HITACHI
Inspire the Next

uCosminexus DocumentBroker Version 3
クラスライブラリ　C++　解説

解説書

3000-3-F13-20

■対象製品

R-1M95D-43※ uCosminexus DocumentBroker Development Kit Version 3 03-60（適用 OS：AIX 5L V5.1，AIX 5L V5.2，AIX 5L V5.3）
R-1595D-43 uCosminexus DocumentBroker Development Kit Version 3 03-70（適用 OS：Windows Server 2003，Windows Server 2003 R2，Windows Server 2003 R2 x64 Edition，Windows Server 2008 x86，Windows Server 2008 R2，Windows Server 2012，Windows XP，Windows Vista，Windows 7，Windows 8）
R-1M95D-53※ uCosminexus DocumentBroker Runtime Version 3 03-60（適用 OS：AIX 5L V5.1，AIX 5L V5.2，AIX 5L V5.3）
R-1595D-53 uCosminexus DocumentBroker Runtime Version 3 03-70（適用 OS：Windows Server 2003，Windows Server 2003 R2，Windows Server 2003 R2 x64 Edition，Windows Server 2008 x86，Windows Server 2008 R2，Windows Server 2012，Windows XP，Windows Vista，Windows 7，Windows 8）
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。
※印の製品については，サポート時期をご確認ください。

■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

2013 年 9 月 3000-3-F13-20

■著作権

変更内容

変更内容（3000-3-F13-20）DocumentBroker Development Kit Version 3　03-70（**適用** OS：Windows Server 2003，Windows Server 2003 R2，Windows Server 2003 R2 x64 Edition，Windows Server 2008 x86，Windows Server 2008 R2，Windows Server 2012，Windows XP，Windows Vista，Windows 7，Windows 8），DocumentBroker Runtime Version 3　03-70（**適用** OS：Windows Server 2003，Windows Server 2003 R2，Windows Server 2003 R2 x64 Edition，Windows Server 2008 x86，Windows Server 2008 R2，Windows Server 2012，Windows XP，Windows Vista，Windows 7，Windows 8）

追加・変更内容	変更個所
前提 OS に Windows Server 2012 をサポートしました。	1.3，5.1.1

単なる誤字・脱字などはお断りなく訂正しました。

はじめに

このマニュアルは，次に示すプログラムプロダクトで提供するクラスライブラリの機能と使い方について説明したものです。

- R-1M95D-43　uCosminexus DocumentBroker Development Kit Version 3
- R-1595D-43　uCosminexus DocumentBroker Development Kit Version 3
- R-1M95D-53　uCosminexus DocumentBroker Runtime Version 3
- R-1595D-53　uCosminexus DocumentBroker Runtime Version 3

■対象読者

このマニュアルは，次の方を対象にしています。

- uCosminexus DocumentBroker Development Kit で提供するクラスライブラリを利用して，クライアントアプリケーションを開発する方
- uCosminexus DocumentBroker Development Kit で開発したクライアントアプリケーションを実行する方

なお，このマニュアルでは，上記の方が次の知識を持っていることを前提としています。

- UNIX または Windows に関する知識
- C++ 言語に関する知識
- XML に関する知識
- SQL 言語に関する知識
- 分散オブジェクト技術に関する知識

■マニュアルの構成

このマニュアルは，次に示す章と付録から構成されています。

第 1 章　クラスライブラリの概要

uCosminexus DocumentBroker での文書管理と，クラスライブラリで実現できる機能について説明しています。

第 2 章　オブジェクトの操作

クラス，オブジェクト，プロパティおよびメソッドの概要と，オブジェクトを操作するために必要な，文書空間への接続方法，プロパティの操作方法について説明しています。

第 3 章　クラスライブラリで実現する文書管理

クラスライブラリの機能で実現できる文書管理について説明しています。

第 4 章　オブジェクトの検索

クラスライブラリが提供する機能によってオブジェクトを検索する方法について説明しています。

第 5 章　環境設定

クラスライブラリを使用する際の環境設定，注意事項などについて説明しています。

第 6 章　エラー処理

メソッドを実行した時に発生したエラーの処理方法について説明しています。

付録 A　DMA オブジェクトの概要

DMA が規定するオブジェクトモデルの概要について説明しています。

付録 B　edmSQL の構文解析エラー情報

edmSQL の構文解析エラー情報として出力される内容と，その対処方法について説明しています。

付録 C　HiRDB のシステム共通定義のオペランド pd_max_access_tables の見積もり方法

HiRDB のシステム共通定義のオペランド pd_max_access_tables の見積もり方法について説明しています。

付録 D　用語解説

uCosminexus DocumentBroker で使用する用語について説明しています。

■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ので，必要に応じてお読みください。なお，本文に記載のマニュアル名称は，「uCosminexus DocumentBroker」を「DocumentBroker」と表記しています。

uCosminexus DocumentBroker **のマニュアル**

DocumentBroker Version 3　**システム導入・運用ガイド**（3000-3-D01）

uCosminexus DocumentBroker Version 3　**システム導入・運用ガイド**（3020-3-U71）

uCosminexus DocumentBroker を使用する環境を定義，管理および運用する場合に参照してください。
UNIX の場合は，資料番号が 3000-3-D01 のマニュアルを参照してください。
Windows の場合は，資料番号が 3020-3-U71 のマニュアルを参照してください。

uCosminexus DocumentBroker Version 3　**クラスライブラリ　C++　リファレンス　基本機能編**（3000-3-F14）

クラスライブラリのクラスの詳細とメソッドの文法について知りたい場合に参照してください。

uCosminexus DocumentBroker Version 3　**オブジェクト操作ツール**（3000-3-F15）

クラスライブラリの API をコマンド形式で実行できるオブジェクト操作ツールについて知りたい場合に参照してください。

uCosminexus DocumentBroker Version 3　**メッセージ**（3000-3-F12）

uCosminexus DocumentBroker が出力するメッセージについて知りたい場合に参照してください。

uCosminexus DocumentBroker Text Search Index Loader Version 3（3020-3-U72）

uCosminexus DocumentBroker に格納されている文書からテキスト情報を抽出して，全文検索用のインデクスファイルを作成し，これを全文検索インデクスに登録するユティリティについて知りたい場合に参照してください。

関連製品のマニュアル（HiRDB）

- **スケーラブルデータベースサーバ**　HiRDB Version 6　**システム定義**（UNIX(R) **用**）（3000-6-233）※1
- **スケーラブルデータベースサーバ**　HiRDB Version 6　**システム定義**（Windows(R) **用**）（3020-6-123）※1
- **スケーラブルデータベースサーバ**　HiRDB Version 7　**システム定義**（UNIX(R) **用**）（3000-6-273）※1
- **スケーラブルデータベースサーバ**　HiRDB Version 7　**システム定義**（Windows(R) **用**）（3020-6-273）※1
- **スケーラブルデータベースサーバ**　HiRDB Version 8　**システム定義**（UNIX(R) **用**）（3000-6-353）※1
- **スケーラブルデータベースサーバ**　HiRDB Version 8　**システム定義**（Windows(R) **用**）（3020-6-353）※1
- **スケーラブルデータベースサーバ**　HiRDB Version 6　**システム運用ガイド**（UNIX(R) **用**）（3000-6-234）※2
- **スケーラブルデータベースサーバ**　HiRDB Version 6　**システム運用ガイド**（Windows(R) **用**）（3020-6-124）※2
- **スケーラブルデータベースサーバ**　HiRDB Version 7　**システム運用ガイド**（UNIX(R) **用**）（3000-6-274）※2
- **スケーラブルデータベースサーバ**　HiRDB Version 7　**システム運用ガイド**（Windows(R) **用**）（3020-6-274）※2
- **スケーラブルデータベースサーバ**　HiRDB Version 8　**システム運用ガイド**（UNIX(R) **用**）（3000-6-354）※2
- **スケーラブルデータベースサーバ**　HiRDB Version 8　**システム運用ガイド**（Windows(R) **用**）（3020-6-354）※

2

- **スケーラブルデータベースサーバ** HiRDB Version 6 UAP **開発ガイド**（UNIX(R)/Windows(R) **用**）（3000-6-236）※3
- **スケーラブルデータベースサーバ** HiRDB Version 6 UAP **開発ガイド**（Windows(R) **用**）（3020-6-126）※3
- **スケーラブルデータベースサーバ** HiRDB Version 7 UAP **開発ガイド**（UNIX(R)/Windows(R) **用**）（3000-6-276）※3
- **スケーラブルデータベースサーバ** HiRDB Version 7 UAP **開発ガイド**（Windows(R) **用**）（3020-6-276）※3
- **スケーラブルデータベースサーバ** HiRDB Version 8 UAP **開発ガイド**（3020-6-356）※3
- **スケーラブルデータベースサーバ** HiRDB Version 6 SQL **リファレンス**（UNIX(R)/Windows(R) **用**）（3000-6-237）※4
- **スケーラブルデータベースサーバ** HiRDB Version 6 SQL **リファレンス**（Windows(R) **用**）（3020-6-127）※4
- **スケーラブルデータベースサーバ** HiRDB Version 7 SQL **リファレンス**（UNIX(R)/Windows(R) **用**）（3000-6-277）※4
- **スケーラブルデータベースサーバ** HiRDB Version 7 SQL **リファレンス**（Windows(R) **用**）（3020-6-277）※4
- **スケーラブルデータベースサーバ** HiRDB Version 8 SQL **リファレンス**（3020-6-357）※4
- HiRDB **全文検索プラグイン** HiRDB Text Search Plug-in Version 2（3000-6-245）（3020-6-140）※5
- HiRDB **全文検索プラグイン** HiRDB Text Search Plug-in Version 7（3000-6-288）※5
- HiRDB **全文検索プラグイン** HiRDB Text Search Plug-in Version 8（3020-6-375）※5
- HiRDB **デジタルコンテンツアクセスプラグイン** HiRDB File Link（UNIX(R) **用**）（3000-6-253）※6
- HiRDB **デジタルコンテンツアクセスプラグイン** HiRDB File Link（Windows(R) **用**）（3020-6-141）※6

注※1

このマニュアルでは，これらのマニュアルを「HiRDB　システム定義」と表記しています。

注※2

このマニュアルでは，これらのマニュアルを「HiRDB　システム運用ガイド」と表記しています。

注※3

このマニュアルでは，これらのマニュアルを「HiRDB　UAP 開発ガイド」と表記しています。

注※4

このマニュアルでは，これらのマニュアルを「HiRDB　SQL リファレンス」と表記しています。

注※5

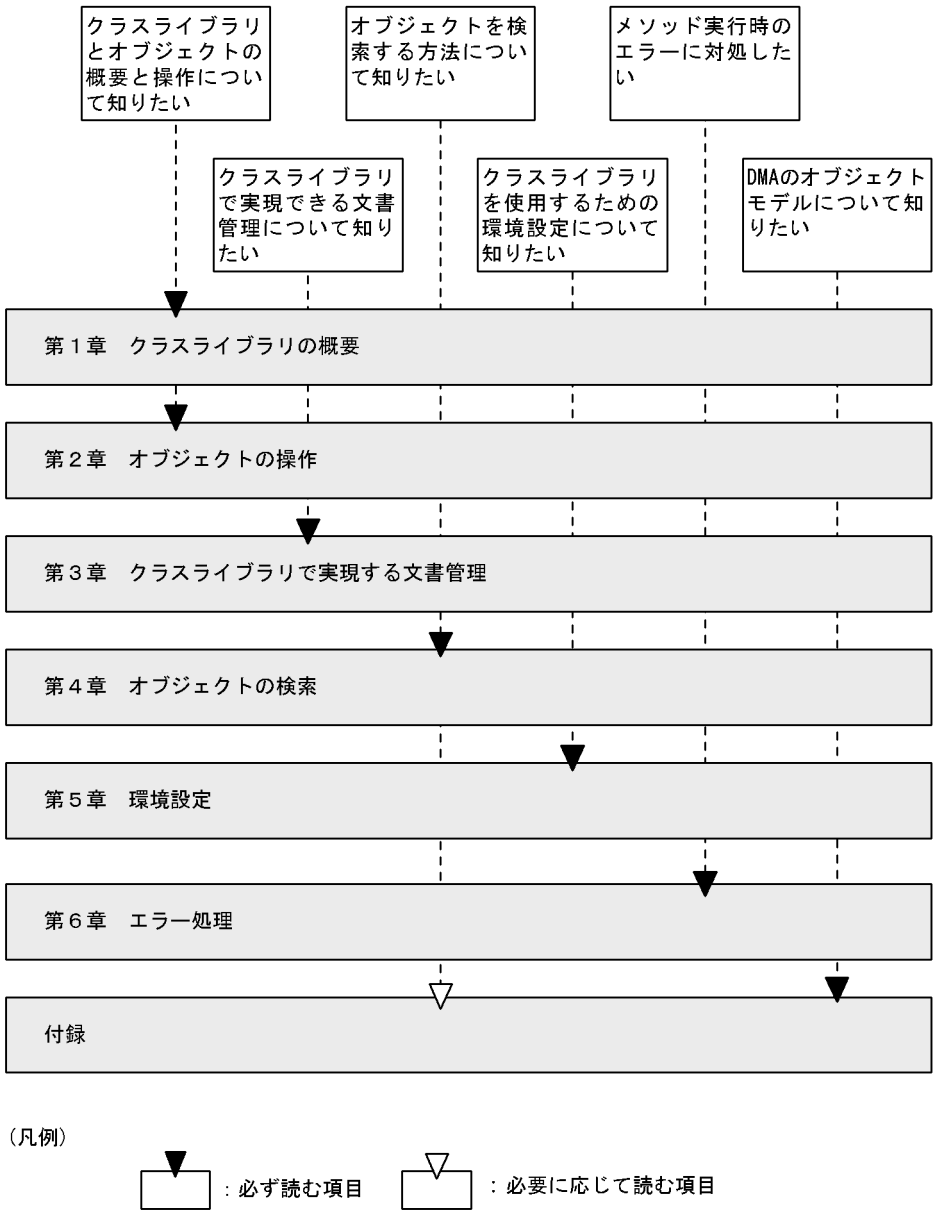
このマニュアルでは，これらのマニュアルを「HiRDB Text Search Plug-in」と表記しています。

注※6

このマニュアルでは，これらのマニュアルを「HiRDB File Link」と表記しています。

■読書手順

このマニュアルは，利用目的に合わせて章を選択してお読みいただけます。利用目的別の流れに従ってお読みいただくことをお勧めします。



■このマニュアルでの表記

このマニュアルでは，製品名称を次に示す略称で表記しています。

製品名称	略称
AIX 5L V5.1	AIX
AIX 5L V5.2	
AIX 5L V5.3	
uCosminexus DocumentBroker Development Kit Version 3	DocumentBroker
uCosminexus DocumentBroker Runtime Version 3	
uCosminexus DocumentBroker Server Version 3	
uCosminexus DocumentBroker Development Kit Version 3	DocumentBroker Development Kit

製品名称	略称
uCosminexus DocumentBroker Runtime Version 3	DocumentBroker Runtime
uCosminexus DocumentBroker Development Kit Version 3	DocumentBroker クライアント
uCosminexus DocumentBroker Runtime Version 3	
uCosminexus DocumentBroker Server Version 3	DocmentBroker Server または DocumentBroker サーバ
uCosminexus DocumentBroker Text Search Index Loader Version 3	DocumentBroker Text Search Index Loader
HiRDB/Single Server Version 6	HiRDB
HiRDB/Single Server Version 7	
HiRDB/Single Server Version 8	
HiRDB /Parallel Server Version 6	
HiRDB /Parallel Server Version 7	
HiRDB /Parallel Server Version 8	
HiRDB Text Search Plug-in Version 2	HiRDB Text Search Plug-in
HiRDB Text Search Plug-in Version 7	
HiRDB Text Search Plug-in Version 8	
HiRDB Text Search Plug-in Conceptual Extension Version 2	HiRDB Text Search Plug-in Conceptual Extension
HiRDB Text Search Plug-in Conceptual Extension Version 7	
HiRDB Adapter for XML - Enterprise Edition	HiRDB Adapter for XML
HiRDB Adapter for XML - Standard Edition	
IBM VisualAge C++ Professional for AIX V5	VisualAge C++ Professional for AIX
IBM VisualAge C++ Professional for AIX V6	
IBM XL C/C++ Enterprise Edition V7 for AIX	IBM XL C/C++ Enterprise Edition for AIX
IBM XL C/C++ Enterprise Edition V8 for AIX	
SANRISE Universal Storage Platform 100 SANRISE Universal Storage Platform 600 SANRISE Universal Storage Platform 1100 SANRISE Network Storage Controller NSC55	SANRISE USP
Sun Java(TM) System Directory Server 5.1	Sun Java System Directory Server
Sun Java(TM) System Directory Server 5.2	
Sun Java(TM) System Directory Server 6.3	
TPBroker Developer for C++	TPBroker V3
TPBroker for C++	
TPBroker	TPBroker V5
Microsoft(R) Active Directory(R)	Active Directory
Microsoft(R) Office Word	Word
Microsoft(R) Word	
Microsoft(R) Visual C++(R)	Visual C++
Microsoft(R) Windows Server(R) 2003, Enterprise Edition 日本語版	Windows Server 2003
Microsoft(R) Windows Server(R) 2003, Standard Edition 日本語版	
Microsoft(R) Windows Server(R) 2003 R2, Enterprise Edition 日本語版	Windows Server 2003 R2
Microsoft(R) Windows Server(R) 2003 R2, Standard Edition 日本語版	

製品名称	略称
Microsoft(R) Windows Server(R) 2003 R2, Enterprise x64 Edition 日本語版	Windows Server 2003 R2 または Windows Server 2003 R2 x64 Edition
Microsoft(R) Windows Server(R) 2003 R2, Standard x64 Edition 日本語版	
Microsoft(R) Windows Server(R) 2008 Enterprise 32-bit 日本語版	Windows Server 2008 または Windows Server 2008 x86
Microsoft(R) Windows Server(R) 2008 Standard 32-bit 日本語版	
Microsoft(R) Windows Server(R) 2008 R2 Enterprise 日本語版	Windows Server 2008 R2
Microsoft(R) Windows Server(R) 2008 R2 Standard 日本語版	
Microsoft(R) Windows Server(R) 2012 Standard 日本語版	Windows Server 2012
Microsoft(R) Windows Server(R) 2012 Datacenter 日本語版	
Microsoft(R) Windows(R) XP Professional Operating System	Windows XP
Microsoft(R) Windows Vista(R) Ultimate 日本語版	Windows Vista
Microsoft(R) Windows Vista(R) Business 日本語版	
Microsoft(R) Windows Vista(R) Enterprise 日本語版	
Microsoft(R) Windows(R) 7 Professional 日本語版 (32 ビット版)	Windows 7
Microsoft(R) Windows(R) 7 Enterprise 日本語版 (32 ビット版)	
Microsoft(R) Windows(R) 7 Ultimate 日本語版 (32 ビット版)	
Microsoft(R) Windows(R) 7 Professional 日本語版 (64 ビット版)	
Microsoft(R) Windows(R) 7 Enterprise 日本語版 (64 ビット版)	
Microsoft(R) Windows(R) 7 Ultimate 日本語版 (64 ビット版)	
Windows(R) 8 Pro 日本語版（32 ビット版）	Windows 8
Windows(R) 8 Pro 日本語版（64 ビット版）	
Windows(R) 8 Enterprise 日本語版（32 ビット版）	
Windows(R) 8 Enterprise 日本語版（64 ビット版）	

このほか，このマニュアルでは，次に示す表記方法を使用しています。

- DocumentBroker Development Kit の 01-21 以前のバージョンについては，Version 1 と表記します。
- AIX を UNIX と表記することがあります。
- TPBroker V3 および TPBroker V5 を合わせて TPBroker と表記することがあります。
- Windows Server 2003，Windows Server 2003 R2，Windows Server 2003 R2 x64 Edition，Windows Server 2008 x86，Windows Server 2008 R2，Windows Server 2012，Windows XP，Windows Vista，Windows 7，および Windows 8 を合わせて Windows と表記することがあります。

■ uCosminexus DocumentBroker のマニュアルで使用する略語

uCosminexus DocumentBroker のマニュアルで使用する英略語を次に示します。

英略語	英字での表記
ACE	Access Control Element
ACFlag	Access Control Flag
ACL	Access Control List

英略語	英字での表記
AIIM	Association for Information and Image Management International
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BES	Back End Server
BLOB	Binary Large Object
BMP	Bit Map
BNF	Backus Normal Form
BOA	Basic Object Adapter
CD-ROM	Compact Disc Read Only Memory
CGI	Common Gateway Interface
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
CR	Carriage Return
CSV	Comma Separated Value
DAP	Directory Access Protocol
DAT	Digital Audio Tape
DB	Database
DBMS	Database Management System
DCD	Document Content Description
DDE	Dynamic Data Exchange
DIT	Directory Information Tree
DLL	Dynamic Linking Library
DMA	Document Management Alliance
DN	Distinguished Name
DTD	Document Type Definition
EOF	End of File
ESIS-B	Element Structure Information Set-Binary Format
EUC	Extended UNIX Code
FAM	File Access Module
GIF	Graphics Interchange Format
GUI	Graphical User Interface
GUID	Globally Unique Identifier
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IANA	Internet Assigned Numbers Authority
ID	Identifier
IPF	Itanium(R) Processor Family
ISO	International Organization for Standardization

英略語	英字での表記
JIS	Japanese Industrial Standards
JPEG	Joint Photographic Expert Group
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
LF	Line Feed
MFC	Microsoft Foundation Class
MIME	Multipurpose Internet Mail Extensions
OCR	Optical Character Reader
OIID	Object Instance Identifier
OLE	Object Linking and Embedding
OMG	Object Management Group
ORB	Object Request Broker
ORDB	Object Relational Database
OS	Operating System
OTS	Object Transaction Service
PC	Personal Computer
PDF	Portable Document Format
RDB	Relational Database
RDN	Relative Distinguished Name
RFC	Request for Comment
RTF	Rich Text Format
SGML	Standard Generalized Markup Language
SQL	Structured Query Language
TCP/IP	Transmission Control Protocol/Internet Protocol
TIFF	Tag Image File Format
UCS-2	Universal Character Set coded in 2 octets
UCS-4	Universal Character Set coded in 4 octets
UOC	User Own Coding
URL	Uniform Resource Locator
UTC	Universal Time Coordinated
UTF-8	8-bit UCS Transformation Format
UTF-16	16-bit UCS Transformation Format
W3C	World Wide Web Consortium
WWW	World Wide Web
XML	Extensible Markup Language

■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
｜	横に並べられた複数の項目に対する項目間の区切りを示し，「または」を意味します。 （例） A ｜ B A または B を指定することを示します。
＿	括弧で囲まれた複数項目のうち 1 項目に対し使用され，括弧内のすべてを省略したときシステムが取る標準値を示します。 （例）〔A ｜ B〕 「何も指定しない」か「A または B を指定する」ことを示します。何も指定しない場合は A が仮定されます。
{ }	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示します。項目が横に並べられ，記号｜で区切られている場合は，そのうちの一つを選択します。 （例） {A ｜ B ｜ C} A，B または C のどれかを指定することを示します。
〔 〕	この記号で囲まれている項目は省略してもよいことを示します。複数の項目が横に並べて記述されている場合には，すべてを省略するか，記号 { } と同じくどれか一つを選択します。 （例 1）〔A〕 「何も指定しない」か「A を指定する」ことを示します。 （例 2）〔B ｜ C〕 「何も指定しない」か「B または C を指定する」ことを示します。
：：＝	この記号の左にあるものを右にあるもので定義することを示します。 （例） A：：＝ B 「A とは B である」と定義することを示します。
…	記述が省略されていることを示します。
< >	この記号で囲まれている項目は，該当する要素を指定することを示します。 （例） < プロパティ > プロパティを記述します。
( )	この記号で囲まれている項目を省略しないで記述することを示します。 （例） (< プロパティ >) 項目 < プロパティ > は省略しないで記述します。
...	この記号の直前に示す記号を繰り返し，複数個指定できることを示します。 （例） (< プロパティ >)... プロパティは一つ以上で複数個，繰り返して指定できます。

■このマニュアルで使用する構文要素

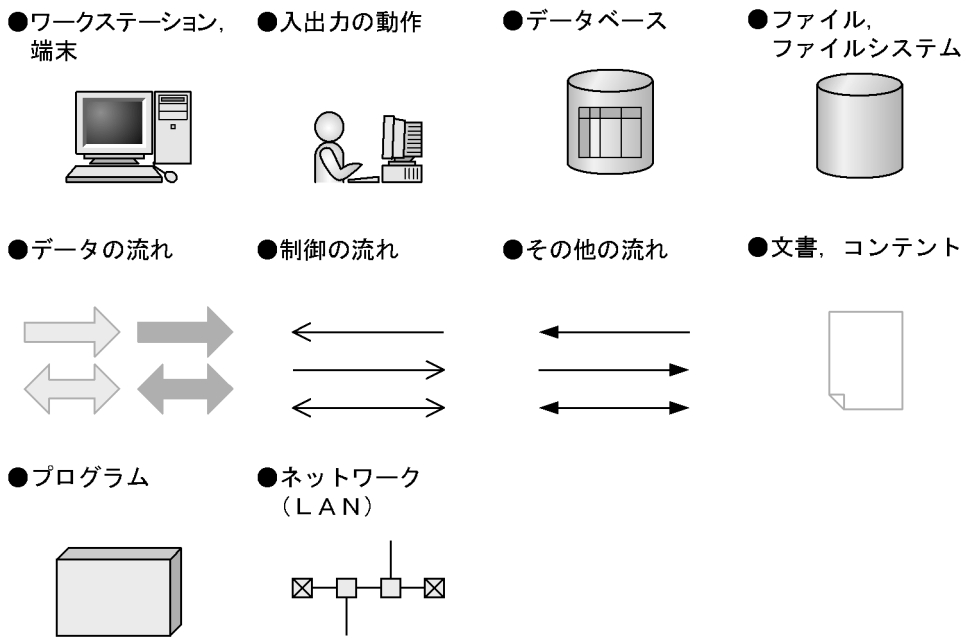
このマニュアルで使用する構文要素の種類を次に示します。

種類	定義
英字	A ～ Z　a ～ z
英小文字	a ～ z
英大文字	A ～ Z
数字	0 ～ 9
英数字	A ～ Z　a ～ z　0 ～ 9
記号	! " # $ % & ' ( ) + , _ ・ . / : ; < = > @ [ ] ^ ― { } ~ タブ 空白

注　すべて半角文字を使用してください。

■このマニュアルの図中で使用する記号

このマニュアルの図中で使用する記号を，次のように定義します。



■常用漢字以外の漢字の使用について

このマニュアルでは常用漢字を使用することを基本としていますが，次に示す用語については，常用漢字以外の漢字を使用しています。

個所（かしょ）

■ KB（キロバイト）などの単位表記について

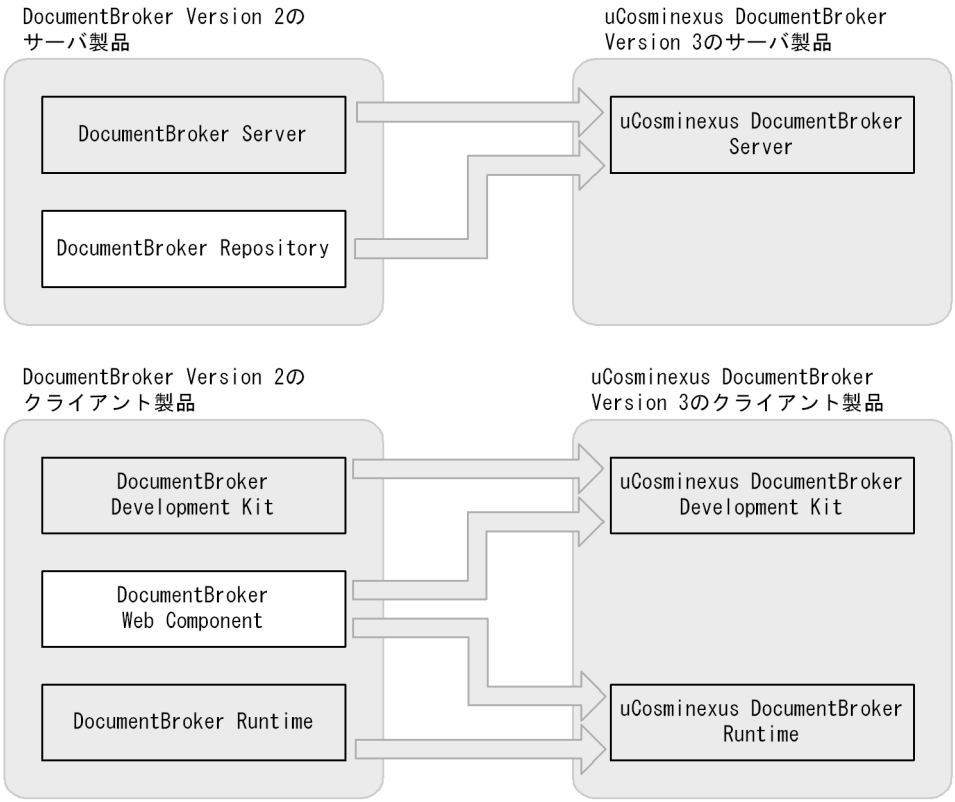
1KB（キロバイト），1MB（メガバイト），1GB（ギガバイト），1TB（テラバイト）はそれぞれ 1,024 バイト，$1{,}024^2$ バイト，$1{,}024^3$ バイト，$1{,}024^4$ バイトです。

■ DocumentBroker Version 2 と uCosminexus DocumentBroker Version 3 の製品体系の違い

uCosminexus DocumentBroker Version 3 では次のように製品の統合を行いました。

- DocumentBroker Repository を uCosminexus DocumentBroker Server に統合しました。
- DocumentBroker Web Component を uCosminexus DocumentBroker Development Kit および uCosminexus DocumentBroker Runtime に統合しました。

DocumentBroker Version 2 と uCosminexus DocumentBroker Version 3 の製品体系の違いを次に示します。



■ DocumentBroker Version 2 と uCosminexus DocumentBroker Version 3 のマニュアルの対応

バージョンアップおよび製品体系の変更に伴い，uCosminexus DocumentBroker Version 3 では次に示すようにマニュアル名称を変更しました。

Version 2 のマニュアル名称	Version 3 のマニュアル名称
DocumentBroker Version 2　システム導入・運用ガイド	uCosminexus DocumentBroker Version 3　システム導入・運用ガイド
DocumentBroker Version 2　クラスライブラリ　解説	uCosminexus DocumentBroker Version 3　クラスライブラリ　C++　解説
DocumentBroker Version 2　クラスライブラリ　リファレンス　基本機能編	uCosminexus DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編
DocumentBroker Version 2　クラスライブラリ　リファレンス　概念 SGML 文書管理機能編	廃版
DocumentBroker Version 2　オブジェクト操作ツール	uCosminexus DocumentBroker Version 3　オブジェクト操作ツール
DocumentBroker Version 2　統計解析ツール	uCosminexus DocumentBroker Version 3　統計解析ツール
DocumentBroker Version 2　メッセージ	uCosminexus DocumentBroker Version 3　メッセージ
DocumentBroker Web Component Version 2　解説	uCosminexus DocumentBroker Version 3　クラスライブラリ　Java　解説
DocumentBroker Web Component Version 2　リファレンス	uCosminexus DocumentBroker Version 3　クラスライブラリ　Java　リファレンス

Version 2 のマニュアル名称	Version 3 のマニュアル名称
DocumentBroker Web Component Version 2　サンプル Web アプリケーション	uCosminexus DocumentBroker Version 3　クラスライブラリ　Java　サンプル Web アプリケーション
DocumentBroker Text Search Index Loader Version 2	uCosminexus DocumentBroker Text Search Index Loader Version 3
DocumentBroker Rendering Option　システム導入・運用ガイド	uCosminexus DocumentBroker Rendering Option Version 3
DocumentBroker Object Loader Version 2	uCosminexus DocumentBroker Object Loader Version 3

目次

1 クラスライブラリの概要

DocumentBroker では，DMA オブジェクトを操作しやすくするために，クラスライブラリを提供しています。
この章では，DocumentBroker Development Kit で提供するクラスライブラリの概要および機能について説明します。

1.1 DocumentBroker での文書管理

この節では，DocumentBroker で文書管理を実現する方法の概要について説明します。

1.1.1 文書とオブジェクト

DocumentBroker では，文書および文書を管理する仕組みを**オブジェクト**の概念で表現します。例えば，文書として表現するためのオブジェクト，文書をまとめるフォルダの役目をするオブジェクト，文書のバージョンを管理するためのオブジェクト，検索に必要なオブジェクトなどがあります。これらのオブジェクトは，DMA が提唱するオブジェクトモデルを基にしています。

オブジェクトは，データとデータを操作するメソッドを抽象化し，クライアントアプリケーションの開発と保守を容易にします。オブジェクトに対する操作をクライアントアプリケーション上に設計することで，文書の管理や検索ができます。

このマニュアルでは，DMA のオブジェクトモデルに基づいたオブジェクトを，**DMA オブジェクト**といいます。

1.1.2 DocumentBroker のシステム構成

ここでは，DocumentBroker のシステム構成について説明します。

(1) DocumentBroker サーバ

DocumentBroker サーバは，プロセスやセッションの管理および各種のユティリティなどのサーバ機能を備えています。DocumentBroker サーバは，**文書空間**というメモリ空間を管理するサーバです。
DocumentBroker クライアントは，文書空間に接続して，データベースに格納されているオブジェクトを操作します。

(2) DocumentBroker クライアント

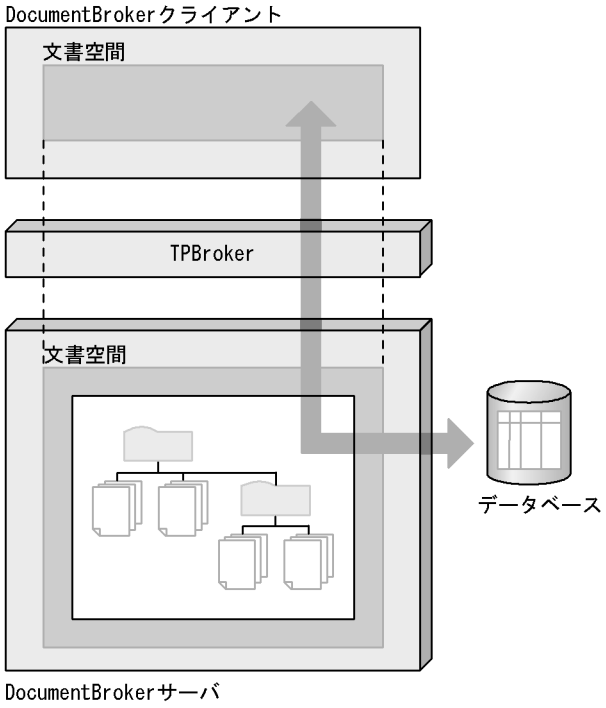
DocumentBroker クライアントでは，業務に対応したクライアントアプリケーションを構築し，それを実行することで DocumentBroker サーバが提供する機能を利用します。

- **開発環境クライアント**
  業務に対応したクライアントアプリケーションを開発するためのクライアントです。開発環境クライアントには，実行環境クライアントの機能も含まれています。
- **実行環境クライアント**
  クライアントアプリケーションを実行するクライアントです。業務に合わせて設計，開発したクライアントアプリケーションを使用して，DocumentBroker サーバの文書空間上で，オブジェクトを操作します。
  なお，このマニュアルでは，開発環境クライアントの機能を提供する DocumentBroker Development Kit の機能について説明しています。

(3) DocumentBroker での文書管理の概要

DocumentBroker での文書管理の概要について，図 1-1 で説明します。

図 1-1 DocumentBroker での文書管理の概要



DocumentBroker クライアントは，TPBroker を介して，データベースに格納されているオブジェクトを DocumentBroker サーバの文書空間上で操作します。

DocumentBroker では，オブジェクトを操作することによって，文書の参照や更新などの操作を実現しています。

1.2 クラスライブラリの特長

ここでは，クラスライブラリの特長と機能について説明します。

DocumentBroker では，DMA のオブジェクトモデルに基づいたオブジェクトを使用して，文書を管理します。DocumentBroker では，DMA のオブジェクトモデルに基づいたオブジェクト（DMA **オブジェクト**）を組み合わせて使用することによって，高度な文書管理機能を実現しています。

DMA オブジェクトを一つ一つ意識してクライアントアプリケーションを設計するためには複雑なプログラミングが必要です。DocumentBroker Development Kit では，より単純なプログラミングによってクライアントアプリケーションを設計できるように，クラスライブラリを提供しています。クラスライブラリには，次のような特長があります。

● **複数の** DMA **オブジェクトをまとめて，一つのオブジェクトとして操作できます。**
例えば，DMA オブジェクトには文書を表現するオブジェクト，文書の表現形式を管理するオブジェクト，文書のコンテント（内容）に相当するオブジェクトなどがあります。一つの文書を操作するためには，これらの DMA オブジェクトをすべて操作する必要があります。
クラスライブラリでは，「文書」「フォルダ」など，文書管理に使用するコンポーネントに必要な DMA オブジェクトを，まとまった一つのオブジェクトとして扱うことができます。このため，文書を表す一つのオブジェクトを操作することで，文書を表現するオブジェクト，文書の表現形式を管理するオブジェクト，文書のコンテントに相当するオブジェクトなどを含んだ，一つの文書が操作できます。

● **文書管理に必要な操作が，メソッドとして提供されています。**
DMA オブジェクトは，プロパティの値を設定，変更することで操作します。文書管理の操作を実現するためには，文書を構成する複数の DMA オブジェクトに対して，それぞれ適切なプロパティを設定する必要があります。
クラスライブラリでは，文書管理に必要な操作をメソッドとして提供しています。クラスライブラリのメソッドをコールすることによって，その操作に必要なプロパティが DMA オブジェクト上に設定されます。このため，それぞれの DMA オブジェクトのプロパティをすべて意識して操作する必要はありません。

クラスライブラリのオブジェクトは，ひな形となるクラスライブラリの**クラス**を基に作成します。このクラスは，実現する機能ごとに，複数の DMA のクラスから構成されています。クラスライブラリのクラスを基にオブジェクトを作成することによって，その機能を実現するのに必要な複数の DMA オブジェクトが作成されます。

クラスライブラリのオブジェクトと DMA オブジェクトの関係について，図 1-2 に示します。

図 1-2 クラスライブラリのオブジェクトと DMA **オブジェクトの関係**

1.3 クラスライブラリで実現する機能

この節では，DocumentBroker Development Kit で提供するクラスライブラリによって実現する，文書管理機能について説明します。

なお，クラスライブラリが提供する機能のうち，次の場合は現在サポートしていない機能があります。

● TPBroker V5 を使用する場合
  • File Link 連携機能

● Windows Server 2003 R2 x64 Edition を使用する場合
  • File Link 連携機能

● Windows Server 2008，Windows Server 2008 R2，Windows Server 2012，Windows XP，Windows Vista，Windows 7，および Windows 8 を使用する場合
  • XML 文書管理機能
  • File Link 連携機能

1.3.1 文書空間（データベース）に接続する機能

DocumentBroker のクライアントの実行環境と DocumentBroker サーバの文書空間を接続する機能は，クラスライブラリのクラスの機能として提供されています。

文書空間との接続には，CdbrSession クラスの機能を使用します。

クラスライブラリの機能によって文書空間のオブジェクトを操作すると，その時点でデータベースの内容も更新されます。この更新を確定またはキャンセルする機能も，CdbrSession クラスによって提供されています。

1.3.2 文書を作成，管理する機能

DocumentBroker によって管理する文書には，次の種類があります。

● 一つのファイルから構成される文書

● 一つの文書の中に複数のバージョンに対応するファイルを持つ文書

● 一つの文書の中に複数の形式に対応するファイルを持つ文書

● 一つの文書の中に複数のファイルを持つ文書

● ファイルの実体を DocumentBroker サーバが存在するマシンから接続可能なファイルシステムに持つ文書

● ファイルの実体をデータベースと連携したファイルサーバ上に持つ文書

文書を管理する方法には，次の種類があります。

● 文書と文書を関連づけて管理する方法

● 文書とコンテナを関連づけて管理する方法

● 文書と構成管理コンテナを関連づけて構成管理する方法

また，次に示す，論理構造を持つ文書を管理することもできます。

● XML 文書
　これは，文書を構成するファイルが XML 形式である文書です。

なお，DocumentBroker の文書空間で使用する文字コード種別が UTF-8 の場合，DocumentBroker で使用できる機能に制限があります。文書空間で使用する文字コード種別が UTF-8 の場合に使用できる機能の詳細は，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。

(1) 一つのファイルから構成される文書を管理する

最も基本的な文書は，一つのファイルから構成される文書です。

例えば，Word で作成した文書「報告書 .doc」を，DocumentBroker の文書として管理できます。

DocumentBroker では，文書にさまざまな属性を付けて管理できます。この属性をプロパティといいます。プロパティには，DocumentBroker によって設定されるプロパティと，ユーザが任意に追加するプロパティがあります。例えば，「報告書 .doc」を DocumentBroker で管理する時に，ユーザが追加したプロパティの値として，「作成者」や「担当部署」，「コメント」などの情報も一緒に管理できます。

文書を作成する基になるクラスは，次のとおりです。

- CdbrDocument クラス
  バージョン管理をしない文書の基になるクラスです。
- CdbrVersionableDocument クラス
  バージョン管理をする文書の基になるクラスです。

バージョン管理機能については，「(2) 一つの文書の中に複数のバージョンに対応するファイルを持つ文書を管理する」で説明します。

(2) 一つの文書の中に複数のバージョンに対応するファイルを持つ文書を管理する

文書を更新するときに，履歴を残して管理することができます。文書に対して，第 1 版，第 2 版などのバージョンを付けておくことで，過去のある時点での文書を取り出したり，その文書を基に新たな文書を作成したりできます。また，その文書が幾つのバージョンを持っているかも管理できます。

この機能を，**バージョン管理機能**といいます。

この機能を使用できる文書の基になるクラスは，次のとおりです。

- CdbrVersionableDocument クラス

(3) 一つの文書の中に複数の形式に対応するファイルを持つ文書を管理する

一つの文書として，複数の形式に対応するファイルを管理できます。

この機能は，一つの文書の内容を，対応するアプリケーションごとの複数の形式に変換した場合などに，使用できます。

例えば，文書「報告書」に，Word で作成したファイル「報告書 .doc」のほかに，「報告書 .doc」を PDF 形式に変換した「報告書 .pdf」も一緒に登録できます。これによって，使用するユーザによって，Word 形式の文書をダウンロードしたり，PDF 形式の文書をダウンロードしたりできます。

この機能を，**マルチレンディション機能**といいます。

この機能を使用できる文書を作成する基になるクラスは，次のとおりです。

- CdbrDocument クラス
- CdbrVersionableDocument クラス

(4) 一つの文書の中に複数のファイルを持つ文書を管理する

一つの文書として，複数のファイルを管理できます。一つの文書で複数のファイルを管理することで，文書を構成する複数のファイルを一括して登録したり，一括して取得したりできます。

この機能を，**マルチファイル管理機能**といいます。

この機能を使用できる文書を作成する基になるクラスは，次のとおりです。

- CdbrDocument クラス
- CdbrVersionableDocument クラス

(5) ファイルの実体を DocumentBroker サーバが存在するマシンから接続可能なファイルシステムに持つ文書を管理する

文書の実体であるコンテントをファイルシステムの任意のディレクトリに格納し，文書のプロパティとコンテントの格納先の情報（**コンテントロケーション**）をデータベースに登録して管理します。コンテントロケーションには，コンテントの格納先の基点となるディレクトリパス（**コンテント格納先ベースパス**）からの相対パスが登録されます。このため，コンテントの格納先を移行する場合などは，データベースに影響を与えることなく，コンテント格納先ベースパスを変更するだけで，コンテントの格納領域を変更できます。また，コンテントをデータベースに格納しないため，データベースの容量を削減できます。

この機能を，**リファレンスファイル管理機能**といいます。

この機能を使用できる文書を作成する基になるクラスは，次のとおりです。

- CdbrDocument クラス
- CdbrVersionableDocument クラス

(6) ファイルの実体をデータベースと連携したファイルサーバ上に持つ文書を管理する

音声・画像などの容量が大きいコンテンツを管理する場合，ファイルの実体はファイルサーバ上で管理したいことがあります。このときに，データベースではファイル名や作成者名などの属性情報だけを管理して，ファイルの実体をファイルサーバ上で管理するためには，データベースとファイルサーバ間で整合性を確保することが必要です。

File Link **連携機能**を使用すると，データベースとファイルサーバの内容の整合性を HiRDB File Link の機能によって確保できます。これによって，ユーザアプリケーションで整合性を取るための処理をする必要がなくなります。

なお，この機能を使用する場合には，関連プログラムが必要です。

この機能を使用できる文書を作成する基になるクラスは，次のとおりです。

- CdbrDocument クラス
- CdbrVersionableDocument クラス

(7) 文書と文書を関連づけて管理する

文書と文書を関連づけて管理できます。ある文書の中で，ほかの文書を参照する記述がある場合などに，文書間に関連づけを設定しておくことで，参照元の文書から参照先の文書をたどることができます。また，参照先の文書から，参照元の文書をたどることもできます。

この機能を持つ文書を作成する基になるクラスは，次のとおりです。

- CdbrDocument クラス
- CdbrVersionableDocument クラス

(8) 文書をコンテナに関連づけて管理する

文書は，フォルダに格納するイメージで，まとめて管理できます。また，フォルダに格納した文書を，別の観点から分類して管理することもできます。

DocumentBroker では，複数の文書をまとめて管理するために，コンテナというオブジェクトを使用します。コンテナと文書を関連づけることによって，文書をフォルダに格納するようにして管理したり，文書を複数の観点から分類して管理したりできます。設定する関連づけの種類によって，コンテナと文書は 1:n（n は 1 以上の整数）の関係で関連づけたり，m:n（m，n は 1 以上の整数）の関係で関連づけたりできます。

例えば，文書をプロジェクト単位でまとめて管理したい場合，プロジェクトごとにコンテナを作成して，コンテナと文書を 1:n の関係で関連づければ，プロジェクトごとのフォルダに文書を格納するイメージで文書を管理できます。また，フォルダに格納された文書を作成者とテーマで分類して管理したい場合，作成者とテーマを表すコンテナをそれぞれ作成して，コンテナと文書を m:n の関係で関連づければ，文書を作成者やテーマごとに分類して管理できます。

コンテナとコンテナを関連づけることによって，フォルダや分類に階層を持たせることもできます。

コンテナの基になるクラスは，次のとおりです。

- CdbrReferentialContainer クラス
  構成管理機能を持たないコンテナです。コンテナ自身のバージョン管理もできません。
- CdbrVersionTraceableContainer クラス
  構成管理機能を持つコンテナです。ただし，コンテナ自身のバージョン管理はできません。
- CdbrConfiguratedReferentialContainer クラス
  構成管理およびコンテナ自身のバージョン管理ができるコンテナです。

(9) 文書を構成管理する

コンテナでは，文書の特定のバージョンを管理することもできます。文書やコンテナのバージョンを指定して管理するには，構成管理機能を持つコンテナ（構成管理コンテナ）を使用します。**構成管理機能**とは，管理している文書やコンテナの，ある時点でのバージョン構成の状態を管理する機能です。例えば，管理する複数の文書が，それぞれ異なるタイミングでバージョンアップする場合などに，構成管理機能を使用します。

構成管理機能を持つコンテナでは，それぞれの文書の最新のバージョンを参照できるようにしたり，文書のバージョンアップに関係なく常にバージョン 1 やバージョン 2 など特定のバージョンの文書を参照できるようにしたりできます。ある文書は常に最新のバージョンを参照して，別の文書は特定のバージョンを参照し続ける，ということもできます。

また，このコンテナをバージョンアップすることで，コンテナごとに関連づけている文書やコンテナのバージョン構成をまとめて管理できます。例えば，バージョン 1 のコンテナからたどる文書をバージョン 1 で固定して，バージョン 2 のコンテナで管理する文書は常に最新のバージョンをたどるようにしたりできます。

構成管理機能を持つコンテナを作成する基になるクラスは，次のとおりです。

- CdbrVersionTraceableContainer クラス
- CdbrConfiguratedReferentialContainer クラス

このうち，構成管理機能を持つコンテナ自身のバージョンを管理する機能を持つコンテナは，

CdbrConfiguratedReferentialContainer クラスを基に作成します。

(10) XML 文書を管理する

XML 文書とは，XML 形式のファイル（XML ファイル）をコンテントとして登録した文書です。XML ファイルは，タグによって定義された論理構造を持っています。DocumentBroker では，任意のタグに囲まれた文字列を文書のプロパティの値として割り当てて管理したり，特定の構造に含まれる文字列を指定した検索を実行したりできます。

このような機能を，**XML 文書管理機能**といいます。なお，この機能を使用する場合には，関連プログラムが必要です。

XML 文書の基になるクラスは，次のとおりです。

- CdbrDocument クラス
- CdbrVersionableDocument クラス

1.3.3 文書やコンテナを検索する機能

すでに登録されている文書やコンテナは，検索によって取得します。

DocumentBroker では次に示す検索が実行できます。

- 文書やコンテナに設定されているプロパティの値を利用した属性検索または全文検索
- 文書に含まれる文字列を指定した全文検索

文書に対する全文検索では，文章を指定して，その文章と類似した概念を持つ文書を検索（概念検索）することもできます。ただし，全文検索を実行する場合には，前提プログラムが必要です。

検索を実行する場合の検索条件は，edmSQL によって指定できます。edmSQL は SQL に基づいた構文で記述できるため，SQL に関する知識を活用して検索を実行できます。

edmSQL を使用する検索機能は CdbrEqlStatement クラスによって提供されています。

1.3.4 独立データを管理する機能

既存の DMA オブジェクトによって作成する文書やコンテナ以外のオブジェクトを，独立データとして作成，管理できます。

独立データを作成する基になるクラスは，次のとおりです。

- CdbrIndependentPersistence クラス

1.3.5 レンディションのコンテント種別変換機能

文書の実体であるコンテントの格納先を，文書の利用価値やアクセス頻度に応じ，最適な格納先に変換するための機能です。

この機能を使用できる文書を作成する基になるクラスは次のとおりです。

- CdbrDocument クラス
- CdbrVersionableDocument クラス

1.3.6 そのほかの機能

そのほかの機能について説明します。

(1) 排他制御機能

DocumentBroker で管理している文書やコンテナにアクセスする場合，排他制御が実行されます。排他制御とは，あるユーザが文書やコンテナにアクセスしているときに，ほかのユーザによってその文書やコンテナの状態が変更されたり，削除されたりしないようにロックを掛けて制御する機能です。

DocumentBroker の排他制御には，メソッドによって暗黙的に設定されるロックと，ユーザが明示的に設定するロックがあります。

(2) アクセス制御機能

DocumentBroker で管理している文書やコンテナに対して参照や更新などのアクセスをする場合，そのユーザが持つアクセス権に応じてアクセスを制御することができます。例えば，作成した文書をほかのユーザに参照はさせたいが更新はさせたくない場合などに，この機能を使用します。これを，アクセス制御機能といいます。アクセスを制御するためのアクセス権は，ユーザやグループごとに設定できます。

2 オブジェクトの操作

DocumentBroker では，オブジェクトを操作することによって，文書管理を実現します。クラスライブラリでオブジェクトを操作するためには，クラス，オブジェクト，プロパティおよびメソッドについて理解しておく必要があります。
この章では，クラス，オブジェクト，プロパティおよびメソッドの概要と，オブジェクトを操作するために必要な，文書空間への接続方法，プロパティの操作方法について説明します。

2.1 クラス，オブジェクト，プロパティおよびメソッドの概要

この節では，クラスライブラリで操作するオブジェクトの概要について説明します。説明する内容は，次のとおりです。

- クラス
- オブジェクト
- プロパティ
- メソッド

2.1.1 クラス

ここでは，クラスライブラリで扱うクラスの概要について説明します。

クラスとは，オブジェクトを作成するためのひな形です。

クラスライブラリを使用する場合，次の 2 種類のクラスについて理解する必要があります。

● DMA クラス

● クラスライブラリのクラス

なお，クラスライブラリのクラス名には，クラス名の先頭に「Cdbr」が付きます。DMA クラスのクラス名には，クラス名の先頭に「dmaClass_」または「edmClass_」が付きます。

(1) DMA クラス

DMA クラスは，DMA オブジェクトのひな形になるクラスです。

DMA では，文書管理に必要なオブジェクトを作成するためのクラスを規定しています。例えば，個々の文書に相当するオブジェクト，文書の形式を表すオブジェクト，文書の実体を管理するオブジェクトなど，DMA の文書管理インターフェースに必要なオブジェクトは，DMA クラスを基に作成されます。また，DMA が規定したクラスに加えて，DocumentBroker では，拡張した機能を持つ DMA クラスを提供しています。例えば，構成管理機能を持つ DMA オブジェクトを管理する機能を持つ DMA オブジェクトは，DocumentBroker が拡張した DMA クラスを基に作成されます。

なお，DMA クラスは，データベースの表に対応します。したがって，DMA オブジェクトは，データベース上に存在するオブジェクトになります。DMA オブジェクトとデータベースの関係については，「2.7 オブジェクトとデータベースの関係」を参照してください。

(2) クラスライブラリのクラス

クラスライブラリのクラスは，DMA オブジェクトをまとめて操作するための機能を持ったクラスです。文書を管理するための機能ごとに，必要な DMA オブジェクトを操作する機能を持ったクラスが定義されています。

また，文書やコンテナを作成，操作するためのクラスは，文書やコンテナに必要な複数の DMA クラスを構成要素として持ち，これらをまとめて作成するためのひな形になります。

例えば，「バージョン管理する文書」を作成，操作するためのクラスライブラリのクラスである CdbrVersionableDocument クラスは，次のような DMA クラスを構成要素として持っています。

- dmaClass_ConfigurationHistory クラス

文書のバージョンを統括する DMA クラス
- dmaClass_VersionSeries クラス
  バージョン構成を管理する DMA クラス
- dmaClass_VersionDescription クラス
  バージョン構成と管理する文書を結び付ける DMA クラス
- dmaClass_DocVersion クラス
  文書の特定のバージョンに相当する DMA クラス
- dmaClass_Rendition クラス
  文書を構成するファイルの種類を管理する DMA クラス
- dmaClass_ContentTransfer クラス
  文書の実体であるコンテントを管理する DMA クラス

クラスライブラリのオブジェクトとして CdbrVersionableDocument オブジェクトを作成すると，これらの DMA クラスを基にした DMA オブジェクトがまとめて作成されます。そして，作成した DMA オブジェクトを操作する機能も，CdbrVersionableDocument クラスのメソッドとして実装されています。

つまり，クラスライブラリのクラスは，クラスライブラリで操作するオブジェクトを定義したひな形であり，同時に複数の DMA オブジェクトをまとめて操作するためのインターフェースを提供しています。

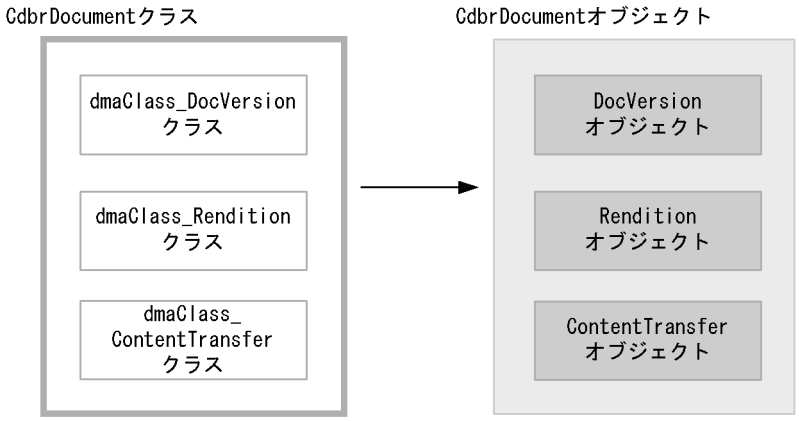
2.1.2 オブジェクト

ひな形であるクラスに，実際の値を格納したものがオブジェクトです。

クラスライブラリのオブジェクトは，各クラスの CreateObject メソッドをコールして作成します。このメソッドによって作成されるクラスライブラリのオブジェクトは，実際には複数の DMA オブジェクトから構成されています。

クラスライブラリのクラスとオブジェクトの関係を，次の図に示します。

図 2-1 クラスライブラリのクラスとオブジェクトの関係



これらのオブジェクトは，クラスライブラリのメソッドによって操作します。

例えば，図 2-1 に示したように，CdbrDocument::CreateObject メソッドをコールして文書を表す CdbrDocument オブジェクトを作成した場合，データベースには DMA オブジェクトの DocVersion オブジェクト，Rendition オブジェクトおよび ContentTransfer オブジェクトが作成されます。しかし，CdbrDocument オブジェクトを操作するときには，CdbrDocument クラスのメソッドをコールすればよく，個々の DMA オブジェクトを意識する必要はありません。

なお，クラスライブラリのオブジェクトを構成する複数の DMA オブジェクトのうち，最上位に当たるオブジェクトのことを，**トップオブジェクト**といいます。例えば，CdbrDocument オブジェクトの場合は，DMA オブジェクトの DocVersion オブジェクトがトップオブジェクトです。

2.1.3 プロパティ

クラスには，プロパティが定義されています。プロパティとは，オブジェクトの状態を表す属性です。プロパティは，オブジェクトを検索したり，オブジェクトに関するデータを格納したりするために使用できます。

DMA クラスには，DMA によって規定されたプロパティが定義されています。DocumentBroker で拡張した DMA クラスには，DocumentBroker 独自のプロパティも定義されています。また，DMA クラスに対してユーザが必要なプロパティを追加することもできます。さらに，クラスライブラリでは，クラスライブラリのクラスごとに定義された独自のプロパティを使用することもできます。

なお，プロパティには継承関係があります。上位のクラスで定義されたプロパティは，そのサブクラスに継承されます。

2.1.4 メソッド

クラスライブラリのメソッドは，クラスごとに提供されています。

DocumentBroker では，メソッドによってプロパティに値を設定したり，プロパティの値を取得したりして，オブジェクトの状態を操作します。プロパティに値を設定するメソッドを**更新系メソッド**，プロパティから値を取得するメソッドを**参照系メソッド**といいます。

なお，メソッドには，継承関係があります。メソッドの継承関係は，クラスの継承関係と同様で，上位のクラスで定義されたメソッドは，そのサブクラスに継承されます。例えば，CdbrVersionable クラスのサブクラスである CdbrVersionableDocument クラスでは，CdbrVersionableDocument クラスで定義されたメソッドのほか，CdbrVersionable クラスで定義されているメソッドも使用できます。

2.2　オブジェクトの操作の流れ

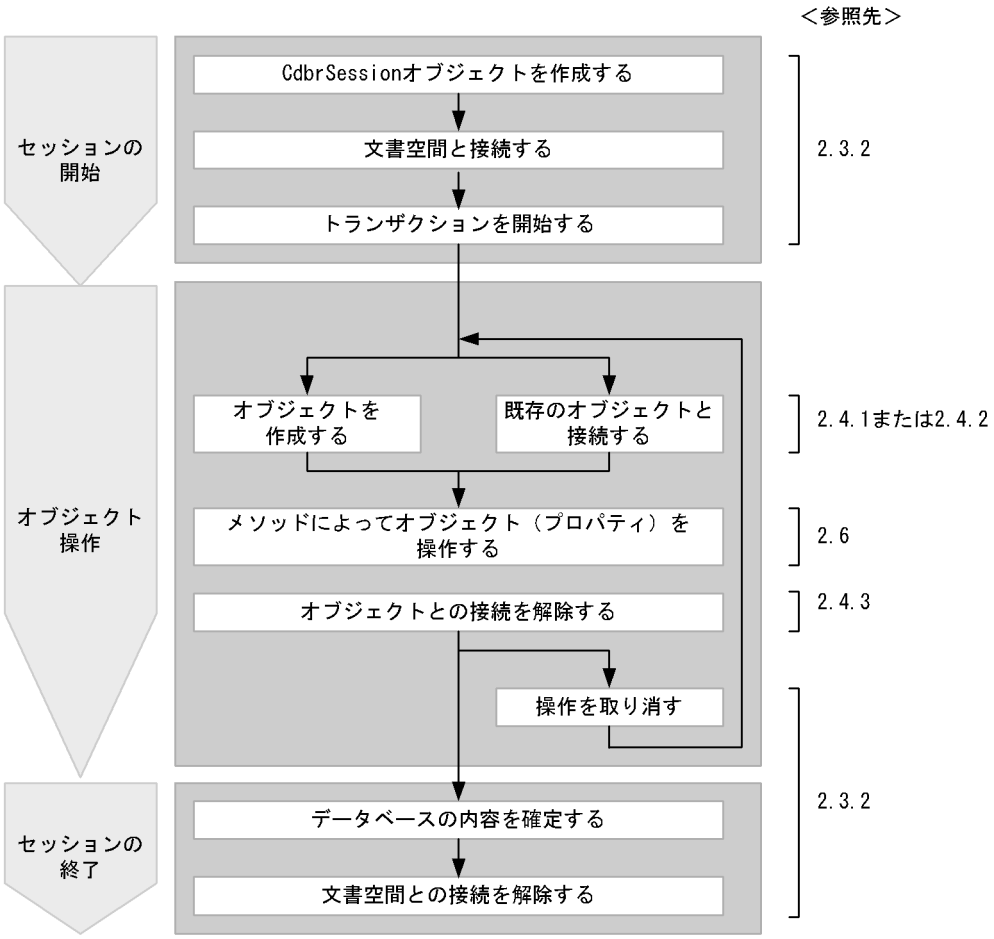
この節では，DocumentBroker のオブジェクトをクラスライブラリの機能で操作する場合の，操作の流れについて説明します。

クラスライブラリを使用してオブジェクトを操作するためには，DocumentBroker サーバの文書空間と接続する必要があります。また，オブジェクトの操作が終了したあと，DocumentBroker の文書空間との接続を解除する必要があります。

なお，文書空間に接続している状態を，**セッション**といいます。また，オブジェクトの操作の処理単位を，**トランザクション**といいます。

文書空間との接続から文書空間との接続解除までの操作の流れを，次の図に示します。

図 2-2　オブジェクトの操作の流れ



2.3 文書空間との接続（セッションの確立）

文書空間上のオブジェクトを操作するためには，まず，クライアントの実行環境とサーバの文書空間を接続して，トランザクションを開始します。

この節では，クライアントの実行環境と文書空間を接続する方法について説明します。文書空間と接続するためには，CdbrSession クラスの機能を使用します。

また，CdbrSession クラスが提供している，ログイン管理機能についても説明します。

2.3.1 CdbrSession クラスの概要

CdbrSession クラスは，文書空間とのセッションを確立する機能を提供します。

CdbrSession クラスの主な機能を次に示します。

- 文書空間との接続・解除
- 文書空間に接続するときのログイン管理
- 文書空間に作成したオブジェクトの管理（オブジェクトの操作の確定およびキャンセル）
- 文書空間の情報の取得
- クライアント環境から要求の ORB を介したサーバでの実行
- ログインユーザが保持している情報の取得（アクセス制御機能を使用する場合）

2.3.2 セッションの確立から終了までの流れ

セッションの確立から終了までの流れを説明します。また，セッションの中でオブジェクトを操作したときの，操作の確定および取り消しについても説明します。なお，それぞれのメソッドの文法については，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。

(1) セッションの確立

オブジェクトを操作する場合，まずセッションを確立します。ここでは，セッションを確立する手順について説明します。セッションの確立とは，文書空間と接続してトランザクションを開始することです。

セッションを確立する流れについて次に示します。

1. **CdbrSession オブジェクトを生成します。**
   CdbrSession オブジェクトは，new によって作成します。
2. **文書空間に接続します。**
   CdbrSession::Connect メソッドをコールします。
3. **トランザクションを開始します。**
   CdbrSession::Begin メソッドをコールします。
   セッションの確立以降，データベースに格納されたオブジェクトを操作できます。

(2) オブジェクトの操作の確定と取り消し

文書空間でオブジェクトを操作したら，操作を確定する必要があります。操作を確定するためには，CdbrSession::Commit メソッドをコールします。これによって，データベースのオブジェクトの状態が確定されます。

また，オブジェクトに対して行った操作を取り消したい場合は，CdbrSession::Rollback メソッドをコー

ルします。これによって，前回 CdbrSession::Commit メソッドによって確定した以降の操作が取り消され，CdbrSession::Commit メソッドをコールした段階のオブジェクトの状態に戻ります。

CdbrSession::Commit メソッド，または CdbrSession::Rollback メソッドをコールすると，その時点でトランザクションは完了します。このため，新たにトランザクションを開始する場合は，CdbrSession::Begin メソッドをコールする必要があります。

(3) セッションの終了

オブジェクトの操作が終了したら，セッションを終了します。セッションは，データベースの内容を確定したあとで終了してください。

セッションは，次の流れで終了します。

1. **CdbrSession::Disconnect メソッドをコールします。**
   これによって，文書空間との接続が解除され，セッションが終了します。

(4) 注意事項

- 同一のセッション上で動作するメソッドを，多重実行することはできません。多重実行した場合は，エラーになります。
- クライアントから CdbrSession クラスの Connect メソッドを発行した際，無応答が発生する場合には，DocumentBroker サーバとクライアントプロセスを再起動したあとに，再度クライアントから CdbrSession クラスの Connect メソッドを発行してください。なお，DocumentBroker サーバの起動と終了については，マニュアル「DocumentBroker Version 3　システム導入・運用ガイド」を参照してください。

2.3.3 ログイン

CdbrSession クラスは，ユーザが文書空間にログインするときのログイン管理機能を提供します。ユーザ名およびパスワードは，CdbrSession::Connect メソッドの引数として指定します。

ユーザ名およびパスワードを指定することで，文書空間への接続時に，DocumentBroker にアクセスできるユーザかどうかを認証できます。クライアントアプリケーションは，リクエストメッセージを要求し，サーバは応答メッセージで応答することでユーザを認証します。

認証に使用するユーザ ID およびパスワードについては，マニュアル「DocumentBroker Version 3　システム導入・運用ガイド」を参照してください。

2.4 オブジェクトの作成およびオブジェクトとの接続

この節では，セッション確立後のオブジェクトの操作について説明します。

新しいオブジェクトを操作する場合は，オブジェクトを作成して操作します。既存のオブジェクトを操作する場合は，オブジェクトに接続して操作します。

2.4.1 オブジェクトの作成

オブジェクトは，CreateObject メソッドをコールして作成します。このメソッドによって，文書空間上に DMA オブジェクトを構成要素としたクラスライブラリのオブジェクトが作成されます。作成するクラスライブラリのオブジェクトを構成する DMA オブジェクトを特定するために，CreateObject メソッドをコールする時に，DMA オブジェクトを作成する基になる DMA クラスを CreateObject メソッドの引数に指定する構造体（SDBR_DMAINFO 構造体）に指定します。

それぞれのクラスライブラリのオブジェクトを作成する時に指定する DMA オブジェクトのクラス識別子を次の表に示します。なお，クラス識別子の代わりに NULL を指定することもできます。NULL を指定すると，デフォルトのクラス識別子を指定したものとしてオブジェクトが作成されます。

表 2-1 クラスライブラリのオブジェクトを作成するときに指定する DMA オブジェクトのクラス識別子

クラスライブラリのクラス	指定する DMA オブジェクトのクラス識別子	デフォルトのクラス識別子
CdbrConfiguratedReferentialContainer クラス	dmaClass_ConfigurationHistory クラスまたはそのサブクラス	dmaClass_ConfigurationHistory
	edmClass_ContainerVersion クラスまたはそのサブクラス	edmClass_ContainerVersion
CdbrDocument クラス	dmaClass_DocVersion クラスまたはそのサブクラス	edmClass_VersionTracedComponent Docversion
CdbrIndependentPersistence クラス	edmClass_IndependentPersistence クラスまたはそのサブクラス	edmClass_IndependentPersistence
CdbrPublicACL クラス	edmClass_PublicACL クラス	edmClass_PublicACL
CdbrReferentialContainer クラス	dmaClass_Container クラスまたはそのサブクラス	dmaClass_Container
CdbrVersionableDocument クラス	dmaClass_ConfigurationHistory クラスまたはそのサブクラス	dmaClass_ConfigurationHistory
	dmaClass_DocVersion クラスまたはそのサブクラス	edmClass_VersionTracedComponent Docversion
CdbrVersionTraceableContainer クラス	edmClass_ContainerVersion クラスまたはそのサブクラス	edmClass_ContainerVersion

CreateObject メソッドによってオブジェクトを作成するコールシーケンスの例を示します。

CreateObject メソッドによってオブジェクトを作成するコールシーケンス

```
CdbrDocument ObjDoc;
SDBR_DMAINFO DmaClassList[1];

//DMA情報構造体にクラス識別子を設定する
DmaClassList[0].ClassId = usrClass_DocVersion;
DmaClassList[0].PropList.pItem = NULL;
DmaClassList[0].PropList.lCount = 0;
pSession->Begin();
ObjDoc.CreateObject(pSession,
     1, DmaClassList,
```

```
    "file:///tmp/sample.txt", "MIME::text/plain", &pOIID,
     DBR_CREATE_INDEX);
ObjDoc.ReleaseObject();
pSession->Commit();
```

2.4.2 既存のオブジェクトとの接続

既存のオブジェクトを操作するためには，まず，そのオブジェクトに接続する必要があります。オブジェクトに接続するメソッドには，次の 2 種類があります。

- CdbrDMA::SetOIID メソッド
- CdbrDMA::ConnectObject メソッド

これらのメソッドの主な違いは，処理性能です。SetOIID メソッドを使用すると，接続時にオブジェクトにロックを設定しない（データベースにアクセスしない）ため，ConnectObject メソッドを使用する場合に比べて高速な処理が実現できます。

ここでは，それぞれのメソッドの特長について説明します。ロックについての詳細は，「3.14　排他制御」を参照してください。

また，既存のオブジェクトに接続する場合，そのオブジェクトの OIID を指定します。OIID がわからない場合は，事前に検索等を実行して，OIID を取得しておく必要があります。

検索については，「4.　オブジェクトの検索」を参照してください。

（1） SetOIID メソッドを使用したオブジェクトへの接続

SetOIID メソッドは，ロックを設定しないでオブジェクトに接続するメソッドです。ロックを設定する場合に比べて，高速に処理できます。

SetOIID メソッドをコールしてオブジェクトに接続した場合，それ以降の操作で，必要に応じてロックを設定します。

SetOIID メソッドを使用したコールシーケンスの例を次に示します。

SetOIID メソッドを使用してプロパティの値を参照，設定するコールシーケンス

```
CdbrDocument ObjDoc;
pSession->Begin();
//ObjDocにOIIDを設定する
ObjDoc.SetOIID(pSession,pOIID);
//プロパティを参照すると同時に，WRITEロックを設定する
ObjDoc.GetPropertyValuesAndLock(...,DMA_LOCK_WRITE);
//プロパティを更新する
ObjDoc.PutPropertyValues(...);
ObjDoc.ReleaseObject();
pSession->Commit();
```

（2） ConnectObject メソッドを使用したオブジェクトへの接続

ConnectObject メソッドは，ロックを設定してオブジェクトに接続するメソッドです。引数に指定した種類のロックがオブジェクトに設定されます。

ConnectObject メソッドを使用したコールシーケンスの例を次に示します。

ConnectObject メソッドを使用してプロパティの値を参照，設定するコールシーケンス

```
CdbrDocument ObjDoc;
pSession->Begin();
//データベースにアクセスして，ObjDocにWRITEロックを設定する
ObjDoc.ConnectObject(pSession,DMA_LOCK_WRITE,pOIID);
```

```
//WRITEロックを設定したオブジェクトのプロパティを参照する
ObjDoc.GetPropertyValues(...);
//プロパティを更新する
ObjDoc.PutPropertyValues(...);
ObjDoc.ReleaseObject();
pSession->Commit();
```

2.4.3 オブジェクトとの接続の解除

オブジェクトとの接続は，操作が完了した段階で解除します。

オブジェクトとの接続の解除には，ReleaseObject メソッドをコールします。

2.5 クラスの種類

この節では，DMA クラスおよびクラスライブラリのクラスの種類について説明します。

2.5.1 DMA クラスのクラス識別子

DMA クラスは，クラス識別子として，Id 型の値（GUID 値）を持ちます。この識別子は，メソッドなどで DMA クラスを指定するとき，DMA クラスを識別するために使用します。

また，DMA が規定した DMA クラスまたは DocumentBroker が拡張した DMA クラスのクラス識別子を指定する場合，GUID 値のほか，クラスライブラリのヘッダファイルで定義している定数をクラス識別子として指定することもできます。例えば，「dmaClass_DocVersion」や「edmClass_VersionTracedComponentDocVersion」などがこれに当たります。

ユーザが定義したサブクラスのクラス識別子を任意の定数（usrClass_XXX など）で指定する場合は，GUID 値の実体と定数の対応を定義したヘッダファイルを作成する必要があります。ヘッダファイルの作成方法については，「5.2.7　ユーザ定義のクラス識別子・プロパティ識別子の定義」を参照してください。

なお，このマニュアルでは，GUID 値の実体と dmaClass_DocVersion のような定数をあわせて**クラス識別子**と呼びます。

2.5.2 クラスライブラリのクラスの一覧

クラスライブラリで提供するクラスの一覧を次の表に示します。なお，それぞれのクラスの詳細については，参照先で説明しています。

表 2-2　クラスライブラリで提供するクラス一覧

クラス名	説明	参照先
文書空間と接続するためのクラス		
CdbrSession クラス	クライアント実行環境と文書空間との接続の管理をする機能を提供するクラスです。また，ログイン管理機能も提供します。	2.3　文書空間との接続（セッションの確立）
文書を作成，管理するためのクラス		
CdbrVersionableDocument クラス	DocumentBroker で扱う文書のうち，バージョン管理をする文書を作成および管理する機能を提供するクラスです。	3.2　クラスライブラリで扱う文書
CdbrDocument クラス	DocumentBroker で扱う文書のうち，バージョン管理をしない文書や，バージョン管理をしている文書の個々のバージョンに当たる文書を作成および管理する機能を提供するクラスです。	
複数の文書をまとめて管理するためのクラス		
CdbrReferentialContainer クラス	文書をコンテナによって包含して管理するための機能を提供するクラスです。	3.9　コンテナを使用した文書管理
文書の構成管理をするためのクラス		
CdbrConfiguratedReferentialContainer クラス	コンテナを使用して文書やコンテナのバージョン構成を管理したり，コンテナ自身のバージョンを管理したりする機能を提供するクラスです。	3.10　構成管理コンテナを使用した文書の構成管理

クラス名	説明	参照先
CdbrVersionTraceableContainer クラス	コンテナを使用して文書やコンテナのバージョン構成を管理する機能を提供するクラスです。また，バージョン管理しているコンテナの，個々のバージョンに当たるコンテナを操作する機能を提供します。	
XML 文書管理機能を使用するためのクラス		
CdbrXmlTranslatorFactory クラス	XML プロパティマッピングに使用する定義ファイル類を解析して，XML 文書管理機能を実行するための環境を管理する機能を提供するクラスです。	3.11　XML 文書の管理
CdbrXmlTranslator クラス	XML 文書管理を実行する機能を提供するクラスです。	
独立データを管理するためのクラス		
CdbrIndependentPersistence クラス	独立したデータを扱うための機能を提供するクラスです。	3.13　独立データの管理
文書やコンテナを検索するためのクラス		
CdbrEqlStatement クラス	edmSQL を基にした検索を実行する機能や，検索結果を取得する機能を提供するクラスです。	4.　オブジェクトの検索
可変長な一次元配列を扱うためのクラス		
CdbrCompound クラス	異なる型の複数のデータから構成される複合データを扱う機能を提供するクラスです。	2.6.7　VariableArray 型プロパティの操作
CdbrVariableArray クラス	同じ型の複数の要素から構成される可変長の配列を扱う機能を提供するクラスです。なお，このクラスで扱うことができる同じ型の要素とは，CdbrCompound クラスを基に作成された CdbrCompound オブジェクトだけです。	
アクセス制御リストを管理するためのクラス		
CdbrPublicACL クラス	アクセス制御リストを管理して，これを複数の文書やコンテナで共有して使用する機能を提供するクラスです。	3.15　アクセス制御
サブクラスに共通な機能を継承させるためのクラス（抽象クラス）		
CdbrContainable クラス	コンテナの包含要素になる機能を提供するクラスです。	2.5.4　抽象クラス
CdbrCore クラス	クラスライブラリのクラスのうち，CdbrVariableArray クラスおよび CdbrCompound クラス以外のすべてのクラスのスーパークラスです。エラー管理機能を持ちます。	
CdbrDMA クラス	DMA のオブジェクトモデルの dmaClass_DMA クラスに対応するクラスです。OIID の取得およびプロパティの参照や設定をするメソッドを定義します。	
CdbrVersionable クラス	バージョン管理機能を提供するクラスです。	

表 2-3　クラスライブラリのクラスと DMA のクラスの対応

クラスライブラリのクラス	構成要素である DMA クラス
CdbrCompound クラス	―

クラスライブラリのクラス	構成要素である DMA クラス
CdbrConfiguratedReferentialContainer クラス	• dmaClass_ConfigurationHistory クラス • dmaClass_DirectContainmentRelationship クラス • dmaClass_ReferentialContainmentRelationship クラス • dmaClass_VersionDescription クラス • dmaClass_VersionSeries クラス • edmClass_ContainerVersion クラス • edmClass_VersionTraceableContainmentRelationship クラス
CdbrContainable クラス	• dmaClass_Containable クラス
CdbrCore クラス	―
CdbrDMA クラス	• dmaClass_DMA クラス
CdbrDocument クラス	**シングルファイル文書の場合** • dmaClass_ContentTransfer クラス • dmaClass_DocVersion クラス • dmaClass_Rendition クラス **マルチファイル文書の場合** • dmaClass_DocVersion クラス • dmaClass_Rendition クラス • edmClass_ContentTransfers クラス **リファレンスファイル文書の場合** • dmaClass_DocVersion クラス • dmaClass_Rendition クラス • edmClass_ContentReference クラス File Link **文書の場合** • dmaClass_DocVersion クラス • dmaClass_Rendition クラス • edmClass_ContentFileLink クラス
CdbrEqlStatement クラス	―
CdbrIndependentPersistence クラス	• edmClass_IndependentPersistence クラス
CdbrPublicACL クラス	• edmClass_PublicACL クラス
CdbrReferentialContainer クラス	• dmaClass_Container クラス • dmaClass_DirectContainmentRelationship クラス • dmaClass_ReferentialContainmentRelationship クラス
CdbrSession クラス	• dmaClass_DMA クラス • dmaClass_DocSpace クラス
CdbrVariableArray クラス	―
CdbrVersionable クラス	• dmaClass_ConfigurationHistory クラス • dmaClass_Versionable クラス • dmaClass_VersionDescription クラス • dmaClass_VersionSeries クラス

クラスライブラリのクラス	構成要素である DMA クラス
CdbrVersionableDocument クラス	**シングルファイル文書の場合** • dmaClass_ConfigurationHistory クラス • dmaClass_ContentTransfer クラス • dmaClass_DocVersion クラス • dmaClass_Rendition クラス • dmaClass_VersionDescription クラス • dmaClass_VersionSeries クラス **マルチファイル文書の場合** • dmaClass_ConfigurationHistory クラス • dmaClass_DocVersion クラス • dmaClass_Rendition クラス • dmaClass_VersionDescription クラス • dmaClass_VersionSeries クラス • edmClass_ContentTransfers クラス **リファレンスファイル文書の場合** • dmaClass_ConfigurationHistory クラス • dmaClass_DocVersion クラス • dmaClass_Rendition クラス • dmaClass_VersionDescription クラス • dmaClass_VersionSeries クラス • edmClass_ContentReference クラス File Link **文書の場合** • dmaClass_ConfigurationHistory クラス • dmaClass_DocVersion クラス • dmaClass_Rendition クラス • dmaClass_VersionDescription クラス • dmaClass_VersionSeries クラス • edmClass_ContentFileLink クラス
CdbrVersionTraceableContainer クラス	• dmaClass_DirectContainmentRelationship クラス • dmaClass_ReferentialContainmentRelationship クラス • edmClass_ContainerVersion クラス • edmClass_VersionTraceableContainmentRelationship クラス
CdbrXmlTranslatorFactory クラス	－
CdbrXmlTranslator クラス	－

（凡例）
－：対応する DMA のクラスがない。

2.5.3 クラスの継承関係

クラスライブラリで提供しているクラスは，その直上にあるクラスが持つプロパティとメソッドを継承します。

継承の基になるクラスを，あるクラスの**スーパークラス**といいます。また，あるクラスの直下にあるクラスを，そのクラスの**サブクラス**といいます。次の図に，クラスの関係を示します。

図 2-3 クラスの関係

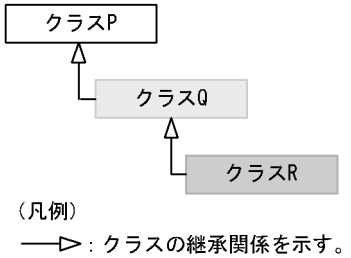
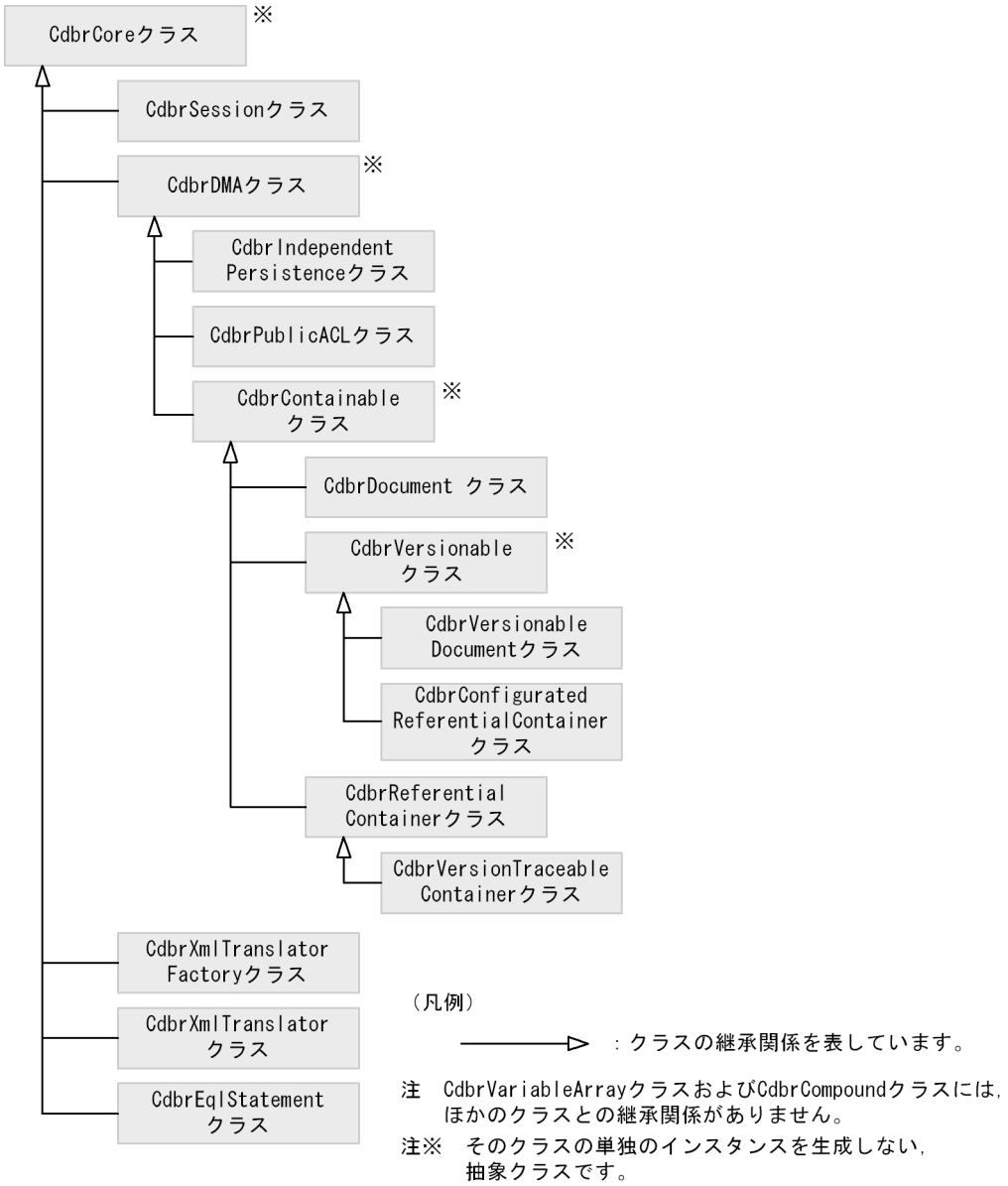


図 2-3 でのクラスの関係を説明します。

- クラス Q はクラス R の直接のスーパークラスである。
- クラス P はクラス Q の直接のスーパークラスである。したがって，クラス P およびクラス Q は，クラス R のスーパークラスである。
- クラス R は，クラス Q の固有のサブクラスである。また，直接のサブクラスである。

クラスライブラリで提供しているクラスの継承関係を，次の図に示します。

図 2-4　クラスの継承関係



2.5.4　抽象クラス

ここでは，クラスライブラリのクラスのうち，抽象クラスについて説明します。抽象クラスとは，そのクラス自身単独のオブジェクトを生成しないクラスです。

クラスライブラリとして提供する抽象クラスは，次の 4 種類です。

● CdbrCore クラス

● CdbrDMA クラス

● CdbrContainable クラス

● CdbrVersionable クラス

クラスライブラリの各クラスは，これらの抽象クラスをスーパークラスに持つことによって，その抽象ク

ラスの機能（プロパティとメソッド）を継承して使用できます。

(1) CdbrCore クラス

エラー管理機能を提供するクラスです。

クラスライブラリのクラスのうち，CdbrVariableArray クラスおよび CdbrCompound クラス以外のすべてのクラスのスーパークラスです。

(2) CdbrDMA クラス

構成要素として DMA オブジェクトを作成するクラスに共通のスーパークラスです。

DMA クラスの dmaClass_DMA クラスに対応するクラスです。

CdbrDMA クラスでは，このクラスを継承するクラスに共通なメソッドとして，主に次のようなメソッドを提供します。

- プロパティを参照，設定するメソッド
- OIID を取得するメソッド
- 指定した OIID を基に既存のオブジェクトと接続するメソッド

OIID とは，文書空間でのオブジェクトの存在や格納位置を表す識別子です。OIID の形式については，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。

(3) CdbrContainable クラス

コンテナで管理される要素（文書やコンテナ）を生成するクラスのスーパークラスです。このクラスは，次のクラスのスーパークラスです。

- CdbrConfiguratedReferentialContainer クラス
- CdbrDocument クラス
- CdbrReferentialContainer クラス
- CdbrVersionable クラス
- CdbrVersionableDocument クラス
- CdbrVersionTraceableContainer クラス

コンテナによってまとめて管理できるのは，これらの CdbrContainable クラスを継承したクラスから作成されたオブジェクトです。

(4) CdbrVersionable クラス

CdbrVersionableDocument クラスおよび CdbrConfiguratedReferentialContainer クラスのスーパークラスです。文書やコンテナをバージョン管理するための機能を提供します。このクラスを継承したクラスを基に作成したオブジェクトは，バージョン管理の対象になります。

2.6 プロパティの種類と操作

この節では，プロパティの種類とその操作方法について説明します。

2.6.1 プロパティ識別子

それぞれのプロパティは，プロパティ識別子として，Id 型の値（GUID 値）を持ちます。この識別子は，プロパティを参照および設定するときに，プロパティを識別するために使用します。

また，DMA が規定したプロパティまたはクラスライブラリ固有のプロパティのプロパティ識別子を指定する場合，GUID 値のほか，クラスライブラリのヘッダファイルで定義している定数をプロパティ識別子として指定することもできます。例えば，「dmaProp_OIID」や「dbrProp_ACL」などがこれに当たります。

ユーザが定義したプロパティのプロパティ識別子を任意の定数（usrProp_XXX など）で指定する場合は，GUID 値の実体と定数の対応を定義したヘッダファイルを作成する必要があります。ヘッダファイルの作成方法については，「5.2.7　ユーザ定義のクラス識別子・プロパティ識別子の定義」を参照してください。

なお，このマニュアルでは，GUID 値の実体と dmaProp_OIID のような定数をあわせて**プロパティ識別子**と呼びます。

2.6.2 プロパティの種類

クラスライブラリで使用できるプロパティには，次の 3 種類があります。

- DMA が規定したプロパティ（DMA プロパティ）
  「dmaProp_」または「edmProp_」で始まる識別子を持つプロパティです。
- クラスライブラリで定義したプロパティ（クラスライブラリ固有のプロパティ）
  「dbrProp_」で始まる識別子を持つプロパティです。
- ユーザが定義したプロパティ
  ユーザが定義したプロパティです。

なお，プロパティのうち，データベースに値が格納されているプロパティを，**永続プロパティ**といいます。

（1）DMA が規定したプロパティ（DMA プロパティ）

DMA では，DMA クラスごとにプロパティを規定しています。また，DocumentBroker で拡張したクラスには，DocumentBroker 独自のプロパティも定義されています。これらのプロパティは，クラスライブラリのクラスのプロパティとしても参照，設定できます。クラスライブラリのメソッドで参照，設定できるプロパティは，それぞれのクラスライブラリのクラスに対応する DMA のクラスが定義しているプロパティです。これらのプロパティは，CreateObject メソッドで指定したクラスの DMA オブジェクト上に定義されます。

クラスライブラリの各クラスのメソッドによって参照，設定できるプロパティの定義元となる DMA クラスを次の表に示します。

表 2-4　各クラスが参照，設定できるプロパティの定義元

クラスライブラリのクラス名	バージョン指定	定義元の DMA クラス
CdbrConfiguratedReferentialContainer クラス	なし	dmaClass_ConfigurationHistory クラス
	あり	edmClass_ContainerVersion クラス

クラスライブラリのクラス名	バージョン指定	定義元の DMA クラス
CdbrDocument クラス	－	dmaClass_DocVersion クラス
CdbrIndependentPersistence クラス	－	edmClass_IndependentPersistence クラス
CdbrPublicACL クラス	－	edmClass_PublicACL クラス
CdbrReferentialContainer クラス	－	dmaClass_Container クラス
CdbrVersionableDocument クラス	なし	dmaClass_ConfigurationHistory クラス
	あり	dmaClass_DocVersion クラス
CdbrVersionTraceableContainer クラス	－	edmClass_ContainerVersion クラス

（凡例）
－：該当しない。
なし：複数のバージョンを持つ文書またはコンテナ全体を指定した場合
あり：複数のバージョンを持つ文書またはコンテナの特定のバージョンを指定した場合

それぞれの DMA クラスに定義されたプロパティについての詳細は，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。

(2) クラスライブラリで定義したプロパティ（クラスライブラリ固有のプロパティ）

クラスライブラリ固有のプロパティとは，表 2-4 に示した定義元の DMA クラスには存在しない情報を，クラスライブラリのプロパティとして扱えるように定義したものです。

例えば，あるコンテナに関連づけられている文書やコンテナの数や，文書が保持しているバージョンの数などは，DMA オブジェクト上には存在しない情報です。このような情報に対して，クラスライブラリでは「dbrProp_」から始まるプロパティ識別子を付けて，プロパティとして扱えるようにしています。「コンテナが関連づけている文書の数を知りたい」場合などに，このプロパティの値を取得するメソッドをコールすれば，必要な値を得ることができます。実際にはメソッドをコールした段階で，クラスライブラリの内部で必要な処理や演算が実行されて，値が作成され，プロパティの値としてユーザに返却されます。

また，DMA で規定されているプロパティのうち，クラスライブラリのオブジェクトのプロパティとして使用できるのは表 2-4 に示した定義元の DMA クラスのプロパティだけです。これに対して，定義元以外のクラスに存在するプロパティの値をクラスライブラリのプロパティとして扱うためにも，クラスライブラリ固有のプロパティが使用されます。例えば，文書を表す CdbrDocument オブジェクトの場合，文書のコンテントを表すファイル名についての情報は，プロパティの定義元である DocVersion オブジェクト上には存在しない情報ですが，実際は CdbrDocument オブジェクト内のほかの DMA オブジェクト上にプロパティとして存在しています。このプロパティを取得するために，クラスライブラリではプロパティ識別子を割り付けています。このプロパティの値を取得するメソッドをコールすれば，必要な値を得ることができます。実際にはメソッドをコールした段階で，クラスライブラリの内部で該当するプロパティが参照され，CdbrDocument オブジェクトのプロパティとしてユーザに返却されます。

注意

クラスライブラリ固有のプロパティは，検索条件には指定できません。

クラスライブラリ固有のプロパティについて，表 2-5 に示します。また，アクセス制御機能を使用する場合に，表 2-5 に示すプロパティに加えて追加されるプロパティを，表 2-6 に示します。アクセス制御機能については，「3.15　アクセス制御」を参照してください。

表 2-5 クラスライブラリ固有のプロパティ

プロパティ識別子	導入されるクラス	説明
dbrProp_ChildrenCount	• CdbrReferentialContainer クラス	直接型のコンテインメントで包含しているオブジェクトの個数
dbrProp_ContaineesCount	• CdbrReferentialContainer クラス	参照型のコンテインメントで包含しているオブジェクトの個数
dbrProp_ContaineesCountVT	• CdbrConfiguratedReferentialContainer クラス	構成管理の対象となるオブジェクトの個数
dbrProp_ContainersCount	• CdbrContainable クラス	参照型のコンテインメントで Containable オブジェクトを包含しているコンテナの個数
dbrProp_ContainersCountVT	• CdbrVersionable クラス • CdbrDocument クラス • CdbrVersionTraceableContainer クラス	構成管理型のコンテインメントでオブジェクトを包含しているコンテナの個数
dbrProp_ContentLocation	• CdbrDocument クラス • CdbrVersionableDocument クラス	管理しているコンテントロケーション
dbrProp_ContentType	• CdbrDocument クラス	管理しているコンテントの種類
dbrProp_CurrentVersion	• CdbrVersionable クラス	カレントバージョンの識別子
dbrProp_HeadRelationsCount	• CdbrVersionableDocument クラス • CdbrDocument クラス	関連づけているリレーション先文書の個数
dbrProp_ParentCount	• CdbrContainable クラス	直接型のコンテインメントでオブジェクトを包含しているコンテナの個数
dbrProp_ReferenceType	• CdbrDocument クラス • CdbrVersionableDocument クラス	管理しているコンテントのリファレンス種別
dbrProp_RenditionStatus	• CdbrDocument クラス	レンディションの状態
dbrProp_RenditionType	• CdbrVersionableDocument クラス • CdbrDocument クラス	RenditionType を表す文字列
dbrProp_RetrievalName	• CdbrVersionableDocument クラス • CdbrDocument クラス	登録されているファイルのファイル名
dbrProp_TailRelationsCount	• CdbrVersionableDocument クラス • CdbrDocument クラス	関連づけられているリレーション元文書の個数
dbrProp_VersionsCount	• CdbrVersionable クラス	オブジェクトが管理しているバージョンの個数

表 2-6 アクセス制御機能を使用する場合に定義されるクラスライブラリ固有のプロパティ

プロパティ識別子	導入されるクラス	説明
dbrProp_ACL	• CdbrContainable クラス • CdbrIndependentPersistence クラス • CdbrPublicACL クラス	ACL（アクセス制御リスト）を表す ACE（アクセス制御エレメント）のリスト
dbrProp_BindObjectCount	• CdbrPublicACL クラス	このプロパティを持つパブリック ACL をバインドしているクラスライブラリのオブジェクトの数
dbrProp_EveryonePermission	• CdbrContainable クラス • CdbrIndependentPersistence クラス	すべてのユーザに与えられるパーミッション
dbrProp_OwnerId	• CdbrDMA クラス	オブジェクトの所有者の識別子
dbrProp_OwnerPermission	• CdbrContainable クラス • CdbrIndependentPersistence クラス	オブジェクトの所有者に与えられるパーミッション

プロパティ識別子	導入されるクラス	説明
dbrProp_PrimaryGroupId	• CdbrContainable クラス • CdbrIndependentPersistence クラス	プライマリグループのグループ識別子
dbrProp_PrimaryGroupPermission	• CdbrContainable クラス • CdbrIndependentPersistence クラス	プライマリグループに与えられるパーミッション
dbrProp_PublicACLCount	• CdbrContainable クラス • CdbrIndependentPersistence クラス	バインドしているパブリック ACL の数
dbrProp_PublicACLIds	• CdbrContainable クラス • CdbrIndependentPersistence クラス	バインドしているパブリック ACL の OIID のリスト
dbrProp_SACL	• CdbrContainable クラス • CdbrIndependentPersistence クラス • CdbrPublicACL クラス	セキュリティ ACL を表す ACE（アクセス制御エレメント）のリスト
dbrProp_UserPermission	• CdbrDMA クラス	オブジェクトに設定されるログインユーザまたはログインユーザが所属するグループに与えられているパーミッションの論理和

それぞれのプロパティの詳細については，マニュアル「DocumentBroker Version 3　クラスライブラリ C++　リファレンス　基本機能編」の，それぞれのプロパティが導入されるクラスを参照してください。

また，アクセス制御機能を使用する場合に定義されるプロパティについては，指定方法によって，検索の対象にできるプロパティがあります。アクセス制御機能を使用する場合に定義されるプロパティについては，「3.15　アクセス制御」を参照してください。

(3) ユーザが定義したプロパティ

文書の作成者やタイトルなど，ユーザは必要に応じて DMA オブジェクトにプロパティを定義できます。このプロパティは，クラスライブラリのクラスを構成する DMA クラスのうち，クラスライブラリのクラスのプロパティの定義元になる DMA クラスに追加します。

例えば，CdbrDocument オブジェクトを扱う場合に使用するユーザ定義プロパティを追加するときは，dmaClass_DocVersion クラスにプロパティを追加定義します。

追加したプロパティは検索の条件に指定することができますので，必要に応じて定義してください。プロパティの追加方法については，マニュアル「DocumentBroker Version 3　システム導入・運用ガイド」を参照してください。

2.6.3 プロパティの型

クラスライブラリで使用するプロパティのデータ型について次の表に示します。

表 2-7　クラスライブラリで使用するプロパティのデータ型

プロパティのデータ型	意味	記号定数	備考
Boolean	真偽	DMA_DATATYPE_BOOLEAN	次のどれかの値を取る • DMA_TRUE • DMA_FALSE • DMA_UNKNOWN
ID	GUID	DMA_DATATYPE_ID	GUID 構造体
Integer32	整数	DMA_DATATYPE_INTEGER32	—

プロパティのデータ型	意味	記号定数	備考
Object	オブジェクト	DMA_DATATYPE_OBJECT	可変長配列オブジェクトが返される。基本単位が VariableArray 型のプロパティを設定できる
String	文字列	DMA_DATATYPE_STRING	最大文字列長は，DocumentBroker サーバのメタ情報で定義する

このうち，ユーザが定義できるのは，Boolean 型，Integer32 型，Object 型および String 型のプロパティです。ただし，Object 型の場合，基本単位には VariableArray 型だけが定義できます。

2.6.4 プロパティの基本単位

プロパティは，保持できる要素の個数や持ち方によって分類できます。これを，プロパティの基本単位といいます。

要素の個数および持ち方によるプロパティの分類について次の表に示します。

表 2-8 要素の個数および持ち方によるプロパティの分類

プロパティの基本単位	記号定数	値の数
Scalar 型	DMA_CARDINALITY_SINGLE	一つの値を取る
VariableArray 型	EDM_DMA_CARDINALITY_VARRAY	複数の値を取る

Scalar 型および VariableArray 型のどちらのプロパティもユーザ定義プロパティとして定義できます。

それぞれの基本単位のプロパティについて説明します。

（1） Scalar 型

データ型に従った値を一つだけ持つプロパティです。

（2） VariableArray 型

VariableArray 型は，すべてのデータ型の値の可変長な一次元配列（可変長配列）です。異なる種類のデータ型は混在できません。また DocumentBroker では，Object 型の値だけを配列要素にできます。

なお，基本単位が VariableArray 型のプロパティには，一次元配列の要素のデータ型である Object 型に従った VariableArrayOfObject オブジェクト（DMA オブジェクト）へのリファレンスが格納されます。実際の値は参照先の VariableArrayOfObject オブジェクトの一次元配列が保持します。一次元配列内の要素にはランダムにアクセスできます。また，要素の挿入，置換および削除ができます。

なお，このマニュアルでは，基本単位が VariableArray 型の Object 型プロパティを，**VariableArray 型プロパティ**と呼びます。

VariableArray 型プロパティは，DMA クラスの次のクラスまたはそのサブクラスに定義できます。

- dmaClass_ConfigurationHistory クラス
- dmaClass_Container クラス
- dmaClass_DocVersion クラス
- edmClass_ContainerVersion クラス
- edmClass_IndependentPersistence クラス

したがって，このプロパティを設定できるクラスライブラリのオブジェクトは，これらの DMA クラスをプロパティの定義元とする，次のオブジェクトになります。

- CdbrConfiguratedReferentialContainer オブジェクト
- CdbrDocument オブジェクト
- CdbrReferentialContainer オブジェクト
- CdbrIndependentPersistence オブジェクト
- CdbrVersionableDocument オブジェクト
- CdbrVersionTraceableContainer オブジェクト

VariableArray 型の特徴についてまとめると，次の表のようになります。

表 2-9　VariableArray 型の特徴

項目	特徴
アクセス方法	ランダム
要素の順序	不定
値の更新	可能
要素の追加	末尾にだけ可能
要素の削除	末尾だけ可能
プロパティの定義での指定	可能 ただし，追加できるクラスには制限がある
要素のデータ型	Object 型 ただし，edmClass_Struct クラスのサブクラスのオブジェクトだけ可能

クラスライブラリでは，VariableArray 型プロパティの値を表す可変長配列は，CdbrVariableArray オブジェクトで表します。また，この可変長配列である複合データは，CdbrCompound オブジェクトとして表します。

したがって，クラスライブラリでは，CdbrVariableArray クラスおよび CdbrCompound クラスのメソッドを使用することによって，VariableArray 型プロパティの値を，参照，設定できます。

2.6.5　プロパティの用途

プロパティには次のような用途があります。ユーザがクラスにプロパティを追加する場合は，どのように使用するかよく検討してから，必要なプロパティを定義してクラスを設計してください。

● 検索

● ユーザ管理

● データの格納

(1) 検索

オブジェクトを操作するとき，目的のオブジェクトは検索によって取得できます。プロパティは，検索を実行するときの条件として指定できます。

このため，ユーザがクラスを作成するときには，どのような検索をしたいかを考慮して，検索のキーになるようなプロパティを追加，設定してください。

「2.6.2　プロパティの種類」で説明した各プロパティの検索での特徴を，次の表に示します。

表 2-10 各プロパティの検索での特徴

指定する個所	DMA プロパティ	クラスライブラリ固有のプロパティ	ユーザ定義プロパティ
検索条件	○※1	×	○※1
検索結果	○※2	×	○※2
検索結果をソートする基準（キー）	○※3	×	○※3

（凡例）
○：指定できる。
×：指定できない。

注※ 1
定義ファイルの dmaProp_IsSearchable プロパティの値が「1」の場合

注※ 2
定義ファイルの dmaProp_IsSelectable プロパティの値が「1」の場合

注※ 3
定義ファイルの dmaProp_IsOrderable プロパティの値が「1」の場合

プロパティの特徴を定義した定義ファイルについては，マニュアル「DocumentBroker Version 3　システム導入・運用ガイド」を参照してください。

なお，文書に対する全文検索を実行する場合には，検索する文書に特定のプロパティが定義されている必要があります。このプロパティは，文書の基になるクラスにユーザが追加して定義してください。全文検索を実行する場合に追加する必要があるプロパティと追加する方法については，マニュアル「DocumentBroker Version 3　システム導入・運用ガイド」を参照してください。また，検索についての詳細は，「4.　オブジェクトの検索」を参照してください。

(2) ユーザ管理

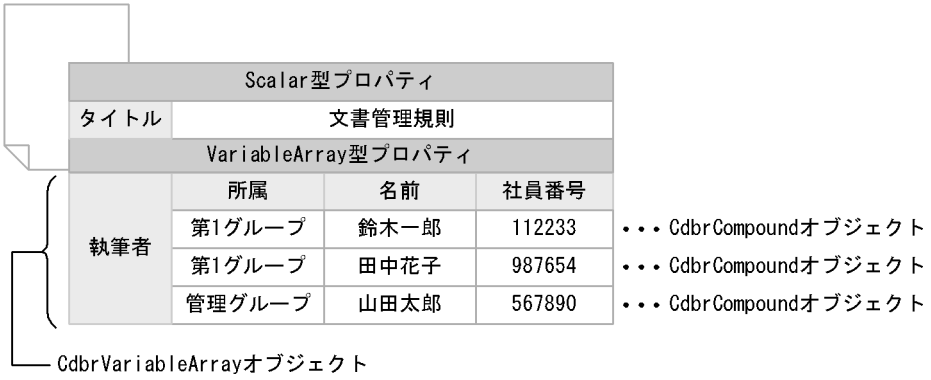
ユーザがオブジェクトの管理情報として保持したい情報を，プロパティとしてオブジェクトごとに設定できます。

(3) データの格納

可変長配列の値を持つ VariableArray 型プロパティは，データを格納するために使用できます。例えば，文書を表すオブジェクトに対して，「執筆者」プロパティを VariableArray 型プロパティとして設定すると，執筆者が複数いる場合も，それぞれの管理者の情報を可変長配列の要素として格納できます。

VariableArray 型プロパティの管理例を，次の図に示します。

図 2-5　VariableArray 型プロパティの管理例



クラスライブラリでは，VariableArray 型プロパティを CdbrVariableArray オブジェクト，VariableArray 型プロパティに設定される個々の要素を CdbrCompound オブジェクトとして作成，管理します。さらに，個々の要素には，CdbrCompound オブジェクトのプロパティとして，複数のプロパティが定義できます。

図の例では，「執筆者」プロパティを VariableArray 型プロパティとして設定しています。この場合，「執筆者」プロパティは CdbrVariableArray オブジェクトとして作成します。「執筆者」プロパティに設定する個々の要素のデータ（個々の執筆者に関するデータ）は，CdbrCompound オブジェクトとして作成します。

そして，CdbrCompound オブジェクトのプロパティとして，個人情報である「所属」，「名前」および「社員番号」を管理します。

CdbrVariableArray クラスおよび CdbrCompound クラスについての詳細は，「2.6.7　VariableArray 型プロパティの操作」を参照してください。

また，ここで設定した VariableArray 型プロパティの要素のプロパティの値も，検索条件として指定できます。例えば，図 2-5 の場合に，「執筆者の名前に『田中花子』が存在する文書を検索して，その文書の『タイトル』を取得する」といった検索条件が指定できます。

2.6.6　プロパティの操作

ここでは，プロパティの操作について説明します。

(1) プロパティの操作で使用する構造体

プロパティの操作には，**プロパティ構造体**（SDBR_PROP 構造体）および**プロパティリスト構造体**（SDBR_PROPLIST 構造体）を使用します。

プロパティ構造体のメンバは，プロパティ識別子，プロパティのデータ型，基本単位，基本単位ごとに決まった要素数，および設定するプロパティの値です。複数のプロパティの値を設定する場合は，プロパティ構造体を配列で作成します。プロパティリスト構造体には，プロパティ構造体の数を指定します。

プロパティに値を設定する場合は，まずプロパティ構造体およびプロパティリスト構造体に値を設定して，その構造体を引数に指定してメソッドをコールします。また，プロパティの値を取得する場合は，これらの構造体を出力の引数に指定してメソッドをコールします。

次に，プロパティの値を設定する場合の，プロパティ構造体およびプロパティリスト構造体の指定例を示します。ここでは，ユーザ定義プロパティ usrProp_Number に 25 を設定するためのプロパティ構造体お

よびプロパティリスト構造体を作成します。

プロパティ構造体およびプロパティリスト構造体の指定例

```
SDBR_PROP Props;
SDBR_PROPLIST Proplist;
DmaInteger32 lNum;

lNum=25L;
Props.PropId=usrProp_Number;
Props.lType=DMA_DATATYPE_INTEGER32;
Props.lCardinality=DMA_CARDINALITY_SINGLE;
Props.lCount=1;
Props.uniValue.plInteger32=&lNum;
Proplist.lCount=1;
Proplist.pItem=&Props;
```

(2) プロパティの初期値の設定

プロパティの初期値は，CreateObject メソッドによってオブジェクトを作成するときに設定されます。このときに設定されるプロパティには，ユーザが作成時に明示的に値を指定するプロパティと，システムによって値が設定されるプロパティがあります。

ユーザが初期値を設定する場合，CreateObject メソッドの引数に指定する SDBR_DMAINFO 構造体のメンバとして，プロパティリスト構造体のアドレスを指定します。

(3) プロパティの更新

ここでは，プロパティの更新について説明します。

(a) PutPropertyValues メソッドによる更新

プロパティの値は，CdbrDMA クラスで定義されている PutPropertyValues メソッドをコールして更新します。更新するプロパティの情報を設定したプロパティ構造体とプロパティリスト構造体を，メソッドの引数に指定します。

プロパティを更新するコールシーケンスの例を次に示します。

プロパティを更新するコールシーケンス

```
pSession->Begin();
CdbrDocument ObjDoc;
SDBR_PROP Prop;
SDBR_PROPLIST Proplist;
//Prop，Proplistに値を設定しておく
//…
//ObjDocにOIIDを設定する
ObjDoc.SetOIID(pSession,pOIID);
//プロパティを更新する
ObjDoc.PutPropertyValues(&Proplist);
ObjDoc.ReleaseObject();
pSession->Commit();
```

