出す UAP を作成して，フィールドの値を COBOL 集団項目に一括して取得・設定するプログラムを完成させます。

```
cbl2kjgen -Type GroupMapper 〔オプション〕 Javaプログラム指定オプション
```

Java プログラム指定オプションは，次の 2 種類の組み合わせで指定します。

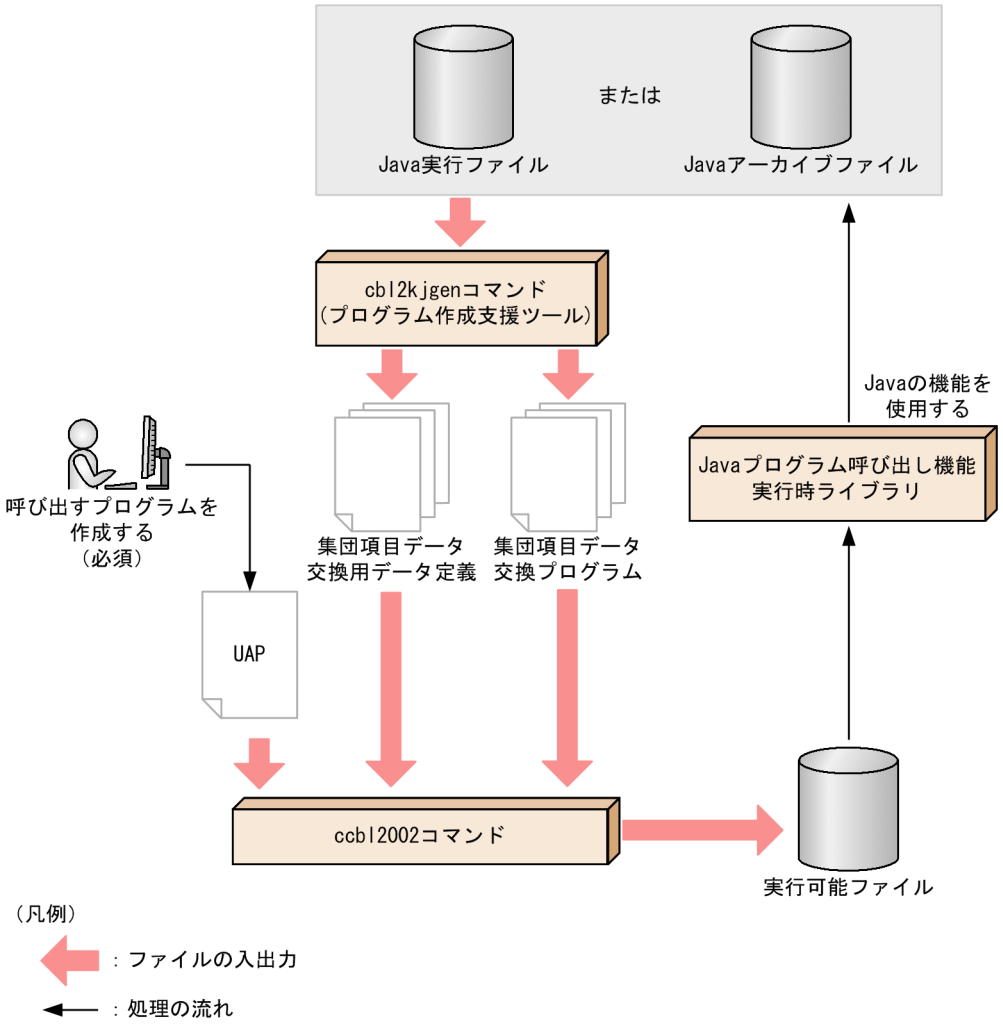
- -Class Java クラス名
- -Jar Java アーカイブファイル名

-Class オプションに指定した Java クラスを Java 実行ファイル（.class）または Java アーカイブファイル（.jar）から検索して解析対象とします。-Jar オプションだけを指定すると，アーカイブファイルの中からプログラム作成支援ツールが解析対象とするすべてのクラスを解析します。

-Class オプションおよび-Jar オプションについては，「7.2　プログラム作成支援ツールの機能」および「7.2.1　プログラム作成支援ツールの機能範囲」の「(5)　使用するファイルとファイルの入出力」を参照してください。

集団項目データ交換プログラムの生成の概要を次の図に示します。

図 7-7　集団項目データ交換プログラムの生成の概要



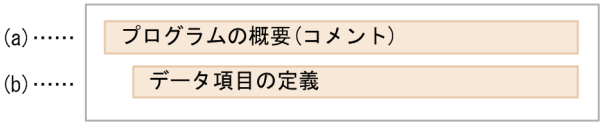

## (1)　集団項目データ交換用データ定義の構造

集団項目データ交換用データ定義は，Java のフィールドに対応する COBOL のデータ項目を集団項目で定義した登録集原文です。

対象とするフィールドについては，「7.2.1　プログラム作成支援ツールの機能範囲」の「(3)　フィールドの設定と取得」を参照してください。

集団項目データ交換用データ定義の構造を次の図に示します。

図 7-8　集団項目データ交換用データ定義の構造




### (a)　プログラムの概要（コメント）

プログラムの概要を先頭のコメントとして記載します。記載する内容を次に示します。

| 項番 | 項目 | 内容（引用符（"）で囲んだ文字列は固定） |
|---|---|---|
| 1 | タイトル | 次の内容を記載します。<br>"COBOL2002 Java プログラム呼び出し機能↓"<br>"集団項目データ交換用データ定義"<br>（凡例）<br>↓：改行 |
| 2 | 利用クラス | 対象のクラス名を記載します。 |
| 3 | cbl2kjgen のバージョン | cbl2kjgen のバージョンを記載します。<br>例："03-02" |
| 4 | オプション | cbl2kjgen に指定されたオプションを記載します。1 行につき 1 オプションを記載します。 |
| 5 | 生成日時 | 集団項目データ交換プログラムの生成日時を記載します。<br>形式は YYYY/mm/dd HH:MM:SS です。 |

### (b)　データ項目の定義

Java のフィールドと対応づける COBOL のデータ項目を定義します。記載内容については，「(3)　データ項目のマッピング」を参照してください。

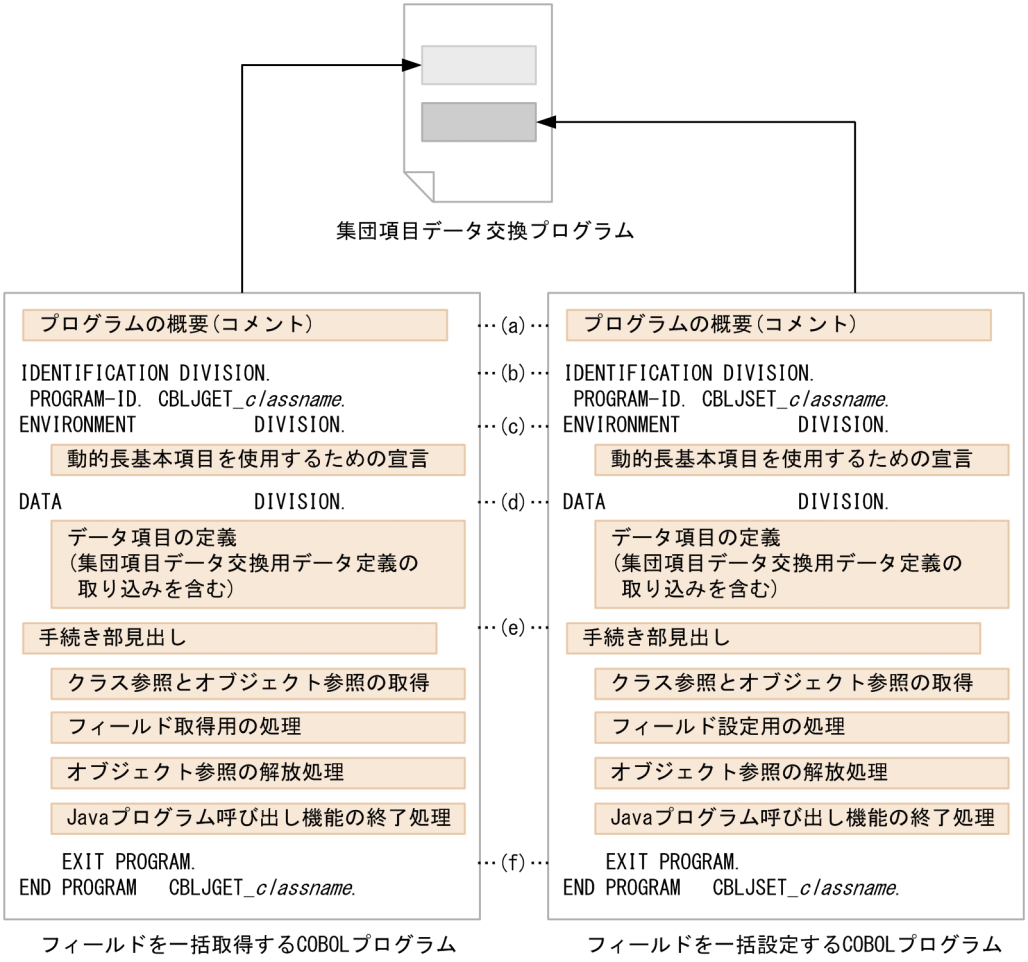
## (2)　集団項目データ交換プログラムの構造

集団項目データ交換プログラムは，Java のフィールドを，Java プログラム呼び出し機能を使用して，一括して取得・設定する COBOL プログラムです。各フィールドの値は集団項目データ交換用データ定義に定義したデータ項目にマッピングします。

一括取得する COBOL プログラムと一括設定する COBOL プログラムを一つの COBOL ソースファイルに生成します。

集団項目データ交換プログラムの構造を次の図に示します。

図 7-9　集団項目データ交換プログラムの構造




## (a)　プログラムの概要（コメント）

プログラムの概要を先頭のコメントとして記載します。記載する内容を次に示します。

| 項番 | 項目 | 内容（引用符（"）で囲んだ文字列は固定） |
|---|---|---|
| 1 | タイトル | 次の内容を記載します。<br>"COBOL2002 Java プログラム呼び出し機能↓"<br>"集団項目データ交換用データ定義"<br>（凡例）<br>↓：改行 |
| 2 | 利用クラス | 対象のクラス名を記載します。 |
| 3 | cbl2kjgen のバージョン | cbl2kjgen のバージョンを記載します。<br>例："03-02" |
| 4 | オプション | cbl2kjgen に指定されたオプションを記載します。1 行につき 1 オプションを記載します。 |
| 5 | 生成日時 | 集団項目データ交換プログラムの生成日時を記載します。<br>形式は YYYY/mm/dd HH:MM:SS です。 |

### (b)　見出し部

各プログラム名を次に示します。

| 用途 | プログラム名※ |
|---|---|
| フィールドを一括取得する。 | CBLJGET_*classname* |
| フィールドを一括設定する。 | CBLJSET_*classname* |

注※

"*classname*"はクラス名または-Class オプションに指定した内部名を示します。クラス名または-Class オプションについては，「7.2.1　プログラム作成支援ツールの機能範囲」の「(2)　データ項目のマッピング（共通)」を参照してください。

### (c)　環境部

動的長基本項目を使用するため，次に示すように構成節を記載します。

```
ENVIRONMENT           DIVISION.
CONFIGURATION         SECTION.
 SPECIAL-NAMES.
  DYNAMIC LENGTH STRUCTURE C-STRING IS C-STATIC-STRUCTURE.
```

### (d)　データ部

データ項目の定義と COPY 文による集団項目データ交換用データ定義の取り込みを行います。作業場所節および連絡節にデータ項目を定義します。ただし，使用しないデータ項目は定義しません。

作業場所節には次に示す順番でデータ項目を定義します。

1. クラス名，フィールド名，およびメソッド名
2. パラメタ
3. 引数リスト
4. String⇔英数字項目の変換領域の長さ
5. 配列オブジェクト⇔可変長繰り返し項目変換領域
6. インスタンス生成フラグ

連絡節には次に示す順番でデータ項目を定義します。

1. CBLJENV 集団項目
2. クラス参照とオブジェクト参照
3. COPY 文による集団項目データ交換用データ定義の取り込み
4. プログラムの戻り値

作業場所節に定義するデータ項目について説明します。データ項目のマッピングについては，「7.2.1　プログラム作成支援ツールの機能範囲」の「(2)　データ項目のマッピング（共通）」および「7.2.2　Java クラス利用サンプルの生成」の「(2)　データ項目のマッピング」を参照してください。

1. クラス名，フィールド名，およびメソッド名

   クラス名，フィールド名，およびメソッド名を設定する英数字の動的長基本項目を定義します。初期値には該当するクラス名，フィールド名またはメソッド名を設定します。

```
*>------------------------------------------------------------------
*> クラス名，フィールド名，メソッド名の定義
*>------------------------------------------------------------------
 01  CLASS-NAME         PIC X DYNAMIC C-STRING VALUE 'SampleClass'.
 01  FLD1-NAME          PIC X DYNAMIC C-STRING VALUE 'classID'.
 01  FLD2-GETTER        PIC X DYNAMIC C-STRING VALUE 'getExecData'.
 01  FLD3-GETTER        PIC X DYNAMIC C-STRING VALUE 'getExecType'.
 01  FLD4-GETTER        PIC X DYNAMIC C-STRING VALUE 'getMemberLi'-
                            'st'.
```

2. パラメタ

   次に示す値を設定するデータ項目を定義します。

   - フィールド※
   - コンストラクタおよびメソッド（フィールドを一括設定するプログラムだけ）に渡す各引数
   - メソッドの戻り値（フィールドを一括取得するプログラムだけ）

   注※

   対象となるフィールドについては，「7.2.1　プログラム作成支援ツールの機能範囲」の「(3)　フィールドの設定と取得」を参照してください。

   パラメタの定義例を次に示します。

```
*>------------------------------------------------------------------
*> パラメタの定義
*>------------------------------------------------------------------
 01  FLD1.
   02  FLD1-TYPE        PIC X(256) VALUE 'Ljava/lang/String;'.
   02  FLD1-AREA        USAGE POINTER.
 01 FLD2.
   02  FLD2-TYPE        PIC X(256) VALUE 'Ljava/lang/String;'.
   02  FLD2-AREA        USAGE POINTER.
 01  FLD3.
   02  FLD3-TYPE        PIC X(1) VALUE 'I'.
   02  FILLER           PIC X(7) VALUE ALL LOW-VALUE.
   02  FLD3-AREA        PIC S9(9) USAGE COMP.
 01  FLD4.
   02  FLD4-TYPE        PIC X(256) VALUE '[[I'.
   02  FLD4-AREA        USAGE POINTER.
```

3. 引数リスト

   Java プログラム呼び出し機能のサービスルーチンに渡す引数リストを定義します。フィールドを一括取得するプログラムは「引数なし」の引数リストだけを作成します。フィールドを一括設定するプログラムは「引数なし」の引数リストと，引数を一つ持つ引数リストを作成します。

フィールドを一括取得するプログラムの場合

```
*>-----------------------------------------------------------------
*> 引数リストの定義(引数なし)
*>-----------------------------------------------------------------
 01  NO-ARG.
   02  FILLER          USAGE POINTER VALUE NULL.
```

フィールドを一括設定するプログラムの場合

```
*>-----------------------------------------------------------------
*> 引数リストの定義
*>-----------------------------------------------------------------
 01  ARG-LIST.
   02  ARGPTR          USAGE POINTER VALUE NULL.
   02  FILLER          USAGE POINTER VALUE NULL.
*> (引数なし)
 01  NO-ARG.
   02  FILLER          USAGE POINTER VALUE NULL.
```

4. String⇔英数字項目の変換領域の長さ

Java プログラム呼び出し機能のサービスルーチンで String⇔英数字項目の変換を行う際，変換結果を受け取るための領域の長さを定義します。

```
*>-----------------------------------------------------------------
*> String オブジェクト <=> COBOLデータ項目変換用領域（長さ）
*>-----------------------------------------------------------------
*> FLD1-LEN: classID フィールドに対応する長さ
 01  FLD1-LEN          PIC S9(9) USAGE COMP VALUE 256.
*> FLD2-LEN: execData フィールドに対応する長さ
 01  FLD2-LEN          PIC S9(9) USAGE COMP VALUE 256.
```