なお，次に示すクラスでも，引数が異なる PutPropertyValues メソッドが提供されています。

● **CdbrVersionable クラス**

特定のバージョンのプロパティを更新するために，引数にバージョン識別子を指定する形式があります。

(b) PutPropertyValues メソッド以外のメソッドによるプロパティの更新

次のプロパティを更新する場合には，PutPropertyValues メソッド以外のメソッドを使用します。

● **レンディションのプロパティ（サブレンディションの場合）**

CdbrDocument クラスまたは CdbrVersionableDocument クラスの PutRenditionPropertyValues メソッドを使用します。

● **リレーションのプロパティ**
CdbrDocument クラスおよび CdbrVersionableDocument クラスの PutRelationPropertyValues メソッドを使用します。

● **コンテインメントのプロパティ**
CdbrReferentialContainer クラス，CdbrVersionTraceableContainer クラスまたは CdbrConfiguratedReferentialContainer クラスの PutLinkPropertyValues メソッドを使用します。

レンディション，リレーションおよびコンテインメントについては，「3. クラスライブラリで実現する文書管理」を参照してください。また，それぞれのメソッドの詳細については，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。

(4) プロパティの取得

ここではプロパティの取得について説明します。

(a) GetPropertyValuesAndLock メソッドおよび GetPropertyValues メソッドによるプロパティの取得

オブジェクトのプロパティに設定されている値を取得するには，CdbrDMA クラスで定義されている GetPropertyValuesAndLock メソッドまたは GetPropertyValues メソッドをコールします。GetPropertyValuesAndLock メソッドをコールすると，プロパティの取得と同時に処理に必要なロックを設定できます。

これらのメソッドの引数には，プロパティ定義情報構造体（SDBR_PROPDEF 構造体）を指定します。プロパティ定義情報構造体には，値を取得するプロパティのプロパティ識別子を指定します。取得した値は，プロパティ構造体およびプロパティリスト構造体に出力されます。

プロパティを取得するコールシーケンスの例を次に示します。この例では，usrProp_Number プロパティと usrProp_Author プロパティの値を取得します。

プロパティを取得するコールシーケンス

```
CdbrDocument ObjDoc;
pSession->Begin();
SDBR_PROPDEF   PropDef[2];
SDBR_PROPLIST* pPropList = NULL;
SDBR_PROP*     pProp = NULL;
//PropDef構造体にプロパティ識別子を設定する
PropDef[0].PropId = usrProp_Number;
PropDef[1].PropId = usrProp_Author;
//ObjDocにOIIDを設定する
ObjDoc.SetOIID(pSession,pOIID);
//プロパティを取得する
ObjDoc.GetPropertyValuesAndLock(2, PropDef, &pPropList,
                                DMA_LOCK_READ);
ObjDoc.ReleaseObject();
pSession->Commit();
```

なお，次に示すクラスでも，引数が異なる GetPropertyValuesAndLock メソッドおよび GetPropertyValues メソッドが提供されています。

● **CdbrVersionable クラス**
特定のバージョンのプロパティを取得するために，引数にバージョン識別子を指定する形式があります。

(b) GetPropertyValues メソッド以外のメソッドによるプロパティの取得

GetPropertyValuesAndLock メソッドおよび GetPropertyValues メソッド以外に，次のメソッドでプロパティを取得することもできます。

● 一覧を取得するメソッド

一覧を取得するメソッドをコールするとき，一覧とともに一覧として取得したオブジェクトに設定されたプロパティも取得できます。例えば，コンテナと関連づけられている文書の一覧を取得する場合に，「コンテナと関連づけられている文書の一覧を取得すると同時に，それらの文書の作成者を表すプロパティも取得する」といった指定ができます。

一覧を取得するメソッドとは，それぞれのクラスで定義されている，Get ～ List メソッドおよび Get ～ ListAndLock メソッドのことです。このうち，同時にプロパティが取得できるのは，引数としてプロパティ定義情報構造体（SDBR_PROPDEF 構造体）が指定できるものです。プロパティ定義情報構造体に取得したいプロパティのプロパティ識別子を指定します。

詳細については，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。

また，次のプロパティを取得する場合は，GetPropertyValues メソッド以外のメソッドを使用します。

● レンディションのプロパティ（サブレンディションの場合）

CdbrDocument クラスまたは CdbrVersionableDocument クラスの GetRenditionListAndLock メソッドまたは GetRenditionList メソッドを使用します。

● リレーションのプロパティ

CdbrDocument クラスおよび CdbrVersionableDocument クラスの GetRelationListAndLock メソッドまたは GetRelationList メソッドを使用します。

● コンテインメントのプロパティ

CdbrReferentialContainer クラス，CdbrVersionTraceableContainer クラスまたは CdbrConfiguratedReferentialContainer クラスの GetLinkPropertyValuesAndLock メソッドまたは GetLinkPropertyValues メソッドを使用します。

レンディション，リレーションおよびコンテインメントについては，「3.　クラスライブラリで実現する文書管理」を参照してください。また，それぞれのメソッドの詳細については，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。

2.6.7　VariableArray 型プロパティの操作

VariableArray **型プロパティ**は，データ型が Object 型で，基本単位が VariableArray 型のプロパティです。VariableArray 型プロパティの値は，複数の要素から構成される可変長配列です。

ここでは，VariableArray 型プロパティの値である可変長配列の操作方法および VariableArray 型プロパティの操作方法について説明します。

(1) VariableArray 型プロパティを操作するクラス

ここでは，VariableArray 型プロパティを操作するクラスである CdbrVariableArray クラスと CdbrCompound クラスについて説明します。

(a) CdbrVariableArray クラス

CdbrVariableArray クラスは，可変長配列全体のデータを格納するために使用するクラスです。

配列の各要素は，0 ～（要素数 -1）のインデクスで指定します。

要素数は，次のどちらかの方法で変更します。

● **Resize メソッドで要素数を変更する**
このメソッドでは，配列の末尾の要素が追加，削除されるので，要素の数を増減しても，各要素のインデクスは変わりません。

● **Add メソッド，Delete メソッドで要素を追加，削除する**
Add メソッドでは，配列の末尾に要素を追加されるので，各要素のインデクスは変わりません。
Delete メソッドで配列中の要素を削除した場合は，削除した位置以降の要素が前に詰まるので，各要素のインデクスが変わります。

また，VariableArray 型プロパティを参照，設定する場合，CdbrCompound オブジェクトだけが要素として使用できます。

CdbrVariableArray クラスで使用するメソッドについては，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。

なお，このクラスに対応する DMA のクラスはありません。ただし，VariableArray 型プロパティは，データベースでは DMA オブジェクトの edmClass_VariableArray クラスのオブジェクトとして定義されています。したがって，CdbrVariableArray オブジェクトに設定した情報は，PutPropertyValues メソッドをコールした時に，DMA オブジェクトの VariableArray 型プロパティを表す edmClass_VariableArrayOfObject クラスのオブジェクトの情報としてコピーされます。

（b）CdbrCompound クラス

複合データを表すクラスです。複合データとは，複数の異なる型によって表されるデータです。VariableArray 型プロパティの要素の値を格納するために使用します。

各要素は，DmaId 型の識別子で指定します。この識別子は，CdbrCompound::SetValue メソッドをコールした時に設定したものです。可変長配列の操作では，この識別子は DMA オブジェクトの edmClass_Struct クラスのサブクラスに定義したプロパティのプロパティ識別子になります。

CdbrCompound クラスで使用するメソッドについては，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。

なお，このクラスには，対応する DMA のクラスはありません。ただし，VariableArray 型プロパティの要素は，データベースでは DMA オブジェクトの edmClass_Struct クラスのオブジェクトとして定義されています。したがって，CdbrCompound オブジェクトに設定した情報は，PutPropertyValues メソッドをコールした時に，DMA オブジェクトの VariableArray 型プロパティの要素を表す edmClass_Struct オブジェクトの情報としてコピーされます。

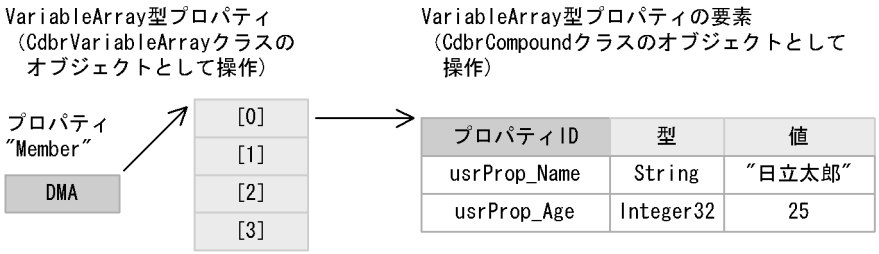
（2）VariableArray 型プロパティの操作手順

CdbrVariableArray クラスと CdbrCompound クラスを使用した VariableArray 型プロパティの操作手順を説明します。

ここでは，文書またはコンテナに定義された「Member」という VariableArray 型プロパティを例にして説明します。このプロパティは，文書やコンテナの管理者情報を格納するために使用します。管理者情報としては，「Name」と「Age」があります。

Member プロパティを次の図に示します。

図 2-6 VariableArray 型プロパティの例



プロパティ"Member"は，基本単位がVariableArray型で，四つの要素を持つ。
プロパティの各要素は"Name"，"Age"という二つのプロパティを持つ。

Member プロパティは，CdbrVariableArray クラスのオブジェクトとして作成します。
CdbrVariableArray オブジェクトは，管理者の人数分の CdbrCompound オブジェクトを要素として持っています。一人の管理者の情報は，一つの CdbrCompound オブジェクトに対応します。管理者のそれぞれの情報（名前：Name および年齢：Age）は，CdbrCompound オブジェクトのプロパティに対応します。

CdbrCompound オブジェクトのプロパティの操作は，CdbrCompound クラスのメソッドによって実行します。プロパティを設定した CdbrCompound オブジェクトは，CdbrVariableArray クラスのメソッドによって CdbrVariableArray オブジェクトの要素として設定することで，VariableArray 型プロパティの要素の一つになります。

CdbrVariableArray オブジェクトとして作成された可変長配列は，Object 型の値として扱います。プロパティの値として設定する場合は，プロパティ構造体（SDBR_PROP 構造体）に Object 型の値として設定します。VariableArray 型プロパティの値を設定する場合には，このプロパティ構造体を引数として PutPropertyValues メソッドをコールします。

(a) VariableArray 型プロパティ設定の手順

VariableArray 型プロパティを設定する手順を示します。

1. **複合データを表す** CdbrCompound **オブジェクトを作成し，値を設定します。**
   複合データの値は，CdbrCompound::SetValue メソッドをコールして設定します。
2. **可変長配列を表す** CdbrVariableArray **オブジェクトを作成し，可変長配列の要素数を設定します。**
   可変長配列の要素数は，CdbrVariableArray::Resize メソッドをコールして設定します。
3. CdbrCompound **オブジェクトを** CdbrVariableArray **オブジェクトの要素として設定します。**
   CdbrVariableArray::SetValue メソッドをコールします。
4. CdbrVariableArray **オブジェクトのアドレスをポインタ（**CdbrVariableArray* **型の変数）に格納します。**
5. 4. **のポインタのアドレスを** SDBR_PROP **構造体（プロパティ構造体）の** uniValue.ppObject **に設定します。**
   ここで指定するアドレスは，CdbrVariableArray オブジェクトのポインタではありませんので，注意してください。
   この操作によって，可変長配列のアドレスが uniValue.ppObject[0] に設定されます。
6. **５．の** SDBR_PROP **構造体を，**SDBR_PROPLIST **構造体（プロパティリスト構造体）に格納し** PutPropertyValues **メソッドで引数に指定して，メソッドをコールします。**

(b) VariableArray 型プロパティ取得の手順

VariableArray 型プロパティを取得する手順を示します。

1. GetPropertyValues **メソッドなどの，プロパティを取得するメソッドをコールします。**
   取得したプロパティ構造体の値のデータ型が Object 型（lType = DMA_DATATYPE_OBJECT）であり，かつ基本単位が VariableArray 型（lCardinality = EDM_DMA_CARDINALITY_VARRAY）である場合，メンバ uniValue.ppObject[0] に CdbrVariableArray クラスのオブジェクトへのポインタが格納されます。
2. CdbrVariableArray::GetValue **メソッドの引数に，可変長配列の要素のインデクスを指定して，メソッドをコールします。**
   VariableArray 型プロパティから取得した CdbrVariableArray オブジェクトの要素のデータ型は，複合データ型（DBR_DATATYPE_COMPOUND）です。CdbrCompound クラスの変数のアドレスを指定します。
3. **複合データ型からの値を，**CdbrCompound::GetValue **メソッドにプロパティ識別子を指定して，取得します。**

(3) VariableArray 型プロパティを操作するコールシーケンス

ここでは，VariableArray 型プロパティを操作する場合のコールシーケンスを示します。

(a) 新しく可変長配列を作成して VariableArray 型プロパティの値を設定する

可変長配列を作成して VariableArray 型プロパティの値を設定するコールシーケンスの例を次に示します。ここでは，Member プロパティの要素として，管理者「日立太郎」の情報を設定します。

可変長配列を作成して VariableArray **型プロパティの値を設定するコールシーケンス**

```
CdbrCompound  CompoValue;
CdbrVariableArray  VArray(DBR_DATATYPE_COMPOUND);
SDBR_PROP Prop;
SDBR_PROPLIST PropList;
Dmapv pObj;
//要素となる複合データの値を設定する
CompoValue.SetValue(&usrProp_Name, "日立太郎");
CompoValue.SetValue(&usrProp_Age, 25L);
//可変長配列の要素数を設定する
VArray.Resize(1);
//可変長配列の一つ目の要素にCompoValueの値を設定する
VArray.SetValue(0,CompoValue);
//可変長配列をプロパティ構造体に設定する
Prop.lType = DMA_DATATYPE_OBJECT;
Prop.lCardinality = EDM_DMA_CARDINALITY_VARRAY;
Prop.lCount = 1;
pObj = &VArray;
Prop.uniValue.ppObject = &Obj;
Proplist.pItem = &Prop;
Proplist.lCount = 1;
//プロパティをオブジェクトに設定する
ObjDoc.PutPropertyValues(&PropList);
ObjDoc.ReleaseObject();
pSession->Commit();
```

(b) 可変長配列の要素を追加および削除する

可変長配列の要素は，CdbrVariableArray::Add メソッドによって追加できます。追加した要素は，可変長配列の末尾に追加されます。

また，可変長配列の要素の削除は，CdbrVariableArray::Delete メソッドで，引数に削除する要素のイン

デクスを指定して実行します。削除された要素以降の要素は，前に詰まります。

可変長配列の要素数を追加および削除するコールシーケンスの例を次に示します。ここでは，Member プロパティの要素として，二人分の管理者情報を追加します。その後，先頭の要素を削除します。

要素を追加および削除するコールシーケンス

```
CdbrVariableArray  VArray(DBR_DATATYPE_COMPOUND);
//要素を作成する
CdbrCompound CompoValue1,CompoValue2;
CompoValue1.SetValue(&usrProp_Name, "鈴木一郎");
CompoValue1.SetValue(&usrProp_Age, 28L);
CompoValue2.SetValue(&usrProp_Name, "山田二郎");
CompoValue2.SetValue(&usrProp_Age, 35L);
//可変長配列に要素を二つ追加する
VArray.Add(CompoValue1);
VArray.Add(CompoValue2);
//先頭の要素を削除する
VArray.Delete(0);
//可変長配列をプロパティ構造体に設定する
Prop.lType = DMA_DATATYPE_OBJECT;
Prop.lCardinality = EDM_DMA_CARDINALITY_VARRAY;
Prop.lCount = 1;
pObj = &VArray;
Prop.uniValue.ppObject = &Obj;
Proplist.pItem = &Prop;
Proplist.lCount = 1;
//オブジェクトのプロパティを更新して
//要素の追加・削除をVariableArray型プロパティに
//反映させる
ObjDoc.PutPropertyValues(&Proplist);
ObjDoc.ReleaseObject();
pSession->Commit();
```

(c) 可変長配列のすべての要素を削除する

可変長配列のすべての要素を一括して削除するには，CdbrVariableArray::Resize メソッドによって要素数に 0 を指定します。要素数 0 の可変長配列を VariableArray 型プロパティに設定すると，VariableArray 型プロパティのすべての要素が削除されます。

すべての要素を削除するコールシーケンスの例を次に示します。

すべての要素を削除するコールシーケンス

```
CdbrDocument Obj;
CdbrVariableArray VArray(DBR_DATATYPE_COMPOUND);
SDBR_PROP Prop;
SDBR_PROPLIST PropList;
Dmapv pObj;

pSession->Begin();
Obj.SetOIID(pSession,pOIID);
//要素の数を0にする
pVArray->Resize(0);
//要素を削除した可変長配列をプロパティ構造体に設定する
//...
//オブジェクトのプロパティを更新する
//VariableArray型のプロパティの要素はすべて削除される
Obj.PutPropertyValues(&PropList);
Obj.ReleaseObject();
pSession->Commit();
```

(d) VariableArray 型プロパティの取得

VariableArray 型プロパティの値を取得する場合のコールシーケンスの例を次に示します。

VariableArray 型プロパティの値を取得するコールシーケンス

```
CdbrDocument ObjDoc;
SDBR_PROPDEF    Propdef;
SDBR_PROPLIST*  pPropList;
SDBR_PROP*      pProp;
pDmaString_T    pName;
DmaInteger32    lAge;
CdbrVariableArray* pVArray;
CdbrCompound       CompoValue;
Propdef.PropId = usrProp_Member;
pSession->Begin();
ObjDoc.SetOIID(pSession, pOIID);
ObjDoc.GetPropertyValuesAndLock(1,&Propdef,&pPropList,
                                DMA_LOCK_WRITE);
//プロパティの型の記号定数がDMA_DATATYPE_OBJECTであり
//プロパティの基本単位がEDM_DMA_CARDINALITY_VARRAYである
//要素を判定し，型を取得する
//...
pVArray = (CdbrVariableArray*) pProp->uniValue.ppObject[0];
//要素の型がDMA_DATATYPE_COMPOUNDである
//複合データを判定する
//...
//値を取得する
pVArray->GetValue(0,&CompoValue);
//usrProp_Nameプロパティの値を取得する
CompoValue.GetValue(&usrProp_Name, &pName);
//usrProp_Ageプロパティの値を取得する
CompoValue.GetValue(&usrProp_Age, &lAge);
ObjDoc.ReleaseObject();
pSession->Commit();
```

2.7 オブジェクトとデータベースの関係

クラスライブラリで操作するオブジェクトとデータベースの関係について説明します。クラスライブラリのオブジェクトは複数の DMA オブジェクトから構成されています。ここでは，クラスライブラリのオブジェクトと DMA オブジェクト，および対応するデータベース構成について説明します。

クラスライブラリのオブジェクトは，メモリ上に存在するオブジェクトです。データベースは DMA のオブジェクトモデルと対応して構成されています。通常，クラスライブラリのオブジェクトに対してメソッドを発行すると，複数の DMA オブジェクトへのメソッドとして配置され，該当するデータベースの表が操作されます。

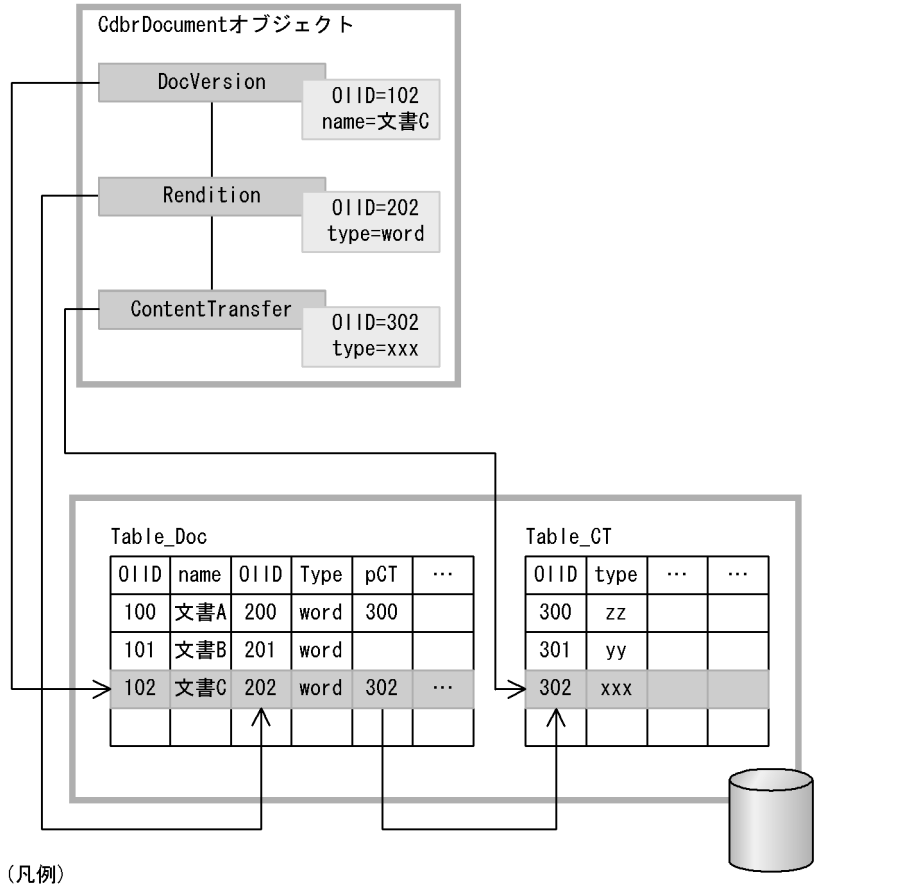
クラスライブラリのメソッドと，それによって操作されるデータベースの関係を表す例を次の図に示します。なお，この例で使用する CdbrDocument オブジェクトの構成は，次のとおりです。

CdbrDocument オブジェクト（クラスライブラリのオブジェクト）

次の DMA のオブジェクトを包含しています。

- DocVersion オブジェクト（dmaClass_DocVersion クラスのオブジェクト）
- Rendition オブジェクト（dmaClass_Rendition クラスのオブジェクト）
- ContentTransfer オブジェクト（dmaClass_ContentTransfer クラスのオブジェクト）

図 2-7 クラスライブラリのオブジェクト，DMA のオブジェクトおよび HiRDB の表の例



（凡例）
Table_Doc：DocVersionオブジェクトを格納する表。dmaClass_DocVersionクラスに対応する。
Table_CT ：ContentTransferオブジェクトを格納する表。
dmaClass_ContentTransferクラスに対応する。

オブジェクトを生成する DMA のクラスは，それぞれが HiRDB の一つの表と対応します※。オブジェクトは各表の行（レコード）に相当します。各 DMA クラスが持つプロパティのうち，永続プロパティは各表の列（カラム）に相当します。

DMA オブジェクト同士の関連は，各オブジェクトが属性として保持する参照先の OIID によって表現されます。この場合，DocVersion オブジェクトは，ContentTransfer オブジェクトの OIID（302：実際とは異なる値です）を保持することで，参照先の Rendition オブジェクトを特定します。

注※

ただし，一つの DocVersion オブジェクトに対応する Rendition オブジェクトが一つしか存在しない場合，Rendition オブジェクトは，dmaClass_DocVersion クラスの表に含まれます。このため，図 2-7 では Rendition オブジェクトは DocVersion オブジェクトと同じ表に含まれています。

DMA オブジェクトとデータベースの関係の詳細については，マニュアル「DocumentBroker Version 3 システム導入・運用ガイド」を参照してください。

3 クラスライブラリで実現する文書管理

この章では，DocumentBroker のクラスライブラリの機能で実現できる文書管理について説明します。

3.1 文書管理で使用するオブジェクトと管理方法

DocumentBroker による文書管理で使用するオブジェクトと，そのオブジェクトを使用した管理方法について説明します。

DocumentBroker の文書管理では，文書とコンテナというオブジェクトを使用します。文書およびコンテナを表すオブジェクトには複数の種類があり，それぞれに特長があります。DocumentBroker によって文書管理を実現するためには，まず，それぞれのオブジェクトの特長を理解して，実現したい業務に合ったオブジェクトを使用する必要があります。

DocumentBroker で扱う文書およびコンテナのオブジェクトの種類を，次の表に示します。

表 3-1 DocumentBroker で扱う文書およびコンテナ

文書とコンテナの種類		クラスライブラリのオブジェクト	オブジェクト名
文書		バージョンなし文書	CdbrDocument
		バージョン付き文書	CdbrVersionableDocument
コンテナ	構成管理機能がないコンテナ	バージョンなしコンテナ	CdbrReferentialContainer
	構成管理機能があるコンテナ（構成管理コンテナ）	バージョンなし構成管理コンテナ	CdbrVersionTraceableContainer
		バージョン付き構成管理コンテナ	CdbrConfiguratedReferentialContainer

なお，それぞれのオブジェクトをまとめて，次のような総称で表記することがあります。

文書
「バージョンなし文書」と「バージョン付き文書」の総称として使用します。

コンテナ
「バージョンなしコンテナ」,「バージョンなし構成管理コンテナ」および「バージョン付き構成管理コンテナ」の総称として使用します。

構成管理コンテナ
「バージョンなし構成管理コンテナ」と「バージョン付き構成管理コンテナ」の総称として使用します。

また，バージョンなし文書およびバージョン付き文書には，管理するコンテントの種別によって，次の種類があります。

シングルファイル文書
マルチファイル管理機能，リファレンスファイル管理機能および File Link 連携機能を使用しないコンテントを管理している文書のことです。
この文書では，データベースに登録されている一つのファイルをコンテントとして管理しています。

マルチファイル文書
マルチファイル管理機能を使用しているコンテントを管理している文書のことです。
この文書では，データベースに登録されている複数のファイルをコンテントとして管理しています。

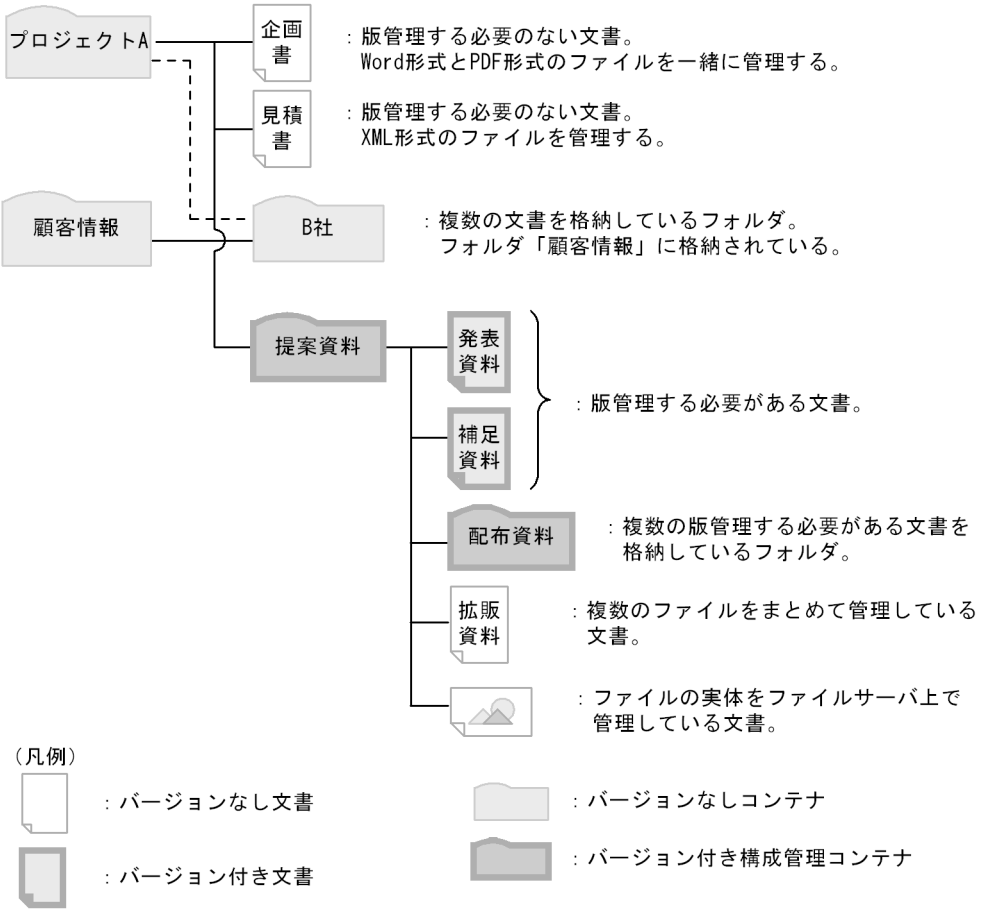
リファレンスファイル文書
リファレンスファイル管理機能を使用しているコンテントを管理している文書のことです。
この文書では，任意のディレクトリに登録されているコンテントを管理しています。データベースでは，そのコンテントロケーションを管理しています。

File Link **文書**

File Link 連携機能を使用しているコンテントを管理している文書のことです。
この文書では，HiRDB File Link のファイルサーバ上にある一つのファイル，または一つのディレクトリおよびそのサブディレクトリ下のすべてのファイルをコンテントとして管理しています。データベースではそのコンテントへのリンク情報を管理しています。

この節では，これらのオブジェクトを利用して実現できる文書管理の概要について説明します。ここでは，次のように文書を管理しているプロジェクトを例にして説明します。

図 3-1　文書管理の例



3.1.1　文書（バージョンなし文書）

ここでは，バージョンなし文書を使用した管理について説明します。

プロジェクト A では，企画書および見積書を，バージョンなし文書として管理しています。

バージョンなし文書は，一つのバージョンとだけ対応する文書です。版管理をして更新履歴などを残す必要がない（バージョン管理をする必要がない）文書を管理する場合に使用します。

バージョンなし文書は，CdbrDocument クラスのオブジェクトとして作成します。詳細については，「3.2　クラスライブラリで扱う文書」を参照してください。

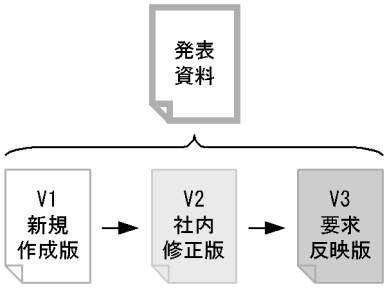
3.1.2 文書（バージョン付き文書）

ここでは，バージョン付き文書を使用した管理について説明します。

プロジェクト A では，発表資料や補足資料をそれぞれバージョン付き文書として管理しています。例えば，発表資料の場合，担当者が作成した資料をバージョン 1，顧客提案前に社内で検討，修正した資料をバージョン 2，提案後，要求を反映して更新した資料をバージョン 3 のようにして管理することができます。

バージョンを管理しているバージョン付き文書の例を次の図に示します。

図 3-2 バージョンを管理しているバージョン付き文書の例



バージョン付き文書は，CdbrVersionableDocument クラスのオブジェクトとして作成します。

なお，バージョン付き文書の個々のバージョンは，バージョンなし文書と対応します。図 3-2 の場合，発表資料を構成する，V1 新規作成版，V2 社内修正版，および V3 要求反映版はバージョンなし文書です。これらを直接 CdbrDocument クラスのオブジェクトとして扱うこともできます。

詳細については，「3.2 クラスライブラリで扱う文書」および「3.3 文書のバージョン管理」を参照してください。

3.1.3 マルチレンディション文書

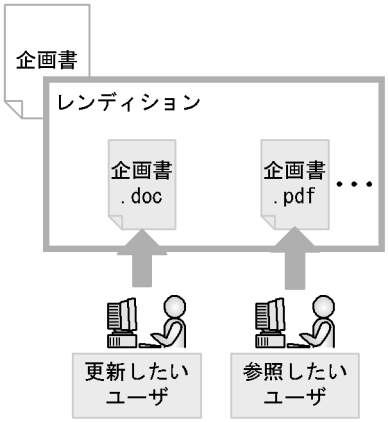
ここでは，文書のマルチレンディション管理について説明します。

文書の実体であるコンテント（Word やテキストエディタなどのアプリケーションプログラムで作成されたファイル）と，その形式を表すレンディションタイプ（MIME 名）の情報を合わせて**レンディション**といいます。DocumentBroker では，一つの文書に同じ内容を表す複数の異なる形式のレンディションを登録することができます。この機能を**マルチレンディション機能**といいます。また，複数のレンディションを登録した文書を特に**マルチレンディション文書**といいます。

例えば，プロジェクト A では，企画書をマルチレンディション文書として管理しています。企画書には，作成者が作成した Word 形式の「企画書 .doc」と，プロジェクトにかかわるメンバが参照する PDF 形式の「企画書 .pdf」を一つの文書として登録します。これによって，必要に応じて，Word 形式のファイルをダウンロードしたり，PDF 形式のファイルをダウンロードしたりすることが可能になります。また，アクセス制御機能と組み合わせたユーザプログラムを作成すると，更新する権利を持つユーザには Word 形式のファイルを，参照する権利しか持たないユーザには PDF 形式のファイルを自動的にダウンロードさせるという運用もできます。

マルチレンディション機能を使用した管理の例を次の図に示します。

図 3-3　マルチレンディション機能を使用した管理の例



DocumentBroker では，一つの文書に最大 10 個のレンディションを登録することができます。

また，DocumentBroker Rendering Option と連携すると，ユーザが登録した形式のファイルからほかの形式のファイルを作成することもできます。例えば，登録した企画書 .doc から，DocumentBroker Rendering Option を使用して，企画書 .pdf を作成することができます。

マルチレンディション文書は，CdbrVersionableDocument クラスのオブジェクトまたは CdbrDocument クラスのオブジェクトとして作成します。詳細については，「3.4　文書のマルチレンディション管理」を参照してください。

3.1.4　マルチファイル文書

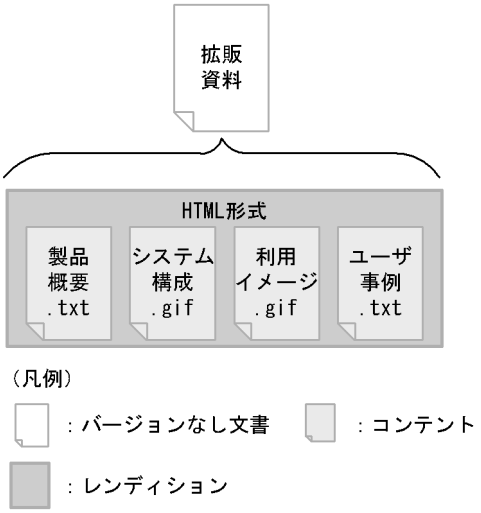
ここでは，文書のマルチファイル管理について説明します。

マルチファイル管理とは，一つの文書に複数のファイルを登録して管理する機能です。複数のファイルが登録された文書を，**マルチファイル文書**といいます。一つの文書で複数のファイルを管理すると，文書を構成する複数のファイルを，一括して登録したり，一括して取得したりできます。この機能を，**マルチファイル管理機能**といいます。

DocumentBroker は，バージョンなし文書またはバージョン付き文書をマルチファイル文書として管理できます。

マルチファイルの管理例を次の図に示します。次の図では，拡販資料をマルチファイル文書として管理しています。拡販資料では，「製品概要 .txt」，「システム構成 .gif」，「利用イメージ .gif」および「ユーザ事例 .txt」という四つのファイルを，HTML 形式のレンディションのコンテントとして管理しています。

図 3-4　マルチファイルの管理例



マルチファイル文書は，CdbrDocument クラスのオブジェクトまたは CdbrVersionableDocument クラスのオブジェクトとして作成します。

なお，マルチファイルを管理するバージョンなし文書およびバージョン付き文書では，マルチレンディション管理機能を使用できません。

詳細については，「3.5　文書のマルチファイル管理」を参照してください。

3.1.5　リファレンスファイル文書

ここでは，リファレンスファイル文書について説明します。

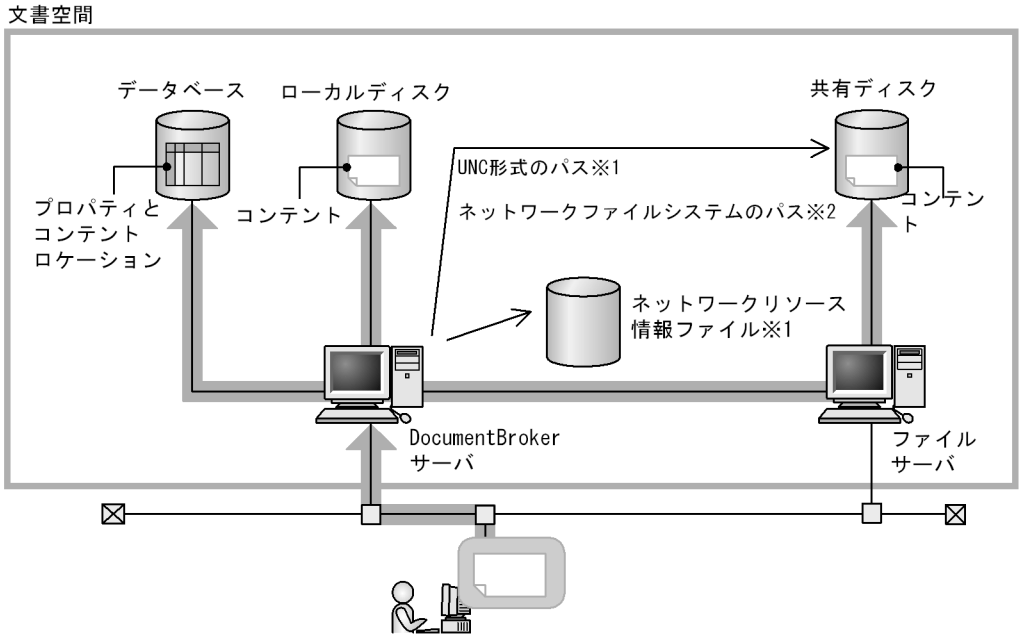
リファレンスファイル管理機能は，コンテントを任意のディレクトリに格納し，文書のプロパティとコンテントロケーションだけをデータベースで管理する機能です。データベースとコンテントを切り離すことによって，ユーザによるコンテント格納先の選択やディスク移行時のコンテント格納ディレクトリの変更など，柔軟にコンテント格納先の管理ができます。また，Windows の場合は UNC 形式のパスでコンテントを操作することもできます。

リファレンスファイル管理機能を使用して管理している文書を，**リファレンスファイル文書**といいます。

DocumentBroker では，バージョンなし文書またはバージョン付き文書をリファレンスファイル文書として管理できます。

リファレンスファイル文書の管理例を，次の図に示します。

図 3-5　リファレンスファイル文書の管理例



注※1　コンテント格納先ベースパスにUNC形式のパスを使用した場合です。（Windows）
注※2　コンテント格納先ベースパスにネットワークファイルシステムのパスを使用した場合です。（UNIX）

詳細については，「3.6　文書のリファレンスファイル管理」を参照してください。

3.1.6　File Link 文書

ここでは，File Link 文書について説明します。

File Link 連携機能は，HiRDB File Link と連携して文書を管理する機能です。

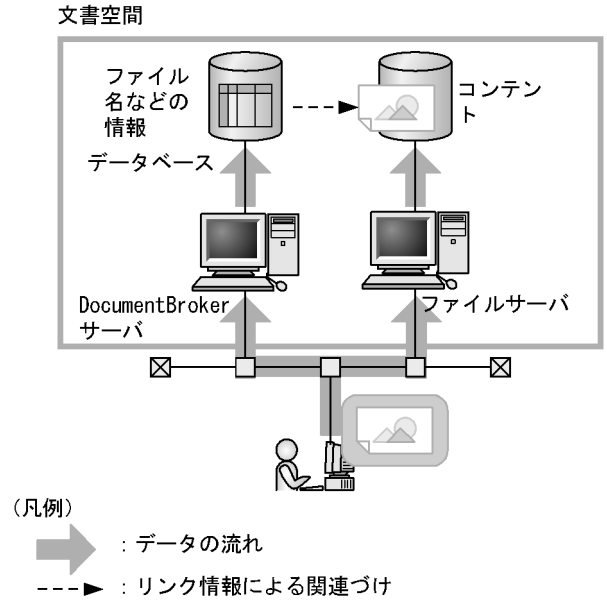
画像・音声などの容量が大きいコンテントを持つファイルをファイルサーバで管理したい場合にこの機能を使用すると，データベースで管理しているファイルの作成者名などの情報とファイルサーバ上のファイルとの整合性を確保できます。クライアントアプリケーションで，データベースとファイルサーバの整合性を取るための処理をする必要はありません。

File Link 連携機能を使用して管理している文書を，**File Link 文書**といいます。

DocumentBroker では，バージョンなし文書またはバージョン付き文書を File Link 文書として管理できます。

File Link 文書の管理例を，次の図に示します。

図 3-6 File Link 文書の管理例



詳細については，「3.7 File Link 連携機能を使用した文書管理」を参照してください。

3.1.7 文書間リレーション

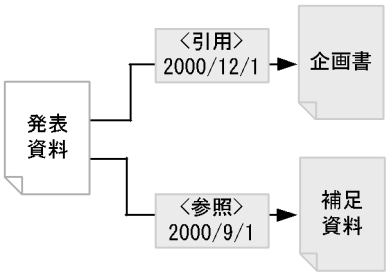
ここでは，文書間リレーションについて説明します。

DocumentBroker では，文書と文書を関連づけて管理できます。この関連づけを，**文書間リレーション**といいます。

例えば，プロジェクト A の発表資料で一部企画書を引用している場合や，発表資料から補足資料を参照している場合などに，発表資料と企画書，発表資料と補足資料を，それぞれ文書間リレーションで関連づけて管理できます。これらを関連づけておくことによって，必要な資料をまとめて管理できます。また，引用後に企画書が更新された場合や，参照している補足資料が削除された場合，発表資料から確認できます。

文書間リレーションの設定例を次の図に示します。

図 3-7 文書間リレーションの設定例



文書間リレーションは，次のクラスのオブジェクト間に設定できます。

- CdbrDocument クラス
- CdbrVersionableDocument クラス

詳細については，「3.8 文書間リレーションを設定した文書管理」を参照してください。

3.1.8　XML 文書管理

ここでは，XML 文書管理について説明します。

DocumentBroker では，登録する文書が XML 形式で記述されたファイル（XML ファイル）である場合，次のような機能が使用できます。

- XML プロパティマッピング機能
- XML インデクスデータ作成機能

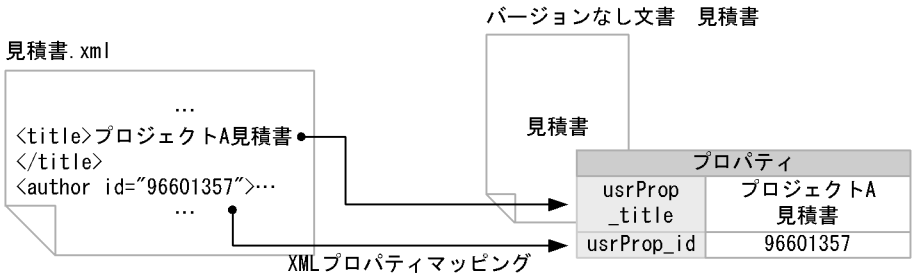
XML プロパティマッピング機能とは，XML ファイル内でタグ間に記述されている情報を，DocumentBroker の文書のプロパティにマッピングする機能です。

例えば，プロジェクト A の見積書は，次のような内容の XML ファイルです。

```
<?xml version="1.0" encoding="Shift_JIS"?>
<doc>
<title>プロジェクトA見積書</title>
<date>2001.1.10</date>
<author id="96601357">
  <name>日立太郎</name>
  <organization>営業部</organization>
</author>
<content>
…
</content>
</doc>
```

このとき，<title> タグ間の情報「プロジェクト A 見積書」や，<author> タグの属性値「96601357」などを，文書のプロパティとしてマッピングできます。

図 3-8　XML プロパティマッピングの例



XML インデクスデータ作成機能は，XML 文書の構造を指定した全文検索（構造指定検索）を実行するためのインデクスデータを，XML ファイルを構文解析して作成する機能です。作成したインデクスデータを全文検索インデクスとして文書に登録すると，見積書を検索する場合に，「構造『title』に『プロジェクト A』という文字列が含まれ，構造『author』の中の構造『name』に『日立』という文字列が含まれる文書」という条件や，「構造『author』に設定されている属性『id』の属性値が『96601357』である文書」という条件を指定して検索できます。また，構造指定検索用の全文検索インデクスのほかにも，XML ファイルからタグ情報を削除したプレーンテキスト形式の全文検索インデクスを作成するためのインデクスデータも作成できます。

XML 文書は，次のどちらかのクラスのオブジェクトとして作成します。

- CdbrDocument クラス
- CdbrVersionableDocument クラス

詳細については，「3.11　XML 文書の管理」を参照してください。

3.1.9 コンテナ

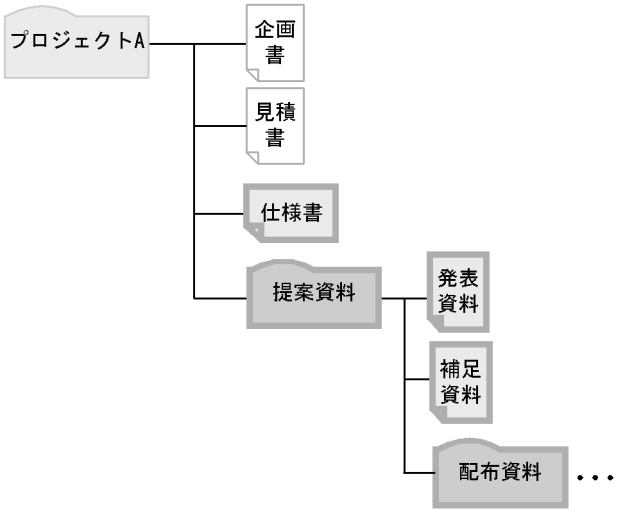
ここでは，コンテナを使用した管理方法について説明します。ここで説明するのは，すべてのコンテナで共通して実現できる管理方法です。

複数の文書を管理する場合，文書をフォルダに格納するイメージでまとめて管理すると便利です。また，複数の観点で分類したい文書などは，分類ごとにリンクを付けておくと便利です。DocumentBroker では，このような管理をコンテナによって実現します。

コンテナには，複数の文書またはコンテナを関連づけられます。コンテナから設定する関連づけをコンテインメントといいます。すべてのコンテナで使用できるコンテインメントには，直接型のコンテインメントと参照型のコンテインメントがあります。直接型のコンテインメントは，一つのコンテナの下位に複数の文書またはコンテナを関連づけられるコンテインメントです。上位になるコンテナは一つです。参照型のコンテインメントは，一つのコンテナの下位に複数の文書またはコンテナを関連づけられ，かつ一つの文書またはコンテナの上位に複数のコンテナを関連づけらるコンテインメントです。

例えば，プロジェクト単位で文書をまとめて管理するために，プロジェクト A コンテナをフォルダとして使用する場合，コンテナと資料を直接型のコンテインメントを使用して関連づけます。これによって，フォルダに資料を格納するイメージで見積書，企画書および仕様書を管理できます。プロジェクト A －提案資料－発表資料，とコンテナの階層をたどって文書を入手することもできます。

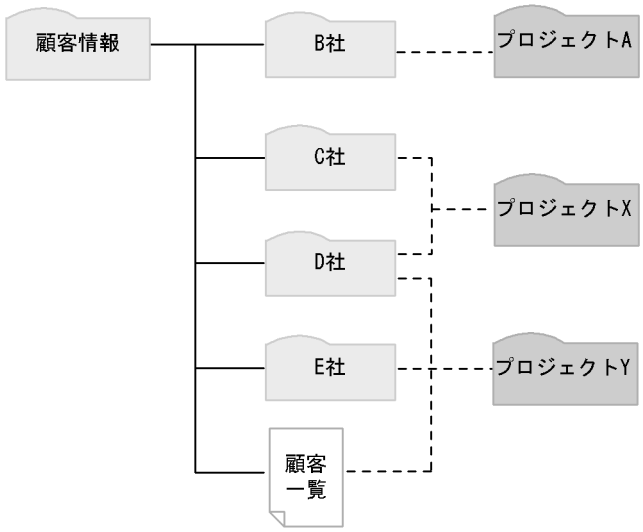
図 3-9　コンテナによる管理の例（直接型のコンテインメントで関連づける場合）



また，プロジェクト A に関連する資料のうち，B 社に関する資料を格納した B 社コンテナは，すでに顧客情報コンテナから直接型のコンテインメントで関連づけられています。一つの文書またはコンテナに対して，二つのコンテナから直接型のコンテインメントで関連づけることはできないため，B 社コンテナをプロジェクト A から直接型のコンテインメントで関連づけることはできません。

このため，プロジェクト A コンテナからは，参照型のコンテインメントを使用して関連づけます。参照型のコンテインメントを使用した場合，複数のプロジェクトから，同じ顧客情報を参照することもできます。

図 3-10　コンテナによる管理の例（参照型のコンテインメントで関連づける場合）



（凡例）

——：直接型のコンテインメントを使用した関連付け

- - -：参照型のコンテインメントを使用した関連付け

このほか，プロジェクトごとのコンテナに関連づけられている文書を作成者別に分類したり，顧客情報コンテナに関連づけられている会社ごとのコンテナを業界別に分類したりする場合にも，コンテナと参照型のコンテインメントを使用して，文書やコンテナを分類できます。

直接型または参照型のコンテインメントを使用できるコンテナは，次のクラスのオブジェクトとして作成します。

- CdbrReferentialContainer クラス
- CdbrVersionTraceableContainer クラス
- CdbrConfiguratedReferentialContainer クラス

詳細については，「3.9　コンテナを使用した文書管理」を参照してください。

3.1.10　構成管理コンテナ

ここでは，構成管理コンテナを使用した管理方法について説明します。

構成管理コンテナでは，関連づけている文書やコンテナがバージョンを持っている場合に，バージョンを特定して文書やコンテナを管理できます。

構成管理コンテナを使用すると，関連づける文書やコンテナに対して，常に最新のバージョンを管理するか，ある特定のバージョンを管理し続けるかを指定できます。常に最新のバージョンを管理するモードを FLOATING モード，特定のバージョンを管理し続けるモードを FIX モードといいます。

構成管理コンテナには，バージョン付き構成管理コンテナとバージョンなし構成管理コンテナがあります。バージョンなし構成管理コンテナは，バージョン付き構成管理コンテナの 1 バージョンに対応します。また，構成管理コンテナ自体をバージョンアップしない場合は，バージョンなし構成管理コンテナを使用できます。

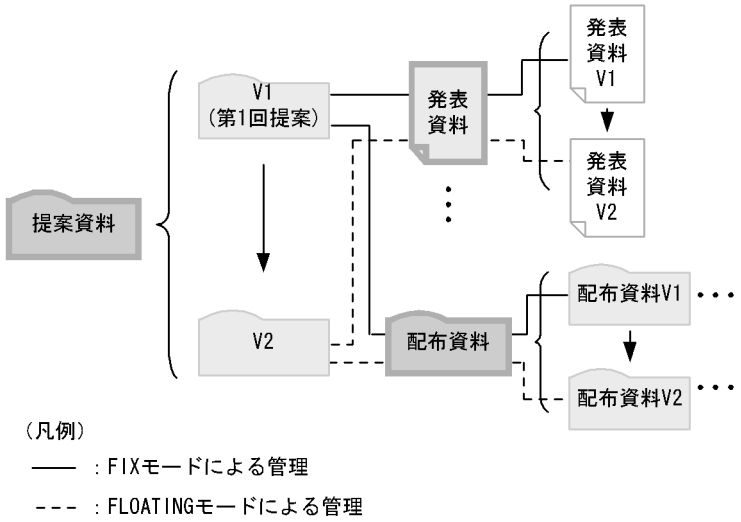
バージョン付き構成管理コンテナを使用すると，構成管理コンテナのバージョンと管理するモードを組み合わせて，関連づける文書やコンテナの構成管理をすることができます。

例えば，プロジェクト A の提案資料コンテナはバージョン付き構成管理コンテナです。関連づけている発

表資料，補足資料はバージョン付き文書，配布資料はバージョン付き構成管理コンテナです。提案資料コンテナでは，第 1 回目の提案をした段階の資料をコンテナのバージョン 1，それ以降に変更した資料をコンテナのバージョン 2 で管理しています。提案資料コンテナのバージョン 1 で資料を FIX モードで管理することによって，第 1 回目の提案以降に資料がバージョンアップした場合も，常に第 1 回提案時点の状態の資料を参照できます。提案資料コンテナのバージョン 2 では資料を FLOATING モードで管理することによって，常に各資料の最新バージョンを参照できます。

構成管理コンテナによる構成管理の例を次の図に示します。

図 3-11　構成管理コンテナによる構成管理の例



同様に，第 2 回目の提案を実施してその状態を管理しておきたい場合は，バージョン 2 のコンテナの管理モードを FIX モードに変更すれば，第 2 回目提案時点の状態の資料を管理し続けることができます。この場合，提案資料コンテナをバージョンアップして，バージョン 3 の提案資料コンテナで資料を FLOATING モードで管理すれば，そのコンテナをたどることで最新の資料を参照できます。

バージョン付き構成管理コンテナは，CdbrConfiguratedReferentialContainer クラスのオブジェクトとして作成します。バージョンなし構成管理コンテナは，CdbrVersionTraceableContainer クラスのオブジェクトとして作成します。

また，構成管理コンテナによる構成管理の対象になるのは次のオブジェクトです。

- CdbrVersionableDocument オブジェクト（バージョン付き文書）
- CdbrConfiguratedReferentialContainer オブジェクト（バージョン付き構成管理コンテナ）

詳細については，「3.10　構成管理コンテナを使用した文書の構成管理」を参照してください。

3.1.11　独立データ

DocumentBroker では，独立データも管理できます。独立データとは，ほかのオブジェクトに従属しない，独立したオブジェクトです。詳細については，「3.13　独立データの管理」を参照してください。

3.2　クラスライブラリで扱う文書

この節では，バージョンなし文書とバージョン付き文書について説明します。ただし，バージョン付き文書のバージョン管理の方法については，「3.3　文書のバージョン管理」で説明します。

バージョンなし文書は，CdbrDocument クラスを基にしたオブジェクト（CdbrDocument オブジェクト）として作成します。

バージョン付き文書は，CdbrVersionableDocument クラスを基にしたオブジェクト（CdbrVersionableDocument オブジェクト）として作成します。

3.2.1　文書を表すクラスの概要

ここでは，CdbrDocument クラスと CdbrVersionableDocument クラスの概要について説明します。

（1） CdbrDocument クラスの概要

CdbrDocument クラスの概要について説明します。

（a） 機能

バージョンなし文書は，CdbrDocument クラスを基にしたオブジェクトとして作成します。

このクラスは，CdbrContainable クラスのサブクラスであり，コンテナの包含要素となる機能を継承しています。

（b） CdbrDocument クラスを構成する DMA オブジェクト

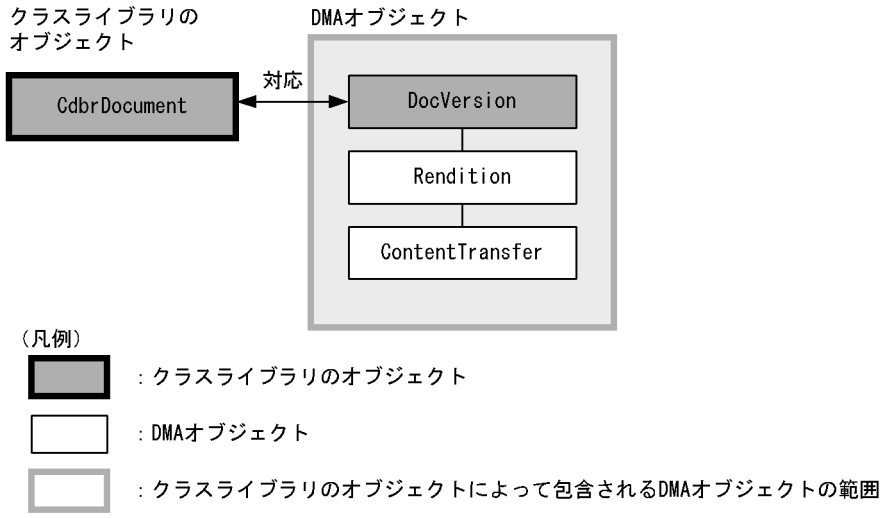
CdbrDocument クラスを構成する DMA オブジェクトの種類を次に示します。

- DocVersion オブジェクト
  バージョンなし文書で，文書を表現するオブジェクトです。dmaClass_DocVersion クラスのサブクラスのオブジェクトも含みます。
- Rendition オブジェクト
  文書を構成するファイルの種類を管理するオブジェクトです。
- ContentTransfer オブジェクト
  文書の実体であるコンテントデータを管理するオブジェクトです。

DMA オブジェクトについての詳細は，「付録 A　DMA オブジェクトの概要」を参照してください。

CdbrDocument クラスのオブジェクトと DMA オブジェクトの関係について次の図に示します。

図 3-12　CdbrDocument クラスのオブジェクトと DMA オブジェクトの関係



なお，この構成は，シングルファイル文書の場合です。マルチファイル文書，リファレンスファイル文書または File Link 文書の場合は，DMA オブジェクトの構成が異なります。詳細については，それぞれ「3.5　文書のマルチファイル管理」,「3.6　文書のリファレンスファイル管理」および「3.7　File Link 連携機能を使用した文書管理」を参照してください。

(2) CdbrVersionableDocument クラスの概要

CdbrVersionableDocument クラスの概要について説明します。

(a) 機能

バージョン付き文書は，CdbrVersionableDocument クラスを基にしたオブジェクトとして作成します。

このクラスは，CdbrVersionable クラスのサブクラスであり，CdbrVersionable クラスのバージョン管理機能を継承しています。したがって，CdbrVersionableDocument オブジェクトは，バージョンを管理できます。

CdbrVersionableDocument オブジェクトでは，複数のバージョンをまとめて一つの文書として扱います。文書のバージョン管理については,「3.3　文書のバージョン管理」を参照してください。

また，このクラスは，CdbrContainable クラスのサブクラスであり，CdbrContainable クラスのコンテナの包含要素となる機能も継承しています。

(b) CdbrVersionableDocument クラスを構成する DMA オブジェクト

CdbrVersionableDocument クラスを構成する DMA オブジェクトの種類を次に示します。Rendition オブジェクト，ContentTransfer オブジェクトについては，CdbrDocument クラスと同様です。

- ConfigurationHistory オブジェクト
  文書の一連のバージョンを統括するオブジェクトです。
- VersionSeries オブジェクト
  バージョンの構成を管理するオブジェクトです。
- VersionDescription オブジェクト
  VersionSeries オブジェクトと DocVersion オブジェクトを結び付けて，VersionSeries オブジェクトで管理するバージョンに対応する文書（DocVersion オブジェクト）が何であるかを管理するオブジェク

トです。

● DocVersion オブジェクト
バージョン付き文書での DocVersion オブジェクトは，文書の特定のバージョンに相当するオブジェクトです。dmaClass_DocVersion クラスのサブクラスのオブジェクトも含みます。

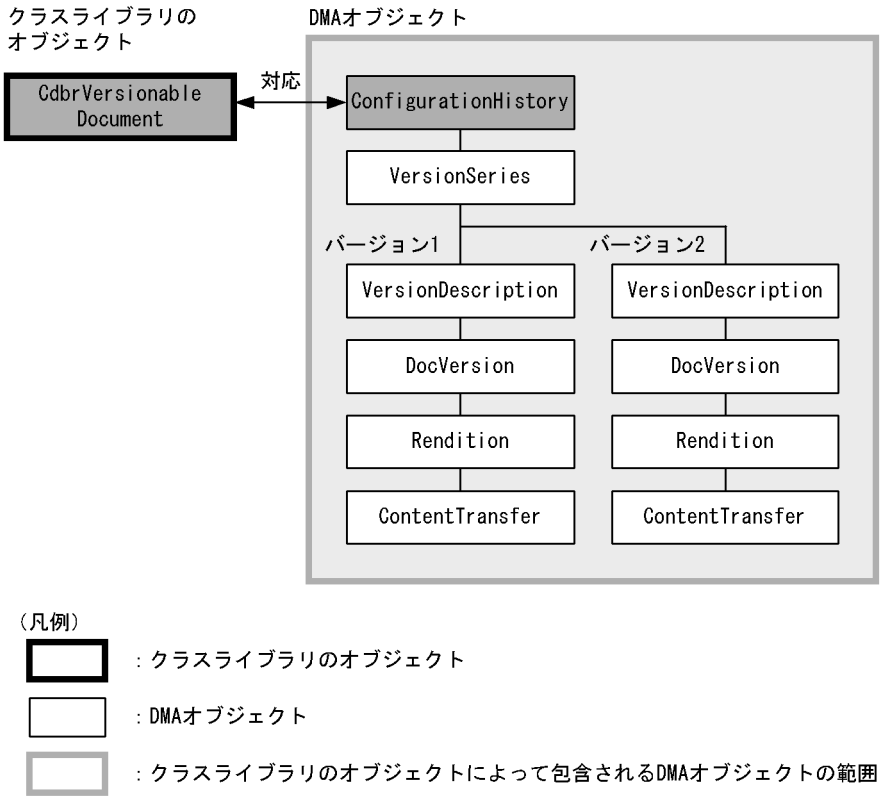
● Rendition オブジェクト

● ContentTransfer オブジェクト

DMA オブジェクトについての詳細は，「付録 A　DMA オブジェクトの概要」を参照してください。

CdbrVersionableDocument クラスのオブジェクトと DMA オブジェクトの関係について次の図に示します。

図 3-13　CdbrVersionableDocument クラスのオブジェクトと DMA オブジェクトの関係



なお，この構成は，シングルファイル文書の場合です。マルチファイル文書，リファレンスファイル文書または File Link 文書の場合は，DMA オブジェクトの構成が異なります。詳細については，それぞれ「3.5　文書のマルチファイル管理」，「3.6　文書のリファレンスファイル管理」および「3.7　File Link 連携機能を使用した文書管理」を参照してください。

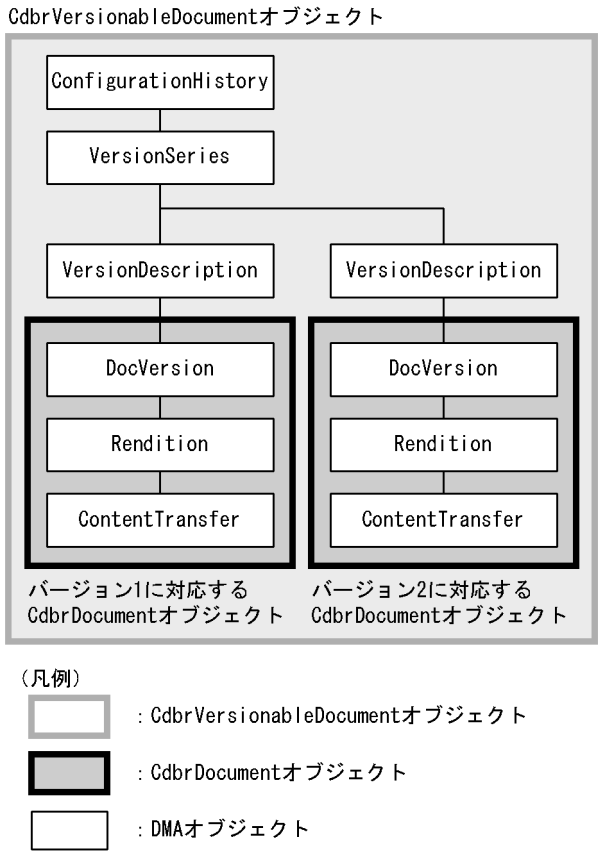
(3) CdbrVersionableDocument オブジェクトと CdbrDocument オブジェクトの関係

ここでは，CdbrVersionableDocument オブジェクトと CdbrDocument オブジェクトの関係について説明します。CdbrVersionableDocument オブジェクトとして作成されたバージョン付き文書は，それぞれのバージョンに対応する複数の CdbrDocument オブジェクト（バージョンなし文書）を包含しています。

次の図に，CdbrVersionableDocument オブジェクトと CdbrDocument オブジェクトの関係について示します。CdbrVersionableDocument オブジェクトとして作成された文書は，二つのバージョンを持ってい

ます。

図 3-14　CdbrVersionableDocument オブジェクトと CdbrDocument オブジェクトの関係



CdbrVersionableDocument オブジェクトの個々のバージョンは，CdbrDocument オブジェクトに対応しています。このため，CdbrVersionableDocument クラスを基に作成した CdbrVersionableDocument オブジェクトの個々のバージョンに対して，CdbrDocument クラスの機能を使用して操作することができます。また，CdbrDocument オブジェクトに接続した状態から，自身を 1 バージョンとしている CdbrVersionableDocument オブジェクトをたどって，CdbrVersionableDocument クラスの機能を使用して操作することもできます。

例えば，バージョン付き文書を全文検索したい場合，実際に全文検索の対象となるコンテントは個々のバージョンごとに存在するため，検索は個々のバージョンに対応する CdbrDocument オブジェクトが対象になります。検索結果としては CdbrDocument オブジェクトのトップオブジェクトである DocVersion オブジェクト（dmaClass_DocVersion クラスのサブクラスを基に作成した DMA オブジェクト）のプロパティ（OIID など）が取得できます。しかし，DocVersion オブジェクトの OIID ではバージョン付き文書である CdbrVersionableDocument オブジェクトに接続できません。このため，全文検索で取得した CdbrDocument オブジェクト（バージョンなし文書）を更新し，それを CdbrVersionableDocument オブジェクト（バージョン付き文書）の新しいバージョンとして追加したい場合などには，次のような手順で CdbrVersionableDocument オブジェクトに接続してください。

CdbrDocument オブジェクトの OIID を基に CdbrVersionableDocument オブジェクトに接続する手順

1. **検索で取得した DMA オブジェクトの DocVersion オブジェクトの OIID を指定して，CdbrDocument::SetOIID メソッドをコールし，CdbrDocument オブジェクトに接続します。**
2. CdbrDocument::GetVersionableList **メソッドまたは** CdbrDocument::GetVersionableListAndLock

メソッドによって，接続した CdbrDocument **オブジェクトを** 1 **バージョンとして管理している** CdbrVersionableDocument **オブジェクトの** OIID **を取得します。**
これによって，1. で接続した CdbrDocument オブジェクトを 1 バージョンとする CdbrVersionableDocument オブジェクトのトップオブジェクトである，DMA オブジェクトの ConfigurationHistory オブジェクトの OIID が取得できます。

3. 2. **で取得した** DMA **オブジェクトの** ConfigurationHistory **オブジェクトの** OIID **を指定して，** CdbrVersionableDocument **オブジェクトに接続します。**

3.2.2 文書のプロパティ

ここでは，文書に設定するプロパティと，プロパティを使用した検索について説明します。

(1) 文書のプロパティの管理

文書は，プロパティを設定して管理できます。

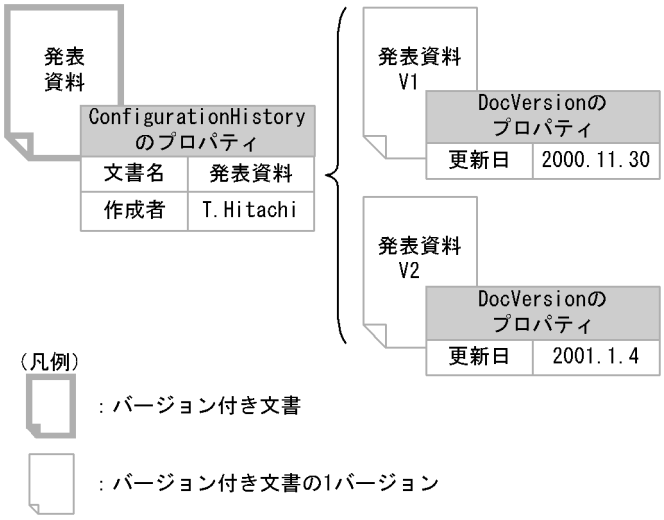
文書には，DMA が規定したプロパティおよびクラスライブラリ固有のプロパティが設定されています。これらのプロパティについては，「2.6.2　プロパティの種類」を参照してください。

このほか，文書には，ユーザ定義プロパティが設定できます。

文書のプロパティとしてタイトルや作成者などを設定しておくと，文書の検索などで使用できます。バージョンなし文書のプロパティは，構成要素である DMA オブジェクトの DocVersion オブジェクトのプロパティとして設定します。バージョン付き文書のプロパティは，構成要素である DMA オブジェクトの ConfigurationHistory オブジェクトと DocVersion オブジェクトのプロパティとして設定します。バージョン共通の情報は ConfigurationHistory オブジェクトのプロパティとして，各バージョン固有の情報は DocVersion オブジェクトのプロパティとして設定してください。

バージョン付き文書のプロパティの設定例を次の図に示します。

図 3-15　**バージョン付き文書のプロパティの設定例**



この例では，バージョン共通の情報を表すユーザ定義プロパティとして，文書名および作成者を設定しています。また，各バージョン固有の情報を表すユーザ定義プロパティとして，更新日を設定しています。

文書のプロパティを操作するメソッドについては，「2.6.6　プロパティの操作」を参照してください。

(2) プロパティを使用した検索

文書の検索では，バージョン付き文書の検索と，バージョンなし文書の検索ができます。それぞれのトップオブジェクトである DMA オブジェクトの ConfigurationHistory オブジェクトまたは DocVersion オブジェクトを対象に，プロパティを指定して検索を実行します。文書のコンテントを対象にした全文検索をする場合は，DocVersion オブジェクトを対象にします。

検索についての詳細は，「4. オブジェクトの検索」を参照してください。

3.2.3 文書の操作

文書の基本的な操作を実行する場合に使用するメソッドと，そのメソッドの発行順序の例を説明します。

ここでは，次の操作について説明します。

- 文書の作成
- 文書の参照
- 文書の更新
- 文書の削除

これらの操作を実行する前に，まず，文書空間と接続してトランザクションを開始してください。

また，DocumentBroker サーバがネットワーク上に存在する場合，DocumentBroker サーバに文書を登録したり DocumentBroker サーバの文書をクライアントで参照したりする場合にはあらかじめファイル転送サービスを起動しておく必要があります。ファイル転送機能については，マニュアル「DocumentBroker Version 3　システム導入・運用ガイド」を参照してください。

なお，この例では，操作する文書として CdbrDocument オブジェクトを使用します。

それぞれの操作に使用するメソッドの詳細については，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。

(1) 文書の作成

新しく文書を作成して，データベースに格納する場合に使用するメソッドと，そのメソッドの発行順序について説明します。なお，文書を登録するとき，全文検索インデクスを同時に作成できます。全文検索インデクスの作成については，「4.4.3　全文検索インデクスの作成」を参照してください。

準備

オブジェクトの構成要素になる DMA オブジェクト作成用のクラス識別子を指定した構造体（SDBR_DMAINFO 構造体）を作成しておきます。

- CdbrDocument オブジェクトを作成する場合は，SDBR_DMAINFO 構造体に，dmaClass_DocVersion クラスまたはそのサブクラスのクラス識別子を指定します。
- CdbrVersionableDocument オブジェクトを作成する場合は，SDBR_DMAINFO 構造体に，dmaClass_ConfigurationHistory クラスまたはそのサブクラスの識別子と，dmaClass_DocVersion クラスまたはそのサブクラスの識別子を指定します。
  このとき，dmaClass_DocVersion クラスのサブクラスとして，edmClass_VersionTracedDocVersion クラスもしくはそのサブクラス，または edmClass_VersionTracedComponentDocVersion クラスもしくはそのサブクラスを指定した場合だけ，構成管理コンテナの構成要素として管理できます。構成管理コンテナについては，「3.10　構成管理コンテナを使用した文書の構成管理」を参照してください。

1. **文書（CdbrDocument オブジェクト）を作成します。**

CdbrDocument::CreateObject メソッドをコールします。このとき，登録するファイルやファイルの種類（レンディションタイプ）などを指定します。

文書を作成，登録するコールシーケンスの例を次に示します。

作成した文書をコンテナと関連づける場合については，「3.9　コンテナを使用した文書管理」を参照してください。

なお，このコールシーケンスは，UNIX の場合です。Windows の場合は，CreateObject メソッドの引数に指定するファイルパスを，「"file:///c:¥temp¥sample.txt"」のように指定してください。

文書を作成，登録するコールシーケンス

```
// あらかじめ登録する内容を構造体 DMAClassList として作成しておく
//…
pSession->Begin();
CdbrDocument Doc;
//文書の作成および登録
Doc.CreateObject(pSession,
                    1, DMAClassList, "file:///tmp/sample.txt",
                    "MIME::text/plain", &pOIID,
                     DBR_CREATE_INDEX);
Doc.ReleaseObject();
//処理の確定
pSession->Commit();
```

レンディションタイプについての注意事項

文書として登録するファイルの種類を表すレンディションタイプは，登録したファイルの拡張子からは設定されません。レンディションタイプは，CdbrDocument::CreateObject メソッドまたは CdbrVersionableDocument::CreateObject メソッドの引数として明示的に指定してください。この引数に指定した値がレンディションごとのファイルの種類を表すプロパティとして設定されます。
特に，文書を作成する時に同時に全文検索に必要な全文検索インデクスを作成したい場合は，適切なレンディションタイプを設定するようにしてください。文書の登録と同時に全文検索インデクスを作成する場合，特定のレンディションタイプを持つ文書オブジェクト以外からは全文検索インデクスが作成できません。また，登録済みの文書から全文検索インデクスを作成する場合も，特定のレンディションタイプを持つ文書オブジェクト以外からは全文検索インデクスが作成できません。全文検索インデクスが作成できないレンディションタイプの文書オブジェクトに対しては，登録した文書とは別に全文検索用インデクスのデータを作成して，登録した文書と関連づける必要があります。
全文検索インデクスを作成できるレンディションタイプおよびその作成方法については，「4.4.3　全文検索インデクスの作成」を参照してください。
なお，レンディションタイプを表すプロパティは PutPropertyValues メソッドでは変更できません。登録済みの文書を更新する時に，文書の実体を表すファイルの種類を変更する場合には，必要に応じて UpdateContentAndRenditionType メソッドをコールして，レンディションタイプを変更するようにしてください。

(2) 文書の参照

すでにデータベースに登録されている文書を参照する場合に使用するメソッドと，そのメソッドの発行順序について説明します。

準備

オブジェクトの構成要素である DMA オブジェクトの OIID を，検索などによってあらかじめ取得しておきます。検索については「4.　オブジェクトの検索」を参照してください。

- CdbrDocument オブジェクトに接続する場合は，構成要素である DMA オブジェクトの，DocVersion オブジェクトの OIID を取得しておきます。

- CdbrVersionableDocument オブジェクトに接続する場合は，構成要素である DMA オブジェクトの，ConfigurationHistory オブジェクトの OIID を取得しておきます。

1. **参照する文書に接続します。**
   CdbrDocument::SetOIID メソッドまたは CdbrDocument::ConnectObject メソッドをコールします。
2. **文書のコンテントデータを，指定したパス名のファイルに複写します。**
   クライアント環境のファイルのパス名を file: プロトコルで指定して，
   CdbrDocument::GetContentAndLock メソッドまたは CdbrDocument::GetContent メソッドをコールします。

文書を参照する場合の，コールシーケンスの例を次に示します。

なお，このコールシーケンスは，UNIX の場合です。Windows の場合は，GetContentAndLock メソッドの引数に指定するファイルパスを，「"file:///c:¥temp¥sample.txt"」のように指定してください。

文書を参照するコールシーケンス

```
pSession->Begin();
//…
//検索によって参照する文書のOIIDを取得する
//…
CdbrDocument Doc;
//検索で取得したOIIDをオブジェクトに設定する
Doc.SetOIID(pSession, pOIID);
//readロックを設定してファイルを取得する
Doc.GetContentAndLock("file:///tmp/sample.txt",
                      DMA_LOCK_READ);
Doc.ReleaseObject();
pSession->Commit();
```

(3) 文書の更新

すでにデータベースに登録されている文書を編集して更新する場合に使用するメソッドと，そのメソッドの発行順序について説明します。なお，(2) と同様に，あらかじめ接続する文書の構成要素である DMA オブジェクトの OIID を取得しておきます。

1. **更新する文書に接続します。**
   CdbrDocument::SetOIID メソッドまたは CdbrDocument::ConnectObject メソッドをコールします。
   ConnectObject メソッドのロックは write ロックを設定します。
2. **文書のコンテントデータを，指定したパス名のファイルに複写します。**
   クライアント環境のファイルのパス名を file: プロトコルで指定して，
   CdbrDocument::GetContentAndLock メソッドまたは CdbrDocument::GetContent メソッドをコールします。
3. **ファイルを編集します。**
4. **編集したファイルで文書を更新します。**
   CdbrDocument::UpdateContent メソッドまたは CdbrDocument::UpdateContentAndRenditionType メソッドをコールします。
   なお，UpdateContent メソッドで更新するファイルのパスは，2. の GetContentAndLock メソッドまたは GetContent メソッドで取得したファイルのパスと同じである必要はありません。例えば，UNIX の場合，2. で文書のコンテントデータを file:///tmp/sample.txt というパス名のファイルに複写したときでも，更新するファイルとして file:///tmp/sample2.txt というパス名のファイルを指定してもかまいません。Windows の場合，2. で文書のコンテントデータを file:///c:¥temp¥sample.txt というパス名のファイルに複写したときでも，更新するファイルとして file:///c:¥temp¥sample2.txt というパス名の

ファイルを指定してもかまいません。
また，更新時に，GetContent メソッドで取得したファイルとは異なる種類のファイルを登録する場合は，必要に応じてレンディションタイプを変更してください。この場合は，UpdateContentAndRenditionType メソッドを使用します。例えば，テキスト形式のファイル（sample.txt）をコンテントデータとして登録していた文書に対して，HTML 形式のファイル（sample.html）に置き換えて更新する場合，UpdateContentAndRenditionType メソッドの引数に HTML 形式を表す MIME 名（MIME::text/html）を指定します。これによって，レンディションタイプが変更されます。
なお，更新するファイルとして，取得したファイルと種類が異なるものを指定した場合でも，UpdateContentAndRenditionType メソッドでレンディションタイプの変更を指定しなかった場合は，レンディションタイプは変更されません。登録したファイルの拡張子によってレンディションタイプが自動的に変更されることはありません。

文書を更新する場合の，コールシーケンスの例を次に示します。

なお，このコールシーケンスは，UNIX の場合です。Windows の場合は，GetContentAndLock メソッドおよび UpdateContent メソッドの引数に指定するファイルパスを，「"file:///c:¥temp¥sample.txt"」のように指定してください。

文書を更新するコールシーケンス

```
pSession->Begin();
//…
//検索によって更新する文書のOIIDを取得する
//…
CdbrDocument Doc;
//検索で取得したOIIDをオブジェクトに設定する
Doc.SetOIID(pSession, pOIID);
//writeロックを設定してファイルを取得する
Doc.GetContentAndLock("file:///tmp/sample.txt",
                      DMA_LOCK_WRITE);
//…
//クライアント環境で取得したファイルの内容の更新する
//…
//文書を更新する
Doc.UpdateContent("file:///tmp/sample.txt",
                  DBR_CREATE_INDEX);
Doc.ReleaseObject();
pSession->Commit();
```

（4） 文書の削除

CdbrDocument オブジェクトおよび CdbrVersionableDocument オブジェクトを削除する場合，RemoveObject メソッドを使用します。これによって，CdbrDocument オブジェクトおよび CdbrVersionableDocument オブジェクトを構成していたすべての DMA オブジェクトが削除されます。削除する CdbrDocument オブジェクトが CdbrVersionableDocument オブジェクトの 1 バージョンに当たる場合は，そのバージョンを表す DMA オブジェクトの VersionDescription オブジェクトも一緒に削除されます。

また，特定のバージョンに対応するバージョンなし文書である CdbrDocument オブジェクトを削除する場合は，バージョン付き文書である CdbrVersionableDocument オブジェクトに接続して，CdbrVersionableDocument::DeleteVersion メソッドをコールする方法もあります。この方法については，「3.3.3　バージョン管理機能の操作」を参照してください。.

ここでは，データベースに登録されている文書を削除する場合に使用するメソッドと，そのメソッドの発行順序について説明します。なお，(2) と同様に，あらかじめ接続する文書の構成要素である DMA オブ

ジェクトの OIID を取得しておきます。

1. **削除する文書に接続します。**
   CdbrDocument::SetOIID メソッドまたは CdbrDocument::ConnectObject メソッドをコールします。
   ConnectObject メソッドのロックには write ロックを設定します。

2. **文書を削除します。**
   CdbrDocument::RemoveObject メソッドをコールします。
   なお，SetOIID メソッドによってオブジェクトに接続している場合，RemoveObject メソッドをコールした時に DocumentBroker によって write ロックが設定されます。

文書を削除する場合のコールシーケンスの例を次に示します。

文書を削除するコールシーケンス

```
pSession->Begin();
//…
//検索によって削除する文書のOIIDを取得する
//…
CdbrDocument Doc;
//検索で取得したOIIDをオブジェクトに設定する
Doc.SetOIID(pSession, pOIID);
Doc.RemoveObject();
Doc.ReleaseObject();
pSession->Commit();
```

3.3 文書のバージョン管理

この節では，文書のバージョン管理について説明します。バージョン管理には，CdbrVersionable クラスの機能を使用します。

3.3.1 CdbrVersionable クラスの概要

ここでは，CdbrVersionable クラスの概要について説明します。

(1) 機能

文書またはコンテナは，バージョンを付けて管理できます。例えば，文書に対して，第 1 版，第 2 版などのバージョンを付けておくことで，過去のある時点での文書を取り出したり，その文書を基に新たな文書を作成したりできます。

バージョンは，新規に文書またはコンテナを保存した時から付けることができます。保存した文書またはコンテナは，更新する時に新しいバージョンを追加して更新するか，既存のバージョンに上書きして更新するか選択できます。新しいバージョンを追加して更新する場合に，バージョン管理機能を使用します。

バージョン管理機能とは，チェックイン，チェックアウトという操作によって新しいバージョンを追加したり，バージョン情報の一覧を取得して特定のバージョンを指定して操作したりする機能です。また，バージョンごとにプロパティの設定もできます。この機能は，CdbrVersionable クラスが提供するバージョン管理機能によって実現します。

なお，CdbrVersionable クラスは抽象クラスです。CdbrVersionable クラスでは，次に示すサブクラスでバージョン管理機能を使用するためのメソッドなどを提供しています。

- CdbrVersionableDocument クラス
- CdbrConfiguratedReferentialContainer クラス

したがって，CdbrVersionableDocument オブジェクトとして作成されたバージョン付き文書および CdbrConfiguratedReferentialContainer オブジェクトとして作成されたバージョン付き構成管理コンテナが，バージョン管理の対象になります。

(2) CdbrVersionable クラスを構成する DMA オブジェクト

CdbrVersionable クラスを構成する DMA オブジェクトの種類を次に示します。

● ConfigurationHistory オブジェクト
このオブジェクトのプロパティが，CdbrVersionable クラスのデフォルトのプロパティになります。また，ConfigurationHistory オブジェクトのプロパティには，管理しているすべてのバージョンに共通のプロパティを設定できます。

● VersionSeries オブジェクト
バージョンの構成を管理するオブジェクトです。

● VersionDescription オブジェクト
VersionSeries オブジェクトと dmaClass_Versionable クラスのサブクラスのオブジェクトを結び付けて，VersionSeries オブジェクトで管理するバージョンに対応するオブジェクトが何であるかを管理するオブジェクトです。

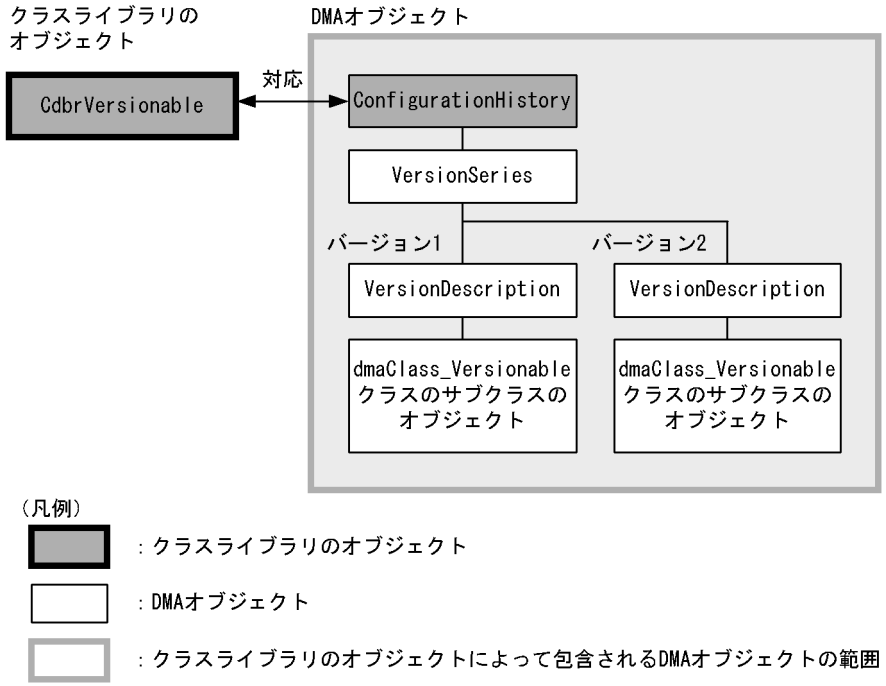
● dmaClass_Versionable クラスのサブクラスのオブジェクト
バージョン管理されるオブジェクトです。文書およびコンテナを表すオブジェクトの個々のバージョン

に当たる，DocVersion オブジェクトや ContainerVersion オブジェクトがこれに当たります。また，これらのオブジェクトのプロパティとして，バージョンごとに異なるプロパティが設定できます。

DMA オブジェクトについての詳細は，「付録 A　DMA オブジェクトの概要」を参照してください。

CdbrVersionable クラスのオブジェクトと DMA オブジェクトの関係について次の図に示します。

図 3-16　CdbrVersionable クラスのオブジェクトと DMA オブジェクトの関係



(3) バージョン管理の対象になるオブジェクトに設定するプロパティ

バージョン管理をしている文書やコンテナでは，プロパティとして，次の 2 種類の情報が設定できます。

- すべてのバージョンに共通する情報
  DMA オブジェクトの ConfigurationHistory オブジェクトのプロパティ
- 個々のバージョンで異なる情報
  DMA クラスの dmaClass_Versionable クラスのサブクラスから作成したオブジェクトのプロパティ
  （DocVersion オブジェクトまたは ContainerVersion オブジェクト）

例えば，文書を管理するときに，すべてのバージョンに共通である文書名を ConfigurationHistory オブジェクトのプロパティとして設定したり，個々のバージョンで異なる情報である更新日を DocVersion オブジェクトのプロパティとして設定したりして，管理できます。

3.3.2　バージョンの管理方法

ここでは，文書またはコンテナをバージョン管理機能を使用して管理する方法について説明します。この説明中の文書とは，バージョン付き文書です。コンテナは，バージョン付き構成管理コンテナです。

(1) バージョンの追加

文書やコンテナを更新する場合に，バージョンを追加して新しいバージョンを作成する時，チェックアウトとチェックインという操作をします。

チェックアウトおよびチェックインについて説明します。

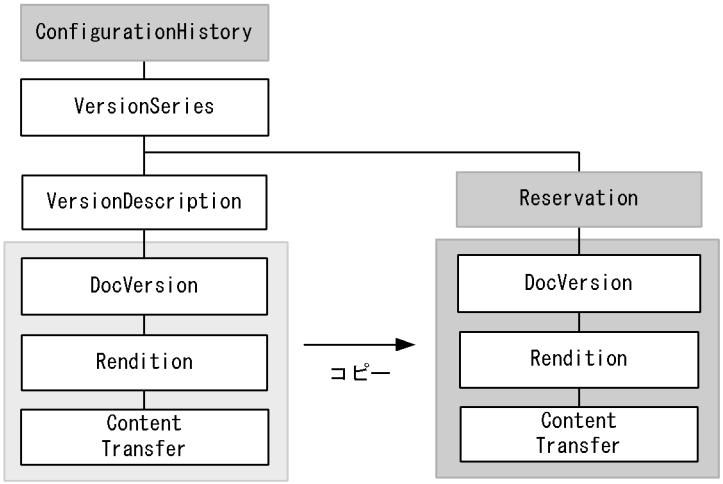
チェックアウト

文書やコンテナのバージョンを追加するために，次バージョンの追加を予約して，最新バージョンの文書またはコンテナのコピーを要求することです。バージョンのチェックアウトは，VersionCheckOut メソッドによって実行します。

なお，チェックアウトした段階でバージョンの追加は完了していません。したがって，メソッドによってバージョンの一覧を取得した場合，仮のバージョンに関する情報は取得できません。

チェックアウトをしている状態の DMA オブジェクトの構成を次の図に示します。

図 3-17　チェックアウトをしている状態の DMA オブジェクトの構成



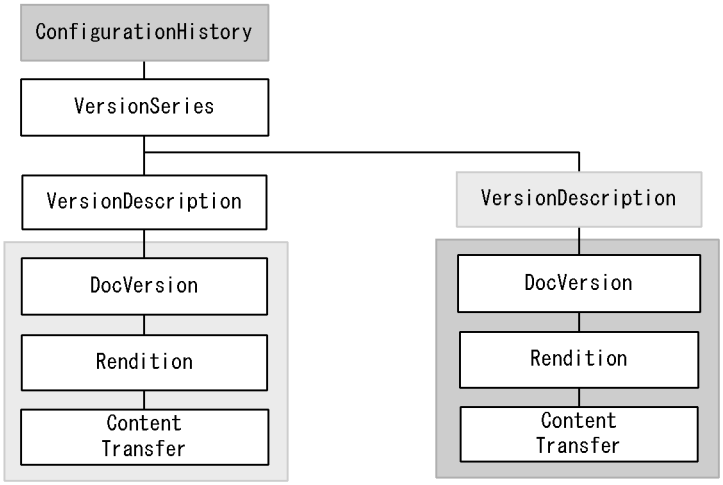
Reservation オブジェクトは，新しいバージョンを作成する権利を VersionSeries オブジェクトに予約する DMA オブジェクトです。チェックインすると，VersionDescription オブジェクトに置き換わります。

チェックイン

バージョンの追加を確定することです。バージョンのチェックインは，VersionCheckIn メソッドによって実行します。

チェックインをした状態の DMA オブジェクトの構成を次の図に示します。

図 3-18　チェックインをした状態の DMA オブジェクトの構成



チェックアウトされたバージョンを参照，操作する場合，仮のバージョン識別子を指定します。仮のバージョン識別子は，チェックアウトを実行した時に取得でき，チェックインを実行した時に無効になります。仮のバージョン識別子は，チェックイン後のバージョンの識別子とは異なりますので，ご注意ください。

バージョンをチェックアウトしたあとで，チェックアウトを取り消す場合は，VersionRevoke メソッドをコールします。このメソッドによって，仮のバージョンおよび仮のバージョン識別子は破棄されます。

(2) バージョンの順序性とカレントバージョン

バージョンには順序があります。バージョンの順序とは，それぞれのバージョンをチェックインした順序です。バージョンの順序は，バージョン管理できるオブジェクトの構成要素である，DMA オブジェクトの VersionSeries オブジェクトが保持しています。なお，最新のバージョンのことを，**カレントバージョン**といいます。

バージョンの順序は，GetVersionList メソッドまたは GetVersionListAndLock メソッドによって取得できます。引数の指定によって，新しいバージョンから取得したり，古いバージョンから取得したりできます。

バージョンの順序性についての注意事項

DMA が規定しているインターフェース（DMA インターフェース）を使用してバージョンを追加したオブジェクトに対しては，バージョンの順序性は保証されません。したがって，このオブジェクトに対して，カレントバージョンを指定した操作などはできません。ただし，このオブジェクトにクラスライブラリのメソッドによって再度バージョンを追加すれば，そのバージョンをカレントバージョンとして扱えます。

(3) バージョンの指定方法

特定のバージョンに対応する文書またはコンテナを操作する場合は，メソッドの引数としてバージョン識別子を指定します。バージョン識別子は，GetVersionList メソッドまたは GetVersionListAndLock メソッドを使用して取得します。

チェックアウトしている文書またはコンテナのバージョンを指定する場合は，チェックアウト時に取得した仮のバージョン識別子を指定します。仮のバージョン識別子は，VersionCheckOut メソッドをコールした時に取得できます。また，すでにチェックアウトされているオブジェクトの仮のバージョン識別子を，GetReservationStatus メソッドまたは GetReservationStatusAndLock メソッドによって，バージョンのチェックアウト状態を確認して取得することもできます。

バージョンを指定して文書やコンテナを操作できるメソッド（引数でバージョン識別子を指定できるメソッド）で，バージョン識別子に NULL を指定した場合，メソッドは最新のバージョンに対して処理を実行します。

3.3.3 バージョン管理機能の操作

バージョン管理機能を使用する場合の，使用するメソッドと，そのメソッドの発行順序について説明します。

ここでは，次の操作について説明します。

- バージョン管理する文書およびコンテナの作成
- バージョンの追加
- バージョンの削除

これらの操作を実行する前に，まず，文書空間と接続してトランザクションを開始してください。

なお，例では，バージョン付き文書である CdbrVersionableDocument オブジェクトを使用します。

それぞれの操作に使用するメソッドの詳細については，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。

(1) バージョン管理する文書またはコンテナの作成

バージョン管理する文書およびコンテナは，CdbrVersionableDocument クラスまたは CdbrConfiguratedReferentialContainer クラスの CreateObject メソッドをコールすることで作成されます。CreateObject メソッドの使用例については，「3.2.3(1) 文書の作成」を参照してください。

(2) バージョンの追加

チェックアウトとチェックインをすることで文書やコンテナにバージョンを追加します。

チェックアウトの取り消しには，CdbrVersionable クラスで定義された VersionRevoke メソッドを使用します。

バージョンを追加する流れを説明します。この説明では，既存の文書をデータベースから取り出して編集し，新しいバージョンとして登録します。

準備

接続するオブジェクトの構成要素である DMA オブジェクトの OIID を，検索などによってあらかじめ取得しておきます。検索については「4.　オブジェクトの検索」を参照してください。
文書やコンテナにバージョンを追加する場合に取得する OIID は，接続するオブジェクトの種類によって，次のようになります。

- CdbrVersionableDocument オブジェクトに接続する場合は，構成要素である DMA オブジェクトの，ConfigurationHistory オブジェクトの OIID を取得しておきます。
- CdbrConfiguratedReferentialContainer オブジェクトに接続する場合は，構成要素である DMA オブジェクトの，ConfigurationHistory オブジェクトの OIID を取得しておきます。

ここでは，文書のチェックアウトとチェックインの流れについて説明します。

● 文書をチェックアウトする

まず，バージョン付き文書をチェックアウトして次バージョンの追加を予約し，取得したファイルのコピーを編集できるようにします。

1. **バージョンを追加するバージョン付き文書に接続します。**
   CdbrVersionableDocument::SetOIID メソッドまたは CdbrVersionableDocument::ConnectObject メソッドをコールします。
2. **バージョン付き文書をチェックアウトします。**
   CdbrVersionableDocument::VersionCheckOut メソッドをコールします。これによって，次バージョンの追加が予約できます。
3. **チェックアウトしたバージョン付き文書の実体を，指定したパス名のファイルに複写します。**
   CdbrVersionableDocument::GetContentAndLock メソッドまたは CdbrVersionableDocument::GetContent メソッドをコールします。

● 文書をチェックインしてバージョンを追加する

次に，チェックアウトしたバージョン付き文書を編集して，最新のバージョンとして登録します。
バージョンアップした文書に登録するファイルはあらかじめ編集しておきます。

1. **バージョンを追加する文書に接続します。**
   CdbrVersionableDocument::SetOIID メソッドまたは CdbrVersionableDocument::ConnectObject メソッドをコールします。
2. **編集した文書を登録します。**
   CdbrVersionableDocument::UpdateContent メソッドまたは

CdbrVersionableDocument::UpdateContentAndRenditionType メソッドをコールして，編集した文書を登録します。引数には，チェックアウトした時に取得したバージョン識別子を指定します。このとき，更新するファイルとしてチェックアウトで取得したファイルではないファイルを指定することもできます。

3. **文書をチェックインします。**
   CdbrVersionableDocument::VersionCheckIn メソッドをコールします。

チェックアウト，チェックインによってバージョンを追加する，コールシーケンスの例を次に示します。

なお，このコールシーケンスは，UNIX の場合です。Windows の場合は，GetContentAndLock メソッドおよび UpdateContent メソッドの引数に指定するファイルパスを，「"file:///c:¥temp¥sample.txt"」のように指定してください。

文書をチェックアウトするコールシーケンス

```
pSession->Begin();
//…
//検索によって更新するバージョン付き文書のOIIDを取得する
//…
CdbrVersionableDocument VrDoc;
//検索で取得したOIIDをオブジェクトに設定する
VrDoc.SetOIID(pSession, pOIID);
//バージョン付き文書をチェックアウトする
VrDoc.VersionCheckOut(&pReservedVerId);
//writeロックを設定してファイルを取得する
VrDoc.GetContentAndLock("file:///tmp/sample.txt",
                        pReservedVerId,
                        DMA_LOCK_WRITE);
VrDoc.ReleaseObject();
pSession->Commit();
```

文書をチェックインするコールシーケンス

```
//クライアント環境でチェックアウトして取得したファイルの内容を
//更新しておく
pSession->Begin();
CdbrVersionableDocument VrDoc;
//OIIDをオブジェクトに設定する
VrDoc.SetOIID(pSession, pOIID);
//チェックアウトで作成されたコピーしたバージョン付き文書のファイルを
//更新する
VrDoc.UpdateContent("file:///tmp/sample.txt",
                    pReservedVerId);
//バージョン付き文書をチェックインする
VrDoc.VersionCheckIn();
VrDoc.ReleaseObject();
pSession->Commit();
```

(3) 文書のバージョンの削除

チェックインの実行によってすでに確定している文書またはコンテナのバージョンを削除する場合は，CdbrVersionable クラスで定義されている DeleteVersion メソッドを使用します。このメソッドによって，指定したバージョンの，次に示す DMA オブジェクトが削除されます。

- VersionDescription オブジェクト
- dmaClass_Versionable クラスのサブクラスのオブジェクトおよびそのオブジェクトとバインドされたオブジェクト

CdbrVersionableDocument オブジェクトのバージョンに対して DeleteVersion メソッドをコールすると，そのバージョンに対応する CdbrDocument オブジェクトが削除されます。つまり，CdbrDocument オブジェクトに対して RemoveObject メソッドをコールした場合と同じ操作になります。

また，バージョン付き構成管理コンテナによって構成管理されている文書のバージョンを削除する場合は，その構成管理モードによって，削除されるオブジェクトが異なります。詳細は，「3.10.2(6) 構成管理コンテナで構成管理しているオブジェクトの削除」を参照してください。

CdbrVersionableDocument オブジェクトのバージョンを削除するコールシーケンスの例を次に示します。

文書のバージョンを削除するコールシーケンス

```
pSession->Begin();
//…
//検索によってバージョンを削除するバージョン付き文書のOIIDを取得する
//…
CdbrVersionableDocument VrDoc;
//検索で取得したOIIDをオブジェクトに設定する
VrDoc.SetOIID(pSession, pOIID);
//バージョンの一覧を取得する
VrDoc.GetVersionListAndLock(&bContinue, 0, NULL, 10,
                            &pObjList,DMA_LOCK_READ);
//バージョンを削除する
VrDoc.DeleteVersion(pVersionId);
VrDoc.ReleaseObject();
pSession->Commit();
```

3.4 文書のマルチレンディション管理

この節では，文書のマルチレンディション管理について説明します。

3.4.1 マルチレンディション管理の概要

ここでは，マルチレンディション管理の概要について説明します。

(1) 機能

文書のマルチレンディション管理とは，バージョンなし文書またはバージョン付き文書に複数のレンディションを登録して管理することです。マルチレンディション管理は，次のような場合に使用できます。

- あるアプリケーションプログラムで作成した文書ファイルを，そのアプリケーションプログラムをインストールしていないマシンでも参照するために，同じ内容の HTML 形式および PDF 形式のファイルを作成して一緒に登録したい場合
- 同じ拡張子の文書ファイルでも，保存状態やアプリケーションプログラムのバージョンの違いごとに複数登録したい場合（例えば，拡張子が同じ「.gif」であっても解像度の異なるファイルや，拡張子が同じ「.doc」であっても Word95 形式で保存されたファイルと Word97 形式で保存されたファイルなど）
- 更新可能な形式のファイルと更新できない形式のファイルを両方登録したい場合（例えば，同じ内容の Word 形式のファイルと PDF 形式のファイルなど）

マルチレンディション管理モデルでは，次の 2 種類のレンディションを区別して扱います。

- マスタレンディション
- サブレンディション

なお，このマニュアルでは，2 種類のレンディションを合わせて「レンディション」と記述します。レンディションの種類によって説明が異なる場合は，それぞれ「マスタレンディション」,「サブレンディション」と記述して区別します。それぞれについて説明します。

マスタレンディション
文書のレンディションとして，最初に登録されたレンディションを**マスタレンディション**といいます。

サブレンディション
追加登録されたレンディションをマスタレンディションと区別して**サブレンディション**といいます。

文書をマルチレンディション文書として管理する場合，一つの文書に対して，1 個のマスタレンディションと最大 9 個のサブレンディションを登録できます。追加したサブレンディションは，削除できます。ただし，マスタレンディションは削除できません。また，サブレンディションをマスタレンディションに変更することもできます。この場合，それまでマスタレンディションだったレンディションは，サブレンディションに変更されます。

なお，マスタレンディションに登録されているコンテントから，サブレンディションに登録するコンテントを作成，登録することもできます。マスタレンディションのコンテントの文書形式を変換してサブレンディションのコンテントを作成，登録することを，**レンディション変換**といいます。レンディション変換は，DocumentBroker Rendering Option を使用して実行できます。

(2) マルチレンディション管理を表す DMA オブジェクト

マルチレンディション文書を構成する DMA オブジェクトの種類を次に示します。

- DocVersion オブジェクト

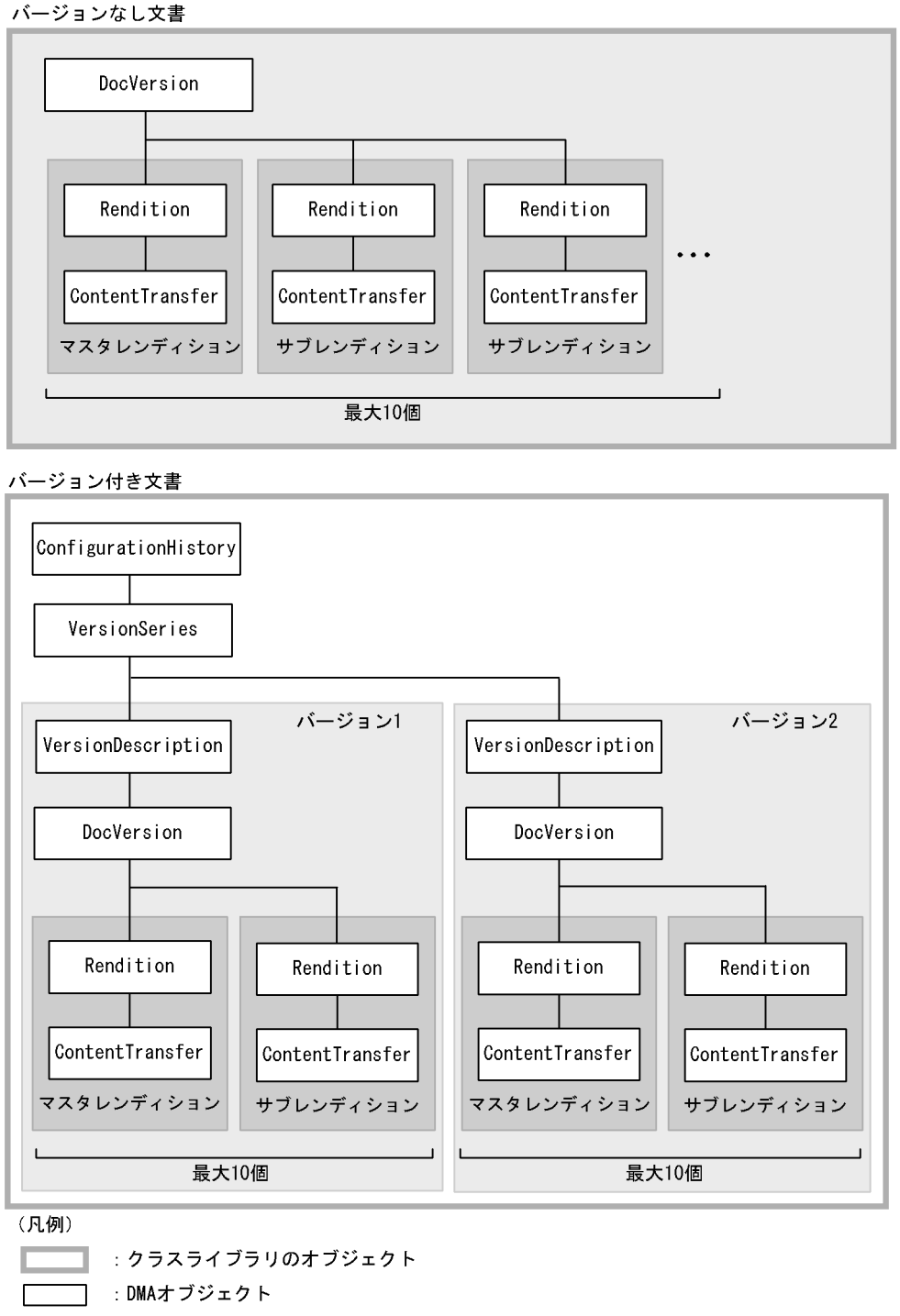
dmaClass_DocVersion クラスのサブクラスのオブジェクトも含みます。

● Rendition オブジェクト
文書のレンディションタイプを管理するオブジェクトです。登録できるのは，マスタレンディションを含めて最大 10 個です。

● ContentTransfer オブジェクト
文書のコンテントである文書ファイルを格納するためのオブジェクトです。レンディションタイプに対応するコンテントを指定して登録します。

マルチレンディション文書として管理しているバージョンなし文書およびバージョン付き文書の，DMA オブジェクトの構成を次の図に示します。

図 3-19　マルチレンディション文書の DMA オブジェクトの構成



3.4.2　マルチレンディション管理を使用した文書管理

ここでは，マルチレンディション文書の管理方法について説明します。

(1) レンディションの追加

バージョン付き文書をマルチレンディション文書として管理してレンディションを追加する例を次の図に示します。

図 3-20　マルチレンディション文書のレンディションの追加例

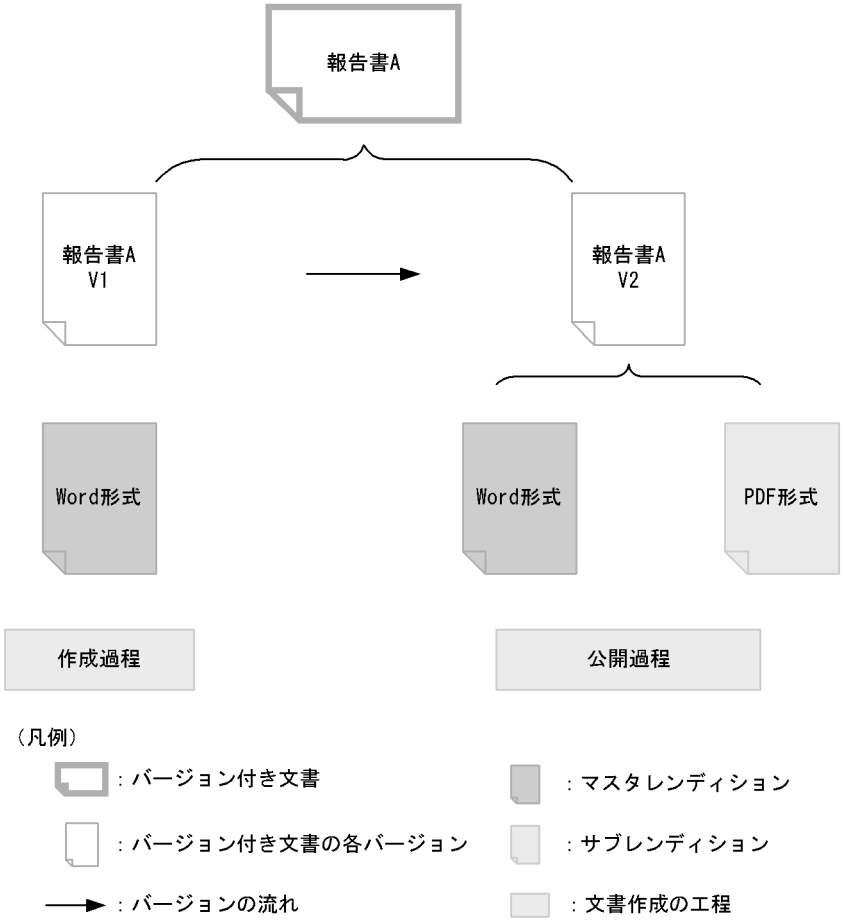


図 3-20 の例では，文書の作成過程で，文書の作成者が更新用にコンテントをダウンロードできるように，コンテントを作成しているアプリケーションプログラム（この例の場合 Word）の形式のレンディションを登録しています。この Word 形式のレンディションがマスタレンディションとなります。そして，文書が完成して公開過程になったタイミングでバージョンを追加して，追加したバージョンをほかのユーザが参照しやすいように，PDF 形式のレンディションを追加登録しています。この追加した PDF 形式のレンディションがサブレンディションとなります。なお，マルチレンディション文書をチェックアウトすると，最新バージョンのマスタレンディションが仮のバージョンにコピーされます。サブレンディションはコピーされません。例えば，「報告書 A」にさらに「V3」というバージョンを追加する場合，チェックアウトすると，Word 形式のレンディションはコピーされますが，PDF 形式のレンディションはコピーされません。マルチレンディション文書であるバージョン付き文書のバージョンを追加して，追加するバージョンにレンディションを追加する手順は次のようになります。