5. 配列オブジェクト⇔可変長繰り返し項目変換領域

Java プログラム呼び出し機能のサービスルーチンで配列オブジェクト⇔可変長繰り返し項目を変換するとき，変換結果を受け取るための領域として使用します。

```
*>-----------------------------------------------------------------
*> 配列オブジェクト <=> 可変長繰り返し項目変換領域
*>-----------------------------------------------------------------
 01  TEMP-ARRAY-ADDR   USAGE POINTER.
 01  ARRAY-INDEX-1     PIC S9(9) USAGE COMP.
 01  ARRAY-INDEX-2     PIC S9(9) USAGE COMP.
 01  ARRAY-ADDR-2      USAGE POINTER.
 01  ARRAY-INDEX-J-1   PIC S9(9) USAGE COMP.
 01  TEMP-MAP-4-LEN    PIC S9(9) USAGE COMP VALUE 256.
 01  TEMP-MAP-4        ADDRESSED BY TEMP-MAP-4-P.
   02  TEMP-MAP-4-ELEM PIC S9(9) USAGE COMP OCCURS 256 TIMES
                       DEPENDING ON TEMP-MAP-4-LEN.
```

6. インスタンス生成フラグ

入力ファイルのクラスのインスタンスを生成した場合，1 を設定します。

```
*>-----------------------------------------------------------------
*> ワーク領域
*>-----------------------------------------------------------------
*> FLAG-NEW: このプログラムでインスタンスを生成したか。
 01  FLAG-NEW          PIC S9(9) USAGE COMP VALUE 0.
```

連絡節に定義するデータ項目について説明します。

1. CBLJENV 集団項目

   CBLJENV 集団項目を定義します。

```
*>-------------------------------------------------------------
*> CBLJENV集団項目
*>-------------------------------------------------------------
 01  CBLJENV.
   02  CBLJENVCORE     USAGE POINTER.
   02  CBLJEXCEPTION   USAGE POINTER.
   02  CBLJFLAGS       PIC 1(32) USAGE BIT.
   02  CBLJSTRMAXLEN   PIC S9(9) USAGE COMP.
   02  CBLJVMOPTIONS.
     03  CBLJOPTCOUNT  PIC S9(9) USAGE COMP.
     03  CBLJOPTION-1  PIC X.
```

2. クラス参照とオブジェクト参照

   クラス参照とオブジェクト参照を定義します。クラス参照とオブジェクト参照は 1 クラスにつき一つずつ定義します。

```
*>-------------------------------------------------------------
*> クラス参照とオブジェクト参照
*>-------------------------------------------------------------
 01  CLASSREF          USAGE POINTER.
 01  OBJREF            USAGE POINTER.
```

3. COPY 文による集団項目データ交換用データ定義の取り込み

   集団項目データ交換用データ定義を COPY 文で取り込みます。

```
*>-------------------------------------------------------------
*> SampleClass クラスに対応するCOBOL集団項目
*>-------------------------------------------------------------
  COPY 'SampleClass_Map_COPY.cbl'.
```

4. プログラムの戻り値

   各プログラムの戻り値として，Java プログラム呼び出し機能のサービスルーチンでエラーが発生した際に，サービスルーチンの戻り値を設定します。

```
*>-------------------------------------------------------------
*> 戻り値領域
*>-------------------------------------------------------------
 01  CBLJ-RETURN-CODE  PIC S9(9) USAGE COMP.
```

## (e) 手続き部

手続き部の内容について説明します。

各プログラムに次に示すような手続き部見出しを定義します。

```
PROCEDURE           DIVISION USING CBLJENV
                                   CLASSREF
                                   OBJREF
                                   SampleClass
                    RETURNING CBLJ-RETURN-CODE.
```

クラス参照とオブジェクト参照の取得

クラスおよびそのインスタンス（オブジェクト参照）を取得する処理です。

クラス参照とオブジェクト参照の取得の例を次に示します。

```
*> クラス参照がNULLのとき，クラス参照を取得する
     IF CLASSREF = NULL THEN
         CALL 'CBLJGETCLASS' USING CBLJENV CLASS-NAME CLASSREF
     END-IF.

*> オブジェクト参照がNULLのとき，インスタンスを生成する
     MOVE 0 TO FLAG-NEW.
     IF OBJREF = NULL THEN
         CALL 'CBLJNEW' USING CBLJENV CLASSREF NO-ARG OBJREF

         IF RETURN-CODE NOT = 0 THEN
             MOVE RETURN-CODE TO CBLJ-RETURN-CODE
             GO TO END-PROC
         END-IF
         MOVE 1 TO FLAG-NEW
     END-IF.
```

クラス参照の取得

オブジェクト参照の取得

サービスルーチンでエラーが発生した場合，戻り値を設定して終了

インスタンスが生成できたことを記憶する

フィールド値の取得・設定処理

フィールド値の取得と設定を行う処理です。フィールドに直接アクセスするか，getter メソッドと setter メソッドを呼びます。getter メソッドと setter メソッドについては，「7.2.1　プログラム作成支援ツールの機能範囲」の「(3)　フィールドの設定と取得」を参照してください。

フィールド値の設定処理

フィールド値の設定処理の例を次に示します。

```
*> フィールド値を設定する
*> ** classID
     CALL 'CBLJXTOSTRING' USING CBLJENV classID FLD1-LEN
                       FLD1-AREA.

     CALL 'CBLJSETSTATICFIELD' USING CBLJENV CLASSREF FLD1-NAME
                       FLD1.

     CALL 'CBLJRELEASE' USING CBLJENV FLD1-AREA.

*> ** execData
     CALL 'CBLJXTOSTRING' USING CBLJENV execData FLD2-LEN
                       FLD2-AREA.

     COMPUTE ARGPTR = FUNCTION ADDR(FLD2).
     CALL 'CBLJINVOKE' USING CBLJENV OBJREF FLD2-SETTER ARG-LIST
                       RTN-VOID.

     IF RETURN-CODE NOT = 0 THEN
       MOVE RETURN-CODE TO CBLJ-RETURN-CODE
       GO TO END-PROC
     END-IF.

     CALL 'CBLJRELEASE' USING CBLJENV FLD2-AREA.

*> ** execType
     COMPUTE FLD3-AREA = execType.

     COMPUTE ARGPTR = FUNCTION ADDR(FLD3).
     CALL 'CBLJINVOKE' USING CBLJENV OBJREF FLD3-SETTER ARG-LIST
                       RTN-VOID.

     IF RETURN-CODE NOT = 0 THEN
       MOVE RETURN-CODE TO CBLJ-RETURN-CODE
       GO TO END-PROC
     END-IF.
```

フィールドに直接アクセス

setterメソッドを呼ぶ

```
*> ** memberList
    CALL 'CBLJNEWARRAY' USING CBLJENV FLD4-TYPE
                              memberList-1-LEN
                              FLD4-AREA.

    PERFORM VARYING ARRAY-INDEX-1 FROM 1 BY 1
     UNTIL ARRAY-INDEX-1 > memberList-1-LEN
      COMPUTE ARRAY-INDEX-J-1 = ARRAY-INDEX-1 - 1
      CALL 'CBLJNEWARRAY' USING CBLJENV FLD4-TYPE(2:)
                                memberList-2-LEN
                                ARRAY-ADDR-2
      MOVE memberList-2-LEN TO TEMP-MAP-4-LEN
      CALL 'CBLJGETARRAYADDR' USING CBLJENV ARRAY-ADDR-2
                                TEMP-ARRAY-ADDR
      COMPUTE TEMP-MAP-4-P = TEMP-ARRAY-ADDR

      PERFORM VARYING ARRAY-INDEX-2 FROM 1 BY 1
       UNTIL ARRAY-INDEX-2 > memberList-2-LEN
        COMPUTE TEMP-MAP-4-ELEM(ARRAY-INDEX-2) =
                memberList-ELEM(ARRAY-INDEX-1, ARRAY-INDEX-2)
      END-PERFORM

      CALL 'CBLJRELEASEARRAY' USING CBLJENV ARRAY-ADDR-2
                                TEMP-ARRAY-ADDR
      CALL 'CBLJSETOBJARRAY' USING CBLJENV FLD4-AREA
                                ARRAY-INDEX-J-1 ARRAY-ADDR-2
    END-PERFORM.

    COMPUTE ARGPTR = FUNCTION ADDR(FLD4).
    CALL 'CBLJINVOKE' USING CBLJENV OBJREF FLD4-SETTER ARG-LIST
                            RTN-VOID.
    IF RETURN-CODE NOT = 0 THEN
      MOVE RETURN-CODE TO CBLJ-RETURN-CODE
      GO TO END-PROC
    END-IF.

    CALL 'CBLJRELEASE' USING CBLJENV FLD4-AREA.
```

配列オブジェクトに値を設定する。配列オブジェクトはフィールドに直接アクセスまたはsetterメソッドで設定する（この例ではsetterメソッドで設定）。

集団項目データ交換プログラムで配列オブジェクトに値を設定する場合，次に示すように配列の長さを表すデータ項目（上記の例では，memberList-1-LEN および memberList-2-LEN）に各次元の配列長を事前に設定する必要があります。

```
     :
  WORKING-STORAGE SECTION.

*> データ定義
      COPY 'SampleClass_Map_COPY.cbl'.
     :
  PROCEDURE DIVISION.

*> 配列memberListの各次元の配列長を設定
*> (この例ではmemberList[10][20]という長さに設定)
      MOVE 10 TO memberList-1-LEN.
      MOVE 20 TO memberList-2-LEN.
     :
*> フィールドデータの一括設定
      CALL 'CBLJSET_SampleClass' USING CBLJENV
                                       CLASSREF
                                       OBJECTREF
                                       SampleClass
                                 RETURNING RETURN-VAL.
     :
```

フィールド値の取得処理

フィールド値の取得処理の例を次に示します。

```
*> フィールド値を取得する
*> ** classID
     CALL 'CBLJGETSTATICFIELD' USING CBLJENV CLASSREF FLD1-NAME
                       FLD1.

     CALL 'CBLJSTRINGTOX' USING CBLJENV FLD1-AREA classID
                       FLD1-LEN.

     CALL 'CBLJRELEASE' USING CBLJENV FLD1-AREA.

*> ** execData
     CALL 'CBLJINVOKE' USING CBLJENV OBJREF FLD2-GETTER NO-ARG
                       FLD2.

     IF RETURN-CODE NOT = 0 THEN
       MOVE RETURN-CODE TO CBLJ-RETURN-CODE
       GO TO END-PROC
     END-IF.

     CALL 'CBLJSTRINGTOX' USING CBLJENV FLD2-AREA execData
                       FLD2-LEN.

     CALL 'CBLJRELEASE' USING CBLJENV FLD2-AREA.

*> ** execType
     CALL 'CBLJINVOKE' USING CBLJENV OBJREF FLD3-GETTER NO-ARG
                       FLD3.

     IF RETURN-CODE NOT = 0 THEN
       MOVE RETURN-CODE TO CBLJ-RETURN-CODE
       GO TO END-PROC
     END-IF.

     COMPUTE execType = FLD3-AREA.
```

フィールドに直接アクセス

getterメソッドを呼ぶ

```
*> ** memberList
     CALL 'CBLJINVOKE' USING CBLJENV OBJREF FLD4-GETTER NO-ARG
                       FLD4.

     IF RETURN-CODE NOT = 0 THEN
       MOVE RETURN-CODE TO CBLJ-RETURN-CODE
       GO TO END-PROC
     END-IF.

     CALL 'CBLJARRAYLENGTH' USING CBLJENV FLD4-AREA
                       memberList-1-LEN.
     PERFORM VARYING ARRAY-INDEX-1 FROM 1 BY 1 UNTIL ARRAY-INDEX-1
                       > memberList-1-LEN

      COMPUTE ARRAY-INDEX-J-1 = ARRAY-INDEX-1 - 1
      CALL 'CBLJGETOBJARRAY' USING CBLJENV FLD4-AREA
                       ARRAY-INDEX-J-1 ARRAY-ADDR-2
      CALL 'CBLJARRAYLENGTH' USING CBLJENV ARRAY-ADDR-2
                       memberList-2-LEN
      MOVE memberList-2-LEN TO TEMP-MAP-4-LEN
      CALL 'CBLJGETARRAYADDR' USING CBLJENV ARRAY-ADDR-2
                       TEMP-ARRAY-ADDR
      COMPUTE TEMP-MAP-4-P = TEMP-ARRAY-ADDR
      PERFORM VARYING ARRAY-INDEX-2 FROM 1 BY 1 UNTIL
                        ARRAY-INDEX-2 > memberList-2-LEN
       COMPUTE memberList-ELEM ( ARRAY-INDEX-1 , ARRAY-INDEX-2 ) =
                       TEMP-MAP-4-ELEM ( ARRAY-INDEX-2 )
      END-PERFORM
      CALL 'CBLJRELEASEARRAY' USING CBLJENV ARRAY-ADDR-2
                       TEMP-ARRAY-ADDR
     END-PERFORM.
```

配列オブジェクトから値を取得する。配列オブジェクトはフィールドに直接アクセスまたはgetterメソッドで取得する（この例ではgetterメソッドで取得）。

オブジェクト参照の解放処理

フィールドを一括取得するプログラムでは，インスタンス生成フラグが立っている場合，クラス参照とオブジェクト参照の取得で取得したオブジェクト参照（インスタンス）を解放する処理です。手続き END-PROC に含みます。

フィールドを一括設定するプログラムでは，この解放処理を生成しません。手続き名だけを生成します。このプログラムから戻ったあとに取得したフィールドに値を設定したオブジェクト参照を使用できます。

オブジェクト参照の解放処理の例を次に示します。

```
END-PROC.
*> 生成したインスタンスを解放する
     IF FLAG-NEW = 1 THEN
        CALL 'CBLJRELEASE'  USING CBLJENV OBJREF
     END-IF.
```

### (f)　プログラムの終了

プログラムを終了します。

```
    EXIT PROGRAM.

END PROGRAM 'CBLJSET_SampleClass'.
```

## (3)　データ項目のマッピング

集団項目データ交換用データ定義および集団項目データ交換プログラムに，各データ項目を定義する際に使用するルールを示します。

### (a)　集団項目データ交換用データ定義

集団項目データ交換用データ定義に，各データ項目を定義する際に使用するルールを示します。

集団項目データ交換用データ定義

集団項目データ交換用データ定義のデータ項目を次に示します。

| 用途 | データ項目※ |
|---|---|
| Java のフィールドを COBOL の集団項目にマッピングする。 | 01 *classname*.<br>  02 *fieldname* *TYPE-DEF*.<br>  :<br>（これ以降は配列オブジェクトをマッピングするための定義）<br>  02 *fieldname*.<br>  03 *fieldname*-1-LEN  PIC S9(9) USAGE COMP.<br>  :<br>  03 *fieldname-m*-LEN  PIC S9(9) USAGE COMP.<br>  03 *fieldname*-ROOT.<br>  04 *fieldname*-1 OCCURS *ARRAYMAXLEN* TIMES. |

| 用途 | データ項目※ |
|---|---|
| | :<br>10 *fieldname-m* OCCURS *ARRAYMAXLEN* TIMES.<br>11 *fieldname*-ELEM *TYPE-DEF*. |

注※

- "*m*"は配列オブジェクトの次元数を示します。
- "*fieldname*-*m*-LEN"は m 次元配列の長さを示します。配列要素を設定する場合は，ユーザがプログラムを呼び出す前に配列の長さを設定する必要があります。
- "*classname*"はクラス名または-Class オプションに指定した内部名を示します。クラス名または-Class オプションについては，「7.2.1　プログラム作成支援ツールの機能範囲」の「(2)　データ項目のマッピング（共通）」を参照してください。
- "*fieldname*"はフィールド名を示します。フィールド名については，「7.2.1　プログラム作成支援ツールの機能範囲」の「(2)　データ項目のマッピング（共通）」を参照してください。
- "*TYPE-DEF*"は Java の型に対応する COBOL の型を示します。ただし，Java のデータ型が String 型の場合は英数字項目を定義します（既定のけた数は 256 です。この値はプログラム作成支援ツールの-StrMaxLen オプションで変更できます）。Java の型に対応する COBOL の型については，「6.1.1　サービスルーチンで使用する引数」の「(3)　パラメタ型集団項目」を参照してください。
- "*ARRAYMAXLEN*"の既定値は 256 です。この値はプログラム作成支援ツールの-MaxArrayLength オプションで変更できます。