マルチレンディション文書のバージョンの追加の手順例

1. **マルチレンディション文書であるバージョン付き文書をチェックアウトします。**
   最新バージョンのマスタレンディションが仮のバージョンにコピーされます。
2. **最新バージョンのマスタレンディションの文書ファイルをダウンロードします。**
3. **アプリケーションプログラムで文書ファイルを編集します。**
4. **必要に応じて仮のバージョンのプロパティを設定します。**
5. **編集した文書ファイルを，仮のバージョンのマスタレンディションのコンテントとして更新します。**
6. **仮のバージョンにサブレンディションを登録します。**

7. **バージョン付き文書をチェックインして，仮のバージョンを最新バージョンとして確定します。**

(2) レンディションの参照

マルチレンディション文書は，レンディションタイプを指定して参照できます。例えば，図 3-20 の場合，「報告書 A」の V2 を参照するときには，レンディション一覧を取得して，Word 形式か PDF 形式かを指定して参照します。文書の作成者が文書をバージョンアップしたいときは，Word で文書ファイルを編集するために Word 形式を指定してダウンロードしたり，Word をインストールしていない環境のユーザは PDF 形式を指定してダウンロードしたりするなど，目的やコンテントを参照するマシン環境に応じて参照できます。また，アクセス制御機能と組み合わせたユーザプログラムを作ることによって，対象の文書に対して基本コンテント更新権のあるユーザには Word 形式，基本コンテント参照権しかないユーザには PDF 形式を自動的に参照させるという運用も考えられます。

アクセス制御機能については，「3.15　アクセス制御」を参照してください。

(3) マスタレンディションおよびサブレンディションのコンテントの更新

マルチレンディション文書のコンテントを更新する場合，明示的にどのレンディションのコンテントを更新するか，指定する必要があります。例えば，600dpi の GIF ファイルをマスタレンディション，300dpi の GIF ファイルをサブレンディションとして管理している場合に，300dpi の GIF ファイルを更新するときは，該当する文書のレンディションタイプを指定してください。

なお，レンディションタイプを指定しない場合は，マスタレンディションのコンテントが更新されます。

レンディションのコンテントを更新する場合は，レンディション間のファイル形式以外の内容が一致するようにご注意ください。

(4) レンディションの削除

マルチレンディション文書の特定のレンディションを削除したい場合は，マルチレンディション文書に接続して，レンディションタイプを指定して削除できます。ただし，指定されたレンディションがマスタレンディションである場合はレンディションを削除できません。また，存在しないレンディションタイプのレンディションを削除しようとするとエラーになります。

なお，マルチレンディション文書自体を削除した場合は，すべてのレンディションが削除されます。

(5) レンディションの検索

レンディションタイプを表す dbrProp_RenditionType プロパティは，属性検索の条件には指定できません。例えば，「マスタレンディションのレンディションタイプが "MIME::application/ms-word" である文書を検索する」のような指定はできません。また，検索結果として，レンディションタイプを取得することもできません。レンディションタイプを参照したいときは，ほかの検索条件によって参照したい文書を特定してから，GetRenditionListAndLock メソッドまたは GetRenditionList メソッドによって取得してください。

また，マルチレンディション文書であっても，作成できる全文検索インデクスは一つです。文書を更新した場合などに更新したコンテントから作成される全文検索インデクスは，マスタレンディションのコンテントから作成されます。

3.4.3　レンディションのプロパティ

マルチレンディションの各レンディションは，プロパティを設定して管理できます。

レンディションに設定されているプロパティは，次の 4 種類です。

● dbrProp_RenditionType プロパティ

● dbrProp_RetrievalName プロパティ

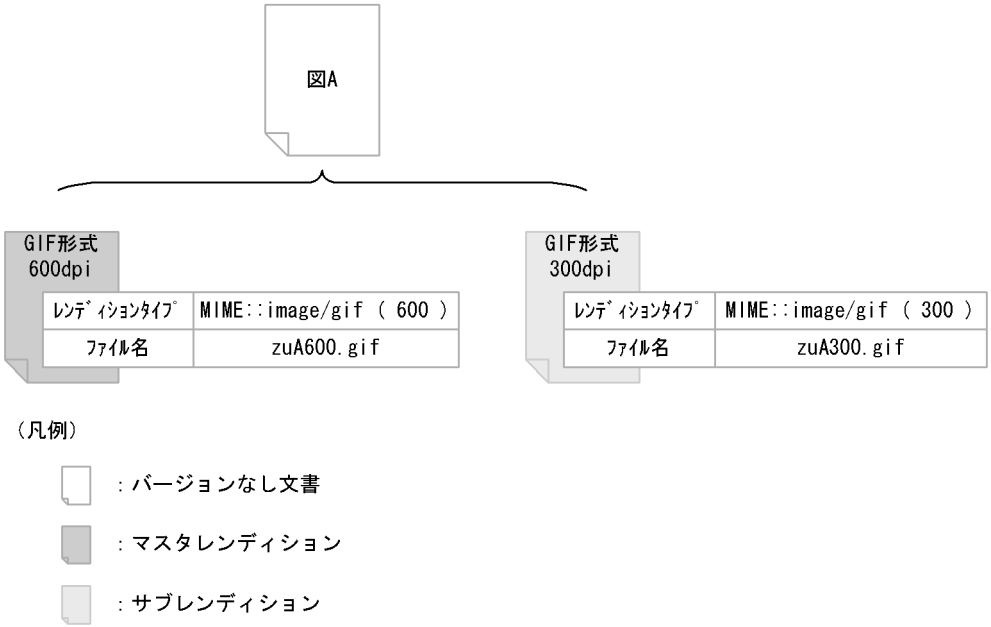
● dbrProp_RenditionStatus プロパティ

● dbrProp_ContentType プロパティ

なお，レンディションのプロパティとしてユーザ定義プロパティは設定できません。

ここでは，次の図に示すようなレンディションのプロパティの管理例を使用して，それぞれのプロパティについて説明します。

図 3-21　レンディションのプロパティの管理例



(1) レンディションタイプを表すプロパティ（dbrProp_RenditionType プロパティ）

図 3-21 の例では，「図 A」というバージョンなし文書のマスタレンディションとサブレンディションに，同じ GIF 形式でも異なる解像度（マスタレンディションは 600dpi，サブレンディションは 300dpi）で保存した文書ファイルを登録しています。マルチレンディション管理では，同じレンディションタイプを重複して登録できません。しかし，例えば，MIME 名でレンディションタイプを統一している場合，"MIME:: タイプ / サブタイプ (コメント)" の形式でレンディションタイプを指定できるため，同じ拡張子のファイルでもレンディションタイプを重複させないで登録できます。この例の場合，"MIME::image/gif (600)" および "MIME::image/gif (300)" のように，コメントまで指定したレンディションタイプを設定して二つの GIF 形式のレンディションを登録しています。

レンディションタイプは，文書の新規作成時や文書ファイルの更新時またはレンディションの追加時に指定します。このとき，レンディションタイプの指定を省略すると，レンディションタイプは設定されません。ファイルの拡張子から判断されて設定されることはありません。したがって，文書の新規作成時または更新時には，適切なレンディションタイプを設定してください。この例のように拡張子が同じでもファイルの保存状態などによって別々のレンディションに登録したい場合は，ファイルの拡張子だけでなく，コメントまで指定したレンディションタイプを明示的に指定してください。

マスタレンディションのレンディションタイプの値は，文書（バージョンなし文書またはバージョン付き文書）の新規作成時または文書ファイルの更新時に，文書の dbrProp_RenditionType プロパティの値とし

て設定されます。したがって，文書の dbrProp_RenditionType プロパティを GetPropertyValues メソッドまたは GetPropertyValuesAndLock メソッドで取得すると，マスタレンディションのレンディションタイプの値が取得されます。

また，UpdateContentAndRenditionType メソッドで，引数に変更前のレンディションタイプを指定しない形式を使用した場合は，マスタレンディションのレンディションタイプの値が更新されます。引数に変更前のレンディションタイプを指定する形式を使用した場合，指定したレンディションタイプを持つマスタレンディションまたはサブレンディションのレンディションタイプの値が更新されます。

サブレンディションのレンディションタイプは，GetRenditionListAndLock メソッドまたは GetRenditionList メソッドでレンディション一覧を取得する時にだけ取得できます。GetPropertyValuesAndLock メソッドまたは GetPropertyValues メソッドや検索結果一覧の情報としては取得できません。

次に，レンディションタイプの指定方法について説明します。

レンディションタイプの指定方法

レンディションタイプは，レンディションのコンテントの形式を表すプロパティです。このプロパティの値によって，それぞれのレンディションを識別します。このため，マルチレンディション文書の各レンディションのレンディションタイプは，一意である必要があります。
レンディションタイプには任意の文字列を指定できます。レンディションタイプは，パラメタやコメントおよびその間に含まれる区切り記号や空白，大文字・小文字の違いなども含めて区別されます。
レンディションタイプには，MIME 名を使用することをお勧めします。
なお，CreateObject メソッドや UpdateContent メソッドなどで全文検索インデクスを作成する場合には，レンディションタイプは「MIME:text/」で始まる値にしてください。
ここでは，MIME 名の推奨形式について説明します。

MIME::**タイプ**/**サブタイプ**△〔;△**パラメタ**△〔;△**パラメタ**△〕…〕〔△(**コメント群**)〔△(**コメント群**)〕…〕

パラメタ ::= 属性 "=" 値
コメント群 ::= コメント | (コメント群)

タイプ，サブタイプおよび属性に指定できる文字

タイプ，サブタイプおよびパラメタの属性を指定するときは，ASCII-7Bit の文字のうち，次の文字を除いた文字を使用して指定してください。
( ) < > @ , ; : ¥ " / [ ] / = および 0x1F 以下の値

属性の値に指定できる文字

パラメタの属性の値を指定するときは，属性と同じ形式の文字列を指定するかまたは ASCII-7Bit の文字のうち，次の文字を除いた文字を使用して指定してください。
( ) < > @ , ; : " / [ ] / = および 0x1F 以下の値
CR のあとには必ず LF を指定してください。

コメント

コメントを指定するときは，ASCII-7Bit 文字列で指定してください。ただし，"(" ")" を指定する場合は，"¥" を直前に指定してエスケープしてください。値として "¥" を指定したい場合は "¥¥" と指定します（"¥" は次に指定した文字が構文上の区切りでないことを示すために使用します。すなわち，"¥" の次に指定した文字は，そのままのデータとして扱われます)。CR のあとには必ず LF を指定してください。
△は空白 (0x20)0 個以上です。

(2) ファイル名を表すプロパティ（dbrProp_RetrievalName プロパティ）

各レンディションには，ファイル名を表す dbrProp_RetrievalName プロパティを設定できます。dbrProp_RetrievalName プロパティは，文書の新規作成時や文書ファイルの更新時またはレンディションの追加時に，指定した文書ファイルのファイル名が設定されます。図 3-21 の例ではマスタレンディションには zuA600.gif，サブレンディションには zuA300.gif という値が設定されています。また，dbrProp_RetrievalName プロパティの値は，登録済みの各レンディションのレンディションタイプを指定して PutRenditionPropertyValues メソッドで設定・更新できます。マスタレンディションの dbrProp_RetrievalName プロパティは，PutPropertyValues メソッドでも設定できます。GetPropertyValues メソッドや検索結果として取得するプロパティとして dbrProp_RetrievalName プロパティを指定すると，マスタレンディションの dbrProp_RetrievalName プロパティの値が取得されます。ただし，dbrProp_RetrievalName プロパティは，属性検索の条件としては指定できません（例えば，「マスタレンディションの文書ファイルのファイル名が "zuA600.gif" である文書を検索する」のように指定できません）。サブレンディションの dbrProp_RetrievalName プロパティの値は，GetRenditionListAndLock メソッドまたは GetRenditionList メソッドでレンディション一覧を取得する時にだけ取得できます。

(3) レンディションの更新状態を表すプロパティ（dbrProp_RenditionStatus プロパティ）

マルチレンディション管理機能は，同じ内容を持つ複数のファイル形式を一つの文書として管理する機能です。このため，マスタレンディションとサブレンディションは，常に同じ内容にしておく必要があります。マスタレンディションのコンテントを更新した場合には，サブレンディションのコンテントも更新する必要があります。

マスタレンディションに対するサブレンディションの更新状態は，CdbrDocument クラスのプロパティである dbrProp_RenditionStatus プロパティによって表されます。このプロパティは，Integer32 型の 4 バイトの値として表されます。上位 2 バイトと下位 2 バイトで，それぞれ意味を持ちます。

サブレンディションの更新状態は，dbrProp_RenditionStatus プロパティの下位 2 バイトで確認できます。下位 2 バイトによって表される内容を**状態フラグ**といいます。この値は，マスタレンディションまたはサブレンディションのコンテントを更新した時に，DocumentBroker によって設定されます。

サブレンディションのコンテントを DocumentBroker Rendering Option のレンディション変換機能を使用して作成，登録するかどうかの設定は，dbrProp_RenditionStatus プロパティの上位 2 バイトで表されます。また，レンディション変換でエラーが発生したことを示す値も，上位 2 バイトの値として設定されます。上位 2 バイトによって表される内容を**変換フラグ**といいます。なお，変換フラグは，DocumentBroker Rendering Option を使用しない場合は使用しません。

マスタレンディションの場合，dbrProp_RenditionStatus プロパティの値は常に「DBR_RENDSTATUS_MASTERREND」（00000000：4 バイトの 16 進数）です。

次に，サブレンディションの状態フラグと変換フラグについて説明します。

(a) 状態フラグ

状態フラグは，dbrProp_RenditionStatus プロパティの下位 2 バイトの値として設定されます。この値は DocumentBroker によって設定されます。

サブレンディションの dbrProp_RenditionStatus プロパティの下位 2 バイトに設定される値を次の表に示します。プロパティのデータ型は，Integer32 型です。

表 3-2 dbrProp_RenditionStatus プロパティの下位 2 バイトの値

定数	対応する値（16 進数値）	説明
DBR_RENDSTATUS_NO_SUBREND	0001	指定したサブレンディションに対応するマスタレンディションのコンテントは登録済みですが，サブレンディションのコンテントが登録されていません。
DBR_RENDSTATUS_SUBREND_EXIST	0002	指定したサブレンディションに対応するマスタレンディションおよびサブレンディションのコンテントが登録されています。
DBR_RENDSTATUS_MASTERREND_UPDATE	0004	指定したサブレンディションに対応するマスタレンディションが更新されていますが，サブレンディションが更新されていないため，マスタレンディションとサブレンディションの内容が一致しません。

(b) 変換フラグ

変換フラグは，dbrProp_RenditionStatus プロパティの上位 2 バイトの値として表されます。この値は PutRenditionPropertyValues メソッドによって変更できます。また，DocumentBroker Rendering Option によるレンディション変換実行時にエラーが発生した場合には，変換エラーを示す値が設定されます。

サブレンディションの dbrProp_RenditionStatus プロパティの上位 2 バイトの値を次の表に示します。プロパティのデータ型は，Integer32 型です。

表 3-3 dbrProp_RenditionStatus プロパティの上位 2 バイトの値

定数	対応する値（16 進数値）	説明
DBR_RENDSTATUS_CONVERT_NOTREQUIRED	0000	DocumentBroker Rendering Option によるレンディション変換の対象になりません。
DBR_RENDSTATUS_CONVERT_REQUIRED	0001	DocumentBroker Rendering Option によるレンディション変換の対象になります。
DBR_RENDSTATUS_CONVERT_ERROR	0002	DocumentBroker Rendering Option で変換エラーが発生しました。

注意

変換エラーが発生した場合，プロパティを更新するメソッド（PutRenditionPropertyValues メソッド）によって変換フラグを DBR_RENDSTATUS_CONVERT_NOTREQUIRED または DBR_RENDSTATUS_CONVERT_REQUIRED に変更することはできません。この場合は，サブレンディションまたはマスタレンディションのコンテントを，UpdateContent メソッドまたは UpdateContentAndRenditionType メソッドによって更新してください。これによって，変換フラグが遷移します。

なお，状態フラグによって，変換フラグが取る値が異なります。

状態フラグと変換フラグの組み合わせについて，次の表に示します。

表 3-4 状態フラグと変換フラグの組み合わせ

状態フラグ	変換フラグ
DBR_RENDSTATUS_SUBREND_EXIST	• DBR_RENDSTATUS_CONVERT_NOTREQUIRED • DBR_RENDSTATUS_CONVERT_REQUIRED

状態フラグ	変換フラグ
DBR_RENDSTATUS_NO_SUBREND	• DBR_RENDSTATUS_CONVERT_REQUIRED • DBR_RENDSTATUS_CONVERT_ERROR
DBR_RENDSTATUS_MASTERREND_UPDATE	• DBR_RENDSTATUS_CONVERT_NOTREQUIRED • DBR_RENDSTATUS_CONVERT_REQUIRED • DBR_RENDSTATUS_CONVERT_ERROR

例えば，レンディションを追加する時にファイルパスを指定しなかった場合，変換フラグに DocumentBroker Rendering Option によるレンディション変換を要求しない DBR_RENDSTATUS_CONVERT_NOTREQUIRED を指定することはできません。

また，ファイルパスを指定して更新したいサブレンディションに対しては，マスタレンディションとサブレンディションが不一致であることを表す DBR_RENDSTATUS_MASTERREND_UPDATE の状態でも，レンディション変換を要求しない DBR_RENDSTATUS_CONVERT_NOTREQUIRED を指定することができます。

(4) dbrProp_RenditionStatus プロパティとメソッドの関係

dbrProp_RenditionStatus プロパティとメソッドの関係について説明します。

次のメソッドについて説明します。

● dbrProp_RenditionStatus プロパティの値を取得するメソッド

● dbrProp_RenditionStatus プロパティの値を遷移させるメソッド

● dbrProp_RenditionStatus プロパティの変換フラグを設定するメソッド

● dbrProp_RenditionStatus プロパティの値によって制限されるメソッド

(a) dbrProp_RenditionStatus プロパティの値を取得するメソッド

dbrProp_RenditionStatus プロパティの値は，次のメソッドで取得できます。

● GetRenditionListAndLock メソッド

● GetRenditionList メソッド

(b) dbrProp_RenditionStatus プロパティの値を遷移させるメソッド

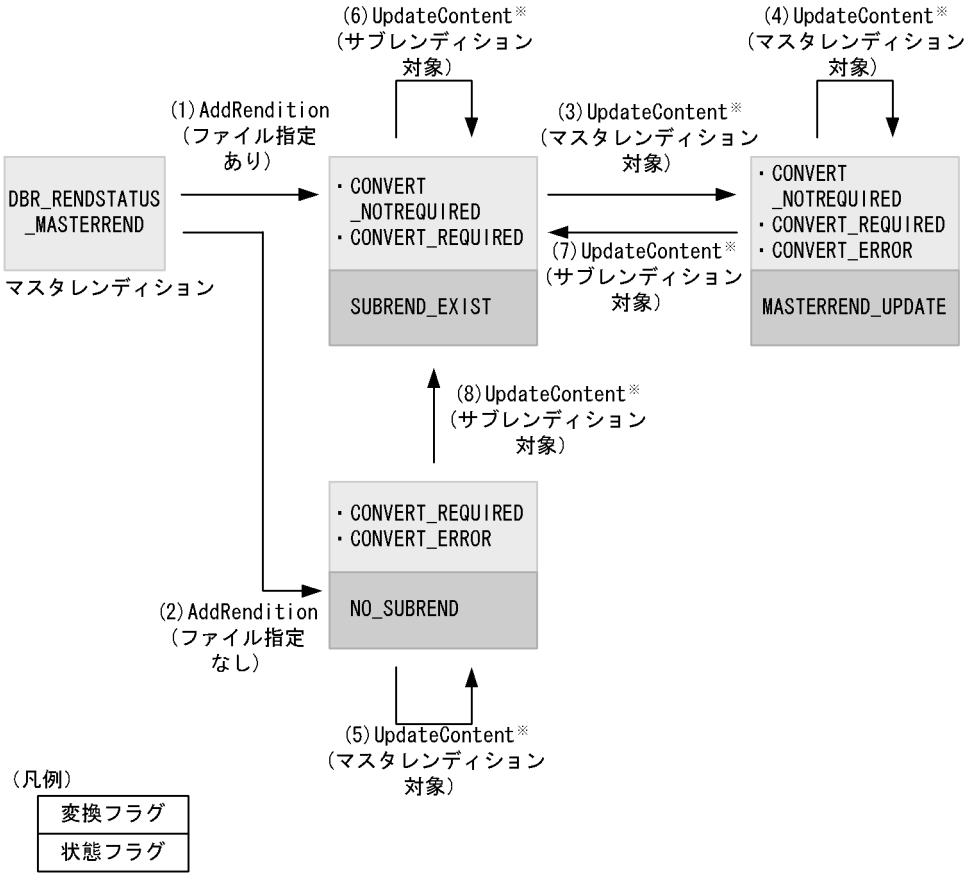
dbrProp_RenditionStatus プロパティの値は，次のメソッドをコールした時に，DocumentBroker によって変更されます。

● AddRendition メソッド

● UpdateContent メソッドまたは UpdateContentAndRenditionType メソッド

メソッドによるプロパティの値の遷移を，次の図に示します。図中の (1) から (8) の番号は，表 3-5 から表 3-7 の説明中の番号と対応しています。

図 3-22 メソッドによる dbrProp_RenditionStatus プロパティの値の遷移



変換フラグ（上位2バイト）

CONVERT_NOTREQUIRED：DBR_RENDSTATUS_CONVERT_NOTREQUIRED（0000：変換不要）

CONVERT_REQUIRED：DBR_RENDSTATUS_CONVERT_REQUIRED（0001：変換要）

CONVERT_ERROR：DBR_RENDSTATUS_CONVERT_ERROR（0002：変換エラー）

状態フラグ（下位2バイト）

NO_SUBREND：DBR_RENDSTATUS_NO_SUBREND
（0001：サブレンディションのコンテントなし）

SUBREND_EXIST：DBR_RENDSTATUS_SUBREND_EXIST
（0002：サブレンディションのコンテントあり。マスタレンディションの状態と一致）

MASTERREND_UPDATE：DBR_RENDSTATUS_MASTERREND_UPDATE
（0004：サブレンディションのコンテントあり。マスタレンディションの状態と不一致）

注※ UpdateContentメソッドまたはUpdateContentAndRenditionTypeメソッドに該当します。

図 3-22 の遷移の詳細を，メソッドごとに説明します。

● AddRendition メソッド

追加したサブレンディションのプロパティには，引数 pFilePath の指定に応じて，次の値が設定されます。

表 3-5 AddRendition メソッドによって設定される dbrProp_RenditionStatus プロパティの値

引数 pFilePath の値	変換フラグ（上位 2 バイト）	状態フラグ（下位 2 バイト）
サブレンディションに登録するファイル ((1) の遷移)	DBR_RENDSTATUS _CONVERT_NOTREQUIRED	DBR_RENDSTATUS _SUBREND_EXIST

引数 pFilePath の値	変換フラグ（上位 2 バイト）	状態フラグ（下位 2 バイト）
NULL （lConvertType に DBR_CONVERT_TYPE_BATCH を指定した場合） （(2) の遷移）	DBR_RENDSTATUS _CONVERT_REQUIRED	DBR_RENDSTATUS _NO_SUBREND

● **UpdateContent メソッドまたは UpdateContentAndRenditionType メソッド（マスタレンディションを更新する場合）**

メソッドをコールした時点の状態フラグに応じて，次のように遷移します。

表 3-6 UpdateContent メソッドまたは UpdateContentAndRenditionType メソッドによる dbrProp_RenditionStatus プロパティの値の遷移（マスタレンディションを更新する場合）

メソッドをコールした時点の状態フラグ	変換フラグ（上位 2 バイト）	メソッドをコールしたあとの状態フラグ（下位 2 バイト）
DBR_RENDSTATUS _SUBREND_EXIST （(3) の遷移）	DBR_RENDSTATUS_CONVERT_NOTREQUIRED の場合： DBR_RENDSTATUS_CONVERT_NOTREQUIRED （遷移しない）	DBR_RENDSTATUS _MASTERREND_UPDATE
	DBR_RENDSTATUS_CONVERT_REQUIRED の場合： DBR_RENDSTATUS_CONVERT_REQUIRED （遷移しない）	
DBR_RENDSTATUS _MASTERREND_UPDATE （(4) の遷移）	DBR_RENDSTATUS_CONVERT_ERROR の場合： DBR_RENDSTATUS_CONVERT_REQUIRED ※	DBR_RENDSTATUS _MASTERREND_UPDATE （遷移しない）
	DBR_RENDSTATUS_CONVERT_REQUIRED の場合： DBR_RENDSTATUS_CONVERT_REQUIRED （遷移しない）	
	DBR_RENDSTATUS_CONVERT_NOTREQUIRED の場合： DBR_RENDSTATUS_CONVERT_NOTREQUIRED （遷移しない）	
DBR_RENDSTATUS _NO_SUBREND （(5) の遷移）	DBR_RENDSTATUS_CONVERT_ERROR の場合： DBR_RENDSTATUS_CONVERT_REQUIRED ※	DBR_RENDSTATUS _NO_SUBREND （遷移しない）
	DBR_RENDSTATUS_CONVERT_REQUIRED の場合： DBR_RENDSTATUS_CONVERT_REQUIRED （遷移しない）	

注※

DocumentBroker Rendering Option によるレンディション変換でエラーが発生したあとにマスタレンディションを更新し直す場合は，更新するコンテントにエラーの要因がないかどうか確認してください。または，DocumentBroker Rendering Option を使用しないで，サブレンディションのコンテントを登録してください。

● UpdateContent メソッドまたは UpdateContentAndRenditionType メソッド（サブレンディションを更新する場合）

メソッドをコールした時点の状態フラグに応じて，次のように遷移します。

表 3-7 UpdateContent メソッドまたは UpdateContentAndRenditionType メソッドによる dbrProp_RenditionStatus プロパティの値の遷移（サブレンディションを更新する場合）

メソッドをコールした時点の状態フラグ	変換フラグ（上位 2 バイト）	メソッドをコールしたあとの状態フラグ（下位 2 バイト）
DBR_RENDSTATUS_SUBREND_EXIST（(6) の遷移）	メソッドをコールした時点の変換フラグ（遷移しない）	DBR_RENDSTATUS_SUBREND_EXIST（遷移しない）
DBR_RENDSTATUS_MASTERREND_UPDATE（(7) の遷移）	DBR_RENDSTATUS_CONVERT_NOTREQUIRED	DBR_RENDSTATUS_SUBREND_EXIST
DBR_RENDSTATUS_NO_SUBREND（(8) の遷移）	DBR_RENDSTATUS_CONVERT_NOTREQUIRED	DBR_RENDSTATUS_SUBREND_EXIST

(c) dbrProp_RenditionStatus プロパティの変換フラグを設定するメソッド

dbrProp_RenditionStatus プロパティの変換フラグは，PutRenditionPropertyValues メソッドで設定できます。

例えば，ファイルパスを指定して追加したサブレンディションのコンテントを，レンディション変換によって更新されるようにする場合や，一度 DocumentBroker Rendering Option によるレンディション変換でエラーが発生したサブレンディションを，マスタレンディションのコンテントを更新したあとでレンディション変換の対象から外す場合などに，明示的に変換フラグを変更してください。

設定できる値は次のどちらかです。

● DBR_RENDSTATUS_CONVERT_NOTREQUIRED

● DBR_RENDSTATUS_CONVERT_REQUIRED

ただし，次の設定はできません。

PutRenditionPropertyValues メソッドで変換フラグを設定できない場合

- 状態フラグが DBR_RENDSTATUS_NO_SUBREND の時，変換フラグに DBR_RENDSTATUS_CONVERT_NOTREQUIRED は設定できません。
  この場合は，UpdateContent メソッドまたは UpdateContentAndRenditionType メソッドをコールするか，DocumentBroker Rendering Option によるレンディション変換を実行して，サブレンディションのコンテントを登録してください。これによって変換フラグが DBR_RENDSTATUS_CONVERT_NOTREQUIRED に遷移します。
- 変換フラグが DBR_RENDSTATUS_CONVERT_ERROR の時，変換フラグに DBR_RENDSTATUS_CONVERT_NOTREQUIRED または DBR_RENDSTATUS_CONVERT_REQUIRED は設定できません。
  この場合は，マスタレンディションまたはサブレンディションに対して，UpdateContent メソッドまたは UpdateContentAndRenditionType メソッドをコールしてコンテントを更新してください。これによって，変換フラグが，DBR_RENDSTATUS_CONVERT_REQUIRED（マスタレンディションに対してメソッドをコールした場合）または DBR_RENDSTATUS_CONVERT_NOTREQUIRED（サブレンディションに対してメソッドをコールした場合）に遷移します。

(d) dbrProp_RenditionStatus プロパティの値によって制限されるメソッド

状態フラグによっては，ChangeMasterRendition メソッドの実行が制限されます。

ChangeMasterRendition メソッドの実行が制限される場合

- 状態フラグが DBR_RENDSTATUS_NO_SUBREND の場合，メソッドは実行できません。
  この場合は，サブレンディションに対して UpdateContent メソッドまたは UpdateContentAndRenditionType メソッドをコールしてコンテントを登録するか，DocumentBroker Rendering Option を使用してレンディション変換を実行してから，ChangeMasterRendition メソッドをコールしてください。
- 状態フラグが DBR_RENDSTATUS_MASTERREND_UPDATE の場合は，変換フラグが DBR_RENDSTATUS_CONVERT_NOTREQUIRED のときだけ，メソッドが実行できます。変換フラグが DBR_RENDSTATUS_CONVERT_REQUIRED または DBR_RENDSTATUS_CONVERT_ERROR のとき，メソッドは実行できません。
  この場合は，変換フラグを DBR_RENDSTATUS_CONVERT_NOTREQUIRED に変更するか，またはマスタレンディションとサブレンディションの状態を一致させてから，ChangeMasterRendition メソッドをコールしてください。

(e) DocumentBroker Rendering Option によるサブレンディション更新の例

dbrProp_RenditionStatus プロパティの値の遷移の例として，DocumentBroker Rendering Option によるサブレンディション更新の場合の例を，次の図に示します。

図 3-23　DocumentBroker Rendering Option によるサブレンディション更新の例

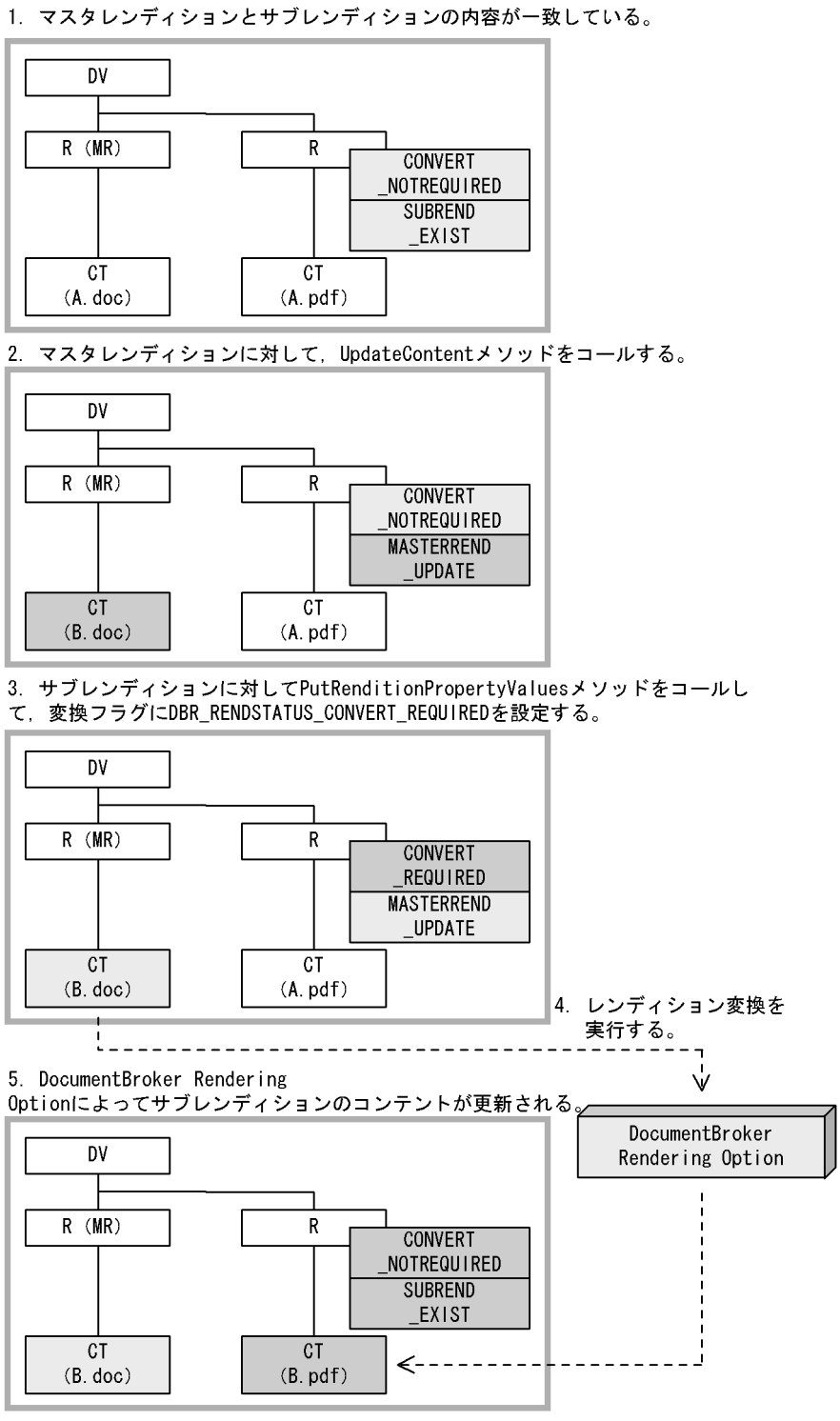


図 3-23 を説明します。

1. 図 3-23 の文書は，マスタレンディションとして Word 形式のファイル A.doc，サブレンディションとして PDF 形式のファイル A.pdf を持つ，バージョンなし文書です。1. の時，マスタレンディションとサブレンディションの内容は一致しています。状態フラグは DBR_RENDSTATUS_SUBREND_EXIST です。

なお，DV は DocVersion オブジェクト，R は Rendition オブジェクト，CT は ContentTransfer オブジェクトを表します。R(MR) はマスタレンディションを表します。

2. UpdateContent メソッドによって，マスタレンディションのコンテントを B.doc に更新します。このとき，マスタレンディションとサブレンディションの状態が不一致になるため，サブレンディションの状態フラグが DBR_RENDSTATUS_MASTERREND_UPDATE に遷移します。
3. PutRenditionPropertyValues メソッドによって，サブレンディションの dbrProp_RenditionStatus プロパティの変換フラグに，DBR_RENDSTATUS_CONVERT_REQUIRED を設定します。
4. DocumentBroker Rendering Option によってレンディション変換を実行します。3. で変換フラグに DBR_RENDSTATUS_CONVERT_REQUIRED を設定したサブレンディションは，レンディション変換の対象になります。レンディション変換は，マスタレンディションのコンテント（B.doc）を基に実行されます。
5. サブレンディションのコンテントとして，マスタレンディションの B.doc を基にレンディション変換したファイル B.pdf が登録されます。このとき，サブレンディションの状態フラグは，DBR_RENDSTATUS_SUBREND_EXIST に遷移します。

(5) レンディションのコンテント種別を表すプロパティ（dbrProp_ContentType プロパティ）

レンディションごとのコンテントの種別は，dbrProp_ContentType プロパティによって知ることができます。コンテントの種別から，レンディションのコンテントが次のうちのどれかを知ることができます。

- シングルファイル文書のコンテント
- マルチファイル文書のコンテント
- リファレンスファイル文書のコンテント
- File Link 文書のコンテント
- 上記以外のコンテント

このプロパティは，Integer32 型の 4 バイトの値として表されます。

ユーザは dbrProp_ContentType プロパティを設定できません。dbrProp_ContentType プロパティを参照する場合は，GetRenditionList メソッドおよび GetRenditionListAndLock メソッドを使用します。また，マスタレンディションの dbrProp_ContentType プロパティは，GetPropertyValues メソッド，GetPropertyValuesAndLock メソッド，または一部の一覧取得メソッドでも参照できます。

dbrProp_ContentType の値とコンテント種別の関係を次の表に示します。

表 3-8　dbrProp_ContentType の値とコンテント種別の関係

dbrProp_ContentType の値	対応する値	コンテント種別
DBR_CONTENTTYPE_CONTENT	0	ContentTransfer オブジェクト（シングルファイル文書であるコンテント）です。
DBR_CONTENTTYPE_FILELINK	1	ContentFileLink オブジェクト（File Link 文書であるコンテント）です。
DBR_CONTENTTYPE_MULTIFILE	2	ContentTransfers オブジェクト（マルチファイル文書であるコンテント）です。
DBR_CONTENTTYPE_REFERENCE	3	ContentReference オブジェクト（リファレンスファイル文書であるコンテント）です。
DBR_CONTENTTYPE_OTHER	-1	上記以外のコンテントを持つオブジェクトです。

3.4.4 マルチレンディション管理に関する操作

マルチレンディション管理に関する操作を実行する場合に使用するメソッドと，そのメソッドの発行順序について説明します。ここでは，次の操作について説明します。

- レンディションの追加
- レンディションの削除
- マスタレンディションの変更
- レンディション一覧の取得
- レンディションのプロパティの設定
- マルチレンディション文書のバージョンアップ

これらの操作を実行する前に，まず，文書空間と接続してトランザクションを開始してください。また，すべての操作は，マルチレンディション文書に接続した状態で実行します。マルチレンディション文書に接続するためには，次の準備をしてください。

準備

レンディションを追加する文書の構成要素である DMA オブジェクトの OIID を，検索などによってあらかじめ取得します。

- CdbrDocument オブジェクトにレンディションを追加する場合は，DMA オブジェクトの DocVersion オブジェクトの OIID を取得します。
- CdbrVersionableDocument オブジェクトにレンディションを追加する場合は，DMA オブジェクトの ConfigurationHistory オブジェクトの OIID を取得します。

それぞれの操作に使用するメソッドの詳細については，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。

(1) レンディションの追加

すでに登録されている文書（マスタレンディション）に対して，サブレンディションを追加する場合の操作について説明します。ここでは，DocumentBroker Rendering Option を使用しない場合と使用する場合について説明します。

(a) DocumentBroker Rendering Option を使用しない場合

1. **レンディションを追加する文書に接続します。**
   接続するオブジェクトに応じて，CdbrDocument クラスまたは CdbrVersionableDocument クラスの，SetOIID メソッドまたは ConnectObject メソッドをコールします。

2. **レンディションを追加します。**
   AddRendition メソッドをコールします。このとき，サブレンディションに登録するコンテントのファイルパスを URL 形式で指定します。また，追加するサブレンディションに設定するレンディションタイプと，追加するレンディションに設定するプロパティ（dbrProp_RetrievalName プロパティ）を指定します。

レンディションを追加するコールシーケンスの例を次に示します。

なお，このコールシーケンスは，UNIX の場合です。Windows の場合は，AddRendition メソッドの引数に指定するファイルパスを，「"file:///c:¥temp¥sample.txt"」のように指定してください。

レンディションを追加するコールシーケンス

```
pSession->Begin();
//…
//レンディションを追加する文書のOIIDを取得する
//レンディションに設定するdbrProp_RetrievalNameプロパティの値を
```

```
//構造体PropListに登録する
//…
CdbrDocument Doc;
//レンディションを追加する文書に接続する
Doc.SetOIID(pSession,pOIID);
//レンディションを追加して，プロパティを設定する
Doc.AddRendition("file:///tmp/sample.txt",
                 "MIME::text/plain",&PropList);
Doc.ReleaseObject();
//処理の確定
pSession->Commit();
```

（b）DocumentBroker Rendering Option を使用する場合

1. **レンディションを追加する文書に接続します。**
   接続するオブジェクトに応じて，CdbrDocument クラスまたは CdbrVersionableDocument クラスの，SetOIID メソッドまたは ConnectObject メソッドをコールします。

2. **レンディションを追加します。**
   AddRendition メソッドをコールします。メソッドは，レンディション変換の方法を指定する引数 lConvertType がある形式を使用します。
   それぞれの引数には，次の値を指定します。
   - pFilePath
     NULL を指定します。
   - pRenditionType
     作成するコンテントに対応するレンディションタイプを指定します。
   - pPropList
     追加するサブレンディションの dbrProp_RetrievalName プロパティの値を設定した構造体を指定します。
   - lConvertType
     DBR_CONVERT_TYPE_BATCH を指定します。

3. **DocumentBroker Rendering Option のレンディション変換コマンドを実行します。**

レンディションを追加するコールシーケンスの例を次に示します。

レンディションを追加するコールシーケンス

```
pSession->Begin();
//…
CdbrDocument Doc;
SDBR_PROP     Props;
SDBR_PROPLIST PropList;
//レンディションを追加する文書のOIIDを取得する
//レンディションに設定するdbrProp_RetrievalNameプロパティの値を
//構造体Propsに登録する
//…
//レンディションを追加する文書に接続する
Doc.SetOIID(pSession,pOIID);
//レンディションを追加して，プロパティを設定する
Doc.AddRendition(NULL,
                 "MIME::text/plain",&PropList,
                  DBR_CONVERT_TYPE_BATCH);
Doc.ReleaseObject();
//処理の確定
pSession->Commit();
```

（2） レンディションの削除

マルチレンディション文書のサブレンディションを削除する場合の操作について説明します。

1. **レンディションを削除する文書に接続します。**
   接続するオブジェクトに応じて，CdbrDocument クラスまたは CdbrVersionableDocument クラスの SetOIID メソッドまたは ConnectObject メソッドをコールします。
2. **レンディションを削除します。**
   DeleteRendition メソッドをコールします。

レンディションを削除するコールシーケンスの例を次に示します。

レンディションを削除するコールシーケンス

```
pSession->Begin();
//…
//レンディションを削除する文書のOIIDを取得する
//…
CdbrDocument Doc;
//レンディションを削除する文書に接続する
Doc.SetOIID(pSession,pOIID);
//レンディションを削除する
Doc.DeleteRendition("MIME::text/plain");
Doc.ReleaseObject();
//処理の確定
pSession->Commit();
```

(3) マスタレンディションの変更

マルチレンディション文書のマスタレンディションを変更する場合の操作について説明します。マスタレンディションを変更すると，それまでマスタレンディションだったレンディションは，サブレンディションに変更されます。

1. **マスタレンディションを変更する文書に接続します。**
   接続するオブジェクトに応じて，CdbrDocument クラスまたは CdbrVersionableDocument クラスの SetOIID メソッドまたは ConnectObject メソッドをコールします。
2. **マスタレンディションに変更するサブレンディションのレンディションタイプを指定して，マスタレンディションを変更します。**
   ChangeMasterRendition メソッドをコールします。

マスタレンディションを変更するコールシーケンスの例を次に示します。

マスタレンディションを変更するコールシーケンス

```
pSession->Begin();
//…
//マスタレンディションを変更する文書のOIIDを取得する
//…
CdbrDocument Doc;
//マスタレンディションを変更する文書に接続する
Doc.SetOIID(pSession,pOIID);
//マスタレンディションを変更する
Doc.ChangeMasterRendition("MIME::text/plain");
Doc.ReleaseObject();
//処理の確定
pSession->Commit();
```

(4) レンディション一覧の取得

マルチレンディション文書に登録されているレンディションの一覧を取得する場合の操作について説明します。

1. **レンディション一覧を取得する文書に接続します。**
   接続するオブジェクトに応じて，CdbrDocument クラスまたは CdbrVersionableDocument クラスの

SetOIID メソッドまたは ConnectObject メソッドをコールします。

2. **レンディション一覧を取得します。**
GetRenditionListAndLock メソッドまたは GetRenditionList メソッドをコールします。

レンディション一覧を取得するコールシーケンスの例を次に示します。

レンディション一覧を取得するコールシーケンス

```
pSession->Begin();
//…
//レンディション一覧を取得する文書のOIIDを取得する
//…
CdbrDocument Doc;
//レンディション一覧を取得する文書に接続する
Doc.SetOIID(pSession,pOIID);
//レンディション一覧を取得する
Doc.GetRenditionListAndLock(2,&PropDefList,
                            ppRenditionList, DMA_LOCK_READ);
Doc.ReleaseObject();
//処理の確定
pSession->Commit();
```

(5) レンディションのプロパティの設定

マルチレンディション文書の各レンディションの dbrProp_RetrievalName プロパティの値を設定する場合の操作について説明します。

1. **レンディションを削除する文書に接続します。**
接続するオブジェクトに応じて，CdbrDocument クラスまたは CdbrVersionableDocument クラスの，SetOIID メソッドまたは ConnectObject メソッドをコールします。

2. **レンディションにプロパティを設定します。**
PutRenditionPropertyValues メソッドをコールします。

レンディションにプロパティを設定するコールシーケンスの例を次に示します。

レンディションにプロパティを設定するコールシーケンス

```
pSession->Begin();
//…
//レンディションにプロパティを設定する文書のOIIDを取得する
//…
CdbrDocument  Doc;
SDBR_PROP     Props;
SDBR_PROPLIST PropList;
//プロパティ識別子にdbrProp_RetrievalNameを指定して,
//必要な情報を設定したプロパティ構造体Propsを作成しておく
//…
//レンディションにプロパティを設定する文書に接続する
Doc.SetOIID(pSession,pOIID);
//レンディションにプロパティを設定する
Doc.PutRenditionPropertyValues("MIME::text/plain",
                               &PropList);
Doc.ReleaseObject();
//処理の確定
pSession->Commit();
```

(6) マルチレンディション文書のバージョンアップ

マルチレンディション文書をバージョンアップする場合の操作について説明します。ここでは，DocumentBroker Rendering Option を使用しない場合と使用する場合について説明します。

(a) DocumentBroker Rendering Option を使用しない場合

1. **バージョンアップする文書に接続します。**
   CdbrVersionableDocument クラスの，SetOIID メソッドまたは ConnectObject メソッドをコールします。
2. **バージョン付き文書をチェックアウトします。**
   CdbrVersionable::CheckOut メソッドをコールします。これによって，次バージョンの追加を予約する，仮のバージョンが作成されます。このとき，仮のバージョンとしてコピーされるのは，マスタレンディションだけです。
3. **マスタレンディションのコンテントを更新します。**
   CdbrVersionableDocument::UpdateContent メソッドまたは UpdateContentAndRenditionType メソッドをコールします。
4. **仮のバージョンに，レンディションを追加します。**
   CdbrVersionableDocument::AddRendition メソッドをコールします。このとき，サブレンディションに登録するファイルパスを指定します。
5. **バージョン付き文書をチェックインします。**
   CdbrVersionableDocument::VersionCheckIn メソッドをコールします。これによって，仮のバージョンが最新バージョンとして確定します。

コールシーケンスの例を次に示します。なお，このコールシーケンスは，UNIX の場合です。Windows の場合は，UpdateContent メソッドの引数に指定するファイルパスを「"file:///c:¥temp¥sample.doc"」，AddRendition メソッドの引数に指定するファイルパスを，「"file:///c:¥temp¥sample.pdf"」のように指定してください。

マルチレンディション文書をバージョンアップするコールシーケンス（DocumentBroker Rendering Option を使用しない場合）

```
//あらかじめマスタレンディションを更新するファイルを基に，
// サブレンディションを更新するファイルを作成しておく
pSession->Begin();
//…
//バージョンアップする文書のOIIDを取得する
//…
CdbrVersionableDocument VrDoc;
//バージョンアップする文書に接続する
VrDoc.SetOIID(pSession,pOIID);
//文書をチェックアウトする
VrDoc.VersionCheckOut(&pReservedVerId);
//マスタレンディションのコンテントを更新する
VrDoc.UpdateContent("file:///tmp/sample.doc",NULL,
                    pReservedVerId, DBR_CREATE_INDEX);
//仮のバージョンにレンディションを追加する
//（ファイルパスには事前に作成しておいた
//  サブレンディションのファイルのパスを指定する）
VrDoc.AddRendition(pReservedVerId,"file:///tmp/sample.pdf,
                   "MIME::application/pdf",&PropList);
//文書をチェックインする
VrDoc.VersionCheckIn();
VrDoc.ReleaseObject();
pSession->Commit();
```

(b) DocumentBroker Rendering Option を使用する場合

1. **バージョンアップする文書に接続します。**
   CdbrVersionableDocument クラスの，SetOIID メソッドまたは ConnectObject メソッドをコールします。

2. **バージョン付き文書をチェックアウトします。**
   CdbrVersionable::CheckOut メソッドをコールします。これによって，次バージョンの追加を予約する，仮のバージョンが作成されます。このとき，仮のバージョンとしてコピーされるのは，マスタレンディションだけです。
3. **マスタレンディションのコンテントを更新します。**
   CdbrVersionableDocument::UpdateContent メソッドまたは UpdateContentAndRenditionType メソッドをコールします。
4. **仮のバージョンに，レンディションを追加します。**
   CdbrVersionableDocument::AddRendition メソッドをコールします。このとき，サブレンディションに登録するファイルパスを指定する引数 pFilePath には NULL を指定します。レンディション変換の方法を指定する引数 lConvertType には，DBR_CONVERT_TYPE_BATCH を指定します。
5. **DocumentBroker Rendering Option のレンディション変換を実行します。**
   これによって，手順 3. で登録したマスタレンディションのコンテントを基に，手順 4. で追加したサブレンディションのコンテントが作成され，登録されます。
6. **バージョン付き文書をチェックインします。**
   CdbrVersionableDocument::VersionCheckIn メソッドをコールします。これによって，仮のバージョンが最新バージョンとして確定します。

コールシーケンスの例を次に示します。なお，このコールシーケンスは，UNIX の場合です。Windows の場合は，UpdateContent メソッドの引数に指定するファイルパスを，「"file:///c:¥temp¥sample.doc"」のように指定してください。

マルチレンディション文書をバージョンアップするコールシーケンス（DocumentBroker Rendering Option を使用する場合）

```
pSession->Begin();
//…
//バージョンアップする文書のOIIDを取得する
//…
CdbrVersionableDocument VrDoc;
//バージョンアップする文書に接続する
VrDoc.SetOIID(pSession,pOIID);
//文書をチェックアウトする
VrDoc.VersionCheckOut(&pReservedVerId);
//マスタレンディションのコンテントを更新する
VrDoc.UpdateContent("file:///tmp/sample.doc",NULL,
                    pReservedVerId, DBR_CREATE_INDEX);
//仮のバージョンにレンディションを追加する
//（ファイルパスにはNULLを指定する）
VrDoc.AddRendition(pReservedVerId,NULL,
                   "MIME::application/pdf",
                   &PropList,DBR_CONVERT_TYPE_BATCH);
//DocumentBroker Rendering Optionを使用して
// レンディション変換を実行する
//…
//文書をチェックインする
VrDoc.VersionCheckIn();
VrDoc.ReleaseObject();
pSession->Commit();
```

3.5 文書のマルチファイル管理

この節では，文書のマルチファイル管理について説明します。

3.5.1 マルチファイル管理の概要

ここでは，マルチファイル管理の概要について説明します。

(1) 機能

マルチファイルの管理とは，一つの文書に複数のファイルを登録し，一括して管理することです。一つの文書で複数のファイルを管理することによって，次のような文書管理ができます。

- 複数のファイルから構成される文書を一括して登録する
- 複数のファイルから構成される文書を一括して取得する

バージョンなし文書およびバージョン付き文書は，マルチファイル文書として管理できます。ただし，マルチファイルを管理するバージョンなし文書およびバージョン付き文書では，マルチレンディション管理機能は使用できません。文書間リレーションを設定したり，コンテナの包含要素として管理したり，バージョンを管理したりなど，マルチレンディション以外の機能は使用できます。

また，マルチファイルを管理する場合と，しない場合とで，バージョンなし文書およびバージョン付き文書を構成する DMA オブジェクトが異なります。このため，バージョンなし文書およびバージョン付き文書の操作に使用するメソッドの引数が異なる場合があります。例えば，マルチファイルを管理しないバージョンなし文書は CdbrDocument::CreateObject メソッドの引数に登録するファイルのパス名を指定して作成し，マルチファイルを管理するバージョンなし文書は CdbrDocument::CreateObject メソッドの引数に登録するファイルのファイルパス情報が設定された SDBR_PATHLIST 構造体を指定して作成します。また，レンディションを追加する場合に使用する CdbrDocument::AddRendition メソッドまたは CdbrVersionableDocument::AddRendition メソッドのように，マルチファイルを管理するバージョンなし文書およびバージョン付き文書では使用できないメソッドもあります。詳細については，各メソッドの説明を参照してください。

(2) マルチファイル文書を構成する DMA オブジェクト

マルチファイルの管理では，バージョンなし文書およびバージョン付き文書をマルチファイル文書として管理できます。マルチファイルを管理する場合としない場合とで，バージョンなし文書およびバージョン付き文書を構成する DMA オブジェクトが異なります。

(a) マルチファイルを管理するバージョンなし文書を構成する DMA オブジェクト

マルチファイルを管理するバージョンなし文書を構成する DMA オブジェクトの種類を次に示します。ContentTransfers オブジェクト以外のオブジェクトについては，マルチファイルを管理しないバージョンなし文書と同様です。

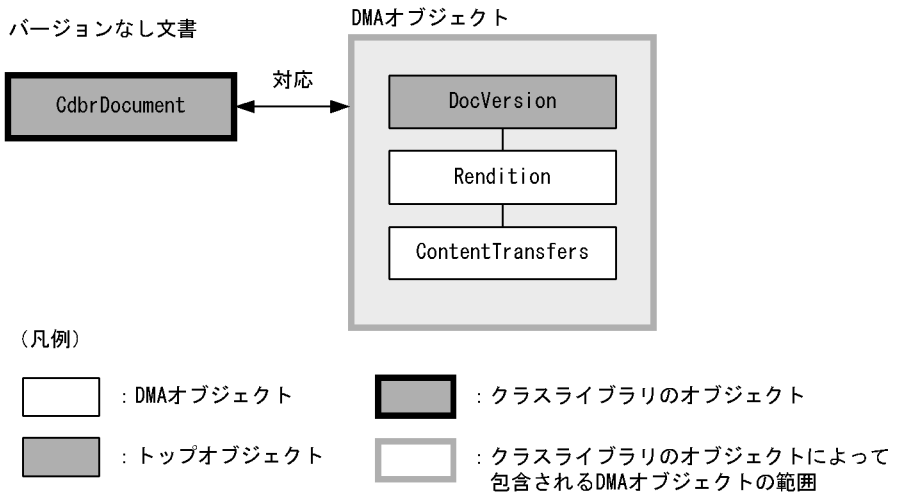
● DocVersion オブジェクト

● Rendition オブジェクト

● ContentTransfers オブジェクト
文書の実体であるコンテントデータを管理するオブジェクトです。複数のファイルを管理できます。

マルチファイルを管理するバージョンなし文書と DMA オブジェクトの関係を次の図に示します。

図 3-24　マルチファイルを管理するバージョンなし文書と DMA オブジェクトの関係



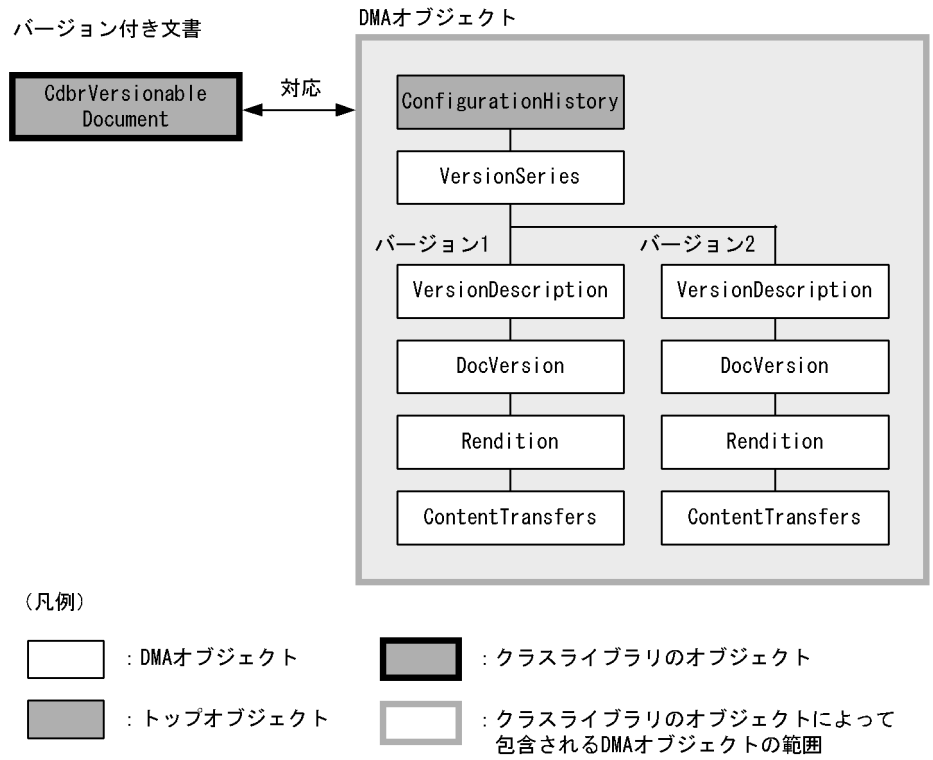
(b) マルチファイルを管理するバージョン付き文書を構成する DMA オブジェクト

マルチファイルを管理するバージョン付き文書を構成する DMA オブジェクトの種類を次に示します。ContentTransfers オブジェクト以外のオブジェクトについては，マルチファイルを管理しないバージョン付き文書と同様です。また，ContentTransfers オブジェクトについては，マルチファイルを管理するバージョンなし文書と同様です。

- ConfigurationHistory オブジェクト
- VersionSeries オブジェクト
- VersionDescription オブジェクト
- DocVersion オブジェクト
- Rendition オブジェクト
- ContentTransfers オブジェクト

マルチファイルを管理するバージョン付き文書と DMA オブジェクトの関係を次の図に示します。

図 3-25　マルチファイルを管理するバージョン付き文書と DMA オブジェクトの関係



3.5.2　マルチファイル管理に関する操作

マルチファイル管理に関する操作を実行する場合に使用するメソッドと，そのメソッドの発行順序について説明します。ここでは，次の操作について説明します。

- マルチファイル文書の作成
- マルチファイル文書の参照
- マルチファイル文書の更新

これらの操作を実行する前に，まず，文書空間と接続してトランザクションを開始してください。

ここでは，マルチファイル文書の操作について説明します。マルチファイルを管理するバージョン付き文書のバージョンの操作については，「3.3.3　バージョン管理機能の操作」を参照してください。

それぞれの操作に使用するメソッドの詳細については，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。

(1) マルチファイル文書の作成

マルチファイル文書を作成して，データベースに格納する場合に使用するメソッドと，そのメソッドの発行順序について説明します。なお，マルチファイル文書を登録する時，全文検索インデクスは作成できません。全文検索インデクスは，CdbrDocument::CreateIndex メソッドまたは CdbrVersionableDocument::CreateIndex メソッドを実行して作成してください。

準備

オブジェクトの構成要素になる DMA オブジェクト作成用の DMA クラスの識別子を指定した SDBR_DMAINFO 構造体（DMA オブジェクト生成用の構造体）を作成しておきます。

- バージョンなし文書を作成する場合は，SDBR_DMAINFO 構造体に，dmaClass_DocVersion クラ

スまたはそのサブクラスのクラス識別子を指定します。
- バージョン付き文書を作成する場合は，SDBR_DMAINFO 構造体に，dmaClass_ConfigurationHistory クラスまたはそのサブクラスの識別子と，dmaClass_DocVersion クラスまたはそのサブクラスの識別子を指定します。

また，登録するファイル数およびファイルパスの情報を持つ SDBR_PATHLIST 構造体（ファイルパス情報リスト構造体）を作成しておきます。

1. **文書を作成します。**

バージョンなし文書を作成する場合は，CdbrDocument::CreateObject メソッドをコールします。バージョン付き文書を作成する場合は，CdbrVersionableDocument::CreateObject メソッドをコールします。このとき，メソッドの引数に SDBR_PATHLIST 構造体（ファイルパス情報リスト構造体），レンディションタイプなどを指定します。
文書作成時に登録したレンディションがマスタレンディションとなります。
作成した文書をコンテナと関連づける場合については，「3.9　コンテナを使用した文書管理」を参照してください。

レンディションタイプについての注意事項

文書として登録するファイルの種類を表すレンディションタイプは，登録したファイルの拡張子からは設定されません。レンディションタイプは，CdbrDocument::CreateObject メソッドまたは CdbrVersionableDocument::CreateObject メソッドの引数として明示的に指定してください。この引数に指定した値がレンディションごとのファイルの種類を表すプロパティとして設定されます。
なお，マルチファイル文書では，レンディションタイプは変更できません。

マルチファイル文書を作成するコールシーケンスの例を次に示します。

マルチファイル文書を作成するコールシーケンス

```
// あらかじめ登録する内容を構造体 DMAClassList として作成しておく
//…
pSession->Begin();
CdbrDocument Doc;
SDBR_PATHLIST PathList;
//マルチファイル文書の作成および登録
Doc.CreateObject(pSession,
                 1, DMAClassList, PathList,
                 "MIME::application/x-edm-undefined", &pOIID);
Doc.ReleaseObject();
//処理の確定
pSession->Commit();
```

(2) マルチファイル文書の参照

すでにデータベースに登録されている文書とコンテントとして登録されているファイルを参照する場合に使用するメソッドと，そのメソッドの発行順序について説明します。マルチファイル管理を使用して登録した文書は，コンテントを一括して取得できます。

準備

オブジェクトの構成要素である DMA オブジェクトの OIID を，検索などによってあらかじめ取得しておきます。検索については「4.　オブジェクトの検索」を参照してください。
- バージョンなし文書に接続する場合は，構成要素である DMA オブジェクトの，DocVersion オブジェクトの OIID を取得しておきます。
- バージョン付き文書に接続する場合は，構成要素である DMA オブジェクトの，ConfigurationHistory オブジェクトの OIID を取得しておきます。

1. **参照するコンテントが含まれる文書に接続します。**

接続するオブジェクトに応じて，CdbrDocument クラスまたは CdbrVersionableDocument クラスの，SetOIID メソッドまたは ConnectObject メソッドをコールします。

2. **参照するコンテントの情報を取得します。**
CdbrDocument::GetContentInfo メソッドまたは CdbrVersionableDocument::GetContentInfo メソッドをコールします。

3. **文書のコンテントデータを，指定したパス名のファイルに複写します。**
取得するファイルの複写先のパス名を指定して，CdbrDocument::GetContentAndLock メソッドまたは CdbrVersionableDocument::GetContentAndLock メソッドをコールします。このとき，dbrDocument::GetContentInfo メソッドまたは CdbrVersionableDocument::GetContentInfo メソッドで取得した SDBR_CONTENTLIST 構造体（コンテント情報リスト構造体）を指定します。

マルチファイル文書を参照する場合の，コールシーケンスの例を次に示します。

なお，このコールシーケンスは，UNIX の場合です。Windows の場合は，GetContentAndLock メソッドの引数に指定するファイルパスを，「"file:///c:¥temp"」のように指定してください。

マルチファイル文書を参照するコールシーケンス

```
pSession->Begin();
//…
//検索によって参照するマルチファイル文書のOIIDを取得する
//…
CdbrDocument Doc;
SDBR_CONTENTLIST* pContentList=NULL;
//検索で取得したOIIDをオブジェクトに設定する
Doc.SetOIID(pSession, pOIID);
//readロックを設定して，マルチファイル文書の複数コンテントの情報を取得する
Doc.GetContentInfo(NULL, &pContentList,
                   DMA_LOCK_READ);
//readロックを設定して，マルチファイル文書の複数コンテントを取得する
Doc.GetContentAndLock("file:///tmp",
                      *pContentList, NULL,
                      DMA_LOCK_READ);
Doc.ReleaseObject();
dbrDelete(pContentList);
//処理の確定
pSession->Commit();
```

(3) マルチファイル文書の更新

すでにデータベースに登録されている文書とコンテントとして登録されているファイルを編集して更新する場合に使用するメソッドと，そのメソッドの発行順序について説明します。なお，(2) と同様に，あらかじめ接続する文書の構成要素である DMA オブジェクトの OIID を取得しておきます。マルチファイル管理を使用して登録した文書は，コンテントを一括して更新できます。

1. **更新するコンテントが含まれる文書に接続します。**
接続するオブジェクトに応じて，CdbrDocument クラスまたは CdbrVersionableDocument クラスの，SetOIID メソッドまたは ConnectObject メソッドをコールします。

2. **更新するコンテントの情報を取得します。**
CdbrDocument::GetContentInfo メソッドまたは CdbrVersionableDocument::GetContentInfo メソッドをコールします。

3. **文書のコンテントデータを，指定したパス名のファイルに複写します。**
取得するファイルの複写先のパス名を指定して，CdbrDocument::GetContentAndLock メソッドまたは CdbrVersionableDocument::GetContentAndLock メソッドをコールします。このとき，dbrDocument::GetContentInfo メソッドまたは CdbrVersionableDocument::GetContentInfo メソッド

で取得した SDBR_CONTENTLIST 構造体（コンテント情報リスト構造体）を指定します。

4. **ファイルを編集します。**

5. **編集したファイルで文書を更新します。**
   登録するファイルの情報を SDBR_PATHLIST 構造体（ファイルパス情報リスト構造体）で指定して，CdbrDocument::UpdateContent メソッドまたは CdbrVersionableDocument::UpdateContent メソッドをコールします。

マルチファイル文書を更新する場合の，コールシーケンスの例を次に示します。

なお，このコールシーケンスは，UNIX の場合です。Windows の場合は，GetContentAndLock メソッドおよび UpdateContent メソッドの引数に指定するファイルパスを，「"file:///c:¥temp"」のように指定してください。

マルチファイル文書を更新するコールシーケンス

```
pSession->Begin();
//…
//検索によって更新するマルチファイル文書のOIIDを取得する
//…
CdbrDocument Doc;
SDBR_CONTENTLIST* pContentList=NULL;
//検索で取得したOIIDをオブジェクトに設定する
Doc.SetOIID(pSession, pOIID);
//writeロックを設定して，マルチファイル文書の複数コンテントの情報を取得する
Doc.GetContentInfo(NULL, &pContentList,
                   DMA_LOCK_WRITE);
//writeロックを設定して，マルチファイル文書の複数コンテントを取得する
Doc.GetContentAndLock("file:///tmp",
                      *pContentList, NULL,
                      DMA_LOCK_WRITE);
//…
//取得したコンテントの内容の更新する
//…
//マルチファイル文書を更新する
Doc.UpdateContent(&PathList,
                  "MIME::application/x-edm-undefined");
Doc.ReleaseObject();
//処理の確定
pSession->Commit();
```

3.6 文書のリファレンスファイル管理

この節では，文書のリファレンスファイル管理について説明します。

3.6.1 リファレンスファイル管理の概要

ここでは，リファレンスファイル管理の概要について説明します。

(1) 機能

リファレンスファイルの管理とは，文書の実体であるコンテントを任意のディレクトリに格納し，文書のプロパティとコンテントロケーションだけをデータベースで管理する機能です。

この機能には，次のような特徴があります。

- ディスク移行時など，コンテント格納ディレクトリを変更する場合，データベースに影響を与えないで，管理するコンテント格納先の基点となるディレクトリパス（**コンテント格納先ベースパス**）だけの変更で対応できます。
- データベース容量を削減できます。

リファレンスファイルの管理では，バージョンなし文書およびバージョン付き文書をリファレンスファイル文書として管理できます。

リファレンスファイルの管理では，コンテントを任意のディレクトリに登録し，そのコンテントの格納先を，コンテントロケーションとして登録します。コンテントロケーションには，ユーザが管理するコンテントのコンテント格納先ベースパスを基点とする相対パス（**コンテント格納先パス**）を登録します。コンテントは，DocumentBroker の機能で操作します。

リファレンスファイルの管理を使用する場合，コンテントを任意のディレクトリへ格納し，文書のプロパティをデータベースで管理することから，データベースとコンテントの整合性を考慮する必要があります。このため，1 メソッド 1 トランザクションとすることを前提としています。文書作成を n 件連続で実行し，エラーが発生したときは，ロールバックしたあとにコンテントだけが存在する状態となります。また，文書削除を n 件連続で実行し，エラーが発生したときは，ロールバックしたあとにコンテントの存在しない文書が残ります。

また，リファレンスファイルの管理では，オブジェクトとコンテントが不整合となる場合があります。不整合となる場合の内容を，次の表に示します。

表 3-9 オブジェクトとコンテントが不整合となる場合

機能	エラー発生個所	不整合の内容	対処
文書の作成	コンテントを登録したあとのエラー	登録コンテントが残ります。	再度実行して文書を作成してください。
文書の更新	更新前のコンテントを削除したあと，更新後のコンテントを登録するまでのエラー	コンテントの存在しないオブジェクトが残ります。	文書を削除したあと，更新後のコンテントを使用して文書を作成してください。
	更新後のコンテントを登録したあとのエラー	コンテントの存在しないオブジェクトが残り，登録コンテントが残ります。	文書を削除したあと，更新後のコンテントを使用して文書を作成してください。
文書の削除	コンテントを削除したあとのエラー	コンテントの存在しないオブジェクトが残ります。	再度実行して文書を削除してください。
バージョンのチェックアウト	コンテントを複写したあとのエラー	複写したコンテントが残ります。	チェックアウトを取り消し，コンテントを削除したあと，ロールバックを実行してください。

機能	エラー発生個所	不整合の内容	対処
バージョンの削除	コンテントを削除したあとのエラー	コンテントの存在しないオブジェクトが残ります。	再度実行してバージョンを削除してください。
チェックアウトの取り消し	コンテントを削除したあとのエラー	コンテントの存在しないオブジェクトが残ります。	再度実行してチェックアウトを取り消してください。

(2) リファレンスファイル文書を構成する DMA オブジェクト

リファレンスファイルの管理では，バージョンなし文書およびバージョン付き文書をリファレンスファイル文書として管理できます。リファレンスファイルを管理する場合としない場合とで，バージョンなし文書およびバージョン付き文書を構成する DMA オブジェクトが異なります。

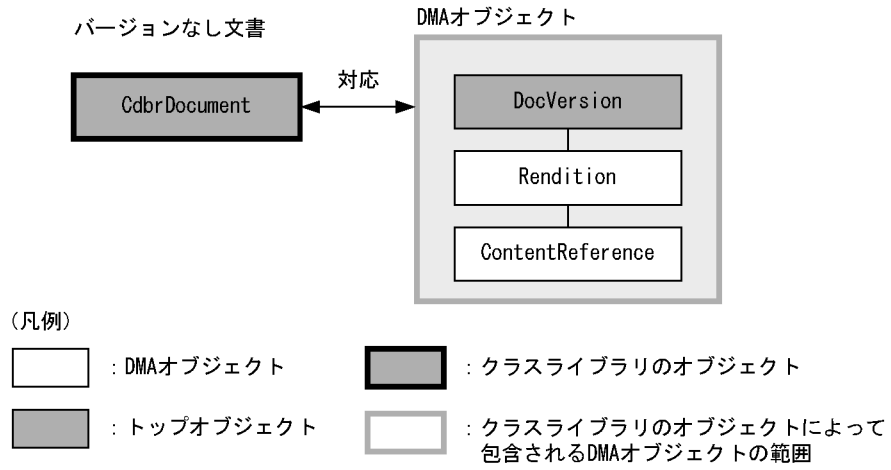
(a) リファレンスファイルを管理するバージョンなし文書を構成する DMA オブジェクト

リファレンスファイルを管理するバージョンなし文書を構成する DMA オブジェクトの種類を次に示します。ContentReference オブジェクト以外のオブジェクトについては，リファレンスファイルを管理しないバージョンなし文書と同様です。

- DocVersion オブジェクト
- Rendition オブジェクト
- ContentReference オブジェクト
  文書の実体であるコンテントデータの格納先情報を管理するオブジェクトです。

リファレンスファイルを管理するバージョンなし文書と DMA オブジェクトの関係を次の図に示します。

図 3-26 リファレンスファイルを管理するバージョンなし文書と DMA オブジェクトの関係



(b) リファレンスファイルを管理するバージョン付き文書を構成する DMA オブジェクト

リファレンスファイルを管理するバージョン付き文書を構成する DMA オブジェクトの種類を次に示します。ContentReference オブジェクト以外のオブジェクトについては，リファレンスファイルを管理しないバージョン付き文書と同様です。また，ContentReference オブジェクトについては，リファレンスファイルを管理するバージョンなし文書と同様です。

- ConfigurationHistory オブジェクト
- VersionSeries オブジェクト

● VersionDescription オブジェクト

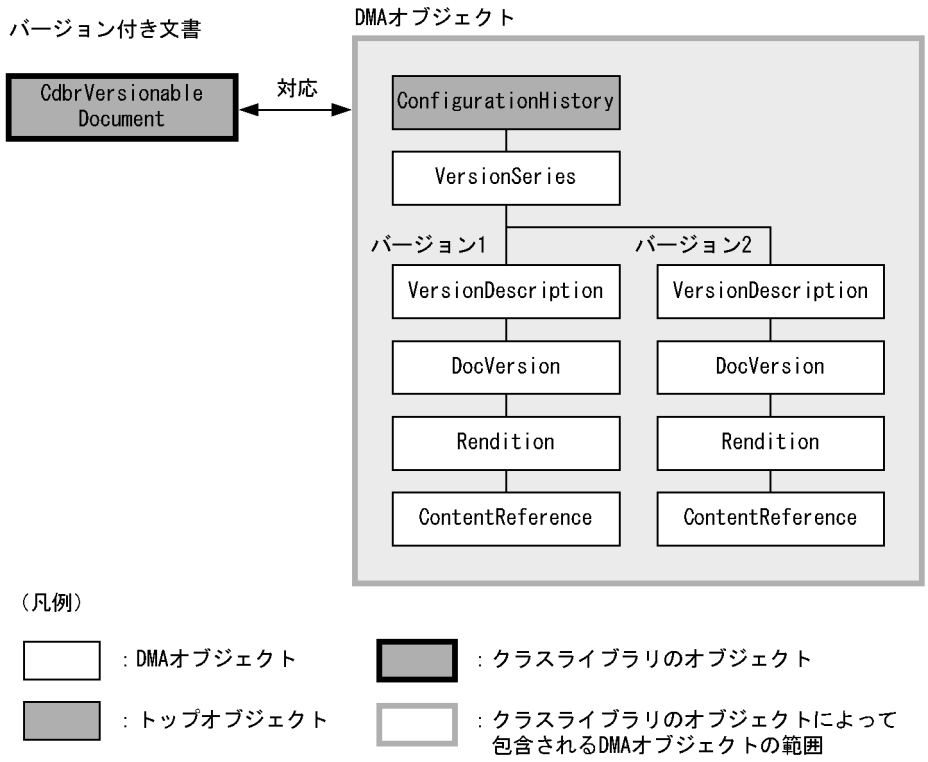
● DocVersion オブジェクト

● Rendition オブジェクト

● ContentReference オブジェクト

リファレンスファイルを管理するバージョン付き文書と DMA オブジェクトの関係を次の図に示します。

図 3-27　リファレンスファイルを管理するバージョン付き文書と DMA オブジェクトの関係



3.6.2　リファレンスファイル管理に関する操作

リファレンスファイル管理に関する操作を実行する場合に使用するメソッドと，そのメソッドの発行順序について説明します。ここでは，次の操作について説明します。

- リファレンスファイル文書の作成
- リファレンスファイル文書の参照
- リファレンスファイル文書の更新
- リファレンスファイル文書の削除

これらの操作を実行する前に，まず，文書空間と接続してトランザクションを開始してください。また，コンテントロケーションをコンテントの相対パスで管理するために，CdbrSession::SetReferencePath メソッドでコンテント格納先ベースパスを設定しておく必要があります。

ここでは，リファレンスファイル文書の操作について説明します。リファレンスファイルを管理するバージョン付き文書のバージョンの操作については，「3.3.3　バージョン管理機能の操作」を参照してください。

それぞれの操作に使用するメソッドの詳細については，マニュアル「DocumentBroker Version 3　クラス

ライブラリ　C++　リファレンス　基本機能編」を参照してください。

(1) リファレンスファイル文書の作成

リファレンスファイル文書を作成して，DocumentBroker サーバに格納する場合に使用するメソッドと，その発行順序について説明します。なお，リファレンス文書を登録するとき，登録するコンテントから全文検索インデクスは作成できません。全文検索インデクスを作成する場合は，CdbrDocument::CreateIndex メソッドまたは CdbrVersionableDocument::CreateIndex メソッドを実行して作成してください。

準備

オブジェクトの構成要素になる DMA オブジェクト作成用の DMA クラスの識別子を指定した SDBR_DMAINFO 構造体（DMA オブジェクト生成用の構造体）を作成しておきます。

- バージョンなし文書を作成する場合は，SDBR_DMAINFO 構造体に，dmaClass_DocVersion クラスまたはそのサブクラスのクラス識別子を指定します。
- バージョン付き文書を作成する場合は，SDBR_DMAINFO 構造体に，dmaClass_ConfigurationHistory クラスまたはそのサブクラスのクラス識別子と，dmaClass_DocVersion クラスまたはそのサブクラスのクラス識別子を指定します。

リファレンスファイルに必要な情報を持つ SDBR_REFERENCE_PATHINFO 構造体（リファレンスパス情報構造体）を作成しておきます。

1. **文書を作成します。**

バージョンなし文書を作成する場合は，CdbrDocument::CreateObject メソッドをコールします。バージョン付き文書を作成する場合は，CdbrVersionableDocument::CreateObject メソッドをコールします。このとき，メソッドの引数に SDBR_REFERENCE_PATHINFO 構造体（リファレンスパス情報構造体）を指定します。

リファレンスファイル文書を作成するコールシーケンスの例を次に示します。

リファレンスファイル文書を作成するコールシーケンス

```
//pTargetContentPathにコンテント格納先ベースパスを設定しておく
//…
pSession->Begin();
pSession->SetReferencePath(pTargetContentPath);
CdbrDocument Doc;
SDBR_REFERENCE_PATHINFO RefInfo;
//RefInfoにリファレンスファイル文書の登録に必要な情報を設定する
//…
//リファレンスファイル文書の作成および登録
Doc.CreateObject(pSession,
                 1, DMAClassList, RefInfo,
                 "MIME::application/x-edm-undefined", &pOIID);
Doc.ReleaseObject();
//処理の確定
pSession->Commit();
```

(2) リファレンスファイル文書の参照

すでにデータベースに登録されている文書と任意のディレクトリに登録されているコンテント（ファイル）を参照する場合に使用するメソッドと，そのメソッドの発行順序について説明します。

準備

オブジェクトの構成要素である DMA オブジェクトの OIID を，検索などによってあらかじめ取得しておきます。検索については，「4. オブジェクトの検索」を参照してください。

- バージョンなし文書に接続する場合は，構成要素である DMA オブジェクトの，DocVersion オブ

ジェクトの OIID を取得しておきます。
- バージョン付き文書に接続する場合は，構成要素である DMA オブジェクトの，ConfigurationHistory オブジェクトの OIID を取得しておきます。

取得するリファレンスファイルに必要な情報を持つ SDBR_REFERENCE_PATHINFO 構造体を作成しておきます。

1. **参照するコンテントが含まれる文書に接続します。**
   接続するオブジェクトに応じて，CdbrDocument クラスまたは CdbrVersionableDocument クラスの，SetOIID メソッドまたは ConnectObject メソッドをコールします。
2. **文書のコンテントデータを，指定したパス名のファイルに複写します。**
   取得するリファレンスファイルに必要な情報を持つ SDBR_REFERENCE_PATHINFO 構造体を指定して，CdbrDocument::GetContentAndLock メソッドまたは CdbrVersionableDocument::GetContentAndLock メソッドをコールします。

リファレンスファイル文書を参照する場合の，コールシーケンスの例を次に示します。

リファレンスファイル文書を参照するコールシーケンス

```
//pTargetContentPathにコンテント格納先ベースパスを設定しておく
//…
pSession->Begin();
pSession->SetReferencePath(pTargetContentPath);
//…
//検索によって参照するリファレンスファイル文書のOIIDを取得する
//…
CdbrDocument Doc;
SDBR_REFERENCE_PATHINFO RefInfo;
//RefInfoにリファレンスファイル文書のコンテント取得に必要な情報を設定する
//…
//検索で取得したOIIDをオブジェクトに設定する
Doc.SetOIID(pSession, pOIID);
//readロックを設定して，リファレンスファイル文書のコンテントを取得する
Doc.GetContentAndLock(RefInfo,
                      NULL, &lReferenceType, &pContentLocation,
                      DMA_LOCK_READ);
Doc.ReleaseObject();
dbrDelete(pContentLocation);
//処理の確定
pSession->Commit();
```

(3) リファレンスファイル文書の更新

すでにデータベースに登録されている文書と任意のディレクトリに登録されている文書のコンテントを編集して更新する場合に使用するメソッドと，そのメソッドの発行順序について説明します。なお，(2) と同様に，あらかじめ接続する文書の構成要素である DMA オブジェクトの OIID を取得しておきます。

1. **更新する文書に接続します。**
   接続するオブジェクトに応じて，CdbrDocument クラスまたは CdbrVersionableDocument クラスの，SetOIID メソッドまたは ConnectObject メソッドをコールします。
2. **文書のコンテントデータを，指定したパス名のファイルに複写します。**
   取得するリファレンスファイルに必要な情報を持つ SDBR_REFERENCE_PATHINFO 構造体を指定して，CdbrDocument::GetContentAndLock メソッドまたは CdbrVersionableDocument::GetContentAndLock メソッドをコールします。
3. **ファイルを編集します。**
4. **編集したファイルで文書を更新します。**

登録するファイルの情報を SDBR_REFERENCE_PATHINFO 構造体（リファレンスパス情報構造体）で指定して，CdbrDocument::UpdateContentAndRenditionType メソッドまたは CdbrVersionableDocument::UpdateContentAndRenditionType メソッドをコールします。

リファレンスファイル文書を更新する場合の，コールシーケンスの例を次に示します。

リファレンス文書を更新するコールシーケンス

```
//pTargetContentPathにコンテント格納先ベースパスを設定しておく
//…
pSession->Begin();
pSession->SetReferencePath(pTargetContentPath);
//…
//検索によって更新するマルチファイル文書のOIIDを取得する
//…
CdbrDocument Doc;
SDBR_REFERENCE_PATHINFO RefInfo;
//RefInfoにリファレンスファイル文書のコンテント取得に必要な情報を設定する
//…
//検索で取得したOIIDをオブジェクトに設定する
Doc.SetOIID(pSession, pOIID);
//writeロックを設定して，リファレンスファイル文書のコンテントを取得する
Doc.GetContentAndLock(RefInfo,
                      NULL, NULL, NULL,
                      DMA_LOCK_WRITE);
//…
//取得したコンテントの内容の更新する
//…
//RefInfoにリファレンスファイル文書の更新に必要な情報を設定する
//…
//リファレンスファイル文書を更新する
Doc.UpdateContentAndRenditionType(RefInfo,
                    "MIME::application/x-edm-undefined", NULL);
Doc.ReleaseObject();
//処理の確定
pSession->Commit();
```

(4) リファレンスファイル文書の削除

リファレンスファイル文書を削除するメソッドには，従来のメソッドとリファレンスファイル用のメソッドの 2 種類あります。ここでは，リファレンスファイル用のメソッドを使用して，リファレンスファイル文書を削除する場合の，コールシーケンスの例を次に示します。なお，(2) と同様に，あらかじめ接続する文書の構成要素である DMA オブジェクトの OIID を取得しておきます。

リファレンスファイル文書を削除するコールシーケンス

```
//pTargetContentPathにコンテント格納先ベースパスを設定しておく
//…
pSession->Begin();
pSession->SetReferencePath(pTargetContentPath);
//…
//検索によって削除する文書のOIIDを取得する
//…
CdbrDocument Doc;
SDBR_REFERENCE_PATHINFO RefInfo;
//RefInfoにリファレンスファイル文書の削除に必要な情報を設定する
//…
//検索で取得したOIIDをオブジェクトに設定する
Doc.SetOIID(pSession, pOIID);
//文書を削除する
Doc.RemoveObject(RefInfo);
Doc.ReleaseObject();
pSession->Commit();
```

3.7 File Link 連携機能を使用した文書管理

この節では，File Link 連携機能を使用した文書管理について説明します。HiRDB File Link の詳細については，マニュアル「HiRDB File Link」を参照してください。

3.7.1 File Link 連携機能の概要

ここでは，File Link 連携機能の概要について説明します。

(1) 機能

File Link **連携機能**とは，HiRDB のプラグインプログラムである HiRDB File Link と連携して文書を管理する機能です。文書の実体であるコンテントを，データベースと連携しているファイルサーバ上で管理します。データベースでは，ファイル名などのリンク情報だけを管理します。

この機能には，次のような特徴があります。

- 画像，音声などの容量の大きいコンテントを持つ文書が増加しても，データベースの負荷を最小限にして管理できます。
- 管理するコンテントが増加した場合も，ファイルサーバのディスク容量を増やしたり，ファイルサーバの数を増やしたりすることで対処できます。
- データベースとファイルサーバの内容の整合性については，HiRDB File Link の機能で確保できます。

File Link 連携機能が使えるのは，バージョンなし文書およびバージョン付き文書です。File Link 連携機能を使用する文書を，File Link **文書**といいます。

File Link 文書では，一つのファイルをコンテントとして管理するほか，クライアント環境のディレクトリをコンテントとしてまとめて登録して管理できます。この場合，指定したディレクトリおよびそのサブディレクトリ下に格納されているすべてのファイルを一つのコンテントとして登録できます。また，すでに HiRDB File Link のファイルサーバ上に登録されているデータも，DocumentBroker の文書のコンテントとして登録・管理できます。

File Link 連携機能は，マルチレンディション管理機能とあわせて使用できます。この場合，例えば，レンディションごとに，容量の大きいデータ形式のコンテントはファイルサーバに，容量の小さいデータ形式のコンテントはデータベースに，というように分けて管理できます。

(2) File Link 文書を構成する DMA オブジェクト

File Link 文書は，バージョンなし文書またはバージョン付き文書として管理します。File Link 連携機能を使用する場合としない場合とで，バージョンなし文書およびバージョン付き文書を構成する DMA オブジェクトが異なります。

(a) File Link 連携機能を使用するバージョンなし文書を構成する DMA オブジェクト

File Link 連携機能を使用するバージョンなし文書を構成する DMA オブジェクトの種類を次に示します。ContentFileLink オブジェクト以外のオブジェクトについては，File Link 連携機能を使用しないバージョンなし文書と同様です。

● DocVersion オブジェクト

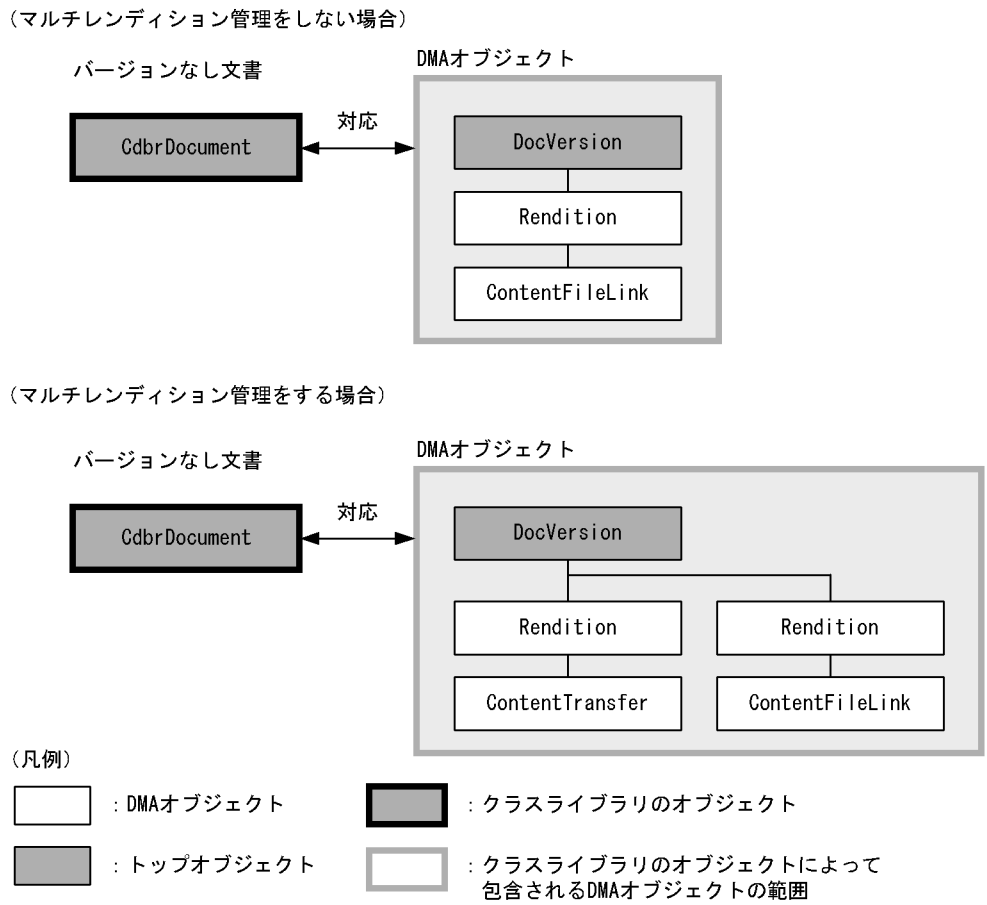
● Rendition オブジェクト

● ContentFileLink オブジェクト
文書の実体であるコンテントデータへのリンク情報を管理するオブジェクトです。なお，コンテント

データは，ファイルサーバ上にあります。

File Link 連携機能を使用するバージョンなし文書と DMA オブジェクトの関係を次の図に示します。

図 3-28　File Link 連携機能を使用するバージョンなし文書と DMA オブジェクトの関係



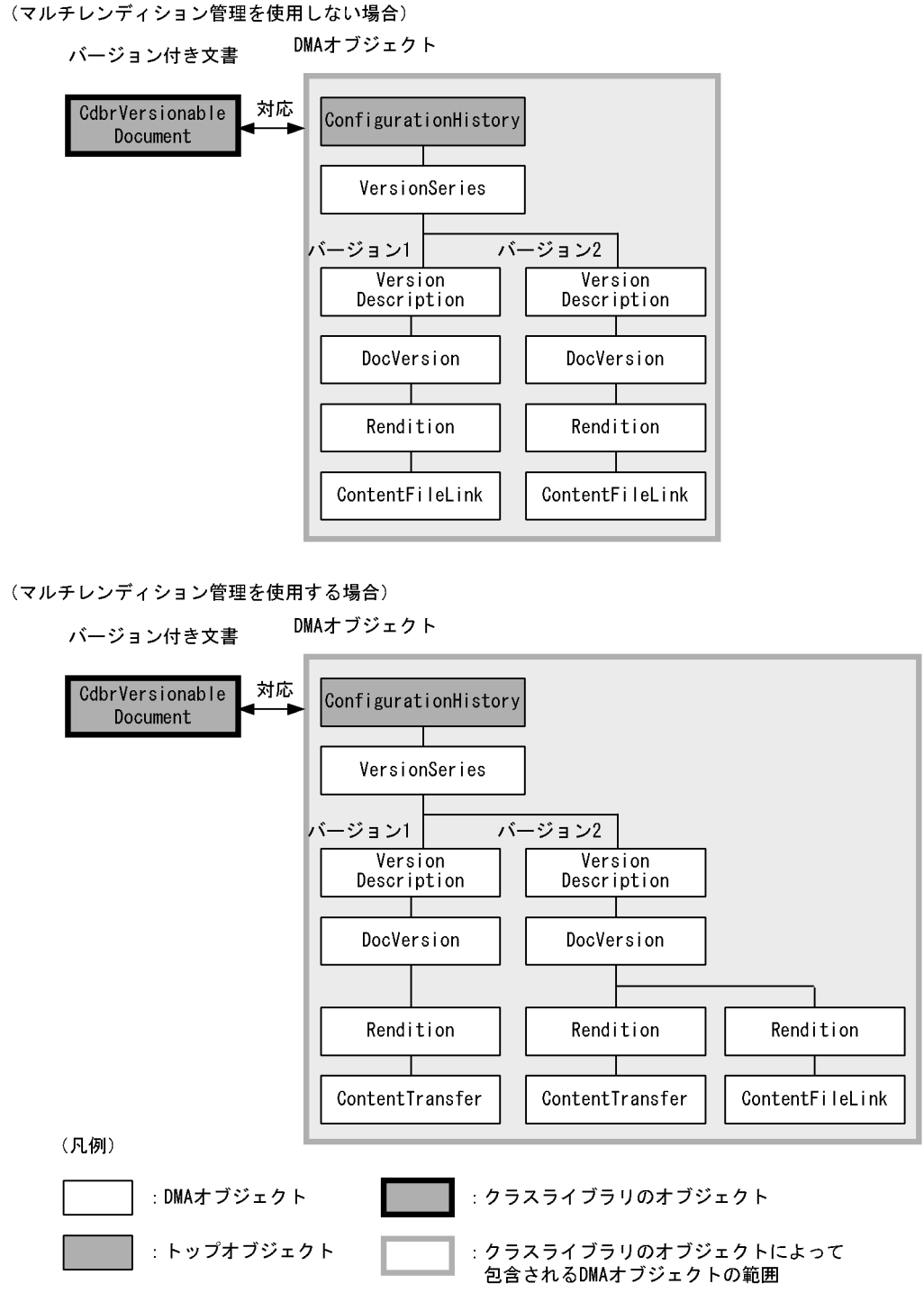
(b) File Link 連携機能を使用するバージョン付き文書を構成する DMA オブジェクト

File Link 連携機能を使用するバージョン付き文書を構成する DMA オブジェクトの種類を次に示します。ContentFileLink オブジェクト以外のオブジェクトについては，File Link 連携機能を使用しないバージョン付き文書と同様です。また，ContentFileLink オブジェクトについては，File Link 連携機能を使用するバージョンなし文書と同様です。

- ConfigurationHistory オブジェクト
- VersionSeries オブジェクト
- VersionDescription オブジェクト
- DocVersion オブジェクト
- Rendition オブジェクト
- ContentFileLink オブジェクト

File Link 連携機能を使用するバージョン付き文書と DMA オブジェクトの関係を次の図に示します。

図 3-29　File Link 連携機能を使用するバージョン付き文書と DMA オブジェクトの関係



3.7.2　File Link 連携機能に関する操作

File Link 連携機能に関する操作を実行する場合に使用するメソッドと，そのメソッドの発行順序について説明します。ここでは，次の操作について説明します。なお，操作を実行する前に，まず，文書空間と接続してトランザクションを開始してください。

- File Link 文書の作成

- File Link 文書の参照
- File Link 文書の更新
- File Link 文書のレンディションの追加

これらの操作は，CdbrDocument クラスまたは CdbrVersionableDocument クラスのメソッドを使用して実行します。これ以外の操作については，File Link 連携機能を使用しないバージョンなし文書およびバージョン付き文書と同じように操作できます。詳細は，「3.2.3　文書の操作」および「3.3.3　バージョン管理機能の操作」を参照してください。また，それぞれの操作に使用するメソッドの詳細については，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。

（1） File Link 文書の作成

File Link 文書を作成してデータベースおよびファイルサーバに格納する場合に使用するメソッドと，その発行順序について説明します。なお，File Link 文書を登録する時，登録するコンテントから全文検索インデクスは作成できません。全文検索インデクスを作成する場合は，CdbrDocument::CreateIndex メソッドまたは CdbrVersionableDocument::CreateIndex メソッドを実行して作成してください。

準備

オブジェクトの構成要素になる DMA オブジェクト作成用の DMA クラスの識別子を指定した SDBR_DMAINFO 構造体（DMA オブジェクト生成用の構造体）を作成しておきます。

- バージョンなし文書を作成する場合は，SDBR_DMAINFO 構造体に，dmaClass_DocVersion クラスまたはそのサブクラスのクラス識別子を指定します。
- バージョン付き文書を作成する場合は，SDBR_DMAINFO 構造体に，dmaClass_ConfigurationHistory クラスまたはそのサブクラスの識別子と，dmaClass_DocVersion クラスまたはそのサブクラスの識別子を指定します。

File Link 連携に必要な情報を指定した SDBR_FILELINK_PATHINFO 構造体（File Link パス情報構造体）を作成しておきます。この構造体には，登録するコンテントのファイルパスまたはディレクトリパス，FAM 名，登録するファイルサーバのディレクトリパス，文書を削除した時のファイルサーバ上のコンテントの扱い方，コンテントの名称の生成方法などを指定します。

1. **文書を作成します。**

バージョンなし文書を作成する場合は，CdbrDocument::CreateObject メソッドをコールします。バージョン付き文書を作成する場合は，CdbrVersionableDocument::CreateObject メソッドをコールします。このとき，メソッドの引数に SDBR_FILELINK_PATHINFO 構造体（File Link パス情報構造体）を指定します。

File Link 文書を作成するコールシーケンスの例を次に示します。

File Link 文書を作成するコールシーケンス

```
//あらかじめ登録する構造体DMAClassListとして作成しておく
//
//…
pSession->Begin();
CdbrDocument Doc;
SDBR_FILELINK_PATHINFO FileLinkInfo;
//FileLinkInfoにFile Link連携に必要な情報を設定する
//…
//File Link文書の作成および登録
Doc.CreateObject(pSession,
                 1, DMAClassList, FileLinkInfo,
                 "MIME::image/bmp", &pOIID);
Doc.ReleaseObject();
```

```
//処理の確定
pSession->Commit();
```

レンディションタイプについての注意事項

文書として登録するファイルの種類を表すレンディションタイプは，登録したファイルの拡張子からは設定されません。レンディションタイプは，CdbrDocument::CreateObject メソッドまたは CdbrVersionableDocument::CreateObject メソッドの引数として明示的に指定してください。この引数に指定した値がレンディションごとのファイルの種類を表すプロパティとして設定されます。

(2) File Link 文書の参照

すでにデータベースに登録されている文書とファイルサーバに登録されているコンテント（ファイル）を参照する場合に使用するメソッドと，そのメソッドの発行順序について説明します。

準備

オブジェクトの構成要素である DMA オブジェクトの OIID を，検索などによってあらかじめ取得しておきます。検索については「4. オブジェクトの検索」を参照してください。

- バージョンなし文書に接続する場合は，構成要素である DMA オブジェクトの，DocVersion オブジェクトの OIID を取得しておきます。
- バージョン付き文書に接続する場合は，構成要素である DMA オブジェクトの，ConfigurationHistory オブジェクトの OIID を取得しておきます。

取得した File Link パス情報の内容を格納するための SDBR_FILELINK_PATHINFO 構造体を作成しておきます。

1. **参照するコンテントへのリンク情報が含まれる文書に接続します。**
   接続するオブジェクトに応じて，CdbrDocument クラスまたは CdbrVersionableDocument クラスの，SetOIID メソッドまたは ConnectObject メソッドをコールします。
2. **文書のコンテントデータを，指定したパス名のファイルに複写します。**
   取得した情報を格納する SDBR_FILELINK_PATHINFO 構造体を指定して，CdbrDocument::GetContentAndLock メソッドまたは CdbrVersionableDocument::GetContentAndLock メソッドをコールします。
   なお，コンテントのリンク情報としてディレクトリパスが登録されている場合（SDBR_FILELINK_PATHINFO 構造体のメンバ pFilePath にディレクトリパスを指定して CreateObject メソッドをコールした文書を参照する場合）は，メンバ pFileName に指定した名前のディレクトリが作成され，そのディレクトリ下に登録したディレクトリおよびサブディレクトリ下のすべてのファイルが取得されます。

File Link 文書を参照する場合の，コールシーケンスの例を次に示します。

File Link 文書を参照するコールシーケンス

```
pSession->Begin();
//…
//検索によって参照するFile Link文書のOIIDを取得する
//…
CdbrDocument Doc;
SDBR_FILELINK_PATHINFO FileLinkPath;
//検索で取得したOIIDをオブジェクトに設定する
FileLinkPath.pFilePath="file:///tmp/temp.bmp"
Doc.SetOIID(pSession, pOIID);
//readロックを設定して，File Link文書のパス情報を取得する
//FileLinkPathのメンバpFileNameに指定したファイル
//またはディレクトリ下（この場合はファイル）に，
//取得したコンテントの内容が出力される
Doc.GetContentAndLock(FileLinkPath, NULL,
```

```
                        NULL, DMA_LOCK_READ);
Doc.ReleaseObject();
//処理の確定
pSession->Commit();
```

(3) File Link 文書の更新

すでにデータベースに登録されている文書のコンテントを編集して更新する場合に使用するメソッドと，そのメソッドの発行順序について説明します。なお，(2) と同様に，あらかじめ接続する文書の構成要素である DMA オブジェクトの OIID を取得しておきます。

SDBR_FILELINK_PATHINFO 構造体に設定する値については「(1) File Link 文書の作成」を参照してください。

1. **更新する文書に接続します。**
   接続するオブジェクトに応じて，CdbrDocument クラスまたは CdbrVersionableDocument クラスの，SetOIID メソッドまたは ConnectObject メソッドをコールします。
2. **文書のコンテントデータを，指定したパス名のファイルに複写します。**
   取得した情報を格納する SDBR_FILELINK_PATHINFO 構造体を指定して，CdbrDocument::GetContentAndLock メソッドまたは CdbrVersionableDocument::GetContentAndLock メソッドをコールします。
   なお，コンテントのリンク情報としてディレクトリパスが登録されている場合（SDBR_FILELINK_PATHINFO 構造体のメンバ pFilePath にディレクトリパスを指定して CreateObject メソッドをコールした文書を参照する場合）は，メンバ pFileName に指定した名前のディレクトリが作成され，そのディレクトリ下に登録したディレクトリおよびサブディレクトリ下のすべてのファイルが取得されます。
3. **ファイルを編集します。**
4. **編集したファイルで文書を更新します。**
   登録するファイルの情報を SDBR_FILELINK_PATHINFO 構造体（File Link パス情報構造体）で指定して，CdbrDocument::UpdateContent メソッドまたは CdbrVersionableDocument::UpdateContent メソッドをコールします。

File Link 文書を更新する場合の，コールシーケンスの例を次に示します。

File Link 文書を更新するコールシーケンス

```
pSession->Begin();
//…
//検索によって更新するFile Link文書のOIIDを取得する
//…
CdbrDocument Doc;
SDBR_FILELINK_PATHINFO FileLinkPath;
//検索で取得したOIIDをオブジェクトに設定する
Doc.SetOIID(pSession, pOIID);
//writeロックを設定して，File Link文書のパス情報を取得する
//取得したコンテントの内容の更新する
//…
//File Link文書を更新する
Doc.UpdateContent(FileLinkPath, NULL);
Doc.ReleaseObject();
//処理の確定
pSession->Commit();
```

(4) File Link 文書のレンディションの追加

File Link 文書のレンディションを追加するには，CdbrDocument クラスおよび

CdbrVersionableDocument クラスの AddRendition メソッドを使用します。CreateObject メソッドと同様に，追加するレンディションのファイルまたはディレクトリ情報を SDBR_FILELINK_PATHINFO 構造体に指定します。

すでにデータベースに登録されている文書に対して，File Link 連携機能を使用するレンディションを追加する場合に使用するメソッドと，そのメソッドの発行順序について説明します。なお，(2) と同様に，あらかじめ接続する文書の構成要素である DMA オブジェクトの OIID を取得しておきます。

1. **レンディションを追加する文書に接続します。**
   接続するオブジェクトに応じて，CdbrDocument クラスまたは CdbrVersionableDocument クラスの，SetOIID メソッドまたは ConnectObject メソッドをコールします。
2. **レンディションを追加します。**
   AddRendition メソッドをコールします。このとき，サブレンディションに登録するコンテントの情報として，SDBR_FILELINK_PATHINFO 構造体を指定します。

File Link 連携機能を使用するレンディションを追加するコールシーケンスの例を次に示します。

File Link 連携機能を使用するレンディションを追加するコールシーケンス

```
pSession->Begin();
//…
//レンディションを追加する文書のOIIDを取得する
//レンディションに設定するdbrProp_RetrievalNameプロパティの値を
//構造体PropListに登録する
//…
CdbrDocument Doc;
SDBR_FILELINK_PATHINFO FileLinkInfo;
//FileLinkInfoにFile Link連携に必要な情報を設定する
//…
//レンディションを追加する文書に接続する
Doc.SetOIID(pSession,pOIID);
//レンディションを追加して，プロパティを設定する
Doc.AddRendition(FileLinkInfo,
                 "MIME::image/bmp",&PropList);
Doc.ReleaseObject();
//処理の確定
pSession->Commit();
```

3.8 文書間リレーションを設定した文書管理

この節では，文書間リレーションを設定して文書を管理する機能について説明します。

3.8.1 文書間リレーション機能の概要

ここでは，文書間リレーション機能の概要について説明します。

(1) 機能

文書と文書を関連づけて管理する機能を，文書間リレーションといいます。

文書間リレーションは，次のような場合に使用できます。

● 参考文献のある論文などの文書を，参考文献とともに管理したい場合

● 別文書として登録しているテキストと図データを関連がわかるように管理したい場合

DocumentBroker では，二つの文書を**文書間リレーション**を利用して関連づけます。このとき，文書間リレーションを設定した文書を**リレーション元文書**，文書間リレーションを設定された文書を**リレーション先文書**といいます。

一つのリレーション元文書は，複数のリレーション先文書と関連づけることができます。また，一つのリレーション先文書も，複数のリレーション元文書から関連づけられることができます。これによって，リレーション元文書とリレーション先文書は，m：n（m，n は 1 以上の整数）で関連づけることができます。

一つのリレーション元文書と一つのリレーション先文書を関連づけるために，一つの文書間リレーションを設定します。したがって，m × n 個（m，n は 1 以上の整数）の文書間リレーションを設定することで，m 個のリレーション元文書と n 個のリレーション先文書を関連づけることができます。

(2) 文書間リレーションを表す DMA オブジェクト

文書間リレーションは，次の DMA オブジェクトによって表されます。

● Relationship オブジェクト
edmClass_Relationship クラスのオブジェクトです。このオブジェクトを，**リレーション**といいます。

リレーションは，リレーション元文書から操作します。Relationship オブジェクトは，リレーション元文書から文書間リレーションを設定するメソッドをコールした時に作成されます。また，リレーションには，プロパティも設定できます。

リレーションは，文書を表す次のクラス間のオブジェクトの関連づけに使用できます。

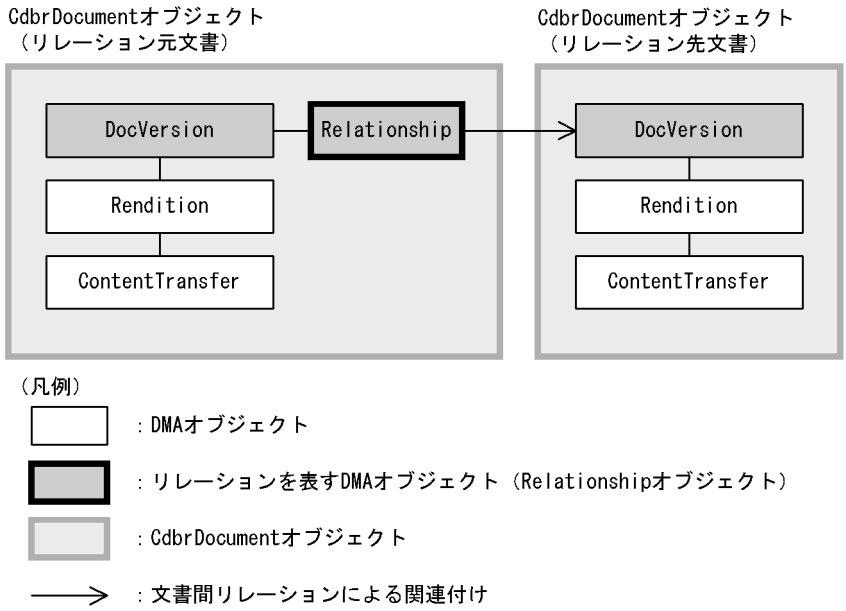
- CdbrDocument クラス
- CdbrVersionableDocument クラス

(a) バージョンなし文書間にリレーションを設定した場合の DMA オブジェクトの構成

バージョンなし文書に文書間リレーションを設定する場合，リレーションは，それぞれの文書の DMA オブジェクトの DocVersion オブジェクトと DocVersion オブジェクトを関連づけます。

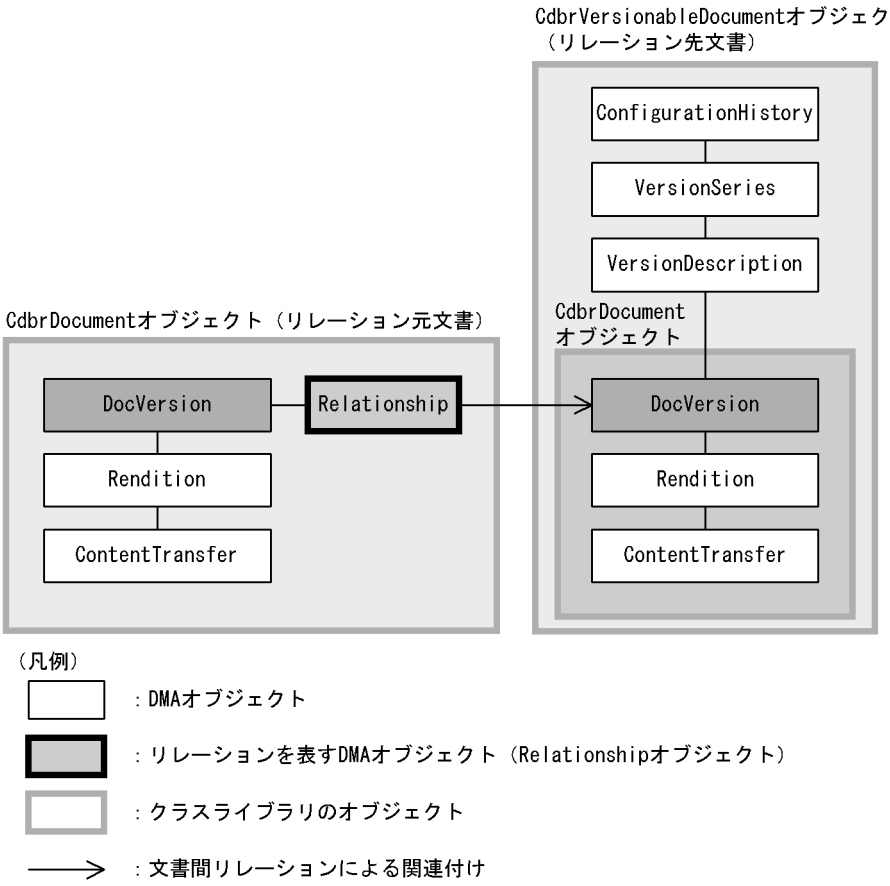
バージョンなし文書間にリレーションを設定した場合の DMA オブジェクトの構成を次の図に示します。

図 3-30　バージョンなし文書間に文書間リレーションを設定した場合の DMA オブジェクトの構成



また，図 3-31 のように，バージョンなし文書とバージョン付き文書の 1 バージョンに対応するバージョンなし文書の DocVersion オブジェクトに対してリレーションを設定することもできます。

図 3-31　バージョンなし文書とバージョン付き文書の間に文書間リレーションを設定した場合の DMA オブジェクトの構成



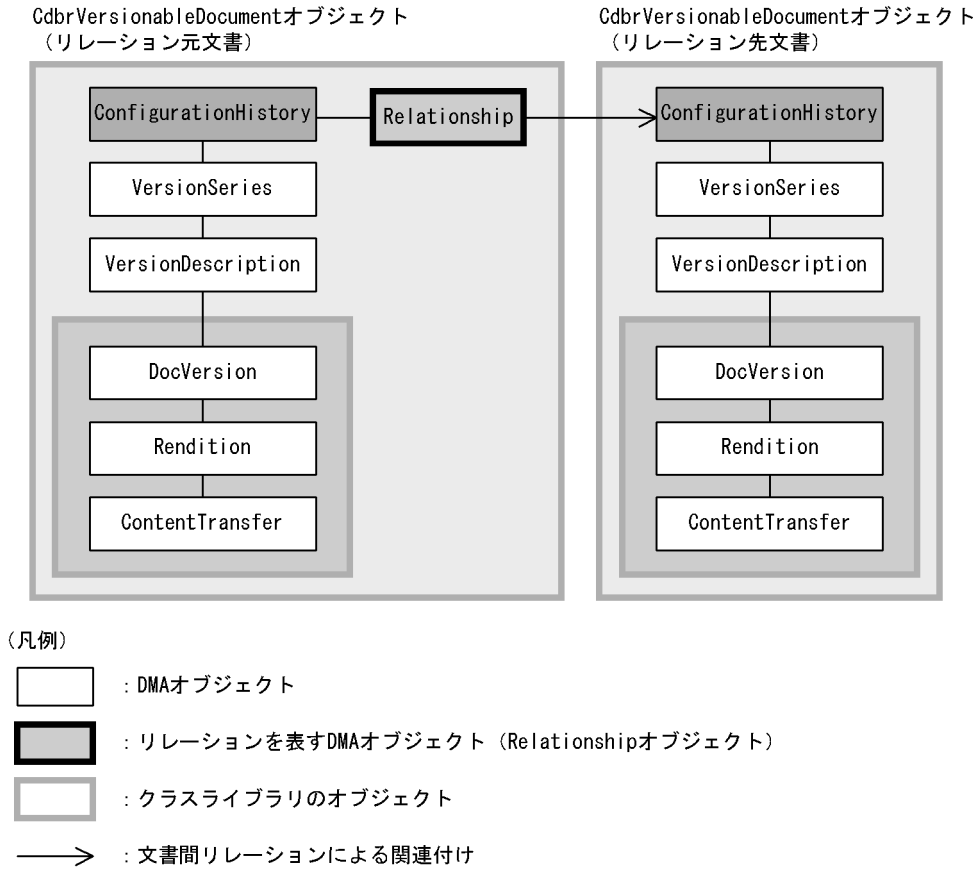
ただし，バージョン付き文書から，1 バージョンに当たるバージョンなし文書のリレーションを操作することはできません。バージョンなし文書のリレーションを操作する場合は，バージョンなし文書に接続する必要があります。

(b) バージョン付き文書間にリレーションを設定した場合の DMA オブジェクトの構成

バージョン付き文書に文書間リレーションを設定する場合，リレーションは，それぞれの文書の DMA オブジェクトの ConfigurationHistory オブジェクトと ConfigurationHistory オブジェクトを関連づけます。

バージョン付き文書間にリレーションを設定した場合の DMA オブジェクトの構成を次の図に示します。

図 3-32 バージョン付き文書間にリレーションを設定した場合の DMA オブジェクトの構成

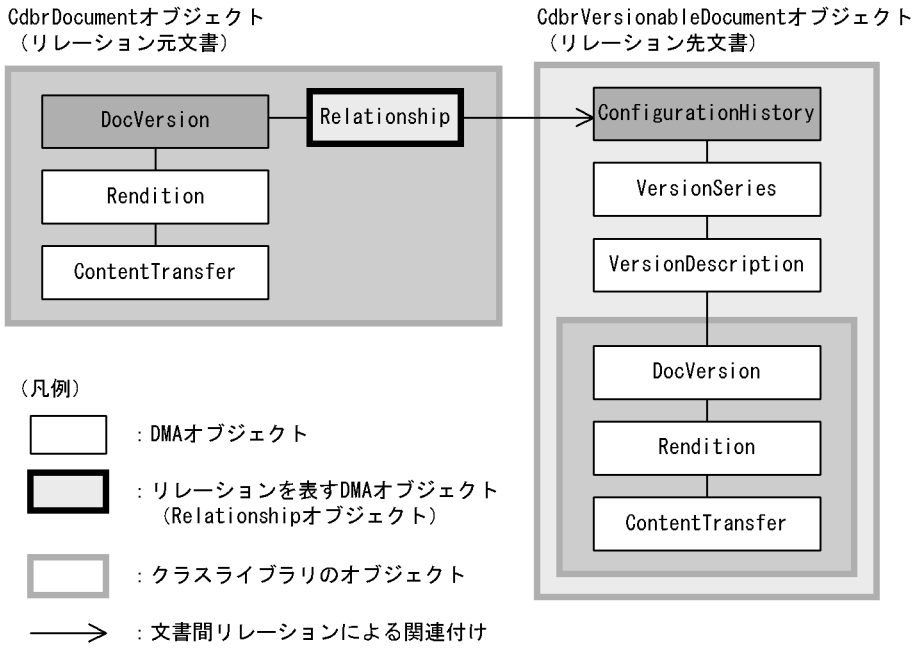


(c) バージョンなし文書とバージョン付き文書間にリレーションを設定した場合の DMA オブジェクトの構成

文書間リレーションは，バージョンなし文書とバージョン付き文書の間に設定することもできます。この場合，リレーションは DocVersion オブジェクトと ConfigurationHistory オブジェクトを関連づけます。

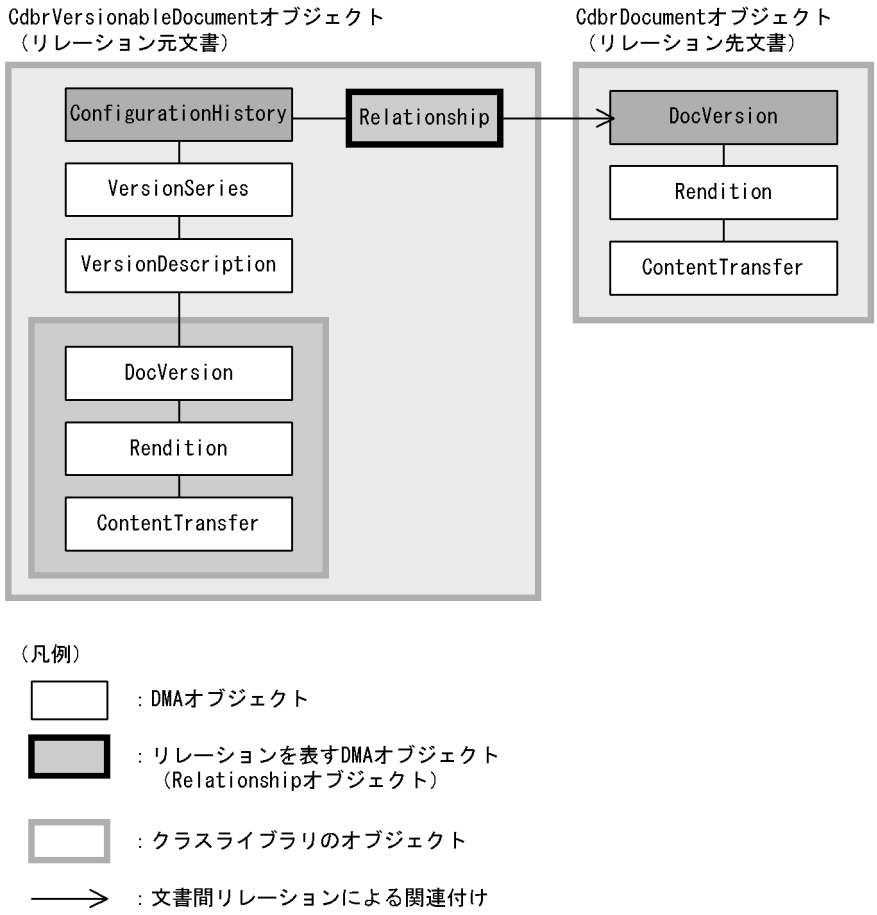
バージョンなし文書をリレーション元文書，バージョン付き文書をリレーション先文書として，文書間リレーションを設定した場合の DMA オブジェクトの構成を，次の図に示します。

図 3-33　バージョンなし文書をリレーション元文書，バージョン付き文書をリレーション先文書として文書間リレーションを設定した場合の DMA オブジェクトの構成



バージョン付き文書をリレーション元文書，バージョンなし文書をリレーション先文書として，文書間リレーションを設定した場合の DMA オブジェクトの構成を，次の図に示します。

図 3-34　バージョン付き文書をリレーション元文書，バージョンなし文書をリレーション先文書として文書間リレーションを設定した場合の DMA オブジェクトの構成



3.8.2　文書間リレーションを使用した文書管理

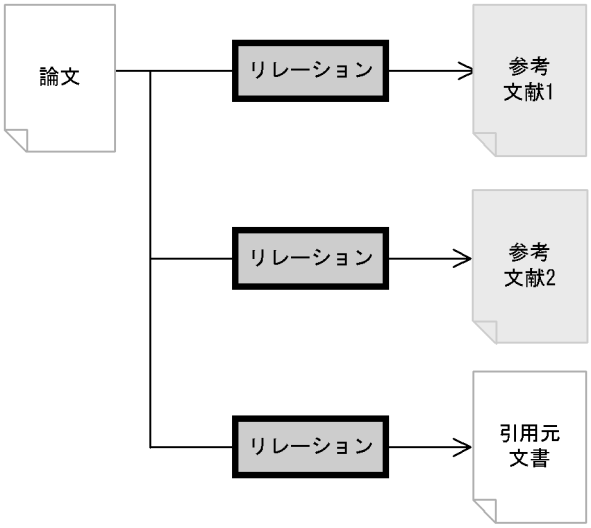
ここでは，文書間リレーションを設定して文書を管理する例について説明します。

（1）文書間リレーションの設定

文書間リレーションを設定した文書の例を次の図に示します。

ここでは，文書「論文」から文書「参考文献 1」,「参考文献 2」および「引用元文書」に対して文書間リレーションを設定します。

図 3-35 文書間リレーションを設定した文書の例



なお，設定したリレーションは，リレーション元文書が管理します。したがって，リレーションを操作する場合は，リレーション元文書から操作します。なお，リレーションを指定した操作をする場合には，リレーション識別子が使用できます。リレーション識別子は，リレーションを設定した時に取得できます。この識別子を使用して，リレーションを指定した操作や，リレーションのプロパティを取得および設定できます。

(2) リレーション先文書またはリレーション元文書の参照

文書間リレーションによって関連づけられた文書は，双方向に参照できます。例えば，図 3-35 の場合に，「論文」からリレーション先の文書である「参考文献 1」，「参考文献 2」および「引用元文書」を参照できます。また，「参考文献 1」から，リレーション元の文書として「論文」を参照できます。

リレーション先文書またはリレーション元文書を参照するための情報は，リレーション情報として取得します。リレーション情報とは，次の情報です。

- リレーション先文書またはリレーション元文書の OIID
- リレーション先文書またはリレーション元文書のオブジェクトの種類
- リレーション識別子
- リレーション先文書またはリレーション元文書のプロパティ
- リレーションのプロパティ

また，リレーション先文書がすでに削除されている場合は，リレーション識別子とリレーションのプロパティだけがリレーション情報として取得できます。

(3) リレーション，リレーション元文書およびリレーション先文書の削除

ここでは，リレーションを削除した場合に削除されるオブジェクトについて説明します。

(a) リレーションの削除

リレーションを削除する場合は，リレーション元文書に接続した状態で，削除するリレーションを指定して RemoveRelation メソッドをコールします。このとき，リレーション元文書およびリレーション先文書は削除されません。

(b) リレーション元文書の削除

リレーション元文書を削除すると，設定していたリレーションも削除されます。

また，バージョン付き文書の 1 バージョンに当たるバージョンなし文書がリレーション元文書の場合に，バージョン付き文書からそのバージョンを削除したときも，バージョンなし文書に設定されていたリレーションが削除されます。

ただし，リレーション先文書は削除されません。

(c) リレーション先文書の削除

リレーション先文書を削除した場合，リレーションおよびリレーション元文書は削除されません。したがって，リレーション元文書には，リレーション先文書がないリレーションが設定されたまま残ります。このリレーションを削除する場合は，リレーション元文書に接続して，RemoveRelation メソッドをコールしてください。

リレーション先文書がないリレーションは，リレーション元文書から GetRelationListAndLock メソッドまたは GetRelationList メソッドによって，リレーション情報の一覧を取得することで確認できます。

3.8.3 リレーションのプロパティ

ここでは，リレーションのプロパティについて説明します。

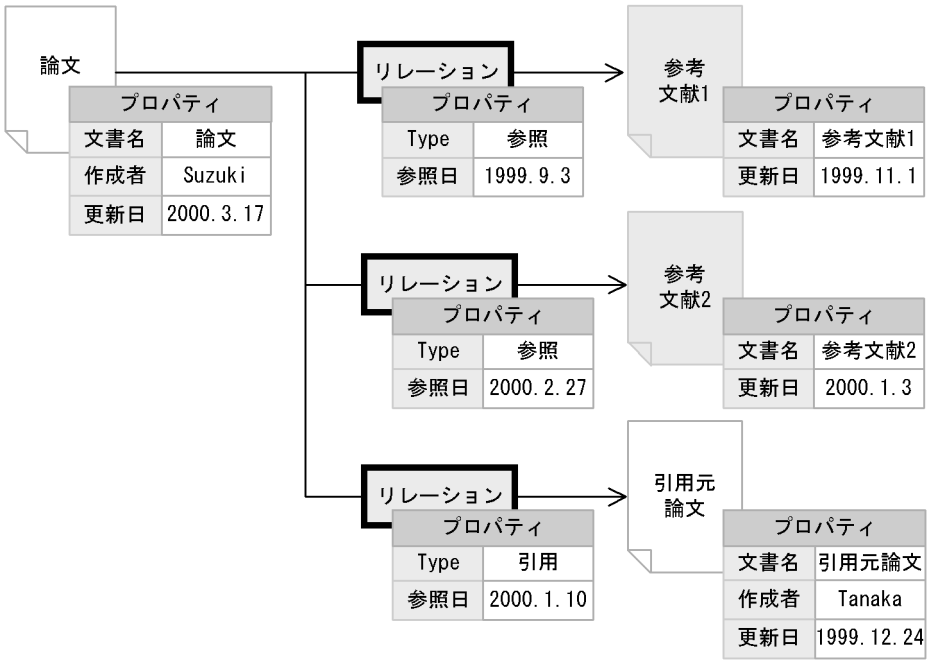
(1) リレーションに対するプロパティの設定

リレーションには，プロパティが設定できます。リレーションのプロパティは，リレーション元文書やリレーション先文書を分類したり，リレーション元文書やリレーション先文書のプロパティと組み合わせた管理に使用したりできます。

リレーションのプロパティは，リレーション元文書に接続した状態で設定します。

プロパティを設定した文書間リレーションの例を次の図に示します。

図 3-36 プロパティを設定した文書間リレーションの例



この例では，リレーションにユーザ定義プロパティとして，「Type」と「参照日」を設定しています。例えば，プロパティ「Type」によって，リレーション先文書を「参照」と「引用」に分類したりできます。

また，プロパティ「参照日」とリレーション先文書のプロパティ「更新日」を比較して，参照した以降に参照先文書が更新されたかどうかを確認したりできます。

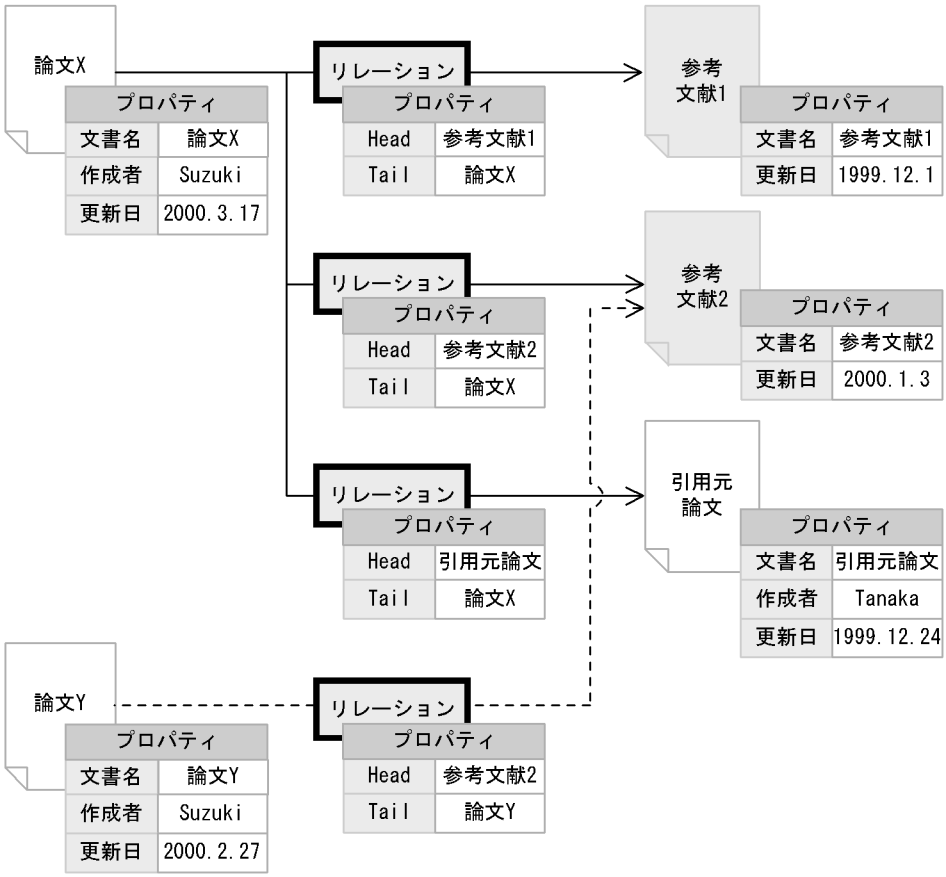
リレーションにプロパティを設定する場合は，PutRelationPropertyValues メソッドを使用します。リレーションを設定する時に同時に設定することもできます。リレーションのプロパティを取得する場合は，GetRelationListAndLock メソッドまたは GetRelationList メソッドを使用します。

(2) リレーションのプロパティを使用した検索

文書間リレーションのプロパティは検索に使用できます。検索では，ユーザが定義したプロパティのほか，リレーションを表す Relationship オブジェクトのプロパティとして設定されている，DMA が規定したプロパティ（dmaProp_ または edmProp_ で始まるプロパティ）も使用できます。

ここでは，リレーションのプロパティによってリレーション元クラスとリレーション先クラスを内部結合（INNER JOIN）した検索の方法について，次の図を例にして説明します。

図 3-37　リレーションを使用した検索に使用する文書の例



この例では，「論文 X」，「論文 Y」および「引用元論文」は dmaClass_DocVersion クラスのサブクラスを「論文クラス」を構成要素として作成されたバージョンなし文書（CdbrDocument オブジェクト）です。また，「参考文献 1」および「参考文献 2」は dmaClass_DocVersion クラスのサブクラス「参考文献クラス」を構成要素として作成されたバージョンなし文書（CdbrDocument オブジェクト）です。

この例の場合に実行できる検索の例を次に示します。

［例］作成者が『Suzuki』である論文のうち，参照先（リレーション先）の文書名が『参考文献 2』である論文を検索して，その論文の OIID と文書名を取得する。

この検索を実現するには，論文クラスと参考文献クラスを結合して実行する必要があります。このとき，それぞれの文書およびリレーションの次に示す DMA オブジェクトプロパティを使用します。

リレーションのプロパティ
- dmaProp_Head プロパティ（リレーション先オブジェクトを表すプロパティ）
- dmaProp_Tail プロパティ（リレーション元オブジェクトを表すプロパティ）

論文および参考文献のプロパティ
- dmaProp_This プロパティ（オブジェクト自身を表すプロパティ）

リレーションのプロパティである dmaProp_Head プロパティと dmaProp_Tail プロパティを使用して次のような内部結合ができます。

指定例

```
SELECT 論文クラス.dmaProp_OIID, 論文クラス.文書名
FROM   ((論文クラス INNER JOIN リレーションクラス
 ON 論文クラス.dmaProp_This = リレーションクラス.dmaProp_Tail)
INNER_JOIN 参考文献クラス
 ON リレーションクラス.dmaProp_Head = 参考文献クラス.dmaProp_This)
WHERE 論文クラス.作成者 = "Suzuki"
 AND 参考文献クラス.文書名 = "参考文献2"
```

注：

実際に edmSQL 検索で指定する場合は，「論文クラス」，「参考文献クラス」および「リレーション」には，該当するクラスのクラス識別子を指定します。また，「文書名」および「作成者」には，該当するプロパティのプロパティ識別子を指定します。

edmSQL 検索の詳細については，「4. オブジェクトの検索」を参照してください。

DMA オブジェクトのプロパティの詳細については，マニュアル「DocumentBroker Version 3 クラスライブラリ C++ リファレンス 基本機能編」を参照してください。

3.8.4 文書間リレーションに関する操作

文書間リレーションに関する操作を実行する場合に使用するメソッドと，そのメソッドの発行順序について説明します。ここでは，次の操作について説明します。

- 文書間リレーションの設定
- 文書のリレーション情報の一覧取得
- リレーション先文書およびリレーション元文書の参照
- リレーションに対するプロパティの設定
- リレーションの削除

これらの操作を実行する前に，まず，文書空間と接続してトランザクションを開始してください。

それぞれの操作に使用するメソッドの詳細については，マニュアル「DocumentBroker Version 3 クラスライブラリ C++ リファレンス 基本機能編」を参照してください。

(1) 文書間リレーションの設定

文書間リレーションは，リレーション元文書に接続した状態で設定します。

準備

リレーション元オブジェクトおよびリレーション先オブジェクトの構成要素である DMA オブジェクトの OIID を，検索などによってあらかじめ取得します。

- CdbrDocument オブジェクトに文書間リレーションを設定する場合は，DMA オブジェクトの DocVersion オブジェクトの OIID を取得します。
- CdbrVersionableDocument オブジェクトに文書間リレーションを設定する場合は，DMA オブジェ

クトの ConfigurationHistory オブジェクトの OIID を取得します。

1. **リレーション元文書にする文書に接続します。**
   接続するオブジェクトに応じて，次のどちらかのクラスの SetOIID メソッドまたは ConnectObject メソッドをコールします。
   - CdbrDocument クラス
   - CdbrVersionableDocument クラス
2. **リレーション先文書を指定して，文書間リレーションを設定します。**
   CreateRelation メソッドをコールします。

文書間リレーションを設定するコールシーケンスの例を次に示します。

文書間リレーションを設定するコールシーケンス

```
pSession->Begin();
//…
//リレーション元文書とリレーション先文書のOIIDを取得する
//…
CdbrDocument Doc1, Doc2;
//リレーション元文書に接続する
Doc1.SetOIID(pSession,pOIID);
//リレーション先文書に接続する
Doc2.SetOIID(pSession,pOIID2);
//文書間リレーションを設定する
Doc1.CreateRelation(&Doc2,NULL,ppRelId);
Doc1.ReleaseObject();
Doc2.ReleaseObject();
//処理の確定
pSession->Commit();
```

(2) 文書のリレーション情報の一覧取得

ここでは，文書のリレーション情報の一覧を取得する操作として，次の二つの操作について説明します。

- リレーション元文書からリレーション情報の一覧を取得する
  リレーション先文書とリレーションについての情報が取得できます。このとき，すでにリレーション先文書が削除されたリレーションについての情報も取得できます。
- リレーション先文書からリレーション情報の一覧を取得する
  リレーション元文書とリレーションについての情報が取得できます。

それぞれ，リレーション元文書またはリレーション先文書に接続した状態から参照します。

準備

- リレーション元文書からリレーション情報の一覧を取得する場合は，リレーション元文書の DMA オブジェクトの OIID を，検索などによってあらかじめ取得します。
- リレーション先文書からリレーション情報の一覧を取得する場合は，リレーション先文書の DMA オブジェクトの OIID を，検索などによってあらかじめ取得します。

1. **リレーション元文書からリレーション情報の一覧を取得する場合は，リレーション元文書にする文書に接続します。**
   リレーション先文書からリレーション情報の一覧を取得する場合は，リレーション先文書に接続します。
   接続するオブジェクトに応じて，次のどちらかのクラスの SetOIID メソッドまたは ConnectObject メソッドをコールします。
   - CdbrDocument クラス
   - CdbrVersionableDocument クラス
2. **リレーション情報の一覧を取得します。**

GetRelationListAndLock メソッドまたは GetRelationList メソッドをコールします。その文書をリレーション元文書とする情報の一覧を取得するか，その文書をリレーション先文書とする情報の一覧を取得するかは，引数 lRelationendStatus に設定します。

- 接続した文書をリレーション元文書とするリレーション情報の一覧を取得する場合は，引数に「DBR_RELATIONEND_HEAD」を指定します。
- 接続した文書をリレーション先文書とするリレーション情報の一覧を取得する場合は，引数に「DBR_RELATIONEND_TAIL」を指定します。

リレーション情報は，リレーション情報構造体として取得できます。リレーション情報構造体では，次の情報が取得できます。

- リレーション先文書またはリレーション元文書の OIID
- リレーション先文書またはリレーション元文書のオブジェクトの種類
- リレーション識別子
- リレーション先文書またはリレーション元文書のプロパティ
- リレーションのプロパティ

リレーション元文書からリレーション先文書の一覧を取得するコールシーケンスの例を次に示します。

リレーション元文書からリレーション先文書の一覧を取得するコールシーケンス

```
pSession->Begin();
//…
//リレーション元文書とリレーション先文書のOIIDを取得する
//…
CdbrDocument Doc;
DmaBoolean   bContinue = DMA_TRUE;
//リレーション元文書に接続する
Doc.SetOIID(pSession,pOIID);
//リレーション先文書の一覧を取得する
Doc.GetRelationListAndLock(&bContinue, DBR_CLASS_ALL, 0,
                           NULL, 0, NULL, 10,
                           ppRelationList,
                           DBR_RELATIONEND_HEAD,
                           DBR_RELATIONEND_STATUS_ALL,
                           DMA_LOCK_READ);
Doc.ReleaseObject();
//処理の確定
pSession->Commit();
```

(3) リレーション先文書およびリレーション元文書の参照

リレーション先文書およびリレーション元文書を参照する場合は，「(2) 文書のリレーション情報の一覧取得」で取得したリレーション情報構造体に設定された OIID を基に，リレーション先文書またはリレーション元文書に接続して，参照してください。

(4) リレーションに対するプロパティの設定

ここでは，リレーションにプロパティを設定する操作について説明します。

準備

- リレーション元文書の DMA オブジェクトの OIID を，検索などによって取得します。
- リレーションに設定するプロパティを，SDBR_PROP 構造体および SDBR_PROPLIST 構造体に設定します。ただし，ここで設定できるプロパティは，ユーザ定義プロパティだけです。

1. **リレーション元文書に接続します。**
   接続するオブジェクトに応じて，次のどちらかのクラスの SetOIID メソッドまたは ConnectObject メソッドをコールします。
   - CdbrDocument クラス

- CdbrVersionableDocument クラス

2. **リレーションにプロパティを設定します。**
   PutRelationPropertyValues メソッドをコールします。

リレーションにプロパティを設定するコールシーケンスの例を次に示します。

リレーションにプロパティを設定するコールシーケンス

```
pSession->Begin();
//…
//リレーション元文書のOIIDとリレーション識別子を取得する
//…
//リレーションに設定するプロパティの値を設定した
//プロパティ構造体を作成しておく
//…
CdbrDocument Doc;
//リレーション元文書に接続する
Doc.SetOIID(pSession,pOIID);
//リレーションにプロパティを設定する
Doc.PutRelationPropertyValues(pRelId, PropList);
Doc.ReleaseObject();
//処理の確定
pSession->Commit();
```

(5) リレーションの削除

ここでは，文書間リレーションを削除する操作について説明します。文書間リレーションは，次のように削除できます。

- リレーション識別子を指定して，特定のリレーションだけ削除する
- 接続した文書をリレーション元文書とする，すべてのリレーションを削除する
- リレーション先文書が削除されているリレーションをすべて削除する

リレーションを削除する場合は，リレーション元文書に接続した状態から削除します。

準備

- リレーション元文書の OIID を，検索などによってあらかじめ取得します。
- 特定のリレーションを削除する場合は，リレーション識別子を取得します。リレーション識別子は，CreateRelation メソッド，GetRelationListAndLock メソッドまたは GetRelationList メソッドのどれかで取得します。

1. **リレーション元文書に接続します。**
   接続するオブジェクトに応じて，次のどれかのクラスの SetOIID メソッドまたは ConnectObject メソッドをコールします。

2. **リレーションを削除します。**
   RemoveRelation メソッドをコールします。どのリレーションを削除するかは，引数 lRelIdCount と ppRelIds に設定します。
   - 特定のリレーションを削除する場合は，lRelIdCount に削除するリレーションの数を，ppRelIds に削除するリレーションのリレーション識別子のポインタを指定します。
   - 接続した文書をリレーション元文書とするすべてのリレーションを削除する場合は，lRelIdCount に DBR_RELATION_ALL を，ppRelIds に NULL を指定します。
   - リレーション先文書が削除されているリレーションをすべて削除する場合は，lRelIdCount に DBR_RELATION_END_NOT_EXIST を，ppRelIds に NULL を指定します。

リレーションを削除するコールシーケンスの例を示します。

リレーションを削除するコールシーケンス

```
pSession->Begin();
//…
//リレーション元文書のOIIDとリレーション識別子を取得する
//…
CdbrDocument Doc;
//リレーション元文書に接続する
Doc.SetOIID(pSession,pOIID);
//リレーション先文書の一覧を取得する
Doc.RemoveRelation(1, ppRelIds);
Doc.ReleaseObject();
//処理の確定
pSession->Commit();
```

3.9　コンテナを使用した文書管理

この節では，コンテナを使用した文書管理について説明します。コンテナを使用すると，文書をフォルダに格納するようにまとめて管理したり，複数の観点で分類して管理したりできます。また，コンテナをコンテナによって管理することで，フォルダや分類に階層構造を持たせて管理することもできます。

コンテナを使用した文書管理は，文書とコンテナを関連オブジェクトによって関連づけることによって実現します。

コンテナを使用した文書管理は，CdbrReferentialContainer クラス，CdbrConfiguratedReferentialContainer クラス，CdbrVersionTraceableContainer クラスの機能によって実現します。

なお，この節では，バージョン管理機能や構成管理機能のない，CdbrReferentialContainer クラスについて説明します。バージョン管理機能および構成管理機能を持つコンテナを表す CdbrConfiguratedReferentialContainer クラスおよび CdbrVersionTraceableContainer クラスについては，「3.10　構成管理コンテナを使用した文書の構成管理」で説明します。

3.9.1　CdbrReferentialContainer クラスの概要

ここでは，CdbrReferentialContainer クラスの概要について説明します。

(1) 機能

DocumentBroker では，コンテナを表すオブジェクトと文書を表すオブジェクトの関連づけによって，文書をまとめて管理できます。この関連づけを**コンテインメント**（Containment：包含関係）といいます。コンテインメントでは，ある二つのオブジェクトを想定して，一方を「包含するオブジェクト」，もう一方を「包含されるオブジェクト」として考えます。文書をフォルダに格納するようにしてまとめて管理する場合，包含するオブジェクトがコンテナ，包含されるオブジェクトが文書になります。

コンテナは，CdbrReferentialContainer クラスのオブジェクトとして作成します。CdbrReferentialContainer クラスは，コンテナを作成，操作したり，包含するオブジェクトとの関連づけを設定したりするための機能を提供しています。

CdbrReferentialContainer クラスを基に作成したオブジェクトは，次の機能を持ちます。

- 包含するオブジェクトとしての機能
  コンテナとして，直接型および参照型のコンテインメントを設定してオブジェクトを包含できます。
- 包含されるオブジェクトとしての機能

なお，これらの機能は，CdbrConfiguratedReferentialContainer クラスおよび CdbrVersionTraceableContainer クラスから作成したオブジェクトでも使用できます。

(2) CdbrReferentialContainer クラスを構成する DMA オブジェクト

CdbrReferentialContainer クラスを構成する DMA オブジェクトの種類を次に示します。

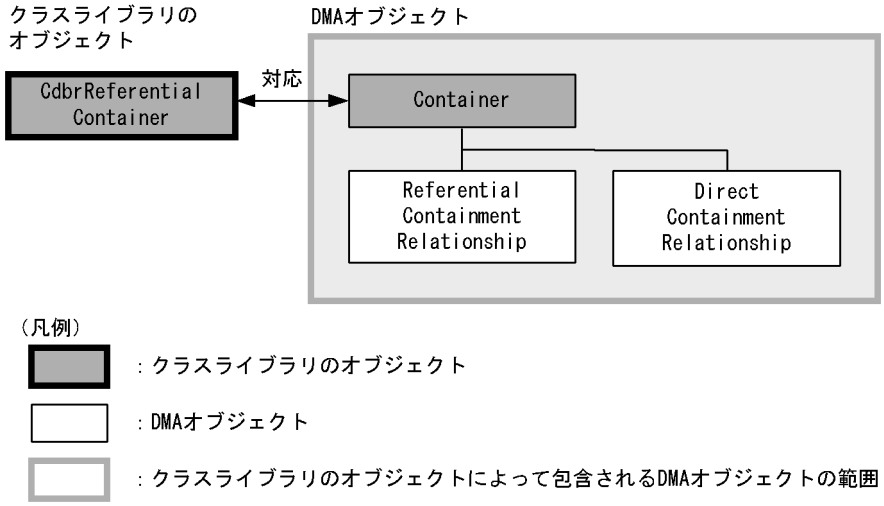
- Container オブジェクト
  コンテナを表すオブジェクトです。
- DirectContainmentRelationship オブジェクト
  直接型のコンテインメントを表すオブジェクトです。

● ReferentialContainmentRelationship オブジェクト
参照型のコンテインメントを表すオブジェクトです。

DMA オブジェクトについての詳細は，「付録 A　DMA オブジェクトの概要」を参照してください。

CdbrReferentialContainer クラスのオブジェクトと DMA オブジェクトの関係について次の図に示します。

図 3-38　CdbrReferentialContainer クラスのオブジェクトと DMA オブジェクトの関係



3.9.2　コンテナを使用した文書管理

ここでは，コンテナを使用した文書管理について説明します。

(1) 関連づけの種類

ここでは，コンテナから設定できる関連づけの種類について説明します。関連づけを表すコンテインメントには，次の 3 種類があります。

- 直接型のコンテインメント
- 参照型のコンテインメント
- 構成管理型のコンテインメント

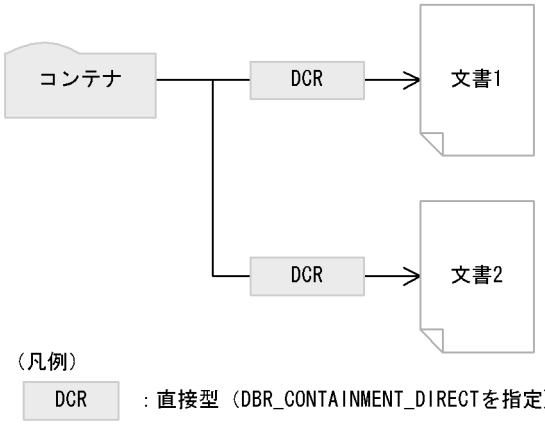
(a) 直接型のコンテインメント

直接型のコンテインメントとは，包含するオブジェクトを Parent（親），包含されるオブジェクトを Child（子）として，1：n（n は 1 以上の整数）で関連づけるコンテインメントです。Parent は複数の Child と関連づけられますが，Child は一つの Parent としか関連づけられません。このコンテインメントを使用して，文書をまとめてフォルダに格納するような管理が実現できます。フォルダには複数の文書を格納できますが，一つの文書は複数のフォルダに格納できません。なお，上位のコンテナを下位以下のコンテナの Child として関連づけることはできません。

クラスライブラリで直接型のコンテインメントを設定する場合は，CdbrReferentialContainer::Link メソッドまたは CdbrReferentialContainer::LinkAndLock メソッドの引数に DBR_CONTAINMENT_DIRECT を指定します。

次の図に直接型のコンテインメントで関連づけられたオブジェクトの例を示します。

図 3-39　直接型のコンテインメントで関連づけられたオブジェクトの例



(b) 参照型のコンテインメント

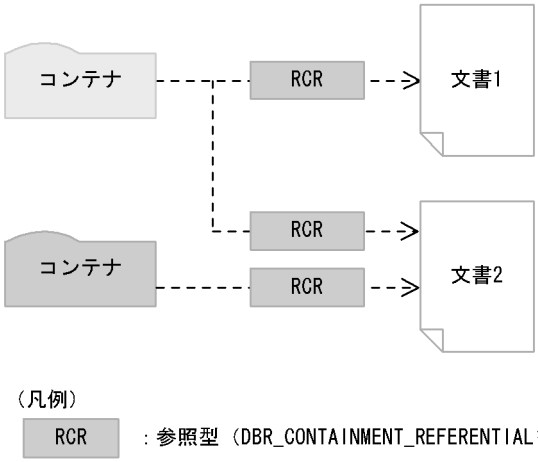
参照型のコンテインメントとは，包含するオブジェクトと包含されるオブジェクトを n：m（n，m は 1 以上の整数）で関連づけるコンテインメントです。

このコンテインメントを使用して，複数の観点で文書を分類する管理が実現できます。複数の文書に対して共通の分類から関連づけたり，一つの文書に対して複数の分類を関連づけたりできます。

クラスライブラリで参照型のコンテインメントを設定する場合は，CdbrReferentialContainer::Link メソッドまたは CdbrReferentialContainer::LinkAndLock メソッドの引数に DBR_CONTAINMENT_REFERENTIAL を指定します。

次の図に参照型のコンテインメントで関連づけられたオブジェクトの例を示します。

図 3-40　参照型のコンテインメントで関連づけられたオブジェクトの例



(c) 構成管理型のコンテインメント

構成管理型のコンテインメントとは，包含されるオブジェクトが複数のバージョンを持っている場合に，特定のバージョンに固定するか，常に最新のバージョンにつなぎ替えるかを選択して，包含されるオブジェクトを関連づけるコンテインメントです。このコンテインメントは，CdbrConfiguratedReferentialContainer クラスまたは CdbrVersionTraceableContainer クラスから作成したオブジェクトで設定できます。CdbrReferentialContainer クラスでは使用できません。

構成管理型のコンテインメントについては，「3.10　構成管理コンテナを使用した文書の構成管理」を参照

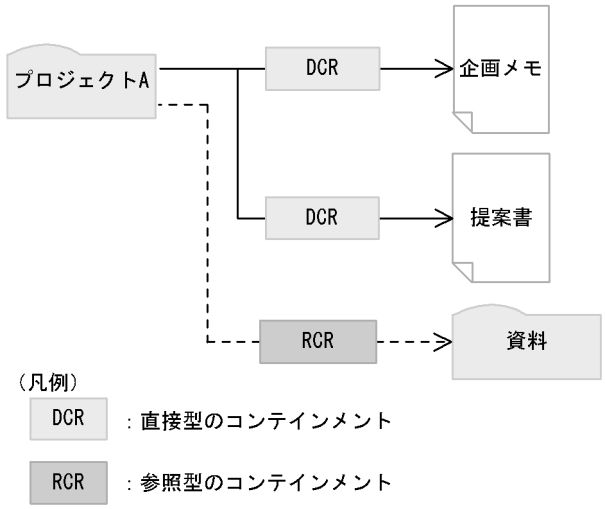
してください。

(2) 関連づけの方法

コンテナと文書またはコンテナへの関連づけは，Link メソッドまたは LinkAndLock メソッドなどをコールして設定します。これらのメソッドをコールすると，直接型または参照型のコンテインメントで関連づけられます。

コンテナ「プロジェクト A」から，文書「企画メモ」および「提案書」を直接型のコンテインメントで，コンテナ「資料」を参照型のコンテインメントで関連づけた例を示します。

図 3-41　コンテインメントで関連づけた文書とコンテナの例



なお，コンテインメントは，コンテナが管理します。コンテインメントは，コンテナに接続して操作します。

コンテインメントを指定した操作をする場合，**リンク識別子**を使用します。リンク識別子は，コンテインメントを設定した時に取得します。また，関連づけている文書やコンテナの一覧を取得する時にも取得できます。この識別子を使用して，コンテインメントを指定した操作や，コンテインメントのプロパティの操作ができます。

(3) 関連づけた文書またはコンテナの参照

コンテナからコンテインメントによって関連づけた文書またはコンテナの情報は，GetContainableListAndLock メソッドなどによって，コンテナから参照できます。また，関連づけられている文書またはコンテナから，GetContainerListAndLock メソッドなどによってコンテナを参照することもできます。例えば，図 3-41 の場合は，コンテナ「プロジェクト A」から関連づけている文書「企画メモ」「提案書」およびコンテナ「資料」を参照できます。また，文書「企画メモ」からコンテナ「プロジェクト A」を参照できます。

3.9.3　コンテナおよびコンテインメントのプロパティ

ここでは，コンテナおよびコンテインメントのプロパティと，プロパティを使用した検索について説明します。

(1) コンテナおよびコンテインメントのプロパティの管理

コンテナおよびコンテインメントは，プロパティを設定して管理できます。

コンテナには，DMA が規定したプロパティおよびクラスライブラリ固有のプロパティが設定されています。コンテインメントには，DMA が規定したプロパティが設定されています。これらのプロパティについては，「2.6.2　プロパティの種類」を参照してください。

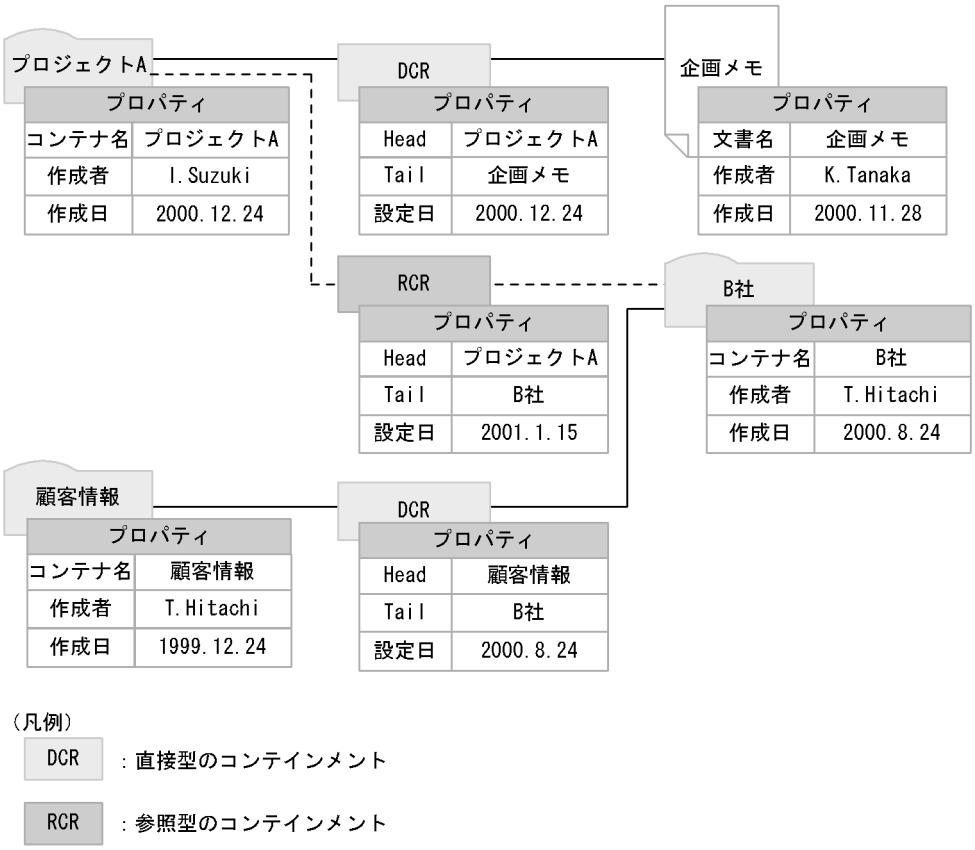
このほか，コンテナおよびコンテインメントには，ユーザ定義プロパティが設定できます。

コンテナのプロパティとしてタイトルや作成者などを設定しておくと，コンテナの検索などで使用できます。コンテナのプロパティは，構成要素である DMA オブジェクトの Container オブジェクトのプロパティとして設定します。

また，コンテインメントのプロパティは，関連づけた日付を管理したり，関連づける文書やコンテナの重要度などを設定して管理したりする場合に使用できます。例えば，重要度を設定して管理した場合は，関連づけている文書とコンテナの一覧を取得した時に，重要度に応じてソートしたりすることができるようになります。コンテインメントのプロパティは，コンテナに接続した状態で操作します。

コンテナおよびコンテインメントにプロパティを設定した例を次の図に示します。

図 3-42　コンテナおよびコンテインメントへのプロパティの設定例



この例では，コンテナにユーザ定義プロパティとして「コンテナ名」「作成者」および「作成日」を設定しています。また，コンテインメントにユーザ定義プロパティとして「設定日」を設定しています。

コンテナのプロパティは，PutPropertyValues メソッドによって設定し，GetPropertyValuesAndLock メソッドまたは GetPropertyValues メソッドによって取得します。コンテインメントのプロパティは，PutLinkPropertyValues メソッドによって設定し，GetLinkPropertyValues メソッドまたは GetLinkPropertyValuesAndLock メソッドによって取得します。また，コンテナおよびコンテインメントのプロパティは，GetContainableListAndLock メソッドや GetContainerListAndLock メソッドなどの一覧を取得するメソッドでも取得できます。詳細は，「2.6.6　プロパティの操作」を参照してください。

(2) プロパティを使用した検索

コンテナおよびコンテインメントのプロパティは検索に使用できます。検索では，ユーザが定義したプロパティのほか，DMA が規定したプロパティ（dmaProp_ または edmProp_ で始まるプロパティ）も使用できます。

例えば，図 3-42 の場合，コンテインメントに設定されている Head（dmaProp_Head）および Tail（dmaProp_Tail プロパティ）を使用して，「作成者が『T.Hitachi』であるコンテナのうち，『B 社』という名称のコンテナを関連づけているコンテナを検索する」という検索などができます。

dmaProp_Head プロパティおよび dmaProp_Tail プロパティを使用した検索の指定例については，文書間リレーションの検索例「3.8.3(2) リレーションのプロパティを使用した検索」を参照してください。また，検索の詳細については，「4. オブジェクトの検索」を参照してください。

3.9.4 コンテナの操作

コンテナを使用した文書管理をする場合に使用するメソッドと，そのメソッドの発行順序について説明します。ここでは，次の操作について説明します。

- コンテナの作成
- コンテナと文書の関連づけ
- コンテナと文書の関連づけの解除
- 階層構造をたどったコンテナの取得
- コンテナの削除

これらの操作を実行する前に，まず，文書空間と接続してトランザクションを開始してください。

それぞれの操作で使用するメソッドの詳細については，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。

(1) コンテナの作成

新しくコンテナを作成して，データベースに格納する場合，CdbrReferentialContainer::CreateObject メソッドを使用します。メソッドの発行順序については，「3.2.3(1)　文書の作成」を参照してください。ここでは，コンテナとして CdbrReferentialContainer オブジェクトを作成する場合の準備について示します。

準備

オブジェクトの構成要素になる DMA オブジェクト作成用のクラス識別子を指定した構造体（SDBR_DMAINFO 構造体）を作成しておきます。

- CdbrReferentialContainer オブジェクトを作成する場合は，SDBR_DMAINFO 構造体に，dmaClass_Container クラスまたはサブクラスの識別子を指定します。

(2) コンテナと文書の関連づけ

ここでは，コンテナと文書を関連づける場合に使用するメソッドと，そのメソッドの発行順序について説明します。

準備

オブジェクトの構成要素である DMA オブジェクトの OIID を，検索などによってあらかじめ取得しておきます。検索については「4. オブジェクトの検索」を参照してください。

- CdbrReferentialContainer オブジェクトに接続する場合は，構成要素である DMA オブジェクトの Container オブジェクトの OIID を取得しておきます。

1. **文書を関連づけるコンテナと接続します。**
   CdbrReferentialContainer::SetOIID メソッドまたは CdbrReferentialContainer::ConnectObject メソッドをコールします。
2. **コンテナと CdbrVersionableDocument オブジェクトを関連づけます。**
   CdbrReferentialContainer::Link メソッドまたは CdbrReferentialContainer::LinkAndLock メソッドをコールします。このとき，直接型のコンテインメントを設定すると，コンテナは文書に対してフォルダとして機能します。参照型のコンテインメントを設定すると，コンテナは文書に対してインデクスとして機能します。

コンテナに文書を格納するコールシーケンスの例を次に示します。

コンテナに文書を格納するコールシーケンス

```
pSession->Begin();
//…
//検索によって関連づける文書およびコンテナのOIIDを取得する
//…
CdbrReferentialContainer Container;
//検索で取得したOIIDをオブジェクトに設定する
Container.SetOIID(pSession, pOIID);
//コンテナにwriteロックを設定して，コンテナと文書を
//直接型のコンテインメントで関連づける
Container.LinkAndLock(pVrDoc, DBR_CONTAINMENT_DIRECT,
                      DMA_LOCK_WRITE);
Container.ReleaseObject();
pSession->Commit();
```

(3) コンテナと文書の関連づけの解除

コンテナと文書の関連づけを解除する場合に使用するメソッドと，そのメソッドの発行順序について説明します。なお，(2) と同様に，あらかじめ接続する文書の構成要素の OIID を取得しておきます。

1. **関連づけを解除するコンテナと接続します。**
   CdbrReferentialContainer::SetOIID メソッドまたは CdbrReferentialContainer::ConnectObject メソッドをコールします。
2. **コンテナが管理している文書の一覧を取得して，取得した文書一覧から関連づけを削除する文書の OIID を取得します。**
3. **コンテナと文書の関連づけを解除します。**
   CdbrReferentialContainer::Unlink メソッドまたは CdbrReferentialContainer::UnlinkAndLock メソッドをコールします。

コンテナと文書の関連づけを解除するコールシーケンスの例を次に示します。

コンテナと文書の関連づけを解除するコールシーケンス

```
pSession->Begin();
//…
//検索によって関連づけを解除するコンテナのOIIDを取得する
//…
CdbrReferentialContainer Container;
//検索で取得したOIIDをオブジェクトに設定する
Container.SetOIID(pSession, pOIID);
//コンテナが管理している文書の一覧を取得して,
//取得した文書一覧から関連づけを削除する文書のOIIDを取得する
Container.GetContainableListAndLock(&bContinue,
                                    DBR_CONTAINMENT_REFERENTIAL,
                                    0,NULL,10,&pObjList,
                                    DMA_LOCK_READ);
//コンテナにwriteロックを設定して,
```

```
//文書との関連づけを解除する
Container.UnlinkAndLock(pVrDoc, DMA_LOCK_WRITE);
Container.ReleaseObject();
pSession->Commit();
```

(4) 階層構造をたどったコンテナの取得

コンテナは，階層構造を付けて管理できます。ここでは，階層を付けて管理されている特定のコンテナを，階層構造をたどって取得する場合に使用するメソッドと，そのメソッドの発行順序について説明します。この説明では，ユーザ定義プロパティとして定義されているコンテナの名称を基に取得します。

準備

最上位のコンテナの OIID を，検索などによって取得しておきます。

1. **階層構造の最上位に位置するコンテナに接続します。**
   CdbrReferentialContainer::SetOIID メソッドまたは CdbrReferentialContainer::ConnectObject メソッドをコールします。
2. **コンテナのプロパティを取得します。**
   CdbrReferentialContainer::GetPropertyValues メソッドをコールします。
3. **コンテナが包含しているコンテナの一覧を取得します。**
   CdbrReferentialContainer::GetContainableListAndLock メソッドまたは CdbrReferentialContainer::GetContainableList メソッドをコールします。
4. **取得したプロパティから，該当する値のプロパティを持つコンテナを取得します。**

階層構造をたどってコンテナを検索するコールシーケンスの例を次に示します。ここでは，直接型のコンテインメントで関連づけられているコンテナを検索します。

階層構造をたどってコンテナを検索するコールシーケンス

```
pSession->Begin();
CdbrReferentialContainer Container;
//最上位のコンテナのOIIDを設定する
Container.SetOIID(pSession, RootOIID);
//最上位のコンテナのプロパティを取得する
Container.GetPropertyValues(2, PropDef, &pPropList);
//最上位のコンテナと直接型のコンテインメントで関連づけられている
//下位のコンテナの一覧を取得する
Container.GetContainableList(&bContinue,
                             DBR_CONTAINMENT_DIRECT, 2,
                             pPropDef, 10, &pObjList);
//取得したコンテナから検索する値のプロパティを持つコンテナを取得する
//…
Container.ReleaseObject();
pSession->Commit();
```

(5) コンテナの削除

コンテナの削除には，CdbrReferentialContainer::RemoveObject メソッドを使用します。これによって，DMA オブジェクトの Container オブジェクトと DirectContainmentRelationship オブジェクトおよび ReferentialContainmentRelationship オブジェクトが削除されます。このとき，削除したコンテナが包含していたオブジェクトに当たる文書やコンテナは削除されません。RemoveObject メソッドの使用方法については，「3.2.3(4) 文書の削除」を参照してください。

コンテナを削除するコールシーケンスの例を次に示します。

コンテナを削除するコールシーケンス

```
pSession->Begin();
//…
```

```
//検索によって削除するコンテナのOIIDを取得する
//…
CdbrReferentialContainer Container;
//検索で取得したOIIDをオブジェクトに設定する
Container.SetOIID(pSession, pOIID);
//コンテナを削除する
Container.RemoveObject();
Container.ReleaseObject();
pSession->Commit();
```

3.10 構成管理コンテナを使用した文書の構成管理

この節では，構成管理コンテナによってバージョンを持つ文書およびコンテナを構成管理する機能について説明します。また，構成管理コンテナ自身のバージョンと，構成管理している文書またはコンテナのバージョンとの関係についても説明します。構成管理コンテナは，CdbrVersionTraceableContainer クラスおよび CdbrConfiguratedReferentialContainer クラスを基に作成します。

3.10.1 構成管理コンテナを表すクラスの概要

ここでは，構成管理コンテナの概要と，このコンテナを作成するための基になる，CdbrVersionTraceableContainer クラスおよび CdbrConfiguratedReferentialContainer クラスの概要について説明します。

(1) 構成管理コンテナとは

DocumentBroker では，複数の文書またはコンテナを一つのコンテナと関連づけることによって，オブジェクトをまとめて管理できます。構成管理コンテナでは，関連づけて管理する文書またはコンテナが複数のバージョンを持つ場合に，どのバージョンと関連づけるかが設定できます。さらに，コンテナ自身をバージョン管理することで，どのバージョンのコンテナとどのバージョンの文書またはコンテナと関連づけるかを設定できます。

例えば，文書が複数のバージョンを持つ場合に，次のような管理ができます。

- その文書と関連づけている構成管理コンテナでは常に最新のバージョンの文書を管理するようにして，文書がバージョンアップされた場合は構成管理コンテナの関連づけも最新のバージョンの文書をたどって変更するように設定する。
- 一つのバージョンだけを管理するようにして，文書がバージョンアップしてもそのコンテナからは常に特定のバージョンだけをたどるように設定する。

この機能によって，複数の文書またはコンテナを管理する時に，そのバージョンの状態も特定して，一つのコンテナに関連づけた要素としてまとめて管理できます。この機能を**構成管理機能**といい，この機能を実現するためのコンテナを**構成管理コンテナ**といいます。

クラスライブラリで提供するクラスから作成できる構成管理コンテナは，CdbrVersionTraceableContainer オブジェクトと CdbrConfiguratedReferentialContainer オブジェクトです。これらのオブジェクトは，次の機能を持ちます。

● 包含するオブジェクトとしての機能

- 直接型および参照型のコンテインメントを設定してオブジェクトを包含できます。この機能は，CdbrReferentialContainer オブジェクトと同様です。
- 構成管理型のコンテインメントを設定できます。これによって，複数のバージョンを持つオブジェクトのバージョン構成を管理できます。
  なお，構成管理の対象になるのは，CdbrVersionableDocument オブジェクトおよび CdbrConfiguratedReferentialContainer オブジェクトです。

● 包含されるオブジェクトとしての機能

CdbrConfiguratedReferentialContainer オブジェクトの場合は，構成管理コンテナ自身が，コンテナおよび構成管理コンテナの包含要素として関連づけられます。

なお，構成管理コンテナには，構成管理コンテナ自身のバージョンも管理できるバージョン付き構成管理コンテナと，バージョン付き構成管理コンテナの個々のバージョンに対応するバージョンなし構成管理コンテナがあります。構成管理コンテナをバージョン管理する必要がない場合も，バージョンなし構成管理

コンテナが使用できます。

また，構成管理コンテナによって包含されるオブジェクトを，**構成要素**といいます。

構成管理コンテナでは，コンテナで管理する構成要素と，構成要素のバージョンを示すポインタを保持しています。これによって，構成要素の集まりを管理するほか，構成要素の任意のバージョンを管理できます。

(2) CdbrVersionTraceableContainer クラスの概要

CdbrVersionTraceableContainer クラスの概要について説明します。

(a) 機能

バージョン付き構成管理コンテナの個々のバージョンに対応する構成管理コンテナおよびバージョン管理をしない構成管理コンテナは，CdbrVersionTraceableContainer オブジェクトとして扱うことができます。このクラスから作成したオブジェクトを，**バージョンなし構成管理コンテナ**といいます。

検索で個々のバージョンに対応する構成管理コンテナの OIID（DMA オブジェクトの ContainerVersion オブジェクトの OIID）を取得した場合など，このクラスのオブジェクトとして接続して，操作できます。また，バージョン管理する必要がない場合は，バージョンなし構成管理コンテナとして作成することで，バージョン付き構成管理コンテナを使用した場合に比べて高速に処理できます。

(b) CdbrVersionTraceableContainer クラスを構成する DMA オブジェクト

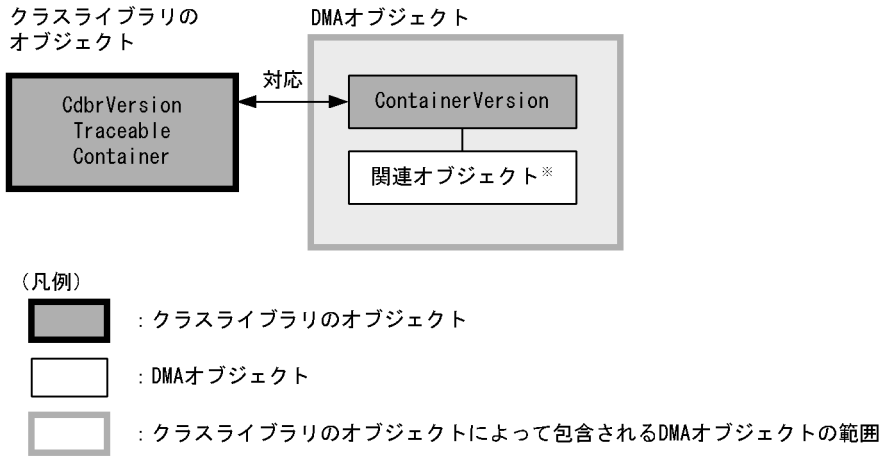
CdbrVersionTraceableContainer クラスを構成する DMA オブジェクトの種類と概要を次に示します。

- ContainerVersion オブジェクト
  構成管理コンテナを表すオブジェクトです。
- DirectContainmentRelationship オブジェクト
  直接型のコンテインメントを表すオブジェクトです。
- ReferentialContainmentRelationship オブジェクト
  参照型のコンテインメントを表すオブジェクトです。
- VersionTraceableContainmentRelationship オブジェクト
  構成管理型のコンテインメントを表すオブジェクトです。

DMA オブジェクトについての詳細は，「付録 A　DMA オブジェクトの概要」を参照してください。

CdbrVersionTraceableContainer クラスのオブジェクトと DMA オブジェクトの関係について次の図に示します。

図 3-43 CdbrVersionTraceableContainer クラスのオブジェクトと DMA オブジェクトの関係



注※ 関連オブジェクトとは，DirectContainmentRelationshipオブジェクト，ReferentialContainmentRelationshipオブジェクトまたはVersionTraceableContainmentRelationshipオブジェクトを指します。
関連オブジェクトは，一つのContainerVersionオブジェクトに対して複数存在できます。

(3) CdbrConfiguratedReferentialContainer クラスの概要

CdbrConfiguratedReferentialContainer クラスの概要について説明します。

(a) 機能

バージョン管理の対象にしたい構成管理コンテナは，CdbrConfiguratedReferentialContainer クラスを基にしたオブジェクトとして作成します。このクラスから作成したオブジェクトを，**バージョン付き構成管理コンテナ**といいます。バージョン付き構成管理コンテナは，構成管理コンテナのバージョンごとに，関連づけられている構成要素のバージョンをまとめて管理できます。

(b) CdbrConfiguratedReferentialContainer クラスを構成する DMA オブジェクト

CdbrConfiguratedReferentialContainer クラスを構成する DMA オブジェクトの種類と概要を次に示します。

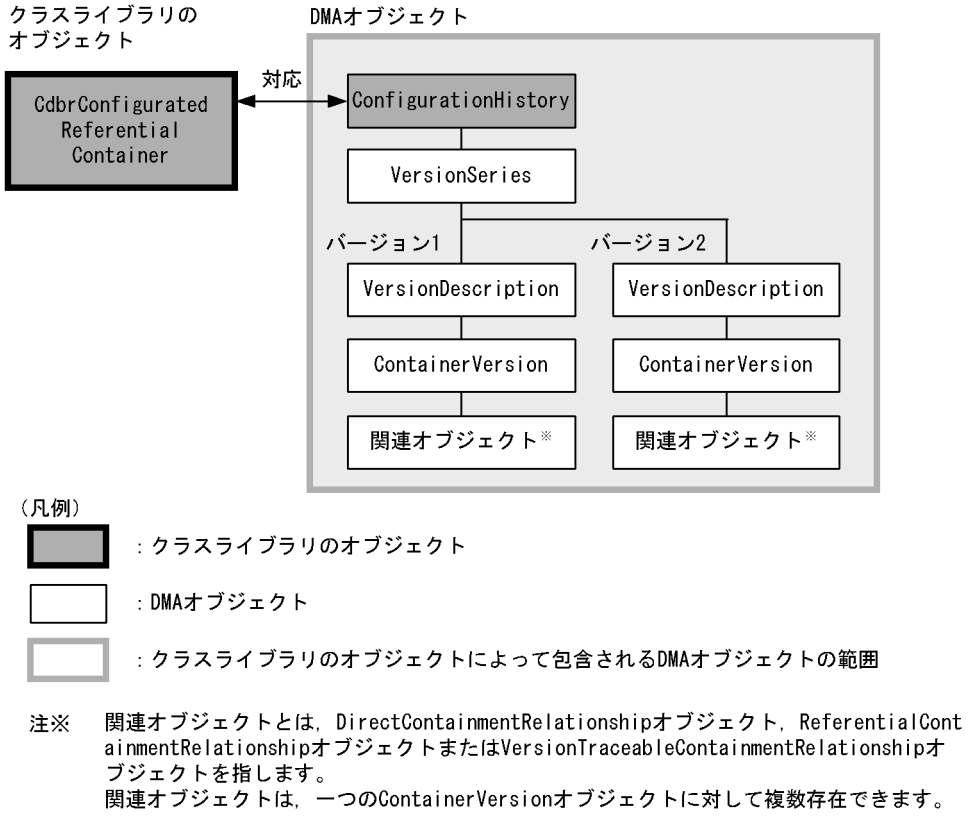
- ConfigurationHistory オブジェクト
  構成管理コンテナの一連のバージョンを統括するオブジェクトです。
- VersionSeries オブジェクト
  バージョンの構成を管理するオブジェクトです。
- VersionDescription オブジェクト
  VersionSeries オブジェクトと ContainerVersion オブジェクトを結び付けて，VersionSeries オブジェクトで管理するバージョンに対応するコンテナ（ContainerVersion オブジェクト）が何であるかを管理するオブジェクトです。
- ContainerVersion オブジェクト
  構成管理コンテナの特定のバージョンを表すオブジェクトです。
- DirectContainmentRelationship オブジェクト
  直接型のコンテインメントを表すオブジェクトです。
- ReferentialContainmentRelationship オブジェクト
  参照型のコンテインメントを表すオブジェクトです。

● VersionTraceableContainmentRelationship オブジェクト
　構成管理型のコンテインメントを表すオブジェクトです。

DMA オブジェクトについての詳細は,「付録 A　DMA オブジェクトの概要」を参照してください。

CdbrConfiguratedReferentialContainer クラスのオブジェクトと DMA オブジェクトの関係について次の図に示します。

図 3-44　CdbrConfiguratedReferentialContainer クラスのオブジェクトと DMA オブジェクトの関係



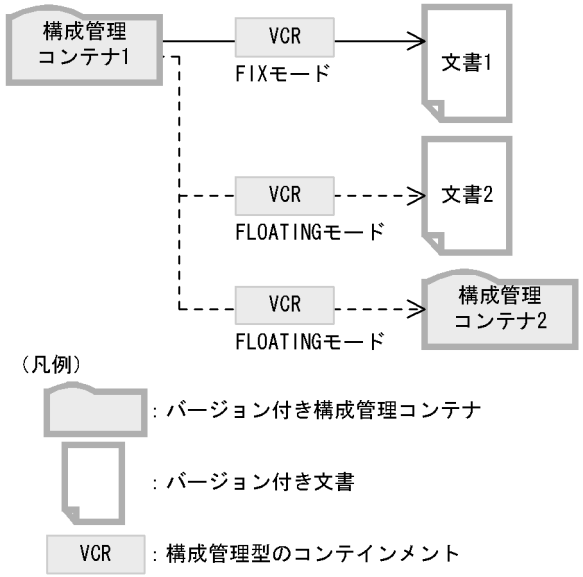
注※　関連オブジェクトとは，DirectContainmentRelationshipオブジェクト，ReferentialContainmentRelationshipオブジェクトまたはVersionTraceableContainmentRelationshipオブジェクトを指します。
関連オブジェクトは，一つのContainerVersionオブジェクトに対して複数存在できます。

3.10.2　構成管理コンテナを使用した構成管理

ここでは，構成管理コンテナを使用した構成管理について説明します。

説明で使用するバージョン付き構成管理コンテナの例を次の図に示します。

図 3-45 文書および構成管理コンテナを管理する構成管理コンテナの例



構成管理コンテナ 1 では，文書 1 に対して FIX モード，文書 2 および構成管理コンテナ 2 に対して FLOATING モードという構成管理モードを設定して関連づけています。

(1) 構成管理モードの種類

構成管理コンテナでは，構成要素である文書またはコンテナのバージョン構成を管理するために，構成管理モードを設定して，文書やコンテナを関連づけます。

(a) FLOATING モード

常に構成管理コンテナを構成要素の最新バージョンと関連づけるモードです。このモードを設定した場合，構成管理コンテナでは，構成要素がバージョンアップするたびに，コンテナを最新バージョンの構成要素につなぎ替えます。

(b) FIX モード

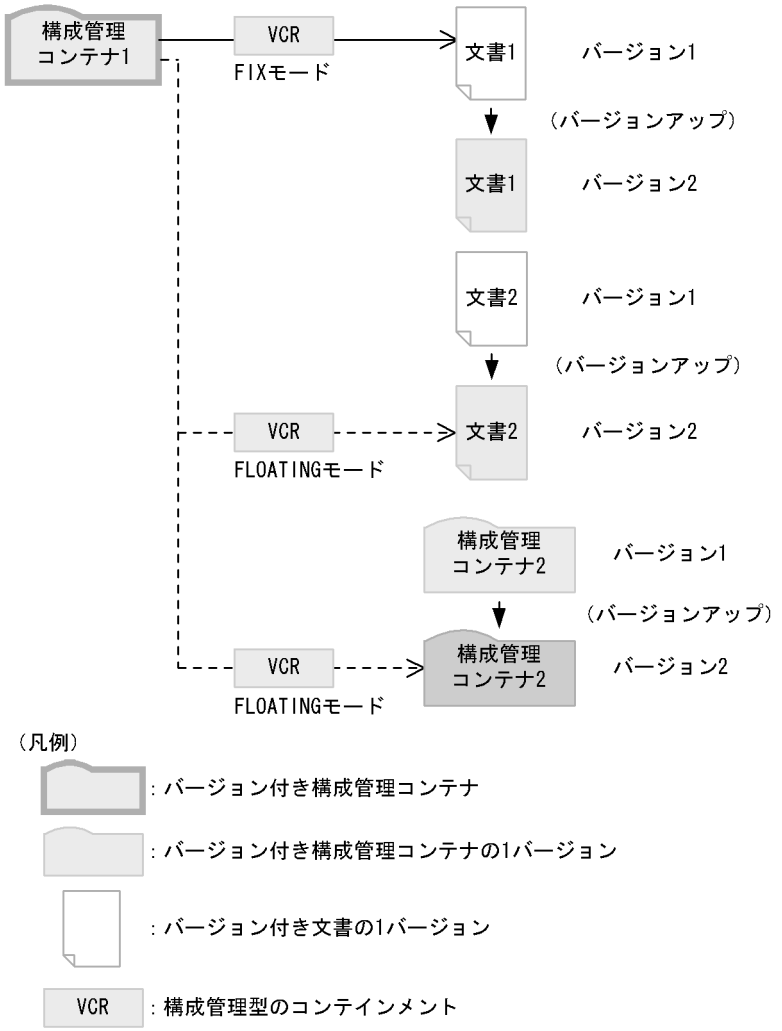
構成管理コンテナと関連づける構成要素を特定のバージョンで固定するモードです。このモードを設定した場合，構成管理コンテナが制御する構成要素は，構成要素のバージョンアップにかかわらず固定されます。

(2) 構成要素のバージョンアップ

構成要素がバージョンアップした場合，設定している構成管理モードによって，構成管理コンテナが管理する構成要素のバージョンが異なります。

ここでは，次の図に示すように，文書 1，文書 2 および構成管理コンテナ 2 をそれぞれ編集してバージョンを追加（バージョンアップ）します。

図 3-46 構成管理コンテナの構成要素の編集



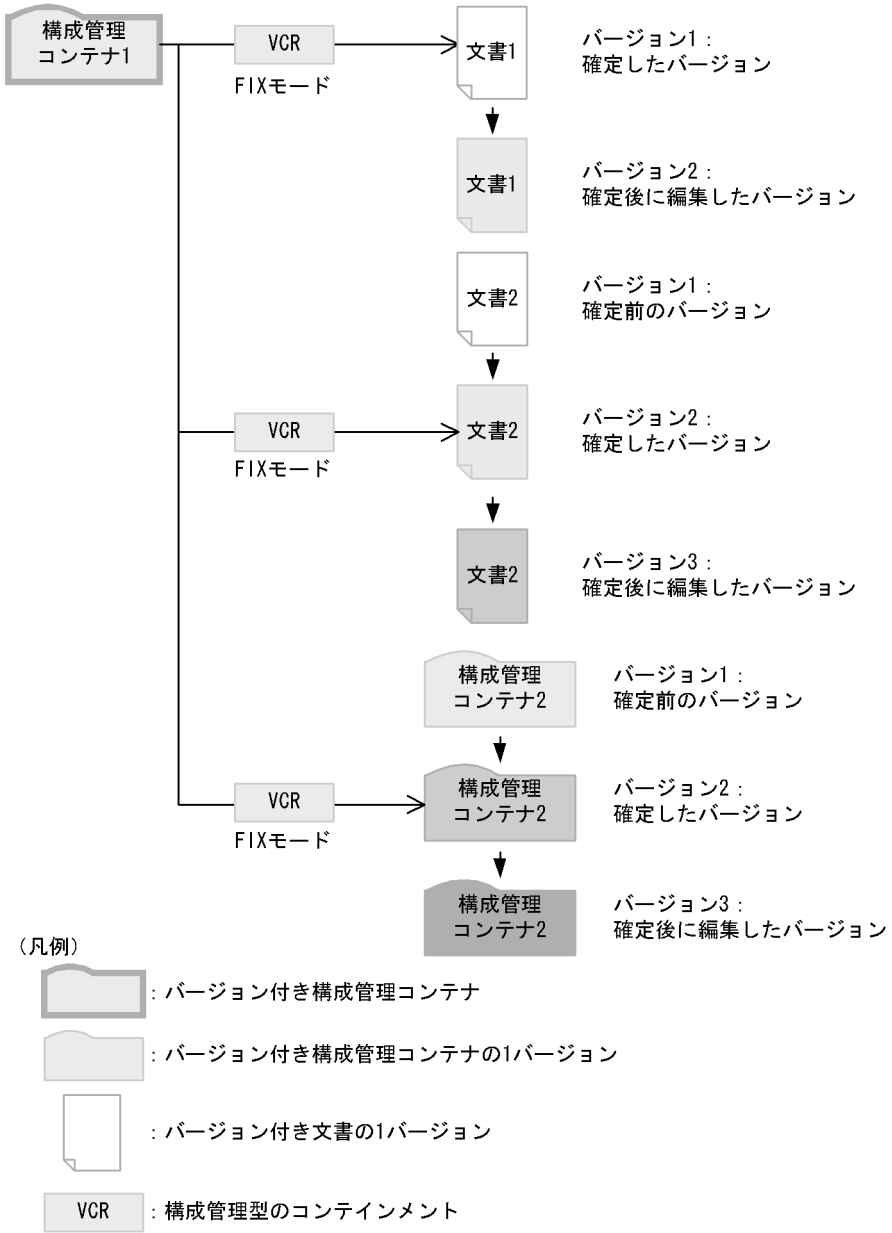
文書 1 は FIX モードで管理しているため，バージョンを追加しても構成管理コンテナ 1 が管理するのはバージョン 1 の文書 1 です。一方，文書 2 および構成管理コンテナ 2 は FLOATING モードで管理しているため，構成管理コンテナ 1 は最新のバージョンであるバージョン 2 の文書 2 および構成管理コンテナ 2 を管理するようにつなぎ替えます。

(3) 構成要素のバージョンの確定

構成要素の編集が終了した場合など，構成管理コンテナで管理する構成要素のバージョンを確定する場合，構成管理モードを FLOATING モードから FIX モードに変更します。これによって，構成管理コンテナが管理する構成要素のバージョンを確定できます。確定後に文書や構成管理コンテナにバージョンが追加されても，構成管理コンテナは確定したバージョンの構成要素から制御をつなぎ替えません。

構成要素のバージョンを確定した構成管理コンテナの例を，次の図に示します。

図 3-47　構成要素のバージョンを確定した構成管理コンテナの例



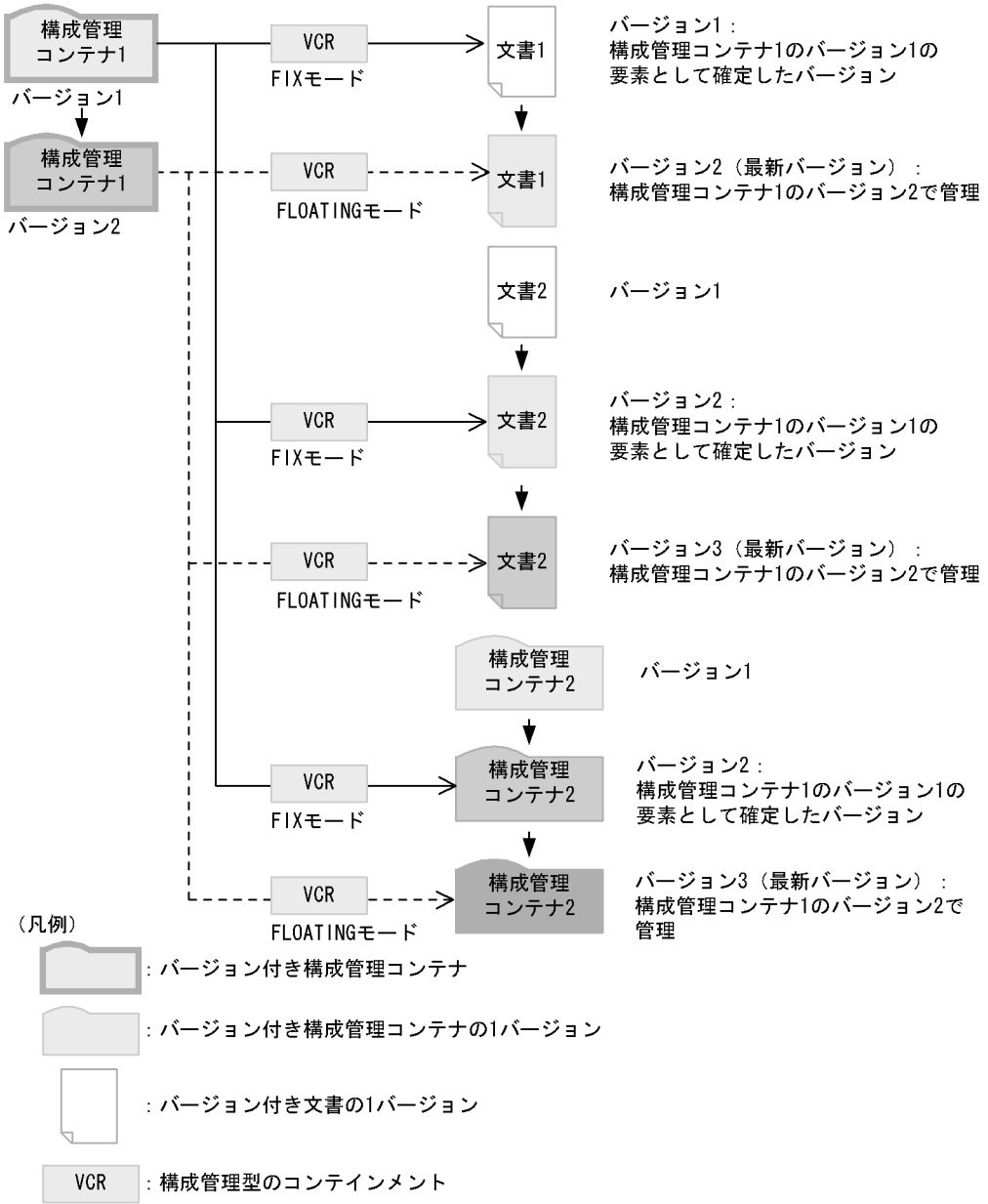
(4) バージョン付き構成管理コンテナのバージョンアップ

バージョン付き構成管理コンテナの場合，構成要素のバージョンを，構成管理コンテナのバージョンごとに管理できます。例えば，構成要素のバージョンを確定したあとでさらに構成要素を編集したい場合，バージョン付き構成管理コンテナをバージョンアップして，確定したバージョンの情報を保持する構成管理コンテナとは別に，新しく最新バージョンの文書を管理する構成管理コンテナを作成できます。

これによって，バージョン付き構成管理コンテナのバージョンごとに管理する文書またはコンテナのバージョンが決められます。

バージョン付き構成管理コンテナのバージョンアップの例を次の図に示します。

図 3-48　バージョン付き構成管理コンテナのバージョンアップの例



(5) バージョン付き構成管理コンテナのバージョンアップと構成要素および構成管理モードの関係

バージョン付き構成管理コンテナをバージョンアップした場合，新しいバージョンとして，DMA オブジェクトの ContainerVersion オブジェクトがコピーされます。つまり，最新バージョンに対応するバージョンなし構成管理コンテナがコピーされます。このとき，構成要素はコピーされません。

バージョンアップするバージョン付き構成管理コンテナに直接型のコンテインメント，参照型のコンテインメントおよび構成管理型のコンテインメントを設定している場合，関連づけを表すオブジェクトとしては参照型の関連づけを表す ReferentialContainmentRelationship オブジェクトおよび構成管理型のコンテインメントを表す VersionTraceableContainmentRelationship オブジェクトがコピーされます。これによって，バージョンアップ前の構成管理コンテナとバージョンアップ後の構成管理コンテナで同じ構成要素を管理することになります。なお，このとき直接型の関連づけを表す DirectContainmentRelationship

オブジェクトはコピーされません。したがって，直接型のコンテインメントで管理していた構成要素は，バージョンアップした構成管理コンテナでは管理されません。

また，バージョンアップの際，バージョンアップ前の構成管理コンテナでバージョンを確定しておきたい構成要素は，すべて FIX モードに確定してからバージョンアップしてください。構成管理コンテナバージョン 1 から構成管理コンテナバージョン 2 にバージョンアップした場合，構成要素ごとに設定した構成管理モードはそのまま引き継がれます。例えば，FLOATING モードのままで構成管理コンテナをバージョンアップして，構成管理コンテナバージョン 2 に関連づけられた文書をバージョンアップすると，構成管理コンテナバージョン 1 から関連づけられている文書のバージョンもつなぎ替えられ，構成管理コンテナのバージョンごとに文書のバージョンを管理することができません。

構成管理コンテナバージョン 1 で FIX モードを設定してから構成管理コンテナバージョン 2 にバージョンアップすると，構成管理コンテナバージョン 2 でも FIX モードで管理されます。構成管理コンテナバージョン 2 で構成要素の最新バージョンを管理する場合は，構成管理コンテナバージョン 2 と構成要素との関連づけを FLOATING モードに変更してください。

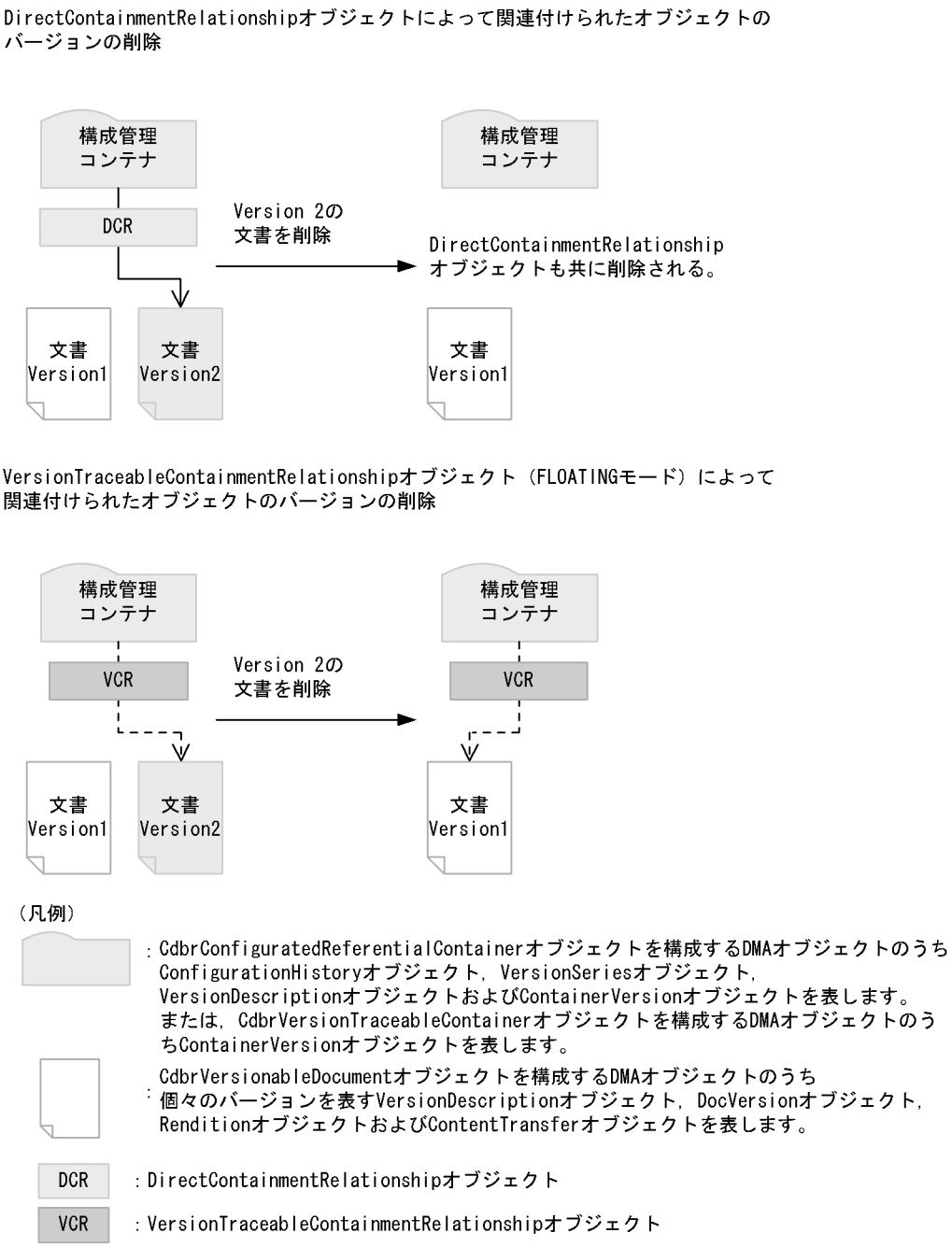
(6) 構成管理コンテナで構成管理しているオブジェクトの削除

構成管理コンテナで構成管理しているオブジェクトのバージョンを削除する場合は，コンテインメントの種類に応じて，次のオブジェクトも削除されます。

- 構成管理コンテナと直接型のコンテインメントで関連づけられているオブジェクトでは，DirectContainmentRelationship オブジェクトが削除されます。
- 構成管理コンテナと参照型のコンテインメントで関連づけられているオブジェクトでは，ReferentialContainmentRelationship オブジェクトが削除されます。
- 構成管理コンテナと構成管理型のコンテインメントで関連づけられているオブジェクトでは，VersionTraceableContainmentRelationship オブジェクトが削除されます。
  ただし，構成管理モードが FLOATING モードで関連づけられているオブジェクトのバージョンを削除する場合，VersionTraceableContainmentRelationship オブジェクトは削除されず，バージョン削除後のカレントバージョンにつなぎ替えます。

構成管理コンテナのコンテインメントの種類ごとに削除される DMA オブジェクトについて，次の図に示します。

図 3-49　構成管理コンテナのコンテインメントの種類ごとに削除される DMA オブジェクト



3.10.3　構成管理コンテナおよびコンテインメントのプロパティ

ここでは，構成管理コンテナおよびコンテインメントのプロパティと，プロパティを使用した検索について説明します。

(1) 構成管理コンテナおよびコンテインメントのプロパティの管理

構成管理コンテナおよびコンテインメントは，プロパティを設定して管理できます。

構成管理コンテナには，DMA が規定したプロパティおよびクラスライブラリ固有のプロパティが設定されています。コンテインメントには，DMA が規定したプロパティが設定されています。これらのプロパ

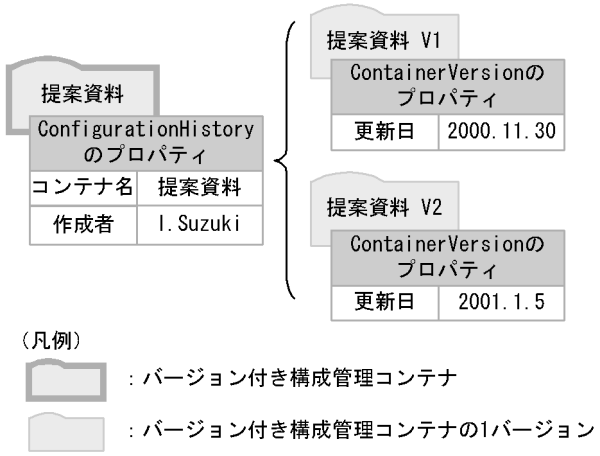
ティについては，「2.6.2　プロパティの種類」を参照してください。

このほか，構成管理コンテナおよびコンテインメントには，ユーザ定義プロパティが設定できます。

コンテナのプロパティとしてタイトルや作成者などを設定しておくと，構成管理コンテナの検索などで使用できます。バージョンなし構成管理コンテナのプロパティは，構成要素である DMA オブジェクトの ContainerVersion オブジェクトのプロパティとして設定します。バージョン付き構成管理コンテナのプロパティは，構成要素である DMA オブジェクトの ConfigurationHistory オブジェクトと ContainerVersion オブジェクトのプロパティとして設定します。バージョン共通の情報は ConfigurationHistory オブジェクトのプロパティとして，各バージョン固有の情報は ContainerVersion オブジェクトのプロパティとして設定してください。

バージョン付き構成管理コンテナのプロパティの設定例を次の図に示します。

図 3-50　バージョン付き構成管理コンテナのプロパティの設定例



この例では，バージョン共通の情報としてコンテナ名および作成者を，各バージョン固有の情報として更新日を，ユーザ定義プロパティとして設定しています。

また，コンテインメントのプロパティは，関連づけた日付を管理したり，関連づける文書やコンテナの重要度などを設定して管理したりする場合に使用できます。例えば，重要度を設定して管理した場合は，関連づけている文書とコンテナの一覧を取得した時に，重要度に応じてソートしたりすることができるようになります。コンテインメントのプロパティは，構成管理コンテナに接続した状態で操作します。コンテインメントのプロパティの設定例については，「3.9.3(1) コンテナおよびコンテインメントのプロパティの管理」を参照してください。

構成管理コンテナのプロパティは，PutPropertyValues メソッドによって設定し，GetPropertyValuesAndLock メソッドまたは GetPropertyValues メソッドによって取得します。コンテインメントのプロパティは，PutLinkPropertyValues メソッドによって設定し，GetLinkPropertyValues メソッドまたは GetLinkPropertyValuesAndLock メソッドによって取得します。また，構成管理コンテナおよびコンテインメントのプロパティは，GetContainableListAndLock メソッドや GetContainerListAndLock メソッドなどの一覧を取得するメソッドでも取得できます。詳細は，「2.6.6 プロパティの操作」を参照してください。

(2) 構成管理コンテナおよびコンテインメントのプロパティを使用した検索

構成管理コンテナおよびコンテインメントのプロパティは検索に使用できます。

構成管理コンテナの検索では，バージョン付き構成管理コンテナの検索と，バージョンなし構成管理コン

テナの検索ができます。それぞれのトップオブジェクトである DMA オブジェクトの ConfigurationHistory オブジェクトまたは ContainerVersion オブジェクトを対象に，それぞれに設定されたプロパティを指定して検索を実行します。また，構成管理コンテナから設定したコンテインメントのプロパティも検索に使用できます。

検索では，ユーザが定義したプロパティのほか，DMA が規定したプロパティ（dmaProp_ または edmProp_ で始まるプロパティ）も使用できます。例えば，コンテインメントに設定されている dmaProp_Head プロパティや dmaProp_Tail プロパティなどが使用できます。

これらのプロパティを使用した検索の指定例については，文書間リレーションの検索例「3.8.3(2) リレーションのプロパティを使用した検索」を参照してください。また，検索の詳細については，「4. オブジェクトの検索」を参照してください。

3.10.4 構成管理コンテナの操作

構成管理コンテナを操作する場合に使用するメソッドと，そのメソッドの発行順序について説明します。ここでは，次の操作について説明します。なお，バージョンアップについては，バージョン付き構成管理コンテナだけで実行できます。それ以外の操作については，バージョン付き構成管理コンテナとバージョンなし構成管理コンテナのどちらでも実行できます。

- 構成管理コンテナの作成
- 構成管理コンテナと文書の関連づけ
- 構成管理コンテナと文書の関連づけの解除
- 構成管理コンテナの構成要素のバージョンの確定
- バージョン付き構成管理コンテナのバージョンアップ
- 構成管理コンテナの削除

これらの操作を実行する前に，まず，文書空間と接続して，トランザクションを開始してください。

それぞれの操作で使用するメソッドの詳細については，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。

(1) 構成管理コンテナの作成

新しく構成管理コンテナを作成して，データベースに格納する場合，CdbrVersionTraceableContainer::CreateObject メソッドまたは CdbrConfiguratedReferentialContainer::CreateObject メソッドを使用します。メソッドの発行順序については，「3.2.3(1) 文書の作成」を参照してください。ここでは，構成管理コンテナとして CdbrVersionTraceableContainer オブジェクトおよび CdbrConfiguratedReferentialContainer オブジェクトを作成する場合の準備について示します。

準備

オブジェクトの構成要素になる DMA オブジェクト作成用のクラス識別子を指定した構造体（SDBR_DMAINFO 構造体）を作成しておきます。

- CdbrVersionTraceableContainer オブジェクトを作成する場合は，SDBR_DMAINFO 構造体に，edmClass_ContainerVersion クラスまたはサブクラスの識別子を指定します。
- CdbrConfiguratedReferentialContainer オブジェクトを作成する場合は，SDBR_DMAINFO 構造体に，dmaClass_ConfigurationHistory クラスまたはサブクラスの識別子と，edmClass_ContainerVersion クラスまたはサブクラスの識別子を指定します。ただし，構成管理の対象になるバージョン付き構成管理コンテナを作成する場合は，SDBR_DMAINFO 構造体に，次のクラス識別子を指定します。

指定するクラス識別子

- dmaClass_ConfigurationHistory クラスまたはサブクラスの識別子
- edmClass_ContainerVersion クラスまたはサブクラスの識別子

(2) 構成管理コンテナと文書の関連づけ

構成管理コンテナと文書を構成管理型のコンテインメントによって関連づける場合に使用するメソッドと，そのメソッドの発行順序について説明します。

準備

既存のオブジェクトに接続する場合，クラスライブラリのオブジェクトの構成要素である DMA オブジェクトの OIID は，検索などによってあらかじめ取得してください。検索については，「4. オブジェクトの検索」を参照してください。

取得する OIID は，接続するオブジェクトの種類によって次のようになります。

- CdbrVersionTraceableContainer オブジェクトに接続する場合は，DMA オブジェクトの ContainerVersion オブジェクトの OIID を指定します。
- CdbrConfiguratedReferentialContainer オブジェクトに接続する場合は，DMA オブジェクトの ConfigurationHistory オブジェクトの OIID を指定します。

1. **構成管理コンテナと接続します。**
   接続するオブジェクトに応じて，次のどれかのメソッドをコールします。
   - CdbrVersionTraceableContainer::SetOIID メソッド
   - CdbrVersionTraceableContainer::ConnectObject メソッド
   - CdbrConfiguratedReferentialContainer::SetOIID メソッド
   - CdbrConfiguratedReferentialContainer::ConnectObject メソッド
2. **構成管理コンテナと文書を関連づけます。**
   CdbrVersionTraceableContainer クラスまたは CdbrConfiguratedReferentialContainer クラスで定義されている，LinkVTFloatAndLock メソッド，LinkVTFloat メソッド，LinkVTFixAndLock メソッドまたは LinkVTFix メソッドのどれかをコールします。このメソッドによって，構成管理モードである，FLOATING モードまたは FIX モードが設定されます。

構成管理コンテナと文書を関連づけるコールシーケンスの例を次に示します。

構成管理コンテナと文書を関連づけるコールシーケンス

```
pSession->Begin();
//…
//検索によって格納する文書および構成管理コンテナのOIIDを取得する
//…
CdbrConfiguratedReferentialContainer VrContainer;
//検索で取得したOIIDをオブジェクトに設定する
VrContainer.SetOIID(pSession, pOIID);
//構成管理コンテナにwriteロックを設定して,
//構成管理コンテナと文書をFLOATINGモードで関連づける
VrContainer.LinkVTFloatAndLock(NULL, pVrDoc, NULL, &pLinkId,
                               DMA_LOCK_WRITE);
VrContainer.ReleaseObject();
pSession->Commit();
```

(3) 構成管理コンテナと文書の関連づけの解除

構成管理コンテナと構成要素の関連づけを解除する場合は，関連づける場合と同様に構成管理コンテナに接続してから，CdbrVersionTraceableContainer クラスまたは CdbrConfiguratedReferentialContainer クラスで定義されている UnlinkVTAndLock メソッドまたは UnlinkVT メソッドをコールします。

構成管理コンテナに格納した文書との関連づけを解除するコールシーケンスの例を次に示します。

構成管理コンテナに格納した文書との関連づけを解除するコールシーケンス

```
pSession->Begin();
//…
//検索によって関連づけを解除する文書および構成管理コンテナの
//OIIDを取得する
//…
CdbrConfiguratedReferentialContainer VrContainer;
//検索で取得したOIIDをオブジェクトに設定する
VrContainer.SetOIID(pSession, pOIID);
//構成管理コンテナが構成管理している文書の一覧を取得して,
//取得した文書一覧から関連づけを削除する文書のリンクIDを取得する
VrContainer.GetVTContaineeList(&bContinue, NULL, 0, NULL, 10,
                                &pVTObjList);
//構成管理コンテナにwriteロックを設定して,
//構成管理コンテナと文書を解除する
VrContainer.UnlinkVTAndLock(pLinkId,DMA_LOCK_WRITE);
VrContainer.ReleaseObject();
pSession->Commit();
```

(4) 構成管理コンテナの構成要素のバージョンの確定

構成管理コンテナの構成要素のバージョンの確定には，CdbrVersionTraceableContainer::SetVTFix メソッドまたは CdbrConfiguratedReferentialContainer::SetVTFix メソッドを使用します。また，バージョンの確定の解除には，CdbrVersionTraceableContainer::SetVTFloat メソッドまたは CdbrConfiguratedReferentialContainer::SetVTFloat メソッドを使用します。

構成要素のバージョンを確定するコールシーケンスの例を次に示します。

構成要素のバージョンを確定するコールシーケンス

```
pSession->Begin();
//…
//検索によって該当する構成管理コンテナのOIIDを取得する
//…
CdbrConfiguratedReferentialContainer VrContainer;
//検索で取得したOIIDをオブジェクトに設定する
VrContainer.SetOIID(pSession,pOIID);
//構成管理コンテナが構成管理している文書の一覧を取得して,
//取得した一覧から構成管理モードをFIXにする文書のリンクIDを取得する
VrContainer.GetVTContaineeList(&bContinue, NULL, 0, NULL, 10,
                                &pVTObjList);
//一覧で取得した文書の一つの構成管理モードをFIXモードに確定する
VrContainer.SetVTFix(pLinkId, DBR_VTVERSION);
VrContainer.ReleaseObject();
//処理の確定
pSession->Commit();
```

(5) バージョン付き構成管理コンテナのバージョンアップ

バージョン付き構成管理コンテナをバージョンアップするには，文書のバージョンアップと同様に，チェックアウトおよびチェックインを実行します。このとき，構成要素（バージョン付き文書またはバージョン付き構成管理コンテナ）を管理している構成管理モードを適切に変更することによって，バージョン管理コンテナごとに構成要素のバージョンの履歴管理ができるようになります。

バージョン付き構成管理コンテナのバージョンごとに構成要素の履歴を管理する場合の手順を，次の図に示します。

図 3-51 バージョンアップ構成管理コンテナのバージョンアップの手順

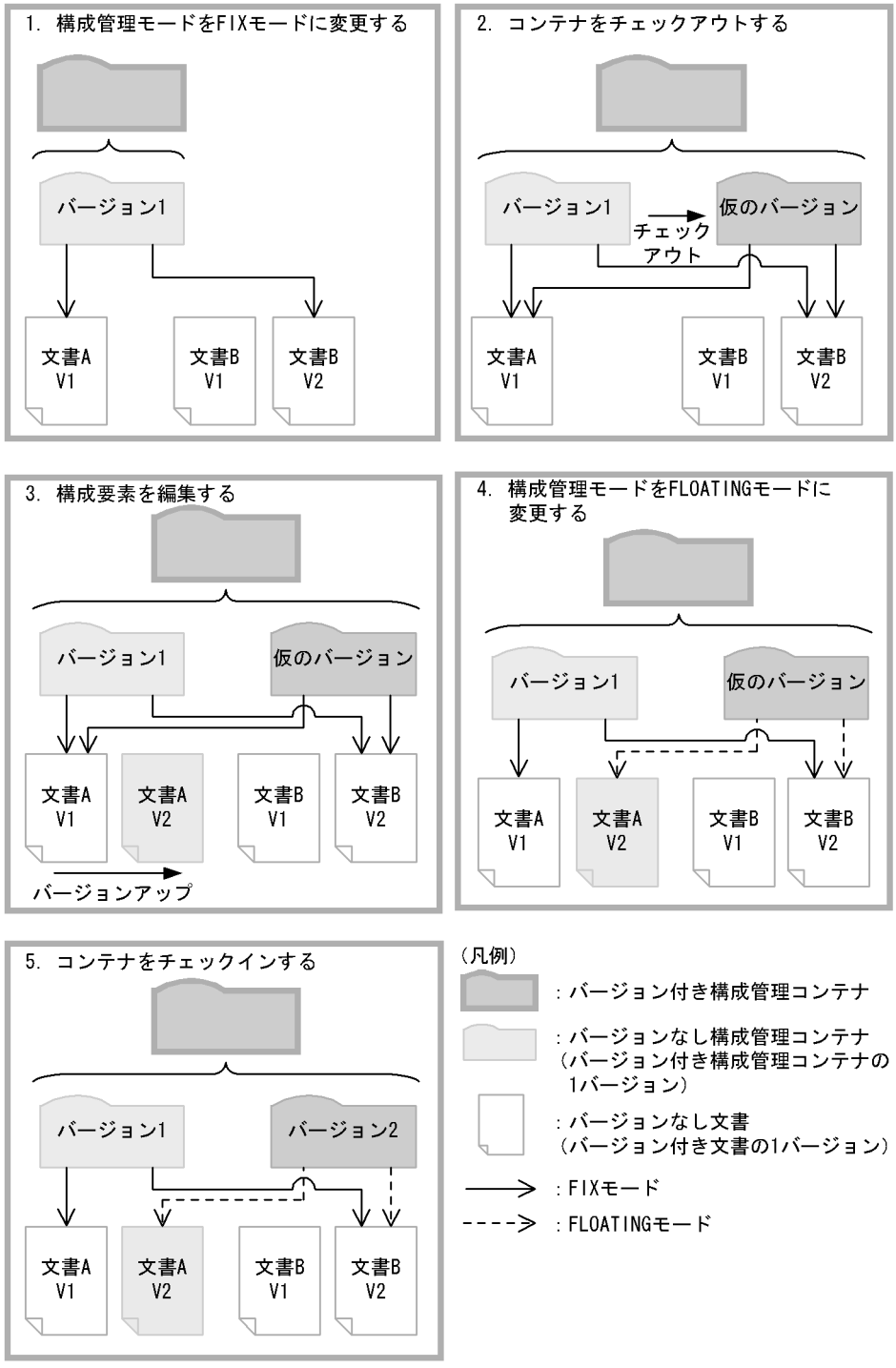


図 3-51 について説明します。

1. **バージョンアップ前の構成管理コンテナが構成要素を管理している構成管理モードを，FIX モードに変更します。**

   CdbrConfiguratedReferentialContainer::SetVTFix メソッドをコールします。

   図 3-51 は，バージョン 1 の構成管理コンテナでは，文書 A V1（バージョン 1）と，文書 B V2（バージョン 2）を管理します。

2. **バージョン付き構成管理コンテナをチェックアウトします。**
   CdbrConfiguratedReferentialContainer::VersionCheckOut メソッドをコールします。
   仮のバージョンの構成管理コンテナが作成されます。仮のバージョンの構成管理コンテナが構成要素を管理する構成管理モードは，チェックアウト前の状態と同じ，FIX モードになります。

3. **構成要素を編集します。**
   図 3-51 では，文書 A をバージョンアップしています。ただし，構成管理モードが FIX モードなので，この時点で仮のバージョンのコンテナが構成管理するのは，文書 A V1 のままです。

4. **仮のバージョンの構成管理コンテナで構成要素を管理している構成管理モードを，すべて** FLOATING **モードにします。**
   CdbrConfiguratedReferentialContainer::SetVTFloat メソッドをコールします。
   この時点で，仮のバージョンの構成管理コンテナが管理する構成要素が，文書 A V1 から文書 A V2 に変わります。

5. **バージョン付き構成管理コンテナをチェックインします。**
   CdbrConfiguratedReferentialContainer::VersionCheckIn メソッドをコールします。
   仮のバージョンの構成管理コンテナが，バージョン 2 の構成管理コンテナとして確定されます。
   これによって，バージョン 1 の構成管理コンテナで管理する文書 A は文書 A V1，バージョン 2 の構成管理コンテナで管理する文書 A は文書 A V2 となり，バージョン付き構成管理コンテナのバージョンごとの文書の履歴管理が実現できます。

注意

バージョンアップ後の構成管理コンテナの構成要素を編集する場合，バージョンアップ前の構成管理コンテナの構成管理モードを FIX モードに変更したあとで，構成要素を編集してください。構成管理モードが FLOATING モードの状態で構成要素を編集した場合，バージョンアップ前の構成管理コンテナが管理する構成要素も編集後の構成要素に切り替わってしまうため，構成管理コンテナのバージョンによる構成要素の履歴管理ができません。

それぞれの構成要素のバージョンアップについては，「3.3.3　バージョン管理機能の操作」を参照してください。

(6) 構成管理コンテナの削除

バージョン付き構成管理コンテナである CdbrConfiguratedReferentialContainer オブジェクトを削除する場合は，CdbrConfiguratedReferentialContainer::RemoveObject メソッドを使用します。これによって，CdbrConfiguratedReferentialContainer オブジェクトを構成していたすべての DMA オブジェクトが削除されます。

また，特定のバージョンに対応するバージョンなし構成管理コンテナである CdbrVersionTraceableContainer オブジェクトは，次のどちらかの方法で削除します。

- バージョン付き構成管理コンテナである CdbrConfiguratedReferentialContainer オブジェクトに接続して，CdbrVersionable クラスで定義されている DeleteVersion メソッドをコールする。
- バージョン付き構成管理コンテナの個々のバージョンに対応する CdbrVersionTraceableContainer オブジェクトに接続して，CdbrVersionTraceableContainer::RemoveObject メソッドをコールする。

どちらのメソッドを使用した場合も，該当するバージョンのバージョンなし構成管理コンテナを構成していた DMA オブジェクトの VersionDescription オブジェクト，ContainerVersion オブジェクト，VersionTraceableContainmentRelationship オブジェクト，DirectContainmentRelationship オブジェクトおよび ReferentialContainmentRelationship オブジェクトが削除されます。

なお，バージョン付き構成管理コンテナおよび特定のバージョンに対応するバージョンなし構成管理コンテナを削除しても，包含していた文書やコンテナは削除されません。

3.11 XML 文書の管理

この節では，DocumentBroker で XML 文書を管理する場合の管理方法について説明します。

3.11.1 XML 文書管理機能の概要

ここでは，XML 文書管理機能の概要について説明します。

(1) DocumentBroker で提供する XML 文書管理機能

XML は，W3C によって標準化が進められている，文書の構造を定義するための言語です。XML では，ユーザが設定した独自のタグによって，論理構造を持つ文書を記述できます。このため，Web 上のデータ交換フォーマットなどとして，多くの業務で適用されています。

DocumentBroker では，XML 形式で記述されたファイルをバージョンなし文書およびバージョン付き文書のコンテントとして管理できます。

なお，このマニュアルでは，XML で記述されたファイルを **XML ファイル**と呼びます。また，XML ファイルをコンテントとして DocumentBroker に登録した文書を，**XML 文書**と呼びます。

DocumentBroker では，XML 文書を管理するために次のような機能を提供しています。

- **XML プロパティマッピング機能**
- **XML インデクスデータ作成機能**

XML プロパティマッピング機能を使用する場合は，前提プログラムとして HiRDB Adapter for XML が必要です。また，XML インデクスデータ作成機能を使用する場合は，前提プログラムとして HiRDB Adapter for XML と，Preprocessing Library for Text Search が必要です。また，全文検索を実行する場合は，サーバ側に HiRDB Text Search Plug-in が必要です。