### (b)　集団項目データ交換プログラム

集団項目データ交換プログラムに，各データ項目を定義する際に使用するルールを示します。

JNI に関するデータ項目

JNI に関するデータ項目を次に示します。

| 用途 | データ項目 |
|---|---|
| Java VM に渡すオプションを 1 個受け取る CBLJENV 集団項目を定義する。 | 01 CBLJENV.<br>02 CBLJENVCORE USAGE POINTER.<br>02 CBLJEXCEPTION USAGE POINTER.<br>02 CBLJFLAGS PIC 1(32) USAGE BIT.<br>02 CBLJSTRMAXLEN PIC S9(9) USAGE COMP.<br>02 CBLJVMOPTIONS.<br>03 CBLJOPTCOUNT PIC S9(9) USAGE COMP.<br>03 CBLJOPTION-1 PIC X. |

クラスやオブジェクトに関するデータ項目

クラスやオブジェクトに関するデータ項目を次に示します。

| 項番 | 用途 | データ項目※ |
|---|---|---|
| 1 | クラス名を設定する。 | 01 CLASS-NAME PIC X DYNAMIC C-STRING VALUE '*classnameOrg*'. |
| 2 | クラス参照のアドレスを受け取る。 | 01 CLASSREF USAGE POINTER. |
| 3 | オブジェクト参照のアドレスを受け取る。 | 01 OBJREF USAGE POINTER. |
| 4 | インスタンスを生成したことを記憶させる。 | 01 FLAG-NEW PIC S9(9) USAGE COMP VALUE 0. |

注※

"*classnameOrg*"はクラス名を示します。クラス名に含まれるピリオドはスラントに置換します。

メソッド（クラスメソッド，インスタンスメソッド）に関するデータ項目

メソッド（クラスメソッド，インスタンスメソッド）に関するデータ項目を次に示します。

| 項番 | 用途 | データ項目※ |
|---|---|---|
| 1 | *n* 番目のフィールドの getter メソッド名を設定する。 | 01 FLD*n*-GETTER PIC X DYNAMIC C-STRING VALUE '*methodname*'. |
| 2 | *n* 番目のフィールドの setter メソッド名を設定する。 | 01 FLD*n*-SETTER PIC X DYNAMIC C-STRING VALUE '*methodname*'. |
| 3 | *n* 番目のフィールドの setter メソッドの第 1 引数を設定する。<br>または，*n* 番目のフィールドの getter メソッドの戻り値を設定する。 | （基本型の場合）<br>01 FLD*n*.<br>02 FLD*n*-TYPE PIC X(1) VALUE '*SIGNATURE*'.<br>02 FILLER PIC X(7) VALUE ALL LOW-VALUE.<br>02 FLD*n*-AREA *TYPE-DEF*.<br>（基本型以外の場合）<br>01 FLD*n*.<br>02 FLD*n*-TYPE PIC X(*STRMAXLEN*) VALUE '*SIGNATURE*'.<br>02 FLD*n*-AREA *TYPE-DEF*. |
| 4 | *n* 番目のフィールドが String の場合，フィールドの長さを設定する。 | 01 FLD*n*-LEN PIC S9(9) USAGE COMP VALUE *STRMAXLEN*. |

注※

- "*n*"は集団項目データ交換用データ定義中の 02 レベルのデータ項目を定義する順番と一致します。
- "*methodname*"はメソッド名を示します。
- "*SIGNATURE*"は Java の型を表す文字列を示します。ただし，引数の型が Class インスタンスの場合，型には java/lang/Class ではなく，実体のクラス名となるように修正してください。指定した Java の型と実体のクラス名が一致しない場合，実行時エラーとなることがあります。Java の型を表す文字列については，「6.1.1　サービスルーチンで使用する引数」の「(3)　パラメタ型集団項目」を参照してください。

- "*TYPE-DEF*"は Java の型に対応する COBOL の型を示します。Java の型に対応する COBOL の型については，「6.1.1　サービスルーチンで使用する引数」の「(3)　パラメタ型集団項目」を参照してください。
- "*STRMAXLEN*"の既定値は 256 です。この値はプログラム作成支援ツールの-StrMaxLen オプションで変更できます。

フィールド（クラスフィールド，インスタンスフィールド）に関するデータ項目

フィールド（クラスフィールド，インスタンスフィールド）に関するデータ項目を次に示します。

| 項番 | 用途 | データ項目※ |
|---|---|---|
| 1 | *n* 番目のフィールド名を設定する。 | 01 FLD*n*-NAME PIC X DYNAMIC C-STRING VALUE '*fieldname*'. |
| 2 | 集団項目データ交換用データ定義を取り込む。 | COPY '*copyfilename*'. |
| 3 | *n* 番目のフィールドが配列オブジェクトの場合，その最下層要素と COBOL の繰り返し項目をマッピングするための集団項目およびサイズを定義する。 | 01　TEMP-MAP-*n*-LEN　PIC S9(9) USAGE COMP VALUE *ARRAYMAXLEN*.<br>01　TEMP-MAP-*n* ADDRESSED BY TEMP-MAP-*n*-P.<br>02　TEMP-MAP-*n*-ELEM *TYPE-DEF*　OCCURS *ARRAYMAXLEN* TIMES<br>DEPENDING ON TEMP-MAP-*n*-LEN. |

注※

- "*n*"は集団項目データ交換用データ定義中の 02 レベルのデータ項目を定義する順番と一致します。
- "*fieldname*"はフィールド名を示します。
- "*copyfilename*"は集団項目データ交換用データ定義のファイル名を示します。集団項目データ交換用データ定義のファイル名については，「7.2.1　プログラム作成支援ツールの機能範囲」の「(5)　使用するファイルとファイルの入出力」の「(c)　ファイルの出力」を参照してください。
- "*TYPE-DEF*"は Java の型に対応する COBOL の型を示します。Java の型に対応する COBOL の型については，「6.1.1　サービスルーチンで使用する引数」の「(3)　パラメタ型集団項目」を参照してください。
- "*ARRAYMAXLEN*"の既定値は 256 です。この値はプログラム作成支援ツールの-MaxArrayLength オプションで変更できます。

その他

その他のデータ項目を次に示します。

| 用途 | データ項目 |
|---|---|
| プログラムの戻り値を設定する。 | 01　CBLJ-RETURN-CODE　PIC S9(9) USAGE COMP. |

## (4)　生成後の使用方法

集団項目データ交換プログラムを呼び出す COBOL プログラムを作成します。

### (a)　作成手順

1. 集団項目データ交換プログラムの呼び出し元プログラムの作業場所節に集団項目データ交換用データ定義を読み込む COPY 文を記述します。

```
   :
WORKING-STORAGE SECTION.
 COPY '集団項目データ交換用データ定義のファイル名'.
   :
```

集団項目データ交換用データ定義のファイル名については，「7.2.1　プログラム作成支援ツールの機能範囲」の「(5)　使用するファイルとファイルの入出力」を参照してください。

2. 集団項目データ交換プログラムを呼び出すコードを記述します。

集団項目データ交換プログラムを呼び出す前に，CBLJINITIALIZE サービスルーチンを呼び出して，Java プログラム呼び出し機能の実行環境を初期化してください。

集団項目データ交換プログラムの形式については，「(b)　集団項目データ交換プログラムの形式」を参照してください。

3. フィールドを一括取得する場合，集団項目データ交換プログラムを呼び出したあとで，集団項目データ交換用データ定義の各データ項目を参照するプログラムを記述します。

**注意事項**

集団項目データ交換プログラムの戻り値が 0 以外の場合，正しい値を取得できないことがあります。

4. フィールドを一括設定する場合，集団項目データ交換プログラムを呼び出す前に，集団項目データ交換用データ定義の各データ項目に値を設定するプログラムを記述します。

**注意事項**

- 配列のフィールドを設定する場合は配列長を設定する必要があります。詳細については，「7.2.3　集団項目データ交換プログラムの生成」の「(2)　集団項目データ交換プログラムの構造」を参照してください。
- 静的フィールドとインスタンスフィールドを一括して処理します。静的フィールドの値を変更する場合は，同じプロセスのほかのプログラムに影響を与えることがあります。

集団項目データ交換用データ定義および集団項目データ交換プログラムのデータ項目名やデータ項目の長さは，用途に従って変更できます。

```
          :
 WORKING-STORAGE SECTION.
*>------------------------------------------------------------------
*> データ定義 (mylib.sample.SampleClass)
*>------------------------------------------------------------------
 COPY 'mylib.sample.SampleClass_Map_COPY.cbl'.
```

```
 PROCEDURE DIVISION.

*> Javaプログラム呼び出し機能の実行環境を初期化
    CALL 'CBLJINITIALIZE' USING CBLJENV.

         :

*> クラス参照の取得
    CALL 'CBLJNEW' USING CBLJENV
                         CLASSREF
                         NO-ARG
                         OBJECTREF.

*> フィールドデータの一括取得（mylib.sample.SampleClass）
    CALL 'CBLJGET_SampleClass' USING CBLJENV
                                     CLASSREF
                                     OBJECTREF
                                     SampleClass
                                RETURNING RETURN-VAL.
          :
```

## (b)　集団項目データ交換プログラムの形式

フィールドを一括取得するプログラムとフィールドを一括設定するプログラムの形式を説明します。

フィールドを一括取得するプログラム

形式

```
CALL 'CBLJGET_classname' USING 引数1 引数2 引数3 引数4
                          〔 RETURNING 返却項目 〕.
```

**注意事項**

"*classname*"は，クラス名または-Class オプションに指定した内部名を示します。詳細については，「7.2.1　プログラム作成支援ツールの機能範囲」の「(2)　データ項目のマッピング(共通)」を参照してください。

引数

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。
- 引数 2 には，Java クラス参照を持つポインタ項目を指定します。
- 引数 3 には，Java オブジェクト参照を持つポインタ項目を指定します。
- 引数 4 には，集団項目データ交換用データ定義の 01 レベルの集団項目名を指定します。集団項目データ交換用データ定義の 01 レベルの集団項目名については，「7.2.3　集団項目データ交換プログラムの生成」の「(3)　データ項目のマッピング」を参照してください。
- 返却項目には，戻り値を受け取る 4 バイト 2 進形式の数字項目を指定します。

#### 戻り値

RETURNING に指定したデータ項目に戻り値を返します。

0：正常終了した。

0 以外：Java オブジェクト参照の取得，または getter メソッドの呼び出しで例外が発生した。

#### 注意事項

- 引数 2 および引数 3 には，NULL を設定できます。NULL を設定した場合，集団項目データ交換プログラムが Java クラス参照および Java オブジェクト参照を取得して，値を設定します。ただし，集団項目データ交換プログラムが取得した Java オブジェクト参照は一括取得するプログラムの終了時に解放するため，呼び出し元では使用できません。

  なお，対象の Java クラスに引数がない public コンストラクタが定義されていない場合，呼び出し元プログラムまたは集団項目データ交換プログラムを修正する必要があります。詳細については，「7.4.1　使用上の注意事項」の「(5)　引数があるコンストラクタの使用について」を参照してください。

- 一括取得するプログラムは，引数 4 に指定した集団項目を初期化しません。また，取得できないフィールドの値は更新しません。
- 戻り値が 0 以外の場合，CBLJENV 集団項目の CBLJEXCEPTION 項目に例外オブジェクトが設定されます。

フィールドを一括設定するプログラム

#### 形式

```
CALL 'CBLJSET classname' USING 引数1 引数2 引数3 引数4
                          〔 RETURNING 返却項目 〕.
```

**注意事項**

"*classname*"は，クラス名または-Class オプションに指定した内部名を示します。詳細については，「7.2.1　プログラム作成支援ツールの機能範囲」の「(2)　データ項目のマッピング(共通)」を参照してください。

#### 引数

- 引数 1 には，CBLJENV 集団項目を指定します。CBLJENV 集団項目については，「6.1.1　サービスルーチンで使用する引数」の「(1)　CBLJENV 集団項目」を参照してください。
- 引数 2 には，Java クラス参照を持つポインタ項目を指定します。
- 引数 3 には，Java オブジェクト参照を持つポインタ項目を指定します。
- 引数 4 には，集団項目データ交換用データ定義の 01 レベルの集団項目名を指定します。集団項目データ交換用データ定義の 01 レベルの集団項目名については，「7.2.3　集団項目データ交換プログラムの生成」の「(3)　データ項目のマッピング」を参照してください。
- 返却項目には，戻り値を受け取る 4 バイト 2 進形式の数字項目を指定します。

戻り値

RETURNING に指定したデータ項目に戻り値を返します。

0：正常終了した。

0 以外：Java オブジェクト参照の取得，または setter メソッドの呼び出しで例外が発生した。

注意事項

- 引数 2 および引数 3 には，NULL を設定できます。NULL を設定した場合，集団項目データ交換プログラムが Java クラス参照および Java オブジェクト参照を取得して，値を設定します。集団項目データ交換プログラムが取得した Java オブジェクト参照は一括設定するプログラムの終了時に解放されません。呼び出し元で不要になったときに CBLJRELEASE サービスルーチンまたは CBLJSETNULL サービスルーチンを呼び出して解放してください。

  なお，対象の Java クラスに引数がない public コンストラクタが定義されていない場合，呼び出し元プログラムまたは集団項目データ交換プログラムを修正する必要があります。詳細については，「7.4.1　使用上の注意事項」の「(5)　引数があるコンストラクタの使用について」を参照してください。
- 引数 4 には，一括設定するプログラムで設定するすべてのフィールドに対応するデータ項目に，呼び出し元で値を設定してください。値を更新しないことが明らかなフィールドは，一括設定するプログラムで，該当するフィールドの設定処理を削除してください。
- 戻り値が 0 以外の場合，CBLJENV 集団項目の CBLJEXCEPTION 項目に例外オブジェクトが設定されます。

### (c)　集団項目データ交換プログラムの例

集団項目データ交換プログラムの例を次に示します。

コマンド

```
cbl2kjgen -Type GroupMapper -StrMaxLen 256 -Class mylib.sample.SampleClass
```

mylib¥sample¥SampleClass.java の内容

```
package mylib.sample;
public class SampleClass {
   /** クラスID */
   public static String classID = "SampleClass";

   /** 実行種別 */
   private       int    execType;
   /** 実行データ */
   private       String execData;

   /** constructor */
   public SampleClass() {
      execType = 0;
      execData = "";
   }

   /**
    * executeProcess
    * @return 実行結果
    */
   public boolean executeProcess(int flag) {
            :
      return true;
   }
   /**
    * 実行種別を取得する
    * @return 実行種別
    */
   public int getExecType() {
      return execType;
   }
   /**
    * 実行種別を設定する
    */
   public void setExecType(int type) {
      execType = type;
   }
   /**
    * 実行データを取得する
    * @return 実行データ
    */
   public String getExecData() {
      return execData;
   }
   /**
    * 実行データを設定する
    */
   public void setExecData(String data) {
      execData = data;
   }
}
```

mylib.sample.SampleClass_Map_COPY.cbl の内容（集団項目データ交換用データ定義）

```
      *>------------------------------------------------------------------
      *> COBOL2002 Javaプログラム呼び出し機能
      *> 集団項目データ交換用データ定義
      *>
      *> 利用クラス: mylib.sample.SampleClass
      *> cbl2kjgen バージョン: 03-05
      *>           オプション: -Type GroupMapper
      *>                       -Class mylib.sample.SampleClass
      *>                       -StrMaxLen 256
      *>
      *> 生成日時: 2016/01/29 13:00:00
      *>------------------------------------------------------------------
       01  SampleClass.
         02  classID         PIC X(256).
         02  execData        PIC X(256).
         02  execType        PIC S9(9) USAGE COMP.
```

mylib.sample.SampleClass_Map.cbl の内容（集団項目データ交換プログラム）

```
*>-----------------------------------------------------------------
*> COBOL2002 Javaプログラム呼び出し機能
*> 集団項目データ交換プログラム
*>
*> 利用クラス: mylib.sample.SampleClass
*> cbl2kjgen バージョン: 03-05
*>           オプション: -Type GroupMapper
*>                       -Class mylib.sample.SampleClass
*>                       -StrMaxLen 256
*>
*> 生成日時: 2016/01/29 13:00:00
*>-----------------------------------------------------------------
*>-----------------------------------------------------------------
*> CBLJGET_SampleClass:
*>   SampleClass クラスのフィールド値を取得する
*>-----------------------------------------------------------------
 IDENTIFICATION        DIVISION.
  PROGRAM-ID.          'CBLJGET_SampleClass'.
 ENVIRONMENT           DIVISION.
 CONFIGURATION         SECTION.
  SPECIAL-NAMES.
   DYNAMIC LENGTH STRUCTURE C-STRING IS C-STATIC-STRUCTURE.
 DATA                  DIVISION.
 WORKING-STORAGE       SECTION.
*>-----------------------------------------------------------------
*> クラス名, フィールド名, メソッド名の定義
*>-----------------------------------------------------------------
 01  CLASS-NAME        PIC X DYNAMIC C-STRING VALUE 'mylib/sampl'-
                       'e/SampleClass'.
 01  FLD1-NAME         PIC X DYNAMIC C-STRING VALUE 'classID'.
 01  FLD2-GETTER       PIC X DYNAMIC C-STRING VALUE 'getExecData'.
 01  FLD3-GETTER       PIC X DYNAMIC C-STRING VALUE 'getExecType'.
*>-----------------------------------------------------------------
*> パラメタの定義
*>-----------------------------------------------------------------
 01  FLD1.
   02  FLD1-TYPE       PIC X(256) VALUE 'Ljava/lang/String;'.
   02  FLD1-AREA       USAGE POINTER.
 01  FLD2.
   02  FLD2-TYPE       PIC X(256) VALUE 'Ljava/lang/String;'.
   02  FLD2-AREA       USAGE POINTER.
 01  FLD3.
   02  FLD3-TYPE       PIC X(1) VALUE 'I'.
   02  FILLER          PIC X(7) VALUE ALL LOW-VALUE.
   02  FLD3-AREA       PIC S9(9) USAGE COMP.
*>-----------------------------------------------------------------
*> 引数リストの定義(引数なし)
*>-----------------------------------------------------------------
 01  NO-ARG.
   02  FILLER          USAGE POINTER VALUE NULL.
*>-----------------------------------------------------------------
*> String オブジェクト <=> COBOLデータ項目変換用領域 (長さ)
*>-----------------------------------------------------------------
*> FLD1-LEN: classID フィールドに対応する長さ
 01  FLD1-LEN          PIC S9(9) USAGE COMP VALUE 256.
*> FLD2-LEN: execData フィールドに対応する長さ
 01  FLD2-LEN          PIC S9(9) USAGE COMP VALUE 256.
```

```
*>-----------------------------------------------------------------
*> ワーク領域
*>-----------------------------------------------------------------
*> FLAG-NEW: このプログラムでインスタンスを生成したか。
 01  FLAG-NEW           PIC S9(9) USAGE COMP VALUE 0.

 LINKAGE                SECTION.
*>-----------------------------------------------------------------
*> CBLJENV集団項目の定義
*>-----------------------------------------------------------------
 01  CBLJENV.
   02  CBLJENVCORE     USAGE POINTER.
   02  CBLJEXCEPTION   USAGE POINTER.
   02  CBLJFLAGS       PIC 1(32) USAGE BIT.
   02  CBLJSTRMAXLEN   PIC S9(9) USAGE COMP.
   02  CBLJVMOPTIONS.
     03  CBLJOPTCOUNT  PIC S9(9) USAGE COMP.
     03  CBLJOPTION-1  PIC X.
*>-----------------------------------------------------------------
*> クラス参照とオブジェクト参照
*>-----------------------------------------------------------------
 01  CLASSREF           USAGE POINTER.
 01  OBJREF             USAGE POINTER.
*>-----------------------------------------------------------------
*> SampleClass クラスに対応するCOBOL集団項目
*>-----------------------------------------------------------------
     COPY 'mylib.sample.SampleClass_Map_COPY.cbl'.

*>-----------------------------------------------------------------
*> 戻り値領域
*>-----------------------------------------------------------------
 01  CBLJ-RETURN-CODE  PIC S9(9) USAGE COMP.

 PROCEDURE              DIVISION USING CBLJENV
                                       CLASSREF
                                       OBJREF
                                       SampleClass
                        RETURNING CBLJ-RETURN-CODE.

     MOVE 0 TO CBLJ-RETURN-CODE.

*> クラス参照がNULLのとき，クラス参照を取得する
     IF CLASSREF = NULL THEN
      CALL 'CBLJGETCLASS' USING CBLJENV CLASS-NAME CLASSREF
     END-IF.

*> オブジェクト参照がNULLのとき，インスタンスを生成する
     MOVE 0 TO FLAG-NEW.
     IF OBJREF = NULL THEN
      CALL 'CBLJNEW' USING CBLJENV CLASSREF NO-ARG OBJREF

      IF RETURN-CODE NOT = 0 THEN
       MOVE RETURN-CODE TO CBLJ-RETURN-CODE
       GO TO END-PROC
      END-IF
      MOVE 1 TO FLAG-NEW
     END-IF.
```

```
*> フィールド値を取得する
*> ** classID
     CALL 'CBLJGETSTATICFIELD' USING CBLJENV CLASSREF FLD1-NAME
                       FLD1.

     CALL 'CBLJSTRINGTOX' USING CBLJENV FLD1-AREA classID
                        FLD1-LEN.

     CALL 'CBLJRELEASE' USING CBLJENV FLD1-AREA.

*> ** execData
     CALL 'CBLJINVOKE' USING CBLJENV OBJREF FLD2-GETTER NO-ARG
                       FLD2.

     IF RETURN-CODE NOT = 0 THEN
       MOVE RETURN-CODE TO CBLJ-RETURN-CODE
       GO TO END-PROC
     END-IF.

     CALL 'CBLJSTRINGTOX' USING CBLJENV FLD2-AREA execData
                       FLD2-LEN.

     CALL 'CBLJRELEASE' USING CBLJENV FLD2-AREA.

*> ** execType
     CALL 'CBLJINVOKE' USING CBLJENV OBJREF FLD3-GETTER NO-ARG
                       FLD3.

     IF RETURN-CODE NOT = 0 THEN
       MOVE RETURN-CODE TO CBLJ-RETURN-CODE
       GO TO END-PROC
     END-IF.

     COMPUTE execType = FLD3-AREA.

 END-PROC.
*> 生成したインスタンスを解放する
     IF FLAG-NEW = 1 THEN
      CALL 'CBLJRELEASE' USING CBLJENV OBJREF
     END-IF.

     EXIT PROGRAM.

 END PROGRAM           'CBLJGET_SampleClass'.

*>-----------------------------------------------------------------
*> CBLJSET_SampleClass:
*>   SampleClass クラスのフィールド値を設定する
*>-----------------------------------------------------------------
 IDENTIFICATION        DIVISION.
  PROGRAM-ID.          'CBLJSET_SampleClass'.
 ENVIRONMENT           DIVISION.
 CONFIGURATION         SECTION.
  SPECIAL-NAMES.
   DYNAMIC LENGTH STRUCTURE C-STRING IS C-STATIC-STRUCTURE.
 DATA                  DIVISION.
 WORKING-STORAGE       SECTION.
```

```
*>-----------------------------------------------------------------
*> クラス名，フィールド名，メソッド名の定義
*>-----------------------------------------------------------------
 01  CLASS-NAME         PIC X DYNAMIC C-STRING VALUE 'mylib/sampl'-
                        'e/SampleClass'.
 01  FLD1-NAME          PIC X DYNAMIC C-STRING VALUE 'classID'.
 01  FLD2-SETTER        PIC X DYNAMIC C-STRING VALUE 'setExecData'.
 01  FLD3-SETTER        PIC X DYNAMIC C-STRING VALUE 'setExecType'.
*>-----------------------------------------------------------------
*> パラメタの定義
*>-----------------------------------------------------------------
 01  FLD1.
   02  FLD1-TYPE        PIC X(256) VALUE 'Ljava/lang/String;'.
   02  FLD1-AREA        USAGE POINTER.
 01  FLD2.
   02  FLD2-TYPE        PIC X(256) VALUE 'Ljava/lang/String;'.
   02  FLD2-AREA        USAGE POINTER.
 01  FLD3.
   02  FLD3-TYPE        PIC X(1) VALUE 'I'.
   02  FILLER           PIC X(7) VALUE ALL LOW-VALUE.
   02  FLD3-AREA        PIC S9(9) USAGE COMP.
 01  RTN-VOID.
   02  RTN-TYPE         PIC X(1) VALUE 'V'.
*>-----------------------------------------------------------------
*> 引数リストの定義
*>-----------------------------------------------------------------
 01  ARG-LIST.
   02  ARGPTR           USAGE POINTER VALUE NULL.
   02  FILLER           USAGE POINTER VALUE NULL.
*> (引数なし)
 01  NO-ARG.
   02  FILLER           USAGE POINTER VALUE NULL.
*>-----------------------------------------------------------------
*> String オブジェクト <=> COBOLデータ項目変換用領域（長さ）
*>-----------------------------------------------------------------
*> FLD1-LEN: classID フィールドに対応する長さ
 01  FLD1-LEN           PIC S9(9) USAGE COMP VALUE 256.
*> FLD2-LEN: execData フィールドに対応する長さ
 01  FLD2-LEN           PIC S9(9) USAGE COMP VALUE 256.

*>-----------------------------------------------------------------
*> ワーク領域
*>-----------------------------------------------------------------
*> FLAG-NEW: このプログラムでインスタンスを生成したか。
 01  FLAG-NEW           PIC S9(9) USAGE COMP VALUE 0.

 LINKAGE                SECTION.
*>-----------------------------------------------------------------
*> CBLJENV集団項目の定義
*>-----------------------------------------------------------------
 01  CBLJENV.
   02  CBLJENVCORE      USAGE POINTER.
   02  CBLJEXCEPTION    USAGE POINTER.
   02  CBLJFLAGS        PIC 1(32) USAGE BIT.
   02  CBLJSTRMAXLEN    PIC S9(9) USAGE COMP.
   02  CBLJVMOPTIONS.
     03  CBLJOPTCOUNT   PIC S9(9) USAGE COMP.
     03  CBLJOPTION-1   PIC X.
```

```
*>-----------------------------------------------------------------
*> クラス参照とオブジェクト参照
*>-----------------------------------------------------------------
 01  CLASSREF           USAGE POINTER.
 01  OBJREF             USAGE POINTER.
*>-----------------------------------------------------------------
*> SampleClass クラスに対応するCOBOL集団項目
*>-----------------------------------------------------------------
     COPY 'mylib.sample.SampleClass_Map_COPY.cbl'.

*>-----------------------------------------------------------------
*> 戻り値領域
*>-----------------------------------------------------------------
 01  CBLJ-RETURN-CODE  PIC S9(9) USAGE COMP.

 PROCEDURE             DIVISION USING CBLJENV
                                      CLASSREF
                                      OBJREF
                                      SampleClass
                       RETURNING CBLJ-RETURN-CODE.

     MOVE 0 TO CBLJ-RETURN-CODE.

*> クラス参照がNULLのとき，クラス参照を取得する
     IF CLASSREF = NULL THEN
      CALL 'CBLJGETCLASS' USING CBLJENV CLASS-NAME CLASSREF
     END-IF.

*> オブジェクト参照がNULLのとき，インスタンスを生成する
     MOVE 0 TO FLAG-NEW.
     IF OBJREF = NULL THEN
      CALL 'CBLJNEW' USING CBLJENV CLASSREF NO-ARG OBJREF

      IF RETURN-CODE NOT = 0 THEN
       MOVE RETURN-CODE TO CBLJ-RETURN-CODE
       GO TO END-PROC
      END-IF
      MOVE 1 TO FLAG-NEW
     END-IF.
```

```
*> フィールド値を設定する
*> ** classID
     CALL 'CBLJXTOSTRING' USING CBLJENV classID FLD1-LEN
                          FLD1-AREA.

     CALL 'CBLJSETSTATICFIELD' USING CBLJENV CLASSREF FLD1-NAME
                          FLD1.

     CALL 'CBLJRELEASE' USING CBLJENV FLD1-AREA.

*> ** execData
     CALL 'CBLJXTOSTRING' USING CBLJENV execData FLD2-LEN
                          FLD2-AREA.

     COMPUTE ARGPTR = FUNCTION ADDR(FLD2).
     CALL 'CBLJINVOKE' USING CBLJENV OBJREF FLD2-SETTER ARG-LIST
                          RTN-VOID.

     IF RETURN-CODE NOT = 0 THEN
       MOVE RETURN-CODE TO CBLJ-RETURN-CODE
       GO TO END-PROC
     END-IF.

     CALL 'CBLJRELEASE' USING CBLJENV FLD2-AREA.

*> ** execType
     COMPUTE FLD3-AREA = execType.

     COMPUTE ARGPTR = FUNCTION ADDR(FLD3).
     CALL 'CBLJINVOKE' USING CBLJENV OBJREF FLD3-SETTER ARG-LIST
                          RTN-VOID.

     IF RETURN-CODE NOT = 0 THEN
       MOVE RETURN-CODE TO CBLJ-RETURN-CODE
       GO TO END-PROC
     END-IF.

 END-PROC.

     EXIT PROGRAM.

 END PROGRAM           'CBLJSET_SampleClass'.
```

## 7.3　プログラム作成支援ツールで出力されるメッセージ

ここでは，プログラム作成支援ツールが出力するメッセージ内容について記載します。

### 7.3.1　プログラム作成支援ツールで出力されるメッセージの形式

プログラム作成支援ツールが出力するメッセージは，次の形式です。

```
KCCCnnnnA-x メッセージテキスト
```

KCCC：メッセージプリフィクス

nnnn：メッセージ番号（4 けたの数字）

A：Java プログラム呼び出し機能のプログラム作成支援ツールのメッセージであることを示す。

x：メッセージレベル

　E：エラー

　W：警告

また，ここでは，メッセージを次の形式で記載します。

KCCCnnnnA-x

　メッセージテキスト

**要因**

　プログラム作成支援ツールでエラーが発生した要因を示します。

**対処**

　プログラム作成支援ツールのエラーに対する対処を示します。

メッセージ中の可変の埋め字部分は*** n ***（n は数字）で示します。

### 7.3.2　プログラム作成支援ツールで出力されるメッセージの一覧

メッセージの一覧を次に示します。

KCCC8001A-E

　前提条件となる製品がインストールされていないため，処理を続行できません。

**要因**

前提条件となる製品がインストールされていない。

**対処**

前提条件となる次に示す製品のどちらかをインストールする。

- COBOL2002 Developer Professional
- COBOL2002 Developer Professional(64)

## KCCC8002A-E

Java プログラムが実行できません。

**要因**

Java VM の実行ファイルが見つからない。

**対処**

Java プログラムを実行できるように環境を見直す。

## KCCC8003A-E

COBOL2002 のインストールフォルダが見つかりません。

**要因**

COBOL2002 がインストールされていない。

**対処**

COBOL2002 Developer Professional をインストールする。

## KCCC8004A-E

cbl2kj.jar が見つかりません。

**要因**

解析用 Jar ファイルである cbl2kj.jar が見つからない。

**対処**

COBOL2002 のインストールフォルダ¥bin に cbl2kj.jar が存在するか確認する。

## KCCC8101A-E

-Type オプションが指定されていません。

**要因**

-Type オプションが指定されていない。

**対処**

-Type オプションを指定する。

## KCCC8102A-E

-Type オプションに引数がありません。

**要因**

-Type オプションの引数を指定していない。

**対処**

-Type オプションの引数に Sample または GroupMapper を指定する。

## KCCC8103A-E

-Type オプションの引数が不正です。

**要因**

-Type オプションの引数に Sample または GroupMapper 以外の値を指定した。

**対処**

-Type オプションの引数に Sample または GroupMapper を指定する。

## KCCC8104A-E

-Format オプションに引数がありません。

**要因**

-Format オプションの引数を指定していない。

**対処**

-Format オプションの引数に fixed または free を指定する。

## KCCC8105A-E

-Format オプションの引数が不正です。

**要因**

-Format オプションの引数に fixed または free 以外の値を指定している。

**対処**

-Format オプションの引数に fixed または free を指定する。

## KCCC8106A-E

-Class オプションに引数がありません。

**要因**

-Class オプションの引数を指定していない。

**対処**

-Class オプションの引数にクラスを指定する。

## KCCC8107A-E

-Jar オプションに引数がありません。

**要因**

-Jar オプションの引数を指定していない。

**対処**

-Jar オプションの引数にクラスを指定する。

## KCCC8108A-E

-Class オプションと-Jar オプションがどちらも指定されていません。

**要因**

-Class オプションと-Jar オプションがどちらも指定されていない。

**対処**

-Class オプションまたは-Jar オプションを指定する。

## KCCC8109A-E

指定した Java アーカイブファイルが見つかりません。

**要因**

-Jar オプションに指定した Java アーカイブファイルが存在しない。

**対処**

-Jar オプションに指定する Java アーカイブファイルのパスを見直す。

## KCCC8110A-E

指定した Java アーカイブファイルが読み込めません。

**要因**

-Jar オプションに指定した Java アーカイブファイルを読み込む権限がない。

**対処**

次に示すどれかを実施する。

- -Jar オプションに読み込み権限のある Java アーカイブファイルを指定する。

- -Jar オプションに指定した Java アーカイブファイルのアクセス権を変更する。
- -Jar オプションに指定した Java アーカイブを読み込める権限を実行ユーザに与える。