(2) XML プロパティマッピング機能

ここでは，XML プロパティマッピング機能について説明します。

(a) XML プロパティマッピング機能とは

XML プロパティマッピング機能とは，XML ファイル中の，タグ間の文字列やタグの属性値を抽出して，XML 文書のプロパティに設定するための情報を作成する機能です。この情報を XML 文書のプロパティにマッピングすることによって，例えば，次のようなことができるようになります。

- プロパティの一覧として情報を取得する
- 属性検索の対象にする
- 検索結果をソートするときのキーにする

XML プロパティマッピング機能を使用する XML ファイルの例を，次の図に示します。

図 3-52　XML プロパティマッピング機能を使用する XML ファイルの例

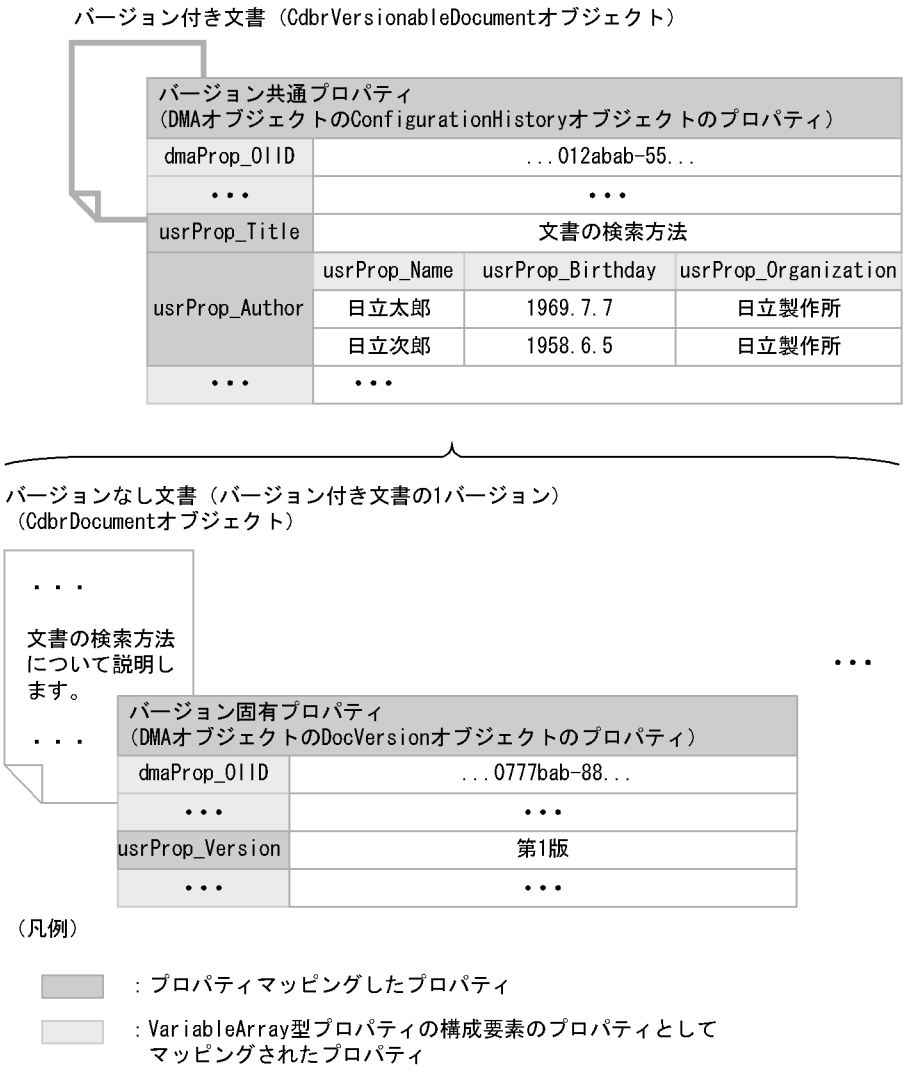
```
<?xml version="1.0" encoding="Shift_JIS"?>

<doc>

  <prop>

    <title>文書の検索方法</title>

    <author name="日立太郎">
      <birthday>1969.7.7</birthday>
      <organization>日立製作所</organization>
    </author>

    <author name="日立次郎">
      <birthday>1958.6.5</birthday>
      <organization>日立製作所</organization>
    </author>

    <version>第１版</version>

  </prop>

  <content>
  文書の検索方法について説明します。
  ..........
  </content>

</doc>
```

<prop> タグで囲まれた内容には，タイトルや著者，著者に関する情報，バージョン情報などが，それぞれのタグ内の文字列または属性値として記述されています。

ここでは，XML ファイルをバージョン付き文書のコンテントとして登録する場合について説明します。タイトルと著者の情報はバージョン付き文書のプロパティ（DMA オブジェクトの ConfigurationHistory オブジェクトのプロパティ）として，バージョン情報はバージョンなし文書（バージョン付き文書の 1 バージョン）のプロパティ（DMA オブジェクトの DocVersion オブジェクトのプロパティ）としてマッピングすることにします。

図 3-52 の XML ファイルに含まれる情報を XML 文書のプロパティにマッピングした例を，次の図に示します。

図 3-53　XML プロパティマッピング機能によって XML 文書のプロパティにマッピングした例



この例では，<title> タグ間の情報は，バージョン付き文書の usrProp_Title プロパティにマッピングしています。<author> タグ間の情報は，複数の情報を表すタグを含むので，VariableArray 型プロパティである usrProp_Author プロパティにマッピングしています。<author> タグの属性である name の値，<author> タグ間に含まれる <birthday> タグおよび <organization> タグ間の情報は，VariableArray 型プロパティの構成要素を表す usrProp_Name プロパティ，usrProp_Birthday プロパティおよび usrProp_Organization プロパティにそれぞれマッピングしています。

(b) 機能

XML プロパティマッピング機能は，XML ファイルに対して構文解析を実行し，タグ間の文字列や属性値を抽出して，プロパティにマッピングするための機能です。抽出した情報は，文書を作成する時に指定する DMA オブジェクト生成用の構造体（SDBR_DMAINFO 構造体）として出力されます。
SDBR_DMAINFO 構造体は，文書の作成時に使用する，DMA オブジェクトのクラス識別子とプロパティの初期値をメンバとして持つ構造体です。SDBR_DMAINFO 構造体の詳細については，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。

作成した SDBR_DMAINFO 構造体を指定して文書を作成することで，XML ファイルのコンテントの情報

をプロパティにマッピングしたXML文書が作成できます。

なお，この機能を使用する場合は，登録するXMLファイルの論理構造を明確にして，どのタグをどのプロパティにマッピングするかの対応を定義しておく必要があります。定義に必要なファイルには，ユーザが作成するファイルと，DocumentBrokerおよびHiRDB Adapter for XMLによって作成されるファイルがあります。

定義ファイルの詳細については，マニュアル「DocumentBroker Version 3　システム導入・運用ガイド」を参照してください。

(3) XMLインデクスデータ作成機能

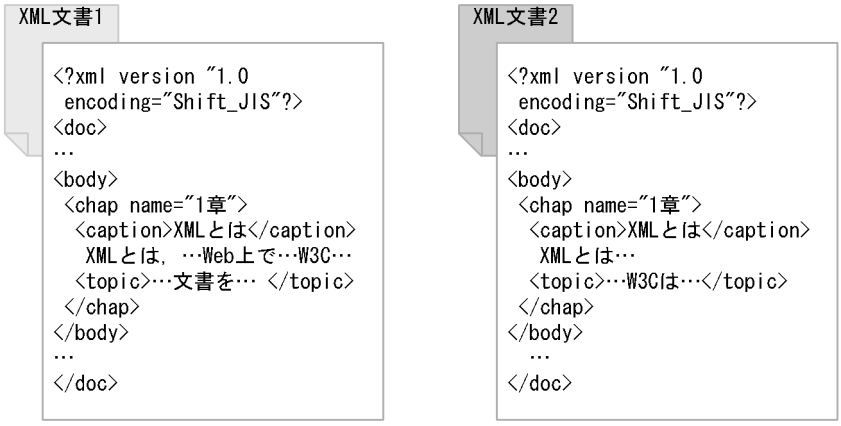
ここでは，XMLインデクスデータ作成機能について説明します。

(a) XMLインデクスデータ作成機能とは

論理構造を持つ文書の構造を指定して実行する全文検索を，**構造指定検索**といいます。XMLインデクスデータ作成機能を使用して全文検索インデクスを作成したXML文書は，構造指定検索の対象になります。

構造指定検索について，次の図に示したXML文書1およびXML文書2の例で説明します。

図3-54　構造指定検索で使用するXML文書の例



構造指定検索では，これらの文書に対して，「構造 <chap> の下の構造 <topic> の中に『W3C』が含まれる文書を検索する」といった検索ができます。この例の場合は，XML文書2が検索結果として取得できます。

検索についての詳細は，「4.　オブジェクトの検索」を参照してください。

(b) 機能

XMLインデクスデータ作成機能は，XMLファイルの構文解析を実行して，構造指定検索に必要なインデクスデータを作成するための機能です。また，構造指定検索の対象にならない，プレーンテキスト形式の全文検索インデクスデータも作成できます。作成したインデクスデータは，**インデクスデータ出力ファイル**に出力されます。このファイルを指定してXML文書を作成することによって，構造指定検索またはプレーンテキストを対象にした全文検索に必要な全文検索インデクスを登録できます。

また，インデクスデータを作成する時には，XMLファイルの内容から不要なタグを表す情報を削除したり，不要なタグおよびそのタグに囲まれた文字列全体を削除したりできます。この場合，削除するタグについての定義は，フィルタリング定義ファイルとして作成しておく必要があります。フィルタリング定義ファイルについては，マニュアル「DocumentBroker Version 3　システム導入・運用ガイド」を参照して

ください。

なお，XML インデクスデータ作成機能は，次の文字コードで記述された XML ファイルを対象に実行できます。

HiRDB Adapter for XML の場合

- Shift-JIS
- EUC
- UTF-8
- UTF-16

ただし，DocumentBroker の文書の全文検索インデクスは，次の文字コードで登録されます。

AIX または Windows の場合

Shift-JIS

このため，AIX または Windows の場合は，XML 文書に対して構造指定検索を実行するとき，検索条件をすべて Shift-JIS で入力してください。

また，構造指定検索の対象になる XML 文書は，次の条件に従って作成する必要があります。

- CdbrDocument オブジェクトまたは CdbrVersionableDocument オブジェクトの構成要素として，全文検索を実行するためのプロパティを追加した dmaClass_DocVersion クラスまたはそのサブクラス（全文検索機能付き文書クラス）を指定してください。
- 全文検索機能付き文書クラスに追加するプロパティのうち，全文検索インデクス用プロパティは，次のプロパティのどれかにしてください。
  - edmProp_StIndex プロパティ（構造指定検索用全文検索インデクス）
  - edmProp_ConceptStIndex プロパティ（構造指定検索用概念検索インデクス）
  - edmProp_Content プロパティ（Version 1 互換用全文検索インデクス）

全文検索機能付き文書クラスおよび全文検索インデクス用プロパティについての詳細は，「4.4　全文検索の対象になる文書の作成」を参照してください。また，登録した文書に対して実行できる全文検索の詳細については，「4.2.2　全文検索」を参照してください。

（4） XML 文書管理機能を実現するクラスと DMA オブジェクト

ここでは，XML 文書管理機能を実現するための，クラスライブラリのクラスと DMA オブジェクトについて説明します。

（a） XML 文書管理機能を実現するためのクラス

XML 文書管理機能は，次の二つのクラスの機能を使用して実現されます。

- CdbrXmlTranslatorFactory クラス
  CdbrXmlTranslator オブジェクトを作成するためのクラスです。このクラスは，XML 文書管理機能を実行するための操作環境を管理します。
- CdbrXmlTranslator クラス
  実際に XML プロパティマッピング機能および XML インデクスデータ作成機能を実行するためのクラスです。
  このクラスの機能は，CdbrXmlTranslator オブジェクトが作成された状態で使用できます。

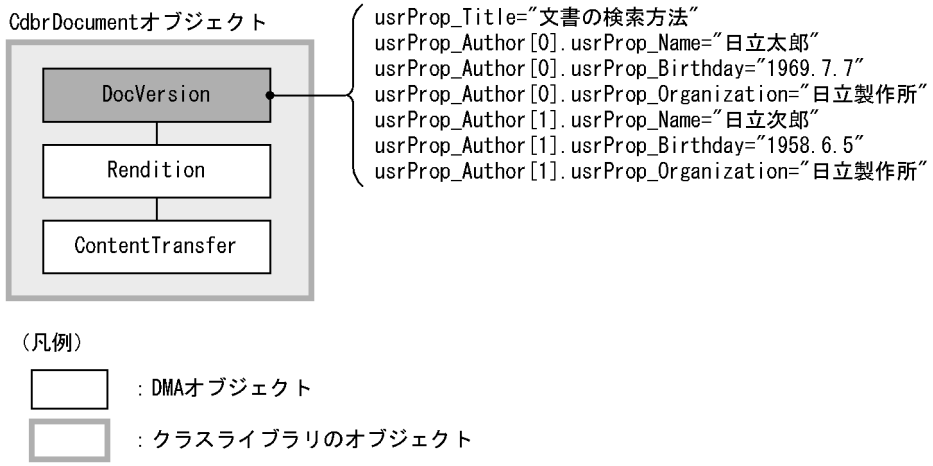
(b) XML 文書を表す DMA オブジェクト

XML 文書は，CdbrDocument オブジェクトまたは CdbrVersionableDocument オブジェクトとして作成，管理します。これらのオブジェクトを構成する DMA オブジェクトについては，「3.2.1　文書を表すクラスの概要」を参照してください。

ここでは，それぞれのオブジェクトに対して，XML 文書管理機能によって設定される情報について説明します。

● CdbrDocument オブジェクトの場合，XML プロパティマッピング機能によって抽出した情報は，DMA オブジェクトの DocVersion オブジェクトのプロパティにマッピングできます。
CdbrDocument オブジェクトに対する XML プロパティマッピングの例を，次の図に示します。

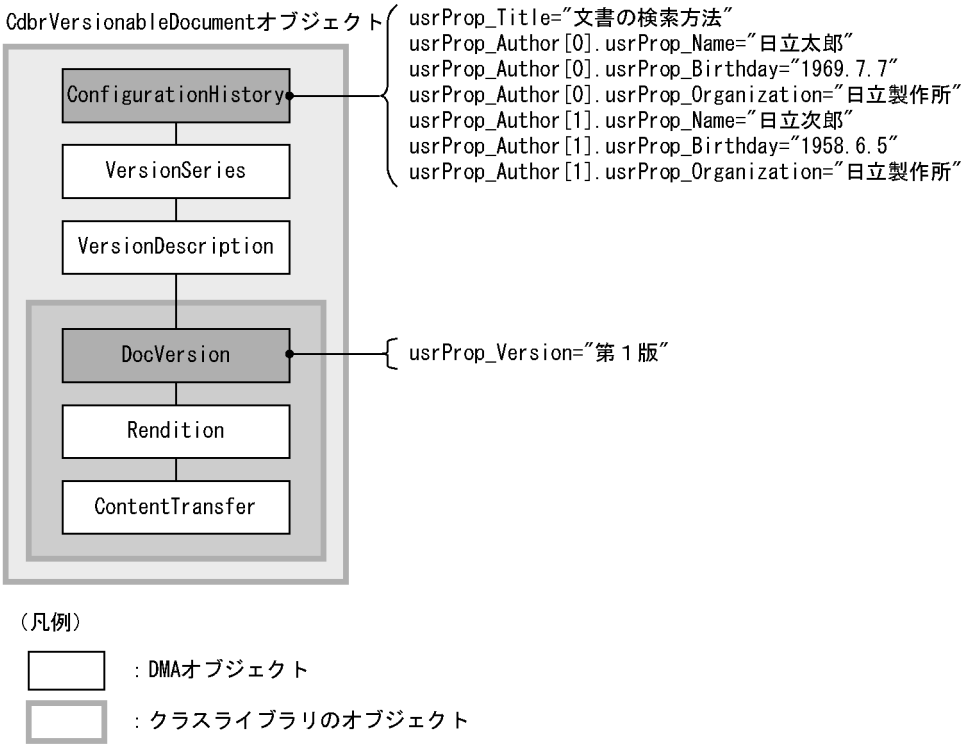
図 3-55　CdbrDocument オブジェクトに対する XML プロパティマッピングの例



また，XML インデクスデータ作成機能で作成したインデクスデータを基に，DocVersion オブジェクトの全文検索インデクス用プロパティが設定できます。

● CdbrVersionableDocument オブジェクトの場合，XML プロパティマッピング機能によって抽出した情報は，DMA オブジェクトの ConfigurationHistory オブジェクトと DocVersion オブジェクトのプロパティにマッピングできます。
例えば，タイトルや著者などバージョン共通の情報を表すタグの情報は，ConfigurationHistory オブジェクトのプロパティにマッピングできます。バージョンや作成日などバージョンごとに異なる情報を表すタグの情報は，DocVersion オブジェクトのプロパティにマッピングできます。
CdbrVersionableDocument オブジェクトに対する XML プロパティマッピングの例を，次の図に示します。

図 3-56　CdbrVersionableDocument オブジェクトに対する XML プロパティマッピングの例



また，XML インデクスデータ作成機能で作成したインデクスデータを基に，DocVersion オブジェクトの全文検索インデクス用プロパティが設定できます。

3.11.2　DocumentBroker が管理できる XML 文書

ここでは，DocumentBroker が XML 文書のコンテントとして管理できる XML ファイルについて説明します。

(1) XML ファイルの形式

DocumentBroker では，次のような XML ファイルを XML 文書のコンテントとして管理できます。

- 外部参照を含まない，単一ファイルで構成された XML ファイル
- イメージファイルなど，構文解析の対象外の外部参照を含む XML ファイル
  ただし，この場合も，XML ファイルそのものは単一ファイルで構成されることが必要です。

また，XML プロパティマッピング機能を使用する場合は，XML ファイルの構造は，次の条件を満たしていることが必要です。

- XML 宣言が記載されている
- 一つのタグで文書全体が囲まれている
- プロパティにマッピングするタグが，一度だけ出現する（VariableArray 型プロパティにマッピングするタグは除く）
- VariableArray 型プロパティにマッピングするタグの構造が，次の図に示す形式である

図 3-57 VariableArray 型プロパティにマッピングできるタグの構造

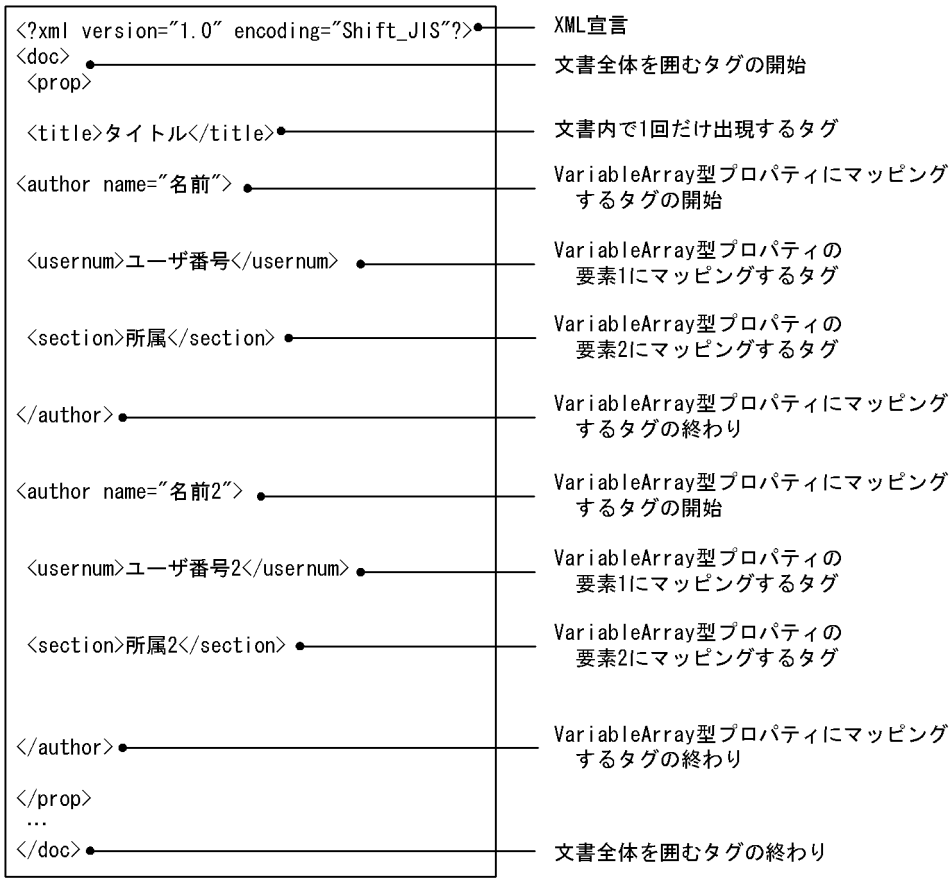
```
〈上位タグ〉
 〈VariableArray型プロパティのタグ〉
  〈要素1のタグ〉要素1の内容〈/要素1のタグ〉
  〈要素2のタグ〉要素2の内容〈/要素2のタグ〉
   …
  〈/VariableArray型プロパティのタグ〉
 〈VariableArray型プロパティのタグ〉
  〈要素1のタグ〉構成要素1の内容〈/要素1のタグ〉
  〈要素2のタグ〉構成要素2の内容〈/要素2のタグ〉
   …
 〈/VariableArray型プロパティのタグ〉
〈上位タグ〉
```

〈VariableArray型プロパティのタグ〉
　CdbrVariableArrayオブジェクトに対応する情報を囲んだタグ
〈要素1のタグ〉および〈要素2のタグ〉
　CdbrCompoundオブジェクトに対応する情報を囲んだタグ

(2) XML プロパティマッピング機能が使用できる XML ファイルの例

XML プロパティマッピング機能を使用できる XML ファイルの例を，次の図に示します。なお，それぞれのタグ名は任意です。

図 3-58 XML プロパティマッピング機能を使用できる XML ファイルの例



次に XML プロパティマッピング機能が使用できない XML ファイルの形式の例と，使用できる形式への変更例を説明します。

使用できない例 1：プロパティにマッピングするタグが複数回出現する XML ファイル

例えば，次の図のような文書です。

図 3-59 マッピングできない例 1：プロパティにマッピングするタグが複数回出現する XML ファイルの例

```
<?xml version="1.0" encoding="Shift_JIS"?>
<doc>
 <prop>
 <title>タイトル</title>
 <title>タイトル2</title>
 </prop>
 …
</doc>
```

例 1 の XML ファイルには，<title> タグで囲まれた情報が複数あるため，一つ値を取るプロパティにマッピングすることはできません。

複数回出現するタグの情報をプロパティにマッピングするためには，VariableArray 型プロパティとしてマッピングする方法があります。例えば，次の図のように複数回出現するタグの前後に VariableArray 型プロパティを表すタグを記述すれば，<title> タグの情報をプロパティにマッピングできます。

図 3-60 例 1 を登録できる構造に変更した例

```
<?xml version="1.0" encoding="Shift_JIS"?>
<doc>
 <prop>
  <list>
   <title>タイトル</title>
  </list>
  <list>
    <title>タイトル2</title>
  </list>
 </prop>
 …
</doc>
```

プロパティの情報

```
usrProp_List[0].usrProp_Title="タイトル"
usrProp_List[1].usrProp_Title="タイトル2"
```

使用できない例 2：要素が繰り返し出現するときに VariableArray 型プロパティを表すタグがない XML ファイル

例えば，次の図のような XML ファイルです。

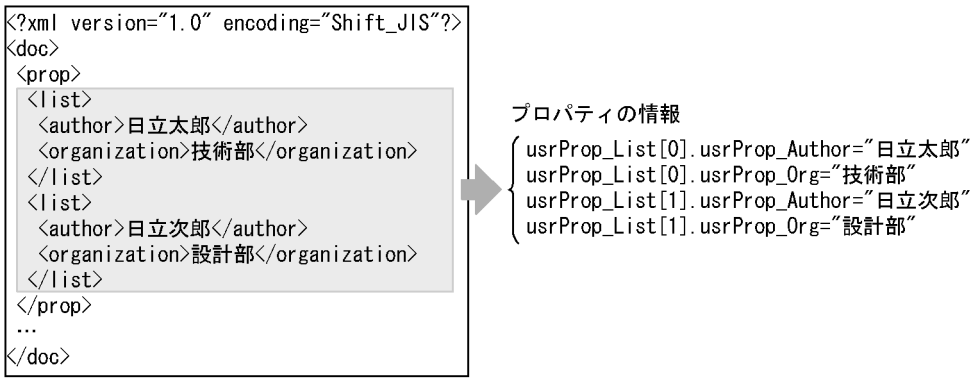
図 3-61 マッピングできない例 2：要素が繰り返し出現するときに VariableArray 型プロパティを表すタグがない XML ファイルの例

```
<?xml version="1.0" encoding="Shift_JIS"?>
<doc>
 <prop>
  <author>日立太郎</author>
  <organization>技術部</organization>
  <author>日立次郎</author>
  <organization>設計部</organization>
 </prop>
 …
</doc>
```

この例では，<author> タグと <organization> タグを繰り返していますが，出現順序が明確にならないため，VariableArray 型プロパティにマッピングできません。

この場合，次の図のように VariableArray 型プロパティを表す <list> タグを追加して構造を変更すれば，VariableArray 型プロパティとして登録できます。

図 3-62　例 2 を登録できる構造に変更した例



3.11.3　XML 文書管理機能を使用した文書の管理

ここでは，XML 文書管理機能を使用した XML 文書の管理方法について説明します。

(1) XML 文書管理機能を使用するための準備

まず，XML プロパティマッピングに使用する定義ファイル類を解析して，XML 文書管理機能を実行するための CdbrXmlTranslator オブジェクトを作成します。これには，CdbrXmlTranslatorFactory クラスの機能を使用します。

手順を次に示します。

1. **CdbrXmlTranslatorFactory::Initialize メソッドをコールして，CdbrXmlTranslatorFactory オブジェクトを初期化します。**
   このメソッドによって，XML プロパティマッピングに使用する定義ファイルが解析されます。
2. **CdbrXmlTranslatorFactory::CreateTranslator メソッドをコールして，CdbrXmlTranslator オブジェクトを作成します。**
   XML 文書管理機能を使用するために必要な CdbrXmlTranslator オブジェクトを作成します。

(2) XML 文書のプロパティにマッピングするプロパティ情報の取得

XML プロパティマッピング機能を使用して，バージョンなしオブジェクトまたはバージョン付きオブジェクトに設定するプロパティの情報を取得します。

「(1)XML 文書管理機能を使用するための準備」で示した手順で CdbrXmlTranslator オブジェクトを作成した状態で，CdbrXmlTranslator::GetDmaInfoList メソッドをコールします。XML ファイルが構文解析されて，プロパティマッピングするための DMA オブジェクト生成用の構造体（SDBR_DMAINFO 構造体）が作成されます。

ここで取得した構造体は，次の時に使用します。

● **XML 文書を作成する時**

XML ファイルをコンテントとする XML 文書を作成する場合に，プロパティの初期値を設定した構造体として使用します。
XML プロパティマッピング機能によって取得した SDBR_DMAINFO 構造体を，

CdbrDocument::CreateObject メソッドまたは CdbrVersionableDocument::CreateObject メソッドの引数に指定します。

● XML 文書のプロパティを更新する時

XML 文書のコンテントを更新した時に，更新するプロパティの値を設定した構造体として使用します。この場合は，XML ファイルから取得した SDBR_DMAINFO 構造体のメンバである，SDBR_PROPLIST 構造体を使用します。
CdbrDocument::PutPropertyValues メソッドまたは CdbrVersionableDocument::PutPropertyValues メソッドの引数に指定します。

(3) XML 文書を構造指定検索するためのインデクスデータの作成

XML インデクスデータ作成機能を使用して，バージョンなしオブジェクトまたはバージョン付きオブジェクトの全文検索インデクスとして登録するインデクスデータを作成します。

「(1)XML 文書管理機能を使用するための準備」で示した手順で CdbrXmlTranslator オブジェクトを作成した状態で，CdbrXmlTranslator::GetDmaInfoList メソッドまたは CdbrXmlTranslator::GetIndexData メソッドをコールします。XML ファイルが構文解析されて，インデクスデータがファイルに出力されます。このとき，フィルタリング定義ファイルの内容によって，不要なタグの情報も除去できます。

なお，GetDmaInfoList メソッドと GetIndexData メソッドのどちらのメソッドをコールした場合も，構文解析が実行されます。XML プロパティマッピング機能も使用する場合は，GetDmaInfoList メソッドで SDBR_DMAINFO 構造体とインデクスデータを同時に作成することをお勧めします。構文解析が一度で済むため，処理時間が短縮できます。

また，すでに XML 文書に登録されているコンテントからは，構造指定検索の対象になるインデクスデータを作成できません。インデクスデータは，クライアント環境にある XML ファイル（file:// で指定できるパスのファイル）を指定して作成してください。

XML インデクスデータ作成機能で出力したインデクスデータ出力ファイルは，次の時に使用できます。

● XML 文書を作成する時

XML ファイルをコンテントとする XML 文書を作成する時に，構造指定検索用全文検索インデクスを作成するためのファイルとして使用できます。
CdbrDocument::CreateObject メソッドまたは CdbrVersionableDocument::CreateObject メソッドで XML 文書を作成してから，CdbrDocument::CreateIndex メソッドまたは CdbrVersionableDocument::CreateIndex メソッドによって全文検索インデクスを作成します。

● XML 文書のコンテントを更新する時

XML 文書のコンテントを更新する時に，更新する XML ファイルの内容に合った全文検索インデクスを作成するためのファイルとして使用できます。
CdbrDocument::UpdateContent メソッドまたは CdbrVersionableDocument::UpdateContent メソッドなどで XML 文書のコンテントを更新してから，CdbrDocument::CreateIndex メソッドまたは CdbrVersionableDocument::CreateIndex メソッドによって全文検索インデクスを作成します。

(4) イメージファイルなど関連を持つファイルの管理

DocumentBroker で XML 文書として管理できるのは，一つのファイルから構成される XML ファイルだけです。

イメージファイルなどの外部エンティティも DocumentBroker でまとめて管理したい場合は，文書間リレーションやコンテナを使用して管理してください。XML 文書およびイメージファイルをバージョン付

き文書，コンテナを構成管理コンテナとして作成すれば，XML 文書とイメージファイルの構成管理もできます。

文書間リレーションについては，「3.8　文書間リレーションを設定した文書管理」を参照してください。コンテナについては，「3.9　コンテナを使用した文書管理」および「3.10　構成管理コンテナを使用した文書の構成管理」を参照してください。

3.11.4　XML 文書の操作

XML 文書の操作を実行する場合に使用するメソッドと，そのメソッドの発行順序の例を説明します。

これらの操作を実行する前に，まず，文書空間と接続してトランザクションを開始してください。

それぞれの操作に使用するメソッドの詳細については，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。

（1）XML 文書の作成

XML プロパティマッピング機能および XML インデクスデータ作成機能を使用して XML 文書を作成する場合の，メソッドと，そのメソッドの発行順序について説明します。

なお，XML プロパティマッピングに必要な定義ファイル類および XML インデクスデータを作成するために必要なフィルタリング定義ファイルは，あらかじめ作成しておきます。それぞれの定義ファイルについては，マニュアル「DocumentBroker Version 3　システム導入・運用ガイド」を参照してください。

1. **CdbrXmlTranslatorFactory クラスを初期化します。**
   CdbrXmlTranslatorFactory::Initialize メソッドをコールします。
2. **CdbrXmlTranslator オブジェクトを作成します。**
   CdbrXmlTranslatorFactory::CreateTranslator メソッドをコールします。
3. **XML プロパティマッピング機能を使用して，DMA 生成用構造体（SDBR_DMAINFO 構造体）を作成します。同時に，XML インデクスデータ作成機能を使用して，インデクスデータをファイルに出力します。**
   CdbrXmlTranslator::GetDmaInfoList メソッドをコールします。
   なお，インデクスデータだけを出力する場合は，CdbrXmlTranslator::GetIndexData メソッドをコールします。
4. **文書空間に接続して，トランザクションを開始します。**
   CdbrSession::Connect メソッドおよび CdbrSession::Begin メソッドをコールします。
5. **手順 3. で作成した DMA 生成用構造体を引数に指定して，CdbrDocument オブジェクトまたは CdbrVersionableDocument オブジェクトを作成します。**
   CdbrDocument::CreateObject メソッドまたは CdbrVersionableDocument::CreateObject メソッドをコールします。
6. **手順 3. でインデクスデータを出力したファイルを引数に指定して，CdbrDocument オブジェクトまたは CdbrVersionableDocument オブジェクトの全文検索インデクスを作成します。**
   CdbrDocument::CreateIndex メソッドまたは CdbrVersionableDocument::CreateIndex メソッドをコールします。

コールシーケンスの例を次に示します。なお，このコールシーケンスは，UNIX の場合です。Windows の場合は，GetDmaInfoList メソッドの引数に指定するファイルパスは「"file:///c:¥temp¥xmlcontent.xml"」，「"file:///c:¥temp¥NorParamFile.txt"」および「"file:///c:¥temp¥index.txt"」

のように指定してください。CreateIndex メソッドの引数に指定するファイルパスは，「"file:///c:¥temp¥index.txt"」のように指定してください。

XML 文書を作成するコールシーケンスの例

```
CdbrSession               Session;
CdbrVersionableDocument   VerDoc;
CdbrXmlTranslatorFactory  XmlTransFactory
CdbrXmlTranslator         XmlTrans;
//CdbrXmlTranslatorFactoryオブジェクトを初期化する
XmlTransFactory.Initialize(pDocSpaceId,
                           pXmsFileName, &pMessage);
//CdbrXmlTranslatorオブジェクトを作成する
XmlTransFactory.CreateTranslator(&XmlTrans);
//XMLファイルからDMAオブジェクト生成用の構造体
//(SDBR_DMAINFO構造体）と
//インデクスデータ出力ファイルを作成する
XmlTrans.GetDmaInfoList("file:///tmp/xmlcontent.xml",
  DBR_XMLPARSE_NO_EXTERNAL_ENTITIES, pMappingId,
  DBR_INDEXTYPE_STRUCTURED, "file:///tmp/NorParamFile.txt",
  "file:///tmp/index.txt", &DmaInfoList, &pMessage);
//セッションを接続してトランザクションを開始する
Session.Connect(pDocSpaceId, pUserId, pPassword);
Session.Begin();
//SDBR_DMAINFO構造体を使用して文書を作成する
VerDoc.CreateObject(&Session, pDmaInfoList->lCount,
pDmaInfoList->pItem, pXmlFile, "text::xml", &pOIID);
//インデクスデータ出力ファイルを使用して全文検索インデクスを作成する
VerDoc.CreateIndex("file:///tmp/index.txt");
Session.Commit();
//セッションを切断する
Session.Disconnect();
```

(2) XML 文書の更新

XML 文書の更新時に，XML プロパティマッピング機能および XML インデクスデータ作成機能を使用して，プロパティの更新と全文検索インデクスの再作成をする場合の，メソッドと，そのメソッドの発行順序について説明します。

なお，XML プロパティマッピングに必要な定義ファイル類および XML インデクスデータを作成するために必要なフィルタリング定義ファイルは，あらかじめ作成しておきます。それぞれの定義ファイルについては，マニュアル「DocumentBroker Version 3　システム導入・運用ガイド」を参照してください。

1. **CdbrXmlTranslatorFactory クラスを初期化します。**
   CdbrXmlTranslatorFactory::Initialize メソッドをコールします。
2. **CdbrXmlTranslator オブジェクトを作成します。**
   CdbrXmlTranslatorFactory::CreateTranslator メソッドをコールします。
3. **XML プロパティマッピング機能を使用して，更新する XML ファイルから DMA 生成用構造体（SDBR_DMAINFO 構造体）を作成します。同時に，XML インデクスデータ作成機能を使用して，インデクスデータをファイルに出力します。**
   CdbrXmlTranslator::GetDmaInfoList メソッドをコールします。
   なお，インデクスデータだけを出力する場合は，CdbrXmlTranslator::GetIndexData メソッドをコールします。
4. **文書空間に接続して，トランザクションを開始します。**
   CdbrSession::Connect メソッドおよび CdbrSession::Begin メソッドをコールします。
5. **更新する XML 文書に接続します。**

接続するオブジェクトに応じて，CdbrDocument クラスまたは CdbrVersionableDocument クラスの，SetOIID メソッドまたは ConnectObject メソッドをコールします。

6. **XML 文書のコンテントを更新します。**
   更新するオブジェクトに応じて，CdbrDocument クラスまたは CdbrVersionableDocument クラスの，UpdateContent メソッドまたは UpdateContentAndRenditionType メソッドをコールします。
   なお，ここで XML 文書のコンテントとして登録する XML ファイルと，手順 3. で XML インデクスデータを作成する時に使用した XML ファイルの内容は，同じにものにしてください。

7. **XML 文書の全文検索インデクスを更新します。このとき，手順 3. で作成したインデクスデータ出力ファイルをメソッドの引数に指定します。**
   更新するオブジェクトに応じて，CdbrDocument クラスまたは CdbrVersionableDocument クラスの，CreateIndex メソッドをコールします。

8. **XML 文書のプロパティを更新します。このとき，手順 3. で作成した DMA オブジェクト生成用構造体（SDBR_DMAINFO 構造体）のメンバである，SDBR_PROPLIST 構造体をメソッドの引数に指定します。**
   更新するオブジェクトに応じて，CdbrDocument クラスまたは CdbrVersionableDocument クラスの，PutPropertyValues メソッドをコールします。

コールシーケンスの例を次に示します。なお，このコールシーケンスは，UNIX の場合です。Windows の場合は，GetDmaInfoList メソッドの引数に指定するファイルパスは「"file:///c:¥temp¥xmlcontent.xml"」，「"file:///c:¥temp¥NorParamFile.txt"」および「"file:///c:¥temp¥index.txt"」のように指定してください。UpdateContent メソッドの引数に指定するファイルパスは，「"file:///c:¥temp¥xmlcontent.xml"」のように指定してください。CreateIndex メソッドの引数に指定するファイルパスは，「"file:///c:¥temp¥index.txt"」のように指定してください。

XML 文書を更新するコールシーケンスの例

```
CdbrSession              Session;
CdbrVersionableDocument  VerDoc;
CdbrXmlTranslatorFactory XmlTransFactory
CdbrXmlTranslator        XmlTrans;
//CdbrXmlTranslatorFactoryオブジェクトを初期化する
XmlTransFactory.Initialize(pDocSpaceId, pXmsFileName,
                           &pMessage);
//CdbrXmlTranslatorオブジェクトを作成する
XmlTransFactory.CreateTranslator(&XmlTrans);
//XMLファイルからDMAオブジェクト生成用の構造体
//(SDBR_DMAINFO構造体）と
//構造指定検索用全文検索インデクスを生成する。
XmlTrans.GetDmaInfoList("file:///tmp/xmlcontent.xml",
  DBR_XMLPARSE_NO_EXTERNAL_ENTITIES, pMappingId,
  DBR_INDEXTYPE_STRUCTURED, "file:///tmp/NorParamFile.txt",
  "file:///tmp/index.txt", &DmaInfoList, &pMessage);
// セッションを接続してトランザクションを開始する
Session.Connect(pDocSpaceId, pUserId, pPassword);
Session.Begin();
//XML文書に接続する
VerDoc.SetOIID(&Session,pOIID);
//XML文書のコンテントを更新する
VerDoc.UpdateContent("file:///tmp/xmlcontent.xml",
                     DBR_NOT_CREATE_INDEX);
//XML文書の全文検索インデクスを更新する
VerDoc.CreateIndex("file:///tmp/index.txt");
//XML文書のプロパティを更新する
SDBR_PROPLIST* pPropList;
pPropList=&DmaInfoList.pItem[0].Proplist;
VerDoc.PutPropertyValues(pPropList);
```

```
Session.Commit();
//セッションの切断
Session.Disconnect();
```

3.12 レンディションのコンテント種別変換機能を使用した文書管理

この節では，レンディションのコンテント種別変換機能を使用した文書管理について説明します。

3.12.1 レンディションのコンテント種別変換機能の概要

ここでは，レンディションのコンテント種別変換機能の概要について説明します。

(1) 機能

レンディションのコンテント種別変換機能は，レンディションのコンテントの格納先（コンテント種別）を，最適な格納先へ変換する機能です。文書の実体であるコンテントの格納先を変化するデータの利用価値に応じて，最適な格納先へ変換したい場合などに使用できます。

(2) 変換できるコンテント種別

レンディションのコンテント種別変換機能で変換できるコンテント種別を，次の表に示します。

表 3-10 レンディションのコンテント種別変換機能で変換できるコンテント種別

変換前のコンテント種別	変換後のコンテント種別
シングルファイル文書	リファレンスファイル文書
リファレンスファイル文書	シングルファイル文書

レンディションのコンテント種別は，レンディションのコンテント種別を表すプロパティを参照することで判別できます。プロパティの詳細は，「3.4.3(5) レンディションのコンテント種別を表すプロパティ（dbrProp_ContentType プロパティ）」を参照してください。

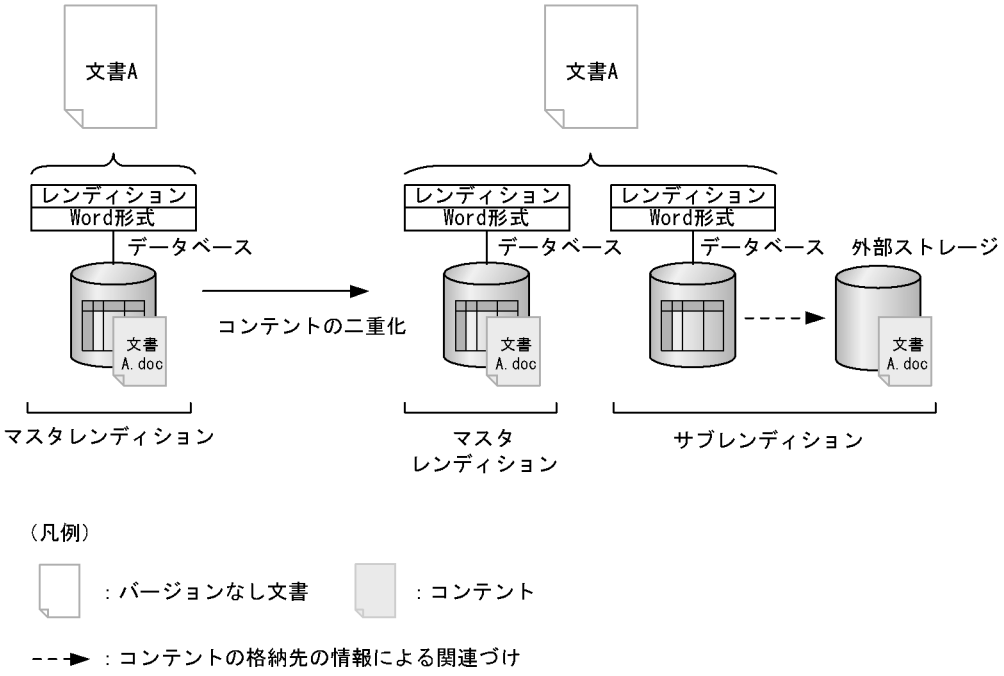
3.12.2 レンディションのコンテント種別変換機能を使用したコンテント種別の変換

ここでは，レンディションのコンテント種別変換機能を使用したコンテント種別の変換について説明します。

(1) コンテント種別の変換

レンディションのコンテント種別変換機能を使用してコンテント種別を変換する例を次の図に示します。

図 3-63　レンディションのコンテント種別変換例



コンテント種別の変換は文書のマルチレンディション管理機能を使用して変換後のコンテントをサブレンディションとして追加します。図 3-63 の例では，シングルファイル文書のコンテントを，リファレンスファイル文書のコンテントに変換し，レンディションを追加します。これをコンテントの二重化と呼びます。コンテントを二重化する場合，追加するレンディションのレンディションタイプにコメントを付与して格納先を分類します。

なお，二重化した後のレンディションを操作する場合は，マルチレンディション管理機能を使用します。

3.12.3　レンディションのコンテント種別変換機能に関する操作

レンディションのコンテント種別変換機能に関する操作を実行する場合に使用するメソッドとそのメソッドの発行順序について説明します。

ここでは，次の操作について説明します。

- シングルファイル文書のコンテントをリファレンスファイル文書のコンテントに変換する。
- リファレンスファイル文書のコンテントをシングルファイル文書のコンテントに変換する。

これらの操作を実行する前に，まず，文書空間と接続してトランザクションを開始してください。また，すべての操作はコンテント種別を変換する文書に接続した状態で実行します。コンテント種別を変換する文書に接続するためには，次の準備をしてください。

準備

コンテント種別を変換する文書の構成要素である DMA オブジェクトの OIID を，検索によってあらかじめ取得します。ここでは，次の操作について説明します。

- CdbrDocument オブジェクトのコンテント種別を変換する場合は，DMA オブジェクトの DocVersion オブジェクトの OIID を取得します。
- CdbrVersionableDocument オブジェクトのコンテント種別を変換する場合は，DMA オブジェクトの ConfigurationHistory オブジェクトの OIID を取得します。

それぞれの操作に使用するメソッドの詳細については，マニュアル「DocumentBroker Version 3　クラス

ライブラリ　C++　リファレンス　基本機能編」を参照してください。

(1) シングルファイル文書のコンテントをリファレンスファイル文書のコンテントに変換

シングルファイル文書として登録されている文書のコンテントをリファレンスファイル文書のコンテントに変換する操作について説明します。

1. **コンテント種別を変換する文書に接続します。**
   接続するオブジェクトに応じて，CdbrDocument クラスまたは CdbrVersionableDocument クラスの，SetOIID メソッド，または ConnectObject メソッドをコールします。

2. **シングルファイル文書のコンテントをリファレンスファイル文書のコンテントに変換します。**
   変換後のリファレンスファイル文書のコンテントを格納するコンテント格納先ベースパスを SetReferencePath メソッドで指定し，ConvertContentType メソッドをコールします。このとき，変換の対象とするレンディションの範囲を１つのレンディションとするかすべてのレンディションとするかを指定し，変換後のレンディションのレンディションタイプにコメントとして設定する文字列を指定します。また，変換後のコンテントを格納するコンテント格納先パスをコンテント格納先ベースパスからの相対パスで指定します。
   シングルファイル文書のコンテントをリファレンスファイル文書のコンテントに変換するコールシーケンスの例を次に示します。

シングルファイル文書のコンテントをリファレンスファイル文書のコンテントに変換するコールシーケンス

```
//pTargetContentPathにコンテント格納先ベースパス設定をしておく。
//…
pSession->Begin
pSession->SetReferencePath(pTargetContentPath);
//…
//コンテントの格納先を変換する文書のOIIDを取得する。
//…
CdbrDocument Doc;
SDBR_REFERENCE_PATHINFO RefInfo;
//RefInfoにリファレンスファイル文書の登録に必要な情報を設定する。
//…
SDBR_RENDITION_COMMENTINFO CommentInfo;
//CommentInfoにコンテント格納先の変換に必要なレンディションコメント情報を設定する。
//…
//コンテントの格納先を変換する文書に接続する。
Doc.SetOIID(pSession, pOIID);
//…
//コンテントの格納先を変換する。
Doc.ConvertContentType(DBR_CONVERT_MODE_VERVOSE,
                       DBR_CONVERT_SOURCE_ALL,
                       DMA_FALSE,
                       NULL,
                       DBR_CONTENTTYPE_CONTENT,
                       CommentInfo,
                       RefInfo);
Doc.ReleaseObject();
//処理の確定
pSession->Commit();
```

(2) リファレンスファイル文書のコンテントをシングルファイル文書のコンテントに変換

リファレンスファイル文書として登録されている文書のコンテントをシングルファイル文書のコンテントに変換する操作について説明します。

1. **コンテント種別を変換する文書に接続します。**
   接続するオブジェクトに応じて，CdbrDocument クラスまたは CdbrVersionableDocument クラスの，

SetOIID メソッド，または ConnectObject メソッドをコールします。

2. **リファレンスファイル文書のコンテントをシングルファイル文書のコンテントに変換します。**
   変換前のリファレンスファイル文書のコンテントを格納したコンテント格納先ベースパスを SetReferencePath メソッドで指定し，ConvertContentType メソッドをコールします。このとき，変換の対象とするレンディションの範囲を 1 つのレンディションとするかすべてのレンディションとするかを指定し，変換後のレンディションのレンディションタイプにコメントとして設定する文字列を指定します。
   リファレンスファイル文書のコンテントをシングルファイル文書のコンテントに変換するコールシーケンスの例を次に示します。

リファレンスファイル文書のコンテントをシングルファイル文書のコンテントに変換するコールシーケンス

```
//pTargetContentPathにコンテント格納先ベースパス設定をしておく。
//…
pSession->Begin
pSession->SetReferencePath(pTargetContentPath);
//…
//コンテント格納先を変換する文書のOIIDを取得する。
//…
CdbrDocument Doc;
SDBR_RENDITION_COMMENTINFO CommentInfo;
//CommentInfoにコンテント格納先の変換に必要なレンディションコメント情報を設定する。
//…
//コンテントの格納先を変換する文書に接続する。
Doc.SetOIID(pSession, pOIID);
//…
//コンテントの格納先を変換する。
Doc.ConvertContentType(DBR_CONVERT_MODE_VERVOSE,
                       DBR_CONVERT_SOURCE_ALL,
                       DMA_FALSE,
                       NULL,
                       DBR_CONTENTTYPE_REFERENCE,
                       CommentInfo);
Doc.ReleaseObject();
//処理の確定
pSession->Commit();
```

3.13 独立データの管理

この節では，独立データの管理について説明します。独立データとは，ほかのオブジェクトに従属しない，独立したオブジェクトです。これは，CdbrIndependentPersistence クラスを継承しているサブクラスを基にして作成します。

3.13.1 CdbrIndependentPersistence クラスの概要

CdbrIndependentPersistence クラスは，ほかのオブジェクトに依存せずに，個々にオブジェクトを生成するクラスです。

CdbrIndependentPersistence クラスは，DMA の拡張クラスである edmClass_IndependentPersistence クラスに対応します。

作成したオブジェクトは，CdbrDMA クラスの機能を継承します。

独立データには，ユーザ定義プロパティを設定しておくことをお勧めします。プロパティを設定しておくことで，DocumentBroker での独立データの取得や検索ができるようになります。

3.13.2 独立データの操作

ここでは，独立データの作成，設定と取得および問い合わせについて説明します。

(1) 独立データの作成

独立データを作成して，データベースに格納する場合，CdbrIndependentPersistence::CreateObject メソッドを使用します。メソッドの発行順序については，「3.2.3(1)　文書の作成」を参照してください。次に，独立データとして CdbrIndependentPersistence オブジェクトを作成する場合の準備について示します。

準備

オブジェクトの構成要素になる DMA オブジェクト作成用のクラス識別子を指定した構造体（SDBR_DMAINFO 構造体）を作成しておきます。CdbrIndependentPersistence オブジェクトを作成する場合は，SDBR_DMAINFO 構造体に，edmClass_IndependentPersistence クラスまたはサブクラスの識別子を指定します。

(2) 独立データの設定と取得

独立データとして作成したオブジェクトでは，プロパティ名を使用してプロパティ値を設定および取得することで，オブジェクトのデータを設定および取得できます。

(3) 独立データの検索

作成した独立データを検索する場合，ユーザ定義プロパティを指定して属性検索することで，独立データのオブジェクトが取得できます。検索については，「4.　オブジェクトの検索」を参照してください。

3.14 排他制御

この節では，DocumentBroker での排他制御について説明します。

オブジェクトを操作するときには，複数のユーザから同時にオブジェクトに対して操作が実行されないように，排他制御をする必要があります。排他制御は，オブジェクトに対してロックを設定することで実現します。

クラスライブラリでは，使用するメソッドによって設定できるロックが異なります。それぞれのメソッドごとの特徴を理解して，適切なメソッドを使用してロックを設定する必要があります。

ここでは，クラスライブラリのメソッドによって設定できるロックの種類と，設定方法について説明します。

3.14.1 ロックの概要

クラスライブラリのメソッドでは，オブジェクトに接続するときに排他制御をするためのロックを設定します。ロックには，次の種類があります。

- read ロック（DMA_LOCK_READ）
- write ロック（DMA_LOCK_WRITE）

それぞれのロックによるオブジェクトの操作について説明します。

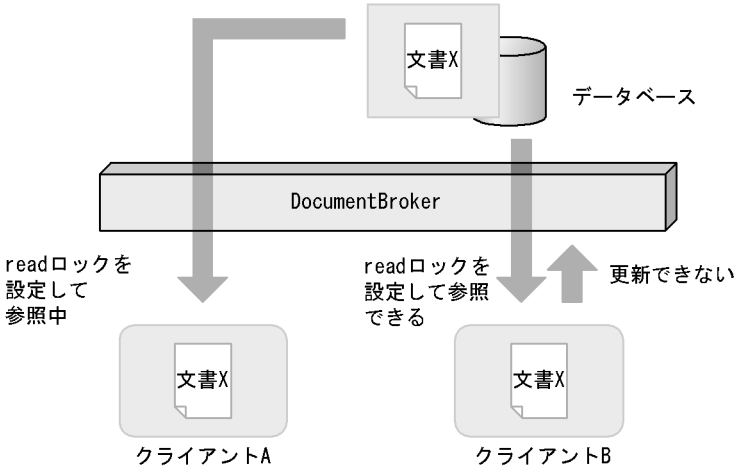
(1) ロックの種類

(a) read ロック（DMA_LOCK_READ）

オブジェクトを参照するときに設定するロックです。あるクライアントが read ロックを設定して取り出したオブジェクトに対して，ほかのクライアントが write ロックを設定してオブジェクトを更新，削除することはできません。ただし，read ロックを設定してオブジェクトを参照することはできます。

read ロックを設定したオブジェクトの操作について次の図に示します。

図 3-64 read ロックを設定したオブジェクトの操作



クライアント A がほかのクライアントより先に文書 X に read ロックを設定して参照している場合，そのほかのクライアントは文書 X に read ロックを設定して参照することはできますが，write ロックを設定して更新および削除することはできません。

●分割取得中の検索結果のオブジェクトに対して更新または削除処理を行う場合（DMA_LOCK_READ | DBR_RLT_FOR_UPDATE**）**

このロック種別（DMA_LOCK_READ | DBR_RLT_FOR_UPDATE）を指定すると，対象オブジェクトには read ロックが設定されて，分割取得中の検索結果のオブジェクトに対して更新または削除処理が実行できます。

このロック種別は次に示すメソッドで指定できます。

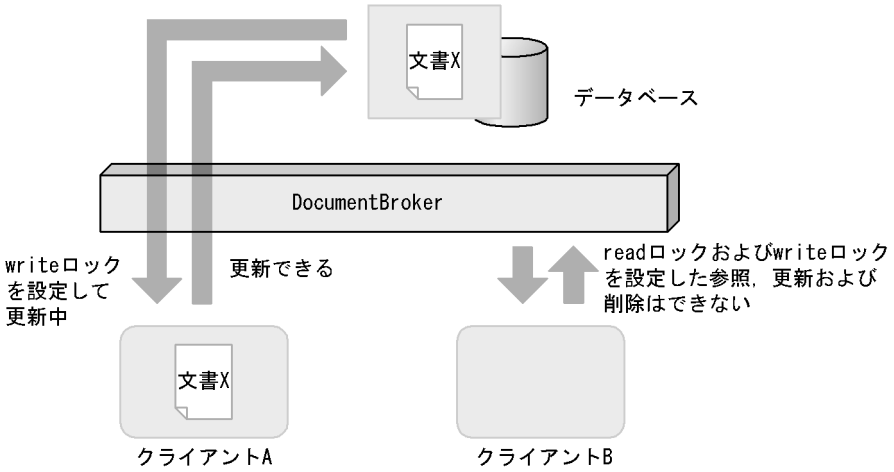
- GetVersionListAndLock
- GetVersionableListAndLock
- GetContainableListAndLock
- GetContainerListAndLock
- GetVTContaineeListAndLock

（b）write ロック（DMA_LOCK_WRITE）

オブジェクトを更新または削除する場合に設定するロックです。あるクライアントが write ロックを設定して取り出したオブジェクトに対して，ほかのクライアントは read ロックまたは write ロックを設定できません。したがって，write ロックを設定した以外のクライアントが，read ロックまたは write ロックを設定してオブジェクトを参照，更新または削除することはできません。

write ロックを設定したオブジェクトの操作について，次の図に示します。

図 3-65　write ロックを設定したオブジェクトの操作



クライアント A がほかのクライアントより先に文書 X に write ロックを設定して更新または削除しようとしている場合，そのほかのクライアントは文書 X に read ロックを設定して参照したり，write ロックを設定して更新および削除することはできません。

●分割取得中の検索結果のオブジェクトに対して更新または削除処理を行う場合（DMA_LOCK_WRITE | DBR_RLT_FOR_UPDATE**）**

このロック種別（DMA_LOCK_WRITE | DBR_RLT_FOR_UPDATE）を指定すると，対象オブジェクトには write ロックが設定されて，分割取得中の検索結果のオブジェクトに対して更新または削除処理が実行できます。

このロック種別は次に示すメソッドで指定できます。

- GetVersionListAndLock
- GetVersionableListAndLock
- GetContainableListAndLock

- GetContainerListAndLock
- GetVTContaineeListAndLock

(2) 複数のクライアント間でのロックの関係

次の表に，複数のクライアント間でのロックの関係について示します。ここでは，クライアント A がオブジェクトにロックを設定している状態でほかのクライアントから設定できるロックの種類を示します。

表 3-11 複数のクライアント間でのロックの関係

クライアント A が設定したロック		read	write
ほかのクライアントが設定するロック	read	○	×
	write	×	×

（凡例）
○：ロックを設定できます。
×：ロックを設定できません。

(3) ロックの有効期間

ロックは，次の時点まで継続します。

- 文書空間との接続が切断されるとき
- トランザクションが決着するとき（CdbrSession::Commit メソッドまたは CdbrSession::Rollback メソッドをコールしたとき）

(4) ロックの遷移

同一トランザクション内でのロックの遷移について説明します。ユーザが同一トランザクション内で一つのオブジェクトに異なる種類のロックを設定するメソッドを発行した場合，オブジェクトに設定されるロックは次の表に示すように遷移します。

表 3-12 同一トランザクション内でのロックの遷移

設定済みのロック		read	write
新たに設定しようとしたロック	read	read のまま	write のまま
	write	write に遷移	write のまま

このように，すでにロックを設定しているオブジェクトに対して異なる種類のロックを設定しようとした場合は，強い種類のロックに変更しようとした場合だけ，ロックの種類が遷移します。

3.14.2 ロックの設定方法

ロックは，メソッドで設定します。クラスライブラリのメソッドによるロックの設定方法には，次の 4 種類があります。

処理と同時にロックを設定するメソッド（～ AndLock メソッド）による明示的なロックの設定

処理を実行するときに明示的にロックを設定するメソッドを使用する方法です。
これらのメソッドでは，事前に SetOIID メソッドによって接続したオブジェクトに対して，処理と同時に指定したロックが設定できます。
SetOIID メソッドと～ AndLock メソッドを組み合わせた操作を実行することで，ConnectObject メソッドによってロックを設定して操作するよりも，データベースへのアクセス回数が減り，高速な処理ができます。

ConnectObject メソッドによる明示的なロックの設定

オブジェクトに接続するときに明示的にロックを設定する，ConnectObject メソッドを使用する方法です。このメソッドでは，クラスライブラリを構成する DMA オブジェクトのうち，トップオブジェクトに当たるオブジェクトにロックを設定します。

検索実行メソッドによる明示的なロックの設定

検索を実行するメソッドによって，検索結果集合に対して明示的にロックを設定する方法です。CdbrEqlStatement::ChangeLockType メソッドの引数として，ロックの種類を明示的に指定します。このメソッドで指定したロックが，CdbrEqlStatement::Execute メソッド実行時に検索結果集合に対して設定されます。

なお，この方法で明示的なロックを設定しない場合は，検索結果集合に対してロックは設定されません。暗黙に設定されるロックはありません。

それ以外のメソッドによる暗黙のロックの設定

ユーザが明示的にロックを設定するメソッドをコールしなかった場合，DocumentBroker によってデータベースのアクセスに必要なロックを暗黙に設定させる方法です。

(1) 処理と同時にロックを設定するメソッド（～ AndLock メソッド）による明示的なロックの設定

ユーザが明示的にロックを設定する場合，処理と同時にロックを設定するメソッドによって必要なロックを設定できます。このメソッドと SetOIID メソッドを組み合わせた操作では，ConnectObject メソッドとロックの指定をしないメソッドを組み合わせて操作する場合に比べて，高速な処理ができます。これは，データベースへのアクセス数が減るためです。

明示的なロックを設定するメソッドについては，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。

処理と同時にロックを設定するメソッドを使用する場合のコールシーケンスを次に示します。

処理と同時にロックを設定するメソッドを使用する例

```
CdbrDocument Obj;
pSession->Begin();
//ObjにOIIDを設定する
Obj.SetOIID( pSession, pOIID);
//データベースからプロパティを参照すると同時に
//WRITE_LOCKモードでロックを設定する
Obj.GetPropertyValuesAndLock( ..., DMA_LOCK_WRITE );
//プロパティを更新する
Obj.PutPropertyValues(...);
//処理の確定
pSession->Commit();
```

(2) ConnectObject メソッドによる明示的なロックの設定

CdbrDMA クラスで定義されている ConnectObject メソッドによって，明示的なロックを設定することもできます。このメソッドでは，オブジェクトに接続すると同時に指定したロックが設定できます。

ConnectObject メソッドでは，そのオブジェクトを構成する DMA オブジェクトのうち，トップオブジェクトにロックを設定します。それ以外の DMA オブジェクトについては，それぞれの参照系メソッドおよび更新系メソッドをコールするときに，必要なロックが設定されます。

なお，ConnectObject メソッドでは，ロックの設定時に WAIT モードまたは NOWAIT モードの指定ができます。モードの指定については，「3.14.3　ロックが設定されているオブジェクトに対する接続」を参照してください。

また，ConnectObject メソッドで read ロックを設定したあとでそのオブジェクトに対して更新系メソッドをコールした場合は，改めて write ロックが設定されます。

ConnectObject メソッドによってロックを設定する場合のコールシーケンスを次に示します。

ConnectObject メソッドを使用する例

```
CdbrDocument Obj;
pSession->Begin();
//データベース中の文書オブジェクトのロックをWRITEモードで設定する
Obj.ConnectObject( pSession, DMA_LOCK_WRITE, pOIID);
//プロパティを参照する
Obj.GetPropertyValues(...);
// プロパティを更新する
Obj.PutPropertyValues(...);
Obj.ReleaseObject();
// 処理の確定
pSession->Commit();
```

(3) DocumentBroker による暗黙のロックの設定

ユーザが明示的にロックを指定しないでデータベースを直接操作するメソッドをコールした場合，DocumentBroker によってデータベースのアクセスに必要なロックが暗黙に設定されます。

暗黙のロックは，クラスライブラリのオブジェクトを構成する DMA オブジェクトのうち，トップオブジェクトに設定されます。それ以外のオブジェクトには，メソッドの実行に必要なロックが設定されます。

DocumentBroker の各メソッドによってトップオブジェクトに暗黙に設定されるロックについては，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。

3.14.3 ロックが設定されているオブジェクトに対する接続

ここでは，すでにほかのユーザによってロックが設定されているオブジェクトに対してロックを設定しようとした場合について説明します。

ロックを設定しようとしたオブジェクトが，すでにほかのユーザからロックを設定されているとき，クラスライブラリのメソッドではデフォルトとしてロックが解除されるまでウエイトするように設定されています。

ただし，ConnectObject メソッドでは，ほかのユーザによってロックが設定されていた場合，対処として次のどちらかを選択できます。

- **ロックが解除されるまでウエイトする（WAIT モード）**
  ほかのユーザがロックを解除するまで，メソッドはウエイトします。ただし，タイムアウトやデッドロックが発生した場合は，終了します。
- **直ちに終了する（NOWAIT モード）**
  ほかのユーザがロックを設定していると，直ちにメソッドを終了します。

3.14.4 ロックの範囲

ここでは，メソッドによって設定されるロックの範囲について説明します。ロックは，クラスライブラリのオブジェクトを構成する DMA オブジェクトのうち，トップオブジェクトとメソッドの処理に必要なオブジェクトに対して設定されます。

まず，メソッドごとのロックの範囲について例を示します。続いて，複数のバージョンを持っているオブ

ジェクトに設定されるロックの範囲について説明します。

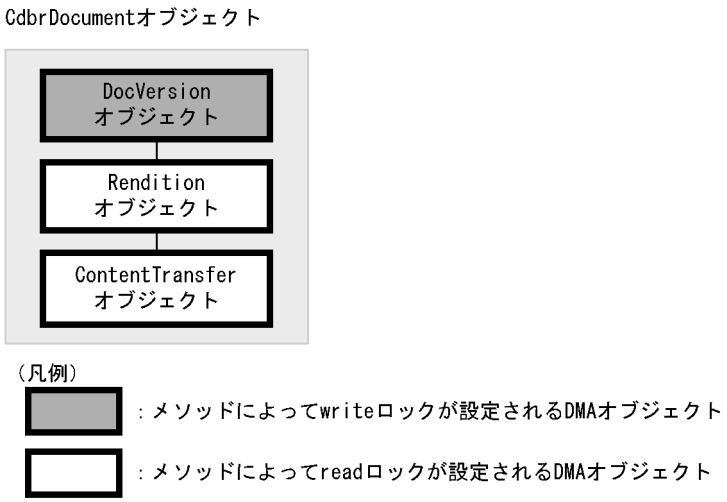
ここでは，CdbrDocument オブジェクトおよび CdbrVersionableDocument オブジェクトを例として説明します。

(1) 処理と同時にロックを設定するメソッド（～ AndLock メソッド）によって設定されるロックの範囲

処理と同時にロックを設定するメソッドでは，DMA オブジェクトのうち，トップオブジェクトと処理に必要なオブジェクトに対してロックを設定します。

例として，CdbrDocument::GetContentAndLock メソッドによって設定されるロックの範囲について次の図に示します。なお，このときメソッドの引数に DMA_LOCK_WRITE を指定して，write ロックを設定することにします。

図 3-66　CdbrDocument::GetContentAndLock メソッドによって設定されるロックの範囲

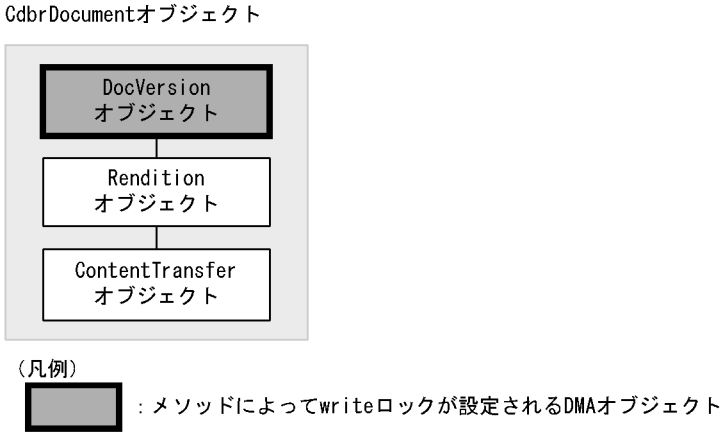


(2) ConnectObject メソッドによって設定されるロックの範囲

ConnectObject メソッドでは，トップオブジェクトにだけロックを設定します。

例として，CdbrDocument::ConnectObject メソッドによって設定されるロックの範囲を次の図に示します。なお，このときメソッドの引数に DMA_LOCK_WRITE を指定して，write ロックを設定することにします。

図 3-67　CdbrDocument::ConnectObject メソッドによって設定されるロックの範囲



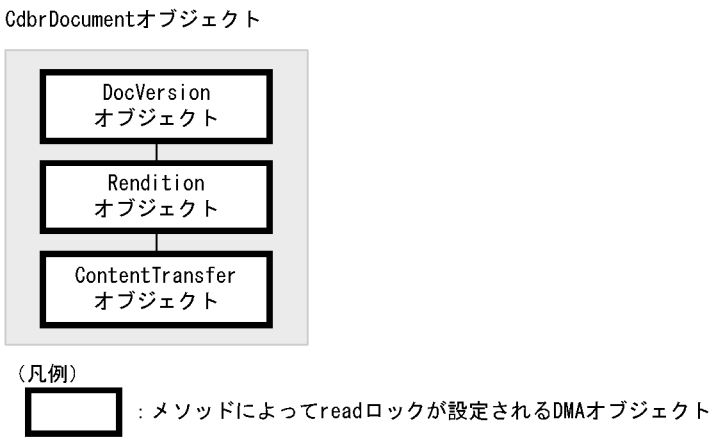
このメソッドによってオブジェクトに接続した場合は，以降のロックを必要とする操作（GetContent メソッドなど）を実行するときには，再びデータベースにアクセスして，操作に関係するオブジェクトに対して必要なロックを設定することになります。

(3) 暗黙のロックを設定するメソッドによって設定されるロックの範囲

ユーザが明示的にロックを指定しないでデータベースを直接操作するメソッドをコールした場合には，DocumentBroker によって最低限必要なロックが暗黙に設定されます。例えば，CdbrDocument::GetContent メソッドをロックの指定なしにコールした場合，必要なオブジェクトに対して read ロックが設定されます。

CdbrDocument::GetContent メソッドによって設定されるロックの範囲を次の図に示します。

図 3-68　CdbrDocument::GetContent メソッドによって設定されるロックの範囲



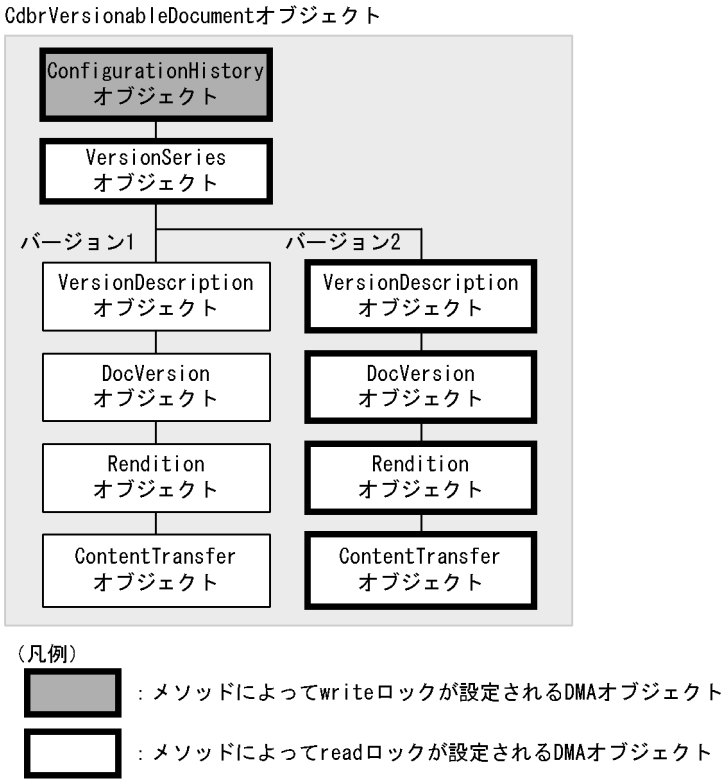
(4) 複数のバージョンを持つオブジェクトに設定されるロックの範囲

複数のバージョンを持つオブジェクトに設定されるロックの範囲について説明します。バージョン管理についての詳細は，「3.3　文書のバージョン管理」を参照してください。

複数のバージョンを持っているオブジェクトに対してメソッドをコールした場合，引数で指定したバージョンに対応する DMA オブジェクトにだけロックを設定します。

例えば複数のバージョンを持つ CdbrVersionableDocument オブジェクトに対して，引数にカレントバージョンのバージョン識別子を指定して CdbrVersionableDocument::GetContentAndLock メソッドをコールした場合，次の図に示す範囲に，ロックが設定されます。なお，このときメソッドの引数に DMA_LOCK_WRITE を指定して，write ロックを設定することにします。

図 3-69　CdbrVersionableDocument::GetContentAndLock メソッドによって設定されるロックの範囲

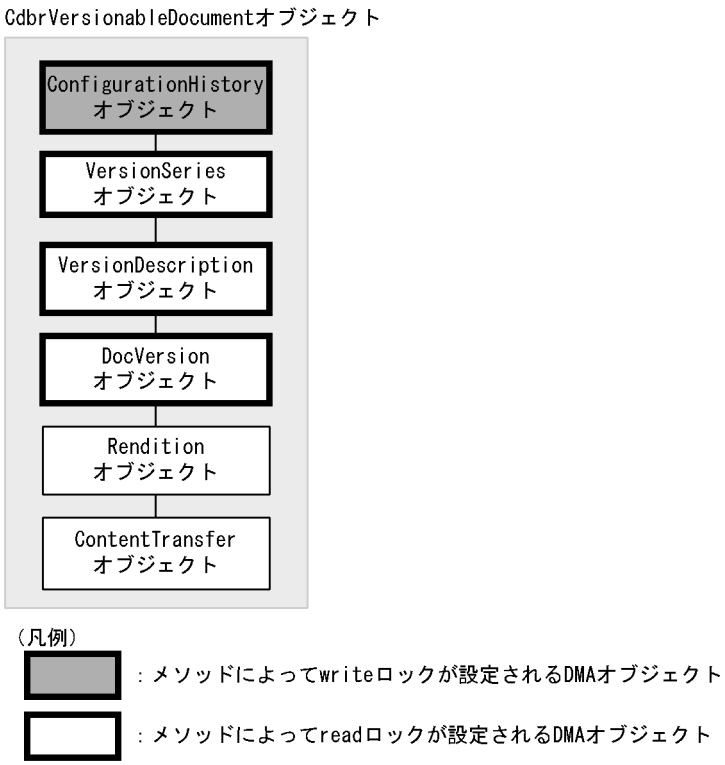


(5) プロパティの取得によって設定されるロックの範囲

プロパティの取得には，CdbrDMA クラスで定義されている GetPropertyValues メソッドまたは GetPropertyValuesAndLock メソッドを使用します。このメソッドをコールすると，トップオブジェクトのほか，複数のバージョンを持つ文書やコンテナの場合は幾つのバージョンを持っているかを示す DMA オブジェクトおよび文書やコンテナを表す DMA オブジェクトにもロックが設定されます。

CdbrVersionableDocument オブジェクトに対して，CdbrVersionableDocument::GetPropertyValuesAndLock メソッドをコールした場合に設定されるロックの範囲を次の図に示します。なお，このときメソッドの引数に DMA_LOCK_WRITE を指定して，write ロックを設定することにします。

図 3-70 CdbrVersionableDocument::GetPropertyValuesAndLock メソッドによって設定されるロックの範囲



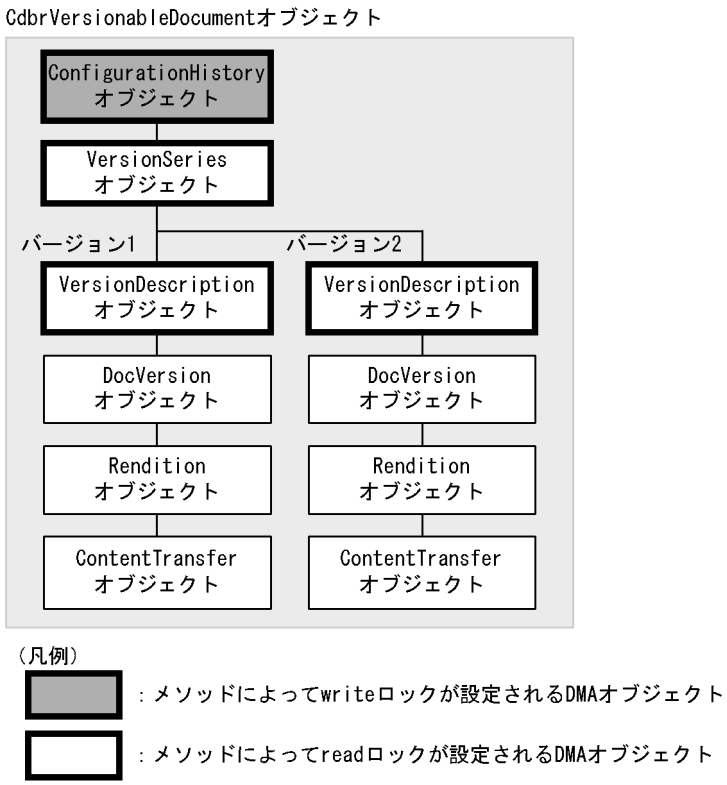
(6) バージョン一覧の取得によって設定されるロックの範囲

バージョン一覧の取得には，CdbrVersionable クラスで定義されている GetVersionList メソッドまたは GetVersionListAndLock メソッドを使用します。このメソッドをコールした場合，トップオブジェクトのほか，幾つのバージョンを持っているかを示す DMA オブジェクトにもロックを設定します。

例えば CdbrVersionableDocument オブジェクトの場合は，ConfigurationHistory オブジェクトのほか，VersionSeries オブジェクトに対してもロックを設定します。

CdbrVersionableDocument::GetVersionListAndLock メソッドによって設定されるロックの範囲を次の図に示します。なお，このときメソッドの引数に DMA_LOCK_WRITE を指定して，write ロックを設定することにします。

図 3-71　CdbrVersionableDocument::GetVersionListAndLock メソッドによって設定されるロックの範囲

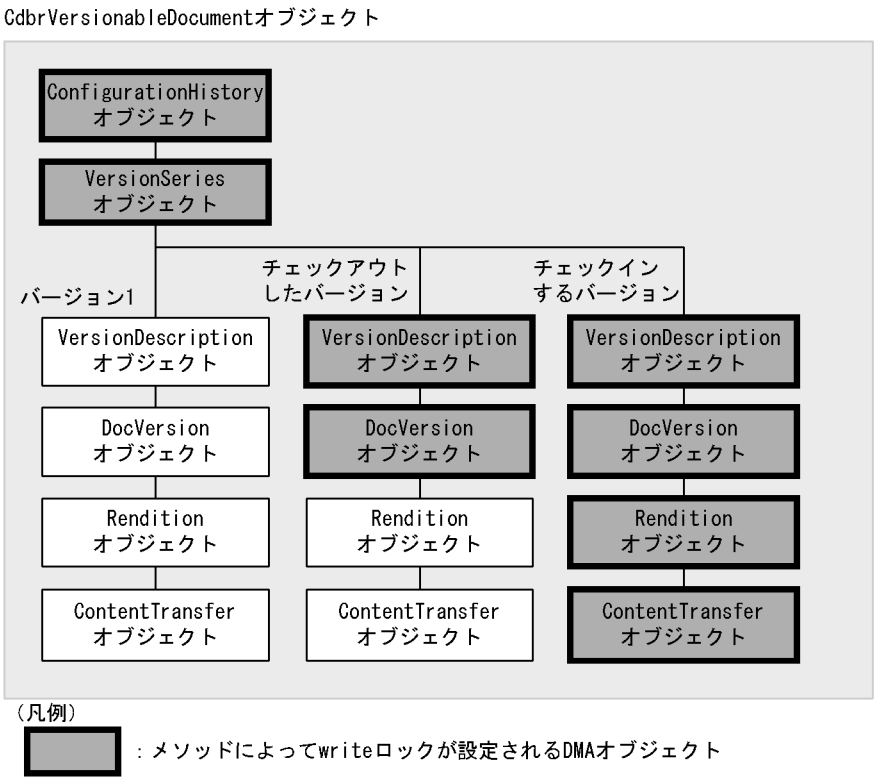


(7) チェックイン，チェックアウトによって設定されるロックの範囲

文書などをバージョンアップするためにチェックイン，チェックアウトをするには，CdbrVersionable クラスなどで定義されている VersionCheckOut メソッドおよび VersionCheckIn メソッドを使用します。これらのメソッドをコールする場合は，バージョンアップする基になるバージョンと，新しく作成するバージョン文書などを構成する DMA オブジェクトに対してロックを設定します。

例えば，CdbrVersionableDocument::VersionCheckOut メソッドおよび CdbrVersionableDocument::VersionCheckIn メソッドによって文書をバージョンアップする場合は，次の図に示す範囲に，ロックが設定されます。

図 3-72 CdbrVersionableDocument::VersionCheckOut メソッドおよび CdbrVersionableDocument::VersionCheckIn メソッドによって設定されるロックの範囲



(8) 検索結果に設定されるロック

オブジェクトの検索を実行した時に，検索結果であるオブジェクトにロックを設定しないと，検索結果を取得する前にデータベースの内容が更新され，正しい検索結果を取得できない可能性があります。

検索結果に設定するロックは，次のメソッドで設定します。

● CdbrEqlStatement::ChangeLockType メソッド
このメソッドでは，検索結果にロックを設定するかどうかの指定，および設定する場合のロックの種別が指定できます。

ただし，検索結果にロックを設定する場合，処理はロックを設定しない場合に比べて検索処理が遅くなります。必要に応じて使い分けてください。

(9) マルチレンディション文書に設定されるロック

マルチレンディション管理をしている文書を操作する場合，ロックは，マルチレンディション文書に対応する CdbrVersionableDocument オブジェクトおよび CdbrDocument オブジェクトを構成する DMA オブジェクトのうち，操作に必要な DMA オブジェクトに対して設定されます。

3.14.5 ロックの設定に関する注意事項

● ロックは，トップオブジェクトとメソッドの実行に必要なオブジェクトにだけ設定されます。しかし，次のような場合は，ロックを設定していてもほかのユーザにロックを設定される可能性がありますので，ご注意ください。
- CdbrVersionableDocument オブジェクトのトップオブジェクトに対して ConnectObject メソッドで

ロックを設定していても，DMA オブジェクトの DocVersion オブジェクトの OIID を使用して，直接 CdbrVersionableDocument オブジェクトを構成する CdbrDocument オブジェクトにロックが設定される可能性があります。

● 次のようなパターンでメソッドをコールする場合は，明示的にロックを設定する必要はありません。
- 参照系メソッドのあとに参照系メソッドをコールする場合
  （例）GetPropertyValues メソッドのあとに GetContent メソッドをコールする場合
- 更新系メソッドのあとに更新系メソッドをコールする場合
  （例）PutPropertyValues メソッドのあとに UpdateContent メソッドをコールする場合
- 更新系メソッドのあとに参照系メソッドをコールする場合
  （例）PutPropertyValues メソッドのあとに GetPropertyValues メソッドをコールする場合

3.15　アクセス制御

この節では，DocumentBroker のアクセス制御機能について説明します。

DocumentBroker で扱うオブジェクトへのアクセスは，必要に応じて制御できます。ここでは，アクセス制御機能の概要，設定，および操作について説明します。

なお，アクセス制御機能を使用するためには，環境設定が必要です。環境設定については，マニュアル「DocumentBroker Version 3　システム導入・運用ガイド」を参照してください。また，アクセス制御機能は，ユーザ管理システムごとのユーザ認証方式によって，使用できる場合と使用できない場合があります。ユーザ認証方式とアクセス制御機能の使用の関係について次の表に示します。

表 3-13　アクセス制御機能とユーザ認証方式との関係

ユーザ管理システム	ユーザ認証方式の指定値	アクセス制御機能の使用可否
OS	BASIC	使用できない
UOC	UOC	使用できる
LDAP 対応のディレクトリサービス	LDAP または LDAPEX	使用できる

3.15.1　アクセス制御機能の概要

ここでは，アクセス制御機能の概要について説明します。

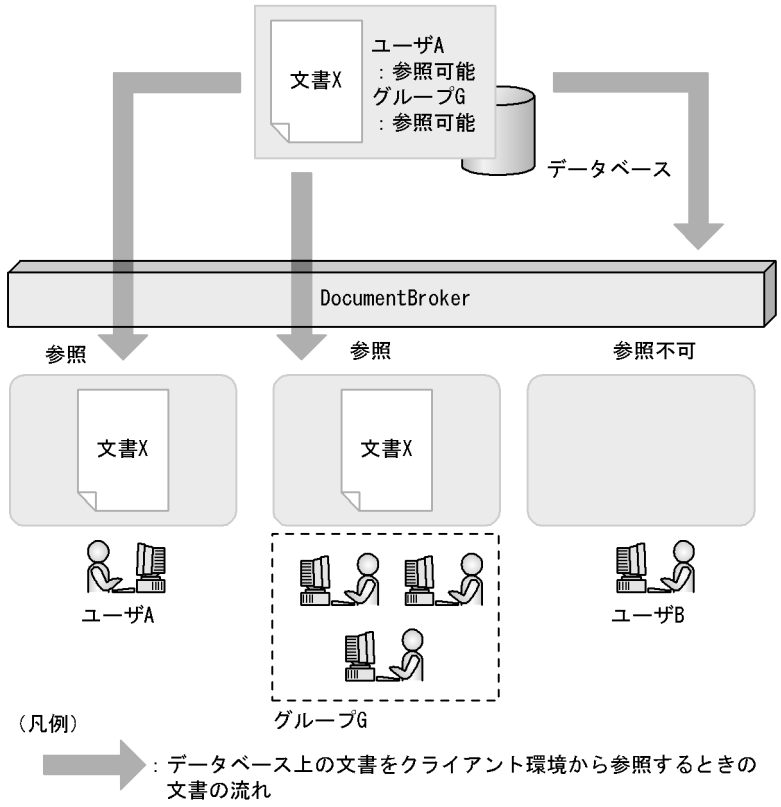
(1) アクセス制御機能とは

DocumentBroker で作成，管理している文書やコンテナなどのオブジェクトは，そのままの状態では，DocumentBroker にログインしたユーザであれば，だれでも参照したり更新したりできます。例えば，あるユーザが作成した文書を DocumentBroker に登録した場合に，ほかのユーザが自由に参照したり，更新したり，削除したりできます。これでは，DocumentBroker 上の文書やコンテナに対して，作成者以外のユーザが悪意を持って変更したり，誤って削除してしまったりすることが考えられます。

こうしたことを防ぐためには，DocumentBroker によって，オブジェクトに対して「どのユーザはどの操作が可能」という情報を設定しておく必要があります。例えば，ユーザ A が作成した「文書 X」に対して，「文書 X は，ユーザ A 以外からは参照だけできて，更新，削除はできないようにしたい」としたり，「文書 X は特定のグループに所属する人だけに参照，更新ができて，それ以外の人には参照もできないようにしたい」としたりしておきます。これによって，文書やコンテナにアクセスするユーザを制限できます。これを実現する機能を，**アクセス制御機能**といいます。また，それぞれのユーザが持つ，オブジェクトにアクセスするための権限を**アクセス権**といいます。

アクセス制御機能の概要を次の図に示します。ここでは，文書 X に対して，ユーザ A とグループ G だけが参照できるように，DocumentBroker によってアクセス制御されています。

図 3-73　アクセス制御の概要（文書 X はユーザ A とグループ G だけが参照できるようにする）



(2) アクセス制御機能の仕組み

ここでは，DocumentBroker のアクセス制御機能の仕組みについて説明します。

DocumentBroker では，ユーザがログインしたときにログインユーザ情報を作成します。ログインユーザ情報には，ログインユーザのユーザ識別子やグループ識別子，そのユーザが持つ特権やそのユーザの権限が設定されます。

また，それぞれのオブジェクトには，そのオブジェクトがだれにどのような操作を許可するかについて表す情報が設定されています。

DocumentBroker では，ユーザ情報と，オブジェクトに設定されている情報を基に，ユーザのアクセス権を判定します。

これらのアクセス権判定に使用するための情報のことを，**アクセス制御情報**といいます。

DocumentBroker によるアクセス制御の概要を次の図に示します。

図 3-74　DocumentBroker によるアクセス制御の概要

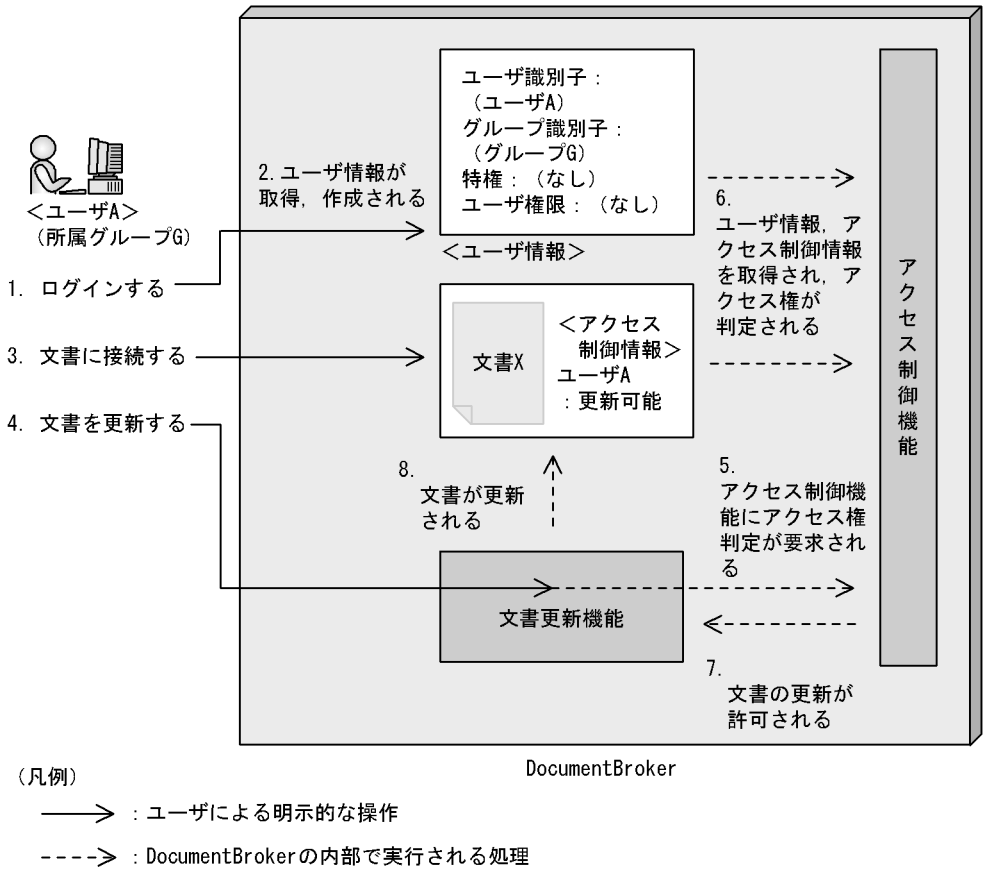


図 3-74 では，ユーザ A（所属しているグループはグループ G）が，文書を更新する場合について説明しています。説明の番号は，図 3-74 の番号と対応しています。なお，説明のうち，ゴシック体の操作はユーザが明示的に実行した操作を，それ以外は DocumentBroker の内部で実行されている処理を示します。

1. **ユーザ A が文書空間にログインします。**
2. 文書空間では，ユーザ A のユーザ識別子とグループ識別子，ユーザの特権およびユーザ権限が取得され，ログインユーザ情報が作成されます。
3. **ユーザ A が文書 X に接続します（SetOIID メソッドまたは ConnectObject メソッドをコールします）。**
4. **ユーザ A が文書 X を更新するメソッド（UpdateContent メソッド）をコールします。**
5. 4. でユーザがコールしたメソッドによって，DocumentBroker の内部で文書を更新する機能（文書更新機能）が動作します。文書更新機能は，アクセス制御機能に対して，ユーザ A のアクセス権の判定を要求します。
   ただし，このときユーザが特権を持つユーザである場合は，アクセス権判定を要求しません。また，ユーザ権限として文書を更新する権限を持つユーザである場合は，文書 X のアクセス制御情報に関係なく，アクセス権が確認されます。
6. アクセス制御機能によって，ユーザ A のログインユーザ情報と，文書 X のアクセス制御情報が取得され，アクセス権が判定されます。
7. ユーザ A のアクセス権が確認できたら，アクセス制御機能は文書更新機能に対して文書の更新を許可します。
8. 文書更新機能によって，文書 X の更新が実行されます。

(3) アクセス制御情報の種類

ここでは，アクセス制御情報の種類について説明します。

アクセス制御情報とは，メソッドを実行したときに，DocumentBroker がアクセス権判定をするために必要な情報のことです。

アクセス制御情報には，次の情報があります。

- **ユーザ情報**
  ユーザごとの識別子や，文書空間全体のオブジェクトに対する権限について示した情報です。
  ユーザ情報は，ユーザが文書空間にログインしたときに作成され，DocumentBroker サーバに保持されます。
- **オブジェクトごとのアクセス制御情報**
  オブジェクトごとに設定された「どのユーザまたはグループ」が「どのような操作」を実行できるかについて示した情報です。
  この情報は，各オブジェクトのプロパティとして保持されています。

(4) アクセス制御情報の構成

アクセス制御情報は，「だれに」「どの操作を」許可するかを示す情報です。この情報は，「アクセスを許可する対象（主体）」と「対象に対して許可する操作」の組み合わせで構成されます。

なお，アクセスを許可する対象を，**サブジェクト**といいます。許可する操作の種別を表す基本単位を**パーミッション**といいます。

3.15.2 ユーザ情報の種類

ここでは，アクセス制御情報のうち，ユーザ情報について説明します。

ユーザ情報は，文書空間にログインしたときにサーバ上に作成されます。なお，ログインしたユーザのユーザ情報は，CdbrSession::GetUserInfo メソッドによって取得して確認できます。

(1) ユーザ識別子とグループ識別子

ユーザ識別子およびグループ識別子は，ユーザが文書空間にログインしたときに使用しているユーザ管理システムから取得される，ログインユーザを識別するための情報です。

個々のオブジェクトに対するアクセス権を判定する場合，ユーザ識別子やグループ識別子によってログインユーザが識別され，オブジェクトに設定されているアクセス制御情報にそのユーザに対するパーミッションが設定されているかどうかが判定されます。

(2) 文書空間に対する特権

特権とは，アクセス権判定なしにすべての操作が実行できる権限です。文書空間に対する特権を持つユーザは，文書空間のすべてのオブジェクトに対しての操作や，オブジェクトの所有者の変更，およびオブジェクトに設定されたアクセス制御情報の変更ができます。

文書空間に対して特権を持つユーザとは，**セキュリティ管理者**のことです。セキュリティ管理者は，文書空間のオブジェクトのアクセス制御情報の保守をするユーザとして，セキュリティ定義ファイル（docaccess.ini）に定義したユーザです。セキュリティ定義ファイルの定義方法については，マニュアル「DocumentBroker Version 3　システム導入・運用ガイド」を参照してください。

ユーザ情報としては，ログインしたユーザがセキュリティ管理者であるかどうかという情報が，サーバに保持されます。

(3) ユーザ権限

ユーザ権限とは，**ユーザ権限定義ファイル**に定義した情報を基に，ユーザ，グループまたはすべてのユーザに対して設定できる権限です。ユーザ権限定義ファイルには，文書空間のすべてのオブジェクトに対するアクセス権を定義します。

ユーザ権限として定義する権限には，次の 2 種類があります。

● オブジェクト作成権限

アクセス制御機能を使用している文書空間で，文書，コンテナおよびパブリック ACL などのオブジェクトを作成する権限です。この権限を与えられたユーザまたはグループだけが，文書空間にオブジェクトを作成できます。

● オブジェクト操作権限

文書空間のすべてのオブジェクトを操作する権限です。文書を参照したり，更新したり，削除したりする権限が定義できます。例えば，ユーザ A には文書空間上のすべての文書またはコンテナなどのオブジェクトを参照できる権限を与えておきたい場合や，すべてのユーザに対して，文書空間上のすべての文書またはコンテナなどのオブジェクトを参照できる権限を与えておきたいという場合に，ユーザ権限として「ユーザ A に参照を許可する」,「すべてのユーザに参照を許可する」という定義をします。これによって，個々のオブジェクトで，これらのアクセス権を設定する必要がなくなります。

なお，ユーザ権限は，すべてのオブジェクトに対する権利であり，強い権利です。例えば，ユーザ権限として参照権が与えられているユーザは，一部のユーザにだけ参照させたいような極秘の文書もすべて参照できるようになります。このため，ユーザ権限を設定する場合は，運用方法を考慮して，注意して設定してください。

ユーザ権限定義ファイルの定義方法や，オブジェクト作成権限およびオブジェクト操作権限として指定できる内容の詳細については，マニュアル「DocumentBroker Version 3　システム導入・運用ガイド」を参照してください。

3.15.3 オブジェクトごとのアクセス制御情報の種類

ここでは，アクセス制御情報のうち，オブジェクトごとに設定するアクセス制御情報について説明します。

オブジェクトごとのアクセス制御情報は，ACFlag または ACL として設定します。また，オブジェクトごとに設定されたアクセス制御情報を管理するための情報も，ACL として設定します。

(1) ACFlag（アクセス制御フラグ）

ここでは，ACFlag について説明します。

ACFlag は，所有者，プライマリグループおよび全ユーザに対して許可する操作を設定できるアクセス制御情報です。

例えば,「所有者にはすべての操作を許可する」としたり,「プライマリグループに所属するユーザには参照，更新を許可する」としたりできます。また,「すべてのユーザに参照を許可する」といった指定もできます。

ACFlag は，データベース上ではトップオブジェクトと同じテーブルにあるプロパティです。このため，ACL を使用したアクセス権判定に比べて高速なアクセス権判定ができるのが特長です。

(a) ACFlag を表すプロパティ

ACFlag は，次の三つのプロパティで表されます。

● dbrProp_OwnerPermission プロパティ
所有者のアクセス権を設定するプロパティです。

● dbrProp_PrimaryGroupPermission プロパティ
プライマリグループに所属するユーザのアクセス権を設定するプロパティです。

● dbrProp_EveryonePermission プロパティ
すべてのユーザのアクセス権を設定するプロパティです。

これらのプロパティは，クラスライブラリのオブジェクトのトップオブジェクトに存在します。

ACFlag の参照には，GetPropertyValues メソッドなどのプロパティを参照するメソッドを使用します。設定や更新には，PutPropertyValues メソッドを使用します。ただし，これらのプロパティを検索に使用することはできません。

(b) ACFlag でアクセス権を設定する対象

ACFlag によってアクセス権を設定できるのは，次の 3 種類のユーザおよびグループです。

所有者

所有者とは，オブジェクトに設定されているアクセス制御情報を変更することができるユーザです。所有者以外のユーザにアクセス制御情報の変更ができるようにしたい場合，まず所有者によってアクセス制御情報を変更する権利を与える必要があります。また，所有者自身を，自分以外のユーザに変更することもできます。
アクセス制御機能を使用している場合，文書空間で作成した文書やコンテナなどのオブジェクトには必ず所有者が存在します。オブジェクトの作成時には，オブジェクトの作成者が所有者として登録されます。
ACFlag では，アクセス制御情報の変更および所有者の変更以外に所有者が実行できる操作を設定できます。

注意

所有者がアクセス制御情報を変更したり所有者を変更したりする場合，対象オブジェクトの OIID を指定して変更します。OIID を取得するために対象オブジェクトのプロパティを参照する場合は，基本プロパティ参照権が設定されている必要がありますので，注意してください。基本プロパティ参照権については，「3.15.4　パーミッションの種類」を参照してください。また，所有者が基本プロパティ参照権を持たない場合，必要に応じてセキュリティ管理者が所有者に基本プロパティ参照権を設定してください。

所有者は，オブジェクトの dbrProp_OwnerId プロパティに設定されたユーザ識別子によって表されます。CreateObject メソッドによってオブジェクトを作成したとき，このメソッドをコールしたユーザのユーザ識別子が，dbrProp_OwnerId プロパティに設定されます。このプロパティは，検索に使用できます。ただし，検索するときにはプロパティ識別子を「edmProp_OwnerId」と指定してください。
所有者を変更する場合は，dbrProp_OwnerId プロパティの値を変更します。このプロパティは，所有者自身またはセキュリティ管理者が変更できます。

プライマリグループ

プライマリグループとは，ACFlag の設定によって ACL よりも先にアクセス権判定が実行されるグループです。
プライマリグループは，オブジェクトごとに一つ設定できます。そのオブジェクトに対してアクセスが多いグループなどを設定しておくことをお勧めします。
例えば，文書を表すオブジェクトのプライマリグループに「第一設計部」を指定して，ACFlag に「プライマリグループに対して文書の参照，編集を許可する」と設定しておけば，その文書は第一設計部に所属する全員が参照，編集できるようになります。

プライマリグループは，オブジェクトの dbrProp_PrimaryGroupId に設定されたグループ識別子によって表されるグループです。オブジェクト作成時にこのプロパティに設定されるグループの情報は，連携しているユーザ管理システムによって異なります。
LDAP 対応のディレクトリサーバと連携してユーザ管理をしている場合，CreateObject メソッドをコールしてオブジェクトを作成した時点では，このプロパティは設定されません。
UOC をユーザ管理システムとして使用する場合，ユーザ管理システムにプライマリグループに対応する概念があり，かつそのグループを返却する UOC が作成されているときは，CreateObject メソッドをコールしたときにプライマリグループが設定されます。それ以外の場合は，CreateObject メソッドをコールしてオブジェクトを作成した時点でこのプロパティは設定されません。
CreateObject メソッドをコールしたときにプライマリグループが設定されない場合，または必要に応じてプライマリグループの値を変更したい場合は，アクセス制御情報変更権を持つユーザが明示的に PutPropertyValues メソッドによって値を設定してください。アクセス制御情報変更権については，「(2)(c) セキュリティ ACL（Security-ACL）」を参照してください。
このプロパティは，検索に使用できます。ただし，検索するときにはプロパティ識別子を「edmProp_PrimaryGroupId」と指定してください。

全ユーザ

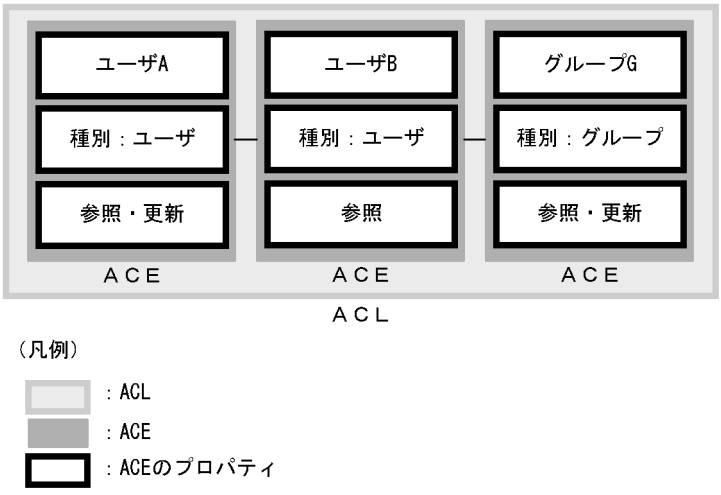
ACFlag では，所有者およびプライマリグループに所属するユーザ以外のすべてのユーザに対するアクセス権も設定できます。

(2) ACL（アクセス制御リスト）

ACL は，ユーザやグループそれぞれに対して，許可する操作を設定して，アクセス権を与えるためのリストです。このリストは，操作を許可する対象（ユーザまたはグループなど），その対象が何であるかの種別，および許可する操作の種類を組み合わせたものを，個々の要素とします。個々の要素は，ACE として作成します。

例えば，文書やコンテナを表すオブジェクトに対して，「ユーザ A に対してはすべての操作を許可する」，「ユーザ B に対しては参照と編集を許可する」，「グループ X に対しては参照だけを許可する」といった詳細なアクセス権の設定ができます。この場合，「ユーザ A に対してすべての操作を許可する」や「ユーザ B に対して参照と編集を許可する」という，個々の対象に対する設定が，ACE に対応します。ACL と ACE の関係を次の図に示します。

図 3-75　ACL と ACE の関係



ACL は，クラスライブラリのオブジェクトのプロパティである，dbrProp_ACL プロパティとして表され

ます。dbrProp_ACL プロパティは，CdbrVariableArray クラスのオブジェクトとして扱う，VariableArray 型プロパティです。個々の ACE は CdbrCompound クラスのオブジェクトとして扱います。このため，ACL および ACE の作成などの操作には，CdbrVariableArray クラスおよび CdbrCompound クラスのメソッドを使用します。

VariableArray 型プロパティとして作成した ACL は，GetPropertyValues メソッドなどを使用して参照できます。また，PutPropertyValues メソッドを使用してクラスライブラリのオブジェクトに設定，更新できます。ただし，このプロパティを検索に使用することはできません。

なお，ACL は，設定されるオブジェクトや用途によって，次の 3 種類に分けられます。

- パブリック ACL
  複数のオブジェクトで共有するための ACL です。
- ローカル ACL
  それぞれのオブジェクトごとに設定する ACL です。
- セキュリティ ACL
  それぞれのオブジェクトごとに設定する，アクセス制御情報を操作するための ACL です。

(a) パブリック ACL

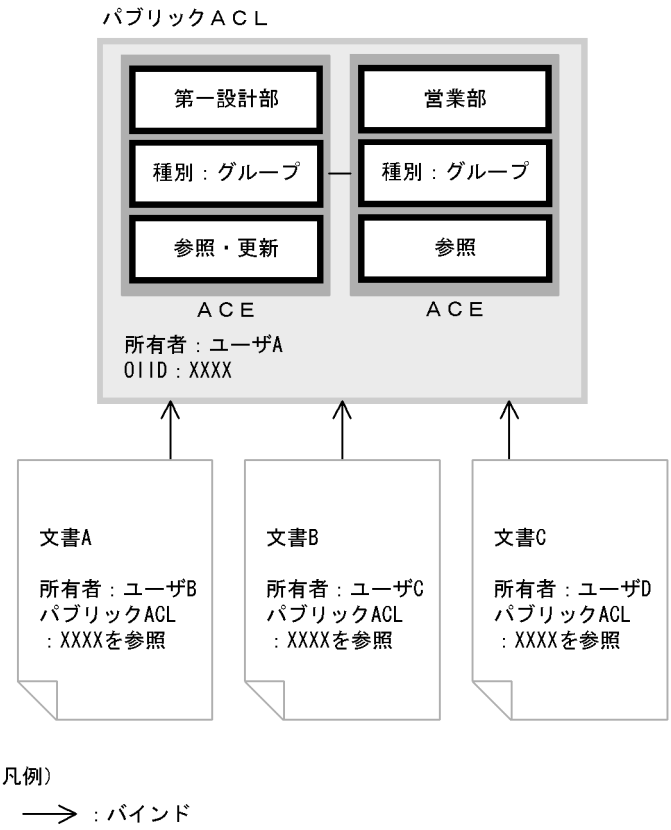
パブリック ACL は，複数のオブジェクトに対して同じアクセス制御情報を設定したい場合に使用します。パブリック ACL を使用すると，複数の文書やコンテナの間で，同じアクセス制御情報を共有できます。

パブリック ACL は，ACL を共有するために使用する，独立したオブジェクトです。このオブジェクトに共有したい ACL を設定し，複数のオブジェクトから参照することで，ACL を共有できます。

例えば，それぞれ異なるユーザが作成した文書 A と文書 B について，どちらの文書にも「第一設計部に所属するユーザ全員に対して参照と編集を許可したい」，また，「営業部に所属するユーザ全員に対して参照だけを許可したい」場合，まず「第一設計部に所属するユーザ全員に対して参照と編集を許可する」という ACE と，「営業部に所属するユーザ全員に対して参照だけを許可する」という ACE を作成します。次に，これらの ACE を要素として，ACL を作成します。この ACL を，パブリック ACL を表すオブジェクトのプロパティ（dbrProp_ACL プロパティ）として設定することで，複数のオブジェクト間で共有できるパブリック ACL が作成できます。文書 A と文書 B には，このパブリック ACL を参照するように設定します。

なお，文書やコンテナを表すオブジェクトからパブリック ACL を参照することを，**バインド**といいます。また，パブリック ACL のバインドを解除することを，**アンバインド**といいます。パブリック ACL の概要を，次の図に示します。

図 3-76　パブリック ACL の概要



パブリック ACL は，CdbrPublicACL クラスのオブジェクトとして CreateObject メソッドをコールして作成します。このオブジェクトは，DMA オブジェクトの，edmClass_PublicACL クラスのオブジェクトを構成要素とします。パブリック ACL として，複数のオブジェクト間で共有するために設定するアクセス制御情報は，CdbrPublicACL オブジェクトの dbrProp_ACL プロパティとして設定します。

また，パブリック ACL は，ローカル ACL やセキュリティ ACL とは異なり，独立して文書空間に存在する，OIID を持つオブジェクトです。ユーザ定義プロパティを設定したり，OIID やユーザ定義プロパティを指定して検索も実行したりすることもできます。ユーザ定義プロパティの設定方法については，マニュアル「DocumentBroker Version 3　システム導入・運用ガイド」を参照してください。

なお，パブリック ACL には，それ自身をアクセス制御するための ACFlag やローカル ACL はありません。パブリック ACL に対するアクセス権は，次に示すユーザに与えられています。

表 3-14　パブリック ACL に対するアクセス権

実行できる操作	対象
作成	文書空間にオブジェクト作成権限を持つユーザ
参照	すべてのユーザ
ローカル ACL とユーザプロパティの変更	セキュリティ ACL でアクセス制御情報変更権が与えられているユーザ，パブリック ACL の所有者およびセキュリティ管理者
セキュリティ ACL の変更	パブリック ACL の所有者およびセキュリティ管理者
削除	パブリック ACL の所有者およびセキュリティ管理者

(b) ローカル ACL

ローカル ACL は，オブジェクトごとのアクセス権を設定するための ACL です。

ローカル ACL は，個々のオブジェクトの dbrProp_ACL プロパティとして設定します。dbrProp_ACL プロパティは，ACE を表すオブジェクトを構成要素とした，VariableArray 型プロパティです。