### KCCC8111A-E

-ClassPath オプションに引数がありません。

**要因**

-ClassPath オプションの引数を指定していない。

**対処**

-ClassPath オプションの引数にクラスパスを指定する。

### KCCC8112A-E

-StrMaxLen オプションに引数がありません。

**要因**

-StrMaxLen オプションの引数を指定していない。

**対処**

-StrMaxLen オプションの引数に 1～1,024 の値を指定する。

### KCCC8113A-E

-StrMaxLen オプションの引数には 1～1024 の値を指定してください。

**要因**

-StrMaxLen オプションの引数に 1～1,024 以外の値を指定している。

**対処**

-StrMaxLen オプションの引数に 1～1,024 の値を指定する。

### KCCC8114A-E

-MaxArrayLength オプションに引数がありません。

**要因**

-MaxArrayLength オプションの引数を指定していない。

**対処**

-MaxArrayLength オプションの引数に 1～256 の値を指定する。

### KCCC8115A-E

-MaxArrayLength オプションの引数には 1～16,777,215 の値を指定してください。

**要因**

-MaxArrayLength オプションの引数に 1～16,777,215 以外の値を指定している。

**対処**

-MaxArrayLength オプションの引数に 1～16,777,215 の値を指定する。

## KCCC8116A-E

データ項目名を変更する場合は 1～31 文字の名前を指定してください。

**要因**

-Class オプションに指定した内部名が 1～31 文字でない。

**対処**

-Class オプションの指定する内部名を 1～31 文字で指定する。

## KCCC8117A-E

-OutDir オプションに引数がありません。

**要因**

-OutDir オプションの引数を指定していない。

**対処**

-OutDir オプションの引数にクラスパスを指定する。

## KCCC8199A-W

無効なオプションが指定されました。(*** 1 ***)

**要因**

無効なオプションを指定している。

*** 1 ***：該当のオプション

**対処**

オプションの指定を見直す。

## KCCC8201A-W

解析で必要なクラスが見つかりません。(*** 1 ***)

**要因**

Java プログラムの解析に必要な次に示すクラスが見つからない。

- 解析対象のクラス
- 解析対象のクラスのスーパークラス

- 解析対象のクラスが実装しているインタフェース
- 解析対象のクラスが import しているクラス

*** 1 ***：解析対象のクラス

**対処**

解析に必要なクラスを参照できるようにクラスの検索パスを見直す。

## KCCC8202A-E

指定した Java アーカイブファイルから Java クラスの一覧を取得できません。

**要因**

次に示すどちらかの理由で，Java アーカイブファイルから Java クラスの一覧を取得できない。

- Java アーカイブファイルに Java 実行ファイルが含まれていない。
- 不正なファイル形式の Java アーカイブファイルを指定している。

**対処**

Java 実行ファイルを 1 つ以上格納して Java アーカイブファイルを生成し直す。

## KCCC8203A-W

Java クラスのロードに失敗しました。(*** 1 ***)

**要因**

次に示すどれかの理由で，Java VM が Java クラスをロードできない。

- Java 実行ファイルのパスと完全修飾クラス名が一致していない。
- Java アーカイブファイルの構造と完全修飾クラス名が一致していない。
- Java 実行ファイルが壊れている。

*** 1 ***：ロードできなかった Java クラス

**対処**

正しい構造で Java 実行ファイルまたは Java アーカイブファイルを生成し直す。

## KCCC8210A-W

解析対象のクラスではありません。(*** 1 ***)

**要因**

指定したクラス名がインタフェースである。

*** 1 ***：Java クラス名

**対処**

-Class オプションに指定する Java クラス名を見直す。

## KCCC8301A-E

ファイルを出力できません。(*** 1 ***)

### 要因

次に示すどちらかの理由によって，出力先に Java クラス利用サンプルを書き込む権限がない。

- -OutDir オプションで出力先を指定している場合，出力先が存在しないか，または書き込み権限がない。
- -OutDir オプションで出力先を指定していない場合，カレントフォルダに書き込み権限がない。

*** 1 ***：出力ファイルパス

### 対処

次に示すどれかを実施する。

- -OutDir オプションで書き込み権限のある出力先を指定する。
- 出力先のアクセス権を変更する。
- 出力先に書き込める権限を実行ユーザに与える。

## KCCC8302A-W

語の長さが 31 文字を超えました。(*** 1 ***, *** 2 ***)

### 要因

生成した Java クラス利用サンプルに 31 文字を超える語が存在する。

*** 1 ***：Java クラス利用サンプルの行番号

*** 2 ***：該当の語の名前

### 対処

語の長さが 31 文字以下となるように Java クラス利用サンプルを修正する。

## KCCC8401A-E

ファイルを出力できません。(*** 1 ***)

### 要因

次に示すどちらかの理由によって，出力先に集団項目データ交換プログラムを書き込む権限がない。

- -OutDir オプションで出力先を指定している場合，出力先が存在しないか，または書き込み権限がない。
- -OutDir オプションで出力先を指定していない場合，カレントフォルダに書き込み権限がない。

*** 1 ***：出力ファイルパス

### 対処

次に示すどれかを実施する。

- -OutDir オプションで書き込み権限のある出力先を指定する。
- 出力先のアクセス権を変更する。
- 出力先に書き込める権限を実行ユーザに与える。

## KCCC8402A-E

ファイルを出力できません。(*** 1 ***)

### 要因

次に示すどちらかの理由によって，出力先に集団項目データ交換用データ定義を書き込む権限がない。

- -OutDir オプションで出力先を指定している場合，出力先が存在しないか，または書き込み権限がない。
- -OutDir オプションで出力先を指定していない場合，カレントフォルダに書き込み権限がない。

*** 1 ***：出力ファイルパス

### 対処

次に示すどれかを実施する。

- -OutDir オプションで書き込み権限のある出力先を指定する。
- 出力先のアクセス権を変更する。
- 出力先に書き込める権限を実行ユーザに与える。

## KCCC8403A-W

語の長さが 31 文字を超えました。(*** 1 ***, *** 2 ***)

### 要因

生成した集団項目データ交換用データ定義に 31 文字を超える語が存在する。

*** 1 ***：集団項目データ交換用データ定義の行番号

*** 2 ***：該当の語の名前

### 対処

語の長さを 31 文字以下となるように集団項目データ交換用データ定義を修正する。

## KCCC8404A-W

語の長さが 31 文字を超えました。(*** 1 ***, *** 2 ***)

### 要因

集団項目データ交換プログラムに 31 文字を超える語が存在する。

*** 1 ***：集団項目データ交換プログラムの行番号

*** 2 ***：該当の語の名前

**対処**

語の長さを 31 文字以下となるようにフィールドの取得・設定のための COBOL プログラムを修正する。

## KCCC8405A-W

対象のフィールドが見つかりません。(*** 1 ***)

**要因**

集団項目データ交換プログラムの対象になるフィールドがない。

*** 1 ***：解析対象の Java クラス名

**対処**

対象のフィールドがある Java クラスを指定する。

## KCCC8601A-E

Out of memory.

**要因**

メモリ不足のため，CBLJENV 内のクラスパスを格納する領域を確保できない。

**対処**

使用中のリソースを調査し，解放できるものは解放する。

## KCCC8602A-E

Out of memory.

**要因**

メモリ不足のため，クラス情報管理テーブルの領域を確保できない。

**対処**

使用中のリソースを調査し，解放できるものは解放する。

## KCCC8603A-E

Out of memory.

**要因**

メモリ不足のため，クラス情報管理テーブルのクラス名を格納する領域を確保できない。

**対処**

使用中のリソースを調査し，解放できるものは解放する。

## KCCC8604A-E

Out of memory.

**要因**

メモリ不足のため，クラス情報管理テーブルの内部名を格納する領域を確保できない。

**対処**

使用中のリソースを調査し，解放できるものは解放する。

## KCCC8605A-E

Out of memory.

**要因**

メモリ不足のため，フィールド情報を格納する内部領域を確保できない。

**対処**

使用中のリソースを調査し，解放できるものは解放する。

## 7.4　プログラム作成支援ツールの注意事項／制限事項

プログラム作成支援ツールの注意事項および制限事項について説明します。

### 7.4.1　使用上の注意事項

プログラム作成支援ツールの注意事項について説明します。

#### (1)　初期化・終了処理について

Java クラス利用サンプルでは Java プログラム呼び出し機能の初期化，終了処理，および CBLJENV 集団項目をプログラム単位に生成しますが，実際の処理ではスレッド単位で一度だけ実行・定義するように構成してください。

#### (2)　String クラスの扱いについて

フィールドやパラメタの型が String クラスの場合，COBOL データ項目の定義は-StrMaxLen オプションに指定した長さの英数字項目で生成されます。必要に応じて，取り扱うデータにあわせて変更してください。対応する長さ領域も変更する必要があるため，注意が必要です。

例

集団項目データ交換プログラムの生成で，-StrMaxLen オプションに 256 を指定したとき，生成ソースでデータ定義を日本語項目（最大長 2,048）に変更する。

集団項目データ交換用データ定義（sample.SampleClass_Map_COPY.cbl）の定義

```
02     execData   PIC X(256).
```

↓

```
02     execData   PIC X(2048).
```

集団項目データ交換プログラム（sample.SampleClass_Map.cbl）の定義

```
*> FLDn-LEN: execData フィールドに対応する長さ
 01     FLDn-LEN  PIC S9(9) USAGE COMP VALUE 256
```

↓

```
*> FLDn-LEN: execData フィールドに対応する長さ
 01     FLDn-LEN  PIC S9(9) USAGE COMP VALUE 2048
```

#### (3)　COBOL85 規格によるコンパイルについて

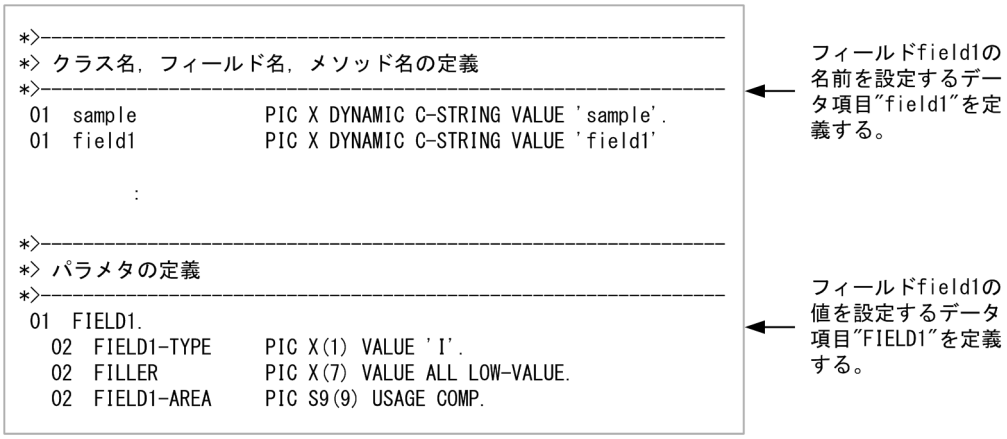
プログラム作成支援ツールが生成する COBOL プログラムは ccbl コマンドまたは-Compati85 オプションを指定した ccbl2002 コマンドでコンパイルしないでください。

プログラム生成ツールは COBOL2002 規格に従って COBOL プログラムを生成します。そのため，プログラム名の長さなどが COBOL85 規格に沿わない形式で出力され，上記の方法でコンパイルした場合にエラーとなるときがあります。

## (4)　データ項目名の重複について

COBOL プログラムのコンパイルエラーの原因の一つとして，プログラム作成支援ツールが生成する COBOL プログラムのデータ項目名が重複していることが考えられます。この場合，名前が重複しないようにデータ項目名を変更する必要があります。

例えば，次に示すようにプログラム作成支援ツールが生成する COBOL プログラムのデータ項目名が Java のフィールド名と同じ場合，データ項目名が重複します。Java のクラス名やメソッド名でも同様です。




なお，コンパイル時に，大文字・小文字の区別や全角・半角の区別をするかどうかはコンパイラオプションに依存します。

## (5)　引数があるコンストラクタの使用について

集団項目データ交換プログラムは，インスタンス（オブジェクト参照）を取得する処理に引数なしで CBLJNEW サービスルーチンを呼び出すコードを生成します。このため，集団項目データ交換プログラムに渡す Java オブジェクト参照に呼び出し元で生成したインスタンスのオブジェクト参照を設定しないで呼び出したとき，引数がある public コンストラクタでインスタンスを取得できません。引数がある public コンストラクタでインスタンスを取得したい場合，次のどちらかの対処をしてください。

- 集団項目データ交換プログラム呼び出し時の第 3 引数（オブジェクト参照）に，呼び出し元でインスタンスを生成して取得したオブジェクト参照を指定する。
- 生成した集団項目データ交換プログラムの CBLJNEW サービスルーチンの呼び出しコードを，ほかの public コンストラクタを呼び出すように修正する。

  CBLJNEW サービスルーチンの呼び出しコードについては，「7.2.3　集団項目データ交換プログラムの生成」の「(2)　集団項目データ交換プログラムの構造」を参照してください。

ほかの public コンストラクタを呼び出すための COBOL プログラムは，Java クラス利用サンプルを生成して参考にしてください。

### (6)　コンパイラオプションの指定について

プログラム作成支援ツールで生成したプログラムを変更しないで，次のどれかのコンパイラオプションを指定してコンパイルすると，実行時に COBOL 実行時エラーなどの不正な動作となることがあります。

- -BigEndian
- -DebugCompati （集団項目データ交換プログラムだけ）
- -DebugRange （集団項目データ交換プログラムだけ）

これらのコンパイラオプションを指定しないか，または生成したプログラムを変更して使用してください。使用方法を次に示します。

-BigEndian コンパイラオプションを指定するとき

次の手順で使用できます。詳細については，「4.1.1　COBOL2002 のコンパイラオプションを指定するときの注意事項」の「(1)　-BigEndian コンパイラオプション（バイナリデータをビッグエンディアン形式で取り扱う）」を参照してください。

1. 生成したプログラムで，サービスルーチンの引数に指定するデータ項目の定義を変更する。
2. 実行時に環境変数 CBLJRTBIGENDIAN を指定する。

-DebugCompati コンパイラオプションまたは-DebugRange コンパイラオプションを指定するとき

集団項目データ交換プログラム中の CBLJENV 集団項目の定義を，呼び出し元の CBLJENV 集団項目の定義と一致するように変更することで使用できます。

## 7.4.2　制限値と制限事項

プログラム作成支援ツールの制限値と制限事項について説明します。

### (1)　使用できる語の長さの制限について

COBOL2002 コンパイラでは，次の制限値を超えるとコンパイル時にコンパイルエラーになります。

- 語の長さ：31 文字
- 定数の長さ：160 文字

Java の名称を使用した定数の長さが 160 文字を超える場合に，そのまま使用したいときは，COBOL プログラムのコンパイル時に-LiteralExtend オプションを指定してください。※

また，データ名などの COBOL の語に Java の名称そのままの文字列を使用するため，語の長さが 31 文字を超えてコンパイルエラーになることがあります。この場合，ツールの警告メッセージに行番号と該当の語を出力するので，生成された COBOL プログラムを必要に応じて修正してください。

注※

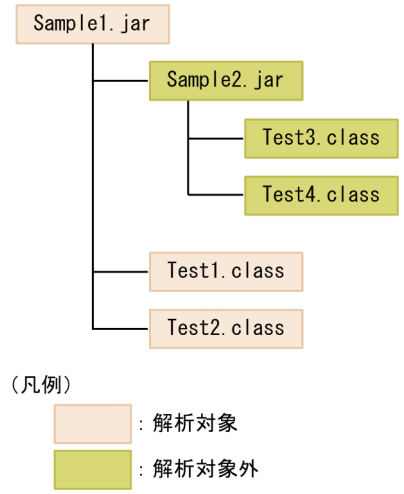
-LiteralExtend オプションを指定すると最大長が拡張されます（プログラム名は 1,024 文字，それ以外では 8,019 文字まで）。

## (2)　Java アーカイブファイルに含まれる Java アーカイブファイルについて

プログラム作成支援ツールでは，Java アーカイブファイルに含まれる Java アーカイブファイル，およびその中の Java 実行ファイル（以降，内包するクラスと表記します）は解析対象としません。

解析対象と解析対象外の例を次の図に示します。

図 7-10　プログラム作成支援ツールの解析対象と解析対象外の例




内包するクラスを解析する場合は，Java アーカイブファイル（上記の例では Sample1.jar）から対象のファイルを取り出してプログラム作成支援ツールで解析してください。

内包するクラスを解析対象としない場合，展開する必要はありません。例えば，Test1.class を解析対象とする場合，Test1.class が Test3.class の機能を使用していても，Test3.class を取り出す必要はありません。

# 付録

# 付録 A　注意事項／制限事項

ここでは，Java プログラム呼び出し機能を使用する際の注意事項，および制限事項について説明します。

## 付録 A.1　実行時のファイル出力の監視での注意事項

Java プログラム呼び出し機能を使用すると，デフォルトの設定で，次に示すファイルが出力されますが，自動的には削除されません。ディスク容量不足などが発生しないように監視してください。

- 実行時エラー情報出力ファイル
  Java プログラム呼び出し機能で実行時エラーが発生するごとに，一つのファイルが作成されます。
  実行時エラー情報出力ファイルについては，「5.3　実行時エラー情報の出力」を参照してください。
- Java VM 起動オプション情報が出力されるファイル
  Java プログラム呼び出し機能で Java VM を起動するごとに，一つのファイルに追加書きされます。
  Java VM 起動オプション情報が出力されるファイルについては，「5.4　Java VM 起動オプション情報の収集」を参照してください。

## 付録 A.2　Windows OS 固有の注意事項

Java プログラム呼び出し機能を使用する場合の，Windows OS 固有の注意事項を次に示します。

- フォルダ名，ファイル名，プログラムへの入力文字列，および環境変数に指定できる文字は，シフト JIS の範囲だけです。JIS X0213 の第 3 水準漢字，および第 4 水準漢字を含む Unicode の文字は使用できません。
- cbl2kjgen コマンドが生成するプログラムの出力先や，Java プログラム呼び出し機能が出力するデバッグ情報ファイルなどの出力先として，Windows リソース保護（WRP）対象のフォルダを指定しないでください。指定した場合，プログラムの実行権限によってファイルを出力できないことがあります。WRP については，各 OS のヘルプなどを参照してください。

## 付録 A.3　制限値と制限事項

Java プログラム呼び出し機能の制限値と制限事項について説明します。

### (1)　制限値

Java プログラム呼び出し機能の制限値を次の表に示します。

表 A–1　Java プログラム呼び出し機能の制限値

| 項番 | 制限項目 | 制限値 |
|---|---|---|
| 1 | 扱えるクラス名の長さ | 1,024 バイト |
| 2 | 扱えるメソッド名の長さ | 1,024 バイト |
| 3 | 扱えるフィールド名の長さ | 1,024 バイト |
| 4 | 扱える型名（'L'と';'を含む）の長さ | 1,024 バイト |
| 5 | 扱えるコンストラクタおよびメソッドのパラメタ数 | 16 個（引数リスト型集団項目で指定する場合，終端を示す NULL 値の要素は含まない） |

## (2)　制限事項

Java プログラム呼び出し機能の制限事項を次に示します。

- Java プログラム呼び出し機能を使用して呼び出した Java プログラムから，同じスレッド内で COBOL プログラムを呼び出すことはできません。

  Java プログラムから COBOL プログラムを呼び出すときに Cosminexus 連携機能を使用した場合は，Cosminexus 連携機能が例外をスローします。

  Cosminexus 連携機能を使用しないで呼び出した場合の動作は保証しません。

  COBOL プログラムを別プロセスで呼び出すなどの対処をしてください。
- Unicode で多バイトとなる文字を含む Java のクラス名，メソッド名，フィールド名を使用する場合，Unicode 機能と同時に使用できません。

# 付録 B　Java プログラム呼び出し機能の詳細メッセージ

Java プログラム呼び出し機能のサービスルーチンの実行時にエラーが発生した場合，実行時のメッセージの「エラー情報」に Java プログラム呼び出し機能の詳細メッセージが出力されます。

Java プログラム呼び出し機能の詳細メッセージが出力される実行時のメッセージを次に示します。

- KCCC7903R-U
  プログラムや環境の設定で対処できないエラーが発生した場合に出力されます。
- KCCC7904R-S
  プログラムや環境の設定で対処できるエラーが発生した場合に出力されます。

これらのメッセージが出力された場合，COBOL プログラムの実行は中止されます。

ここでは，「エラー情報」に出力される Java プログラム呼び出し機能の詳細メッセージについて説明します。実行時メッセージの形式や説明については，マニュアル「COBOL2002　メッセージ」の「メッセージの説明の形式（実行時のメッセージ)」を参照してください。

なお，詳細メッセージが出力される実行時メッセージのメッセージ本文，要因，および対処は次に示すとおりです。

| メッセージ ID | メッセージテキスト | 要因 | 対処 |
|---|---|---|---|
| KCCC7903R-U | Java プログラム呼び出し機能で実行時エラーが発生しました。<br>プログラム名＝*** 1 ***<br>行番号／欄＝*** 2 ***<br>エラー情報＝*** 3 *** | Java プログラム呼び出し機能で実行時エラーが発生した。 | 当社保守員に連絡する。 |
| KCCC7904R-S | Java プログラム呼び出し機能で実行時エラーが発生しました。<br>プログラム名＝*** 1 ***<br>行番号／欄＝*** 2 ***<br>エラー情報＝*** 3 *** | Java プログラム呼び出し機能で実行時エラーが発生した。 | エラー情報を参考に原因を調査し，プログラムを修正するか，または実行環境の設定を見直して再実行する。 |

## 付録 B.1　詳細メッセージの形式

Java プログラム呼び出し機能が出力する詳細メッセージの形式を次に示します。

```
［詳細メッセージ番号］メッセージテキスト
```

詳細メッセージ番号：

　詳細メッセージの番号（4 けたの数字）

メッセージテキスト：

詳細メッセージのテキスト

また，ここでは，詳細メッセージを次の形式で記載します。

メッセージテキスト中の可変の埋め字部分は*** n ***（n は数字）で示します。

### [詳細メッセージ番号]

メッセージテキスト

**要因**

実行時エラーが発生した要因を示します。

**対処**

実行時エラーに対するプログラム作成者またはオペレータの対処を示します。

## 付録 B.2　詳細メッセージの一覧

Java プログラム呼び出し機能が出力する詳細メッセージを次に示します。

### [0001]

引数に誤りがあります。(*** 1 ***)

**要因**

CBLJENV 集団項目を示すアドレスが NULL である。引数の指定が BY REFERENCE 指定でないことが考えられる。

*** 1 ***：サービスルーチン名

**対処**

サービスルーチンに指定した CBLJENV 集団項目を確認し，正しく指定する。

### [0002]

引数に誤りがあります。(*** 1 ***)

**要因**

引数を示すアドレスが NULL である。引数の指定が BY REFERENCE 指定でないことが考えられる。

*** 1 ***：サービスルーチン名

**対処**

サービスルーチンに指定した引数を確認し，正しく指定する。

## [0003]

引数の個数が上限値を超えています。(*** 1 ***)

**要因**

引数の個数が上限値を超えている。

*** 1 ***：サービスルーチン名

**対処**

引数の個数が 16 個を超えるクラスメソッドを使用しない。

## [0004]

予約領域が LOW-VALUE ではありません。(*** 1 ***)

**要因**

パラメタ型集団項目の予約領域が LOW-VALUE ではない。

*** 1 ***：サービスルーチン名

**対処**

予約領域に LOW-VALUE を設定する。

## [0005]

引数のデータ型を示す文字列が正しくありません。(*** 1 ***)

**要因**

引数で指定するパラメタ型集団項目の Java のデータ型を示す文字が正しくない。

*** 1 ***：サービスルーチン名

**対処**

正しい値を指定する。

## [0009]

返却項目のデータ型を示す文字列が正しくありません。(*** 1 ***)

**要因**

返却項目の Java のデータ型を示す文字列が正しくない。

*** 1 ***：サービスルーチン名

**対処**

正しい値を指定する。

## [0010]

Java VM が起動されていません。(*** 1 ***)

### 要因

Java VM が起動されていない。

*** 1 ***：サービスルーチン名

### 対処

明示的または暗黙的に CBLJINITIALIZE サービスルーチンを呼び出しているかを確認する。

## [0011]

Java VM の初期化スレッドを開始できませんでした。(*** 1 ***)

### 要因

スレッドを開始するシステム関数でエラーが発生した。

*** 1 ***：エラーの番号

### 対処

エラー番号から原因を調査する。原因がリソース不足の場合は，不要なリソースを解放して再実行する。それ以外の場合は，保守員に連絡する。

## [0012]

Java VM の初期化でスレッド同期処理に失敗しました。(*** 1 ***)

### 要因

同期処理を実行したシステム関数でエラーが発生した。

*** 1 ***：エラーの番号

### 対処

エラー番号から原因を調査する。原因がリソース不足の場合は，不要なリソースを解放して再実行する。それ以外の場合は，保守員に連絡する。

## [0013]

Java VM の初期化でスレッド同期処理に失敗しました。(*** 1 ***)

### 要因

同期処理を実行したシステム関数でエラーが発生した。

*** 1 ***：エラーの番号

**対処**

エラー番号から原因を調査する。原因がリソース不足の場合は，不要なリソースを解放して再実行する。それ以外の場合は，保守員に連絡する。

## [0014]

Java VM の初期化でスレッド同期処理に失敗しました。(*** 1 ***)

**要因**

同期処理を実行したシステム関数でエラーが発生した。

*** 1 ***：エラーの番号

**対処**

エラー番号から原因を調査する。原因がリソース不足の場合は，不要なリソースを解放して再実行する。それ以外の場合は，保守員に連絡する。

## [0015]

Java VM の初期化でスレッド同期処理に失敗しました。(*** 1 ***)

**要因**

同期処理を実行したシステム関数でエラーが発生した。

*** 1 ***：エラーの番号

**対処**

エラー番号から原因を調査する。原因がリソース不足の場合は，不要なリソースを解放して再実行する。それ以外の場合は，保守員に連絡する。

## [0016]

Java VM の初期化でスレッド同期処理に失敗しました。(*** 1 ***)

**要因**

同期処理を実行したシステム関数でエラーが発生した。

*** 1 ***：エラーの番号

**対処**

エラー番号から原因を調査する。原因がリソース不足の場合は，不要なリソースを解放して再実行する。それ以外の場合は，保守員に連絡する。

## [0017]

Java VM の初期化でスレッド同期処理に失敗しました。(*** 1 ***)

**要因**

同期処理を実行したシステム関数でエラーが発生した。

*** 1 ***：エラーの番号

**対処**

エラー番号から原因を調査する。原因がリソース不足の場合は，不要なリソースを解放して再実行する。それ以外の場合は，保守員に連絡する。

## [0018]

Java VM の初期化でスレッド同期処理に失敗しました。(*** 1 ***)

**要因**

同期処理を実行したシステム関数でエラーが発生した。

*** 1 ***：エラーの番号

**対処**

エラー番号から原因を調査する。原因がリソース不足の場合は，不要なリソースを解放して再実行する。それ以外の場合は，保守員に連絡する。

## [0020]

Out of memory.(*** 1 ***)

**要因**

メモリ不足が発生した。

*** 1 ***：サービスルーチン名

**対処**

使用中のリソースを調査し，解放できるものは解放する。

また，オブジェクト参照の解放漏れがないかも確認する。

## [0021]

Out of memory.(*** 1 ***)

**要因**

メモリ不足が発生した。

*** 1 ***：サービスルーチン名

**対処**

使用中のリソースを調査し，解放できるものは解放する。

また，オブジェクト参照の解放漏れがないかも確認する。

## [0022]

```
Out of memory.(*** 1 ***)
```

**要因**

メモリ不足が発生した。

*** 1 ***：サービスルーチン名

**対処**

使用中のリソースを調査し，解放できるものは解放する。

また，オブジェクト参照の解放漏れがないかも確認する。

## [0023]

```
Out of memory.(*** 1 ***)
```

**要因**

メモリ不足が発生した。

*** 1 ***：サービスルーチン名

**対処**

使用中のリソースを調査し，解放できるものは解放する。

また，オブジェクト参照の解放漏れがないかも確認する。

## [0024]

```
Out of memory.(*** 1 ***)
```

**要因**

メモリ不足が発生した。

*** 1 ***：サービスルーチン名

**対処**

使用中のリソースを調査し，解放できるものは解放する。

また，オブジェクト参照の解放漏れがないかも確認する。

## [0025]

```
Out of memory.(*** 1 ***)
```

**要因**

メモリ不足が発生した。

*** 1 ***：サービスルーチン名

**対処**

使用中のリソースを調査し，解放できるものは解放する。

また，オブジェクト参照の解放漏れがないかも確認する。

## [0026]

```
Out of memory.(*** 1 ***)
```

**要因**

メモリ不足が発生した。

*** 1 ***：サービスルーチン名

**対処**

使用中のリソースを調査し，解放できるものは解放する。

また，オブジェクト参照の解放漏れがないかも確認する。

## [0030]

```
論理エラーが発生しました。(*** 1 ***)
```

**要因**

論理エラーが発生した。

**対処**

当社保守員に連絡する。

## [0031]

```
論理エラーが発生しました。(*** 1 ***／*** 2 ***)
```

**要因**

論理エラーが発生した。

**対処**

当社保守員に連絡する。

## [0032]

```
論理エラーが発生しました。(*** 1 ***)
```

**要因**

論理エラーが発生した。

**対処**

当社保守員に連絡する。

[0033]

論理エラーが発生しました。(*** 1 ***)

**要因**

論理エラーが発生した。

**対処**

当社保守員に連絡する。

[0034]

論理エラーが発生しました。(*** 1 ***／*** 2 ***)

**要因**

論理エラーが発生した。

**対処**

当社保守員に連絡する。

[0035]

論理エラーが発生しました。(*** 1 ***)

**要因**

論理エラーが発生した。

**対処**

当社保守員に連絡する。

[0036]

論理エラーが発生しました。(*** 1 ***／*** 2 ***)

**要因**

論理エラーが発生した。

**対処**

当社保守員に連絡する。

[0037]

論理エラーが発生しました。(*** 1 ***／*** 2 ***)

**要因**

論理エラーが発生した。

**対処**

当社保守員に連絡する。

## [0038]

論理エラーが発生しました。(*** 1 ***／*** 2 ***)

**要因**

論理エラーが発生した。

**対処**

当社保守員に連絡する。

## [0039]

論理エラーが発生しました。(*** 1 ***)

**要因**

論理エラーが発生した。

**対処**

当社保守員に連絡する。

## [0040]

論理エラーが発生しました。(*** 1 ***)

**要因**

論理エラーが発生した。

**対処**

当社保守員に連絡する。

## [0041]

論理エラーが発生しました。(*** 1 ***／*** 2 ***)