(c) セキュリティ ACL（Security-ACL）

DocumentBroker では，ACFlag や ACL を使用して，オブジェクトに対するユーザのアクセスを制御します。オブジェクトに対してアクセス権を持たないユーザでも，ACFlag や ACL を変更すれば，アクセス権を得ることができます。このため，ACFlag や ACL は，適切な権利を持つユーザだけが変更でき，それ以外のユーザからは変更されないように制御する必要があります。

アクセス制御情報を変更するための権利を，**アクセス制御情報変更権**といいます。アクセス制御情報変更権は，各オブジェクトのプロパティに，ACL として設定されています。アクセス制御情報変更権を制御するための ACL を**セキュリティ** ACL といいます。

セキュリティ ACL は，オブジェクトの dbrProp_SACL プロパティに，VariableArray 型プロパティとして設定します。

CreateObject メソッドをコールしてオブジェクトを作成した時点では，オブジェクトの所有者およびセキュリティ管理者がアクセス制御情報変更権を持っています。また，CreateObject メソッドの引数として dbrProp_SACL プロパティの初期値を設定した場合は，初期値として設定されたユーザやグループもアクセス制御情報変更権を持ちます。

作成後のオブジェクトで，アクセス制御情報変更権をほかのユーザやグループに対して与えたい場合は，所有者またはセキュリティ管理者が必要に応じてセキュリティ ACL を変更してください。

なお，パブリック ACL はオブジェクトとして独自のセキュリティ ACL を持っています。このため，パブリック ACL をバインドしている文書やコンテナを表すオブジェクトのアクセス制御情報変更権を持つユーザであっても，パブリック ACL に設定された ACL を変更することはできません。また，パブリック ACL のアクセス制御情報変更権を持つユーザが，そのパブリック ACL をバインドしているオブジェクトの ACFlag または ACL を変更することはできません。

3.15.4 パーミッションの種類

アクセス制御情報は，サブジェクトとパーミッションの組み合わせで構成されます。ここでは，アクセス制御情報に設定できるパーミッションの種類について説明します。

パーミッションは，実行可能な操作の範囲を表す値です。アクセス制御情報として設定するパーミッションは，用途によって，次の 3 種類に分けられます。

- 個々のオブジェクトに対する操作を許可するパーミッション
- アクセス制御情報に対する操作を許可するパーミッション（アクセス制御情報変更権）
- 文書空間全体のオブジェクトに対する操作を許可するパーミッション（ユーザ権限）

ユーザ権限はユーザ権限定義ファイルで定義します。このファイルで定義するパーミッションについては，マニュアル「DocumentBroker Version 3　システム導入・運用ガイド」を参照してください。

ここでは，ACFlag および ACL で指定できるパーミッションの種類について説明します。

(1) パーミッションの種別

パーミッションの種別には，次のものがあります。

● 基本パーミッション

● 組み合わせパーミッション

● アクセス制御情報変更権

個々のオブジェクトに対する操作を許可するパーミッション（ACFlag や ACL に指定するパーミッション）には，基本パーミッションおよび組み合わせパーミッションを使用します。パーミッションは複数指定できます。例えば，複数の基本パーミッションを指定したり，複数の組み合わせパーミッションを指定したり，基本パーミッションと組み合わせパーミッションを混在させて指定したりできます。この場合，パーミッションとして設定される内容は，指定したパーミッションの論理和になります。

アクセス制御情報変更権は，アクセス制御情報に対する操作を許可するパーミッションであり，セキュリティ ACL にだけ指定できます。

(2) 基本パーミッション

オブジェクトを操作する範囲を限定するために，DocumentBroker が提供する基本的なパーミッションは次のとおりです。これを**基本パーミッション**といいます。基本パーミッションの種類を次の表に示します。

表 3-15　基本パーミッションの種類

パーミッションの名称	パーミッション文字列	定数	説明
基本プロパティ参照権	PRIM_READ_PROPS	DBR_PERM_PRIM_READ_PROPS	プロパティの参照を許可する
基本プロパティ更新権	PRIM_WRITE_PROPS	DBR_PERM_PRIM_WRITE_PROPS	プロパティの更新を許可する
基本コンテント参照権	PRIM_READ_CONTENTS	DBR_PERM_PRIM_READ_CONTENTS	文書のコンテントの参照を許可する
基本コンテント更新権	PRIM_WRITE_CONTENTS	DBR_PERM_PRIM_WRITE_CONTENTS	文書のコンテントの更新を許可する
基本リンク権	PRIM_LINK	DBR_PERM_PRIM_LINK	リンクに関する操作を許可する
基本バージョン管理権	PRIM_VERSION	DBR_PERM_PRIM_VERSION	バージョンに関する操作を許可する
基本オブジェクト削除権	PRIM_DELETE	DBR_PERM_PRIM_DELETE	オブジェクトの削除を許可する

基本パーミッションでは，それぞれのパーミッションは独立であり，包含関係はありません。複数の基本パーミッションを設定した場合は，それらの論理和がサブジェクトに対して許可される操作になります。ただし，基本プロパティ参照権は，すべての基本パーミッションに含まれます。例えば，基本コンテント参照権を設定すれば，同時に，基本プロパティ参照権も設定したことになります。基本プロパティ参照権と基本コンテント参照権を個別に設定する必要はありません。

次に，それぞれの基本パーミッションの内容について説明します。

基本プロパティ参照権

オブジェクトのプロパティを参照する権利です。
このパーミッションを設定されたユーザは，文書やコンテナの OIID や，ユーザ定義プロパティを参照できます。プロパティを指定した属性検索を実行する場合も，このパーミッションが必要です。
また，このパーミッションを与えないユーザに対しては，オブジェクトの存在もわからないようにで

きます。
このパーミッションには，コンテナに設定されたリンクのプロパティを参照したり，コンテナと関連づけられている要素のプロパティを参照したりする権利も含まれます。

基本プロパティ更新権
オブジェクトのプロパティを更新する権利です。
このパーミッションを設定されたユーザは，文書やコンテナに設定されたユーザ定義プロパティや，そのほかの読み取り専用以外のプロパティを更新できます。

基本コンテント参照権
文書を表すオブジェクトのコンテントを参照する権利です。
このパーミッションを設定されたユーザは，GetContent メソッドなどによって文書の内容を参照できます。
このパーミッションには，全文検索を実行する権利も含まれます。つまり，全文検索インデクスを使用した検索を実行する場合には，このパーミッションが必要です。

基本コンテント更新権
文書を表すオブジェクトのコンテントを更新する権利です。
このパーミッションを設定されたユーザは，UpdateContent メソッドなどによって，文書の内容を更新できます。
このパーミッションには，全文検索インデクスを作成，削除する権利も含まれます。

基本リンク権
リンクに関する操作をする権利です。
リンクに関する操作とは，コンテナと文書またはコンテナとコンテナを関連づけたり，関連づけを解除したりする操作です。また，このパーミッションにはリンクのプロパティを更新する権利も含まれます。これによって，例えば，コンテナと文書を関連づけた時に，関連づけた日時をリンクのプロパティに設定したりできます。構成管理コンテナの場合は，構成管理モードを変更する権利も含まれます。

基本バージョン管理権
バージョンに関する操作をする権利です。
バージョンに関する操作とは，バージョンアップやバージョン削除をする権利です。チェックアウト，チェックイン，チェックアウトの取り消しなどをする権利が含まれます。

基本オブジェクト削除権
RemoveObject メソッドなどによって，オブジェクトを削除する権利です。

(3) 組み合わせパーミッション

組み合わせパーミッションとは，基本パーミッションを組み合わせて定義されたパーミッションです。あるオブジェクトに対するパーミッションを指定する場合に，一般的に同時に設定すると考えられる複数の基本パーミッションが，まとめて一つのパーミッションとして指定できるようにされています。

組み合わせパーミッションと基本パーミッションとの対応を次の表に示します。

表 3-16 組み合わせパーミッションと基本パーミッションの対応

組み合わせパーミッションの名称	パーミッション文字列	基本パーミッション（パーミッション文字列）						
		PRIM_READ_PROPS	PRIM_WRITE_PROPS	PRIM_READ_CONTENTS	PRIM_WRITE_CONTENTS	PRIM_LINK	PRIM_VERSION	PRIM_DELETE
プロパティ参照権	READ_PROPS	○	－	－	－	－	－	－
参照権	READ	○	－	○	－	－	－	－
プロパティ更新権	WRITE_PROPS	○	○	－	－	－	－	－
参照更新権	READ_WRITE	○	○	○	○	－	－	－
削除権	DELETE	○	－	－	－	－	－	○
リンク権	LINK	○	－	－	－	○	－	－
バージョン権	VERSION	○	○	○	○	－	○	－
フルコントロール	FULL_CONTROL	○	○	○	○	○	○	○

（凡例）
○：組み合わせパーミッションに対応する基本パーミッションです。
－：組み合わせのパーミッションに対応しない基本パーミッションです。

組み合わせパーミッションを表すパーミッション文字列を表す定数について，次の表に示します。

表 3-17 組み合わせパーミッションのパーミッション文字列を表す定数の対応

パーミッションの名称	パーミッション文字列	定数
プロパティ参照権	READ_PROPS	DBR_PERM_READ_PROPS
参照権	READ	DBR_PERM_READ
プロパティ更新権	WRITE_PROPS	DBR_PERM_WRITE_PROPS
参照更新権	READ_WRITE	DBR_PERM_READ_WRITE
削除権	DELETE	DBR_PERM_DELETE
リンク権	LINK	DBR_PERM_LINK
バージョン権	VERSION	DBR_PERM_VERSION
フルコントロール	FULL_CONTROL	DBR_PERM_FULL_CONTROL

組み合わせパーミッションを使用すると，一つ指定することで複数の基本パーミッションを指定した場合と同じパーミッションが設定できます。

例えば，あるユーザに対して文書のコンテントの更新を許可したい場合，運用によっては，「検索によって文書を取得し，既存のコンテントの内容を参照した上で，修正，更新する」という操作の流れが考えられます。また，「コンテントを更新する場合には，更新した日時を表すプロパティを更新する」という操作があるとします。この場合，基本パーミッションでは，「基本プロパティ更新権」，「基本コンテント参照権」および「基本コンテント更新権」の 3 種類のパーミッションをそのユーザに設定する必要があります。これに対して，組み合わせパーミッションの「参照更新権」だけ指定すれば，3 種類の基本パーミッション

を指定した場合と同じパーミッションとして定義できます。

次に，それぞれの運用での，組み合わせパーミッションの設定例について示します。

プロパティ参照権（READ_PROPS）
コンテナの階層をたどったり，コンテナの名称や文書の名称などのオブジェクトのプロパティとして設定されている情報だけを参照できるようにするユーザに対して設定します。

参照権（READ）
文書名などのプロパティを参照した上で，文書のコンテントの参照も許可したいユーザに対して設定します。

プロパティ更新権（WRITE_PROPS）
文書名などのプロパティを参照して，そのプロパティの値の変更を許可したいユーザに対して設定します。
複数のユーザが作成した文書を集めて整理するユーザなどに設定します。

参照更新権（READ_WRITE）
コンテナや文書をプロパティ，コンテントも含めて参照，更新することを許可したいユーザに対して設定します。

削除権（DELETE）
文書名やコンテナ名など，プロパティとして設定されている情報を参照させて，その文書やコンテナの削除を許可したいユーザに対して設定します。
使われていない文書を削除してデータベースを保守するユーザなどに設定します。

リンク権（LINK）
コンテナに対して，コンテナの名称等は変更させずに，コンテナに文書を格納させるユーザに対して設定します。
すでにフォルダの体系が決定されている場合に，あるユーザにその体系に従って文書を格納させたいときなどに設定します。

バージョン権（VERSION）
バージョン管理を実行させたいユーザに対して設定します。
ただし，文書をバージョンアップする場合などには，バージョン管理の対象になる文書に対してこのパーミッションを与えると同時に，個々のバージョンに対応する文書に対して参照更新権が必要です。
詳細は，「3.15.7　オブジェクトの操作とアクセス制御情報の関係」を参照してください。

フルコントロール（FULL_CONTROL）
そのオブジェクトに対するすべての操作を許可したいユーザに対して設定します。

また，組み合わせパーミッションを指定する場合に，そのパーミッションに含まれない操作をする権利も含ませたいときは，組み合わせパーミッションと基本パーミッションを組み合わせて指定できます。例えば，「参照更新権」を与えたユーザに対して，「オブジェクトを削除する権利も与えたい」という場合は，組み合わせパーミッションの「参照更新権」と，基本パーミッションの「基本オブジェクト削除権」をあわせて指定できます。

(4) アクセス制御情報変更権

アクセス制御情報変更権は，セキュリティ ACL に設定できるパーミッションです。

セキュリティ ACL には，オブジェクトのアクセス制御情報を操作するユーザに対するパーミッションを指定します。

アクセス制御情報変更権について次の表に示します。

表 3-18　アクセス制御情報を操作するためのパーミッション

パーミッションの名称	パーミッション文字列	定数
アクセス制御情報変更権	CHANGE_PERM	DBR_PERM_CHANGE_PERM

文書やコンテナなどを示すオブジェクトのセキュリティ ACL でアクセス制御情報変更権を与えられたユーザは，そのオブジェクトの ACFlag およびローカル ACL の変更，パブリック ACL のバインドおよびアンバインドができます。

パブリック ACL のセキュリティ ACL のパーミッションとしてアクセス制御情報変更権を設定されたユーザは，そのパブリック ACL に設定されている ACL やユーザ定義プロパティを変更できます。

注意

アクセス制御情報変更権が設定されているユーザがアクセス制御情報を変更したり所有者が所有者を変更したりする場合，対象オブジェクトの OIID を指定して変更します。OIID を取得するために対象オブジェクトのプロパティを参照する場合は，基本プロパティ参照権が設定されている必要がありますので，注意してください。

3.15.5　それぞれのアクセス制御情報の特徴

ここでは，これまで説明してきたアクセス制御情報について，それぞれの特徴を説明します。アクセス制御情報は，アクセス権判定の順序や用途によって，それぞれ特徴を持っています。これらの特徴を考慮した上で，文書空間またはオブジェクトに適切なアクセス制御情報を設定してください。

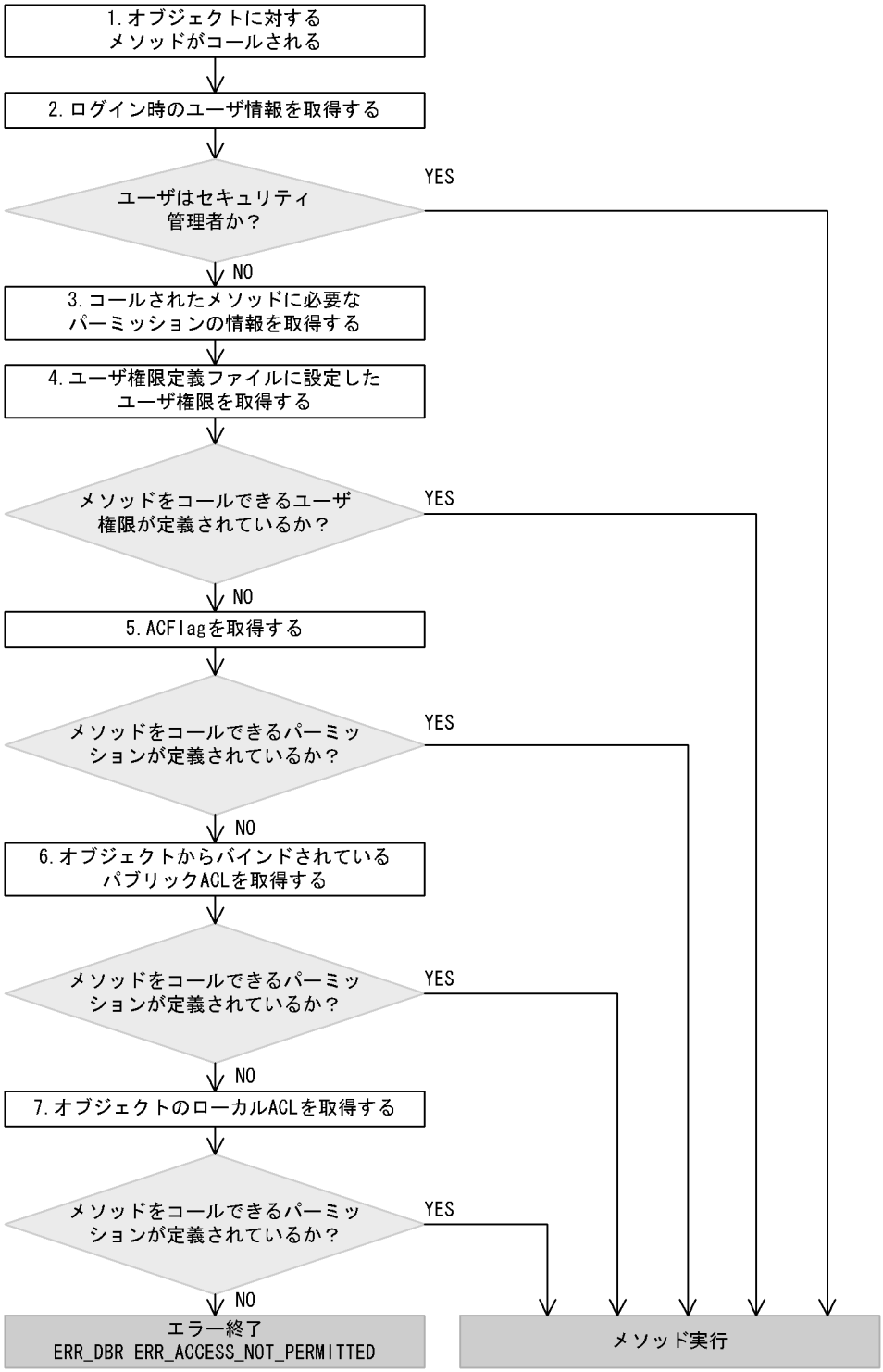
(1) アクセス権判定の流れ

ここでは，アクセス制御情報によって，どのようにオブジェクトに対するアクセス権が判定されるかについて説明します。

DocumentBroker によるアクセス権の判定処理の流れについて，次の図に示します。

なお，メソッドをコールするユーザに関するユーザ情報は，ログインしたときに DocumentBroker によって取得されています。このユーザ情報の詳細については，マニュアル「DocumentBroker Version 3　システム導入・運用ガイド」を参照してください。

図 3-77　アクセス権の判定処理の流れ



1. オブジェクトに対してメソッドがコールされます。
2. アクセス制御機能が，ユーザがログインした時に取得されたユーザ情報を取得します。
   ユーザ情報には，ログインユーザに関する次の情報が設定されています。
   - ユーザ識別子
   - グループ識別子
   - 文書空間に対するユーザの特権（セキュリティ管理者であるかどうかの情報）

- ユーザ権限

メソッドをコールしたユーザがセキュリティ管理者である場合は，メソッドを実行します。セキュリティ管理者でない場合は，3. に進みます。

3. ユーザがコールしたメソッドに必要なパーミッションの情報を取得します。
   各メソッドの実行に必要なパーミッションについては，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。
4. ユーザ権限定義ファイルに設定したパーミッション（ユーザ権限）を取得します。ユーザに設定されているパーミッションで，メソッドが実行できるか判定します。
   メソッドの実行が可能なパーミッションが設定されていた場合は，メソッドが実行されます。
   メソッドの実行が可能なパーミッションが設定されていなかった場合は，5. に進みます。
5. ACFlag を取得します。メソッドをコールしたユーザが ACFlag でパーミッションを与えられているユーザまたはグループかどうかを判定します。また，そのパーミッションでメソッドが実行できるかどうかを判定します。
   メソッドの実行が可能なパーミッションが設定されている場合，メソッドを実行します。
   ユーザにパーミッションが設定されていない場合，またはメソッドの実行が可能なパーミッションが設定されていない場合は，6. に進みます。
6. オブジェクトにバインドされているパブリック ACL を取得します。ユーザ情報に登録されているユーザが，パブリック ACL でパーミッションを与えられているかを判定します。また，そのパーミッションでメソッドが実行できるか判定します。
   メソッドの実行が可能なパーミッションが設定されている場合は，メソッドを実行します。
   ユーザにパーミッションが設定されていない場合，またはメソッドの実行が可能なパーミッションが設定されていない場合は，7. に進みます。
7. ローカル ACL を取得します。ユーザ情報に登録されているユーザまたはグループが，ローカル ACL でパーミッションを与えられているかを判定します。また，そのパーミッションでメソッドが実行できるかどうかを判定します。
   メソッドの実行が可能なパーミッションが設定されている場合は，メソッドを実行します。
   ユーザにパーミッションが設定されていない場合，またはメソッドの実行が可能なパーミッションが設定されていない場合は，DocumentBroker はメソッドを実行せずに，エラー（ERR_DBR ERR_ACCESS_NOT_PERMITTED）を返却して，エラー終了します。

このように，例えば，ACFlag でユーザの操作に必要なパーミッションが設定されている場合は，DocumentBroker はパブリック ACL やローカル ACL を参照しません。したがって，例えば，アクセスが多くなるユーザに対しては ACFlag でパーミッションを与えるようにすることで，アクセス権判定の処理速度が向上します。

注意

メソッドをコールした時点で，アクセス権判定の前に，オブジェクトにはロックが設定されます。このロックは，アクセス権がないためにメソッドがアクセスエラーになった場合も解除されません。アクセスエラーが発生した場合は，ロールバックして，トランザクションを終了することで，ロックを解除してください。

(2) アクセス制御情報の特徴

それぞれのアクセス制御情報の特徴を，次の表に示します。

表 3-19 それぞれのアクセス制御情報の特徴

アクセス制御情報の種類	処理順序※	特徴
文書空間に対する特権	1	• アクセス制御情報のうち，最初に参照されます。この特権を持つユーザに対してはアクセス権判定処理が実行されません。 • 文書空間内のセキュリティに関する保守を行うために登録されたセキュリティ管理者に与えられた権限です。
ユーザ権限	2	• アクセス権判定のとき，最初に参照される情報なので，最も早い段階でアクセス権が判定されます。 • あるユーザに対して，文書空間内で同一の操作を許可する場合に使用すると便利です。 • ユーザ権限を多数定義すると，起動処理および文書空間への接続処理に時間が掛かることがあります。
ACFlag	3	• 所有者，プライマリグループの情報と組み合わせて判定されます。 • ユーザ権限の次に参照される情報なので，ローカル ACL やパブリック ACL に比べて早い時点で判定されます。 • データベース上ではトップオブジェクトのプロパティと同じテーブルに存在するプロパティなので，処理が高速です。 • 所有者，プライマリグループまたはすべてのユーザに対して許可する操作だけ設定できます。
パブリック ACL	4	複数のオブジェクトに対して共通のアクセス制御情報が設定できます。
ローカル ACL	5	そのオブジェクト固有のアクセス制御情報が設定できます。

注※

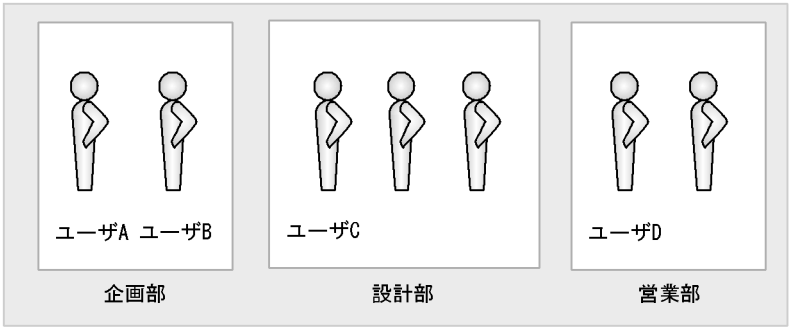
DocumentBroker がアクセス権を判定する場合に，処理する順序です。

(3) アクセス制御情報の用途による使い分けの例

ユーザ権限，ACFlag，パブリック ACL およびローカル ACL の使い分け方について例を使用して説明します。なお，セキュリティ管理者が持つ特権については，文書空間全体のアクセス制御情報を保守するユーザの権限ですので，ここでは説明しません。文書空間のセキュリティについて考慮した上で，別に設定してください。

ここでは，次の図のような組織でのアクセス制御機能を使用した文書管理について説明します。

図 3-78 アクセス制御情報の使い分けの例で使用する組織の構成



以降，ユーザ権限，ACFlag，パブリック ACL およびローカル ACL を設定する場合の考え方を，アクセス権判定の順に説明していきます。なお，セキュリティ ACL は，アクセス権判定の処理とは無関係に設定するアクセス制御情報です。

1. **ユーザ権限定義ファイルの指定によるユーザ権限の設定**

   ユーザ権限定義ファイルでは，文書空間内で共通で利用するアクセス制御情報を設定できます。

   - **オブジェクト作成権限の設定**

アクセス制御機能を使用している文書空間では，オブジェクトを作成できるユーザを設定する必要があります。この設定は，ユーザ権限定義ファイルで定義します。
例えば，図 3-78 の組織の場合に，ユーザ A，ユーザ C にオブジェクト作成権限を設定します。これによって，ユーザ A およびユーザ C は，文書空間に文書やコンテナが作成できるようになります。

- **オブジェクト操作権限の設定**
  ユーザ権限定義ファイルには，特定のユーザおよびすべてのユーザに，文書空間内のすべてのオブジェクトに対して許可する操作が設定できます。例えば，すべてのユーザに対してすべての文書またはコンテナを参照する権利を与える場合など，このファイルで設定できます。

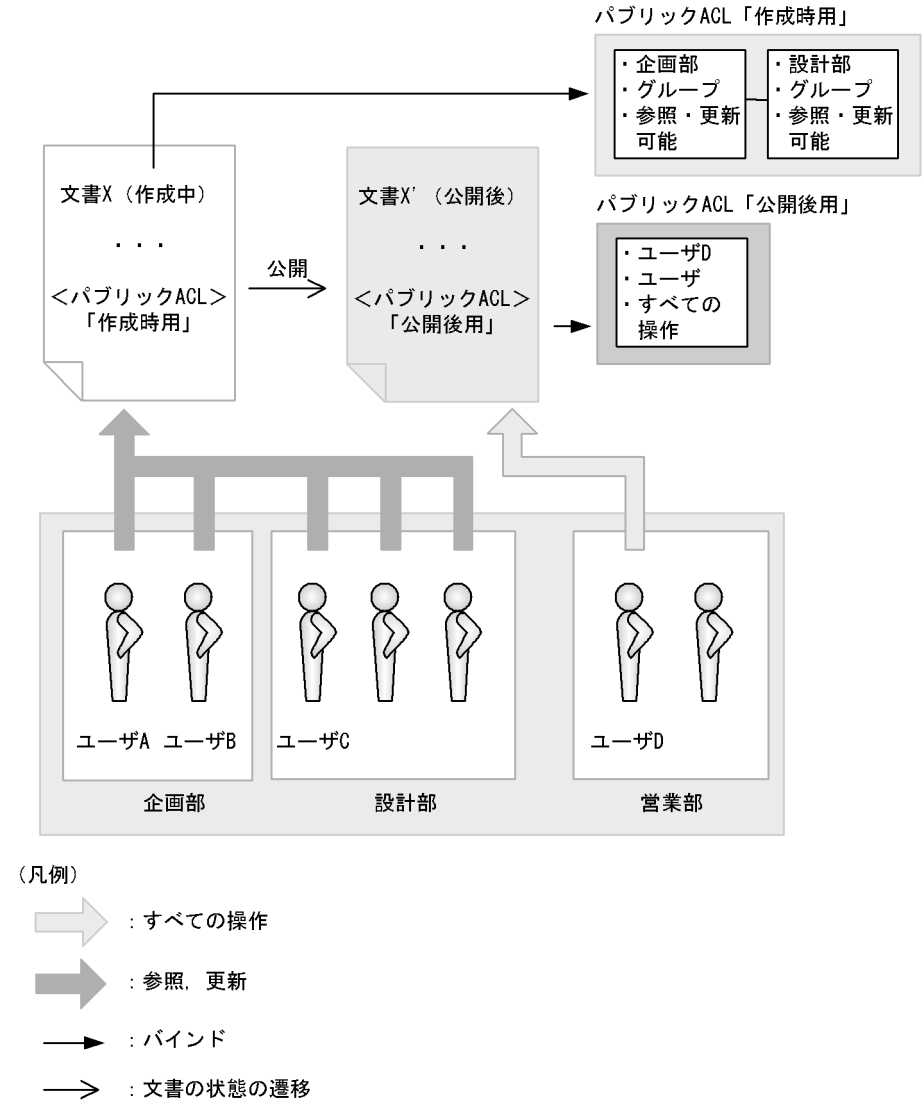
ユーザ権限定義ファイルの具体的な定義方法については，マニュアル「DocumentBroker Version 3 システム導入・運用ガイド」を参照してください。

2. **ACFlag の設定**
   アクセス制御機能に対応している文書空間でオブジェクトを作成すると，オブジェクトには「所有者」として，オブジェクトを作成したユーザが設定されます。また，オブジェクトには，プライマリグループが設定されている場合があります。プライマリグループは，連携しているユーザ管理システムによって，オブジェクト作成時に設定される場合と，ユーザが明示的に設定する場合があります。プライマリグループを設定する場合は，運用によって，オブジェクトを作成したユーザが所属するグループを設定したり，オブジェクトに対してアクセスが多いと考えられるユーザのグループを設定したりできます。
   図 3-78 の組織の場合，ユーザ A がオブジェクトを作成すると，オブジェクトの所有者としてユーザ A が設定されます。また，プライマリグループには，企画部を明示的に設定したとします。
   ACFlag では，この所有者およびプライマリグループに対して，許可する操作が設定できます。
   例えば，ユーザ A が作成した文書に対して，「所有者はすべての操作が可能で，プライマリグループは参照と編集，バージョンアップが可能」としたい場合，所有者に許可する操作として「すべての操作」を設定して，プライマリグループに対して許可する操作として「参照と更新およびバージョンアップ」と設定できます。
   ACFlag に設定したアクセス制御情報は，アクセス権判定時にパブリック ACL やローカル ACL よりも先に処理されます。このため，ACFlag によって所有者やプライマリグループに適切なアクセス権を設定しておけば，アクセス権判定処理が高速で実行されるため，オブジェクトへのアクセスを速くできます。
   また，ACFlag では，すべてのユーザに対して許可する操作も設定できます。例えば，「この文書はすべてのユーザに対して参照を許可する」などの設定もできます。

3. **パブリック ACL の設定**
   パブリック ACL には，複数の文書やコンテナに対して共通して設定したいアクセス制御情報を設定します。
   例えば，図 3-78 の組織で，「ユーザ A，ユーザ C が作成した文書を，グループ『企画部』およびグループ『設計部』内のすべてのユーザが参照，更新できるようにしたい」場合があるとします。このとき，この条件を設定したパブリック ACL を作成します。「グループ『企画部』に対して参照，更新する許可を与える」という ACE と「グループ『設計部』に対して参照，更新する許可を与える」という ACE を作成し，これらを構成要素とした ACL を，パブリック ACL に設定します。ユーザ A およびユーザ C が作成したすべての文書に対してこのパブリック ACL をバインドさせれば，すべての文書に同じアクセス制御情報が設定できます。
   なお，パブリック ACL は複数のオブジェクトで共有されますので，作成時に明確な運用方法を決めてからアクセス制御情報を設定してください。複数のオブジェクトからのバインドを設定したあとはできるだけアクセス制御情報を変更しない運用をお勧めします。
   例えば，「文書の作成時には『企画部』および『設計部』に参照，更新させたいが，一度公開したあとでは『営業部』内の特定のユーザ（ユーザ D）だけにすべての操作をする権利を与え，ほかのユーザには更新させないようにしたい」というような場合は，公開後にパブリック ACL の内容を変更するのではなく，作成時用のパブリック ACL と，公開後用のパブリック ACL を複数作成しておく運用が考え

られます。ある文書を公開したら，その文書からバインドするパブリック ACL を「作成時用」のものから「公開後用」のものにバインドし直すことで，アクセス制御情報は変更できます。パブリック ACL の運用例を，次の図に示します。

図 3-79　パブリック ACL の運用例



4. **ローカル ACL の設定**

ローカル ACL には，個々のオブジェクト固有のアクセス制御情報を設定します。
例えば，「ユーザ A が作成した文書 X は，企画部，設計部のユーザ以外に，営業部のユーザ D にも，参照，更新させたい」という場合，文書 X に ACFlag，パブリック ACL の設定のほかに「ユーザ D に参照，更新を許可する」というローカル ACL を設定します。

5. **セキュリティ ACL の設定**

セキュリティ ACL には，オブジェクト（パブリック ACL を含む）のアクセス制御情報を操作することを許可するユーザを設定します。これは，それぞれのオブジェクトごとに設定します。
例えば，「ユーザ A が作成した文書 X は，ユーザ B の承認を得たあとで公開する。ユーザ B が承認した段階で，バインドするパブリック ACL を『作成時用』から『公開後用』に変更したい」という場合，文書 X のセキュリティ ACL に，ユーザ B を設定しておきます。これによって，ユーザ B は承認

後，文書 X からバインドするパブリック ACL を変更できます。
ただし，文書の所有者は，常に所有するオブジェクトのすべてのアクセス制御情報変更権を持ちます。このため，「ユーザ B の承認後，ユーザ A からはアクセス制御情報変更権を無くしたい」という場合は，ユーザ A またはセキュリティ管理者によって，ほかのユーザ（例えばユーザ B）に所有者を変更する必要があります。
また，パブリック ACL に設定されているアクセス制御情報は，バインドしているオブジェクトのセキュリティ ACL を持つユーザでは変更できません。つまり，図 3-79 の場合，文書 X に対してアクセス制御情報変更権を持つユーザ A によって，文書 X からバインドするパブリック ACL「作成時用」に設定されている内容を変更することはできません。
パブリック ACL のアクセス制御情報は，それぞれのパブリック ACL の所有者およびセキュリティ ACL に設定されたユーザによって変更できます。

3.15.6 アクセス制御情報を表すプロパティの詳細

ここでは，アクセス制御情報を表すプロパティの詳細について説明します。

それぞれのアクセス制御情報は，オブジェクトのプロパティの値として設定されます。このため，アクセス制御情報を取得，設定および変更する場合は，プロパティを操作するメソッドを使用します。プロパティを操作するメソッドについては，「2.6.6　プロパティの操作」を参照してください。

アクセス制御情報は，サブジェクトを示すプロパティと，パーミッションを示すプロパティの組み合わせによって表されます。

(1) ACFlag で設定するサブジェクトとパーミッションの組み合わせ

ACFlag では，「所有者」「プライマリグループ」「すべてのユーザ」に対するパーミッションを設定します。所有者とは，dbrProp_OwnerId プロパティに設定されたユーザで，オブジェクト作成時にそのオブジェクトを作成したユーザが設定され，必要に応じて所有者自身によって別のユーザに変更できます。プライマリグループは，dbrProp_PrimaryGroupId プロパティに設定されたグループです。プライマリグループはオブジェクト作成時に設定される場合と，オブジェクト作成後にユーザが設定する場合があります。すべてのユーザとは，特定のプロパティとして設定されている値でなく，文書空間にログインできるすべてのユーザを示します。

「所有者」，「プライマリグループ」および「すべてのユーザ」を表すサブジェクトに当たるプロパティと，ACFlag でそれぞれのサブジェクトにパーミッションを設定するプロパティとの対応を，次の表に示します。これらのプロパティは，すべてクラスライブラリのオブジェクトのトップオブジェクトに設定されます。

表 3-20　ACFlag によって設定できるサブジェクトとパーミッションの組み合わせ

アクセス権が与えられる対象	サブジェクトを表すプロパティ識別子	サブジェクトに対するパーミッションを指定するプロパティ識別子（ACFlag）
所有者	dbrProp_OwnerId	dbrProp_OwnerPermission
プライマリグループ	dbrProp_PrimaryGroupId	dbrProp_PrimaryGroupPermission
すべてのユーザ	—	dbrProp_EveryonePermission

パーミッションに設定する値については，「3.15.4　パーミッションの種類」を参照してください。

(2) ACL で設定するサブジェクトとパーミッションの組み合わせ

ACL は，VariableArray 型プロパティです。個々の要素として，ACE を持ちます。ACE は，

CdbrCompound クラスのオブジェクトとして扱うことができます。

個々の ACE には，CdbrCompound クラスのオブジェクトのプロパティとして，「サブジェクト」，「サブジェクト種別」および「パーミッション」を指定します。

ACE に指定するサブジェクトとサブジェクト種別，およびパーミッションを表すプロパティの組み合わせを次の表に示します。なお，サブジェクト種別を表すプロパティとは，そのサブジェクトがユーザなのか，グループなのか，システムなのかを表すプロパティです。

表 3-21　ACE によって設定できるサブジェクトとパーミッションの組み合わせ

設定する内容	プロパティ識別子
サブジェクト	dbrProp_Subject
サブジェクト種別	dbrProp_SubjectType
サブジェクトに対するパーミッション	dbrProp_Permission

注意

一つの ACL に設定できる ACE の個数は 64 個までです。セキュリティ ACL として指定する場合も，一つのオブジェクトのセキュリティ ACL に設定できる ACE の個数は 64 個までです。

(3) 個々のオブジェクトに設定するアクセス制御情報を表すプロパティ

個々のオブジェクトに設定するアクセス制御情報を表すプロパティについて，次の表に示します。

表 3-22　個々のオブジェクトに設定するアクセス制御情報を表すプロパティ

アクセス制御情報の種類	プロパティ識別子	データ型	基本単位	制限値	検索の可否	説明
所有者	dbrProp_OwnerId	String 型	Scalar 型	1 ～ u バイト	○※1	所有者のユーザ識別子 u：ユーザ識別子の最大長※2
プライマリグループ	dbrProp_PrimaryGroupId	String 型	Scalar 型	0 ～ g バイト	○※3	プライマリグループのグループ識別子 g：グループ識別子の最大長※4
ACFlag	dbrProp_OwnerPermission	Integer32 型	Scalar 型	4 バイト	×	所有者のパーミッション
	dbrProp_PrimaryGroupPermission	Integer32 型	Scalar 型	4 バイト	×	プライマリグループのパーミッション
	dbrProp_EveryonePermission	Integer32 型	Scalar 型	4 バイト	×	すべてのユーザのパーミッション
ローカル ACL	dbrProp_ACL	Object 型	VariableArray 型	ACE は 64 個まで	×	ACL
セキュリティ ACL	dbrProp_SACL	Object 型	VariableArray 型	ACE は 64 個まで	×	セキュリティ ACL
パブリック ACL	dbrProp_PublicACLCount※5	Integer32 型	Scalar 型	10 個	×	バインドしているパブリック ACL の個数

アクセス制御情報の種類	プロパティ識別子	データ型	基本単位	制限値	検索の可否	説明
	dbrProp_PublicACLIds	Object 型	VariableArray 型	一つのオブジェクトにバインドできるパブリック ACL は 10 個まで。 個々のパブリック ACL で設定できる ACE は 64 個まで	×	バインドしているパブリック ACL の OIID のリスト（可変長配列）
ログインユーザのパーミッション	dbrProp_UserPermission ※5	Integer32 型	Scalar 型	―	×	ログインユーザに設定されているパーミッションの論理和

（凡例）
○：検索に使用できます。
×：検索に使用できません。
―：該当ありません。

注※ 1

検索する場合は，プロパティ識別子を「edmProp_OwnerId」と指定する必要があります。

注※ 2

通常，ユーザ識別子の最大長は 254 バイトですが，次に示すコマンドでユーザ識別子の最大長を拡張できます。

- メタ情報の初期設定コマンド（EDMInitMeta）
  この場合，-n オプションの指定値がユーザ識別子の最大長になります。
- 文書空間の定義コマンド（EDMCDefDocSpace）
  この場合，-s オプションに指定する文書空間情報ファイル中の UserIDMaxLength エントリの指定値がユーザ識別子の最大長になります。

メタ情報の初期設定コマンド（EDMInitMeta）および文書空間の定義コマンド（EDMCDefDocSpace）については，マニュアル「DocumentBroker Version 3　システム導入・運用ガイド」を参照してください。

注※ 3

検索する場合は，プロパティ識別子を「edmProp_PrimaryGroupId」と指定する必要があります。

注※ 4

通常，グループ識別子の最大長は 254 バイトですが，次に示すコマンドでグループ識別子の最大長を拡張できます。

- メタ情報の初期設定コマンド（EDMInitMeta）
  この場合，-g オプションの指定値がグループ識別子の最大長になります。
- 文書空間の定義コマンド（EDMCDefDocSpace）
  この場合，-s オプションに指定する文書空間情報ファイル中の GroupIDMaxLength エントリの指定値がグループ識別子の最大長になります。

メタ情報の初期設定コマンド（EDMInitMeta）および文書空間の定義コマンド（EDMCDefDocSpace）については，マニュアル「DocumentBroker Version 3　システム導入・運用

ガイド」を参照してください。

注※ 5
読み取り専用のプロパティです。

(a) 所有者（dbrProp_OwnerId）

所有者のユーザ識別子を表すプロパティです。所有者は，ACFlag によってパーミッションを与える対象になります。

このプロパティは検索に使用できます。ただし，検索に使用する場合は，プロパティ識別子を「edmProp_OwnerId」と指定します。

(b) プライマリグループ（dbrProp_PrimaryGroupId）

プライマリグループのグループ識別子を表すプロパティです。プライマリグループは，ACFlag によってパーミッションを与える対象になります。

このプロパティは検索に使用できます。ただし，検索に使用する場合は，プロパティ識別子を「edmProp_PrimaryGroupId」と指定します。

(c) ACFlag（dbrProp_OwnerPermission ／ dbrProp_PrimaryGroupPermission ／ dbrProp_EveryonePermission）

所有者，プライマリグループおよびすべてのユーザに対して与えるパーミッションを表すプロパティです。

パーミッションとして指定する値については，「3.15.4　パーミッションの種類」を参照してください。

(d) ローカル ACL（dbrProp_ACL）

ローカル ACL を表す VariableArray 型プロパティです。

ACL の要素である ACE のプロパティについては，「(4) アクセス制御情報の構成要素のプロパティ」を参照してください。

(e) セキュリティ ACL（dbrProp_SACL）

ローカル ACL を表す VariableArray 型プロパティです。

ACL の要素である ACE のプロパティについては，「(4) アクセス制御情報の構成要素のプロパティ」を参照してください。

(f) パブリック ACL（dbrProp_PublicACLCount ／ dbrProp_PublicACLIds）

バインドしているパブリック ACL の個数と，パブリック ACL のリストを表すプロパティです。

dbrProp_PublicACLCount プロパティは，バインドしているパブリック ACL の個数を表すプロパティです。一つのオブジェクトには 10 個のパブリック ACL をバインドできます。なお，このプロパティは読み取り専用です。

dbrProp_PublicACLIds プロパティは，バインドしているパブリック ACL の OIID のリストを表す VariableArray 型プロパティです。

個々のパブリック ACL の OIID は，VariableArray 型プロパティの要素を表す CdbrCompound オブジェクトの dbrProp_ACLIdElem プロパティとして設定します。

dbrProp_ACLIdsElem プロパティの詳細を表 3 35 に示します。

表 3-23 dbrProp_ACLIdsElem プロパティの詳細

プロパティ識別子	データ型	基本単位	制限値	検索の可否	説明
dbrProp_ACLIdElem	String 型	Scalar 型	OIID 長	×	バインドしているパブリック ACL の OIID

（凡例）
×：検索に使用できません。

パブリック ACL を新規にバインドする場合，またはすべて更新する場合は，まず，この CdbrCompound オブジェクトを要素とした VariableArray 型プロパティを作成して，PutPropertyValues メソッドによって，オブジェクトに設定します。

また，すでにパブリック ACL をバインドしているオブジェクトに対してバインドするパブリック ACL を追加する場合は，BindPublicACL メソッドを使用して，パブリック ACL を追加します。

パブリック ACL の設定方法の詳細については，「3.15.9(5) 文書またはコンテナへのパブリック ACL の設定」および「3.15.9(6) 文書またはコンテナへのパブリック ACL のバインド」を参照してください。

（g）ログインユーザのパーミッション（dbrProp_UserPermission）

ログインユーザがユーザ権限定義ファイル，ACFlag，ローカル ACL およびバインドしているパブリック ACL で与えられているパーミッションの論理和を表すプロパティです。なお，このプロパティは読み取り専用です。

（4） アクセス制御情報の構成要素のプロパティ

ここでは，アクセス制御情報のうち，ACL の構成要素である ACE のプロパティについて説明します。

ACE のプロパティを，次の表に示します。

表 3-24 ACE のプロパティ

データ名称	プロパティ識別子	データ型	基本単位	制限値	検索の可否	説明
サブジェクト	dbrProp_Subject	String 型	Scalar 型	s バイト	×	サブジェクトの識別子 s：サブジェクトの最大長※
サブジェクト種別	dbrProp_SubjectType	Integer32 型	Scalar 型	4 バイト	×	サブジェクト種別を表す定数
パーミッション	dbrProp_Permission	Integer32 型	Scalar 型	4 バイト	×	パーミッションを表す定数

（凡例）
×：検索に使用できません。

注※
通常，ユーザ識別子およびグループ識別子の最大長は 254 バイトですが，次に示すコマンドでユーザ識別子とグループ識別子の最大長を拡張できます。
- メタ情報の初期設定コマンド（EDMInitMeta）
  この場合，-n オプションの指定値がユーザ識別子の最大長に，-g オプションの指定値がグループ識別子の最大長になります。
- 文書空間の定義コマンド（EDMCDefDocSpace）

この場合，-s オプションに指定する文書空間情報ファイル中の UserIDMaxLength エントリの指定値がユーザ識別子の最大長に，GroupIDMaxLength エントリの指定値がグループ識別子の最大長になります。

メタ情報の初期設定コマンド（EDMInitMeta）および文書空間の定義コマンド（EDMCDefDocSpace）については，マニュアル「DocumentBroker Version 3　システム導入・運用ガイド」を参照してください。
なお，システムサブジェクトの場合，識別子の最大長は常に 254 バイトになります。

(a) サブジェクト（dbrProp_Subject）

サブジェクトには，次のどれかを指定します。

● ユーザ識別子
ユーザ単位でパーミッションを設定する場合に指定します。

● グループ識別子
グループ単位でパーミッションを設定する場合に指定します。

● システムサブジェクトを表す定数
システムサブジェクトに対してパーミッションを設定する場合に指定します。
システムサブジェクトは，DocumentBroker によって設定された情報に対して，パーミッションを設定する場合に指定するサブジェクトです。
このサブジェクトは，主にパブリック ACL に指定して使用します。
システムサブジェクトの定数と意味は，次のとおりです。

表 3-25　システムサブジェクトの定数と意味

定数	意味
DBR_SYSSUBJECT_SELF	対象オブジェクトの所有者を表すサブジェクトです。 • パブリック ACL の ACL に指定した場合，そのパブリック ACL をバインドしている文書やコンテナを表すオブジェクトの所有者を示します。 • 文書やコンテナを表すオブジェクトのローカル ACL およびセキュリティ ACL に指定した場合は，そのオブジェクトの所有者を示します。 • パブリック ACL のセキュリティ ACL に指定した場合は，パブリック ACL の所有者を示します。
DBR_SYSSUBJECT_EVERYONE	すべてのユーザを表すサブジェクトです。

(b) サブジェクト種別（dbrProp_SubjectType）

サブジェクト種別には，サブジェクトに指定した内容に応じて，種別を示す定数を指定します。

● ユーザ識別子を指定した場合
定数「DBR_SUBJECTTYPE_USR」を指定します。

● グループ識別子を指定した場合
定数「DBR_SUBJECTTYPE_GRP」を指定します。

● システムサブジェクトを表す定数を指定した場合
定数「DBR_SUBJECTTYPE_SYS」を指定します。

(c) パーミッション（dbrProp_Permission）

サブジェクトに指定したユーザまたはグループに対して設定するパーミッションを表す定数を指定します。

パーミッションとして指定する値については，「3.15.4　パーミッションの種類」を参照してください。

(5) パブリック ACL のプロパティ

パブリック ACL は，文書空間に独立して存在するオブジェクトです。

パブリック ACL に設定するプロパティについて，次の表に示します。

表 3-26 パブリック ACL に設定するプロパティ

プロパティ識別子	データ型	基本単位	制限値	検索の可否	説明
dmaProp_OIID	Object 型	Scalar 型	OIID 長	○	パブリック ACL を識別する OIID
dbrProp_OwnerId	String 型	Scalar 型	1 ～ u バイト	○	パブリック ACL 所有者のユーザ識別子 u：ユーザ識別子の最大長※1
dbrProp_ACL	Object 型	VariableArray 型	ACE は 64 個まで	×	パブリック ACL のローカル ACL。 ほかのオブジェクトによって共有される ACL を指定する
dbrProp_SACL	Object 型	VariableArray 型	ACE は 64 個まで	×	パブリック ACL のセキュリティ ACL
dbrProp_BindObjectCount	Integer32 型	Scalar 型	－	×	バインドしているオブジェクトの数※2
dbrProp_UserPermission	Integer32 型	Scalar 型	－	×	ログインユーザに設定されているパーミッションの論理和※3
（ユーザ定義プロパティの識別子）	－	－	－	○	ユーザが任意に追加したプロパティ

（凡例）
○：検索に使用できます。
×：検索に使用できません。
－：該当ありません。

注※ 1

通常，ユーザ識別子の最大長は 254 バイトですが，次に示すコマンドでユーザ識別子の最大長を拡張できます。

- メタ情報の初期設定コマンド（EDMInitMeta）
  この場合，-n オプションの指定値がユーザ識別子の最大長になります。
- 文書空間の定義コマンド（EDMCDefDocSpace）
  この場合，-s オプションに指定する文書空間情報ファイル中の UserIDMaxLength エントリの指定値がユーザ識別子の最大長になります。

メタ情報の初期設定コマンド（EDMInitMeta）および文書空間の定義コマンド（EDMCDefDocSpace）については，マニュアル「DocumentBroker Version 3　システム導入・運用ガイド」を参照してください。

注※ 2

読み取り専用のプロパティです。

注※ 3

ユーザ権限定義ファイルおよびセキュリティ ACL によって与えられているパーミッションの論理和

です。

なお，パブリック ACL のローカル ACL には，複数のオブジェクト間で共有したいアクセス制御情報を設定します。パブリック ACL に対するアクセス制御情報を設定するものではありません。

パブリック ACL 自身に対するアクセス権については，「3.15.3(2)(a) パブリック ACL」を参照してください。

3.15.7 オブジェクトの操作とアクセス制御情報の関係

ここでは，オブジェクトの操作とアクセス制御情報の関係について説明します。

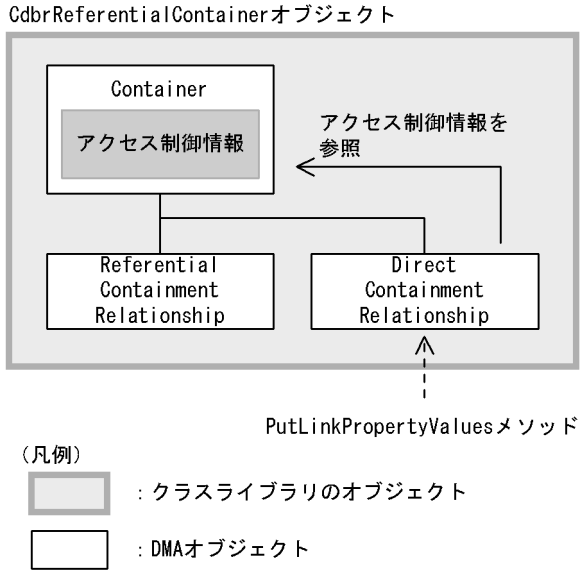
アクセス制御情報は，クラスライブラリのオブジェクトのトップオブジェクトのプロパティとして設定されています。トップオブジェクト以外のオブジェクトに対する操作は，トップオブジェクトに設定されたアクセス制御情報に従って処理されます。

例として，CdbrReferentialContainer オブジェクトに対する操作について説明します。

CdbrReferentialContainer オブジェクトは，DMA オブジェクトの Container オブジェクトをトップオブジェクトとして，DirectContainmentRelationship オブジェクト，ReferentialContainmentRelationship オブジェクトとともに構成されています。したがって，CdbrReferentialContainer オブジェクトのアクセス制御情報は，DMA オブジェクトの Container オブジェクトのプロパティとして設定されています。

ここで，CdbrReferentialContainer オブジェクトで設定されているリンクに対して，プロパティを設定する場合について説明します。リンクのプロパティとは，DMA オブジェクトの DirectContainmentRelationship オブジェクトまたは ReferentialContainmentRelationship オブジェクトのプロパティです。PutLinkPropertyValues メソッドによってこのプロパティを更新しようとすると，DocumentBroker はアクセス制御情報としてトップオブジェクトに設定されているアクセス制御情報を参照してアクセス権を判定し，アクセス権がある場合にメソッドを実行します。トップオブジェクト以外のオブジェクトにアクセスする場合の，アクセス制御の例を，次の図に示します。

図 3-80 トップオブジェクト以外のオブジェクトにアクセスする場合のアクセス制御の例（CdbrReferentialContainer オブジェクトの例）



また，バージョン管理を実行したり，オブジェクト間のコンテインメントによる関連づけを実行したりす

る場合，複数のオブジェクトに対してパーミッションが必要なときがあります。

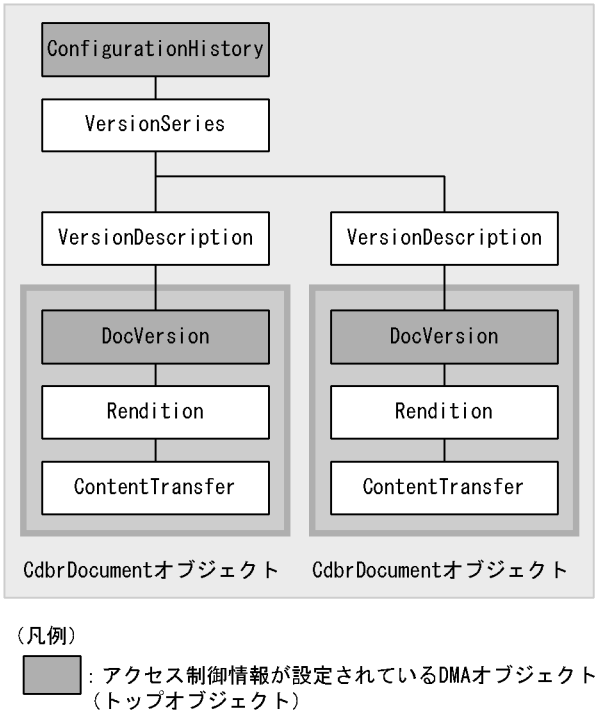
ここでは，複数のオブジェクトに対してパーミッションが必要な操作について説明します。

(1) バージョン管理に関する操作

バージョン管理をしている文書の場合，一つのバージョン付き文書は，複数のバージョンなし文書によって構成されています。また，バージョン付き構成管理コンテナの場合も同様に，複数のバージョンなし構成管理コンテナによって構成されています。この場合，一つのバージョン付き文書を操作する場合に，バージョン付き文書のトップオブジェクトに設定されたパーミッションと，バージョンなし文書のトップオブジェクトに設定されたパーミッションが必要な場合があります。

バージョン付きオブジェクトとバージョンなしオブジェクトに設定されているアクセス制御情報について，次の図に示します。

図 3-81 バージョン付きオブジェクトとバージョンなしオブジェクトに設定されているアクセス制御情報（CdbrVersionableDocument オブジェクトおよび CdbrDocument オブジェクトの場合）



ここでは，バージョン付き文書およびバージョン付き構成管理コンテナをバージョン付きオブジェクト，バージョンなし文書およびバージョンなし構成管理コンテナをバージョンなしオブジェクトとして，これら二つのオブジェクトにパーミッションが必要な操作について説明します。バージョン付きオブジェクトと 1 バージョンに相当するバージョンなしオブジェクトの対応について，次の表に示します。

表 3-27 バージョン付きオブジェクトとバージョンなしオブジェクトの対応

バージョン付きオブジェクト	1 バージョンに相当するバージョンなしオブジェクト
CdbrVersionableDocument オブジェクト	CdbrDocument オブジェクト
CdbrConfiguratedReferentialContainer オブジェクト	CdbrVersionTraceableContainer オブジェクト

例として，CdbrVersionableDocument オブジェクトと CdbrDocument オブジェクトについて説明します。

CdbrDocument オブジェクトは，CdbrVersionableDocument オブジェクトの一つのバージョンに相当します。ユーザ A が CdbrVersionableDocument オブジェクトの 1 バージョンを削除しようとした場合，まず，CdbrVersionableDocument オブジェクトに対して基本バージョン管理権が必要です。しかし，バージョンを削除することは実際には一つの CdbrDocument オブジェクトを削除することと同じことになります。したがって，この場合は，CdbrVersionableDocument オブジェクトの基本バージョン管理権のほかに，CdbrDocument オブジェクトの基本オブジェクト削除権が必要になります。

● バージョン付きオブジェクトを操作する場合に必要なパーミッション

バージョン付きオブジェクトに対してバージョン管理に関する操作をする時，操作の種類によってはバージョンなしオブジェクトに対してもアクセス権が必要な場合があります。
メソッドをコールするユーザがバージョン付きオブジェクトに対して必要なパーミッションとバージョンなしオブジェクトに対して必要なパーミッションを次の表に示します。

表 3-28　バージョン付きオブジェクトとバージョンなしオブジェクトに必要なパーミッション（バージョン付きオブジェクトを操作する場合）

バージョン管理に関係する操作（メソッド）	バージョン付きオブジェクトに必要なパーミッション	バージョンなしオブジェクトに必要なパーミッション
バージョンチェックアウト • VersionCheckOut	基本バージョン管理権 （PRIM_VERSION）	• CdbrDocument オブジェクトの場合： 基本コンテント参照権 （PRIM_READ_CONTENTS ※ 1） • CdbrVersionTraceableContainer オブジェクトの場合： 基本プロパティ参照権 （PRIM_READ_PROPS ※ 1）
バージョンチェックイン • VersionCheckIn	基本バージョン管理権 （PRIM_VERSION）	―
バージョンチェックアウトのキャンセル • VersionRevoke	基本バージョン管理権 （PRIM_VERSION）	―
バージョンの削除 • DeleteVersion	基本バージョン管理権 （PRIM_VERSION）	基本オブジェクト削除権 （PRIM_DELETE）
バージョン情報の一覧取得 • GetVersionList • GetVersionListAndLock	基本プロパティ参照権 （PRIM_READ_PROPS）	基本プロパティ参照権※ 2 （PRIM_READ_PROPS）
バージョン識別子を指定したバージョンなしオブジェクトの操作	基本プロパティ参照権 （PRIM_READ_PROPS）	バージョンなしオブジェクトに直接メソッドをコールした場合に必要なパーミッション

（凡例）
　―：パーミッションは必要ありません。

注※ 1
　カレントバージョンに対応するバージョンなしオブジェクトの場合に必要です。

注※ 2
　バージョンなしオブジェクトのプロパティを取得しない場合は不要です。また，バージョンなしオブジェクトのプロパティを取得する場合に，基本プロパティ参照権が設定されていないオブジェクトについては，一覧として取得されません。

● バージョンなしオブジェクトを操作する場合に必要なパーミッション

バージョン付きオブジェクトの 1 バージョンに相当するバージョンなしオブジェクトを操作する時，操作の種類によってはバージョン付きオブジェクトに対してもアクセス権が必要な場合があります。
メソッドをコールするユーザがバージョン付きオブジェクトに対して必要なパーミッションとバージョンなしオブジェクトに対して必要なパーミッションを次の表に示します。

表 3-29 バージョン付きオブジェクトとバージョンなしオブジェクトに必要なパーミッション（バージョンなしオブジェクトを操作する場合）

バージョンなしオブジェクトに対する操作（メソッド）	バージョン付きオブジェクトに必要なパーミッション	バージョンなしオブジェクトに必要なパーミッション
オブジェクトの削除 • RemoveObject	基本バージョン管理権 （PRIM_VERSION）	基本オブジェクト削除権 （PRIM_DELETE）
自オブジェクトを 1 バージョンとして包含しているバージョン付きオブジェクトの一覧取得 • GetVersionableList • GetVersionableListAndLock	基本プロパティ参照権※ （PRIM_READ_PROPS）	基本プロパティ参照権 （PRIM_READ_PROPS）

注※

バージョン付きオブジェクトのプロパティを取得しない場合は不要です。また，バージョン付きオブジェクトのプロパティを取得する場合に，基本プロパティ参照権が設定されていないオブジェクトについては，一覧として取得されません。

バージョンなしオブジェクトに対して表 3-29 に示した以外のメソッドをコールする場合，バージョン付きオブジェクトのパーミッションは必要ありません。

● チェックアウトによってコピーされたオブジェクトのパーミッション

VersionCheckOut メソッドをコールすると，カレントバージョンに該当するバージョンなしオブジェクトがコピーされて，仮のバージョンに該当するバージョンなしオブジェクトが作成されます。
このとき，仮のバージョンに該当するバージョンなしオブジェクトの所有者およびプライマリグループには，コピー元のオブジェクトに設定されていた所有者およびプライマリグループが設定されます。
また，仮のバージョンに該当するバージョンなしオブジェクトには，コピー元のオブジェクトに設定されていた ACFlag およびローカル ACL が設定され，チェックアウトしたオブジェクトからバインドしていたパブリック ACL がバインドされます。

(2) 文書間リレーションに関する操作

文書間リレーションを設定している文書間では，メソッドによってはリレーション元文書とリレーション先文書に対して適切なパーミッションが与えられている必要があります。

メソッドをコールするユーザがリレーション元文書とリレーション先文書に対して必要なパーミッションを次の表に示します。

表 3-30 リレーション元文書とリレーション先文書に対して必要なパーミッション

リレーションに関係する操作（メソッド）	リレーション元文書に必要なパーミッション	リレーション先文書に必要なパーミッション
リレーションの作成 • CreateRelation	基本リンク権 （PRIM_LINK）	基本プロパティ参照権※ （PRIM_READ_PROPS）
リレーション情報の取得 • GetRelationList • GetRelationListAndLock	基本プロパティ参照権※ （PRIM_READ_PROPS）	基本プロパティ参照権※ （PRIM_READ_PROPS）
リレーションの削除 • RemoveRelation	基本リンク権 （PRIM_LINK）	−
リレーションのプロパティの設定 • PutRelationPropertyValues	基本リンク権 （PRIM_LINK）	−

（凡例）

－：パーミッションは必要ありません。

注※

リレーション先文書のプロパティを取得しない場合は不要です。

(3) コンテナへの関連づけに関する操作

直接型や参照型のコンテインメントによってコンテナと関連づけている文書やコンテナでは，コンテナを表すオブジェクトと関連づけられるコンテナまたは文書を表す二つのオブジェクトに対して，適切なパーミッションを与えられている必要があります。

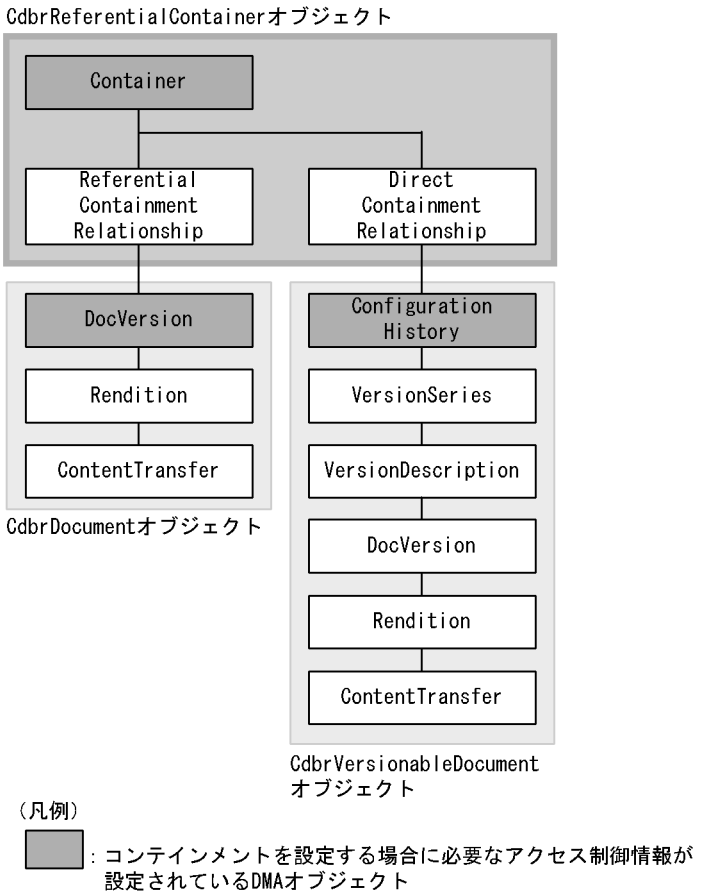
コンテナを表すオブジェクトとコンテナに関連づけられるオブジェクトの対応について，次の表に示します。

表 3-31 コンテナとコンテナに関連づけられているオブジェクトの対応

コンテナになるオブジェクト	コンテナに関連づけられるオブジェクト
CdbrReferentialContainer オブジェクト	CdbrContainable クラスのサブクラスのオブジェクト
CdbrConfiguratedReferentialContainer オブジェクト（1 バージョン）	CdbrContainable クラスのサブクラスのオブジェクト
CdbrVersionTraceableContainer オブジェクト	CdbrContainable クラスのサブクラスのオブジェクト

例えば，CdbrReferentialContainer オブジェクトに対して，CdbrDocument オブジェクトおよび CdbrVersionableDocument オブジェクトとのコンテインメントを設定する場合に必要なアクセス制御情報が設定されている DMA オブジェクトは，次の図のようになります。

図 3-82　コンテナとコンテナに関連づけられているオブジェクトに設定されているアクセス制御情報



● コンテナを操作する場合に必要なパーミッション

コンテナを操作する時，操作の種類によってはコンテナに関連づけられている文書またはコンテナに対してもアクセス権が必要な場合があります。

メソッドをコールするユーザがコンテナに対して必要なパーミッションとコンテナに関連づけられる文書またはコンテナに対して必要なパーミッションを次の表に示します。

表 3-32　コンテナとコンテナと関連づけられている文書またはコンテナに対して必要なパーミッション（コンテナを操作する場合）

リンクに関係する操作（メソッド）	コンテナに必要なパーミッション	コンテナに関連づけられるオブジェクトに必要なパーミッション
コンテナとオブジェクトの関連づけ • Link • LinkAndLock	基本リンク権 （PRIM_LINK）	基本プロパティ参照権 （PRIM_READ_PROPS）
コンテナとオブジェクトの関連づけの解除 • Unlink • UnlinkAndLock	基本リンク権 （PRIM_LINK）	－
リンクのプロパティの参照 • GetLinkPropertyValues • GetLinkPropertyValuesAndLock	基本プロパティ参照権 （PRIM_READ_PROPS）	－

リンクに関係する操作（メソッド）	コンテナに必要なパーミッション	コンテナに関連づけられるオブジェクトに必要なパーミッション
リンクのプロパティの更新 • PutLinkPropertyValues	基本リンク権 （PRIM_LINK）	―
コンテナと関連づけている要素の一覧取得 • GetContainableList • GetContainableListAndLock	基本プロパティ参照権 （PRIM_READ_PROPS）	基本プロパティ参照権※ （PRIM_READ_PROPS）

（凡例）

―：パーミッションは必要ありません。

注※

コンテナと関連づけられるオブジェクトのプロパティを取得しない場合は不要です。また，コンテナと関連づけられるオブジェクトのプロパティを取得する場合に，基本プロパティ参照権が設定されていないオブジェクトについては，一覧として取得されません。

● コンテナと関連づけられるオブジェクトを操作する場合に必要なパーミッション

コンテナと関連づけられている文書またはコンテナを操作する時，操作の種類によってはコンテナに対してもアクセス権が必要な場合があります。

メソッドをコールするユーザがコンテナに対して必要なパーミッションとコンテナと関連づけられる文書またはコンテナに対して必要なパーミッションを次の表に示します。

表 3-33　コンテナとコンテナに関連づけられている文書またはコンテナに対して必要なパーミッション（コンテナと関連づけられるオブジェクトを操作する場合）

コンテナに関連づけられているオブジェクトに対する操作（メソッド）	コンテナに必要なパーミッション	コンテナに関連づけられている文書またはコンテナに必要なパーミッション
自オブジェクトを包含要素として関連づけているコンテナの一覧取得 • GetContainerList • GetContainerListAndLock	基本プロパティ参照権※ （PRIM_READ_PROPS）	基本プロパティ参照権 （PRIM_READ_PROPS）

注※

コンテナであるオブジェクトのプロパティを取得しない場合は不要です。また，コンテナであるオブジェクトのプロパティを取得する場合に，基本プロパティ参照権が設定されていないオブジェクトについては，一覧として取得されません。

コンテナに関連づけられているオブジェクトに対して表 3-33 に示した以外のメソッドがコールされた場合，コンテナであるオブジェクトのパーミッションは必要ありません。

（4） 構成管理に関する操作

構成管理コンテナ（CdbrConfiguratedReferentialContainer オブジェクトまたは CdbrVersionTraceableContainer オブジェクト）によって構成管理機能を使用している場合は，構成管理コンテナと構成管理されているオブジェクトの二つのオブジェクトに対して，適切なパーミッションが必要です。

バージョン付き構成管理コンテナまたはバージョンなし構成管理コンテナを表すオブジェクトに構成管理型のコンテインメントで関連づけられるオブジェクトの対応について，次の表に示します。

表 3-34　構成管理コンテナと構成管理型で関連づけられるオブジェクトの対応

構成管理コンテナになるオブジェクト	構成管理コンテナに関連づけられるオブジェクト
CdbrConfiguratedReferentialContainer オブジェクト	CdbrVersionable クラスのサブクラスのオブジェクト

構成管理コンテナになるオブジェクト	構成管理コンテナに関連づけられるオブジェクト
CdbrVersionTraceableContainer オブジェクト	CdbrVersionable クラスのサブクラスのオブジェクト

● 構成管理コンテナを操作する場合に必要なパーミッション

構成管理コンテナを操作する時，操作の種類によっては，構成管理コンテナに構成管理型のコンテインメントで関連づけられる文書またはコンテナに対してもアクセス権が必要な場合があります。
メソッドをコールするユーザが，構成管理コンテナに対して必要なパーミッションと，構成管理コンテナに構成管理型のコンテインメントで関連づけられる文書またはコンテナに対して必要なパーミッションを，次の表に示します。

表 3-35 構成管理コンテナと構成管理型のコンテインメントで関連づけられる文書またはコンテナに必要なパーミッション

構成管理型のリンクに関係する操作（メソッド）	構成管理コンテナに必要なパーミッション	構成管理型のコンテインメントで関連づけられる文書またはコンテナに必要なパーミッション
構成管理コンテナとオブジェクトの構成管理型での関連づけ • LinkVTFix • LinkVTFixAndLock • LinkVTFloat • LinkVTFloatAndLock	基本リンク権（PRIM_LINK）	基本プロパティ参照権（PRIM_READ_PROPS ※）
構成管理コンテナとオブジェクトの構成管理型での関連づけの解除 • UnlinkVT • UnlinkVTAndLock	基本リンク権（PRIM_LINK）	－
構成管理モードの FIX モードへの変更 • SetVTFix	基本リンク権（PRIM_LINK）	－
構成管理モードの FLOATING モードへの変更 • SetVTFloat	基本リンク権（PRIM_LINK）	－

（凡例）

－：パーミッションは必要ありません。

注※

バージョン付きオブジェクトと，FIX モードまたは FLOATING モードで関連づける 1 バージョンに対応するバージョンなしオブジェクトの基本プロパティ参照権が必要です（例えば，CdbrVersionableDocument オブジェクトを構成管理型で関連づける場合，CdbrVersionableDocument オブジェクトの基本プロパティ参照権と，FIX モードまたは FLOATING モードで関連づけるバージョンに対応する CdbrDocument オブジェクトの基本プロパティ参照権が必要です）。

なお，FLOATING モードによって構成管理している場合，構成管理型で関連づけているオブジェクトがバージョンアップすると，関連づけているオブジェクトが変更されますが，表 3-35 のパーミッションは，メソッドによって関連づけを作成，変更した場合にだけチェックされます。バージョンアップ時にはチェックされません。

(5) 一覧取得メソッドを使用する操作

一覧取得メソッドとは，Get ～ List メソッドまたは Get ～ ListAndLock メソッドなどのメソッドです。クラスライブラリのオブジェクトがほかのクラスライブラリのオブジェクトとコンテインメントやバージョン管理機能によって関連づけられている場合に，その関連づけられているオブジェクトの一覧を取得

します。

一覧取得メソッドには，メソッドで設定されたデフォルトの情報だけを取得する以外に，明示的に指定した接続先オブジェクトのプロパティが取得できるメソッドがあります。これらのメソッドで，接続先のオブジェクトのプロパティを取得する場合は，接続先オブジェクトに基本プロパティ参照権が必要です。このパーミッションが設定されていない場合は，一覧に取得されません。なお，デフォルトの情報だけを取得する場合は，接続先のオブジェクトにパーミッションは必要ありません。これは，ユーザプログラムの内部処理に使うことを想定しています。システムを使用するエンドユーザに対してプロパティを表示して文書一覧を表示する場合などには，接続先のオブジェクトのプロパティを取得するメソッドを使用して，アクセス権があるものだけを表示するといった運用方法をお勧めします。

また，GetPropertyValues メソッドを使用して，該当するプロパティ識別子を指定した場合も，接続先オブジェクトの個数は取得できます。ただし，このメソッドでは，接続先オブジェクトに対するパーミッションに関係なく，接続されているすべてのオブジェクトの個数が取得されます。

(6) マルチレンディション管理に関する操作

マルチレンディション管理をしている文書の場合，レンディションの操作は，そのレンディションを含む CdbrDocument オブジェクトのトップオブジェクトに当たる，DMA オブジェクトの DocVersion オブジェクトに対して設定されているアクセス権に従って制御されます。

(7) 各メソッドの実行に必要なパーミッション

アクセス制御機能を使用している文書空間で，メソッドをコールするユーザはそのメソッドのコールに必要なパーミッションを持っている必要があります。

各メソッドのコールに必要なパーミッションについては，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。

3.15.8 アクセス制御機能と検索

ここでは，アクセス制御機能を使用した検索について説明します。なお，検索の詳細については，「4. オブジェクトの検索」を参照してください。

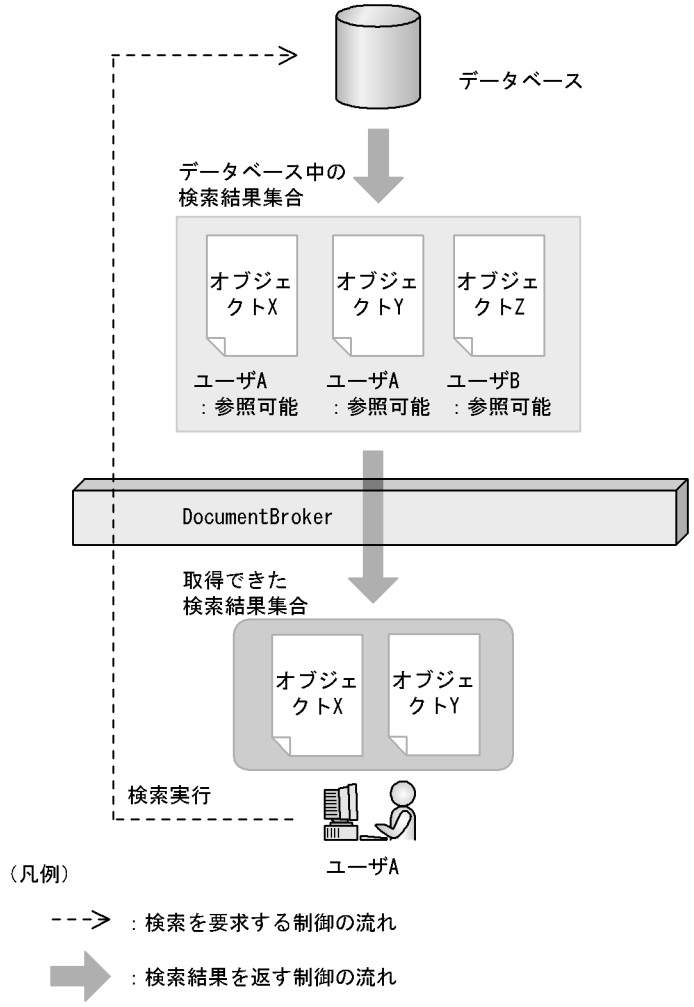
(1) アクセス制御機能付き検索とは

アクセス制御機能を使用している文書空間では，検索を実行したときに，ユーザが指定した問い合わせの条件を満たすオブジェクトの集合のうち，そのユーザが参照を許可されているオブジェクトだけを検索結果として取得させることができます。

このように，アクセス権を参照して，参照が許可されているオブジェクトだけを検索結果として取得する検索を，**アクセス制御機能付き検索**といいます。

アクセス制御機能付き検索の概要を次の図に示します。

図 3-83　アクセス制御機能付き検索の概要



ユーザ A は，オブジェクト X とオブジェクト Y について，参照権を持っていますが，オブジェクト Z については参照権を持っていません。このような場合に，ユーザ A がアクセス制御機能付き検索を実行すると，検索結果集合のうちオブジェクト X とオブジェクト Y だけが検索結果として返却されます。オブジェクト Z については，参照権がないため，ユーザ A は検索結果としてオブジェクト Z を取得できません。

DocumentBroker では，まずデータベースから検索条件を満たすオブジェクトの集合を取得します。そして，それぞれのオブジェクトに設定されているアクセス制御情報のパーミッションを取得します。これを検索を実行したユーザのユーザ情報と比較して，ユーザがアクセス権を持つオブジェクトだけを検索結果として返却します。

(2) 検索に必要なパーミッション

検索を実行して検索結果としてオブジェクトを取得するためには，そのオブジェクトに対してユーザは基本プロパティ参照権（PRIM_READ_PROPS）を与えられている必要があります。

ただし，全文検索を実行する場合，検索実行時にオブジェクトのコンテントを参照する必要があるため，基本コンテント参照権（PRIM_READ_CONTENTS）も必要になります。

(3) 検索実行時のアクセス制御モードの設定

アクセス制御機能付き検索は，アクセス権判定をする処理が増えるため，アクセス制御機能を使用しない

場合の検索に比べて処理が遅くなります。

このため，DocumentBroker では，アクセス制御機能付き検索を実行するかどうかが選択できます。なお，検索実行時にアクセス権を参照しないで検索結果をユーザに返却する検索を，**アクセス制御機能なし検索**といいます。

アクセス制御機能付き検索を実行するか，アクセス制御機能なし検索を実行するかは，**アクセス制御モード**によって設定します。

アクセス制御モードには，次の 2 種類があります。

● **アクセス制御機能付き検索モード**（DBR_QUERY_WITH_ACL）
アクセス制御機能付き検索を実行するモードです。

● **アクセス制御機能なし検索モード**（DBR_QUERY_WITHOUT_ACL）
アクセス制御機能なし検索を実行するモードです。

通常，アクセス制御モードは文書空間の設定に従って設定されており，アクセス制御機能を使用している文書空間では，デフォルトの設定としてアクセス制御機能付き検索モードが設定されています。変更する場合は，CdbrEqlStatement::ChangeACLMode メソッドで，アクセス制御機能なし検索モードに変更してください。

アクセス制御機能なし検索モードで検索を実行した場合，ユーザはパーミッションに関係なくオブジェクトを参照できます。したがって，文書空間の設計や文書管理の運用方法を決定する時に，セキュリティについて考慮しておくことが必要です。

なお，アクセス制御機能を使用していない文書空間では，アクセス制御機能付き検索モードを設定しても，アクセス制御機能なし検索モードに強制的に切り替わります。

（4） アクセス制御機能付き検索を実行する場合の制限事項

アクセス制御機能付き検索を実行する場合の制限事項については，「4.8.4　アクセス制御機能付き検索を実行する場合の制限事項」を参照してください。

3.15.9　アクセス制御機能に関する操作

アクセス制御機能を使用するため，アクセス制御情報を操作する場合に使用するメソッドと，そのメソッドの発行順序の例を説明します。

ここでは，次の操作について説明します。

- ACFlag の設定
- ACE，ACL の作成と設定
- 文書またはコンテナへの ACL の設定
- パブリック ACL の作成
- 文書またはコンテナへのパブリック ACL の設定
- 文書またはコンテナへのパブリック ACL のバインド

なお，これらの操作を実行する前に，まず，文書空間と接続してトランザクションを開始してください。

それぞれの操作に使用するメソッドの詳細については，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。

（1） ACFlag の設定

ACFlag は，クラスライブラリのオブジェクトのプロパティである，dbrProp_OwnerPermission プロパ

ティ，dbrProp_PrimaryGroupPermission プロパティおよび dbrProp_EveryonePermission プロパティとして設定します。これらの設定方法は，ほかのプロパティの設定方法と同じです。プロパティの設定については，「2.6.6　プロパティの操作」を参照してください。

ただし，ACFlag が設定できるのは，設定しようとしているクラスライブラリのオブジェクトの所有者，セキュリティ ACL に登録されているユーザまたはセキュリティ管理者だけです。

（2）ACE，ACL の作成と設定

ACE は，可変長配列の個々の要素を表す CdbrCompound クラスのオブジェクトのプロパティとして設定します。それぞれ，サブジェクトを dbrProp_Subject プロパティに，サブジェクト種別を dbrProp_SubjectType プロパティに，パーミッションを dbrProp_Permission プロパティに設定します。

ACL は可変長配列を表す VariableArray 型プロパティです。CdbrVariableArray クラスのオブジェクトとして作成します。

ここでは，CdbrDocument オブジェクトとして作成した文書に対して，USER_A というユーザ識別子を持つユーザに参照更新権を設定する場合について説明します。

準備

ローカル ACL を設定するオブジェクトの OIID を検索などによってあらかじめ取得しておきます。検索については，「4.　オブジェクトの検索」を参照してください。

1. CdbrCompound クラスのオブジェクトの ACE を表すプロパティ（dbrProp_Subject プロパティ，dbrProp_SubjectType プロパティ，dbrProp_Permission プロパティ）に，それぞれサブジェクト，サブジェクト種別，パーミッションを設定します。
2. 1. で作成した CdbrCompound オブジェクトを，CdbrVariableArray クラスのオブジェクトの要素として設定します。
3. 2. を SDBR_PROP 構造体の uniValue.ppObject に設定します。
4. 3. を SDBR_PROPLIST 構造体に格納して，PutPropertyValues メソッドの引数として，メソッドをコールします。

ACE，ACL を作成して，オブジェクトのローカル ACL として設定するコールシーケンスの例を次に示します。

ACE，ACL を作成して，オブジェクトのローカル ACL として設定するコールシーケンス

```
// 変数DocumentはCdbrDocumentクラスから作成した文書
SDBR_PROP     Prop;
SDBR_PROPLIST PropList;
Dmapv         pObj;
CdbrCompound ACE;
CdbrVariableArray  ACL(DBR_DATATYPE_COMPOUND);
pSession->Begin();
Document.SetOIID(pSession, pOIID);
// ACEにサブジェクト，サブジェクト種別およびパーミッションを設定する
ACE.SetValue(&dbrProp_Subject, "USER_A");
ACE.SetValue(&dbrProp_SubjectType, DBR_SUBJECTTYPE_USR);
ACE.SetValue(&dbrProp_Permission, DBR_PERM_READ_WRITE);
// ACLの要素数を1にして，ACEをACLに格納する
ACL.Resize(1);
ACL.SetValue(0,ACE);
// ACLをプロパティ構造体に設定する
Prop.PropId = dbrProp_ACL;
Prop.lType = DMA_DATATYPE_OBJECT;
Prop.lCardinality = EDM_DMA_CARDINALITY_VARRAY;
```

```
Prop.lCount = 1;
pObj = &ACL;
Prop.uniValue.ppObject = &pObj;
// プロパティ構造体の値（＝ACL）をDMAオブジェクトに設定する
PropList.pItem = &Prop;
PropList.lCount = 1;
Document.PutPropertyValues(&PropList);
Document.ReleaseObject();
pSession->Commit();
```

(3) 文書またはコンテナへの ACL の設定

VariableArray 型プロパティである ACL を文書またはコンテナのローカル ACL に設定する場合は，PutPropertyValues メソッドをコールして設定します。操作の詳細については，「2.6.7　VariableArray 型プロパティの操作」を参照してください。

ただし，ACL をオブジェクトに設定できるのは，ローカル ACL を設定しようとしている文書やコンテナの所有者，セキュリティ ACL に登録されているユーザまたはセキュリティ管理者だけです。

(4) パブリック ACL の作成

パブリック ACL は，文書やコンテナなどのオブジェクトと同じように，CreateObject メソッドをコールして作成します。メソッドの発行順序については，「3.2.3(1)　文書の作成」を参照してください。ここでは，パブリック ACL を作成する場合の準備について示します。

準備

オブジェクトの構成要素になる DMA オブジェクト作成用のクラス識別子を指定した構造体（SDBR_DMAINFO 構造体）を作成しておきます。
このとき，ClassId には edmClass_PublicACL クラスのクラス識別子を指定します。これ以外のクラスは指定できません。
また，設定する ACL をプロパティ構造体（SDBR_PROP）に設定して，プロパティリスト構造体（SDBR_PROPLIST）に設定します。これを，SDBR_DMAINFO 構造体に設定しておきます。

(5) 文書またはコンテナへのパブリック ACL の設定

作成したパブリック ACL を文書やコンテナからバインドする場合の操作について説明します。

ここでは，CdbrDocument オブジェクトからパブリック ACL をバインドする場合について説明します。

なお，パブリック ACL をオブジェクトにバインドできるのは，バインドしようとしているオブジェクトの所有者，セキュリティ ACL に登録されているユーザまたはセキュリティ管理者だけです。

準備

パブリック ACL（CdbrPublicACL オブジェクト）の構成要素である edmClass_PublicACL クラスのオブジェクトの OIID を，検索などによってあらかじめ取得しておきます。

1. CdbrCompound クラスのオブジェクトの dbrProp_ACLIdElem プロパティに，バインドするパブリック ACL の OIID を設定します。
2. VariableArray 型プロパティとして，CdbrVariableArray オブジェクトの構成要素に 1. を設定します。
3. 2. を SDBR_PROP 構造体の uniValue.ppObject に設定します。このプロパティ構造体を，SDBR_PROPLIST 構造体に設定して，PutPropertyValues メソッドをコールします。

パブリック ACL を CdbrDocument オブジェクトからバインドするコールシーケンスの例を次に示します。

CdbrDocument オブジェクトにパブリック ACL を設定するコールシーケンス

```
SDBR_PROP     Prop;
SDBR_PROPLIST PropList;
Dmapv         pObj;
CdbrDocument  Document;
CdbrCompound  PublicACLOIID1,PublicACLOIID2;
CdbrVariableArray  Varray(DBR_DATATYPE_COMPOUND);
pSession->Begin();
Document.SetOIID(pSession, pOIID);
//パブリックACLのOIIDを可変長配列の要素として指定する
PublicACLOIID1.SetValue(&dbrProp_ACLIdElem,pPubOIID1);
PublicACLOIID2.SetValue(&dbrProp_ACLIdElem,pPubOIID2);
//VariableArrayプロパティの要素数を設定する
Varray.Resize(2):
//要素を設定する
Varray.SetValue(0,PublicACLOIID1);
Varray.SetValue(1,PublicACLOIID2);
//OIIDリスト（VariableArray型プロパティ）を
//プロパティ構造体に設定する
Prop.PropId = dbrProp_PublicACLIds,
Prop.lType = DMA_DATATYPE_OBJECT;
Prop.lCardinality = EDM_DMA_CARDINALITY_VARRAY;
Prop.lCount = 1;
pObj = &Varray;
Prop.uniValue.ppObject = &pObj;
//プロパティ構造体の値をCdbrDocumentオブジェクトに設定する
PropList.pItem = &Prop;
PropList.lCount = 1;
Document.PutPropertyValues(&PropList);
Document.ReleaseObject();
pSession->Commit();
```

（6）文書またはコンテナへのパブリック ACL のバインド

文書やコンテナがすでにパブリック ACL をバインドしている場合，PutPropertyValues メソッドによってパブリック ACL を追加すると，それまでバインドしていたパブリック ACL はアンバインドされます。

ここでは，文書やコンテナに設定された既存のパブリック ACL に，ほかのパブリック ACL を追加してバインドする場合の操作について説明します。

ここでは，CdbrDocument オブジェクトにパブリック ACL を追加してバインドする場合について説明します。

なお，パブリック ACL をオブジェクトからバインドできるのは，バインドしようとしているオブジェクトの所有者，セキュリティ ACL に登録されているユーザまたはセキュリティ管理者だけです。

準備

パブリック ACL（CdbrPublicACL オブジェクト）の構成要素である edmClass_PublicACL クラスのオブジェクトの OIID を，検索などによってあらかじめ取得しておきます。

1. **CdbrDocument オブジェクトに接続します。**
   CdbrDocument::SetOIID メソッドまたは CdbrDocument::ConnectObject メソッドをコールします。
2. **CdbrDocument オブジェクトと CdbrPublicACL オブジェクトをバインドします。**
   CdbrDocument::BindPublicACL メソッドをコールします。
   これによって，CdbrDocument オブジェクトと CdbrPublicACL オブジェクトがバインドされ，CdbrDocument オブジェクトの VariableArray 型プロパティである dbrProp_PublicACLIds プロパティに OIID が設定されます。

CdbrDocument オブジェクトにパブリック ACL をバインドするコールシーケンス

```
pSession->Begin();
//…
//検索によってバインドするパブリックACLのOIIDを取得する
//…
CdbrDocument Document;
pDmaString_T ppIdList[2];
DmaInteger32 lIdListCount;
//CdbrDocumentオブジェクトに接続する
Document.SetOIID(pSession, pOIID);
//CdbrDocumentオブジェクトとパブリックACLをバインドする
lIdListCount = 2;
ppIdList[0] = pPubOIID1;
ppIdList[1] = pPubOIID2;
Document.BindPublicACL(lIdListCount,ppIdList);
Document.ReleaseObject();
pSession->Commit();
```

3.15.10 アクセス制御と排他制御の関係

ここでは，アクセス制御と排他制御の関係について説明します。

(1) アクセス権判定時に設定されるロック

作成したオブジェクトに対してメソッドをコールすると，DocumentBroker ではオブジェクトの操作に必要なロックを設定して処理します。アクセス制御では，まず，クラスライブラリのオブジェクトに対してロックを設定してから，ACFlag，パブリック ACL およびローカル ACL を参照して，アクセス権を判定します。

ACFlag およびローカル ACL は，クラスライブラリのオブジェクトのプロパティとして設定されています。したがって，アクセス権判定でこれらを参照するとき，ACFlag およびローカル ACL には，オブジェクトに設定されているロックと同じ種類のロックが設定されます。

また，アクセス権判定でパブリック ACL を参照する場合は，参照するパブリック ACL に read ロックを設定します。

注意

オブジェクトの排他制御では，メソッドをコールした時点で，アクセス権判定の前にオブジェクトに対してロックが設定されます。このロックは，アクセス権がないためにメソッドがアクセスエラーになっても解除されません。アクセスエラーが発生した場合は，ロールバックをするか，トランザクションを終了してロックを解除してください。

(2) アクセス制御情報の参照，更新時に設定されるロック

GetPropertyValues メソッドなどのプロパティを参照するメソッドによって ACFlag や ACL などのアクセス制御情報を参照する場合や，PutPropertyValues メソッドによってアクセス制御情報を更新する場合，アクセス制御情報をプロパティとして設定してあるクラスライブラリのオブジェクトに対してロックを設定します。このロックの設定方法には，次の 2 種類があります。

● ユーザが明示的に read ロックまたは write ロックを設定する場合

● DocumentBroker が暗黙的にメソッドの実行に必要なロックを設定する場合

次の手順でメソッドがコールされた場合の排他制御の関係を，表 3-36 に示します。

1. ユーザ A がメソッドをコールしてオブジェクトのロックを設定する。
2. 1. でユーザ A がロックを設定したオブジェクトに対して，ユーザ B がロックを設定するメソッドをコールする。

表 3-36　コールするメソッドの種類とアクセス制御情報の排他制御の関係

ユーザ A がコールしたメソッドの種類	ユーザ B がコールしたメソッドの種類						
	read ロックを設定するメソッド	write ロックを設定するメソッド	ACFlag またはローカル ACL を参照するメソッド	ACFlag またはローカル ACL を更新するメソッド	パブリック ACL を参照するメソッド	パブリック ACL を更新するメソッド	パブリック ACL を削除するメソッド
read ロックを設定するメソッド	○	×	○	×	○	△	×
write ロックを設定するメソッド	×	×	×	×	○	△	×
ACFlag またはローカル ACL を参照するメソッド	○	×	○	×	○	○	×
ACFlag またはローカル ACL を更新するメソッド	×	×	×	×	○	○	×
パブリック ACL を参照するメソッド	○	○	○	○	○	×	×
パブリック ACL を更新するメソッド	△	△	○	○	×	×	×
パブリック ACL を削除するメソッド	×	×	×	×	×	×	×

（凡例）
○：排他されます。
×：排他されません。
△：アクセス権判定時にパブリック ACL が参照されている場合は，排他されます。

参考

read ロックまたは write ロックに DBR_RLT_FOR_UPDATE を指定しても，この表で説明している排他制御の関係は変わりません。DBR_RLT_FOR_UPDATE については，「3.14.1　ロックの概要」を参照してください。

パブリック ACL に対するメソッドが排他制御されるのは，アクセス権判定のときにパブリック ACL の内容が参照されている場合です。つまり，ユーザ A があるオブジェクトを参照しようとしたときに，ユーザ権限や ACFlag の内容でアクセス権が確認できず，パブリック ACL の内容を参照してアクセス権判定を行った場合，そのパブリック ACL にはユーザ A によって read ロックが設定されます。このとき，ユーザ B はパブリック ACL の内容を更新できません。

また，パブリック ACL をユーザ A が更新している場合に，パブリック ACL によるアクセス権判定が必要なメソッドをユーザ B がコールすることはできません。パブリック ACL を削除するときには，このパブリック ACL をバインドしているオブジェクトにも write ロックが設定されます。

4 オブジェクトの検索

この章では，オブジェクトを検索する方法について説明します。DocumentBroker では，SQL に準じた構文規則を持つ edmSQL を使用してオブジェクトを検索できます。この検索では，プロパティの値を指定した検索や，検索タームを指定した検索が実行できます。

4.1 検索の概要

この節では，DocumentBroker で提供する検索機能の概要について説明します。

4.1.1 DocumentBroker での検索

DocumentBroker で作成した文書やコンテナを表すオブジェクトは，データベースに格納されます。

既存のオブジェクトを操作するためには，そのオブジェクトの情報（プロパティ）を取得して，オブジェクトに接続する必要があります。

オブジェクトの情報を取得するには，次のような方法があります。

● 検索を実行するメソッドをコールする方法
OIID を知らない場合でも，そのオブジェクトについて知っている情報を検索条件に指定して，検索を実行してオブジェクトの情報を取得できます。DocumentBroker では，検索条件を指定する検索の実行方法として，次の方法を提供しています。
- CdbrEqlStatement クラスの機能を使用して edmSQL 文によって検索条件を設定する検索

● プロパティを取得するメソッドをコールする方法
すでに OIID を知っているオブジェクトの場合は，その OIID を指定してオブジェクトと接続して，OIID 以外のプロパティを取得できます。プロパティを取得するメソッドとしては，GetPropertyValues メソッドなどのほか，一覧取得メソッドにも同時にプロパティを取得できるものがあります。例えば，コンテインメントの一覧を取得したり，リレーションの一覧を取得したりすることで，コンテインメントやリレーションをたどったオブジェクトの検索ができます。
プロパティを取得するメソッドについては，「2.6.6(4)　プロパティの取得」を参照してください。また，それぞれのメソッドについては，マニュアル「DocumentBroker Version 3　クラスライブラリ C++　リファレンス　基本機能編」を参照してください。

この章では，CdbrEqlStatement クラスの機能で検索を実行する場合の方法と，検索条件を設定するための edmSQL 文の概要について説明します。

4.1.2 edmSQL とは

edmSQL とは，クラスライブラリのオブジェクトを検索するための手段として DocumentBroker が提供する検索方法です。SQL の文法に関する知識を活用して，DocumentBroker でのオブジェクトの検索ができます。なお，これ以降，このマニュアルでは，edmSQL を使用した検索を，**edmSQL 検索**といいます。また，edmSQL に従って記述した条件式を，**edmSQL 文**といいます。

edmSQL には，次のような特長があります。

● 文書管理で使用する概念に基づいた検索条件を指定できます。
edmSQL では，DocumentBroker の文書管理で使用するクラス名やプロパティ名を使用して，検索条件を指定できます。

● データベース標準問い合わせ言語である SQL に基づいた文法です。
edmSQL の文法は，標準的な SQL の文法に基づいています。SQL の文法に関する知識を活用して edmSQL 文を作成できます。

● クラスライブラリと違和感なく組み合わせて使用するためのインターフェースが提供されています。
クラスライブラリのほかのクラスやメソッドと違和感なく組み合わせて使用するために，

CdbrEqlStatement クラスを提供しています。

● **デバッグ情報を取得できます。**
edmSQL 検索のデバッグ情報を取得できます。

なお，edmSQL の検索モデルは，DMA の Query Model に基づきますが，拡張機能のサポートなどに伴い，一部 DMA の仕様と異なる部分があります。

4.1.3 検索とアクセス制御

アクセス制御機能を使用している環境の場合，検索結果として取得される情報に対して，アクセス制御ができます。アクセス制御情報は，検索対象のオブジェクトごとに設定できます。アクセス制御情報の詳細については，「3.15　アクセス制御」を参照してください。アクセス制御機能を実行するか実行しないかは，検索ごとに指定できます。

アクセス制御を実行すると，アクセス制御を実行しない場合に比べて，処理に時間がかかります。このため，検索結果に対してアクセス制御をするかしないかは，業務に応じて決定してください。

また，アクセス制御を実行する場合の検索には，次の制限があります。

● アクセス制御の対象になるオブジェクトは，主問い合わせの検索対象であるオブジェクトだけです。したがって，副問い合わせの検索対象であるオブジェクトには，アクセス制御は適用されません。

● アクセス制御は，オブジェクトを対象に実行されます。この場合の問い合わせはオブジェクトの識別性に基づいた検索になります。したがって，重複排除のようなオブジェクトのプロパティ値に基づく問い合わせ，集合関数のような統計演算をする検索，および複数のオブジェクトを一つのグループとする問い合わせは実行できません。

4.1.4 edmSQL 検索の流れ

ここでは，edmSQL 検索を実行する場合の検索の流れについて説明します。edmSQL 検索の流れは，? パラメタを使用しない場合と使用する場合で異なります。

なお，それぞれのメソッドの詳細については，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。

(1) ? パラメタを使用しない場合の edmSQL 検索の流れ

? パラメタを使用しない場合の edmSQL 検索の流れを次に示します。

1. **CdbrEqlStatement オブジェクトを作成します。**
   CdbrEqlStatement オブジェクトは，new によって作成します。
2. **CdbrEqlStatement オブジェクトを初期化します。**
   CdbrEqlStatement::Initialize メソッドをコールします。
3. **実行する edmSQL 文を設定します。**
   CdbrEqlStatement::Set メソッドをコールします。
   edmSQL の文法については，「4.5　edmSQL の文法」を参照してください。
4. **検索を実行します。**
   CdbrEqlStatement::Execute メソッドをコールします。
5. **検索結果を取得します。**
   CdbrEqlStatement::GetResult メソッドをコールします。

6. edmSQL 検索を終了します。
   CdbrEqlStatement::Terminate メソッドをコールします。
7. CdbrEqlStatement オブジェクトを削除します。
   CdbrEqlStatement オブジェクトは，delete によって削除します。

(2) ? パラメタを使用する場合の edmSQL 検索の流れ

? パラメタを使用する場合の edmSQL 検索の流れを次に示します。

1. CdbrEqlStatement オブジェクトを作成します。
   CdbrEqlStatement オブジェクトは，new によって作成します。
2. CdbrEqlStatement オブジェクトを初期化します。
   CdbrEqlStatement::Initialize メソッドをコールします。
3. 実行する edmSQL 文を設定します。
   CdbrEqlStatement::Set メソッドをコールします。
   edmSQL の文法については，「4.5　edmSQL の文法」を参照してください。
4. edmSQL 文中の ? パラメタの値を設定します。
   CdbrEqlStatement::SetParam メソッドをコールします。
   ? パラメタの値として OIID を設定する場合は，CdbrEqlStatement::SetOIIDParam メソッドをコールします。
   ? パラメタの値として DMA オブジェクトを設定する場合は，CdbrEqlStatement::SetObjParam メソッドをコールします。
   edmSQL 文中に ? パラメタが複数存在する場合には，設定する ? パラメタの数だけメソッドをコールしてください。
5. 検索を実行します。
   CdbrEqlStatement::Execute メソッドをコールします。
6. 検索結果を取得します。
   CdbrEqlStatement::GetResult メソッドをコールします。
7. 同一の edmSQL 文に対して，? パラメタの値を変更します。
   コールするメソッドについては，4. と同様です。
8. ? パラメタの値を変更した検索条件で，検索を実行します。
   CdbrEqlStatement::Execute メソッドをコールします。
9. 検索結果を取得します。
   CdbrEqlStatement::GetResult メソッドをコールします。
   なお，以降，変更する ? パラメタの値の数だけ，7. ～ 9. を繰り返し実行してください。
10. edmSQL 検索を終了します。
    CdbrEqlStatement::Terminate メソッドをコールします。
11. CdbrEqlStatement オブジェクトを削除します。
    CdbrEqlStatement オブジェクトは，delete によって削除します。

(3) データが保持される期間

ここでは，edmSQL 検索で設定したデータがどの時点まで保持されるかについて，説明します。

(a) 設定した edmSQL 文および？パラメタの値

CdbrEqlStatement::Set メソッドで設定した edmSQL 文は，次に Set メソッドをコールして異なる edmSQL 文を設定するまで保持されます。

CdbrEqlStatement::SetParam メソッド，CdbrEqlStatement::SetOIIDParam メソッドおよび CdbrEqlStatement::SetObjParam メソッドで設定した？パラメタの値は，次に Set メソッドをコールするまで保持されます。

(b) ロック種別・検索結果の取得個数・アクセス制御モード

設定したロック種別は，次に CdbrEqlStatement::ChangeLockType メソッドをコールするまで有効です。

検索結果の取得個数の指定は，次に CdbrEqlStatement::ChangeGetObjCount メソッドをコールするまで有効です。

アクセス制御モードは，次に CdbrEqlStatement::ChangeACLMode メソッドをコールするまで有効です。

4.2 検索の種類

ここでは，edmSQL で実現できる検索の種類について説明します。edmSQL では，次に示す検索が実行できます。

- 属性検索
- 全文検索
- 全文検索機能付き文字列型プロパティを使用した全文検索

これらの検索は，組み合わせて指定することもできます。ここでは，それぞれの検索の概要について説明します。

また，edmSQL 検索では？パラメタも使用できます。？パラメタを使用した検索の概要についても説明します。

4.2.1 属性検索

属性検索とは，オブジェクトの一つまたは複数個のプロパティの値を指定して実行する検索です。例えば，「文書名」がプロパティとして設定されている文書に対して，「文書名が『SQL 入門』である文書を探す」といった検索が実行できます。複数個のプロパティの値を指定する場合は，論理条件を指定して，「文書名が『SQL 入門』で，かつ，出版社が『ABC 出版』である文書を探す」といった検索もできます。

検索の対象として，Scalar 型のプロパティのほか，VariableArray 型のプロパティも指定できます。例えば，著者が複数人存在する可能性がある文書を管理するために，「著者」を VariableArray 型プロパティで管理して，それぞれの「名前」をその要素のプロパティとして管理している場合，「『田中』という名前の著者が含まれる文書を探す」といった条件を指定した検索が実行できます。

4.2.2 全文検索

全文検索とは，登録した全テキストを対象に，キーワードとなる文字列（**検索ターム**）を一つまたは複数個指定して実行する検索です。例えば，「文書の中に『コンピュータ』が含まれる文書を探す」といった検索が実行できます。

また，全文検索には，検索条件に単語ではなく文章を指定できる検索もあります。これを**概念検索**といいます。概念検索では，検索条件に指定した文章（**種文章**）に近い概念を持つ（同じ単語が多く含まれる）文書が検索できます。

なお，全文検索を実行する場合は，HiRDB Text Search Plug-in が必要です。概念検索を実行する場合は，HiRDB Text Search Plug-in のほか，HiRDB Text Search Plug-in Conceptual Extension が必要です。

全文検索の対象にする文書は，全文検索機能付き文書クラスを基に作成されている必要があります。全文検索機能付き文書クラスの作成方法については，「4.4.2　全文検索機能付き文書クラスの作成」を参照してください。

edmSQL では，全文検索の次の機能を実現するための関数を提供しています。

- プレーンテキストおよび構造を持った文書に対する全文検索
- プレーンテキストおよび構造を持った文書に対する概念検索
- 検索結果にスコアを付ける全文検索
- 検索結果にハイライトタグの埋め込みをして SGML 形式で出力する全文検索

ここでは，これらの検索の概要と，edmSQL で提供する関数について説明します。関数についての詳細は，「4.5.9　関数指定の構文規則」を参照してください。

また，全文検索では，HiRDB Text Search Plug-in の機能によって，検索タームや種文章をそのまま指定する検索だけでなく，同義語や異表記を同時に検索したり，検索タームに重みを指定したりすることもできます。

これらの，全文検索で指定できる機能の概要についても説明します。

(1) 全文検索の種類

ここでは，全文検索の種類について説明します。

(a) プレーンテキストに対する全文検索

プレーンテキストとは，構造を持たないテキスト形式の文書のことです。文書の全テキストに対して，一つまたは複数個の検索タームを指定した検索が実行できます。複数個の検索タームを指定する場合，それらの論理条件も指定できます。

例えば，「文書中に『コンピュータ』または『データベース』を含む文書」を検索したりできます。また，属性検索条件と組み合わせて，「文書のタイトル（プロパティ）が『データベース入門』であり，文書中に『コンピュータ』を含む文書」のような検索も実行できます。

また，検索タームとして指定した単語と完全一致または前方一致する単語を含む文書を検索したり，複数の単語をフレーズとして指定して，そのフレーズを含む文書を検索したりできます。

例えば，「文書中に『DocumentBroker』を含む文書」を検索したり，「文書中に『DocumentBroker Development Kit』というフレーズを含む文書」を検索したりできます。

この検索には，contains 関数を使用します。

(b) 構造を持った文書に対する全文検索

プレーンテキストに対して実行できる全文検索に加えて，XML 文書のような構造を持った文書に対しては，その構造を条件に指定した全文検索ができます。このような検索を**構造指定検索**といいます。

構造指定検索では，XML 文書のように論理構造を持っている文書に対して，その構造を指定した検索が実行できます。例えば，「文書・章・節」という構造で定義された XML 文書に対して，「構造『章』の中に『XML』という単語が含まれる文書」などが検索できます。

また，**構造に付けられた属性の値を指定した検索**もできます。例えば，「文書・タイトル・本文」という構造で定義され，構造「タイトル」が「分類」という属性を持っている XML 文書に対して，「構造『タイトル』の属性『分類』が『政治経済』である文書」などが検索できます。

さらに，複数の検索条件を指定するときに，それらの検索条件が特定の構造内に含まれることを指定することもできます。例えば，「文書・著者」という構造に，「所属」という構造と「名前」という構造が含まれる XML 文書に対して，「特定の『文書・著者』という構造の中の，構造『所属』に『設計部』が含まれ，構造『名前』に『山田』が含まれる文書」などが検索できます。

この検索には，contains 関数を使用します。

(c) プレーンテキストに対する概念検索

全文検索を実行する時に，特定の検索タームを基に検索する以外に，「ある文章に似た内容の文書を検索したい」というような検索が実行できます。例えば，「リサイクルを含む環境問題」について書かれた文書を探したいときに，「ゴミのリサイクル」に関する文章を基に検索が実行できます。なお，以降このマニュアルでは，全文検索のうち概念検索だけに特有の説明については，概念検索と明記して，それ以外の全文検

索と区別します。

概念検索は，edmSQL などの検索条件に種文章を指定することで実行されます。ここでは，種文章に「…ゴミのリサイクルによって，資源が有効活用され，ゴミが減量化されます。…」という文字列を指定した場合の，概念検索の実行の流れを説明します。なお，種文章として指定できる容量は，5 メガバイト以内です。

1. HiRDB Text Search Plug-in Conceptual Extension **の機能によって，種文章から特徴タームが抽出されます。**
   特徴タームとは，種文章に指定した文章から，漢字，かたかな，およびアルファベットを抽出した単語です。
   この例の場合は，「ゴミ」，「リサイクル」，「資源」，「有効活用」，「減量化」というような単語が抽出されます。
2. HiRDB Text Serach Plug-in Conceptual Extension **によって，**1. **の特徴タームに優先順位が付与された検索用特徴タームが選出されます。**
   検索用特徴タームの抽出基準については，HiRDB Text Search Plug-in の環境定義ファイルに指定します。詳細は，マニュアル「HiRDB Text Search Plug-in」を参照してください。
3. **検索用特徴タームを検索タームとして，全文検索が実行されます。さらに，種文章との適合度順に，スコア値が設定されて，検索結果が出力されます。**
   概念検索では，この検索用特徴タームを基に，検索が実行されます。

なお，概念検索に指定する概念検索条件は，?パラメタを使用して指定する必要があります。

この検索には，concept_with_score 関数を使用します。

(d) 構造を持った文書に対する概念検索

構造を持った文書に対して概念検索を実行する場合も，検索条件に構造が指定できます。例えば，「文書・章・節」という構造が定義されている文書に対して，「構造『章』の中に『…ゴミのリサイクルによって，資源が有効に活用され，…』という文章に似た概念を含んでいる文書を検索する」というような検索が実行できます。

この検索には，concept_with_score 関数を使用します。

(e) 検索結果にスコアを付けて，スコア順にソートして取得できる全文検索

全文検索の検索結果には，スコアが付けられます。スコアの算出方法は，検索タームを指定する全文検索と，種文章を指定する概念検索で異なります。

検索タームを指定する全文検索の場合

スコアは，「検索結果として取得する文書の中に含まれる検索タームの数×検索タームに指定した重み×構造に指定した重み」の値として算出されます。ただし，検索条件に複数の検索タームの論理条件を指定したり，近傍検索条件を指定したりする場合，スコアの算出方法は異なります。
スコアの算出方法の詳細については，マニュアル「HiRDB Text Search Plug-in」を参照してください。
この検索には，contains_with_score 関数を使用します。スコア値は score 関数を使用して取得します。

種文章を指定する概念検索の場合

スコアは，種文章との適合度によって算出されます。適合度とは，種文章から抽出した，検索用特徴タームが，検索結果に含まれる割合のことです。
また，種文章を検索した場合のスコアを 100 として，スコアを正規化して算出することもできます。
スコアの算出方法については，マニュアル「HiRDB Text Search Plug-in」を参照してください。

この検索には，concept_with_score 関数を使用します。スコア値は score_concept 関数を使用して取得します。

(f) 検索結果にハイライトタグの埋め込みをして SGML 形式で出力する検索

全文検索の実行時に，検索結果として検索タームがハイライト表示される SGML 形式のファイルを出力します。例えば，「ネットワーク」という検索タームが含まれるテキスト形式の文書を検索する場合に，検索結果と同時に「<STRONG>」というハイライト表示用タグが埋め込まれた SGML 形式のファイルを出力できます。

なお，検索結果として取得するハイライト表示用の文書は，検索対象である文書とは異なる文書として作成されます。

次に，検索対象文書とハイライト表示用文書の例を示します。

検索対象文書

```
<!DOCTYPE body SYSTEM "DOC.dtd">
<body>
<p>
連絡事項
<note>
ネットワーク変更については，添付資料を参照してください。
</body>
```

検索結果として取得するハイライト表示用の文書

```
<!DOCTYPE body SYSTEM "DOC.dtd">
<body>
<p>
連絡事項
<note>
<STRONG>ネットワーク</STRONG>変更については，添付資料を参照してください。
</body>
```

(2) 全文検索で指定できる機能

edmSQL では，全文検索条件として，HiRDB Text Search Plug-in の規則に従った検索条件が指定できます。HiRDB Text Search Plug-in では，単純文字列としての検索タームおよび種文章のほか，検索タームを展開したり，重みを設定したりする指定ができます。ここでは，これらの全文検索条件に指定できる機能の概要について説明します。

edmSQL では，これらの指定は全文検索を実行する関数の引数として指定します。指定方法については，マニュアル「HiRDB Text Search Plug-in」を参照してください。

ここでは，概要を説明します。

(a) 検索タームの異表記を検索対象にした検索（異表記展開指定）

異なる表記方法で表記された検索タームを含む文書も検索条件に指定できる検索です。

検索タームに指定したかたかなの異表記を含む文書を検索したり，全角アルファベットの大文字・小文字の異表記を含む文書を検索したり，アルファベットの全角・半角の異表記を含む文書を検索したりできます。例えば，検索ターム「バイオリン」を検索するときに，「ヴァイオリン」のように表記されている文書も検索できます。また，検索ターム「Ski」を検索するときに，「ski」や「Ｓｋｉ」のように表記されている文書も検索できます。

異表記展開検索では，データベースで決められた規則に従って，検索タームの異表記を含む文書を検索します。

(b) 検索タームの同義語を検索対象にした検索（同義語展開指定）

検索タームと同じ意味を持つ単語（同義語）を含む文書も検索条件に指定できる検索です。

例えば，検索ターム「Ski」を検索するときに，「スキー」を含む文書も一緒に検索できます。同義語展開検索では，あらかじめ作成してある同義語辞書を基に，検索タームの同義語を含む文書を検索します。

(c) 複数の検索ターム間の距離条件や出現順序を指定した検索（近傍条件指定）

複数の検索タームを指定した場合に，検索タームが出現する距離（文字数）を検索条件に指定できる検索です。

例えば，「『最新』と『技術』という検索タームを含み，これらの二つの単語が 10 文字以内に存在する文書」などが検索できます。この場合には，「最新印刷技術」や「最新の技術」などの文字列を含む文書が検索されます。

(d) 検索タームおよび構造に重みを付ける検索（重み指定）

検索タームに**重み**（重要度）を付ける検索です。重みを付けた検索とは，「『インターネット』および『データベース』が含まれる文書が検索したい。ただし，『インターネット』をより多く含む文書から取り出したいので，『インターネット』に『5』，『データベース』に『2』の重みを付けて検索する」のように指定する検索です。重みは構造にも付けられます。重みを付けた場合，スコアは重みによって算出されます。

概念検索では，重みは構造だけに指定できます。

(e) 種文章のスコアを 100 として，検索結果に相対的なスコアを付ける検索（概念検索のスコアオプション指定）

概念検索を実行する時に取得するスコア値を，正規化して取得する指定です。正規化した値は，種文章を検索した場合を 100 とした，相対的な値として算出されます。

(f) 構造の属性値を指定した検索（属性値指定）

構造を指定する場合に，その属性値も指定する検索です。

例えば，構造「文章」の属性「Author」に「Tanaka」を含む文書などが検索できます。

なお，概念検索では，属性値指定はできません。

(g) 複数の構造を対象にした検索条件を満たす構造が，ある特定の構造に含まれていることを指定する検索（特定構造検索指定）

複数の構造を対象にした検索を実行する場合に，その検索条件を満たす構造が，ある特定の構造に含まれていることを指定する検索です。構造「文章」の下位構造として「作成者」と「所属」がある場合に，特定の構造「文章」の構造「作成者」に「Tanaka」が含まれ，かつ構造「所属」に「営業部」が含まれる文書などが検索できます。

なお，概念検索では，特定構造検索指定はできません。

4.2.3 全文検索機能付き文字列型プロパティを使用した全文検索

ここでは，全文検索機能付き文字列型プロパティを使用した全文検索について説明します。なお，全文検索機能付き文字列型プロパティを使用するには HiRDB Text Search Plug-in が必要です。

(1) 全文検索機能付き文字列型プロパティとは

全文検索が実行できる文字列型プロパティのことを**全文検索機能付き文字列型プロパティ**といいます。全

文検索機能付き文字列型プロパティを使用すると，edmSQL で次に示す検索条件を指定して文書の検索ができます。

1. 指定した文字列がプロパティの値に含まれているか
2. 指定した文字列を異表記展開した文字列がプロパティの値に含まれているか
3. 指定した文字列を同義語展開した文字列がプロパティの値に含まれているか
4. 指定した文字列の距離（文字数）が指定の距離内にあるか

また，これらの検索条件を AND または OR で結ぶことができます。

(2) 全文検索機能付き文字列型プロパティの全文検索

edmSQL を使用して全文検索機能付き文字列型プロパティの値を取得する場合は，SELECT 句に extracts 関数を指定してください。また，全文検索機能付き文字列型プロパティを全文検索する場合は，WHERE 句に contains 関数を指定してください。

全文検索機能付き文字列型プロパティを使用した全文検索の例を次に示します。

[例]

全文検索機能付き文字列型プロパティ usrProp_DocSummary に対応する文書中に，「コンピュータ」を同義語展開した文字列があるかどうかを検索し，文字列があればそのテキストデータを抽出します。

edmSQL の指定例

```
SELECT extracts(usrProp_DocSummary)
  FROM   usrClass_PropTextSearch
  WHERE  contains(usrProp_DocSummary,'{SYNONYM(myDic, "コンピュータ")}') is
true
```

注　usrClass_PropTextSearch は dmaClass_DocVersion クラスのサブクラスです。

注意事項

全文検索機能付き文字列型プロパティは extracts 関数または contains 関数の第一引数に指定してください。それ以外の個所には指定できません。

(3) 全文検索機能付き文字列型プロパティに対応しているクラスおよびメソッド

全文検索機能付き文字列型プロパティに対応しているクラスおよびメソッドを次の表に示します。

表 4-1　全文検索機能付き文字列型プロパティに対応しているクラスおよびメソッド

対応しているクラス	対応するメソッド
CdbrVersionableDocument	• CreateObject • GetPropertyValues • GetPropertyValuesAndLock • PutPropertyValues
CdbrDocument	• CreateObject • GetPropertyValues • GetPropertyValuesAndLock • PutPropertyValues
CdbrReferentialContainer	• CreateObject • GetPropertyValues • GetPropertyValuesAndLock • PutPropertyValues

対応しているクラス	対応するメソッド
CdbrConfiguratedReferentialContainer	• CreateObject • GetPropertyValues • GetPropertyValuesAndLock • PutPropertyValues
CdbrVersionTraceableContainer	• CreateObject • GetPropertyValues • GetPropertyValuesAndLock • PutPropertyValues
CdbrIndependentPersistence	• CreateObject • GetPropertyValues • GetPropertyValuesAndLock • PutPropertyValues
CdbrEqlStatement	• すべてのメソッド

4.2.4 ？パラメタを使用した検索

業務によっては，検索条件のうち，一部の定数値だけを変更して繰り返し同じ検索を実行したい場合があります。このような場合に，？パラメタを使用できます。

例えば，文書の作成日がプロパティとして設定されている場合に，作成日の値だけを「20000103」，「20000104」，「20000105」と変更して検索したい場合があります。このような，検索条件の特定の個所だけ変更して繰り返し検索を実行する場合に，このパラメタが使用できます。この検索では，検索条件式の作成日の値には「?」を設定しておきます。実行するたびに変更する値については別のパラメタとして指定します。これによって，複数の検索に対して一つの検索条件式を繰り返し使用でき，2 回目以降の検索にかかる処理時間を短縮できます。このパラメタを，**？パラメタ**といいます。

なお，？パラメタを使用した検索は edmSQL 検索だけで実行できます。

4.3 検索対象になるクラスおよびプロパティ

この節では，edmSQL 検索の対象になるクラスとプロパティについて説明します。

4.3.1 検索対象になるクラス

検索では，DMA クラスを検索対象として指定します。DMA クラスを指定すると，文書空間内に存在する，そのクラスを基に作成されているすべてのオブジェクトが，検索の対象になります。

例えば，DMA クラスの usrClass_DocVersion クラス（dmaClass_DocVersion クラスのサブクラス）を検索対象に指定すると，そのクラスを基に作成された DocVersion オブジェクトを構成要素とする，バージョン付き文書およびバージョンなし文書（またはバージョン付き文書の 1 バージョン）がすべて検索対象になります。

edmSQL 文で検索条件を指定する場合，FROM 句に検索対象にする DMA クラスを指定します。

検索対象には，複数のクラスを結合して指定することもできます。実現できる結合の種類は，データベースによって異なります。

edmSQL では，次の種類の結合ができます。

- 内部結合（INNER JOIN）
- 左外部結合（LEFT OUTER JOIN）

検索対象として指定できるのは，次のクラスまたはそのサブクラスとしてユーザがデータベースに登録したクラスです。

● dmaClass_ConfigurationHistory クラス

● dmaClass_Container クラス

● dmaClass_DirectContainmentRelationship クラス

● dmaClass_DocVersion クラス

● dmaClass_ReferentialContainmentRelationship クラス

● edmClass_ComponentDocVersion クラス

● edmClass_ContainerVersion クラス

● edmClass_ContentSearch クラス

● edmClass_Relationship クラス

● edmClass_VersionTraceableContainer クラス

● edmClass_VersionTraceableContainmentRelationship クラス

● edmClass_VersionTracedComponentDocVersion クラス

● edmClass_VersionTracedDocVersion クラス

● edmClass_PublicACL クラス（アクセス制御機能を使用している環境の場合）

これらのクラスは，検索対象として，DocumentBroker サーバのメタ情報ファイル（edms.ini）に検索可能なクラスとして登録されています。

4.3.2 検索結果として取得できるプロパティ

検索結果としては，検索条件に合ったオブジェクトのプロパティが取得できます。取得できるプロパティは，DMA が規定したプロパティ（dmaProp_ または edmProp_ で始まるプロパティ）や，ユーザが定義したプロパティなどです。

edmSQL 文で検索条件を指定する場合，検索結果として取得するプロパティは，SELECT 句に指定します。データベース上のデータである検索結果は，データ型ごとに対応するクラスライブラリのデータ型に変換されて，構造体の形式で取得できます。検索結果として指定した項目が一つで，検索結果として取得した件数が一つの場合，検索結果は，直接そのプロパティのデータ型の値として扱えます。また，検索結果の件数が複数の場合は，そのデータ型の値の集合として扱えます。これらの値は，副問い合わせの評価値として，述語や演算子にも使用できます。

検索結果として取得できるプロパティは，永続プロパティ（データベースに存在するプロパティ）または VariableArray 型プロパティで，かつプロパティ定義に「dmaProp_IsSelectable = bool = 1」と指定されているプロパティです。

4.3.3 検索条件に指定できるプロパティ

検索対象に指定したクラスのオブジェクト集合から必要な検索結果だけを絞り込んで取得したり，複数の検索対象クラスを指定するときにどのような条件で結合するかを指定したりする場合，検索条件を指定します。

検索条件は，WHERE 句や FROM 句の ON 節に指定できます。検索条件には，検索条件の対象になるデータ型に従って，四則演算などを指定するための演算，比較演算子などを指定するための述語，全文検索などを指定するための関数を指定します。

検索条件に指定できるプロパティは，永続プロパティ（データベースに存在するプロパティ）または VariableArray 型プロパティで，かつプロパティ定義に「dmaProp_IsSearchable = bool = 1」と指定されているプロパティです。

4.4 全文検索の対象になる文書の作成

ここでは，全文検索の対象になるクラスとその作成方法について説明します。

DocumentBroker では，全文検索に必要なプロパティを持ったクラスを基に作成した文書に対してだけ全文検索を実行できます。また，すべての形式（レンディションタイプ）の文書に対して全文検索が実行できますが，全文検索をする文書に対応するテキストデータを基に，全文検索インデクスをあらかじめ作成，登録しておく必要があります。この全文検索インデクスは，文書の形式によって，作成方法が異なります。

ここでは，全文検索の対象になる文書および全文検索インデクスの作成方法について説明します。

4.4.1 全文検索対象文書の作成手順

全文検索の対象となる文書は，次のように登録します。

1. **全文検索機能付き文書クラスを作成します。**
   全文検索機能付き文書クラスの作成方法については，「4.4.2　全文検索機能付き文書クラスの作成」を参照してください。
2. **CdbrDocument クラスまたは CdbrVersionableDocument クラスの CreateObject メソッドをコールして，全文検索が可能なクラスライブラリのオブジェクトを作成します。**
   CreateObject メソッドの引数には，1. で作成した全文検索機能付き文書クラスのクラス識別子を指定します。
   また，このとき，全文検索に必要なデータの構文解析情報を同時に作成するかどうか，指定できます。
   全文検索に必要なデータ（全文検索インデクス）の作成方法については，「4.4.3　全文検索インデクスの作成」を参照してください。

4.4.2 全文検索機能付き文書クラスの作成

ここでは，全文検索の対象になる文書の基になるクラスの作成方法について説明します。

(1) 全文検索機能付き文書クラスとは

全文検索は，全文検索機能を持ったクラスを基に作成された文書に対して実行できます。

全文検索に必要な機能を持ったプロパティを追加した dmaClass_DocVersion クラスのサブクラスを，**全文検索機能付き文書クラス**といいます。

また，全文検索のうち，概念検索の対象になるクラスについては，全文検索用の機能を持ったプロパティのほか，概念検索用の機能を持ったプロパティも追加します。全文検索機能付き文書クラスのうち，概念検索に必要な機能を持ったプロパティを追加した dmaClass_DocVersion クラスのサブクラスを特に，**概念検索機能付き文書クラス**といいます。

クラスライブラリで全文検索の対象になるオブジェクトを作成する場合は，全文検索機能付き文書クラスまたは概念検索機能付き文書クラスを，次に示すクラスライブラリのオブジェクトの構成要素として指定します。

- CdbrVersionableDocument クラス
- CdbrDocument クラス

(2) 全文検索機能付き文書クラスに追加するプロパティ

全文検索機能付き文書クラスを作成する場合には，次のプロパティを追加する必要があります。

- 全文検索インデクス用プロパティ
- edmProp_Score プロパティ
- edmProp_RawScore プロパティ
- edmProp_DocLength プロパティ
- edmProp_ContentIndexStatus プロパティ

プロパティの追加方法については，マニュアル「DocumentBroker Version 3　システム導入・運用ガイド」を参照してください。

次に，全文検索インデクス用プロパティと，edmProp_ContentIndexStatus プロパティについて説明します。

全文検索インデクス用プロパティ

全文検索インデクス用プロパティには，表 4-2 に示す 5 種類があります。作成する文書で実現したい全文検索の種類によって，これらのプロパティを使い分けてください。なお，このマニュアルでは，以降，これらのプロパティとして生成される全文検索用のインデクスを，**全文検索インデクス**と呼びます。

全文検索インデクス用プロパティの種類と，それぞれの全文検索インデクス用プロパティで実行できる全文検索の種類について，次の表に示します。

表 4-2　全文検索インデクス用プロパティの種類

プロパティ	全文検索※	構造指定検索	概念検索
edmProp_TextIndex （プレーンテキスト検索用全文検索インデクス）	○	×	×
edmProp_StIndex （構造指定検索用全文検索インデクス）	○	○	×
edmProp_ConceptTextIndex （プレーンテキスト検索用概念検索インデクス）	○	×	○
edmProp_ConceptStIndex （構造指定検索用概念検索インデクス）	○	○	○
edmProp_Content （Version 1 互換用全文検索インデクス）	○	○	×

（凡例）

○：実行できます。

×：実行できません。

注※

プレーンテキストに対して検索タームを指定して実行する全文検索を示します（構造指定検索および概念検索を含みません）。

edmProp_ContentIndexStatus プロパティ

全文検索インデクスの登録状態を表すプロパティです。Integer32 型のデータが設定されます。このプロパティの値は，DocumentBroker Text Search Index Loader で使用されます。

全文検索インデクスの登録状態ごとに，DocumentBroker によって次の表に示す値が設定されます。

表 4-3　edmProp_ContentIndexStatus プロパティの内容

値	内容
0	文書が未登録です。
1	文書は登録されていますが，全文検索インデクスは作成されていません。

値	内容
2	文書が登録されており，対応する全文検索インデクスも作成されています。
3	文書が更新されていますが，全文検索インデクスが更新されていないため，文書と全文検索インデクスが一致しません。
100 以上	エラー情報など，DocumentBroker Text Search Index Loader が使用する値です。

(3) 概念検索機能付き文書クラスに追加するプロパティ

概念検索機能付き文書クラスを作成する場合には，次のプロパティを追加する必要があります。

- 全文検索インデクス用プロパティ（edmProp_ConceptTextIndex プロパティまたは edmProp_ConceptStIndex プロパティ）
- edmProp_Score プロパティ
- edmProp_RawScore プロパティ
- edmProp_DocLength プロパティ
- edmProp_ContentIndexStatus プロパティ
- edmProp_ScoreConcept プロパティ

プロパティの追加方法については，マニュアル「DocumentBroker Version 3　システム導入・運用ガイド」を参照してください。

4.4.3 全文検索インデクスの作成

ここでは，全文検索インデクスの作成方法について説明します。全文検索インデクスとは，全文検索機能付き文書クラスに追加した全文検索インデクス用プロパティに格納される，全文検索の対象になるテキストデータです。

全文検索インデクスは，どの時点で作成するか，また，どのレンディションタイプを持つ文書に対して作成するかによって，作成方法が異なります。

(1) 全文検索インデクスの作成時点

全文検索インデクスは，次のどちらかの時点で作成，登録できます。それぞれの特長について説明します。

● 文書の登録と同時に全文検索インデクスを作成・登録する

個々の文書の登録ごとに全文検索インデクスを作成します。大量の文書を登録するときにこの方法を利用すると，まとめて作成する場合に比べて処理時間がかかりますが，登録と同時に全文検索が実行できます。

また，新規に作成した文書の全文検索インデクスを，全文検索対象文書全体の全文検索インデクスとは別に，一時的なインデクスとして作成する方法もあります。ここで作成したインデクスを**差分インデクス**といい，差分インデクスを作成する機能を**差分インデクス作成機能**といいます。

差分インデクス作成機能を使用すると，全文検索インデクスの追加登録が高速に処理できます。ただし，個々の検索処理は差分インデクスを使用しない場合に比べて若干遅くなります。また，差分インデクスは，容量が増えた段階で，まとめて全体のインデクスに反映させる必要があります。

なお，マルチファイル管理機能を使用している場合は，文書の登録と同時に全文検索インデクスは作成できません。

● 文書の登録時には全文検索インデクスを作成しないで，あとで複数の文書の全文検索インデクスをまとめて作成・登録する（全文検索インデクスの遅延一括作成）

個々の文書作成時には全文検索インデクスを作成しないで，あとで大量の文書の全文検索インデクスをまとめて作成する方法です。これを，**全文検索インデクスの遅延一括作成**といいます。この機能を使用すると，文書ごとに全文検索インデクスを作成する場合に比べて，処理時間を短縮できます。ただし，

文書登録と同時に全文検索を実行することはできません。

差分インデクスの作成には，HiRDB Text Search Plug-in の差分インデクス作成機能を使用します。差分インデクス作成機能を使用する場合は，インデクス情報ファイルに差分インデクス作成を定義する必要があります。インデクス情報ファイルの定義については，マニュアル「DocumentBroker Version 3　システム導入・運用ガイド」を参照してください。差分インデクス作成機能の詳細については，マニュアル「HiRDB Text Search Plug-in」を参照してください。また，全文検索インデクスの遅延一括作成には，HiRDB のプラグインインデクスの遅延一括作成機能を使用します。全文検索インデクスを遅延一括作成する場合の環境設定および運用方法については，マニュアル「HiRDB　システム運用ガイド」の，プラグインインデクスの遅延一括作成機能についての説明を参照してください。

(2) レンディションタイプと全文検索インデクスの作成方法との対応

DocumentBroker では，全文検索インデクスを持つすべての形式の文書に対して全文検索を実行できます。ただし，全文検索インデクスは，レンディションタイプで表される形式によって，作成方法が異なります。

レンディションタイプごとの全文検索インデクスの作成方法について説明します。

なお，バージョンなし文書およびバージョン付き文書の全文検索インデクスを作成する場合に，文書がマルチレンディション文書である場合には，マスタレンディションのレンディションタイプが対象になります。

● 特定のレンディションタイプを持つテキストファイルの全文検索インデクスの作成

特定のレンディションタイプとは，次のレンディションタイプです。

- 先頭が「MIME::text/」で始まるレンディションタイプ
  例えば，「MIME::text/plain」や「MIME::text/sgml」などです。
- MIME::application/x-edm-sgml

これらのレンディションタイプを持つ文書は，文書の登録時に，登録した文書のコンテントから自動的に全文検索インデクスが作成できます。また，文書登録後に登録されている文書のコンテントから全文検索インデクスを作成したり，文書のコンテント以外のファイルから作成した全文検索インデクスと関連づけたりすることもできます。

● 特定のレンディションタイプを持たない文書の全文検索インデクスの作成

前に示したレンディションタイプ以外の形式を持つ文書です。ビットマップなどのイメージデータや，Word などのアプリケーションプログラムで作成した文書などが該当します。

これらの文書は登録した文書のコンテントから全文検索インデクスを作成できません。この場合は，別に作成した全文検索インデクス作成用のテキストデータから，明示的に全文検索インデクスを作成して，登録した文書と関連づける必要があります。

なお，文書のコンテントから全文検索インデクスを作成した場合，レンディションタイプによって，可能な検索形式が異なります。レンディションタイプと，登録した文書のコンテントから作成した全文検索インデクスで可能な検索形式について，次の表に示します。

表 4-4　全文検索文書として登録されるレンディションタイプと検索形式

レンディションタイプ	検索形式
「MIME::text/」で始まるレンディションタイプ	プレーンテキストの全文検索※
MIME::application/x-edm-sgml	プレーンテキストの全文検索 構造指定検索

注※

ただし，XML インデクスデータ作成機能を使用してインデクスデータを作成，登録した XML 文書に対しては，構造指定検索も実行できます。

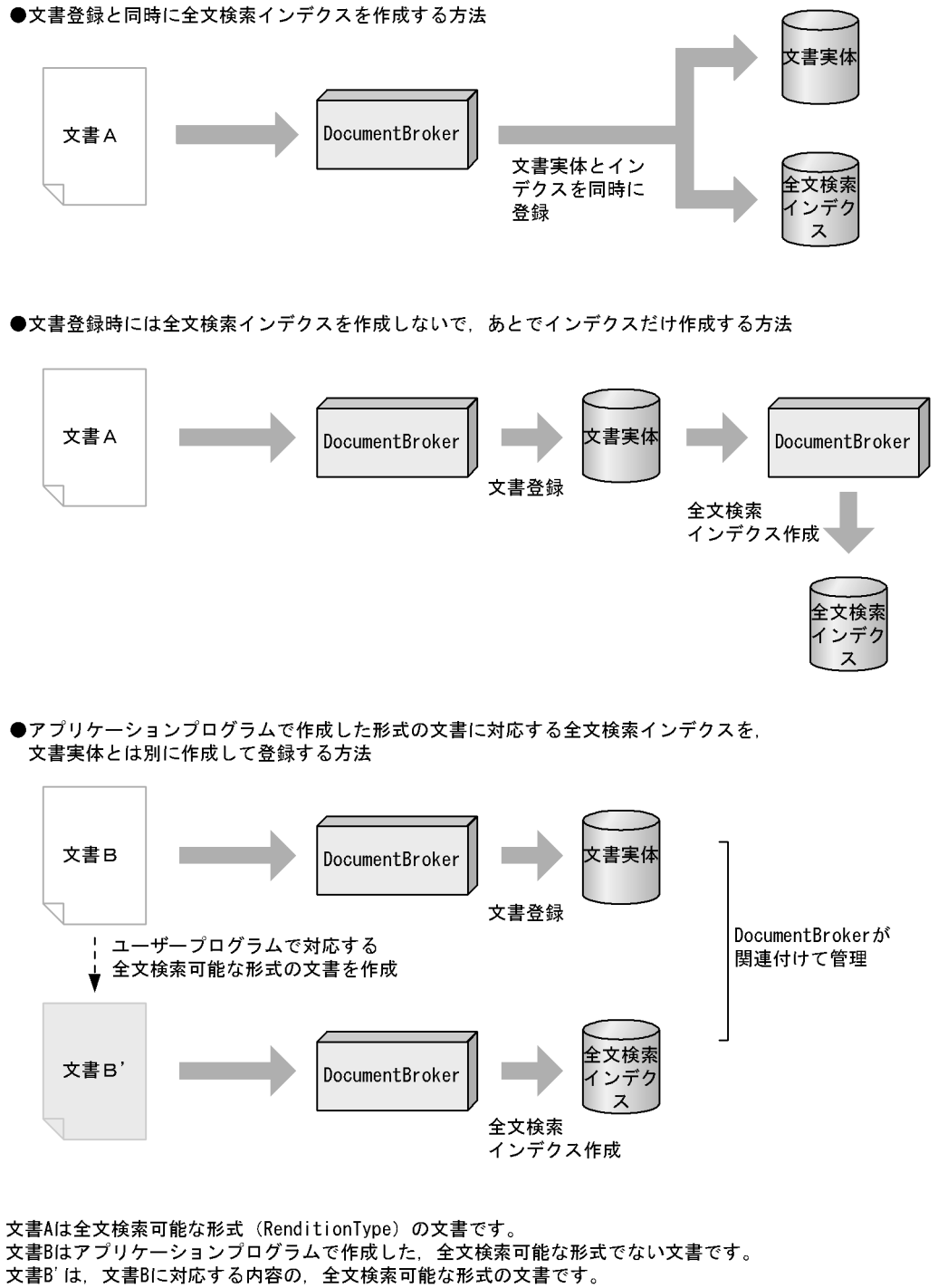
(3) 全文検索インデクスの作成方法

ここでは，作成する時期およびレンディションタイプごとの作成方法を考慮した，全文検索インデクスの作成方法について説明します。作成方法には，作成時期および文書のレンディションタイプに応じて，次の 3 通りの作成方法があります。

- **文書を登録する時に，全文検索インデクスも同時に作成する方法**
  特定のレンディションタイプを持つ文書に対して，文書の作成と同時に全文検索インデクスを作成する方法です。ただし，この方法は，マルチファイル管理機能を使用しているバージョンなし文書およびバージョン付き文書では使用できません。
- **文書を登録する時には全文検索インデクスを作成しないで，全文検索インデクスだけあとから作成する方法**
  特定のレンディションタイプを持つ文書に対して，文書を作成したあとで全文検索インデクスだけ別に作成する方法です。マルチレンディション文書の場合は，特定のレンディションタイプをマスタレンディションのレンディションタイプとして持つ文書が対象になります。なお，オブジェクトのレンディションタイプは，必要に応じて登録後に変更できます。ただし，この方法は，マルチファイル管理機能を使用しているバージョンなし文書およびバージョン付き文書では使用できません。
- **アプリケーションプログラムで作成した形式の文書に対応する全文検索インデクスを，文書とは別に作成して登録する方法**
  特定のレンディションタイプを持たない文書に対して，文書とは別に作成した全文検索インデクスを関連づける方法です。

次の図に，それぞれの方法の概要を示します。

図 4-1 全文検索インデクスの作成方法



文書Aは全文検索可能な形式（RenditionType）の文書です。
文書Bはアプリケーションプログラムで作成した，全文検索可能な形式でない文書です。
文書B'は，文書Bに対応する内容の，全文検索可能な形式の文書です。

それぞれの方法について説明します。

● 文書を登録する時に，全文検索インデクスも同時に作成する方法

文書を作成，登録する時に，CreateObject メソッドの引数に DBR_CREATE_INDEX を指定する方法です。

なお，この場合，次の条件に合致する必要があります。

- 構成要素になる DMA オブジェクトを指定する引数に，全文検索機能付き文書クラスのクラス識別子

を指定している
- レンディションタイプが表 4-4 に示した形式である

なお，UpdateContent メソッドなどによって文書を更新する場合にも，同じように引数を指定することで，同時に全文検索インデクスが作成できます。

● 文書を登録する時には全文検索インデクスを作成しないで，全文検索インデクスだけあとから作成する方法

文書を作成，登録する時に，CreateObject メソッドの引数に DBR_NOT_CREATE_INDEX を指定する方法です。このとき，全文検索インデクスは作成されません。登録済みの文書に対して，全文検索インデクスを作成する CreateIndex メソッドをコールします。

この方法を使用して，プラグインインデクスの遅延一括作成機能によって，全文検索インデクスをバッチ処理で作成することができます。

なお，この場合，全文検索インデクスを作成する文書は，構成要素の DMA オブジェクトが全文検索機能付き文書クラスのオブジェクトであり，かつレンディションタイプが表 4-4 の形式として登録されている必要があります。または，文書を登録するまたは更新する時点で表 4-4 の形式に変更することが必要です。

登録の手順を示します。

1. CreateObject メソッドによって，文書を作成，登録します。文書を更新する場合は，UpdateContent メソッドまたは UpdateContentAndRenditionType メソッドによって，文書を更新します。
2. 1. で登録した文書に対して，CreateIndex メソッドを使用して全文検索インデクスを作成します。

● アプリケーションプログラムで作成した形式の文書に対応する全文検索インデクスを，文書とは別に作成して登録する方法

表 4-4 で示したレンディションタイプ以外の形式を持つ文書に対して，全文検索インデクスを作成したい場合に使用する方法です。アプリケーションプログラムで作成した文書に全文検索インデクスを付けたい場合に使用します。

Word で作成した文書など，表 4-4 で示したレンディションタイプ以外の形式を持つファイルを文書として作成，登録した場合，引数にどの値を指定しても，全文検索インデクスは作成されません。この文書を全文検索の対象にしたい場合は，全文検索インデクスの作成対象とする文書（プレーンテキスト）を別に作成して，登録する必要があります。この場合は，ユーザプログラムによって，Word などのアプリケーションプログラムで作成した文書からテキストデータを抽出してください。

DocumentBroker では，すでに登録されている全文検索インデクスを持たない文書に対して，クライアント環境で作成した全文検索可能なデータを関連づけることができます。また，すでに全文検索インデクスを持つ文書に対して，異なる内容のファイルを利用して，全文検索インデクスを更新することもできます。

登録の手順を示します。ここでは，Word で作成した文書を「文書 B.doc」，これに対応する全文検索インデクスの作成対象とする文書を「文書 B'.txt」とします。

1. Word で作成した形式の文書「文書 B.doc」と，この文書に対応する全文検索可能なレンディションタイプの文書「文書 B'.txt」を作成します。
   この処理は，ユーザプログラムで行ってください。
2.「文書 B.doc」を CreateObject メソッドまたは UpdateContent メソッドなどで DocumentBroker の文書として作成，登録します。
3. 手順 2. で登録した文書に接続して，CreateIndex メソッドをコールします。このとき，引数のファイルパスに file: プロトコルを使用した「文書 B'.txt」のファイルパスを指定します。

これによって，「文書 B'.txt」の内容を基に全文検索インデクスが作成され，「文書 B.doc」の全文検索インデクスとして登録されます。

注意事項

DocumentBroker では，CreateIndex メソッドで登録するファイルの内容とすでに登録されている文書の実体との対応は管理しません。CreateIndex メソッドの引数に指定したファイルの内容と登録されている文書の内容の対応については，ユーザプログラムで管理してください。

4.4.4 全文検索インデクスの削除

全文検索インデクスは，RemoveIndex メソッドによって削除できます。このとき，文書そのものは削除されません。文書も同時に削除する場合は，RemoveObject メソッドをコールしてください。

4.4.5 注意事項

次のメソッドをコールすると，メソッドの処理を終了する時に，RD エリア全体に排他ロックが設定されます。

● 全文検索インデクスを作成・削除するメソッド
CdbrDocument クラスおよび CdbrVersionableDocument クラスの次のメソッドに該当します。
- CreateIndex メソッド
- RemoveIndex メソッド

このため，これらのメソッドをコールしたあとには，CdbrSession::Commit メソッドをコールしてください。CdbrSession::Commit メソッドをコールしないで他テーブルに対して検索メソッドなどをコールすると，デッドロックが発生する可能性が高くなります。

4.5 edmSQL の文法

この節では，edmSQL の文法について説明します。

edmSQL で使用する文字列の長さやデータ型などは，データベースの制限に従います。DocumentBroker で使用できるデータベースは，HiRDB です。

HiRDB の SQL で指定できる文字列の長さやデータ型の制限については，マニュアル「HiRDB　SQL リファレンス」を参照してください。

4.5.1 edmSQL の文法の概要

ここでは，edmSQL の文法の概要として，次の項目について説明します。

- データ型
- 字句規則
- 構文規則

(1) データ型

edmSQL では，クラスライブラリで扱えるデータ型のうち，Id 型を除いたデータ型が使用できます。また，HiRDB Text Search Plug-in と連携した検索の検索条件などには，クラスライブラリでは扱えない Binary 型のデータも扱えます。Binary 型のデータは，クラスライブラリでは String 型の値として取得できます。

なお，それぞれのデータ型ごとに，表現形式や使用できる演算子・関数・述語などが決まっています。

データ型と演算子，関数および述語の関係については，「4.5.3　使用できるデータ型と演算子・関数・述語の関係」を参照してください。

(2) 字句規則

edmSQL では，使用できる字句が定義されています。

定義されているのは，< 区切り文字 >，< トークン >，< キーワード >，< リテラル >，< 識別子 >，< 名前 >，<ID 文字列 >，< 特殊なプロパティ >，<? パラメタ > および <OIID> です。

字句規則については，「4.5.4　字句規則」を参照してください。

(3) 構文規則

edmSQL 文で指定できる構文は，SELECT 文です。したがって，edmSQL では，SELECT 文の構文規則が定義されています。

(a) 構文の概要

● 検索の実行単位

検索は，edmSQL プログラム単位で実行されます。edmSQL プログラムは，一つの edmSQL 文によって構成されています。edmSQL プログラムは，1 回の CdbrEqlStatement::Set メソッドのコールで設定できる検索条件です。
edmSQL 文に指定できるのは，問い合わせ文（SELECT 文）だけです。
検索の実行単位については，「4.5.5　検索の実行単位の構文規則」を参照してください。

● 問い合わせ式（SELECT 文）

問い合わせは，問い合わせ文によって表現します。
問い合わせ文には，SELECT 句，FROM 句，WHERE 句，GROUP BY 句，HAVING 句および ORDER BY 句が指定できます。最も単純な問い合わせ文は，SELECT 句と FROM 句から構成されます。
それぞれの句は，次のような要素で構成されています。

- 値
- 述語
- 関数
- データ操作

問い合わせ表現については，「4.5.6　問い合わせ式の構文規則」を参照してください。

(b) 構文の詳細

SELECT 文は，次のような要素で構成されます。

● 値（スカラー式）
値は，スカラー式として指定します。スカラー式によって，プロパティの指定やルーチンの起動，数値関数および集合関数によって得られる値を表現します。
値を表現するスカラー式については，「4.5.7　スカラー式表現の構文規則」を参照してください。

● 述語
FROM 句の ON 条件，WHERE 句，または HAVING 句に指定する検索条件は，述語によって表現します。述語では，比較述語，論理述語，Between 述語，In 述語，Like 述語，Null 述語および Exists 述語で構成される検索条件が指定できます。
述語の結果は，「真」，「偽」または「不定」のどれかで得られます。
述語の表現については，「4.5.8　述語の構文規則」を参照してください。

● 関数
DocumentBroker では，基本的な SQL の文法では表現できない edmSQL での拡張機能を，関数で表現します。
関数には，HiRDB Text Search Plug-in の機能を使用した検索を実現する関数と，DocumentBroker で扱うオブジェクトや文字列の変換機能を実現する関数があります。
関数の表現については，「4.5.9　関数指定の構文規則」を参照してください。

● データ操作
edmSQL で実行できるデータ操作とは，ORDER BY 句による検索結果の並び替えです。
データ操作の表現については，「4.5.10　データ操作の構文規則」を参照してください。

4.5.2　表記規則

ここでは，edmSQL の文法を説明する際に使用する文法規則について説明します。

edmSQL の構文規則は，BNF 表記を使用して説明します。

BNF 表記について，次に示します。

● <a> ::= <b><c> は，構文要素 <a> は，構文要素 <b> および <c> から構成されることを意味します。

● 特殊記号およびキーワードは，終端記号であるため，このマニュアルに記載してあるとおりに記述する必要があります。「<」と「>」で囲まれた構文要素は，終端記号ではない構文要素です。

●「|」（ストローク）は，選択肢の分割を意味します。A | B | C の場合は，選択肢として，A，B，または C を選択することを意味します。

●「〔」と「〕」で囲まれた要素は，オプションであり，省略できます。

●「{」と「}」で囲まれている要素は，その内部の「|」で区切られた要素の中からどれか一つを選択することを意味します。

●「...」は，直前の構文要素の 1 回以上の繰り返しを意味します。

●「!!」は，注釈の開始を意味します。注釈は改行で終わります。

4.5.3 使用できるデータ型と演算子・関数・述語の関係

ここでは，edmSQL 文で使用できるデータ型と演算子・関数・述語との関係について説明します。

(1) edmSQL で使用できるデータ型

edmSQL 文で使用できるデータ型は，次のとおりです。

● Boolean 型

● Integer32 型

● String 型

● Object 型

● Binary 型

なお，Id 型は，edmSQL のデータ型としては提供しませんが，クラスやプロパティを ID で指定することはできます。

使用できるデータ型を，次の項目ごとに説明します。

● edmSQL 文の表現形式

● 演算子

● 関数

● 述語

● クラスライブラリのデータ型との値の受け渡しのインタフェース

(2) Boolean 型

Boolean 型は，値と評価の二つの概念を持ちます。それぞれに対して，TRUE，FALSE，UNKNOWN の三つの表現があります。

値としての Boolean 型

値として扱う Boolean 型のデータ型は，物理的実体として存在します。これは，Boolean 型のプロパティの値または定数値として表現します。
値には，TRUE，FALSE，UNKNOWN の三つの値があります。表現と評価の定数値は，SQL の概念にない，edmSQL で導入される概念で対応付けられます。
値としての Boolean 型の表現と定数値の対応を，次の表に示します。

表 4-5 値としての Boolean 型の表現と定数値の対応

表現	定数値
TRUE	1

表現	定数値
FALSE	0
UNKNOWN	2

評価としての Boolean 型

評価として扱う Boolean 型のデータ型は，物理的な実体を持ちません。検索条件や検索関数の評価結果として返ってくるデータ型です。

評価には，真，偽，不定の三つの値があります。表現と評価の定数値は，SQL と同じ概念で対応付けられます。

評価としての Boolean 型の表現と定数値の対応を，次の表に示します。

表 4-6　評価としての Boolean 型の表現と定数値の対応

表現	評価の値
TRUE	真
FALSE	偽
UNKNOWN	不定

次に，Boolean 型の値または評価を使用する表現形式，演算子，関数および述語について説明します。また，クラスライブラリの値との対応について説明します。

(a) 表現形式

値として扱う Boolean 型の表現形式を次の表に示します。

表 4-7　Boolean 型の値の表現形式

値	定義
リテラル	TRUE FALSE UNKNOWN
プロパティ	Boolean 型のプロパティ 格納値：0，1，2
変数として指定できる値	? パラメタ

Boolean 型の評価には，直接の表現形式はありません。

(b) 演算子・関数

Boolean 型の値を被演算子として扱う演算子・関数はありません。

Boolean 型の評価を被演算子として扱う演算子を，次の表に示します。

表 4-8　Boolean 型の評価を被演算子として扱う演算子

演算子の種類	演算子
論理演算子	AND OR NOT

Boolean 型の値を被演算子として扱う演算子・関数はありません。

(c) 述語

Boolean 型の値

評価対象が，Boolean 型の値である述語を次の表に示します。

表 4-9 評価対象が Boolean 型の値である述語

述語の種類	述語
比較述語	= <>
In 述語	IN

Boolean 型の評価

評価対象が，Boolean 型の評価である述語を次の表に示します。

表 4-10 評価対象が Boolean 型の評価である述語

述語の種類	述語
論理述語	IS TRUE

(d) クラスライブラリのデータ型との対応

Boolean 型の値と，クラスライブラリのデータ型との対応を，次の表に示します。

表 4-11 Boolean 型の値とクラスライブラリのデータ型との対応

データ型	値
DmaBoolean 型	DMA_TRUE（=1） DMA_FALSE（=0） DMA_UNKNOWN（=2）

Boolean 型の評価に，クラスライブラリのデータ型と対応するものはありません。

(3) Integer32 型

Integer32 型で表現するのは，4 バイト（32 ビット）で表現できる符号付き整数値，または符号と数字文字列で表現する，整数リテラルです。

次に，Integer32 型のデータ型を使用する表現形式，演算子，関数および述語について説明します。また，クラスライブラリとの値との対応についても説明します。

(a) 表現形式

Integer32 型の値の表現形式を次の表に示します。

表 4-12 Integer32 型の値の表現形式

値の種類	定義
リテラル	符号および数字列
プロパティ	Integer32 型のプロパティ 格納値：-2,147,483,648 ～ 2,147,483,647
変数として指定できる値	? パラメタ

(b) 演算子・関数

Integer32 型の値を被演算子または引数として扱う演算子および関数を次の表に示します。

表 4-13 Integer32 型の値を被演算子または引数とする演算子・関数

演算子・関数の種類	演算子・関数
整数の四則演算子	+ - * /
単項演算子	+ -
数値関数	ABS()

(c) 述語

評価対象が Integer32 型の値である述語を次の表に示します。

表 4-14 Integer32 型の値を評価対象とする述語

述語の種類	述語
比較述語	= < > <= >= <>
In 述語	IN
Between 述語	BETWEEN

(d) クラスライブラリのデータ型との対応

クラスライブラリのデータ型との対応を次の表に示します。

表 4-15 Integer32 型の値とクラスライブラリのデータ型との対応

データ型	値
DmaInteger32 型	-2,147,483,648 ～ 2,147,483,647

(4) String 型

String 型で表現するのは，DocumentBroker で使用できる文字コードセットから構成される任意の長さの文字列です。DocumentBroker で使用できる文字コードセットについては，「4.5.4(1) 文字コードセットとの対応」を参照してください。

次に，String 型のデータ型を使用する表現形式，演算子，関数および述語について説明します。また，クラスライブラリとの値との対応についても説明します。

(a) 表現形式

String 型の値の表現形式を，次の表に示します。

表 4-16 String 型の値の表現形式

値の種類	定義
リテラル	' 任意の文字列 ' 詳細は，「4.5.4(6) ＜リテラル＞」を参照してください。
プロパティ	String 型のプロパティ 格納値： 定義長（dmaProp_MaximumLengthString の値）以下の文字列
変数として指定できる値	? パラメタ

(b) 演算子・関数

String 型の値を被演算子または引数とする，演算子を次の表に示します。

表 4-17 String 型の値を被演算子または引数とする演算子

演算子の種類	演算子
文字列結合演算子	||

(c) 述語

評価対象が String 型の値である述語を次の表に示します。

表 4-18 String 型の値を評価対象とする述語

述語の種類	述語
比較述語	= < > <= >= <>
In 述語	IN
Between 述語	BETWEEN
Like 述語	LIKE XLIKE

(d) クラスライブラリのデータ型との対応

クラスライブラリのデータ型との対応を次の表に示します。

表 4-19 String 型の値とクラスライブラリのデータ型との対応

データ型	値
pDmaString_T 型	4,294,967,295 バイト以下の文字列

(5) Object 型

Object 型の値には，次の 2 種類があります。

- VariableArray 型プロパティの値
- DMA オブジェクトを識別するためのオブジェクトリファレンスの値

オブジェクトリファレンスとは，DMA オブジェクトへのリファレンスを示す Object 型プロパティの値です。例えば，dmaProp_ParentContainer プロパティの値がこれに当たります。

Object 型の値のうち，オブジェクトリファレンスは検索条件に直接指定できません。検索で指定する場合には，オブジェクトリファレンスが示す実体である DMA オブジェクトの OIID 文字列を指定します。例えば，dmaProp_ParentContainer プロパティの値を指定したい場合，検索条件には dmaProp_ParentContainer プロパティが参照している実体の DMA オブジェクト（Container オブジェクトなど）の OIID を指定します。

この OIID 文字列の値は，検索実行時には，次のどちらかの方法でオブジェクトリファレンスである Object 型プロパティの値に変換する必要があります。

● **objref 関数で変換する。**
edmSQL 文にオブジェクトリファレンスの実体である OIID を直接指定する場合の変換方法です。
objref 関数の詳細については，「4.5.9(2)(d)　<objref 関数 >」を参照してください。

● **CdbrEqlStatement::SetObjParam メソッドで変換する。**
edmSQL 文に <? パラメタ > を指定しておく場合の変換方法です。SetObjParam メソッドの引数として，OIID 文字列を指定します。
CdbrEqlStatement::SetObjParam メソッドについては，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。

また，検索結果として Object 型の値を取得した場合は，oiidstr 関数を使用して，その Object 型の値が表すオブジェクトリファレンスの実体であるオブジェクトの OIID が取得できます。oiidstr 関数の詳細については，「4.5.9(2)(c)　<oiidstr 関数 >」を参照してください。

次に，Object 型のデータ型を使用する表現形式および述語について説明します。また，クラスライブラリとのデータ型との対応についても説明します。Object 型の値を使用する演算子および関数はありません。

(a) 表現形式

Object 型の値の表現形式を，次の表に示します。

表 4-20　Object 型データの表現形式

値の種類	定義
プロパティ	Object 型のプロパティ（VariableArray 型）

(b) 述語

評価対象が Object 型の値である述語を次の表に示します。

表 4-21　Object 型の値を評価対象とする述語

述語の種類	述語
比較述語	=
In 述語	IN

なお，比較述語，IN 述語では，オブジェクトリファレンス同士を比較します。

(c) クラスライブラリのデータ型との対応

クラスライブラリのデータ型との対応を次の表に示します。

表 4-22 Object 型の値とクラスライブラリのデータ型との対応

データ型	値
CdbrVariableArray 型オブジェクト （基本単位が VariableArray 型）	次のように定義されているプロパティの値としてだけ対応します。 • dmaProp_DataType = DMA_DATATYPE_OBJECT • dmaProp_Cardinality=255

Object 型のデータ型の値として VariableArray 型プロパティ以外の値（オブジェクトリファレンス）を扱う場合は，< 変換関数 > を使用するか，? パラメタを使用してください。

（6） Binary 型

Binary 型で表現するのは，任意のバイト列を表現するデータです。主に，文書のコンテンツを表現します。関数の引数や，戻り値に指定する型としてだけ使用できます。プロパティの型としては定義できません。

Binary 型が使用できる関数についての詳細は，「4.5.9 関数指定の構文規則」を参照してください。

（a） 表現形式

Binary 型のデータの表現形式を，次の表に示します。

表 4-23 Binary 型のデータの表現形式

値の種類	定義
変数	? パラメタ

Binary 型のデータは，? パラメタ以外の形式では表現できません。

（b） クラスライブラリのデータ型との対応

クラスライブラリのデータ型との対応を，次の表に示します。

表 4-24 Binary 型のデータとクラスライブラリのデータ型との対応

データ型	値
pDmaString_T 型	4,294,967,295 バイト以下の文字列※

注※

Binary 型のデータはクラスライブラリでは String 型の値として取得できます。

4.5.4 字句規則

ここでは，edmSQL で使用する字句に関する規則について説明します。

（1） 文字コードセットとの対応

ここでは，edmSQL で使用できる字句と，文字コードセットとの対応について説明します。

● 各国語を表記するための文字などは，文字コードセットに依存します。
edmSQL で使用できる文字コードセットは，前提プログラムである，OS と HiRDB に依存します。したがって，これらのプログラムで使用できない文字を使用した場合は，前提プログラムのエラーになります。

● edmSQL で使用する字句の規則は，検索の対象になる文書のコンテントには適用されません。全文検索機能を使用する場合に検索対象になる文書の字句については，全文検索機能を提供する HiRDB Text Search Plug-in に依存します。

● 次の文字コードセットが使用できます。
- DocumentBroker のメタ定義で使用できる文字コードセット
- HiRDB で使用できる文字コードセット
- HiRDB Text Search Plug-in で使用できる文字コードセット

(2) edmSQL で使用できる文字

edmSQL で使用できる文字を，次の表に示します。

表 4-25 edmSQL で使用できる文字

文字	値
英大文字	A ～ Z
英小文字	a ～ z
数字	0 ～ 9
16 進数字	0 ～ 9，a ～ f，および A ～ F
特殊文字（半角文字コード）	< 空白 > < ダブルクォート >（"） < パーセント >（%） < アンパサンド >（&） < シングルクォート >（'） < 左括弧 >（(） < 右括弧 >（)） < アスタリスク >（*） <+ 符号 >（+） < コンマ >（,） <- 符号 >（-） < ピリオド >（.） < 斜線 >（/） < コロン >（:） < セミコロン >（;） < 小なり演算子 >（<） < 等号演算子 >（=） < 大なり演算子 >（>） < 疑問符 >（?） < 左角括弧または trigraph >（[または (??） < 右角括弧または trigraph >（] または ??)） < サーカムフレックス >（^） < 下線文字 >（_） < 垂直棒 >（|） < 左中括弧 >（{） < 右中括弧 >（}）

(a) 規約詳細

edmSQL で使用できる文字の詳細について説明します。

● < 空白 > は，JIS X 0201 では，0x20 に対応します。

● < 拡張文字 > とは，使用できる文字コードセットに対応して，使用可能な文字として拡張した文字です。使用できる文字コードセットについては，「(1) 文字コードセットとの対応」を参照してください。

(3) < 区切り文字 >

< トークン > は，< 区切り文字 > によって分割されます。< 区切り文字 > は，< トークン > を抽出するための字句解析で削除される文字です。このため，構文解析を実行する時点では，< 区切り文字 > は出現しません。

(a) < 区切り文字 > として指定できる字句

< 区切り文字 > として指定できる字句は，<white space> です。

(b) 規約詳細

< 区切り文字 > の詳細について説明します。

edmSQL では，次の定義を <white　space> として定義します。

```
Horizontal Tab      [0x09]
Line Feed          [0x0A]
Vertical Tabulation  [0x0B]
Form Feed          [0x0C]
Carriage Return     [0x0D]
Space             [0x20]
```

[　] 内は，対応する ASCII コードです。

(4) < トークン >

< トークン > の定義について説明します。< トークン > は，構文の基本要素です。

< トークン > の定義には，次の要素の定義が含まれます。

- < キーワード >
- < リテラル > の一部
- < 識別子 >

それぞれの要素については，「(5)< キーワード >」，「(6)< リテラル >」および「(7)< 識別子 >」を参照してください。

なお，< トークン > として指定できる文字列の長さなどは，データベースに依存します。

(a) < トークン > として指定できる字句

< トークン > として指定できる字句を，次の表に示します。

表 4-26　< トークン > として指定できる字句

種類	値	参照先
正規識別子	先頭が < 英字 > で，先頭以外が < 英字 >，< 数字 > および < 下線文字 > の値	(7)< 識別子 >
キーワード	< 予約されたキーワード > および < 予約されないキーワード > の値	(5)< キーワード >
符号なし数値リテラル	符号なしの数字の値	(6)< リテラル >
バイナリ長トークン	< 符号なし数値 > に <multiplier>（k，m，g）が付いた値	「4.5.9　関数指定の構文規則」の AS< データ型指定 >

種類	値	参照先
区切られた識別子	< ダブルクォート > で囲まれた値 (< 区切り文字 >, < ダブルクォート以外の文字 > および < 二重ダブルクォート > を含むことができる)。	(7)< 識別子 >
文字列リテラル	< シングルクォート > で囲まれた値 (< シングルクォート以外の文字 > または < 二重シングルクォート >('') が指定できる)。	(6)< リテラル >
特殊文字	edmSQL で使用できる特殊文字	(2)edmSQL で使用できる文字
そのほかの文字	• < 不等号演算子 >(<>) • < 大なり等号演算子 >(>=) • < 小なり等号演算子 >(<=) • < 文字列連結演算子 >(||) • < 右矢印演算子 >(->) • < 二重コロン >(::) • < 左大括弧 >([) • < 右大括弧 >(]) • < アンパサンド >(&)	−

(凡例)

−:該当しません。

(b) 規約詳細

< トークン > の詳細について説明します。

< トークン > が区切られる単位についての詳細

- < トークン > で指定できる要素のうち,< 正規識別子 >,< キーワード >,< 符号なし数値リテラル > および < バイナリ長トークン > は,< トークン > を区切る要素として扱われます。これ以外の < トークン > を指定する場合は,前後に < 区切り文字 > または区切る要素になる < トークン > を指定して,前後との区切りを明確にしてください。区切る要素を指定しないで指定した場合,複数の < トークン > が一つの < トークン > として扱われたり,字句解析エラーになったりします。
  < トークン > の表記例と edmSQL 検索実行時の値の対応を,次の表に示します。

表 4-27 トークンの表記例と edmSQL 検索実行時の値の対応

トークンの表記例	edmSQL 検索実行時の値
ABC □ 123	正規識別子 ABC と数値リテラル 123
ABC123	正規識別子 ABC123
ABC'123'	正規識別子 ABC と文字列リテラル '123'
123ABC	正規識別子の先頭に数字は指定できないため,字句解析エラーになります。

(凡例)

□:区切り文字

- < トークン > と < トークン > の間には任意の個数の区切り文字を指定できます。
  例えば,□を区切り文字とした場合,「prop □□□ = □□□□□□□□□ 10」は,「prop=10」と同じ意味になります。また,「COUNT □□□□□ (□□□ * □□)」は,「COUNT(*)」と同じ意味になります。
- < トークン > の中に区切り文字を指定することはできません。ただし,< シングルクォート > に囲まれた文字列リテラルや < ダブルクォート > に囲まれた区切られた識別子の中には,区切り文字を指定できます。

例えば，□を区切り文字とした場合，「< □ =」は，演算子「<=」ではなく，二つの演算子「<」と「=」と識別されるため，構文解析エラーになります。また，「'< □ ='」は，文字列定数「'< □ ='」と識別されるため，エラーにはなりません。

< ダブルクォート >< ダブルクォート以外の文字 >< 二重ダブルクォート > に関する詳細

- < ダブルクォート以外の文字 > は，使用できる文字コードセットの文字から，< ダブルクォート > を除外したものです。edmSQL で使用できる文字で構成されている必要はありません。
  ただし，DocumentBroker のメタ定義で使用できる文字および HiRDB で使用できる文字以外は指定できません。
- < 二重ダブルクォート > は，< 区切られた識別子 > 内で < ダブルクォート > を指定する場合に使用します。

< バイナリ長トークン > についての詳細

- < バイナリ長トークン > は，Binary 型の定義長を指定するためのトークンです。関数の引数に指定する ? パラメタに対して，データ型を明示するための AS< データ型指定 > で使用します。

使用できる < 特殊文字 > についての詳細

- 次に示す < 特殊文字 > および演算子は，文字列リテラルの字句としてだけ使用できます。それ以外で使用した場合，字句解析時にエラーになります。
  % & : (?? ??) ^ | { } -> ::

(5) < キーワード >

< キーワード > の定義について説明します。< キーワード > には < 予約されるキーワード >（予約語）と < 予約されないキーワード > があります。

< キーワード > では，大文字と小文字は区別されません。

ただし，このマニュアルの例では，< 予約されるキーワード > を大文字で，< 予約されないキーワード > を小文字で表記します。

edmSQL のキーワードは SQL のキーワードを基盤として，次のように定義されています。

- edmSQL では，SQL のキーワードおよび HiRDB のキーワードのうち，最小限のものをキーワードとして定義しています。すべての SQL のキーワードおよび HiRDB のキーワードが定義されているのではありません。ただし，ユーザが識別子を指定する場合には，edmSQL のキーワードのほか，HiRDB のキーワードも指定できません。識別子として HiRDB のキーワードを指定した場合，HiRDB のエラーになります。なお，クラス名やプロパティ名については，定義する時点で HiRDB のキーワードは指定できません。このため，登録済みのクラス名やプロパティ名を edmSQL のキーワードとして指定した場合に，HiRDB のエラーになることはありません。
- edmSQL では，全文検索関数などの関数名は < 予約されないキーワード > として定義されています。そのほかのキーワードは < 予約されるキーワード > として定義されています。
- DMA の検索モデルで定義されている演算子のうち，キーワードとして表現されているものは，edmSQL では使用しないものも含めてすべて < 予約されるキーワード > として定義されています。
- DocumentBroker の文書管理モデルで使用する用語として，幾つかのキーワードが < 予約されるキーワード > として定義されています。

(a) < キーワード > として指定できる字句

< キーワード > として指定できる字句を，次の表に示します。

表 4-28 < キーワード > として指定できる字句

種類	値	特徴
関数名	concept_with_score contains cotains_with_score extracts objref oiid oiidstr score score_concept	予約されないキーワードです。
multiplier	k m g	
句に使用するキーワード	SELECT FROM WHERE ORDER GROUP BY HAVING	予約されるキーワードです。
述語に使用するキーワード	IS ALL SOME ANY IN LIKE XLIKE EXISTS BETWEEN AS ESCAPE	
論理演算に使用するキーワード	NOT AND OR	
リテラルに使用するキーワード	NULL TRUE FALSE UNKNOWN	
結合条件に使用するキーワード	JOIN INNER OUTER LEFT ON	
データ操作および重複排除に使用するキーワード	DISTINCT ASC DESC	
集合関数で使用するキーワード	COUNT	
数値関数で使用するキーワード	ABS	
DMA の検索モデルで使用するキーワード	ID	
プロパティのデータ型で使用するキーワード	BINARY BOOL BOOLEAN INT INTEGER STRING	

種類	値	特徴
DocumentBroker の文書モデルで使用するキーワード	ACL CARDINALITY CLASS CONTAINER CONTENT DOCUMENT EVERYONE FOLDER FOR GROUP LINK LOCK OBJECT OF OWNER PARENT PRIVATE PROPERTY PUBLIC READ TO TYPE UNLINK UPDATE VARRAY VERSION VERSIONABLE WRITE	

(b) 規約詳細

< キーワード > の詳細について説明します。

< キーワード > の表記についての詳細

キーワードでは，大文字と小文字が区別されません。正規識別子の規約とは異なりますので，ご注意ください。edmSQL では，キーワードはすべて大文字として管理されます。

例えば，次の表記は，すべて < 予約されるキーワード >「SELECT」として識別されます。

```
Select  SeLect  select  selecT
```

また，次の表記はすべて < 予約されないキーワード >「contains_with_score」として識別されます。

```
Contains_With_SCORE  contains_with_scorE
CONTAINS_WITH_SCORE
```

< キーワード > と < 正規識別子 > の関係についての詳細

- < 予約されるキーワード > は正規識別子として使用できません。
- < 予約されないキーワード > は正規識別子として使用できますが，HiRDB で予約語として定義されている場合は，使用できません。使用すると，データベースエラーになります。

(6) < リテラル >

< リテラル > の定義について説明します。< リテラル > とは，NULL 値以外の値です。

< リテラル > には，次の種類があります。

- 文字列リテラル
- 数値リテラル
- 論理リテラル

(a) < リテラル > として指定できる字句

< リテラル > として指定できる字句を，次の表に示します。

表 4-29 < リテラル > として指定できる字句

種類	値
文字列リテラル	< シングルクォート > で囲まれた文字列 （文字列には，< シングルクォート以外の文字 >※または < 二重シングルクォート >（"）が指定できる）。
数値リテラル	符号（+ または）付きの数字 （符号は省略できる）。
論理リテラル	TRUE FALSE UNKNOWN

注※

< シングルクォート以外の文字 > は，使用できる文字コードセットの文字から，< シングルクォート > を除外したものです。edmSQL で使用できる文字で構成される必要はありません。ただし，文字列リテラルは String 型の値であるため，String 型を格納する HiRDB のデータ型（MVARCHAR 型）で制限される文字は使用できません。

(b) 規約詳細

< リテラル > の詳細について説明します。

文字列リテラルの詳細

文字列リテラルは，任意の文字列定数を表現する場合に使用します。文字列リテラルは，String 型の値に対応します。したがって，文字列リテラルで使用できる文字列の長さは，データベースの制限に従います。データベースの制限以上の長さの文字列を文字列リテラルとして指定したい場合は，? パラメタを使用してください。

文字列リテラルの表記例と edmSQL 検索実行時の値の対応を，次の表に示します。なお，文字列リテラルを表す String 型の値は，HiRDB では MVARCHAR 型のデータとして格納されます。

表 4-30 文字列リテラルの表記例と edmSQL 検索実行時の値の対応

文字列リテラルの表記例	値の扱われ方	edmSQL 検索実行時の値
'aaa'	文字列データ	aaa
'124'	文字列データ	124
'abc あいうえお '	混在文字列データ	abc あいうえお
'ab"c あ " いうえお '	シングルクォートのエスケープ	ab'c あ ' いうえお

数値リテラル（符号付き数値リテラル）の詳細

数値リテラルは，任意の数値定数を表現する場合に使用します。なお，DocumentBroker で使用できる数値リテラルは，4 バイトの整数値だけです。

なお，4 バイトの整数値の有効値は，-2,147,483,648 ～ 2,147,483,647 までの値です。この範囲外の値を指定した場合，データベースの制限に従って，目的の検索が実行できなかったり，データベースのエラーになったりします。

数値リテラルの表記例と edmSQL 検索実行時の値の対応を，次の表に示します。

表 4-31 数値リテラルの表記例と edmSQL 検索実行時の値の対応

数値リテラルの表記例	値の扱われ方	edmSQL 検索実行時の値
123	数値データ	123
0000000123	数値データ	123
+1234567890	数値データ（正の整数値）	1234567890
-1234567890	数値データ（負の整数値）	-1234567890
987654321000000000	リテラルとしては正しい表記ですが，4 バイト整数値の有効値ではないため，正しく扱われません。	－

（凡例）

－：該当する値がありません。

論理リテラルの詳細

論理リテラルは，論理値である「真」,「偽」および「不定」の三つの値を，それぞれ「TRUE」,「FALSE」および「UNKNOWN」で表現します。

論理型の値を表す Boolean 型のプロパティの値は，TRUE は「1」，FALSE は「0」，UNKNOWN は「2」として，数値で格納されます。ただし，この値を edmSQL で数値として扱うことはできません。edmSQL で指定する場合は，必ず TRUE，FALSE または UNKNOWN として表現してください。

また，論理リテラルを整数の演算に使用する値として使用することもできません。ただし，クラスライブラリを使用して，C++ のデータモデルに変換した場合は，論理リテラルの値を数値として扱うこともできます。この場合の数値のデータベースでの扱われ方は，データベースでのデータモデルの対応付けに依存します。

論理リテラルの表記ごとの，論理値を表す TRUE の扱われ方について，例に示します。

- 次の TRUE はデータベースによって，論理述語として処理されます。

```
contains(edmProp_Content,'文章[概要{"コンピュータ"}]') IS TRUE
```

- 次の TRUE はクラスライブラリを通して数値として処理されます。ただし，myProp_OK は Boolean 型のプロパティです。

```
myProp_OK = TRUE
```

(7) < 識別子 >

< 識別子 > の定義について説明します。< 識別子 > には，< 正規識別子 > と < 区切られた識別子 > があります。

なお，識別子には，日本語の文字コードセットは使用できません。

(a) < 識別子 > として指定できる字句

< 識別子 > として指定できる字句を，次の表に示します。

表 4-32 < 識別子 > として指定できる字句

種類	値
正規識別子	先頭は < 英字 > で，先頭以外が < 英字 >，< 数字 > および < 下線文字 > の値 例 usrClass_ABC

種類	値
区切られた識別子	< ダブルクォート > で囲まれた値 （< 区切り文字 >，< ダブルクォート以外の文字 >※または < 二重ダブルクォート >（""）を含むことができる） 例 "user class" "user""class"

注※

< ダブルクォート以外の文字 > として指定できるのは，英字，数字および「"」を除いた特殊文字です。

（b）規約詳細

< 識別子 > の詳細について説明します。

< 正規識別子 > に関する詳細

- edmSQL では，< 正規識別子 > に使用される英字の < 英大文字 > と < 英小文字 > が区別して扱われます。このため，SQL を使用する時のように，< 英大文字 > と < 英小文字 > を区別するために，< 区切られた識別子 > を使用する必要はありません。
- < 正規識別子 > に < キーワード > は使用できません。< キーワード > と同じ識別子を使用する場合は，< 区切られた識別子 > として指定してください。

< 区切られた識別子 > に関する詳細

- 値が空の区切られた識別子「""」は，無効な識別子として扱われます。これを指定した場合は，字句解析エラーになります。

（8）< 名前 >

< 名前 > の定義について説明します。< 名前 > では，クラス名，プロパティ名，相関名および関数名を表現します。

< 名前 > の基本要素は，< 識別子 > と <ID 文字列 > です。

（a）< 名前 > として指定できる字句

< 名前 > として指定できる字句を，次の表に示します。

表 4-33 < 名前 > として指定できる字句

種類	値	参照先
クラス名	• < 識別子 > • <ID 文字列 >	• (7)< 識別子 > • (9)<ID 文字列 >
プロパティ名	• < 識別子 > • <ID 文字列 > • < 特殊なプロパティ >	• (7)< 識別子 > • (9)<ID 文字列 > • (10)< 特殊なプロパティ >

種類	値	参照先
関数名	（変換関数） • oiidstr • objref • oiid （全文検索関数） • contains • contains_with_score • score • extracts （概念検索関数） • concept_with_score • score_concept	• 4.5.9　関数指定の構文規則
相関名	< 識別子 >	• (7)< 識別子 >

(b) 規約詳細

< 名前 > の詳細について説明します。

クラス名の詳細

< クラス名 > は，文書空間内でクラスを識別するための名称（dmaProp_DisplayName の値），または GUID 値の ID 文字列表現です。使用できる文字列の長さは，データベースのテーブル名に使用できる文字列の長さに従います。

一つの edmSQL 文の中で，同じクラスを識別するための < クラス名 > の表現に，名称と ID 文字列を混用することはできません。一つの edmSQL 文では，< クラス名 > は，名称または ID 文字列で統一して表記してください。例えば，<FROM 句 > の < クラス名 > に ID 文字列を指定した場合は，<WHERE 句 > などでプロパティを修飾する < クラス名 > にも ID 文字列を指定する必要があります。

< 識別子 > および <ID 文字列 > の詳細は，「(7)< 識別子 >」および「(9)<ID 文字列 >」を参照してください。

プロパティ名の詳細

< プロパティ名 > は，文書空間内でプロパティを識別するための名称（dmaProp_DisplayName の値），または GUID 値の ID 文字列表現です。使用できる文字列の長さは，データベースのカラム名に使用できる文字列の長さに従います。

一つの edmSQL 文の中で，同じプロパティを識別するための < プロパティ名 > の表現に，名称と ID 文字列を混用することはできません。なお，そのプロパティがどのクラスのプロパティであるかが明確でない場合は，< プロパティ名 > を < クラス名 > または < 相関名 > で修飾する必要があります。

一つの edmSQL 文では，< プロパティ名 > は，名称または ID 文字列で統一して表記してください。例えば，< 選択項目 > の < プロパティ名 > に ID 文字列を指定した場合は，ORDER BY 句で指定する < プロパティ名 > にも，ID 文字列を指定する必要があります。

< 識別子 >，<ID 文字列 > および < 特殊なプロパティ > の詳細は，「(7)< 識別子 >」，「(9)<ID 文字列 >」および「(10)< 特殊なプロパティ >」を参照してください。

注意事項

DocumentBroker では，プロパティの定義は文書空間ごとに定義されています。したがって，異なるクラスに登録されたプロパティであっても，同じ文書空間内のクラスであれば，プロパティの性質は同じことが保証されています。ほかのオブジェクト指向言語で，クラスのスコープ単位にプロパティが定義されているものとは異なりますので，ご注意ください。

関数名の詳細

< 関数名 > は，edmSQL で予約されないキーワードとして登録されている関数を識別するための名称

です。
関数についての詳細は，「4.5.9　関数指定の構文規則」を参照してください。

相関名の詳細

< 相関名 > は，同じクラスを結合する場合に，結合するクラスを区別するための名称です。一つの <FROM 句 > の中に，同じクラスを複数回指定する場合には，必ず < 相関名 > を指定して，それぞれのクラスを一意に識別できるようにしてください。
そのほか，< 相関名 > は，< クラス名 > の別名としても使用できます。
< 相関名 > として使用できる文字列の長さなどの制限については，データベースの相関名に関する制限に従います。

(9) <ID 文字列 >

<ID 文字列 > の定義について説明します。<ID 文字列 > は，< クラス名 > や < プロパティ名 > を，名称（dmaProp_DisplayName に定義した値）以外で表記するための文字列です。

DocumentBroker の文書空間では，クラスやプロパティは GUID によって定義され，識別されます。クラスライブラリのメソッドでも，クラスやプロパティの識別には，この GUID を指定します。これと同様に，edmSQL でも，GUID でクラスやプロパティを指定できます。

なお，クラスライブラリでは，GUID 値は，DmaId 型という構造体で管理されますが，edmSQL では，データ型ではなく，GUID を <ID 文字列 > という記述形式で表現します。< クラス名 > や < プロパティ名 > と同じ扱いになるため，<ID 文字列 > に ? パラメタによって値を指定することはできません。

(a) <ID 文字列 > として指定できる字句

<ID 文字列 > は，< シングルクォート > で囲まれた GUID 文字列として指定します。

GUID 文字列とは，「X」を 0 ～ 9 および a ～ f（小文字）で表される 16 進数とした「XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX（8 けた -4 けた -4 けた -4 けた -12 けた）」の形式の文字列です。

なお，<ID 文字列 > に指定する「ID」と <ID 本体 > の間に，区切り文字は指定できません。

(b) 規約詳細

<ID 文字列 > の詳細について説明します。

<ID 文字列 > が区切られる単位についての詳細

ID 文字列の「ID」は，< 予約されるキーワード > です。したがって，< 非区切りトークン > として扱われます。また，<ID 本体 > は，< 区切りトークン > として扱われます。このため，<ID 文字列 > の前に < 区切り文字 > なしで < 非区切りトークン > は指定できませんが，<ID 文字列 > の後ろに < 区切り文字 > なしで < 非区切りトークン > は指定できます。
<ID 文字列 > の表記例と edmSQL 検索実行時の値の扱われ方を，次の表に示します。

表 4-34　<ID 文字列 > の表記例と edmSQL 検索実行時の値の扱われ方

<ID 文字列 > の表記	扱われ方
ID'FA7DA34B-CFF3-11d3-A03E-00A0C9967923'	正しい <ID 文字列 > として扱われます。
ID □ 'FA7DA34B-CFF3-11d3-A03E-00A0C9967923'	「ID」と <ID 本体 > の間に区切り文字が入っているため，字句解析エラーになります。
ID'FA7DA34B-CFF3-11d3-A03E-00A0C9967923'FROM	<ID 文字列 > と「FROM」として扱われます。

<ID 文字列 > の表記	扱われ方
SELECTID'FA7DA34B-CFF3-11d3-A03E-00A0C9967923'	「SELECTID」と文字リテラルとして扱われます。

（凡例）
□：区切り文字または < 空白 >

（10）< 特殊なプロパティ >

< 特殊なプロパティ > の定義について説明します。< 特殊なプロパティ > とは，全文検索で使用する全文検索インデクス用プロパティです。

（a）< 特殊なプロパティ > として指定できる字句

< 特殊なプロパティ > として指定できる字句を，次の表に示します。

表 4-35 < 特殊なプロパティ > として指定できる字句

種類	値	説明
特殊なプロパティ	edmProp_TextIndex	edmProp_TextIndex プロパティを表す < 正規識別子 >
	edmProp_StIndex	edmProp_StIndex プロパティを表す < 正規識別子 >
	edmProp_ConceptTextIndex	edmProp_ConceptTextIndex プロパティを表す < 正規識別子 >
	edmProp_ConceptStIndex	edmProp_ConceptStIndex プロパティを表す < 正規識別子 >
	edmProp_Content	edmProp_Content プロパティを表す < 正規識別子 >

これらのプロパティの特長については，「4.4.2 全文検索機能付き文書クラスの作成」を参照してください。

（b）規約詳細

< 特殊なプロパティ > の詳細について説明します。

< 特殊なプロパティ > と < プロパティ名 > についての詳細

- < 特殊なプロパティ > は，edmSQL で，DMA の検索モデルを拡張するための導入されたプロパティです。このプロパティは，edmSQL によって予約されたプロパティ名であるため，< プロパティ名 > としては使用できません。なお，DocumentBroker としても，ユーザプロパティに「dmaProp_」，「edmProp_」および「dbrProp_」で始まるプロパティ名称は使用できません。
  なお，クラスライブラリで拡張したプロパティのうち，< 特殊なプロパティ > として定義されていないプロパティについては，ほかのプロパティと同様の処理ができます。

< 特殊なプロパティ > を指定できる関数についての詳細

< 特殊なプロパティ > を指定できる関数を示します。

edmProp_TextIndex，edmProp_StIndex，edmProp_Content を第 1 引数に指定できる関数

- contains 関数
- contains_with_score 関数
- score 関数
- extracts 関数

edmProp_ConceptTextIndex，edmProp_ConceptStIndex を第 1 引数に指定できる関数

- contains 関数
- contains_with_score 関数
- score 関数
- extracts 関数
- concept_with_score 関数
- score_concept 関数

< 特殊なプロパティ > を引数に指定できる関数の詳細については，「4.5.9　関数指定の構文規則」を参照してください。

(11) <? パラメタ >

<? パラメタ > は，検索条件の定数を構文解析時には固定しないで，検索実行時に定数を指定するために使用するパラメタです。edmSQL 文には定数を指定する代わりに「?」を指定して，問い合わせの実行直前に CdbrEqlStatement クラスの SetParam メソッドなどで値を「?」に設定します。<? パラメタ > を使用することで，構文解析済みの edmSQL 文を定数値だけを変えて繰り返し利用できるので，処理性能があがります。

また，概念検索を実行する場合には，検索条件は <? パラメタ > を使用しないと指定できません。

(a) <? パラメタ > として指定できる字句

<? パラメタ > として指定できるのは，? だけです。

(b) 規約詳細

<? パラメタ > の詳細について説明します。

<? パラメタ > に指定する値のデータ型に関する詳細

- <? パラメタ > で指定される値のデータ型は不定です。このため，edmSQL 文の構文解析時にはデータ型はチェックされません。ただし，AS< データ型指定 > で明示的にデータ型を指定した <? パラメタ > については，データ型のチェックの対象になります。
- < ルーチンの起動 > の引数として <? パラメタ > を使用する場合は，AS< データ型指定 > で明示的にデータ型を指定してください。

<? パラメタ > に値を設定するメソッドについての詳細

- CdbrEqlStatement::SetParam メソッド，CdbrEqlStatement::SetObjParam メソッドおよび CdbrEqlStatement::SetOIIDParam メソッドで <? パラメタ > に値を設定する場合，メソッドの引数には，edmSQL の定数表現ではなくクラスライブラリのデータ型に従った値を指定してください。
- CdbrEqlStatement::SetParam メソッドによって <? パラメタ > の値を設定した場合，メソッドの引数に指定した値がそのまま <? パラメタ > の値として設定されます。
- CdbrEqlStatement::SetOIIDParam メソッドによって <? パラメタ > の値を設定した場合，メソッドの引数に指定した値（OIID 文字列）を dmaProp_OIID プロパティの格納形式（16 バイト）に変換された値が <? パラメタ > の値として設定されます。
- CdbrEqlStatement::SetObjParam メソッドによって <? パラメタ > の値を設定した場合，メソッドの引数に指定した値（OIID 文字列）をオブジェクトリファレンスの形式に変換された値が <? パラメタ > の値として設定されます。

(12) <OIID>

<OIID> の定義について説明します。

検索対象になるオブジェクトは，OIID によって識別されます。OIID の値は，データベース上では，各オブジェクトの dmaProp_OIID プロパティに，String 型の 16 バイトの値として格納されています。

一方，クラスライブラリでは，OIID を「dma:」で始まる OIID 文字列（DMA URL）で表記します。これは，実際に dmaProp_OIID プロパティに格納されている値とは異なります。したがって，クラスライブラリで使用する OIID 文字列を直接 edmSQL 文に指定しても，OIID を対象にした検索はできません。また，検索結果として dmaProp_OIID プロパティの値を取得した場合は，クラスライブラリで使用する OIID 文字列に変換されたものを使用します。

(a) OIID 文字列を dmaProp_OIID プロパティの値に変換する指定（OIID 変換インタフェース）

検索条件に OIID 文字列を指定する場合に，指定した OIID 文字列を dmaProp_OIID プロパティの値に変換するためには，次のどちらかの方法を使用します。

● **oiid 関数で変換する。**
edmSQL 文中に OIID 文字列を直接指定する場合の変換方法です。
oiid 関数の詳細については，「4.5.9(2)(d)　<objref 関数 >」を参照してください。

● **CdbrEqlStatement::SetOIIDParam メソッドで変換する。**
edmSQL に <? パラメタ > として指定しておく場合の変換方法です。SetOIIDParam メソッドの引数として，OIID 文字列を指定します。
CdbrEqlStatement::SetObjParam メソッドについては，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。

OIID 変換インタフェースの使用例

次に，OIID 変換インタフェースの使用例について説明します。

OIID 変換インタフェースを使用しないで OIID を検索しようとしている例

dmaProp_OIID プロパティの値と OIID 文字列の形式は異なるため，検索結果が「真」になる（検索結果が取得できる）ことはありません。

```
SELECT  myProp_Foo
FROM    myClass
WHERE  dmaProp_OIID = 'dma:///xxx/xxx/xxx…x'
```

oiid 関数を使用して OIID を検索する例

検索結果が「真」になる（検索結果が取得できる）可能性があります。

```
SELECT  myProp_Foo
FROM    myClass
WHERE  dmaProp_OIID = oiid('dma:///xxx/xxx/xxx…x')
```

SetOIIDParam メソッドを使用して OIID を検索する例

検索結果が「真」になる（検索結果が取得できる）可能性があります。

```
SELECT  myProp_Foo
FROM    myClass
WHERE  dmaProp_OIID = ?
```

このとき，SetParam メソッドや SetObjParam メソッドを使用して <? パラメタ > の値を設定しても，正しい検索はできません。

（b）dmaProp_OIID プロパティの値を OIID 文字列に変換する指定（dmaProp_OIID 取得インタフェース）

検索結果として dmaProp_OIID プロパティの値として取得した場合，検索結果として取得する構造体には，edmSQL によって OIID 文字列に変換された値が格納されます。明示的に変換する必要はありません。

4.5.5 検索の実行単位の構文規則

edmSQL 検索は，一つの <edmSQL 文 > から構成される，<edmSQL プログラム > ごとに実行されます。

（1）形式

!! <edmSQLプログラム>の形式

```
<edmSQLプログラム> ::= <edmSQL文>
                     | <edmSQL文> <セミコロン>
```

!! <edmSQL文>の形式

```
<edmSQL文>::= <問い合わせ文>
```

```
!! <問い合わせ文>については，「4.5.6 問い合わせ式の構文規則」を
!!参照してください。
```

（2）<edmSQL プログラム >

形式

```
<edmSQLプログラム> ::= <edmSQL文>
                     | <edmSQL文> <セミコロン>
```

<edmSQL プログラム > についての規則

- <edmSQL プログラム > は，一つの <edmSQL 文 > から構成されます。
- <edmSQL プログラム > が <edmSQL 文 > で構成されていない場合は，構文エラーになります。また，< セミコロン > だけを指定した場合も，構文エラーになります。

（3）<edmSQL 文 >

形式

```
<edmSQL文>::= <問い合わせ文>
```

<edmSQL 文 > についての規則

- <edmSQL 文 > に指定できるのは，< 問い合わせ文 >（SELECT 文）だけです。
- <edmSQL 文 > として指定できる文字列の長さの最大値は，データベースに依存します。

4.5.6 問い合わせ式の構文規則

問い合わせは，< 問い合わせ文 >（SELECT 文）によって表現します。< 問い合わせ文 > には，SELECT 句，FROM 句，WHERE 句，ORDER BY 句，GROUP BY 句および HAVING 句を指定できます。最も基本的な < 問い合わせ文 > は，SELECT 句と FROM 句から構成されます。

SELECT 句には，検索結果として取得する項目を指定します。

FROM 句，WHERE 句，GROUP BY 句および HAVING 句は，検索対象式として指定します。FROM 句には，検索対象になるクラスを指定します。WHERE 句には，検索条件を指定します。検索条件は，述語で表現します。述語については，「4.5.8 述語の構文規則」を参照してください。GROUP BY 句には，複数のオブジェクトを一つのグループとするかどうかを指定します。HAVING 句には，WHERE 句，FROM 句，GROUP BY 句の結果，得られる各グループを選択する条件を指定します。なお，WHERE 句および HAVING 句には，副問い合わせも指定できます。

ORDER BY 句には，検索結果として取得した集合をソートするかどうかを指定します。ORDER BY 句については，「4.5.10　データ操作の構文規則」を参照してください。

(1) 形式

!! **<問い合わせ文>の形式**

```
<問い合わせ文> ::= <問い合わせ式> 〔 <ORDER BY句>〕
<問い合わせ式> ::= <問い合わせ指定>
                 |<左括弧> <問い合わせ式> <右括弧>
```

!! **<問い合わせ指定>の形式**

```
<問い合わせ指定> ::= SELECT 〔 <集合指定子> 〕
                     <選択項目の並び> <検索対象式>
<選択項目の並び> ::= <アスタリスク>
                   |<選択項目> 〔 { <コンマ> <選択項目> }... 〕
<選択項目> ::= <一次子>
<集合指定子> ::= DISTINCT | ALL
```

!! **<検索対象式>の形式**

```
<検索対象式> ::= <FROM句>
                 〔 <WHERE句> 〕
                 〔 <GROUP BY句> 〕
                 〔 <HAVING句> 〕
```

!! **<FROM句>の形式**

```
<FROM句> ::= FROM <検索対象>
<検索対象> ::=  <検索対象参照リスト>
               <結合された検索対象>
<検索対象参照リスト> ::=<検索対象参照>
                       〔 { <コンマ> <検索対象参照> }... 〕
<検索対象参照> ::= <クラス名> 〔<相関名>〕

<結合された検索対象> ::= <条件指定結合>
<条件指定結合> ::= <検索対象一次子> 〔 <結合種別> 〕
                 JOIN <検索対象参照> <結合指定>

<検索対象一次子> ::= <検索対象参照>
                     |<結合された検索対象>
                     |<左括弧><結合された検索対象><右括弧>

<結合指定> ::= <結合条件>
<結合条件> ::= ON <検索条件>

<結合種別> ::= INNER
              |<外部結合種別> 〔 OUTER 〕
<外部結合種別> ::= LEFT
```

!! **<WHERE句>の形式**

```
<WHERE句> ::= WHERE <検索条件>
```

!! **<GROUP BY句>の形式**

```
<GROUP BY句> ::= GROUP BY <グループ化項目の並び>
<グループ化項目の並び> ::= <グループ化項目>
                         [ { <コンマ> <グループ化項目> }... ]
<グループ化項目> ::= <一次子>
```

!! **<HAVING句>の形式**

```
<HAVING句> ::= HAVING <検索条件>
```

!! **<副問い合わせ>の形式**

```
<副問い合わせ> ::=<左括弧> <問い合わせ式> <右括弧>

!!検索条件については，「4.5.8　述語の構文規則」を
!!参照してください。
```

(2) < 問い合わせ文 >

形式

```
<問い合わせ文> ::= <問い合わせ式> 〔 <ORDER BY句>〕
<問い合わせ式> ::= <問い合わせ指定>
                 | <左括弧> <問い合わせ式> <右括弧>
```

< 問い合わせ式 > についての規則

< 問い合わせ式 > は，少なくとも SELECT 句と FROM 句から構成される問い合わせの表現です。< 問い合わせ式 > は，検索結果集合として評価されます。

<ORDER BY 句 > についての規則

<ORDER BY 句 > には，< 問い合わせ式 > の評価である検索結果集合に対して，検索結果要素を並べ替えるためのソート方法を指定します。

(3) < 問い合わせ指定 >

形式

```
<問い合わせ指定> ::= SELECT 〔 <集合指定子> 〕
                     <選択項目の並び> <検索対象式>
<選択項目の並び> ::= <アスタリスク>
                   | <選択項目> 〔 { <コンマ> <選択項目> }... 〕
<選択項目> ::= <一次子>
```

< 選択項目 > および < 選択項目の並び > についての規則

- < 選択項目の並び > には，検索を実行した場合の結果として出力する項目を指定します。一つまたは複数の < 選択項目 >（< 選択項目の並び >）と < 検索対象式 > を指定することで，検索結果集合が求められます。
- < 選択項目 > に指定できるプロパティは，< 検索対象式 > に指定したクラスに定義されているプロパティだけです。
- < 選択項目の並び > に < アスタリスク > を指定できるのは，< 検索対象式 > の中の <Exists 述語 > で指定された < 副問い合わせ > の中だけです。
- < 検索対象式 > 中の < 比較述語 > や <In 述語 > で副問い合わせを指定する場合は，それぞれの述語によって求める検索結果と整合性のあるデータ型の項目を < 選択項目 > として指定してください。
- < 選択項目 > の < 一次子 > に指定できる要素は，< プロパティ指定 >，< 集合関数 > または < ルーチンの起動 > だけです。
  < ルーチンの起動 > に指定できる関数については，「4.5.9　関数指定の構文規則」を参照してください。
- < 選択項目 > に，VariableArray 型プロパティ以外の Object 型プロパティを直接指定することはできません。オブジェクトリファレンスである Object 型プロパティを指定する場合は，oiidstr 関数によって，OIID 文字列に変換して指定してください。

< 集合指定子 > についての規則

- < 集合指定子 > は，検索結果集合内の結果として同じ要素が返却された場合に，その重複した要素を排除（**重複排除**）する場合に指定します。
- DISTINCT を指定すると，検索結果集合に対して重複排除が実行され，重複を排除した検索結果集合が返却されます。
- ALL を指定すると，重複している要素も含めて，すべての検索結果集合が返却されます。
- 省略した場合は，ALL が仮定されます。
- < 選択項目 > に VariableArray 型プロパティを指定した場合は，< 集合指定子 > に DISTINCT は指定できません。DISTINCT を指定できる選択項目のデータ型については，データベースの制限に従います。

- アクセス制御機能を使用している場合，< 集合指定子 > に DISTINCT は指定できません。

（4）< 検索対象式 >

形式

```
<検索対象式> ::= <FROM句>
                 〔 <WHERE句> 〕
                 〔 <GROUP BY句> 〕
                 〔 <HAVING句> 〕
```

なお，<WHERE 句 >，<GROUP BY 句 > および <HAVING 句 > は，省略できます。

（5）<FROM 句 >

<FROM 句 > には，検索対象になる一つまたは複数のクラスを指定します。複数のクラスを検索対象に指定する場合には，結合条件も指定できます。

edmSQL では，結合方法として，次の方法が使用できます。

- 複数のクラスを並べて指定した暗黙的な結合
- 結合種別に内部結合（INNER JOIN）を指定した結合
- 結合種別に左外部結合（LEFT OUTER JOIN）を指定した結合

形式

```
<FROM句> ::= FROM <検索対象>
<検索対象> ::=   <検索対象参照リスト>
               ｜<結合された検索対象>
<検索対象参照リスト> ::= <検索対象参照>
                         〔{ <コンマ> <検索対象参照> }...〕
<検索対象参照> ::= <クラス名> 〔<相関名>〕

<結合された検索対象> ::= <条件指定結合>
<条件指定結合> ::= <検索対象一次子> 〔 <結合種別> 〕
                   JOIN <検索対象参照> <結合指定>

<検索対象一次子> ::= <検索対象参照>
                       ｜<結合された検索対象>
                       ｜<左括弧><結合された検索対象><右括弧>

<結合指定> ::= <結合条件>
<結合条件> ::= ON <検索条件>

<結合種別> ::= INNER
              ｜<外部結合種別> 〔 OUTER 〕
<外部結合種別> ::= LEFT
```

< 検索対象 > についての規則

- < 検索対象 > には < 検索対象参照リスト > を指定します。< 検索対象参照リスト > として < コンマ > で区切った < 検索対象参照 > を指定すると，それぞれの < 検索対象参照 > は暗黙的な結合で結合されて，検索対象として扱われます。
  暗黙的な結合によって検索対象を結合した場合は，結合条件を <WHERE 句 > に指定することで，内部結合を指定した場合と同じ結合が表現できます。<WHERE 句 > に結合条件を指定しない場合は，指定したクラスの直積空間が検索の対象になります。

< 結合された検索対象 > についての規則

- < 結合された検索対象 > は，複数の検索対象（クラス）を明示的に結合種別と結合条件を指定して結合する場合に指定します。

< 条件指定結合 > についての規則

- < 条件指定結合 > には，内部結合（INNER JOIN）または左外部結合（LEFT OUTER JOIN）が指定できます。
- < 条件指定結合 > では，< 結合種別 > の左側に指定する < 検索対象一次子 > に，さらに < 結合された検索対象 > を指定できます。つまり，結合のネストができます。
- < 条件指定結合 > を指定する場合は，必ず < 結合指定 >（ON 条件）を指定してください。

< 結合条件 > についての規則

- < 結合条件 > に指定するプロパティには，その < 結合条件 > を含む < 検索対象一次子 > および < 検索対象 > で指定しているクラスのプロパティ，またはその < 問い合わせ指定 > の外側の検索対象のクラスのプロパティを指定してください。

< 結合種別 > についての規則

- 内部結合を指定する場合は，< 結合種別 > に「INNER」を指定します。内部結合を指定すると，指定したクラスの直積空間のうち，< 結合条件 > を満たす結果の集合が検索対象になります。< 結合条件 > に結合キーとして指定したプロパティが NULL 値の場合，< 結合条件 > を満たす集合が存在しないので，指定したクラスの直積空間は検索対象にはなりません。このため，結合を指定したどちらのクラスのオブジェクトも検索対象になりません。
- 左外部結合を指定する場合は，< 結合種別 > に「LEFT」または「LEFT OUTER」を指定します。外部結合を指定した場合，< 結合条件 > に結合キーとして指定したプロパティが NULL 値であっても，< 結合種別 > の左側に指定した < 検索対象一次子 > のクラスのオブジェクトは，すべて検索対象になります。この場合，< 結合種別 > の右側に指定した検索対象のクラスのオブジェクトは存在しないことになるため，該当する項目には NULL 値が設定されます。
- < 結合種別 > を省略すると，「INNER」が仮定されます。

<FROM 句 > 全体についての規則

- 次の項目については，データベースの制限に従います。
  結合できるクラスの数や，結合で指定できるネストの深さ。
  < 結合条件 > に指定できるプロパティ。
  実行できる結合および結合の組み合わせ。
- 一つの FROM 句内で，相関名は重複できません。また，検索対象クラスのクラス名と相関名も，重複できません。
- FROM 句で指定する < クラス名 > および < 相関名 > の有効範囲は，次のとおりです。副問い合わせでこれらの < クラス名 > および < 相関名 > を指定するときには，ご注意ください。

< クラス名 > と < 相関名 > の有効範囲

- FROM 句で指定した < 相関名 > および < 相関名 > なしで指定した < クラス名 > の有効範囲は，その <FROM 句 > を含む < 問い合わせ指定 > 内全体です。したがって，その < 問い合わせ指定 > 内にある < 副問い合わせ > でも有効になります。
- < 相関名 > を指定した < クラス名 > の有効範囲は，その FROM 句を含む < 問い合わせ指定 > だけです。したがって，< 問い合わせ指定 > 内にある < 副問い合わせ > では，その名前は有効になりません。

（6）<WHERE 句 >

<WHERE 句 > には，検索条件を指定します。

検索対象のオブジェクトに対して，<WHERE 句 > で指定した < 検索条件 > が真である場合に，そのオブジェクトが検索結果として取得できます。取得した検索結果は，< 選択項目 > に指定した項目の値として取得できます。

形式

```
<WHERE句> ::= WHERE <検索条件>
```

< 検索条件 > については，「4.5.8　述語の構文規則」を参照してください。

(7) < 副問い合わせ >

形式

```
<副問い合わせ> ::=<左括弧> <問い合わせ式> <右括弧>
```

< 副問い合わせ > についての規則

- < 副問い合わせ > は，<In 述語 >，< 比較述語 > または <Exists 述語 > の対象として指定します。それぞれの述語については，「4.5.8　述語の構文規則」を参照してください。
- < 副問い合わせ > の < 選択項目 > に VariableArray 型プロパティは指定できません。
- アクセス制御機能を使用している場合も，< 副問い合わせ > ではアクセス制御は実行されません。< 副問い合わせ > の検索結果として取得できる検索結果には，アクセス権を持たないものが含まれる可能性があります。
- < 副問い合わせ > の外側に指定したクラスのプロパティを < 副問い合わせ > で指定する場合は，< 副問い合わせ > で指定するプロパティに < プロパティ修飾子 > を指定してください。

(8) GROUP BY 句

<GROUP BY 句 > には，複数のオブジェクトを一つのグループとする条件を指定します。

形式

```
<GROUP BY句> ::= GROUP BY <グループ化項目の並び>
<グループ化項目の並び> ::= <グループ化項目>
                    [ { <コンマ> <グループ化項目> }... ]
<グループ化項目> ::= <一次子>
```

<GROUP BY 句 > 全体についての規則

- GROUP BY 句で指定したプロパティの検索結果がすべて同じオブジェクトを一つのグループとして，検索結果をグループごとに取得できます。
- GROUP BY 句にはプロパティだけが指定できます。
- < グループ化項目の並び > に指定できるプロパティは，GROUP BY 句を指定した問い合わせの FROM 句で指定したクラスのプロパティです。
- < グループ化項目の並び > に指定できる < 一次子 > の最大数は，前提となる DBMS の仕様に制限されます。
- SELECT 句に指定できる選択項目は GROUP BY 句で指定したプロパティおよび集合関数です。
- オブジェクト型，バイナリ型のプロパティ，特殊なプロパティおよび関数は GROUP BY 句に指定できません。
- 同一のプロパティを重複して GROUP BY 句に指定することはできません。

(9) HAVING 句

<HAVING 句 > には，GROUP BY 句，WHERE 句，FROM 句の結果，得られる各グループを選択する条件を指定します。

形式

```
<HAVING句> ::= HAVING <検索条件>
```

<HAVING 句 > 全体についての規則

- GROUP BY 句，WHERE 句，または FROM 句の結果，得られる各グループを選択する条件を指定

します。

- NULL 述語および LIKE 述語は HAVING 句に直接指定できません。
- HAVING 句に指定できるプロパティは，GROUP BY 句で指定したプロパティ，集合関数の引数として指定したプロパティ，または外側の問い合わせの FROM 句に指定したクラスのプロパティです。

4.5.7 スカラー式表現の構文規則

値は，スカラー式によって表現できます。スカラー式で表現できる値は，プロパティの指定，ルーチン起動，数値関数および集合関数によって得られる値です。

(1) 形式

!! <プロパティ指定>の形式

```
<プロパティ指定> ::= 〔<プロパティ修飾子> <ピリオド>〕 <プロパティ名>
<プロパティ修飾子> ::= <相関名>       !! <相関名>による修飾
                     | <クラス指定> !! <クラス指定>による修飾
```

!! <要素参照>の形式

```
<要素参照> ::= <一次子>
                 <左角括弧> ANY <右角括弧>
```

!! <フィールド参照>の形式

```
<フィールド参照> ::= <要素参照> <ピリオド> <フィールド名>
<フィールド名> ::= <プロパティ名>
```

!! <ルーチンの起動>の形式

```
<ルーチンの起動> ::= <ルーチン名> <引数リスト>
<引数リスト> ::= <左括弧> 〔 <引数>
                   〔 { <コンマ> <引数> }... 〕 〕 <右括弧>
<引数> ::=  <値式>
          | <値式> AS <データ型指定>
<ルーチン名> ::= <関数指定>

<データ型指定> ::= INT
                 | INTEGER
                 | BOOL
                 | BOOLEAN
                 | STRING <左括弧> <符号なし数値> <右括弧>
                 | BINARY <左括弧> <バイナリ長> <右括弧>

<バイナリ長> ::=  <符号なし数値>
                | <バイナリ長トークン>
```

!! <数値関数>の形式

```
<数値関数> ::= <絶対値関数>
<絶対値関数> ::= ABS <左括弧> <値式> <右括弧>
```

!! <集合関数>の形式

```
<集合関数> ::= COUNT <左括弧> <アスタリスク> <右括弧>
             | <一般集合関数>
<一般集合関数> ::= <集合関数種別>
                     <左括弧> 〔 <集合指定子> 〕 <値式> <右括弧>
<集合関数種別> ::= COUNT
<集合指定子> ::= DISTINCT | ALL
```

!! <値式>の形式

```
<値式> ::= <一次子>
         | <符号> <一次子>
         | <値式> <+符号> <値式>
         | <値式> <-符号> <値式>
         | <値式> <アスタリスク> <値式>
         | <値式> <斜線> <値式>
         | <値式> <文字列連結演算子> <値式>
```

```
<一次子> ::= <左括弧> <値式> <右括弧>
           | <プロパティ指定>
           | <符号なし値指定>
           | <集合関数>
           | <数値関数>
           | <ルーチンの起動>
           | <フィールド参照>
<符号なし値指定> ::= <符号なし数値リテラル>
                   | <文字列リテラル>
                   | <疑問符>
                   | <論理リテラル>
<値指定> ::= 〔<符号>〕<符号なし数値リテラル>
           | <文字列リテラル>
           | <疑問符>
           | <論理リテラル>
```

(2) < プロパティ指定 >

形式

```
<プロパティ指定> ::= 〔<プロパティ修飾子> <ピリオド>〕 <プロパティ名>
<プロパティ修飾子> ::= <相関名>       !! <相関名>による修飾
                     | <クラス指定> !! <クラス指定>による修飾
```

< プロパティ指定 > についての規則

- 同じ名称のプロパティを持つクラスを結合する場合など，< プロパティ名 > だけではプロパティを一意に識別できない場合は，< プロパティ修飾子 > を指定して < プロパティ指定 > が一意になるようにしてください。

< プロパティ修飾子 > についての規則

- < 相関名 > および < クラス指定 > として指定できるのは，<FROM 句 > に指定したものだけです。

< プロパティ指定 > の指定例を，次の表に示します。

表 4-36 < プロパティ指定 > の指定例

指定例	説明
myProp_Foo	< 識別子 > によって < プロパティ名 > だけを指定しています。
ID'FA7DA34B-CFF3-11d3-A03E-00A0C9967923'	<ID 文字列 > によって < プロパティ名 > だけを指定しています。
myClass_XX.myProp_Foo	< プロパティ名 > を < クラス名 > によって修飾して指定しています。
P0.myProp_Foo	< プロパティ名 > を < 相関名 > によって修飾して指定しています。
myClass_XX.ID'FA7DA34B-CFF3-11d3-A03E-00A0C9967923'	<ID 文字列 > によって表現した < プロパティ名 > を，< クラス名 > によって修飾して指定しています。

(3) < 要素参照 >

< 要素参照 > には，VariableArray 型プロパティの要素を参照するための表現を指定します。

形式

```
<要素参照> ::= <一次子>
                 <左角括弧> ANY <右角括弧>
```

< 要素参照 > についての規則

- < 要素参照 > は，<WHERE 句 > だけで指定できます。
- < 要素参照 > は，必ず < フィールド参照 > とともに指定してください。
- 要素を指定する値には，「ANY」以外の要素は指定できません。

- < 一次子 > には，< プロパティ指定 > だけが指定できます。

(4) < フィールド参照 >

< フィールド参照 > では，VariableArray 型プロパティの要素を参照する時の，フィールドの参照を表現します。

形式

```
<フィールド参照> ::= <要素参照> <ピリオド> <フィールド名>
<フィールド名> ::= <プロパティ名>
```

< フィールド参照 > についての規則

- < フィールド参照 > は，VariableArray 型プロパティの要素を参照する場合だけ指定します。
- < フィールド名 > には，VariableArray 型プロパティの < プロパティ名 > を指定します。

例えば，「VariableArray 型プロパティ Authors の要素 Age が 30 である」という指定は，次のように記述します。

```
Authors[ANY].Age = 30
```

(5) < ルーチンの起動 >

< ルーチンの起動 > には，edmSQL で提供する関数を起動するための表現を指定します。

edmSQL で使用するルーチンは，関数だけです。

edmSQL によって起動する関数を次に示します。なお，これらの関数名はすべて，< 予約されないキーワード > として登録されています。

全文検索を実行するための関数

- contains 関数
- contains_with_score 関数
- score 関数
- extracts 関数

概念検索を実行するための関数

- concept_with_score 関数
- score_concept 関数

変換関数

- oiidstr 関数
- objref 関数
- oiid 関数

それぞれの関数の詳細については，「4.5.9　関数指定の構文規則」を参照してください。

形式

```
<ルーチンの起動> ::= <ルーチン名> <引数リスト>
<ルーチン名> ::= <関数指定>
<引数リスト> ::= <左括弧> 〔 <引数>
                 〔 { <コンマ> <引数> }... 〕 〕 <右括弧>
<引数> ::= <値式>
         |<値式> AS <データ型指定>

<データ型指定> ::= INT
                 | INTEGER
                 | BOOL
```

```
                    | BOOLEAN
                    | STRING <左括弧> <符号なし数値> <右括弧>
                    | BINARY <左括弧> <バイナリ長> <右括弧>

<バイナリ長> ::= <符号なし数値>
                 | <バイナリ長トークン>
!!<バイナリ長トークン>については，「4.5.4(4)<トークン>」を
!!参照してください。
```

< ルーチンの起動 > についての規則

- 関数の < 引数リスト > に指定できる引数の数は，< ルーチン名 > に指定した，それぞれの関数の仕様に従います。
- < 引数 > に指定できる < 一次子 > は，< ルーチン名 > に指定した関数の仕様に従います。各引数は，関数に規定された順序で指定する必要があります。
- < 引数リスト > に < 一次子 > として <? パラメタ > を指定する場合は，AS< データ型指定 > によって，データ型を明示してください。
  また，<? パラメタ > 以外の < 一次子 > に対して AS< データ型指定 > を指定することはできません。

AS< データ型指定 > についての規則

- AS< データ型指定 > に指定できるデータ型について説明します。関数の仕様に従って，適切なデータ型を指定してください。また，String 型および Binary 型を指定する場合には，定義長も指定してください。
  AS< データ型指定 > に指定できる値と指定例を，次の表に示します。

表 4-37 AS< データ型指定 > に指定できる値と指定例

	値	指定例
Boolean 型の引数	BOOL BOOLEAN	? AS BOOL ? AS BOOLEAN
Integer32 型の引数	INT INTEGER	? AS INT ? AS INTEGER
String 型の引数	STRING(n ※ 1)	? AS STRING(3200) 定義長 3200 バイトの String 型データ
Binary 型の引数	BINARY(n ※ 2)	? AS BINARY(256) 定義長 256 バイトの Binary 型データ ? AS BINARY(256k) 定義長が 256 キロバイトの Binary 型データ ? AS BINARY(256m) 定義長が 256 メガバイトの Binary 型データ ? AS BINARY(2g) 定義長が 2 ギガバイトの Binary 型データ

注※ 1

String 型データの定義長を指定します。単位はバイトです。

注※ 2

Binary 型データの定義長を指定します。次の単位が指定できます。

（省略）：バイト

k：キロバイト

m：メガバイト

g：ギガバイト

数字と単位の間に < 区切り文字 > を入れることはできません。

(6) < 数値関数 >

< 数値関数 > としては，絶対値関数（ABS 関数）があります。< 数値関数 > は，< 結合条件 > には指定できません。また，<SELECT 句 > には指定できません。

< 数値関数 > は，次の形式で定義されています。

```
<数値関数> ::= <絶対値関数>
<絶対値関数> ::= ABS <左括弧> <値式> <右括弧>
```

(a) 絶対値関数

< 絶対値関数 > の詳細について示します。

関数名
ABS

形式
ABS < 左括弧 > < 値式 > < 右括弧 >

引数

< 値式 >
Integer32 型の値を指定します。

機能
< 値式 > の値の絶対値を算出します。

被演算子のデータ型
Integer32 型

評価値
整数値（Integer32 型）

(7) < 集合関数 >

< 集合関数 > には，検索結果集合に対する演算を実行するための表現を指定します。

形式

```
<集合関数> ::= COUNT <左括弧> <アスタリスク> <右括弧>
             | <一般集合関数>
<一般集合関数> ::= <集合関数種別><左括弧> 〔 <集合指定子> 〕
                   <値式> <右括弧>
<集合関数種別> ::= COUNT
<集合指定子> ::= DISTINCT | ALL
```

< 集合関数 > についての規則

< 集合関数 > には，次の 2 種類があります。

- COUNT(*)
  検索結果の個数を算出します。
- COUNT
  検索結果のプロパティのキー数を算出します。

< 集合指定子 > についての規則

- < 集合指定子 > では，演算対象である集合の要素に対して，同じ要素がある場合に，重複排除を実施するかどうかを指定します。DISTINCT か ALL を指定します。
- DISTINCT を指定した場合，演算の対象である要素に対して，重複排除が実施されてから，演算が

実行されます。

- ALL を指定した場合，重複排除は実施されません。
- < 集合指定子 > を省略した場合は，「ALL」が仮定されます。
  「COUNT(*)」を指定した場合には，「DISTINCT」の指定は無視されます。
- < 集合指定子 > に「DISTINCT」を指定した場合，データベースによっては，この集合関数を選択項目とする問い合わせ指定の < 集合指定子 > 指定が制限されることがあります。
- アクセス制御機能を使用している場合，主問い合わせの選択項目に集合関数は指定できません。

(a) COUNT(*) 関数

COUNT(*) 関数の詳細について説明します

関数名
　COUNT(*)

形式
　COUNT < 左括弧 > < アスタリスク > < 右括弧 >

引数
　なし (アスタリスク)

機能
　検索結果集合の要素の数を算出します。

評価値
　整数値（Integer32 型）

属性
　この関数は，<SELECT 句 > に指定できます。
　また，<ORDER BY 句 > のソートキーとして指定できます。

(b) COUNT 関数

COUNT 関数の詳細について説明します。

関数名
　COUNT

形式
　COUNT < 左括弧 > 〔< 集合指定子 >〕< 値式 > < 右括弧 >

引数

　< 値式 >
　　Integer32 型，String 型，Boolean 型の < プロパティ指定 > だけが指定できます。

機能
　検索結果集合から，< 値式 > に指定した要素の数を算出します。

評価値
　整数値（Integer32 型）

属性
　この関数は，<SELECT 句 > に指定できます。
　また，<ORDER BY 句 > のソートキーとして指定できます。

(8) < 値式 >

< 値式 > とは，評価として値を得ることができる式の定義です。

形式

```
<値式> ::= <一次子>
          |<符号> <一次子>
          |<値式> <+符号> <値式>
          |<値式> <-符号> <値式>
          |<値式> <アスタリスク> <値式>
          |<値式> <斜線> <値式>
          |<値式> <文字列連結演算子> <値式>
<一次子> ::= <左括弧> <値式> <右括弧>
          |<プロパティ指定>
          |<符号なし値指定>|<集合関数>|<数値関数>
          |<要素参照>
          |<ルーチンの起動>|<フィールド参照>
<符号なし値指定> ::= <符号なし数値リテラル>
                   |<文字列リテラル>
                   |<疑問符>
                   |<論理リテラル>
<値指定> ::= 〔<符号>〕<符号なし数値リテラル>
           |<文字列リテラル>|<疑問符>|<論理リテラル>
```