**要因**

論理エラーが発生した。

**対処**

当社保守員に連絡する。

## [0042]

論理エラーが発生しました。(*** 1 ***)

**要因**

論理エラーが発生した。

**対処**

当社保守員に連絡する。

## [0043]

論理エラーが発生しました。(*** 1 ***)

**要因**

論理エラーが発生した。

**対処**

当社保守員に連絡する。

## [0044]

論理エラーが発生しました。(*** 1 ***)

**要因**

論理エラーが発生した。

**対処**

当社保守員に連絡する。

## [0045]

論理エラーが発生しました。(*** 1 ***)

**要因**

論理エラーが発生した。

**対処**

当社保守員に連絡する。

## [0046]

論理エラーが発生しました。(*** 1 ***)

**要因**

論理エラーが発生した。

**対処**

当社保守員に連絡する。

## [0050]

論理エラーが発生しました。(*** 1 ***／*** 2 ***)

**要因**

論理エラーが発生した。

**対処**

当社保守員に連絡する。

## [0051]

論理エラーが発生しました。(*** 1 ***／*** 2 ***)

**要因**

論理エラーが発生した。

**対処**

当社保守員に連絡する。

## [0060]

JNI で例外が発生しました。(*** 1 ***／*** 2 ***，例外名＝*** 3 ***)

**要因**

JNI で例外が発生した。

*** 1 ***：サービスルーチン名

*** 2 ***：JNI 関数名

*** 3 ***：Java 例外名

発生する Java 例外

- java.lang.ArrayIndexOutOfBoundsException：インデクス範囲外参照
- java.lang.ArrayStoreException：オブジェクトが配列オブジェクトの要素の型またはそのサブクラスのオブジェクトでない
- java.lang.ExceptionInInitializerError：クラス初期化失敗
- java.lang.InstantiationException：クラスがインタフェースまたは abstract クラスである
- java.lang.NoSuchMethodError：メソッドが見つからない
- java.lang.OutOfMemoryError：メモリ不足
- その他の JNI 実行中に発生した例外

**対処**

Java 例外の内容に対応した対処を行う。

## [0061]

*** 1 *** に対して JNI で例外が発生しました。(*** 2 ***／*** 3 ***，例外名＝*** 4 ***)

### 要因

JNI で例外が発生した。

*** 1 ***：Java のメソッド名／クラス名／フィールド名など

*** 2 ***：サービスルーチン名

*** 3 ***：JNI 関数名

*** 4 ***：Java 例外名

発生する Java 例外

- java.lang.ClassCircularityError：クラス定義が循環している
- java.lang.ClassFormatError：.class ファイルが不正である
- java.lang.ExceptionInInitializerError：クラス初期化失敗
- java.lang.NoClassDefFoundError：クラス定義が見つからない
- java.lang.NoSuchFieldError：フィールドが見つからない
- java.lang.NoSuchMethodError：メソッドが見つからない
- java.lang.OutOfMemoryError：メモリ不足
- その他の JNI 実行中に発生した例外

### 対処

Java 例外の内容に対応した対処を行う。

## [0064]

*** 1 *** の構築中に Java で例外が発生しました。(*** 2 ***，例外名＝*** 3 ***)

### 要因

Java で例外が発生した。

*** 1 ***：Java のメソッド名／クラス名／フィールド名など

*** 2 ***：サービスルーチン名

*** 3 ***：Java 例外名

発生する Java 例外

Java クラスを構築している間に発生した例外

### 対処

Java 例外の内容に対応した対処を行う。

## [0065]

*** 1 *** の呼び出し中に Java で例外が発生しました。(*** 2 ***，例外名＝*** 3 ***)

### 要因

Java で例外が発生した。

*** 1 ***：Java のメソッド名／クラス名／フィールド名など

*** 2 ***：サービスルーチン名

*** 3 ***：Java 例外名

発生する Java 例外

Java メソッドを実行している間に発生した例外

### 対処

Java 例外の内容に対応した対処を行う。

## [0091]

Cannot present the detail message.(*** 1 ***/*** 2 ***)

### 要因

詳細メッセージの出力処理で論理エラーが発生した。COBOL2002 でインストールしたファイルがないか，または壊れている。

*** 1 ***：サービスルーチン名

*** 2 ***：本来出力する詳細メッセージ番号

### 対処

COBOL2002 を再インストールする。再インストールしてもこのメッセージが出力される場合は，当社保守員に連絡する。

## [0101]

CBLJSTRMAXLEN 項目に指定した値が正しくありません。

### 要因

CBLJSTRMAXLEN 項目に指定した値が正しくない。

### 対処

CBLJSTRMAXLEN 項目に 1～1,024 の範囲の値を設定する。

## [0102]

CBLJVMOPTIONS 項目の CBLJOPTCOUNT に指定した値が正しくありません。

**要因**

CBLJVMOPTIONS 項目の CBLJOPTCOUNT に指定した値が正しくない。負の値が設定されたことが考えられる。

**対処**

CBLJVMOPTIONS 項目の CBLJOPTCOUNT に 0 以上の値を設定する。

## [0103]

実行環境の初期化に失敗しました。

**要因**

メモリが不足しているか，または実行中のスレッド数が多過ぎる。

**対処**

使用中の領域やプロセスを調査し，削除できるものは削除して再実行する。

## [0110]

前提バージョンより古い JDK を使用しています。

**要因**

前提バージョンより古い JDK を使用している。

**対処**

前提バージョン以降の JDK を使用する。

## [0111]

Java VM をロードおよび初期化できません。(*** 1 ***)

**要因**

JNI_CreateJavaVM 関数で，Java VM をロードおよび初期化できない。または同じプロセスで，すでに Java 実行環境が削除された。

*** 1 ***：JNI_CreateJavaVM 関数の戻り値

**対処**

戻り値をもとにエラーの原因を調査し，対策する。

JNI 関数の戻り値とエラーの原因については，JNI 関数のドキュメントを参照のこと。

## [0112]

作成済みの Java VM を取得できません。(*** 1 ***)

**要因**

JNI_GetCreatedJavaVMs 関数で，作成済みの Java VM を取得できない。

*** 1 ***：JNI_GetCreatedJavaVMs 関数の戻り値

**対処**

戻り値をもとにエラーの原因を調査し，対策する。

JNI 関数の戻り値とエラーの原因については，JNI 関数のドキュメントを参照のこと。

## [0113]

Java VM へ現在のスレッドを接続できません。(*** 1 ***)

**要因**

AttachCurrentThread 関数で，Java VM へ現在のスレッドを接続できない。

*** 1 ***：AttachCurrentThread 関数の戻り値

**対処**

戻り値をもとにエラーの原因を調査し，対策する。

JNI 関数の戻り値とエラーの原因については，JNI 関数のドキュメントを参照のこと。

## [0201]

クラス名が指定されていません。

**要因**

長さが 0 か，または空白だけのクラス名を指定した。

**対処**

正しいクラス名を指定する。

## [0202]

指定したクラス名の長さが正しくありません。

**要因**

指定したクラス名の長さが制限値を超えている。

**対処**

制限値の範囲のクラス名を指定する。

## [0301]

クラス参照が NULL です。

**要因**

引数で指定したクラス参照が NULL である。

**対処**

正しいクラス参照を指定する。

### [0302]

正しいクラス参照ではありません。

**要因**

不正なクラス参照か，またはクラス参照以外を指定した。

**対処**

正しいクラス参照を指定する。

### [0303]

クラスフィールド名が指定されていません。

**要因**

長さが 0 か，または空白だけのクラスフィールド名を指定した。

**対処**

正しいクラスフィールド名を指定する。

### [0304]

クラスフィールド名の長さが正しくありません。

**要因**

引数で指定したクラスフィールド名の長さが制限値を超えている。

**対処**

制限値の範囲のクラスフィールド名を指定する。

### [0305]

データ型を示す文字列が正しくありません。

**要因**

パラメタ型集団項目のデータ型を示す文字列が正しくない。

**対処**

パラメタ型集団項目のデータ型に正しい文字列を指定する。

[0307]

引数で指定したクラス名が正しくありません。(*** 1 ***／*** 2 ***)

**要因**

引数で指定したクラス名と Java が返したクラスオブジェクトのクラス名が異なる。

*** 1 ***：引数で指定したクラス名

*** 2 ***：Java が返したクラスオブジェクトのクラス名

**対処**

パラメタ型集団項目のデータ型に正しい文字列を指定する。

[0401]

クラス参照が NULL です。

**要因**

引数で指定するクラス参照が NULL である。

**対処**

正しいクラス参照を指定する。

[0402]

正しいクラス参照ではありません。

**要因**

不正なクラス参照か，またはクラス参照以外を指定した。

**対処**

正しいクラス参照を指定する。

[0403]

クラスメソッド名が指定されていません。

**要因**

長さが 0 か，または空白だけのクラスメソッド名を指定した。

**対処**

正しいクラスメソッド名を指定する。

[0404]

クラスメソッド名の長さが正しくありません。

**要因**

引数で指定したクラスメソッド名の長さが制限値を超えている。

**対処**

制限値を超えない範囲のクラスメソッド名を指定する。

## [0409]

返却値で指定したクラス名が正しくありません。(*** 1 ***／*** 2 ***)

**要因**

返却値で指定したクラス名と Java が返したクラスオブジェクトのクラス名が異なる。

*** 1 ***：返却値で指定したクラス名

*** 2 ***：Java が返したクラスオブジェクトのクラス名

**対処**

パラメタ型集団項目のデータ型に正しい文字列を指定する。

## [0501]

クラス参照が NULL です。

**要因**

クラス参照が NULL である。

**対処**

正しいクラス参照を指定する。

## [0502]

正しいクラス参照ではありません。

**要因**

不正なクラス参照か，またはクラス参照以外を指定した。

**対処**

正しいクラス参照を指定する。

## [0601]

オブジェクト参照が NULL です。

**要因**

オブジェクト参照が NULL である。

**対処**

正しいオブジェクト参照を指定する。

### [0602]

正しいオブジェクト参照ではありません。

**要因**

不正なオブジェクト参照か，またはオブジェクト参照以外を指定した。

**対処**

正しいオブジェクト参照を指定する。

### [0603]

フィールド名が指定されていません。

**要因**

長さが 0 か，または空白だけのフィールド名を指定した。

**対処**

正しいフィールド名を指定する。

### [0604]

フィールド名の長さが正しくありません。

**要因**

引数で指定したフィールド名の長さが制限値を超えている。

**対処**

制限値の範囲のフィールド名を指定する。

### [0605]

データ型を示す文字列が正しくありません。

**要因**

データ型を示す文字列が正しくない。

**対処**

パラメタ型集団項目のデータ型に正しい文字列を指定する。

### [0607]

引数で指定したクラス名が正しくありません。(*** 1 ***／*** 2 ***)

**要因**

引数で指定したクラス名と Java が返したクラスオブジェクトのクラス名が異なる。

*** 1 ***：引数で指定したクラス名

*** 2 ***：Java が返したクラスオブジェクトのクラス名

**対処**

パラメタ型集団項目のデータ型に正しい文字列を指定する。

## [0701]

オブジェクト参照が NULL です。

**要因**

オブジェクト参照を示すアドレスが NULL である。

**対処**

正しいオブジェクト参照を指定する。

## [0702]

正しいオブジェクト参照ではありません。

**要因**

不正なオブジェクト参照か，またはオブジェクト参照以外を指定した。

**対処**

正しいオブジェクト参照を指定する。

## [0703]

メソッド名が指定されていません。

**要因**

長さが 0 か，または空白だけのメソッド名を指定した。

**対処**

正しいメソッド名を指定する。

## [0704]

指定したメソッド名の長さが正しくありません。

**要因**

指定したメソッド名の長さが制限値を超えている。

**対処**

制限値の範囲のメソッド名を指定する。

### [0709]

返却値で指定したクラス名が正しくありません。(*** 1 ***／*** 2 ***)

**要因**

返却値で指定したクラス名と Java が返したクラスオブジェクトのクラス名が異なる。

*** 1 ***：返却値で指定したクラス名

*** 2 ***：Java が返したクラスオブジェクトのクラス名

**対処**

パラメタ型集団項目のデータ型に正しい文字列を指定する。

### [0801]

オブジェクト参照が NULL です。

**要因**

オブジェクト参照を示すアドレスが NULL である。

**対処**

正しいオブジェクト参照を指定する。

### [0802]

正しいオブジェクト参照ではありません。

**要因**

不正なオブジェクト参照か，またはオブジェクト参照以外を指定した。

**対処**

正しいオブジェクト参照を指定する。

### [0901]

クラス参照が NULL です。

**要因**

クラス参照を示すアドレスが NULL である。

**対処**

正しいクラス参照を指定する。

[0902]

正しいクラス参照ではありません。

**要因**

不正なクラス参照か，またはクラス参照以外を指定した。

**対処**

正しいクラス参照を指定する。

[1001]

Java VM から現在のスレッドを切り離しできません。(*** 1 ***)

**要因**

DetachCurrentThread 関数で，Java VM から現在のスレッドを切り離しできない。

*** 1 ***：DetachCurrentThread 関数の戻り値

**対処**

戻り値をもとにエラーの原因を調査し，対策する。

JNI 関数の戻り値とエラーの原因については，JNI 関数のドキュメントを参照のこと。

[2101]

オブジェクト参照が NULL です。

**要因**

オブジェクト参照を示すアドレスが NULL である。

**対処**

正しいオブジェクト参照を指定する。

[2102]

正しいオブジェクト参照ではありません。

**要因**

不正なオブジェクト参照か，またはオブジェクト参照以外を指定した。

**対処**

正しいオブジェクト参照を指定する。

[2201]

クラス参照が NULL です。

**要因**

クラス参照を示すアドレスが NULL である。

**対処**

正しいクラス参照を指定する。

### [2202]

正しいクラス参照ではありません。

**要因**

不正なクラス参照か，またはクラス参照以外を指定した。

**対処**

正しいクラス参照を指定する。

### [2203]

指定した英数字項目の長さが正しくありません。

**要因**

指定した英数字項目の長さが正しくない。

**対処**

正しい長さを指定する。

### [2301]

クラス参照が NULL です。

**要因**

クラス参照を示すアドレスが NULL である。

**対処**

正しいクラス参照を指定する。

### [2302]

正しいクラス参照ではありません。

**要因**

不正なクラス参照か，またはクラス参照以外を指定した。

**対処**

正しいクラス参照を指定する。

[3101]

オブジェクト参照が NULL です。

**要因**

オブジェクト参照を示すアドレスが NULL である。

**対処**

正しいオブジェクト参照を指定する。

[3102]

正しいオブジェクト参照ではありません。

**要因**

不正なオブジェクト参照か，またはオブジェクト参照以外を指定した。

**対処**

正しいオブジェクト参照を指定する。

[3103]

指定した英数字項目の長さが正しくありません。

**要因**

指定した英数字項目の長さが正しくない。

**対処**

正しい長さを指定する。

[3201]

クラス参照が NULL です。

**要因**

クラス参照を示すアドレスが NULL である。

**対処**

正しいクラス参照を指定する。

[3202]

正しいオブジェクト参照ではありません。

**要因**

不正なオブジェクト参照か，またはオブジェクト参照以外を指定した。

**対処**

正しいオブジェクト参照を指定する。

### [3203]

正しいクラス参照ではありません。

**要因**

不正なクラス参照か，またはクラス参照以外を指定した。

**対処**

正しいクラス参照を指定する。

### [3301]

正しいオブジェクト参照ではありません。

**要因**

引数 2 に不正なオブジェクト参照か，またはオブジェクト参照以外を指定した。

**対処**

正しいオブジェクト参照を指定する。

### [3302]

正しいオブジェクト参照ではありません。

**要因**

引数 3 に不正なオブジェクト参照か，またはオブジェクト参照以外を指定した。

**対処**

正しいオブジェクト参照を指定する。

### [3401]

正しいオブジェクト参照ではありません。

**要因**

不正なオブジェクト参照か，またはオブジェクト参照以外を指定した。

**対処**

正しいオブジェクト参照を指定する。

### [3402]

正しいクラス参照ではありません。

**要因**

不正なクラス参照か，またはクラス参照以外を指定した。

**対処**

正しいクラス参照を指定する。

### [4101]

指定したデータ項目の長さが正しくありません。

**要因**

指定したデータ項目の長さが正しくない。

**対処**

正しい長さを指定する。

### [4201]

オブジェクト参照が NULL です。

**要因**

String オブジェクトのオブジェクト参照を示すアドレスが NULL である。

**対処**

正しい String オブジェクトのオブジェクト参照を指定する。

### [4202]

正しいオブジェクト参照ではありません。

**要因**

不正なオブジェクト参照か，または String オブジェクトのオブジェクト参照以外を指定した。

**対処**

正しい String オブジェクトのオブジェクト参照を指定する。

### [4203]

指定したデータ項目の長さが正しくありません。

**要因**

指定したデータ項目の長さが正しくない。

**対処**

正しい長さを指定する。

[4301]

オブジェクト参照が NULL です。

**要因**

String オブジェクトのオブジェクト参照を示すアドレスが NULL である。

**対処**

正しいオブジェクト参照を指定する。

[4302]

正しいオブジェクト参照ではありません。

**要因**

不正なオブジェクト参照か，または String オブジェクトのオブジェクト参照以外を指定した。

**対処**

正しいオブジェクト参照を指定する。

[4401]

オブジェクト参照が NULL です。

**要因**

String オブジェクトのオブジェクト参照を示すアドレスが NULL である。

**対処**

正しいオブジェクト参照を指定する。

[4402]

正しいオブジェクト参照ではありません。

**要因**

不正なオブジェクト参照か，または String オブジェクトのオブジェクト参照以外を指定した。

**対処**

正しいオブジェクト参照を指定する。

[5101]

配列の型に指定した文字列が正しくありません。

**要因**

指定した文字列は，配列の型を示す文字'['から始まっていない。

**対処**

正しい配列の型を示す文字列を指定する。

### [5102]

配列の型に指定した文字列が正しくありません。

**要因**

指定した文字列は，配列の型を示す有効な文字列ではない。

**対処**

正しい配列の型を示す文字列を指定する。

### [5103]

指定した配列要素数が正しくありません。

**要因**

指定した配列要素数が正しくない。

**対処**

正しい配列要素数を指定する。

### [5201]

オブジェクト参照が NULL です。

**要因**

配列オブジェクトのオブジェクト参照を示すアドレスが NULL である。

**対処**

正しいオブジェクト参照を指定する。

### [5202]

正しいオブジェクト参照ではありません。

**要因**

不正なオブジェクト参照か，または配列オブジェクトのオブジェクト参照以外を指定した。

**対処**

正しいオブジェクト参照を指定する。

### [5301]

オブジェクト参照が NULL です。

**要因**

配列オブジェクトのオブジェクト参照を示すアドレスが NULL である。

**対処**

正しいオブジェクト参照を指定する。

### [5302]

正しいオブジェクト参照ではありません。

**要因**

不正なオブジェクト参照か，または配列オブジェクトのオブジェクト参照以外を指定した。

**対処**

正しいオブジェクト参照を指定する。

### [5303]

指定したインデクス値が正しくありません。

**要因**

指定したインデクス値が正しくない。

**対処**

正しいインデクス値を指定する。

### [5304]

正しいオブジェクト参照ではありません。

**要因**

不正なオブジェクト参照か，またはオブジェクト参照以外を指定した。

**対処**

正しいオブジェクト参照を指定する。

### [5401]

オブジェクト参照が NULL です。

**要因**

配列オブジェクトのオブジェクト参照を示すアドレスが NULL である。

**対処**

正しいオブジェクト参照を指定する。

[5402]

正しいオブジェクト参照ではありません。

**要因**

不正なオブジェクト参照か，または配列オブジェクトのオブジェクト参照以外を指定した。

**対処**

正しいオブジェクト参照を指定する。

[5403]

指定したインデクス値が正しくありません。

**要因**

指定したインデクス値が正しくない。

**対処**

正しいインデクス値を指定する。

[5501]

オブジェクト参照が NULL です。

**要因**

配列オブジェクトのオブジェクト参照を示すアドレスが NULL である。

**対処**

正しいオブジェクト参照を指定する。

[5502]

正しいオブジェクト参照ではありません。

**要因**

不正なオブジェクト参照か，または配列オブジェクトのオブジェクト参照以外を指定した。

**対処**

正しいオブジェクト参照を指定する。

[5503]

オブジェクト型の配列オブジェクトは指定できません。

**要因**

オブジェクト型の配列オブジェクトを指定した。

**対処**

- 基本型の配列オブジェクトを指定する。
- オブジェクト型の配列オブジェクトへアクセスする場合は，CBLJSETOBJARRAY サービスルーチンおよび CBLJGETOBJARRAY サービスルーチンを使用する。

## [5504]

データ型を示す文字列が正しくありません。

**要因**

指定したデータ型を示す文字列が正しくない。

**対処**

正しい配列の型を示す文字列を指定する。

## [5601]

オブジェクト参照が NULL です。

**要因**

配列オブジェクトのオブジェクト参照を示すアドレスが NULL である。

**対処**

正しいオブジェクト参照を指定する。

## [5602]

正しいオブジェクト参照ではありません。

**要因**

不正なオブジェクト参照か，または配列オブジェクトのオブジェクト参照以外を指定した。

**対処**

正しいオブジェクト参照を指定する。

## [5603]

オブジェクト型の配列オブジェクトは指定できません。

**要因**

オブジェクト型の配列オブジェクトを指定した。

**対処**

- 基本型の配列オブジェクトを指定する。

- オブジェクト型の配列オブジェクトへアクセスする場合は，CBLJSETOBJARRAY サービスルーチンおよび CBLJGETOBJARRAY サービスルーチンを使用する。

## [5604]

基本型配列のアドレスが NULL です。

**要因**

基本型配列のアドレスが NULL である。

**対処**

正しい基本型配列のアドレスを指定する。

## [5605]

データ型を示す文字列が正しくありません。

**要因**

指定した配列の型を示す文字列が正しくない。

**対処**

正しい配列の型を示す文字列を指定する。

# 付録 C　Java プログラム呼び出し機能サービスルーチンファイル

Java プログラム呼び出し機能では，次のサービスルーチンファイルを提供しています。

## 表 C-1　Java プログラム呼び出し機能で提供するサービスルーチンファイル

| ファイル名 | 登録内容 |
|---|---|
| COBOL2002Java プログラム呼び出し機能サービスルーチン.svw | Java プログラム呼び出し機能のサービスルーチン名 |

サービスルーチンファイルは，次のフォルダに格納されます。

```
COBOL2002のインストールフォルダ¥template
```

COBOL エディタのカスタマイズでこのサービスルーチンファイルを登録すると，Java プログラム呼び出し機能で提供しているサービスルーチンのキーワードを入力したとき，色分け表示や補完ができます。登録方法の詳細，色分け表示，およびキーワード補完については，マニュアル「COBOL2002　操作ガイド」を参照してください。

# 付録 D　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 D.1　関連マニュアル

このマニュアルは次のマニュアルと関連があります。必要に応じてお読みください。

- COBOL2002　ユーザーズガイド　（4010-1J-800）
- COBOL2002　操作ガイド　（4010-1J-801）
- COBOL2002　言語　標準仕様編　（4010-1J-804）
- COBOL2002　言語　拡張仕様編　（4010-1J-805）
- COBOL2002　Cosminexus 連携機能ガイド　（4010-1J-806）
- COBOL2002　メッセージ　（4010-1J-809）
- COBOL2002 Professional 製品 導入ガイド　（4010-1J-815）
- Cosminexus V9 アプリケーションサーバ　システム構築・運用ガイド　（3020-3-Y02）
- Cosminexus V9 アプリケーションサーバ　リファレンス　コマンド編　（3020-3-Y15）
- Cosminexus V9 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）（3020-3-Y16）
- Cosminexus V9 アプリケーションサーバ　アプリケーション開発ガイド　（3020-3-Y20）
- Cosminexus V11 アプリケーションサーバ　システム構築・運用ガイド　（3021-3-J02）
- Cosminexus V11 アプリケーションサーバ　リファレンス　コマンド編　（3021-3-J15）
- 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）（3021-3-J16）
- Cosminexus V11 アプリケーションサーバ　アプリケーション開発ガイド　（3021-3-J20）

なお，このマニュアルでは，Cosminexus Version 9，Cosminexus Version 11 を前提に記述しています。

## 付録 D.2　このマニュアルでの表記

このマニュアルでは，マイクロソフト製品の名称を次のように表記しています。

| マニュアルでの表記 | | 製品名 |
|---|---|---|
| Windows | Windows 10(x64) | Windows 10 Pro 日本語版(64 ビット版) |
| | | Windows 10 Enterprise 日本語版(64 ビット版) |
| | Windows 11 | Windows 11 Pro 日本語版 |
| | | Windows 11 Enterprise 日本語版 |

| マニュアルでの表記 | | 製品名 |
|---|---|---|
| | Windows Server 2019 | Windows Server 2019 Standard 日本語版 |
| | | Windows Server 2019 Datacenter 日本語版 |
| | Windows Server 2022 | Windows Server 2022 Standard 日本語版 |
| | | Windows Server 2022 Datacenter 日本語版 |

このマニュアルは，製品種別によって相違点があります。本文中での製品種別ごとの表記を次に示します。

| マニュアルでの表記 | | 該当する製品の形名 |
|---|---|---|
| COBOL2002 | 32bit 版 COBOL2002 | P-2636-2354<br>P-2436-5354<br>P-2636-3354<br>P-2436-6354<br>P-2636-4354<br>P-2636-7354 |
| | 64bit 版 COBOL2002 | P-2936-2354<br>P-2936-5354<br>P-2936-6354<br>P-2936-7354 |

- COBOL2002 Developer Professional と COBOL2002 Developer Professional(64)を総称して，COBOL2002 Developer Professional と表記しています。マニュアルでの表記と製品の形名を次に示します。

| マニュアルでの表記 | | 該当する製品の形名 |
|---|---|---|
| COBOL2002 Developer Professional | 32bit 版 COBOL2002 Developer Professional | P-2636-7354 |
| | 64bit 版 COBOL2002 Developer Professional | P-2936-7354 |

また，このマニュアルでは各製品を次のように表記しています。

| マニュアルでの表記 | | | 製品名 |
|---|---|---|---|
| Cosminexus | Cosminexus Version 9 または Cosminexus Version 11 | Cosminexus Application Server | uCosminexus Application Server |
| | | | uCosminexus Service Platform |
| | | Cosminexus Developer | uCosminexus Developer |
| | | | uCosminexus Service Architect |
| Oracle Java SE | | | Java Platform, Standard Edition Development Kit 8 |
| | | | Java Platform, Standard Edition Development Kit 11 |

このマニュアルで使用している表記と，対応する Java 関連用語を次に示します。

| マニュアルでの表記 | Java 関連用語 |
|---|---|
| EJB | Enterprise JavaBeans |
| JAR | Java Archive |
| Java VM | Java Virtual Machine |
| JDK | Java Development Kit |
| JIT | Just In Time |
| JNI | Java Native Interface |

## 付録 D.3　KB（キロバイト）などの単位表記について

1KB（キロバイト），1MB（メガバイト），1GB（ギガバイト），1TB（テラバイト）はそれぞれ 1,024 バイト，$1{,}024^2$ バイト，$1{,}024^3$ バイト，$1{,}024^4$ バイトです。

# 付録 E　用語解説

## (英字)

### BOM（バイトオーダマーク）

ファイルの先頭に付加された，Unicode の表現形式を表す情報。COBOL2002 では，テキスト編成ファイルに対してこの情報を付加します。本文中では，Unicode シグニチャと表記します。

### COBOL2002 Developer Professional

COBOL2002 Net Developer と COBOL2002 Professional Tool Kit を一つにしたプログラムプロダクトです。Windows の稼働環境に応じて，次の 2 製品があります。

- COBOL2002 Developer Professional（32bit 版）
- COBOL2002 Developer Professional(64)（64bit 版）

このマニュアルでは，上記の製品を総称して，COBOL2002 Developer Professional と表記します。

また，Java プログラム呼び出し機能のプログラム作成支援ツールを提供します。

### Cosminexus 連携機能

日立アプリケーションサーバ Cosminexus の Java アプリケーションから，COBOL プログラムを JavaBeans または EJB として呼び出せます。これによって，Web アプリケーションの基本機能となるビジネスロジック部分を COBOL で作成できます。

### getter

プロパティから値を取得するメソッドです。

### IVS

漢字を表す Unicode の直後に Variation Selector と呼ばれるコードを付加し，漢字の「異体字」を表現する方法のことです。

COBOL2002 では，用途が DISPLAY および NATIONAL の項目で使用します。

### Servlet

サーバ・サイド・プログラムとして稼働する Java プログラムです。

### setter

プロパティに値を設定するメソッドです。

### UCS-2（Universal multi-octet Character Set 2）

符号化文字集合の一つの形式です。1 文字を 2 バイトで表現します。

COBOL2002 では，用途が DISPLAY および NATIONAL の項目でサポートします。

### UCS-4（Universal multi-octet Character Set 4）

符号化文字集合の一つの形式です。1 文字を 4 バイトで表現します。

COBOL2002 では，用途が DISPLAY および NATIONAL の項目で UCS-4 の範囲をサポートします。

### Unicode

1 バイトでは表現できない文字セットを含めあらゆる文字セットをサポートした文字コードです。代表的な符号化文字集合として UCS-2，UCS-4 があります。代表的なエンコーディングスキーマとして UTF-8，UTF-16 があります。

COBOL2002 では，UCS-4 の範囲（UCS-2 の範囲を含む）をサポートします。

本文中では，符号化文字集合，およびエンコーディングスキーマを文字コードと表記します。

### UTF-16（16-bit UCS Transformation Format）

エンコーディングスキーマの一つの形式です。1 つのコード単位を 2 バイトとし，1 文字を 1 コード単位（2 バイト），または 2 コード単位（4 バイト）で表現します。UTF-16 では 2 バイトのコード単位の 1 バイト目を先に書くビッグエンディアン形式（UTF-16BE）と 1 バイト目を後に書くリトルエンディアン形式（UTF-16LE）があります。

COBOL2002 では，用途が NATIONAL の項目で UCS-4 の範囲（UCS-2 の範囲を含む）をサポートします。

### UTF-8（8-bit UCS Transformation Format）

エンコーディングスキーマの一つの形式です。ASCII 文字を 1 バイト，日本語文字を 3~8 バイト，半角かたかなを 3 バイトで表現します。

COBOL2002 では，用途が DISPLAY の項目で使用します。

### WRP（Windows Resource Protection）

Windows システムの安定性，および信頼性を向上させるための機能です。特定の OS ファイル，フォルダ，レジストリ キーなど，Windows の読み取り専用リソースを保護します。

## （ア行）

### アプリケーションサーバ

Web での業務開発のためのアプリケーション基盤機能を提供する製品です。日立アプリケーションサーバ Cosminexus は，ブラウザなどのフロントエンド層と DB や既存システムなどのバックエンド層の間に位置づけられ，業務の開発から運用までに一貫した環境を提供します。

## （サ行）

### サロゲート

UTF-16 の拡張で，2 つのコード単位（4 バイト）で 1 文字を表す機能です。この 2 つのコード単位の組み合わせのことをサロゲートペアと呼びます。

COBOL2002 では，用途が NATIONAL の項目で使用します。

## （タ行）

### 登録集原文

COBOL プログラム中でよく利用される標準化した手続き，ファイル記述，レコード記述，または完全な一つのプログラムを，コンパイルするプログラムとは別のファイルにしたデータのことです。

## （ハ行）

### バイトオーダ

2 バイト以上のデータの記録を行なう順序のことです。例えば，0x1234 のデータを 0x1234 のように最上位のバイトから順番に記録する方式をビッグエンディアン，0x3412 のように最下位のバイトから順番に記録する方式をリトルエンディアンといいます。2 バイトの UTF-16 は，バイトオーダを意識します。

## （マ行）

### 見た目幅

Unicode 機能の組み込み関数で使用します，文字の見た目の幅です。半角文字の幅は 1，全角文字の幅は 2 として扱います。

Unicode 機能の組み込み関数で，文字を見た目幅で数える場合に使用します。

# 索引

## J



## K



## S

## す



## せ

## た



## ち



## て



## と



## な



## は



## ひ



## ふ

## ほ



## み



## も



## ゆ



## り



## れ