< 値式 > についての規則

- < 値式 > 内の演算子は，次の順序で評価されます。

演算子の評価順序

1. 括弧内
2. 単項演算子 < 符号 > の <+ 符号 > または <- 符号 >
3. < アスタリスク > または < 斜線 >
4. 二項演算子の <+ 符号 >，<- 符号 > または < 結合演算子 >

- 演算子の両側に指定する < 値式 > には，同じデータ型のものを指定してください。
- 演算の途中でオーバフローが発生したり，0 で除算を実行したりした場合，NULL 値が返却されるかエラーが発生するかについては，データベースに依存します。
- 指定できる演算子のネストの深さは，データベースの制限に従います。
- < 値指定 > と < 符号なし値指定 > の違いは，符号が付けられるかどうかです。< 値式 > で指定できるのは，< 符号なし値指定 > です。符号を付ける場合は，< 値式 > を < 符号 > と < 符号なし値指定 > で構成してください。
  < 値指定 > は，<In 述語 > などで使用します。

演算子の詳細

< 値式 > で指定する演算子の詳細を，次の表に示します。

表 4-38 < 値式 > で指定する演算子の詳細

演算子	意味	被演算子のデータ型	評価値
< 符号 >（単項演算子 +）	値に正符号を付けます。評価値に変化はありません。	Integer32 型	整数値（Integer32 型）
< 符号 >（単項演算子 -）	値の符号を反転します。	Integer32 型	整数値（Integer32 型）
<+ 符号 >（+）	被演算子を加算します。	Integer32 型	整数値（Integer32 型）

演算子	意味	被演算子のデータ型	評価値
<- 符号 > (-)	左辺の被演算子から右辺の被演算子を減算します。	Integer32 型	整数値（Integer32 型）
< アスタリスク > (*)	被演算子を乗算します。	Integer32 型	整数値（Integer32 型）
< 斜線 > (/)	左辺の被演算子を右辺の被演算子で除算します。	Integer32 型	整数値（Integer32 型）
< 文字列連結演算子 > (||)	文字列を連結します。	String 型	文字列値（String 型）

注※

edmSQL の <+ 符号 > および < アスタリスク > では多項演算は実行できません。このため，多項演算と同じ演算を実行したい場合は，2 項演算を組み合わせてください。

4.5.8 述語の構文規則

ここでは，述語の構文について説明します。

FROM 句の ON 条件や WHERE 句に指定する検索条件は，述語によって表現します。述語では，次のような演算ができます。

- 比較演算
- 論理演算
- Between 演算
- In 演算
- Like 演算
- Null 演算
- Exists 演算

これらの演算では，評価が「真」,「偽」または「不定」として得られます。

(1) 形式

!! <検索条件>の形式

```
<検索条件> ::= <左括弧> <検索条件> <右括弧>
             | <述語>
             | NOT <検索条件>
             | <検索条件> OR <検索条件>
             | <検索条件> AND <検索条件>
```

!! <述語>の形式

```
<述語> ::= <比較述語>
         | <論理述語>
         | <Between述語>
         | <In述語>
         | <Like述語>
         | <Null述語>
         | <Exists述語>
```

!! <比較述語>の形式

```
<比較述語> ::= <値式> <比較演算子> <比較述語値式>
<比較述語値式> ::= <値式> | <副問い合わせ>
```

```
<比較演算子> ::= <等号演算子>
               |<不等号演算子>
               |<小なり演算子>
               |<大なり演算子>
               |<小なり等号演算子>
               |<大なり等号演算子>
```

!! <論理述語>の形式

```
<論理述語> ::= <値式> IS TRUE
```

!! <Between述語>の形式

```
<Between述語> ::= <値式>
                  〔 NOT 〕 BETWEEN <値式> AND <値式>
```

!! <In述語>の形式

```
<In述語> ::= <値式> 〔 NOT 〕 IN <In述語値>
<In述語値> ::= <副問い合わせ>
             |<左括弧> <In項目リスト> <右括弧>
<In項目リスト> ::= <値指定> { <コンマ> <値指定> }...
```

!! <Like述語>の形式

```
<Like述語> ::= <文字列Like述語>
<文字列Like述語> ::= <値式>
                     〔 NOT 〕 <Like種別> <パターン文字列>
                     〔 ESCAPE <エスケープ文字> 〕
<Like種別> ::=  LIKE
              |XLIKE
<パターン文字列> ::= <値指定>
<エスケープ文字> ::= <値指定>
```

!! <Null述語>の形式

```
<Null述語> ::= <値式> IS 〔 NOT 〕 NULL
```

!! <Exists述語>の形式

```
<Exists述語> ::= EXISTS <副問い合わせ>
```

(2) < 検索条件 >

< 検索条件 > は，<FROM 句 > の ON 条件式 < 結合指定 > や，<WHERE 句 > に指定します。< 検索条件 > では，複数の < 検索条件 > の論理演算を指定できます。

形式

```
<検索条件> ::= <左括弧> <検索条件> <右括弧>
             |<述語>
             |NOT <検索条件>
             |<検索条件> OR <検索条件>
             |<検索条件> AND <検索条件>
```

< 検索条件 > についての規則

< 検索条件 > の論理演算は，次の順序で評価されます。

論理演算の評価順序

1. 括弧内
2. NOT
3. AND
4. OR

- 論理演算で指定できるネスト数については，データベースの制限に従います。
- edmSQL では，DMA の検索モデルの edmQueryOperator_AndNot オペレータに該当する演算子はありません。指定する場合は，AND と NOT を組み合わせて指定してください。
- edmSQL で提供している論理演算子 AND および OR は，2 項演算子です。DMA の検索モデルのオペレータの仕様とは異なりますので，ご注意ください。複数の被演算子を指定したい場合は，演

算子を複数組み合わせて指定してください。
例えば，「a」，「b」，「c」および「d」の AND 演算は，「a AND b AND c AND d」または「((a AND b) AND c) AND d」のように表記してください。

論理演算子の詳細

否定演算

演算子：NOT
形式：NOT < 検索条件 >
否定演算の演算結果を次の表に示します。

表 4-39 否定演算の演算結果

NOT	演算結果
真	偽
偽	真
不定	不定

論理和演算

演算子：OR
形式：< 検索条件 > OR < 検索条件 >
論理和演算の演算結果を次の表に示します。

表 4-40 論理和演算の演算結果

OR	真	偽	不定
真	真	真	真
偽	真	偽	不定
不定	真	不定	不定

論理積演算

演算子：AND
形式：< 検索条件 > AND < 検索条件 >
論理積演算の演算結果を次の表に示します。

表 4-41 論理積演算の演算結果

AND	真	偽	不定
真	真	偽	不定
偽	偽	偽	偽
不定	不定	偽	不定

(3) < 述語 >

< 述語 > では，edmSQL で提供されている述語の一覧が定義されています。

形式

```
<述語> ::= <比較述語>
         | <論理述語>
         | <Between述語>
         | <In述語>
         | <Like述語>
```

```
        | <Null述語>
        | <Exists述語>
```

(4) < 比較述語 >

< 比較述語 > は，演算子の両辺の比較によって結果を返却します。

形式

```
<比較述語> ::= <値式> <比較演算子> <比較述語値式>
<比較述語値式> ::= <値式> | <副問い合わせ>
<比較演算子> ::= <等号演算子> | <不等号演算子> | <小なり演算子>
                | <大なり演算子> | <小なり等号演算子>
                | <大なり等号演算子>
```

< 比較述語 > の評価

- 演算子が示す比較演算子が，両辺の被演算子の関係を満たす場合，「真」になります。
- 演算子の示す比較条件が，両辺の被演算子の関係を満たさない場合，「偽」になります。
- 指定した被演算子が NULL 値の場合，「不定」になります。
- 指定した被演算子が < 副問い合わせ > であり，その検索結果集合が空の場合，「不定」になります。

< 比較述語 > の規則

- < 比較演算子 > の両辺に指定する被演算子には，同じデータ型の値を指定してください。値が NULL 値である場合も，同じデータ型にしてください。
- < 比較演算子 > によって比較できるのは，データ型が Boolean 型，Integer32 型，String 型および Object 型の値です。ただし，演算子によって使用できるデータ型は異なります。
- 比較できるデータ型は，データベースによって制限されます。
- < 副問い合わせ > が指定できるのは，< 比較演算子 > の右辺だけです。
- < 副問い合わせ > を < 比較演算子 > の被演算子に指定する場合，< 副問い合わせ > で指定できる < 選択項目 > は一つだけです。また，< 副問い合わせ > の検索結果として得られる検索件数も，1 件または 0（空集合）になる必要があります。
- <FROM 句 > に指定する < 比較述語 > には，< 副問い合わせ > は指定できません。
- VariableArray 型プロパティを指定する場合は，必ず < フィールド参照 >（要素指定の添え字は ANY）を指定して，比較できるデータ型のプロパティとして指定してください。また，VariableArray 型プロパティのフィールド参照同士の比較はできません。

< 比較述語 > で使用する演算子の詳細

演算子の詳細を，次の表に示します。

表 4-42 < 比較述語 > で使用できる演算子の詳細

演算子	意味	比較可能なデータ型
< 等号演算子 >（=）	値が等しい場合に「真」になります。	• Boolean 型 • Integer32 型 • String 型 • Object 型
< 不等号演算子 >（<>）	値が等しくない場合に「真」になります。	• Boolean 型 • Integer32 型 • String 型
< 小なり演算子 >（<）	左辺の値が右辺の値よりも小さい場合に「真」になります。	• Integer32 型 • String 型
< 大なり演算子 >（>）	左辺の値が右辺の値よりも大きい場合に「真」になります。	• Integer32 型 • String 型

演算子	意味	比較可能なデータ型
< 小なり等号演算子 >（<=）	左辺の値が右辺の値よりも小さいか，等しい場合に「真」になります。	• Integer32 型 • String 型
< 大なり等号演算子 >（>=）	左辺の値が右辺の値よりも大きいか，等しい場合に「真」になります。	• Integer32 型 • String 型

(5) < 論理述語 >

<論理述語>では，Boolean型の値を評価して結果を返却します。

形式

```
<論理述語> ::= <値式> IS TRUE
```

< 論理述語 > の評価

- < 値式 > の Boolean 型の値が，真（TRUE）の場合に「真」になります。
- < 値式 > の Boolean 型の値が，真（TRUE）以外の場合，「偽」になります。

< 論理述語 > の規則

- < 値式 > に指定できるのは，戻り値が Boolean 型の < ルーチンの起動 > だけです。なお，< 論理述語 > に指定する < ルーチンの起動 > で呼び出せる関数は，HiRDB Text Search Plug-in と連携するための，次の関数だけです。

  contains 関数

  contains_with_score 関数

  concept_with_score 関数

  例えば，WHERE 句に，次のように指定できます。

```
WHERE contains(edmProp_StIndex, '{"コンピュータ"}') IS TRUE
```

(6) <Between 述語 >

形式

```
<Between述語> ::= <値式>
                  〔 NOT 〕 BETWEEN <値式> AND <値式>
```

<Between 述語 > の評価

- 第 1 の < 値式 > を「値式 1」，第 2 の < 値式 > を「値式 2」，第 3 の < 値式 > を「値式 3」とした場合，「値式 1」の値が，「値式 2」と「値式 3」の範囲内の値の場合，「真」になります。範囲外の場合，「偽」になります。
- 第 1 の < 値式 > を「値式 1」，第 2 の < 値式 > を「値式 2」，第 3 の < 値式 > を「値式 3」とした場合，BETWEEN 述語は，次の表記と同じ意味を表します。

  〔NOT〕((値式 2 <= 値式 1) AND (値式 1 <= 値式 3))

<Between 述語 > の規則

「値式 2 ≦ 値式 3」である必要はありません。

(7) <In 述語 >

形式

```
<In述語> ::= <値式> 〔 NOT 〕 IN <In述語値>
<In述語値> ::= <副問い合わせ>
             |<左括弧> <In項目リスト> <右括弧>
<In項目リスト> ::= <値指定> { <コンマ> <値指定> }...
```

<In 述語 > の評価

- 第 1 の < 値式 > が，<In 述語値 > の中の任意の値と一致する場合，<IN 述語 > は「真」になります。
  NOT を指定した場合は，第 1 の < 値式 > が <In 述語値 > の中の任意の値と一致するとき，「偽」になります。
- 第 1 の < 値式 > が，<In 述語値 > の中のすべての値と一致しない場合，<IN 述語 > は「偽」になります。
  NOT を指定した場合は，第 1 の < 値式 > が，<In 述語値 > の中のすべての値と一致しないとき，「真」になります。

<In 述語 > の規則

- 第 1 の < 値式 > のデータ型と，<In 述語値 > のデータ型は一致させてください。指定できるデータ型は，Boolean 型，Integer32 型，String 型および Object 型です。
- データベースによって，指定できるデータ型や，<In 項目リスト > に指定できる < 値指定 > が制限される場合があります。
- <In 項目リスト > を指定する場合は，第 1 の < 値式 > に < 値指定 > は指定できません。
- <In 述語＞で < 副問い合わせ > の検索結果と比較する場合，< 副問い合わせ > で指定できる < 選択項目 > は一つだけです。
- < 副問い合わせ > の検索結果集合が空集合の場合は，<IN 述語 > は「偽」になります。
  NOT を指定した場合に，< 副問い合わせ > の検索結果集合が空集合のときは，「真」になります。

（8）<Like 述語 >

形式

```
<Like述語> ::= <文字列Like述語>
<文字列Like述語> ::= <値式>
                     〔 NOT 〕 <Like種別> <パターン文字列>
                     〔 ESCAPE <エスケープ文字> 〕
<Like種別> ::= LIKE
             | XLIKE
<パターン文字列> ::= <値指定>
<エスケープ文字> ::= <値指定>
```

パターンの詳細

< パターン文字列 >

- _：1 文字の任意の文字を示します。
- %：0 文字以上の任意の文字数の文字列を示します。

< エスケープ文字 >

< エスケープ文字 > は，パターン文字列中に「_」や「%」を記述したい場合に指定する文字です。< エスケープ文字 > には，任意の 1 文字を指定します。

<Like 述語 > の評価

- < 値式 > の値が，< パターン文字列 > で表すパターンと一致した場合，「真」になります。
  NOT を指定した場合は，< 値式 > の値が，< パターン文字列 > の表すパターンと一致したとき，「偽」になります。
- < 値式 > の値が，< パターン文字列 > の表すパターンと一致しない場合，「偽」になります。
  NOT を指定した場合，< 値式 > の値が，< パターン文字列 > の表すパターンと一致しないとき，「真」になります。

<Like 述語 > の規則

- < 値式 > に指定できるのは，String 型の値だけです。
- < パターン文字列 > に指定できるのは，String 型の値だけです。

- < エスケープ文字 > に指定できるのは，String 型の値だけです。
- <Like 述語 > は，<FROM 句 > には指定できません。
- データベースによっては，< 値式 >，< パターン文字列 >，< エスケープ文字 > に指定できる値に制限があります。また，処理性能を上げるための指定方法については，データベースの使用方法に従ってください。
- 指定した文字の認識のされ方は，String 型がデータベース上でどの文字列型に対応するかに依存します。

<Like 種別 > の規則

- <Like 種別 > が LIKE の場合，パターン文字列とのパターン一致で，大文字・小文字を区別して評価します。
- <Like 種別 > が XLIKE の場合，パターン文字列とのパターン一致で，大文字・小文字を区別しないで評価します。

(9) <Null 述語 >

形式

```
<Null述語> ::= <値式> IS 〔 NOT 〕 NULL
```

<Null 述語 > の評価

- < 値式 > が NULL 値の場合に「真」になります。
  NOT を指定した場合は，< 値式 > が NULL 値のとき，「偽」になります。
- < 値式 > が NULL 値でない場合に「偽」になります。
  NOT を指定した場合は，< 値式 > が NULL 値でないとき，「真」になります。

<Null 述語 > の規則

- < 値式 > に使用できるデータ型は，Boolean 型，Integer32 型，String 型および Object 型（VariableArray 型を除く）のプロパティです。ただし，Boolean 型のプロパティは，NULL 値をとることができません。

データベースによって，指定できるデータ型に制限があります。

(10) <Exists 述語 >

形式

```
<Exists述語> ::= EXISTS <副問い合わせ>
```

<Exists 述語 > の評価

- < 副問い合わせ > の検索結果が，空集合でなければ「真」になります。
- < 副問い合わせ > の検索結果が，空集合の場合は「偽」になります。

<Exists 述語 > の規則

- < 副問い合わせ > の検索結果が空集合とは，検索結果の件数が 0 件であることです。つまり，検索結果の件数が 1 以上の場合に，<Exists 述語 > は「真」になります。
- <Exists 述語 > の < 副問い合わせ > では，SELECT 句の < 選択項目 > には，< アスタリスク > が指定できます。これ以外の < 選択項目 > に，< アスタリスク > は指定できません。

4.5.9 関数指定の構文規則

edmSQL では，基本的な SQL の文法で表現できない DocumentBroker の拡張検索機能を，関数として提供します。

edmSQL が提供する関数には，HiRDB の拡張検索機能を使用するための関数と，edmSQL で独自に拡張した関数があります。

それぞれの関数について説明します。

● **HiRDB の拡張検索機能（**HiRDB Text Search Plug-in **および** HiRDB Text Search Plug-in Conceptual Extension **を使用した検索の機能）を使用するための関数（**DBMS **関数）**

DBMS 関数には，次の 2 種類があります。

- 全文検索関数（概念検索を含まない全文検索をするための関数）
- 概念検索関数

DBMS 関数は，データベース上で，データベースの制限に従って実行されます。

● **edmSQL で独自に拡張した関数（**edmSQL **関数）**

edmSQL 関数として提供されているのは，変換関数です。

edmSQL 関数は，データベースへのアクセスの前後で実行されます。

なお，この項で説明する < 関数 > には，SQL の基本機能として提供されている < 集合関数 > および < 数値関数 > は含みません。

（1） 文書検索関数（DBMS 関数）

< 文書検索関数 > には，次の機能があります。

- 全文検索機能（概念検索を含まない全文検索）
- 概念検索機能

< 文書検索関数 > では，次に示す関数が定義されています。なお，以降，関数の説明で使用する "< 全文検索関数 >" という表記は，概念検索の機能を持たない全文検索を実行するための関数を表します。

● 全文検索関数

- contains 関数
- cotains_with_score 関数
- score 関数
- extracts 関数

● 概念検索関数

- concept_with_score 関数
- score_concept 関数

なお，全文検索関数を実行する場合は，HiRDB Text Search Plug-in が必要です。

また，概念検索関数を実行する場合は，HiRDB Text Search Plug-in および HiRDB Text Search Plug-in Conceptual Extension が必要です。

（a） 形式

!! <文書検索関数>の形式

```
<文書検索関数> ::= <全文検索関数>
                 | <概念検索関数>
```

!! <全文検索関数>の形式

```
<全文検索関数> ::= <contains関数>
                 | <contains_with_score関数>
                 | <score関数>
                 | <extracts関数>
```

!! <contains関数>の形式
```
<contains関数> ::= contains <左括弧><プロパティ指定>
                        <コンマ><全文検索条件><右括弧>
```

!! <contains_with_score関数>の形式
```
<contains_with_score関数> ::= contains_with_score
                            <左括弧><プロパティ指定>
                            <コンマ><全文検索条件><右括弧>
```

!! <score関数>の形式
```
<score関数> ::= score <左括弧><プロパティ指定><右括弧>
```

!! <extracts関数>の形式

形式1
```
<extracts関数> ::= extracts <左括弧><プロパティ指定><コンマ>
                        <抽出対象構造文字列> <コンマ>
                        <全文検索条件> <コンマ>
                        <ハイライトタグ文字列> <右括弧>
```

形式2
```
<extracts関数> ::= extracts <左括弧><全文検索機能付き文字列型プロパティ><右括弧>
```

!! <概念検索関数>の形式
```
<概念検索関数> ::=  <concept_with_score関数>
                  |<score_concept関数>
```

!! <concept_with_score関数>の形式
```
<concept_with_score関数> ::= concept_with_score
                            <左括弧><プロパティ指定>
                            <コンマ><概念検索条件><右括弧>
```

!! <score_concept関数>の形式
```
<score_concept関数> ::= score_concept
                            <左括弧><プロパティ指定><右括弧>
```

!! <全文検索条件><概念検索条件><抽出構造文字列>および<ハイライトタグ文字列>の形式
```
<全文検索条件> ::= <文字列リテラル> | <?パラメタ>
<概念検索条件> :: = <?パラメタ>
<抽出対象構造文字列> ::= <文字列リテラル> | <?パラメタ>
<ハイライトタグ文字列> ::= <文字列リテラル> | <?パラメタ>

!! <全文検索条件>，<概念検索条件>，<抽出対象構造文字列>
!! および<ハイライトタグ文字列>の指定方法については，
!! マニュアル「HiRDB Text Search Plug-in」を参照してください。
```

(b) < 文書検索関数 >

< 文書検索関数 > は，次の形式で定義されています。

```
<文書検索関数> ::= <全文検索関数>|<概念検索関数>
```

< 文書検索関数 > の規則

- < 文書検索関数 > は，HiRDB Text Search Plug-in の機能によって提供されている検索機能を edmSQL で使用するための関数です。
  この関数で実行する検索の検索条件の指定方法については，マニュアル「HiRDB Text Search Plug-in」を参照してください。ただし，関数の第 1 引数には，列ではなく DocumentBroker のプロパティを指定します。
- 第 1 引数の < プロパティ指定 > に指定できるのは，その関数で指定できる < 特殊なプロパティ > だけです。
- < 文書検索関数 > は，< 結合条件 > には指定できません。

(c) < 全文検索関数 >

全文検索関数には，次の種類があります。なお，この関数で実行する全文検索に，概念検索は含まれません。

● contains 関数
全文検索を実行します。

● contains_with_score 関数
全文検索を実行すると同時に，スコア値を算出します。

● score 関数
contains_with_score 関数で算出したスコア値を取得します。

● extracts 関数
文書からテキストデータを抽出します。抽出するデータは，構造を持つ文書の特定の構造にハイライトタグを埋め込んだデータです。

(d) <contains 関数 >

contains 関数の詳細について説明します。

関数名
contains

形式
```
contains <左括弧><プロパティ指定><コンマ><全文検索条件><右括弧>
```

引数

< プロパティ指定 >（< 特殊なプロパティまたは全文検索機能付き文字列型プロパティ >）

●特殊なプロパティの場合
検索対象の文書クラスが特定できる形式でプロパティを指定します。
指定できるのは，次の < 特殊なプロパティ > です。
- edmProp_StIndex プロパティ
- edmProp_TextIndex プロパティ
- edmProp_Content プロパティ
- edmProp_ConceptTextIndex プロパティ
- edmProp_ConceptStIndex プロパティ

●全文検索機能付き文字列型プロパティの場合
全文検索機能付き文字列型プロパティを指定します。

< 全文検索条件 >（String 型の値：STRING(32000)）
全文検索条件を指定します。
検索タームを含む文書が検索されます。
< 全文検索条件 > には，検索タームのほか，構造指定検索，同義語異表記展開検索および近傍条件検索などを実行するための検索条件を文字列で指定します。指定方法については，マニュアル「HiRDB Text Search Plug-in」を参照してください。
< 全文検索条件 > に <? パラメタ > を指定する場合は，次のように AS< データ型指定 > をします。
```
? AS STRING(32000)
```

機能
< プロパティ指定 > で指定したプロパティに対応する文書の内容（コンテント）に対して，< 全文検索条件 > で指定した条件で，全文検索を実行します。

評価値

Boolean 型（評価）

属性

この関数は，<WHERE 句 > の < 検索条件 > に指定できます。

(e) <contains_with_score 関数 >

contains_with_score 関数の詳細について説明します。

関数名

```
contains_with_score
```

形式

```
contains_with_score <左括弧><プロパティ指定><コンマ><全文検索条件><右括弧>
```

引数

< プロパティ指定 >（< 特殊なプロパティ >）

検索対象の文書クラスが特定できる形式でプロパティを指定します。
指定できるのは，次の < 特殊なプロパティ > です。

- edmProp_TextIndex プロパティ
- edmProp_StIndex プロパティ
- edmProp_Content プロパティ
- edmProp_ConceptTextIndex プロパティ
- edmProp_ConceptStIndex プロパティ

< 全文検索条件 >（String 型の値：STRING(32000)）

全文検索条件を指定します。全文検索条件に指定した検索タームが含まれる文書が検索されます。定義長は 32,000 バイトです。
< 全文検索条件 > には，検索タームのほか，構造指定検索や同義語異表記展開検索，近傍条件検索などの検索を実行するための検索条件を文字列で指定します。指定方法については，マニュアル「HiRDB Text Search Plug-in」を参照してください。
< 全文検索条件 > に <? パラメタ > を指定する場合は，次のように AS< データ型指定 > をします。

```
? AS STRING(32000)
```

機能

< プロパティ指定 > で指定したプロパティに対応する文書に対して，< 全文検索条件 > で指定した条件で，全文検索を実行します。
このとき，検索結果として取得した文書から，全文検索条件が含まれる割合を，スコアとして算出します。
算出したスコアの取得には，<score 関数 > を使用します。

評価値

Boolean 型（評価）

属性

この関数は，<WHERE 句 > の < 検索条件 > に指定できます。

(f) <score 関数 >

<score 関数 > の詳細について説明します。

関数名

score

形式

score < 左括弧 >< プロパティ指定 >< 右括弧 >

引数

< プロパティ指定 >（< 特殊なプロパティ >）

スコアの値を取得する文書クラスが特定できる形式でプロパティを指定します。
指定できるのは，次の < 特殊なプロパティ > です。

- edmProp_TextIndex プロパティ
- edmProp_StIndex プロパティ
- edmProp_Content プロパティ
- edmProp_ConceptTextIndex プロパティ
- edmProp_ConceptStIndex プロパティ

機能

<conatins_with_score 関数 > で検索結果として取得した文書に対して，文書の検索条件に対するスコア値（全文検索条件が含まれる割合に応じた値）を返却します。

評価値

スコア値（Integer32 型）

属性

この関数は，<SELECT 句 > の < 選択項目 > に指定できます。

(g) <extracts 関数 >

<extracts 関数 > は，検索結果として取得する文書から，テキストデータを抽出する関数です。

<extracts 関数 > は，次の関数と組み合わせて使用することで，全文検索条件を満たした文書のテキストデータを抽出できます。

- contains 関数
- contains_with_score 関数

また，次の関数と組み合わせて使用することで，全文検索条件を満たした全文検索機能付き文字列型プロパティを抽出できます。

- contains 関数

<extracts 関数 > の詳細について説明します。

関数名

extracts

形式 1

extracts <左括弧><プロパティ指定><コンマ><抽出対象構造文字列><コンマ><全文検索条件><コンマ><ハイライトタグ文字列><右括弧>

形式 2

extracts＜左括弧＞＜全文検索機能付き文字列型プロパティ＞＜右括弧＞

引数

< プロパティ指定 >（< 特殊なプロパティ >）

抽出対象の文書クラスが特定できる形式でプロパティを指定します。

指定できるプロパティは，次の＜特殊なプロパティ＞です。

- edmProp_TextIndex プロパティ
- edmProp_StIndex プロパティ
- edmProp_Content プロパティ
- edmProp_ConceptStIndex プロパティ
- edmProp_ConceptTextIndex プロパティ

＜全文検索機能付き文字列型プロパティ＞

全文検索機能付き文字列型プロパティを指定します。

＜抽出対象構造文字列＞（String 型の値：STRING(1024)）

抽出対象にする構造を表す文字列を指定します。
構造を指定しない場合は，「"」（空文字列）を指定してください。
＜抽出対象構造文字列＞に <? パラメタ＞を指定する場合は，次のように AS＜データ型指定＞をします。

```
? AS STRING(1024)
```

＜全文検索条件＞（String 型の値：STRING(32000)）

ハイライト表示する検索タームを指定します。
条件を指定しない場合は，「"」（空文字列）を指定してください。この場合，ハイライトタグは挿入されません。
＜全文検索条件＞に <? パラメタ＞を指定する場合は，次のように AS＜データ型指定＞をします。

```
? AS STRING(32000)
```

＜ハイライトタグ文字列＞（String 型の値：STRING(255)）

ハイライト表示に使用するタグを表す文字列を指定します。
ハイライトタグを指定しない場合は，「"」（空文字列）を指定してください。この場合，ハイライトタグは挿入されません。
＜ハイライトタグ文字列＞に <? パラメタ＞を指定する場合は，次のように AS＜データ型指定＞をします。

```
? AS STRING(255)
```

機能

＜プロパティ指定＞で指定したプロパティに対応する文書の内容（コンテント）から，＜抽出指定構造文字列＞および＜全文検索条件＞で指定した文字列に＜ハイライトタグ文字列＞に指定したタグ（文字列）を埋め込んだ状態で抽出します。
HiRDB Text Search Plug-in の「SGML 出力」に対応します。
また，＜全文検索機能付き文字列型プロパティ＞で指定した全文検索機能付き文字列型プロパティの値を抽出します。

評価値

抽出したテキストまたはプロパティの値（Binary 型データ）
抽出するデータは，HiRDB Text Search Plug-in の BLOB 型で 2 ギガバイトまでのデータです。ただし，DocumentBroker のクラスライブラリの検索結果としては，String 型の値に変換されて，取得します。

属性

この関数は，<SELECT 句＞の＜選択項目＞に指定できます。
ただし，この関数を <SELECT 句＞に指定した場合，＜集合指定子＞として DISTINCT は指定できません。

(h) < 概念検索関数 >

概念検索関数には，次の種類があります。

● concept_with_score 関数
概念検索を実行すると同時に，スコア値を算出します。

● score_concept 関数
concept_with_score 関数で算出したスコア値を取得します。

(i) <concept_with_score 関数 >

<concept_with_score 関数 > の詳細について説明します。

関数名
concept_with_score

形式
```
concept_with_score <左括弧><プロパティ指定><コンマ><概念検索条件><右括弧>
```

引数

< プロパティ指定 >（< 特殊なプロパティ >）
検索対象の文書クラスが特定できる形式でプロパティを指定します。
指定できるプロパティは，次の < 特殊なプロパティ > です。
- edmProp_ConceptTextIndex
- edmProp_ConceptStIndex

< 概念検索条件 >（Binary 型のデータ：BINARY(5m)）
概念検索条件を指定します。概念検索条件には，種文章を指定します。種文章を含む < 概念検索条件 > を指定する場合には，必ず <? パラメタ > を使用して指定してください。また，ここで指定する <? パラメタ > には，構造指定検索や同義語異表記検索などの検索を実行するための検索条件も指定します。指定方法については，マニュアル「HiRDB Text Search Plug-in」を参照してください。
< 概念検索条件 > を <? パラメタ > で指定する場合は，次のように AS< データ型指定 > をします。
```
? AS BINARY(5m)
```

AS< データ型指定 > を含んだ，concept_with_score 関数を呼び出す論理述語は，次のように指定します。
```
concept_with_score(edmProp_ConceptStIndex,
                   ? AS BINARY(5m)) IS TRUE
```

機能
< プロパティ指定 > で指定したプロパティに対応する文書の内容（コンテント）に対し，< 概念検索条件 > で指定した条件で，概念検索を実行します。
また，検索実行時に，検索結果として取得した文書から検索用特徴タームが含まれる割合をスコアとして算出します。
このスコアを取得する場合は，<score_concept 関数 > を使用します。

評価値
Boolean 型（評価）

属性
この関数は，<WHERE 句 > の < 検索条件 > に指定できます。

(j) <score_concept 関数 >

<score_concept 関数 > の詳細について説明します。

関数名
score_concept

形式
score_concept < 左括弧 >< プロパティ指定 >< 右括弧 >

引数

< プロパティ指定 >（< 特殊なプロパティ >）
スコアの値を取得する文書クラスが特定できる形式でプロパティを指定します。
指定できるのは，次の < 特殊なプロパティ > です。
- edmProp_ConceptTextIndex プロパティ
- edmProp_ConceptStIndex プロパティ

機能
<concept_with_score 関数 > で検索結果として取得した文書に対して，文書の検索条件に対するスコア値（検索用特徴タームが含まれる割合に応じた値）を返却します。

評価値
スコア値（Integer32 型）

属性
この関数は，<SELECT 句 > の < 選択項目 > に指定できます。

(2) 変換関数（edmSQL 関数）

< 変換関数 > は，Object 型の値を，クラスライブラリで扱えるデータ型に変換するための関数です。クラスライブラリでは，VariableArray 型プロパティを表す Object 型の値以外は，すべて OIID 文字列として扱います。

< 変換関数 > には，< 選択可能な変換関数 > と < 検索可能な変換関数 > があります。

(a) 形式

!! **<変換関数>の形式**
```
<変換関数> ::= <選択可能な変換関数>
             | <検索可能な変換関数>

<選択可能な変換関数> ::= <oiidstr関数>

<検索可能な変換関数> ::= <objref関数>
                        | <oiid関数>
```

!! **<oiidstr関数>の形式**
```
<oiidstr関数> ::= oiidstr <左括弧> <プロパティ指定> <右括弧>
```

!! **<objref関数>の形式**
```
<objref関数> ::= objref<左括弧> <OIID文字列> <右括弧>
```

!! **<oiid関数>の形式**
```
<oiid関数> ::= oiid<左括弧> <OIID文字列> <右括弧>

<OIID文字列> ::= <文字列リテラル>
```

(b) < 変換関数 >

< 変換関数 > は，データベースで処理する関数ではなく，DocumentBroker サーバで edmSQL を発行し

た前後にデータの変換を実行する関数です。

< 変換関数 > には，< 選択可能な変換関数 > と < 検索可能な変換関数 > があります。< 選択可能な変換関数 > は，主問い合わせの < 選択項目 > だけに指定できます。< 検索可能な変換関数 > は，< 検索条件 > だけに指定できます。なお，< 変換関数 > は，< 結合条件 > には指定できません。

これら関数では，次の変換ができます。

- オブジェクトリファレンス（Object 型の値）から OIID 文字列（String 型の値）への変換
- OIID 文字列からオブジェクトリファレンス（Object 型の値）を OIID 文字列（String 型の値）への変換
- OIID 文字列から dmaProp_OIID プロパティの値への変換

(c) <oiidstr 関数 >

<oiidstr 関数 > の詳細について説明します。

関数名
　oiidstr

形式
　oiidstr < 左括弧 >< プロパティ指定 >< 右括弧 >

引数
　< プロパティ指定 >（VariableArray 型以外の Object 型の値）

機能
　Object 型の値であるオブジェクトリファレンスを，OIID 文字列表記に変換します。

評価値
　String 型の値。
　OIID 文字列表記を，String 型の値として返却します。

属性
　< 選択可能な変換関数 >
　この関数は，<SELECT 句 > の < 選択項目 > に指定できます。

(d) <objref 関数 >

<objref 関数 > の詳細について説明します。

関数名
　objref

形式
　objref < 左括弧 > <OIID 文字列 >< 右括弧 >

引数
　<OIID 文字列 >（String 型の値）

機能
　<OIID 文字列 >（String 型の値）を，オブジェクトリファレンス（Object 型の値）に変換します。

評価値
　Object 型の値。

オブジェクトリファレンスを，Object 型の値として返却します。

属性

< 検索可能な変換関数 >

この関数は，<WHERE 句 > の < 検索条件 > に指定できます。

(e) <oiid 関数 >

<oiid 関数 > の詳細について説明します。

関数名

oiid

形式

oiid < 左括弧 ><OIID 文字列 >< 右括弧 >

引数

<OIID 文字列 >（Sring 型の値）

評価値

String 型の値。

実際に dmaProp_OIID プロパティに格納されている形式（16 バイトの値）の String 型の値を返却します。

属性

< 検索可能な変換関数 >

この関数は，<WHERE 句 > の < 検索条件 > に指定できます。

4.5.10 データ操作の構文規則

ここでは，データ操作を表現する構文について説明します。edmSQL で実行できるデータ操作は，ORDER BY 句による検索結果の並び替えです。

(1) 形式

!! <ORDER BY句>の形式

```
<ORDER BY句> ::= ORDER BY <ソート指定リスト>
<ソート指定リスト> ::= <ソート指定> 〔 { <comma> <ソート指定> }... 〕
<ソート指定> ::= <ソートキー>  〔 <順序指定> 〕
<ソートキー> ::= <プロパティ指定>
               |<符号なし整数>
<順序指定> ::= ASC|DESC
```

(2) <ORDER BY 句 >

<ORDER BY 句 > では，検索結果を昇順または降順に並び替える場合の，そのソート方法を指定します。<ORDER BY 句 > を省略した場合，検索結果の取得順序は不定になります。これは，検索結果が集合として管理されているためです。<ORDER BY 句 > では，この検索結果集合から要素を取り出す時の，ソート方法を指定します。

形式

```
<ORDER BY句> ::= ORDER BY <ソート指定リスト>
<ソート指定リスト> ::= <ソート指定> 〔 { <コンマ> <ソート指定> }... 〕
<ソート指定> ::= <ソートキー>  〔 <順序指定> 〕
<ソートキー> ::= <プロパティ指定>
               |<符号なし整数>
<順序指定> ::= ASC|DESC
```

< ソート指定リスト > についての規則

- < ソート指定リスト > に指定できる < ソート指定 > の最大数は，データベース制限に従います。

< ソートキー > についての規則

- < ソートキー > に指定できるのは，< プロパティ指定 > またはソート項目を指定する番号である < 符号なし整数 > です。ただし，指定できる < ソートキー > のデータ型については，データベースの制限に従います。
- < ソートキー > を < プロパティ指定 > で指定する場合，指定できるのは，最も外側の SELECT 句（主問い合わせの SELECT 句）の < 選択項目 > に指定したプロパティだけです。< 副問い合わせ > の SELECT 句で指定した < 選択項目 > のプロパティは，指定できません。
- 最も外側の SELECT 句（主問い合わせの SELECT 句）の < 選択項目 > が < ルーチンの起動 > や < 集合関数 > の場合，< プロパティ指定 > で < ソートキー > は指定できません。この場合は，ソート項目指定番号を指定してください。
- ソート項目指定番号とは，検索結果として指定した < 選択項目 > のうち，どの選択項目を < ソートキー > とするかを，< 選択項目 > の出現番号で表現したものです。したがって，SELECT 句で先頭に指定した < 選択項目 > が「1」になります。
- < ソートキー > の < 選択項目 > として指定できるプロパティは，ソート可能（Orderable）なプロパティです。これは，プロパティ定義に次のように指定されたプロパティのことです。
  `dmaProp_IsOrderable = bool = 1`
- 複数の < ソートキー > を指定した場合は，< ソートキー > を指定した順番（指定の左側からの順番）でソートします。

< 順序指定 > についての規則

- < 順序指定 > では，< ソートキー > を並べる順序の方向を，昇順にするか，降順にするかを指定します。
- ASC は < ソートキー > を昇順に並べる場合に指定します。
- DESC は < ソートキー > を降順に並べる場合に指定します。
- < 順序指定 > を省略した場合は，ASC が仮定されます。

4.6 edmSQL の指定例

この節では，edmSQL の指定例について説明します。

4.6.1 属性検索

ここでは，属性検索（プロパティを指定した検索）での edmSQL の指定例を示します。

(1) 指定例で使用するクラスとプロパティ

この指定例で検索の対象になる文書は，次のようなクラスから作成された文書です。

文書 X

Document X クラスを基に作成された文書です。ユーザ定義プロパティとして，次の表に示すプロパティが設定されています。

表 4-43 Document X クラスのプロパティ

プロパティ識別子	内容
DocCode	文書を管理するためのコード
Title	タイトル
Author	著者
AuthorId	著者 ID
Publisher	出版元
Abstract	概要
CreateDate	作成日

所有書一覧

OwnersList クラスを基に作成された文書です。ユーザ定義プロパティとして，次の表に示すプロパティが設定されています。

表 4-44 OwnersList クラスのプロパティ

プロパティ識別子	内容
DocCode	文書を管理するためのコード
Owner	所有者

辞書 Y

Dictionary Y クラスを基に作成された文書です。ユーザ定義プロパティとして，次の表に示すプロパティが設定されています。

表 4-45 Dictionary Y クラスのプロパティ

プロパティ識別子	内容
Name	名前
Birthday	誕生日
EntryId	エントリー ID

論文

myPaper クラスを基に作成された文書です。次の表に示すプロパティが設定されています。

表 4-46 myPaper クラスのプロパティ

プロパティ識別子		内容
Title		タイトル
Authors ※	Name	名前
	Org	所属

注※
VariableArray 型プロパティです。

(2) 一つのクラスに対して実行する属性検索

ここでは，一つのクラスに対して実行する属性検索の指定例を示します。

(a) SELECT 句と FROM 句だけを指定する検索

< 検索対象 > として Document X クラスを，< 選択項目 > として Title プロパティ，Author プロパティおよび Abstract プロパティを取得する例を示します。

指定例

```
SELECT Title,Author,Abstract
FROM   "Document X"
```

(b) SELECT 句，FROM 句および WHERE 句を指定する検索

(a) に加えて，検索条件として WHERE 句に「Author プロパティが『日立太郎』で，CreateDate プロパティが 20000101 より大きい」という条件を指定する例を示します。

指定例

```
SELECT Title,Author,Abstract
FROM "Document X"
WHERE Author = '日立太郎' AND CreateDate > '20000101'
```

(c) 重複排除を指定する検索

Document X クラスから，Author プロパティを，重複を排除して取得する例を示します。

指定例

```
SELECT DISTINCT Author
FROM "Document X"
```

(d) 検索条件として OIID を指定した検索

Document X クラスから作成された文書から，OIID を指定して，Title プロパティ，Author プロパティおよび Abstract プロパティを取得する例を示します。なお，OIID は，oiid 関数を使用して指定します。

指定例

```
SELECT Title,Author,Abstract
FROM "Document X"
WHERE dmaProp_OIID = oiid('dma:///xxx/xxx/xxxxxxxxxx...xxx')
```

(3) 結合したクラスに対して実行する属性検索

ここでは，複数のクラスを結合した検索対象クラスに対して実行する属性検索の指定例を示します。

なお，edmSQL で実行できる結合の方法には，次の 2 種類があります。

● 内部結合

● 左外部結合

なお，実現できる結合の種類については，データベースに従います。

それぞれの場合の指定例を示します。

(a) 二つのクラスを内部結合した検索

Document X クラスと OwnersList クラスを内部結合して，Document X クラスの文書の Title プロパティ，Document X クラスの Author プロパティおよび OwnersList クラスの Owner プロパティを取得する例を示します。ここでは，次の二つの方法の指定例を示します。

● FROM 句に < クラス指定 > を並べて内部結合して，結合の条件を WHERE 句に指定する方法

● INNER JOIN を使用して，二つのクラスの結合を指定する方法

結合するときには，Document X クラスの DocCode プロパティと OwnersList クラスの DocCode プロパティが一致することを条件とします。

なお，Document A クラスの < 相関名 > として DX，OwnersList クラスの < 相関名 > として OL を使用します。

FROM 句に < クラス指定 > を並べて内部結合して，結合の条件を WHERE 句に指定する方法

次のように指定します。

指定例

```
SELECT DX.Title,DX.Author,OL.Owner
FROM "Document X" DX, OwnersList OL
WHERE DX.DocCode = OL.DocCode
```

INNER JOIN を使用して二つのクラスの結合を指定する検索

次のように指定します。

指定例

```
SELECT DX.Title,DX.Author,OL.Owner
FROM   "Document X" DX INNER JOIN OwnersList OL
           ON DX.DocCode = OL.DocCode
```

(b) 三つのクラスを内部結合した検索

Document X クラス，OwnersList クラスおよび Dictionary Y クラスを結合して，Document X クラスの Title プロパティと Document X クラスの Author プロパティ，Dictionary Y クラスの Birthday プロパティおよび OwnersList クラスの Owner プロパティを取得する例を示します。ここでは，次の二つの方法の指定例を示します。

● FROM 句に < クラス指定 > を並べて内部結合して，結合の条件を WHERE 句に指定する方法

● INNER JOIN を使用して，三つのクラスの結合を指定する方法

結合するときには，Document X クラスの DocCode プロパティと OwnersList クラスの DocCode プロパティが一致することと，Document X クラスの AuthorId プロパティと Dictionary Y クラスの EntryId プロパティが一致することを条件とします。

FROM 句に < クラス指定 > を並べて内部結合して，結合の条件を WHERE 句に指定する方法

次のように指定します。

指定例

```
SELECT DX.Title,DX.Author,DY.Birthday,OL.Owner
```

```
FROM  "Document X" DX,
              OwnersList OL,"Dictionary Y" DY
WHERE  DX.DocCode = OL.DocCode
 AND DX.AuthorId = DY.EntryId
```

INNER JOIN を使用して三つのクラスの結合を指定する検索

次のように指定します。

指定例

```
SELECT DX.Title,DX.Author,DY.Birthday,OL.Owner
FROM ("Document X" HD INNER JOIN OwnersList OL
 ON DX.DocCode = OL.DocCode)
 INNER JOIN "Dictionary Y" DY
  ON DX.AuthorId = DY.EntryId
```

(c) 二つのクラスを外部結合した検索

Document X クラスと OwnersList クラスを外部結合して，Document X クラスの文書の Title プロパティと Document X クラスの Author プロパティおよび OwnersList クラスの Title プロパティを取得する例を示します。

次のように指定します。

指定例

```
SELECT DX.Title,DX.Author,OL.Owner
FROM "Document X" DX LEFT OUTER JOIN OwnersList OL
        ON DX.DocCode = OL.DocCode
```

(d) 三つのクラスを外部結合した検索

Document X クラス，OwnersList クラスおよび Dictionary Y クラスを外部結合して，Document X クラスの Title プロパティ，Document X クラスの Author プロパティ，Dictionary Y クラスの Birthday プロパティおよび OwnersList クラスの Owner プロパティを取得する例を示します。

指定例

```
SELECT DX.Title,DX.Author,DY.Birthday,OL.Owner
FROM ("Document X" DX LEFT OUTER JOIN OwnersList OL
    ON DX.DocCode = OL.DocCode) LEFT OUTER JOIN
  "Dictionary Y" DY ON DX.AuthorId = DY.EntryId
```

(4) 副問い合わせを実行する属性検索

ここでは，副問い合わせを指定する場合の属性検索の指定例を示します。

(a) 比較述語を指定した副問い合わせ

OIID がわかっている Document X クラスのオブジェクトの CreateDate プロパティよりも，Birthday プロパティが新しい（値が大きい）Dictionary Y クラスのオブジェクトの，Name プロパティと Birthday プロパティを取得する例を示します。

指定例

```
SELECT Name, Birthday
FROM "Dictionary Y"
WHERE  Birthday > (SELECT CreateDate
        FROM "Document X"
        WHERE dmaProp_OIID =
               oiid( 'dma:///xxx/xxx/xxxxxxxxxx...xxx'))
```

(b) IN 述語を指定した副問い合わせ

CreateDate プロパティが 19990101 よりも値が大きい Document X クラスのオブジェクトの AuthorId プ

ロパティのどれかと，Dictionary Y クラスの EntryId プロパティの値が等しい，Dictionary Y クラスのオブジェクトの Name プロパティと Birthday プロパティを取得する例を示します。

指定例

```
SELECT Name, Birthday
FROM "Dictionary Y"
WHERE EntryId IN (SELECT AuthorId
                      FROM "Document X"
                      WHERE CreateDate > '19990101')
```

(c) EXISTS 述語を指定した副問い合わせ

Dictionary Y クラスのオブジェクトの Name プロパティと Document X クラスのオブジェクトの Author プロパティが一致する文書が一つでもあれば，その Dictionary Y クラスの文書の Name プロパティと Birthday プロパティを取得する例を示します。

指定例

```
SELECT Name, Birthday
FROM "Dictionary Y" DY
WHERE EXISTS (SELECT *
                    FROM "Document X" DX
                    WHERE DY.Name = DX.Author)
```

(5) 検索結果を取得する順序を指定した属性検索

ここでは，取得する検索結果の順序を指定する場合の属性検索の指定例を示します。

Document X クラスの文書のうち，CreateDate プロパティの値が 20000101 よりも大きいオブジェクトを検索して，Title プロパティ，Author プロパティ，および CreateDate プロパティを取得します。このとき，Author プロパティによって昇順に並び替え，その中で CreateDate プロパティで降順に並び替えて検索結果を取得する例を示します。まず，並び替えのキーとしてソート項目指定番号を指定する例を示します。ソート項目番号は，Author プロパティは SELECT 句で 2 番目に指定されているので「2」，CreateDate プロパティは SELECT 句で 3 番目に指定されているので「3」になります。

指定例

```
SELECT Title,Author,CreateDate
FROM "Document X"
WHERE CreateDate > '20000101'
ORDER BY 2 ASC,3 DESC
```

これを，プロパティ名で指定すると，次のようになります。

指定例

```
SELECT Title,Author,CreateDate
FROM "Document X"
WHERE CreateDate > '20000101'
ORDER BY Author ASC,CreateDate DESC
```

(6) VariableArray 型プロパティを指定した属性検索

ここでは，VariableArray 型のプロパティの検索方法について説明します。

(a) VariableArray 型のプロパティを取得する検索

ここでは，myPaper クラスの VariableArray 型プロパティである Authors プロパティを取得する例を示します。

指定例

```
SELECT Title,Authors
FROM myPaper
WHERE CreateDate > '20000101'
```

(b) VariableArray 型プロパティの要素の値を指定した検索

ここでは，myPaper クラスの VariableArray 型プロパティである Authors プロパティの，Org プロパティが「日立製作所」であるオブジェクトの Title プロパティと Authors プロパティを取得します。

指定例

```
SELECT Title,Authors
FROM myPaper
WHERE Authors[ANY].Org = '日立製作所'
```

(7) 集合関数を指定した属性検索

ここでは，COUNT(*) 関数および COUNT 関数を使用した検索の指定例を示します。

Document X クラスの文書のうち，作成日が「2000 年 1 月 1 日」よりも新しい（CreateDate プロパティの値が「20000101」よりも大きい）文書の数を取得します。

指定例

```
SELECT COUNT(*) FROM "Document X"
WHERE CreateDate > '20000101'
```

次に，Document X クラスの文書のうち，作成日が「2000 年 1 月 1 日」よりも新しい（CreateDate プロパティの値が「20000101」よりも大きい）文書の Authors プロパティの値の数を，重複排除して取得します。

指定例

```
SELECT COUNT(DISTINCT Authors) FROM "Document A"
WHERE CreateDate > '20000101'
```

(8) グループ分けをする属性検索

ここでは，GROUP BY 句を使用した検索方法について説明します。

Document X クラスの文書について，文書の Author プロパティでグループ化したときのグループ数を取得します。

指定例

```
SELECT Author, COUNT(*) FROM "Document X"
GROUP BY Author
```

次に，Document X クラスの文書について，文書の Author プロパティと Publisher プロパティとの組み合わせでグループ化し，Publisher プロパティの値が「日立製作所」であるグループの数を取得します。

指定例

```
SELECT Author, Publisher, COUNT(*)
FROM "Document X"
GROUP BY Author, Publisher
HAVING Publisher= '日立製作所'
```

4.6.2 全文検索（HiRDB Text Search Plug-in を利用した検索）

ここでは，HiRDB Text Search Plug-in を利用して実現する全文検索の，edmSQL での指定例を示します。

なお，< 全文検索条件 > および < 概念検索条件 > の指定方法については，マニュアル「HiRDB Text Search Plug-in」を参照してください。

(1) 指定例で使用するクラスとプロパティ

この指定例で使用する文書は，次のようなクラスから作成された文書です。

文書 X

Document X クラスを基に作成された文書です。Document X クラスは，全文検索の対象になる文書クラスとして作成されており，次の表に示すプロパティが設定されています。

表 4-47 Document X クラスのプロパティ

プロパティ識別子	内容
Code	文書を管理するためのコード
Title	タイトル
Author	著者
Abstract	概要
CreateDate	作成日
edmProp_ConceptStIndex	全文検索インデクス（概念検索および構造検索が可能な全文検索インデクス）

全文検索の対象になる文書の作成方法については，「4.4 全文検索の対象になる文書の作成」を参照してください。

また，Document X クラスは，構造を持った XML 文書の基になる DMA クラスです。構造『文章』の下位に，構造『名前』，構造『キーワード』および構造『本文』が定義されています。

Document X クラスを基に作成した XML 文書の例を次に示します。

Document X クラスを基に作成した XML 文書の例

```
<?xml version="1.0" encoding="Shift_JIS"?>
・・・
<文章>
  <名前>
   日立太郎
  </名前>
  <キーワード>
   edmSQL・・・
  </キーワード>
  <本文>
   edmSQLとは，SQLに準拠した・・・。
  </本文>
</文章>
```

(2) contains 関数を使用した全文検索

ここでは，contains 関数を使用した全文検索の指定方法について説明します。

Document X クラスの文書のうち，最上位の構造「文章」の下にある構造「本文」に，「コンピュータ」を含む文書の OIID を取得します。ただし，「コンピュータ」は，同義語辞書「myDic」を使用して，同義語展開して検索します。

指定例

```
SELECT dmaProp_OIID
FROM   "Document X"
WHERE  contains(edmProp_ConceptStIndex,
       '文章[本文{SYNONYM(myDic,"コンピュータ")}]') IS TRUE
```

（3） extracts 関数を使用した全文検索

ここでは，extracts 関数を使用した全文検索の指定方法について説明します。

Document X クラスの文書のうち，最上位の構造「文章」の下にある構造「本文」に，「Computer」を含む文書のコンテントを検索して，「Computer」に <STRONG> タグを付けたテキストを出力します。

指定例

```
SELECT extracts(edmProp_ConceptStIndex
 '文章.本文','文章[本文{"COMPUTER"}]', 'STRONG')
FROM  "Document X"
WHERE contains(edmProp_ConceptStIndex,
       '文章[本文{"COMPUTER"}]') IS TRUE
```

（4） score 関数と contains_with_score 関数を使用した全文検索

ここでは，score 関数と contains_with_score 関数を使用した全文検索の指定方法について説明します。

Document X クラスの文書のうち，最上位の構造「文章」の下にある構造「本文」に，「コンピュータ」を含む文書の OIID を取得します。ただし，「コンピュータ」は，同義語辞書「myDic」を使用して，同義語展開して検索します。

また，検索結果として，Title プロパティ，Author プロパティとともに，検索条件に対する文書のスコア（score）を取得します。score は，contains_with_score 関数によって算出されます。したがって，スコア値を取得する検索を実行する場合は，このように score 関数と contains_with_score 関数を組み合わせて指定します。

さらに，この例では，取得したスコア値を基に，検索結果を昇順にソートします。

指定例

```
SELECT dmaProp_OIID,Title,Author,score(edmProp_ConceptStIndex)
FROM   "Document X"
WHERE contains_with_score(edmProp_ConceptStIndex,
        '文章[本文{SYNONYM(myDic,"コンピュータ")}]') IS TRUE
ORDER BY 3 ASC
```

（5） score_concept 関数と concept_with_score 関数を使用した概念検索

ここでは，score_concept 関数と concept_with_score 関数を使用した概念検索の指定方法について説明します。

Document X クラスの文書のうち，種文章と近い概念を含んでいる文書を検索します。種文章は，? パラメタとして指定します。したがって，実際の種文章の内容は，Binary 型のデータとして，CdbrEqlStatement::SetParam メソッドによって指定してください。種文章は 5 メガバイトまで指定できます。

この例では，種文章から抽出された検索用特徴タームを，同義語辞書 myDic を使用して同義語展開して検索します。

指定例

```
SELECT dmaProp_OIID,score_concept(edmProp_ConceptStIndex)
FROM   "Document X"
WHERE concept_with_score
       (edmProp_ConceptStIndex, ? AS BINARY (5m));
```

4.7 edmSQL の障害対策とデバッグ

ここでは，edmSQL 検索の実行時に障害が発生したときの障害対策と，デバッグ情報について説明します。

4.7.1 CdbrEqlStatement クラスが出力する障害情報

edmSQL の検索を実行する CdbrEqlStatement クラスの各メソッドには，エラーが発生した時に，次の障害情報を出力する機能があります。これらの情報は，エラーの原因を調査するために使用できます。

- 戻り値
- 詳細エラーコード
- トレース出力
- 詳細エラーメッセージ
- edmSQL 文構文解析情報

詳細エラーメッセージおよび edmSQL 文構文解析情報の出力先と，必要な環境変数の設定および使用方法を，次の表に示します。なお，これらの情報はすべてクライアントに出力されます。

表 4-48 CdbrEqlStatement クラスで出力する障害情報

情報の種類	出力先	必要な環境変数	出力方法
詳細エラーメッセージ	詳細エラーメッセージファイル	• DBR_DETAIL_ERRORLOG • DBR_DETAIL_ERRORLOG_DIR • DBR_DETAIL_ERRORLOG_SIZE	DBR_DETAIL_ERRORLOG_DIR に指定したパスのファイルを取得します。
	詳細エラーメッセージ	特になし。	CdbrSession::GetLastDetailError メソッドをコールします。
edmSQL 文構文解析情報	edmSQL 構文解析情報ファイル	• DBR_EQL_PARSELOG • DBR_EQL_PARSELOG_DIR	DBR_EQL_PARSELOG_DIR に指定したパスのファイルを取得します。
	構文解析結果メッセージ	特になし。	CdbrEqlStatement::Execute メソッドの ppParseMessage を指定します。

詳細エラーメッセージに関する環境変数の設定方法については，「5. 環境設定」を参照してください。

4.7.2 詳細エラーメッセージ

DocumentBroker のほかのクラスと同様，エラーになったメソッド，戻り値の詳細およびその原因を，クライアント環境のファイルおよびメモリに出力します。

なお，CdbrEqlStatement クラスでは，メソッドをコールした時にエラーになった原因を，詳細エラーメッセージとして出力します。

出力されるメッセージについては，マニュアル「DocumentBroker Version 3 メッセージ」を参照してください。

4.7.3 edmSQL 構文解析情報

edmSQL 文を構文解析して，構文解析情報をクライアントのファイルおよびメモリに出力します。構文解析情報は，実行した edmSQL 文の情報および構文解析時に出力されたメッセージの情報です。

構文解析情報には，次の 2 種類があります。

● edmSQL **構文解析情報ファイル**
edmSQL **構文解析情報ファイル**とは，edmSQL を構文解析した時のトレース情報が出力されるクライアントのファイルです。

● **構文解析結果メッセージ**
検索を実行する CdbrEqlStatement::Execute メソッドをコールした時に，引数 ppParseMessage として出力されます。

（1）edmSQL 構文解析情報ファイル

edmSQL 構文解析情報ファイルの出力方法および出力される情報について説明します。

出力方法

edmSQL 構文解析情報ファイルを出力する場合は，次の環境変数の指定が必要です。

- DBR_EQL_PARSELOG
  出力する場合は，この環境変数に ON を設定します。なお，デフォルトは OFF です。したがって，この環境変数を設定していない場合，edmSQL 構文解析情報ファイルは出力されません。
- DBR_EQL_PARSELOG_DIR
  edmSQL 構文解析情報ファイルの出力パスを指定します。なお，デフォルトは実行ファイルと同じパスになります。指定したパスが存在しない場合は，edmSQL 構文情報ファイルは出力されません。
  文書空間で使用する文字コード種別が UTF-8 の場合，印刷可能な ASCII コードで記述できる値を設定してください。

edmSQL 構文解析情報ファイルのファイル名

edmSQL 構文解析情報ファイルのファイル名は，日時を基に，次のように決定されます。なお，ファイル名は，マルチスレッド対応とシングルスレッド対応の場合で異なります。

形式

（マルチスレッド対応の場合）

```
EDMEQLINFOxxxxxxxxxxxxx_yyyyyyyy_zzzzzzzz.log
- EDMEQLINFO：プレフィクス
- xxxxxxxxxxxxx(13バイト)：日時から生成された値
- yyyyyyyy(8バイト)：スレッドIDで生成した値
- zzzzzzzz(8バイト)：プロセスIDで生成した値
```

（シングルスレッド対応の場合）

```
EDMEQLINFOxxxxxxxxxxxxx_yyyyyyyy.log
- EDMEQLINFO：プレフィクス
- xxxxxxxxxxxxx(13バイト)：日時から生成された値
- yyyyyyyy(8バイト)：プロセスIDで生成した値
```

説明

例えば，2000 年 1 月 1 日 午後 6 時 30 分 1.111 秒に出力する場合，xxxxxxxxxxxxx は「0101183001111」になります（年の値は含まれません）。
なお，出力したファイルの名称は，詳細メッセージ（KMBR22021-E）に出力されます。詳細メッセージについては，マニュアル「DocumentBroker Version 3　メッセージ」を参照してください。

edmSQL 構文解析情報ファイルに出力される内容

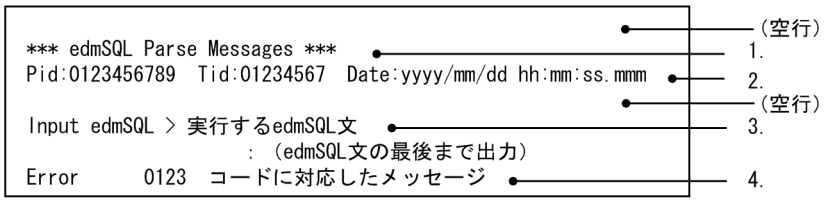
ファイルには，次の場合に，CdbrEqlStatement::Execute メソッドで出力される構文解析情報が出力されます。

- 構文解析を実行した時に構文解析エラーが検出された場合
- 構文解析を実行した時にワーニングが検出された場合

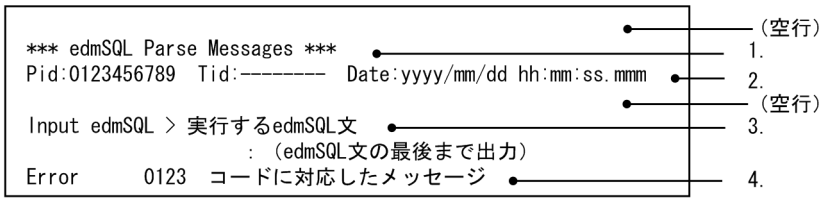
なお，構文解析を実行した edmSQL 文が正しい構文である場合は，ファイルは出力されません。
次に，edmSQL 構文解析情報ファイルの出力形式を，次の図に示します。なお，出力形式は，マルチスレッド対応の場合とシングルスレッド対応の場合で異なります。

図 4-2　edmSQL 構文解析情報ファイルの出力形式

（マルチスレッド対応の場合）

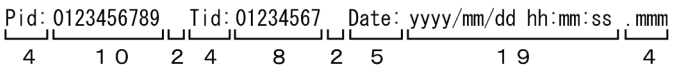


（シングルスレッド対応の場合）

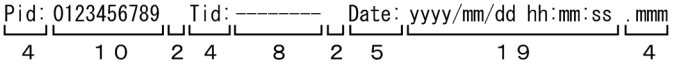


1. ヘッダー情報が 29 カラムで出力されます。
2. 次のように出力されます。マルチスレッド対応の場合とシングルスレッド対応の場合で出力内容が異なります。

（マルチスレッド対応の場合）



（シングルスレッド対応の場合）



文字の下の数字はカラム数です。

3.「Input edmSQL >」は 15 カラムで出力されます。「実行する edmSQL 文」は，任意の指定した edmSQL 文の長さです。
4.「Error」は，10 カラムで出力されます。空白が 2 カラム出力されて，コードを表す数字が 4 カラムで出力され，「コードに対応したメッセージ」が任意の長さで出力されます。
出力されるメッセージの内容と対処方法については，「付録 B　edmSQL の構文解析エラー情報」を参照してください。

(2) 構文解析結果メッセージ

メソッドをコールした時に出力される構文解析結果メッセージについて説明します。

必要な設定

このメッセージは，CdbrEqlStatement::Execute メソッドで，出力引数 ppParseMessage を指定した場合に出力されます。なお，ppParseMessage を指定しないでこのメソッドをコールした場合は，エラーが発生しても，メッセージは出力されません。
また，ほかの詳細メッセージとは異なり，環境変数「DBR_EQL_PARSELOG」に OFF を指定した

場合も，引数を指定すれば，メッセージが取得できます。

返却されるメッセージの内容

返却されるメッセージの内容については，マニュアル「DocumentBroker Version 3　メッセージ」を参照してください。なお，構文解析した edmSQL 文の構文が正しい場合に，ppParseMessage を指定しているときは，指定した領域に NULL が設定されて返却されます。

構文解析中にエラーまたはワーニングが検出された場合，この領域に返却されるメッセージは最初に検出したエラーまたはワーニングに関するメッセージです。

構文解析のトレースを確認する場合は，「(1)edmSQL 構文解析情報ファイル」を参照して，トレース内容をファイルに出力して確認してください。

構文解析結果メッセージを取得したエラー処理の例

次に，構文解析結果メッセージを取得する場合の，エラー処理の例を示します。

```
DmaBoolean brc = DMA_FALSE;
pDmaString_T pParseMessage = NULL;
DmaInteger32 i32MajorCode = ( -1 );
DmaInteger32 i32MinorCode = ( -1 );

brc = cEql->Execute( &pParseMessage );

if ( DMA_TRUE != brc ) {
  //エラー処理
  i32MajorCode = GetLastError( &i32MinorCode );
  if ( ( ERR_DBR == i32MajorCode ) &&
       ( ERR_EQL_BAD_STATEMENT == i32MinorCode ) )
  {
    //構文解析エラーを表示
    printf( "構文解析エラー：%s¥n", pParseMessage );
    dbrDelete( pParseMessage );
    pParseMessage = NULL;
    goto ErrorExit;
  }
}
```

4.7.4　CdbrEqlStatement クラスで出力するトレース情報

CdbrEqlStatement クラスでは，サーバトレースとクライアントトレースにトレース情報を出力します。エラーが発生した場合，トレース情報を基にエラーの原因が調査できます。

トレースの出力機能についての詳細は，マニュアル「DocumentBroker Version 3　システム導入・運用ガイド」を参照してください。ここでは，それぞれのトレースレベルで CdbrEqlStatement クラスが出力するトレース情報の内容について説明します。

トレースレベルごとの，サーバのトレースファイルに出力される CdbrEqlStatement クラスの出力情報を，次の表に示します。

表 4-49　トレースレベルとサーバトレースファイルへの出力情報

トレースレベル	出力情報
0 以上	• 入力した edmSQL 文と構文解析後に生成した SQL 文 この情報は，構文解析エラーの原因を調査するための情報として，必ず出力されます。 CdbrEqlStatement::ExecuteSearch メソッドをコールするたびに出力されます。 • エラー情報 エラーが発生した場合の，詳細エラーコードとその原因が出力されます。
10 以上	ほかのメソッドで出力される情報と同じです。

クライアントトレースファイルに出力される情報は，ほかのメソッドで出力される情報と同じです。

4.8 edmSQL 検索での留意事項

ここでは，edmSQL を使用して検索を実行する場合に，留意する必要がある内容について説明します。

4.8.1 プロパティに関する制限事項

「dbrProp_」で始まるクラスライブラリで定義したプロパティは，検索条件には指定できません。

また，「dmaProp_」および「edmProp_」で始まるプロパティについては，検索できるプロパティと検索できないプロパティがあります。検索できるプロパティについては，「4.3　検索対象になるクラスおよびプロパティ」を参照してください。

4.8.2 検索条件に指定する値の制限事項

ここでは，検索条件に設定できる値についての制限事項を示します。

なお，制限事項には，DocumentBroker としての制限のほか，データベースの制限に基づくものもあります。指定する値のバイト数などについては，データベースである HiRDB の制限に従います。HiRDB の制限を超えた検索を実行した場合は，データベースエラーとして，次のエラーが返却されます。

major_code：ERR_DB　minor_code：ERR_DB_FAILED

HiRDB の制限事項の詳細については，マニュアル「HiRDB　SQL リファレンス」を参照してください。

ここでは，FROM 句，SELECT 句，WHERE 句および ORDER BY 句に指定する内容に関する制限事項などについて説明します。

(1) FROM 句に関する制限事項

ロックを設定するインターフェースを使用した場合に，ロック種別に DBR_LOCK_WRITE を指定したときに，次の条件で検索を実行すると，データベースエラーになります。

- 主問い合わせで FROM 句に指定したクラスを，副問い合わせの FROM 句に指定した条件
- 主問い合わせでクラスの結合を指定した条件

(2) SELECT 句に関する制限事項

● 副問い合わせの SELECT 句に指定できるプロパティは一つです。複数指定した場合は，次のエラーが返却されます。
major_code：ERR_DBR　minor_code：ERR_EQL_BAD_STATEMENT

● また，ここで指定できるプロパティのデータ型も，副問い合わせで指定している < 述語 > の規則に従います。不正なデータ型のプロパティを指定した場合には，次のエラーが返却されます。
major_code：ERR_DBR　minor_code：ERR_EQL_BAD_STATEMENT

● SELECT 句に String 型のプロパティを指定して重複排除をする場合，SELECT 句には 255 バイト以内で定義している String 型のプロパティを指定してください。定義が 255 バイトを超える String 型のプロパティを指定した場合，重複排除は実行できません。実行した場合，データベースエラーになります。

● ロックを取得するインターフェースを使用する場合に，ロックの種別に DBR_LOCK_WRITE を指定して，次の条件での検索を実行すると，データベースエラーになります。
- 主問い合わせに重複排除（DISTINCT）を指定した条件。

- 主問い合わせに COUNT 関数を指定した条件。

● VariableArray 型プロパティを指定して，重複排除を指定すると，次のエラーが返却されます。
major_code：ERR_DBR minor_code：ERR_EQL_BAD_STATEMENT

● VariableArray 型プロパティのヒット件数を COUNT 関数で取得しようとすると，データベースエラーになります。

(3) WHERE 句に関する制限事項

● WHERE 句に文字列定数を使用した条件を指定する場合，255 バイト以内で指定してください。制限を超えて指定した場合は，データベースエラーになります。

● WHERE 句に String 型のプロパティを指定する場合は，255 バイト以内で定義されたプロパティを指定してください。制限を超えたプロパティを指定した場合は，データベースエラーになります。

● WHERE 句で指定する論理演算のネスト数は，255 個以内で指定してください。255 個を超えた場合は，データベースエラーになります。

(4) ORDER BY 句に関する制限事項

● ORDER BY 句で指定できる SELECT 句でのプロパティのインデクス，またはプロパティは，255 個以内で指定してください。255 個を超えた場合は，データベースエラーになります。

● ORDER BY 句に String 型のプロパティを指定する場合は，255 バイト以内で定義されたプロパティを指定してください。制限を超えたプロパティを指定した場合は，データベースエラーになります。

(5) 検索条件として指定できる値についての制限

● 検索条件として指定するプロパティの値や，全文検索で指定できる文字については，制限があります。検索条件として指定できるクラスやプロパティについては，「4.3　検索対象になるクラスおよびプロパティ」を参照してください。また，edmSQL 文に指定できる値については，「4.5.4　字句規則」を参照してください .。

(6) オペレータに指定する値についての制限

● 次のような演算を指定した場合は，データベースエラーになります。
- 比較演算の両辺に値を指定した検索は，データベースエラーになります。
例えば，< 比較演算子 >「=」の両辺に，< リテラル > または <? パラメタ > を指定した検索などはできません。
- 右辺に値を指定する IN 述語の左辺に値を指定した検索は，データベースエラーになります。
例えば，<In 述語 > の両辺に < 文字列リテラル > を指定した検索などはできません。

● / 演算子の右辺（除数）として，0 を指定すると，データベースエラーになります。

そのほか，edmSQL 文を指定する際の詳細な規則については，「4.5　edmSQL の文法」を参照してください。

(7) ? パラメタに関する制限事項

<? パラメタ > に文字列を指定する場合は，32,000 バイト以内の文字列を指定してください。制限を超えて指定した場合は，データベースエラーになります。

(8) 問い合わせ指定全体に関する制限事項

一つの問い合わせ指定内で，DISTINCT を 2 回以上指定することはできません。

4.8.3 複数のクラスを対象にした検索の制限事項

複数クラスを対象にした検索での制限は，HiRDB の制限に従います。また，複数クラスを対象にした全文検索での制限は，HiRDB Text Search Plug-in および HiRDB の制限に従います。

それぞれの制限については，マニュアル「HiRDB　SQL リファレンス」およびマニュアル「HiRDB Text Search Plug-in」の，表の結合に関する規則の説明を参照してください。

4.8.4 アクセス制御機能付き検索を実行する場合の制限事項

- アクセス制御機能付き検索は，必ずアクセス制御機能を使用している文書空間で実行してください。また，アクセス制御機能に対応していないクライアントアプリケーションをアクセス制御機能を使用している文書空間で使用した場合，検索結果が不正になることがあります。文書空間の設定と，クライアントアプリケーションの機能は一致させてください。
  アクセス制御機能付き検索では，検索結果の重複排除は指定できません。指定した場合は，次のエラーが返却されます。
  `major:ERR_DBR　minor：ERR_EQL_BAD_STATEMENT`

- 検索結果の個数を取得する検索は実行できません。SELECT 句に COUNT 関数を指定することはできません。指定した場合は，次のエラーが返却されます。
  `major:ERR_DBR　minor：ERR_EQL_BAD_STATEMENT`

  検索結果の個数は，GetResult メソッドで，検索結果を全件取得することで取得してください。

- 副問い合わせは指定できません。
  副問い合わせを実行した場合，副問い合わせの検索結果に対してアクセス制御がされません。
  例えば，次のような検索の場合，検索結果が不正になる可能性があります。
  ［例］

```
SELECT S0.PropA
FROM ClassA As S0
WHERE S0,PropB In
(SELECT S1.PropC FROM ClassB AS S1 WHERE S1.PropD='mojiretsu')
```

  このとき，斜体で記述した副問い合わせの結果に対してはアクセス制御されません。つまり，ユーザが参照権を持たないオブジェクトも検索結果として取得できます。このため，副問い合わせの結果として取得したオブジェクトには，ユーザのアクセス権では参照できないオブジェクトが含まれている可能性があります。
  また，全文検索を含む副問い合わせの検索結果には，プロパティを参照する権利を持たないオブジェクトのほか，コンテントを参照する権利がないオブジェクトが含まれる可能性があります。
  アクセス制御機能付き検索で副問い合わせに相当する検索を実行したい場合は，複数の SELECT 句に分けて，複数回 Execute メソッドをコールして検索を実行してください。

- GROUP BY 句を使用した問い合わせは指定できません。

- アクセス制御機能付き検索では，ユーザが SELECT 句に指定したプロパティ以外に DocumentBroker によってアクセス制御情報を表すプロパティが取得されています。このため，ユーザが SELECT 句に指定できるプロパティ数が，通常の検索で指定できる数（データベースの制限値）よりも少なくなります。

- 左外部結合（LEFT OUTER JOIN）を <FROM 句 > に指定し，左側のクラスのオブジェクトが < 検索条件 > を満たさない場合，左側のクラスのプロパティは NULL となります。ユーザはこの NULL となったプロパティに対して参照権を持ちます。

4.8.5 コーディング上の留意事項

複数の CdbrEqlStatement オブジェクトを使用して検索を実行する場合に，個々の検索を終了しないで多数の検索を実行すると，エラーになることがあります。

複数の検索を実行する場合も，一つ一つの検索をそのつど終わらせるようなコーディングにすることをお勧めします。

ここでは，エラーになる可能性があるコーディング例と，それを改善した推奨コーディング例を示します。

(1) エラーになる可能性のあるコーディング

ここでは，エラーになる可能性があるコーディング例を示します。

なお，処理の流れを明確にするため，引数は省略しています。

```
// 変数宣言
CdbrEqlStatement* pEqlStatement1 = NULL;
CdbrEqlStatement* pEqlStatement2 = NULL;
     .
     .
     .
// オブジェクトを作成する。
pEqlStatement1 = new CdbrEqlStatement();
pEqlStatement2 = new CdbrEqlStatement();
     .
     .
     .
////////////////////////////////////
// オブジェクトを初期化する。
////////////////////////////////////
rc = pEqlStatement1->Initialize(...);
if(DMA_TRUE != rc)
{
    // エラー処理
}
rc = pEqlStatement2->Initialize(...);
if(DMA_TRUE != rc)
{
    // エラー処理
}
     .
     .
     .
////////////////////////////////////
// 実行するedmSQL文を設定する。
////////////////////////////////////
rc = pEqlStatement1->Set(...);
if(DMA_TRUE != rc)
{
    // エラー処理
}
rc = pEqlStatement2->Set(...);
if(DMA_TRUE != rc)
{
    // エラー処理
}
     .
     .
     .
////////////////////////////////////
// 検索を実行する。
////////////////////////////////////
rc = pEqlStatement1->Execute(...);
```

```
if(DMA_TRUE != rc)
{
    // エラー処理
}
rc = pEqlStatement2->Execute(...);
if(DMA_TRUE != rc)
{
    // エラー処理
}
     .
     .
     .
////////////////////////////////////
// 検索結果を取得する。
////////////////////////////////////
rc = pEqlStatement1->GetResult(...);
if(DMA_TRUE != rc)
{
    // エラー処理
}
rc = pEqlStatement2->GetResult(...);
if(DMA_TRUE != rc)
{
    // エラー処理
}
     .
     .
     .
```

(2) 推奨するコーディング

ここでは，推奨するコーディング例を示します。

なお，処理の流れを明確にするため，引数は省略しています。

```
// 変数宣言
CdbrEqlStatement* pEqlStatement1 = NULL;
CdbrEqlStatement* pEqlStatement2 = NULL;
     .
     .
     .
// オブジェクトを作成する。
pEqlStatement1 = new CdbrEqlStatement();
pEqlStatement2 = new CdbrEqlStatement();
     .
     .
     .
////////////////////////////////////
// オブジェクトを初期化する。
////////////////////////////////////
rc = pEqlStatement1->Initialize(...);
if(DMA_TRUE != rc)
{
    // エラー処理
}
////////////////////////////////////
// 実行するedmSQL文を設定する。
////////////////////////////////////
rc = pEqlStatement1->Set(...);
if(DMA_TRUE != rc)
{
    // エラー処理
}
////////////////////////////////////
// 検索を実行する。
```

```
////////////////////////////////////
rc = pEqlStatement1->Execute(...);
if(DMA_TRUE != rc)
{
    // エラー処理
}
////////////////////////////////////
// 検索結果を取得する。
////////////////////////////////////
rc = pEqlStatement1->GetResult(...);
if(DMA_TRUE != rc)
{
    // エラー処理
}
////////////////////////////////////
// 検索を終了する。
////////////////////////////////////
rc = pEqlStatement1->Terminate();
if(DMA_TRUE != rc)
{
    // エラー処理
}
////////////////////////////////////
// オブジェクトを削除する。
////////////////////////////////////
delete pEqlStatement1;
////////////////////////////////////
// オブジェクトを初期化する。
////////////////////////////////////
rc = pEqlStatement2->Initialize(...);
if(DMA_TRUE != rc)
{
    // エラー処理
}
////////////////////////////////////
// 実行するedmSQL文を設定する。
////////////////////////////////////
rc = pEqlStatement2->Set(...);
if(DMA_TRUE != rc)
{
    // エラー処理
}
////////////////////////////////////
// 検索を実行する。
////////////////////////////////////
rc = pEqlStatement2->Execute(...);
if(DMA_TRUE != rc)
{
    // エラー処理
}
////////////////////////////////////
// 検索結果を取得する。
////////////////////////////////////
rc = pEqlStatement2->GetResult(...);
if(DMA_TRUE != rc)
{
    // エラー処理
}
////////////////////////////////////
// 検索を終了する。
////////////////////////////////////
rc = pEqlStatement2->Terminate();
if(DMA_TRUE != rc)
{
    // エラー処理
}
```

```
////////////////////////////////////
// オブジェクトを削除する。
////////////////////////////////////
delete pEqlStatement2;
```

5 環境設定

この章では，DocumentBroker のクラスライブラリを使用するときに必要な環境設定について説明します。

5.1 インストール後の環境設定

DocumentBroker をインストールしたあとの環境設定について説明します。

5.1.1 起動ユーザとユーザ認証

DocumentBroker のクライアントアプリケーションとサーバを同じマシン上で動作させる場合，DocumentBroker のクライアントアプリケーションを起動するユーザと DocumentBroker のシステム管理者は一致させておいてください。また，DocumentBroker は，OS のファイルシステムにアクセスする場合，DocumentBroker のシステム管理者の権限でアクセスします。DocumentBroker に対して認証したユーザの権限ではアクセスしませんのでご注意ください。

なお，Windows Server 2008，Windows Server 2008 R2，または Windows Server 2012 を使用している場合，DocumentBroker のクライアントアプリケーションは管理者特権で実行する必要があります。

5.1.2 TPBroker での環境設定

DocumentBroker のクラスライブラリとサーバの通信には，TPBroker の CORBA 通信を利用します。

TPBroker の CORBA 通信を利用するために必要な環境設定については，マニュアル「DocumentBroker Version 3　システム導入・運用ガイド」の TPBroker での環境設定についての説明を参照してください。

5.1.3 実行環境の設定

DocumentBroker のクラスライブラリの起動に必要な環境変数や，パスの指定などについて説明します。なお，DocumentBroker のクライアントアプリケーションとサーバを同じマシン上で動作させる場合，クライアント用とサーバ用の環境変数を混在させないようにご注意ください。

また，XML 文書管理機能（XML プロパティマッピング機能・XML インデクスデータ作成機能）を使用する場合は，ここで説明する環境変数のほかに，HiRDB Adapter for XML の環境変数を設定する必要があります。

XML 文書管理機能を使用するための環境設定については，マニュアル「DocumentBroker Version 3　システム導入・運用ガイド」を参照してください。

指定する環境変数を，OS ごとに説明します。

(1) 環境変数（AIX の場合）

DocumentBroker のクライアントライブラリを起動するために必要な次の環境変数を指定してください。

● LANG

環境変数「LANG」には，使用する文字コードセットを設定します。DocumentBroker の文書空間で使用する文字コード種別に合わせて値を設定してください。

文書空間で使用する文字コード種別が Shift-JIS の場合

設定する値は「Ja_JP」です。

文書空間で使用する文字コード種別が UTF-8 の場合

使用する言語に合わせて適切な値を設定してください。

ただし，次のときは「C」を設定してください。

- 文書空間で使用する文字コード種別に関係なく，ASCII コードのデータだけを扱う英語環境の場合

- 一つのアプリケーションから文字コード種別の異なる複数の文書空間に接続する場合に，接続先に文字コード種別が UTF-8 の文書空間が含まれるとき

なお，この環境変数に設定する値によって，メッセージの言語種別が変わります。「Ja_JP」を設定した場合は，日本語のメッセージが出力されます。「Ja_JP」以外を設定した場合は，英語のメッセージが出力されます。

● TZ

環境変数「TZ」には，マシン時間のタイムゾーンを設定します。設定する値は「JST-9」です。

● XDK_HOME

環境変数「XDK_HOME」には，SystemManager オブジェクトのレジストリファイルのロケーションを設定します。指定する値は「/opt/HiEDMS/client/etc」です。

● LIBPATH

環境変数「LIBPATH」には，DocumentBroker のクライアントライブラリのインストールディレクトリを指定します。次の値を追加してください。

TPBroker V3 を使用している場合

```
:/opt/hitachi/common/lib
:/usr/vacpp/lib※1
:/opt/TPBroker/lib
:/opt/HiEDMS/client/lib
:/opt/HiEDMS/ACLibrary/lib
:/opt/hirdb_xml/lib※2
:/opt/hitachi/xpk/lib※2
```

TPBroker V5 を使用している場合

```
:/opt/hitachi/common/lib
:/usr/vacpp/lib※1
:TPBroker V5インストールディレクトリ/lib
:/opt/HiEDMS/client/lib_tp5
:/opt/HiEDMS/ACLibrary/lib_tp5
:/opt/hirdb_xml/lib※2
:/opt/hitachi/xpk/lib※2
```

注※ 1　VisualAge C++ Professional for AIX または IBM XL C/C++ Enterprise Edition for AIX のライブラリの格納ディレクトリを設定します。これは，デフォルトのインストールディレクトリにインストールされている場合の値です。
注※ 2　XML 文書管理機能（XML プロパティマッピング機能・XML インデクスデータ作成機能）を使用する場合に設定します。

● PSALLOC

環境変数「PSALLOC」には，次の値を設定します。

```
early
```

● NODISCLAIM

環境変数「NODISCLAIM」には，次の値を設定します。

```
true
```

● DBR_CONNECTSV_ENVID

環境変数「DBR_CONNECTSV_ENVID」には，接続先の実行環境識別子を設定します。同一文書空間に実行環境が複数作成されていて，それらが同時に稼働しているサーバに接続する場合に，クライアントで接続先を選択するときに設定します。
設定できる値は，「0 ～ 254」です。
同一文書空間の実行環境として複数のサーバが存在している場合に，この環境変数を設定しないとき

は，クライアントで接続するサーバを選択しません。設定できる値以外を指定した場合は，設定されていないものとして動作します。
接続先の実行環境識別子を CdbrSession::Connect メソッドの引数に指定することもできます。引数で指定する場合については，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。
CdbrSession::Connect メソッドの引数の指定と，この環境変数の関係は，次のようになります。

表 5-1　CdbrSession::Connect メソッドの引数の指定と，環境変数「DBR_CONNECTSV_ENVID」の関係（AIX の場合）

環境変数		CdbrSession::Connect メソッドへの引数		
		指定あり		指定なし
		0 ～ 254	範囲外	
指定あり	0 ～ 254	引数で指定した実行環境識別子のサーバに接続	環境変数で指定した実行環境識別子のサーバに接続	環境変数で指定した実行環境識別子のサーバに接続
	範囲外	引数で指定した実行環境識別子のサーバに接続	接続するサーバを選択しない	接続するサーバを選択しない
指定なし		引数で指定した実行環境識別子のサーバに接続	接続するサーバを選択しない	接続するサーバを選択しない

クライアントで接続先を選択する場合は，接続先の DocumentSpace 構成定義ファイルの SelectServerInMultiServer エントリに Yes を指定して起動しておく必要があります。

● DBR_CON_TIMEOUT

環境変数「DBR_CON_TIMEOUT」には，サーバへの接続要求に対するクライアントでの接続確立待ち時間（秒）を設定します。
設定できる値は，「0 ～ 2,147,483,647」です。
0 を設定した場合，TCP/IP で固有のサーバへの接続確立待ち時間（秒）が設定されます。
範囲外の値を設定した場合は，0 が仮定されます。

● DBR_FLINK_CONTENT_DELETE

環境変数「DBR_FLINK_CONTENT_DELETE」には，File Link 連携機能を使用して管理している文書のコンテントを RemoveObject メソッドまたは DeleteRendition メソッドによって削除する場合に，ファイルサーバ上のファイルの実体を削除することを指定します。次の値を指定します。

```
ON
```

● DBR_DETAIL_ERRORLOG

環境変数「DBR_DETAIL_ERRORLOG」は，詳細メッセージを取得する場合に指定します。詳細メッセージを取得する場合には，次の値を指定してください。

```
ON
```

● DBR_DETAIL_ERRORLOG_DIR

環境変数「DBR_DETAIL_ERRORLOG_DIR」には，詳細メッセージを取得する場合に，詳細エラーログファイルを出力するためのパス名を指定します。なお，文書空間で使用する文字コード種別が UTF-8 の場合，印刷可能な ASCII コードで記述できる値を設定してください。詳細エラーログファイルは，指定したパス名のディレクトリ下に，次のファイル名で出力されます。

```
EDMErrTraceCLXXXXX_1.log
（XXXXXはプロセスID番号）
```

詳細エラーログファイルにはエラーが発生していない場合もログが出力されますので，必要に応じて詳細エラーログファイルを削除するようにしてください。

詳細エラーログファイルに出力される内容については，マニュアル「DocumentBroker Version 3 メッセージ」を参照してください。なお，指定を省略した場合，環境変数「DBR_DETAIL_ERRORLOG」に ON を指定していても，詳細エラーログファイルは出力されません。

● DBR_DETAIL_ERRORLOG_NUM

環境変数「DBR_DETAIL_ERRORLOG_NUM」には，詳細メッセージを取得する場合に使用する，ラップアラウンドする詳細エラーログファイルの個数を指定します。2,147,483,647 以下の値を指定してください。省略した場合の仮定値は 2 です。また，2 未満の値を指定した場合も，2 が仮定されます。

● DBR_DETAIL_ERRORLOG_SIZE

環境変数「DBR_DETAIL_ERRORLOG_SIZE」には，詳細メッセージを取得する場合に使用する詳細エラーログファイルの容量の上限値（バイト）を指定します。2,147,483,647 以下の値を指定してください。省略した場合の仮定値は 10,000 です。

0 以下の値を設定した場合，または 1 回の出力データ量よりも小さい値を指定した場合には，詳細メッセージは出力されません。

環境変数の値と詳細エラーログファイルのラップアラウンド処理

1 プロセスで長時間に及ぶ処理を実行する場合，出力される詳細エラーログファイルの容量が過剰に大きくなる可能性があります。

詳細エラーログファイルが必要以上の容量に増加するのを防ぐため，詳細エラーログファイルはラップアラウンド処理されます。詳細エラーログファイルの容量が環境変数で指定した最大容量になった場合，ラップアラウンド処理されて，詳細メッセージのログは最初の詳細エラーログファイルに出力されます。

最大容量とは，次の値です。

（詳細エラーログファイルのサイズ（DBR_DETAIL_ERROR_SIZEの指定値））×（詳細エラーログファイルの個数（DBR_DETAIL_ERROR_NUMの指定値））

この場合の詳細エラーログファイルの出力処理は，次のようになります。

1. 環境変数「DBR_DETAIL_ERRORLOG_DIR」に指定したディレクトリに，次の名称で詳細エラーログファイルが作成されます。（XXXXXX はプロセス ID，N はプロセス内のファイル番号）
   EDMErrTraceClXXXXXX_N.log
2. ファイルの容量が環境変数「DBR_DETAIL_ERROR_SIZE」で指定した容量になったら，次の詳細エラーログファイルが作成されます。
3. 環境変数「DBR_DETAIL_ERRORLOG_NUM」に指定した個数のファイルがすべて最大容量になったら，ラップアラウンドして再び先頭のファイルにログが出力されます。

また，edmSQL 構文解析情報ファイルを出力する場合は，次の環境変数の設定が必要です。

● DBR_EQL_PARSELOG

● DBR_EQL_PARSELOG_DIR

これらの環境変数の詳細は，「4.7.3(1)edmSQL 構文解析情報ファイル」を参照してください。

(2) 環境変数（Windows の場合）

DocumentBroker のクライアントライブラリを起動するために必要な次の環境変数を指定してください。

● DBR_CONNECTSV_ENVID

環境変数「DBR_CONNECTSV_ENVID」には，接続先の実行環境識別子を設定します。同一文書空間に実行環境が複数作成されていて，それらが同時に稼働しているサーバに接続する場合に，クライアントで接続先を選択するときに設定します。

設定できる値は，「0 ～ 254」です。

同一文書空間の実行環境として複数のサーバが存在している場合に，この環境変数を設定しないときは，クライアントで接続するサーバを選択しません。設定できる値以外を指定した場合は，設定されていないものとして動作します。
接続先の実行環境識別子を CdbrSession::Connect メソッドの引数に指定することもできます。引数で指定する場合については，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。
CdbrSession::Connect メソッドの引数の指定と，この環境変数の関係は，次のようになります。

表 5-2　CdbrSession::Connect メソッドの引数の指定と，環境変数「DBR_CONNECTSV_ENVID」の関係（Windows の場合）

環境変数		CdbrSession::Connect メソッドへの引数		
		指定あり		指定なし
		0 ～ 254	範囲外	
指定あり	0 ～ 254	引数で指定した実行環境識別子のサーバに接続	環境変数で指定した実行環境識別子のサーバに接続	環境変数で指定した実行環境識別子のサーバに接続
	範囲外	引数で指定した実行環境識別子のサーバに接続	接続するサーバを選択しない	接続するサーバを選択しない
指定なし		引数で指定した実行環境識別子のサーバに接続	接続するサーバを選択しない	接続するサーバを選択しない

クライアントで接続先を選択する場合は，接続先の DocumentSpace 構成定義ファイルの SelectServerInMultiServer エントリに Yes を指定して起動しておく必要があります。

● DBR_CON_TIMEOUT

環境変数「DBR_CON_TIMEOUT」には，サーバへの接続要求に対するクライアントでの接続確立待ち時間（秒）を設定します。
設定できる値は，「0 ～ 2,147,483,647」です。
0 を設定した場合，TCP/IP で固有のサーバへの接続確立待ち時間（秒）が設定されます。
範囲外の値を設定した場合は，0 が仮定されます。
既に一度接続したクライアントからサーバへの接続要求でタイムアウトが発生した場合，CdbrSession::Connect メソッドで，次のエラーが返却されます。
major_code：ERR_DMA　minor_code：DMARC_NETWORK_UNAVAILABLE
TPBroker V3 と連携して動作する環境の場合，エラーが返却されず，正常に接続できる場合があります。

● DBR_FLINK_CONTENT_DELETE

環境変数「DBR_FLINK_CONTENT_DELETE」には，File Link 連携機能を使用して管理している文書のコンテントを RemoveObject メソッドまたは DeleteRendition メソッドによって削除する場合に，ファイルサーバ上のファイルの実体を削除することを指定します。次の値を指定します。
ON

● DBR_DETAIL_ERRORLOG

環境変数「DBR_DETAIL_ERRORLOG」は，詳細メッセージを取得する場合に指定します。詳細メッセージを取得する場合には，次の値を指定してください。
ON

● DBR_DETAIL_ERRORLOG_DIR

環境変数「DBR_DETAIL_ERRORLOG_DIR」には，詳細メッセージを取得する場合に，詳細エラーログファイルを出力するためのパス名を指定します。なお，文書空間で使用する文字コード種別が

UTF-8 の場合，印刷可能な ASCII コードで記述できる値を設定してください。詳細エラーログファイルは，指定したパス名のディレクトリ下に，次のファイル名で出力されます。

```
EDMErrTraceCLXXXXX_1.log
（XXXXXはプロセスID番号）
```

詳細エラーログファイルにはエラーが発生していない場合もログが出力されますので，必要に応じて詳細エラーログファイルを削除するようにしてください。
詳細エラーログファイルに出力される内容については，マニュアル「DocumentBroker Version 3 メッセージ」を参照してください。なお，指定を省略した場合，環境変数「DBR_DETAIL_ERRORLOG」に ON を指定していても，詳細エラーログファイルは出力されません。

● DBR_DETAIL_ERRORLOG_NUM

環境変数「DBR_DETAIL_ERRORLOG_NUM」には，詳細メッセージを取得する場合に使用する，ラップアラウンドする詳細エラーログファイルの個数を指定します。2,147,483,647 以下の値を指定してください。省略した場合の仮定値は 2 です。また，2 未満の値を指定した場合も，2 が仮定されます。

● DBR_DETAIL_ERRORLOG_SIZE

環境変数「DBR_DETAIL_ERRORLOG_SIZE」には，詳細メッセージを取得する場合に使用する詳細エラーログファイルの容量の上限値（バイト）を指定します。2,147,483,647 以下の値を指定してください。省略した場合の仮定値は 10,000 です。
0 以下の値を設定した場合，または 1 回の出力データ量よりも小さい値を指定した場合には，詳細メッセージは出力されません。
環境変数の値と詳細エラーログファイルのラップアラウンド処理の関係については，「(1) 環境変数（AIX の場合）」を参照してください。
また，edmSQL 構文解析情報ファイルを出力する場合は，次の環境変数の設定が必要です。

● DBR_EQL_PARSELOG

● DBR_EQL_PARSELOG_DIR

これらの環境変数の詳細は，「4.7.3(1)edmSQL 構文解析情報ファイル」を参照してください。

(3) カレントディレクトリ

カレントディレクトリは，クライアントアプリケーションが格納されているディレクトリから移動しないでください。

5.2 プログラミング時の注意と設定

DocumentBroker のアプリケーションプログラムをクラスライブラリを使用して作成するときの注意事項および設定について説明します。

5.2.1 文字コード種別とプログラミング

クラスライブラリを使用して，アプリケーションを開発したり，開発したクライアントアプリケーションを使用して，文書空間オブジェクトを操作する場合，クラスライブラリを使用したプログラミングで使用する文字コード種別は，接続する文書空間で使用する文字コード種別に合わせる必要があります。DocumentBroker では，次の文字コードセットをサポートしています。

● Shift-JIS

● UTF-8（使用できる文字コードの範囲は UCS-2 または UCS-4 です）

このため，プログラムでは locale 機能を用いてロケールを設定する必要があります。

ユーザの言語要求に従ってロケールを設定するプログラミングの例を次に示します。

ロケールを設定するプログラミングの例

```
#include <locale.h>
:
setlocale(LC_ALL, "");
:
```

(1) 文書空間の文字コード種別の取得

接続する文書空間で使用している文字コード種別は，CdbrSession::GetDocSpaceCharacterSet メソッドで取得できます。1つのクライアントアプリケーションが文字コード種別の異なる複数の文書空間に接続する場合に，このメソッドを使用することで動的に文書空間の文字コードに合わせた処理を行うことができます。CdbrSession::GetDocSpaceCharacterSet メソッドの詳細については，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。

(2) 文字コード種別が UTF-8 の場合に使用できる機能

文書空間で使用する文字コード種別が UTF-8 の場合，DocumentBroker で使用できる機能や各クラスで提供するメソッドに制限があります。文書空間で使用する文字コード種別が UTF-8 の場合に使用できる範囲を次に示します。なお，文書空間で使用する文字コード種別が，Shift-JIS の場合は，すべての機能が使用できます。

表 5-3　文書空間で使用する文字コード種別が UTF-8 の場合に使用できる機能

分類	機能名	使用可否
文書管理	文書の登録機能	○
	バージョン管理機能	○
	マルチレンディション管理機能	○
	マルチファイル管理機能	○
	コンテナ管理機能	○
	文書の構成管理機能	○
	文書間リレーション管理機能	○

分類	機能名	使用可否
	文書の属性情報の管理機能	○
	File Link 連携機能	－
	リファレンスファイル管理機能	○
	レンディションのコンテント種別変換機能	○
	XML 文書管理機能	－
独立データの管理機能	独立データの管理機能	○
アクセス制御	アクセス制御機能	○
検索	属性検索	○
	全文検索	○
	構造指定検索	－
	概念検索	○
	文字列型プロパティに対する全文検索	○

（凡例）
○：使用できます。
－：使用できません。

なお，文書空間で使用する文字コード種別が UTF-8 の場合に使用できるメソッドの詳細は，「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。

5.2.2　ヘッダファイル

クライアントがインクルードしなければならないヘッダファイルについて説明します。

ヘッダファイルには，クラスライブラリのヘッダファイル，DMA の基本ヘッダファイルおよび DocumentBroker の拡張ヘッダファイルがあります。

インクルードするヘッダファイルを，OS ごとに説明します。

（1）ヘッダファイル（UNIX の場合）

DocumentBroker のアプリケーションプログラムをクラスライブラリを使用して作成する際には，クラスライブラリのヘッダファイルをインクルードしてください。

次に，クラスライブラリのヘッダファイルについて説明します。クラスライブラリのヘッダファイルは，次のディレクトリに格納されています。

```
/opt/HiEDMS/ACLibrary/include
/opt/HiEDMS/client/include
```

クラスライブラリのヘッダファイルを次の表に示します。

表 5-4　クラスライブラリのヘッダファイルの一覧

ヘッダファイル名	定義内容
DBR.h	クラスライブラリで使用するすべての定義
DBR_CfgRfct.h	CdbrConfiguratedReferentialContainer クラスの定義
DBR_Common.h	共通のマクロの定義

ヘッダファイル名	定義内容
DBR_CommonStruct.h	構造体についての共通の定義
DBR_Compound.h	CdbrCompound クラスの定義
DBR_Containable.h	CdbrContainable クラスの定義
DBR_Core.h	CdbrCore クラスの定義
DBR_Delete.h	構造体メモリの解放関数の定義
DBR_DMA.h	CdbrDMA クラスの定義
DBR_Document.h	CdbrDocument クラスの定義
DBR_ErrorCode.h	エラーコードの定義
DBR_EqlStatement.h	CdbrEqlStatement クラスの定義
DBR_IndependentPersistence.h	CdbrIndependentPersistence クラスの定義
DBR_PublicACL.h	CdbrPublicACL クラスの定義
DBR_QueryStruct.h	検索に使用する構造体の定義
DBR_Rfct.h	CdbrReferentialContainer クラスの定義
DBR_Session.h	CdbrSession クラスの定義
DBR_Tools.h	クラスライブラリが提供する関数の定義
DBR_VArray.h	CdbrVariableArray クラスの定義
DBR_Versionable.h	CdbrVersionable クラスの定義
DBR_Vrdoc.h	CdbrVersionableDocument クラスの定義
DBR_Vtct.h	CdbrVersionTraceableContainer クラスの定義
DBR_XmlTranslator.h	CdbrXmlTranslator クラスの定義
DBR_XmlTranslatorFactory.h	CdbrXmlTranslatorFactory クラスの定義

DBR.h は，クラスライブラリのメインヘッダファイルです。このヘッダファイルをインクルードすると，すべてのヘッダファイルをインクルードしたことになります。なお，DBR.h をインクルードする場合は，コンパイル時のインクルードパスに，次のディレクトリを追加してください。

```
/opt/TPBroker/include
```

ただし，TPBroker V5 と連携して動作する環境では，ディレクトリを追加する必要はありません。

(2) ヘッダファイル（Windows の場合）

DocumentBroker のアプリケーションプログラムをクラスライブラリを使用して作成する際には，次のヘッダファイルをインクルードします。

- stdafx.h または windows.h
- クラスライブラリのヘッダファイル

まず，stdafx.h または windows.h をインクルードしてから，クラスライブラリのヘッダファイルをインクルードしてください。

stdafx.h と windows.h は，MFC を使用するかどうかで，次のように選択してください。

MFC を使用する場合

```
stdafx.h
```

それ以外の場合

```
windows.h
```

クラスライブラリのヘッダファイルは，次のディレクトリに格納されています。

＜インストールディレクトリ＞¥include

クラスライブラリのヘッダファイルについては，「(1) ヘッダファイル（UNIX の場合）」を参照してください。なお，クラスライブラリのヘッダファイル DBR.h をインクルードする場合は，コンパイル時のインクルードパスに，次のディレクトリを追加してください。

<TPBrokerのインクルードディレクトリ>¥include

ただし，TPBroker V5 と連携して動作する環境では，ディレクトリを追加する必要はありません。

5.2.3 コンパイルオプション

指定するコンパイルオプションについて説明します。

（1） コンパイルオプション（AIX の場合）

IBM VisualAge C++ Professional for AIX V5 でコンパイルする場合，次のオプションを追加してください。

```
-qchars=signed
-D_ALL_SOURCE
-D_POSIX_SOURCE
-D_REENTRANT
-D_THREAD_SAFE
-DPTHREADS
-DTHREAD
-DAIXV3
-DUNIX
-DDMA_ASTR
-DDMA_NO_BSTR
```

IBM VisualAge C++ Professional for AIX V6 または IBM XL C/C++ Enterprise Edition for AIX でコンパイルする場合，次のオプションを追加してください。

```
-qchars=signed
-qnamemangling=v5
-D_ALL_SOURCE
-D_POSIX_SOURCE
-D_REENTRANT
-D_THREAD_SAFE
-DPTHREADS
-DTHREAD
-DAIXV3
-DUNIX
-DDMA_ASTR
-DDMA_NO_BSTR
```

TPBroker V5 と連携して動作する環境で，コンパイルするときは，上記コンパイルオプションに加えて，以下を設定してください。

```
-DDBR_VB_V5
```

注意事項

TPBroker V3 と連携して動作する環境で作成したアプリケーションを TPBroker V5 と連携して動作

する環境で実行する場合，-DDBR_VB_V5 を追加して，再度コンパイルしてください。

(2) コンパイルオプション（Windows の場合）

Visual C++6.0 でコンパイルする場合，次のコンパイルオプションを指定してください。

/MD
ランタイムライブラリの利用［マルチスレッド DLL］

/Gd
呼び出し規則［_cdecl］

/GX
例外処理の有効化

/D
次のシンボルを設定してください。
`WIN32，WINNT，DMA_ASTR，DMA_NO_BSTR`

TPBroker V5 と連携して動作する環境で，コンパイルするときは，上記シンボルに加えて，以下を設定してください。
`DBR_VB_V5`

注意事項
TPBroker V3 と連携して動作する環境で作成したアプリケーションを TPBroker V5 と連携して動作する環境で実行する場合，DBR_VB_V5 を追加して，再度コンパイルしてください。

5.2.4 ライブラリの指定

ライブラリの指定を，OS ごとに説明します。

(1) ライブラリの指定（AIX の場合）

クライアントアプリケーションを作成する際には，次のライブラリを指定してください。

- libdsacl.a
- libdsxml.a（XML 文書管理機能を使用する場合※）

クライアントアプリケーションを作成する場合は，次のリンケージディレクトリを指定してください。

● TPBroker V3 を使用している場合

- /opt/hitachi/common/lib
- /opt/HiEDMS/client/lib
- /opt/HiEDMS/ACLibrary/lib
- /opt/TPBroker/lib
- /opt/hirdb_xml/lib（XML 文書管理機能を使用する場合※）
- /opt/hitachi/xpk/lib（XML 文書管理機能を使用する場合※）

● TPBroker V5 を使用している場合

- /opt/hitachi/common/lib
- /opt/HiEDMS/client/lib_tp5
- /opt/HiEDMS/ACLibrary/lib_tp5
- TPBroker V5 インストールディレクトリ /lib

- /opt/hirdb_xml/lib（XML 文書管理機能を使用する場合※）
- /opt/hitachi/xpk/lib（XML 文書管理機能を使用する場合※）

注※

XML 文書管理機能（XML プロパティマッピング機能・XML インデクスデータ作成機能）を使用する場合，クライアントアプリケーションの開発環境および実行環境に，HiRDB Adapter for XML が必要です。また，XML インデクスデータ作成機能を使用する場合，実行環境に Preprocessing Library for Text Search が必要です。Preprocessing Library for Text Search がない場合，CdbrXmlTranslator::GetIndexData メソッドおよび CdbrXmlTranslator::GetDmaInfoList メソッド（全文検索インデクスデータ出力を有効にしたとき）をコールしたとき，ERR_DMA, DMARC_NOT_SUPPORTED のエラーになります。

また，次のリンケージオプションを指定してください。

● +s

(2) ライブラリの指定（Windows の場合）

クライアントアプリケーションを作成する際には，次のライブラリを指定してください。

● マルチスレッド対応のクライアントアプリケーションを作成する場合

- <インストールディレクトリ >¥lib¥edmdsacl.lib
- <インストールディレクトリ >¥lib¥edmdsxml.lib（XML 文書管理機能を使用する場合※）

● シングルスレッド対応のクライアントアプリケーションを作成する場合

- <インストールディレクトリ >¥lib¥edmdsacl_s.lib
- <インストールディレクトリ >¥lib¥edmdsxml_s.lib（XML 文書管理機能を使用する場合※）

シングルスレッド対応のクライアントアプリケーションを作成する場合，TPBroker のライブラリをリンクする際には "_r" が付いていないものを指定してください。なお，TPBroker V5 と連携して動作する環境では，シングルスレッド対応のクライアントアプリケーションを作成することはできません。

注※

XML 文書管理機能（XML プロパティマッピング機能・XML インデクスデータ作成機能）を使用する場合，クライアントアプリケーションの作成環境および実行環境に，HiRDB Adapter for XML が必要です。また，XML インデクスデータ作成機能を使用する場合，実行環境に Preprocessing Library for Text Search が必要です。

5.2.5 ほかのプログラムとの併用の際の注意

TPBroker のヘッダファイル（corba.h）を，DocumentBroker のヘッダファイルよりも先にインクルードしてください。

5.2.6 クラス識別子，プロパティ識別子の定義

ここでは，クラス識別子およびプロパティ識別子の GUID 値の実体を定義するために必要な定義について説明します。

DMA が規定しているクラス識別子およびプロパティ識別子は，次のヘッダファイルに定義されています。

UNIX の場合

```
/opt/HiEDMS/client/include/dmaids.h
/opt/HiEDMS/client/include/dmaidvar.h
/opt/HiEDMS/client/include/edmids.h
/opt/HiEDMS/client/include/edmidvar.h
```

Windows の場合

```
＜インストールディレクトリ＞¥dmaids.h
＜インストールディレクトリ＞¥dmaidvar.h
＜インストールディレクトリ＞¥edmids.h
＜インストールディレクトリ＞¥edmidvar.h
```

これらのヘッダファイルは，DBR.h をインクルードすることで間接的にインクルードされるため，明示的にインクルードする必要はありません。ただし，ユーザが作成するアプリケーションプログラムを構成するソースファイルの中の一つに，DMA_INIT_ID および EDM_INIT_ID の定義が必要です。

DMA_INIT_ID および EDM_INIT_ID の定義は，クラス識別子およびプロパティ識別子の GUID 値の実体を定義するために必要な定義です。必ず一つのソースファイルに定義してください。これらを定義しなかったり，二つ以上のソースファイルに定義した場合，リンケージエラーになります。

また，この定義は，必ず DBR.h のインクルードよりも前に定義してください。DBR.h のインクルードよりもあとに定義している場合，DMA_INIT_ID および EDM_INIT_ID の定義は無効になります。

定義例を次に示します。

```
#define DMA_INIT_ID
#define EDM_INIT_ID

#include "DBR.h"
      :
```

なお，Windows の場合に，stdafx.h や windows.h をインクルードするときは，まず，これらのヘッダファイルをインクルードしてから，DMA_INIT_ID および EDM_INIT_ID を定義してください。この場合も，DBR.h のインクルードは，定義のあとにしてください。

定義例を次に示します。

```
#include "stdafx.h"

#define DMA_INIT_ID
#define EDM_INIT_ID

#include "DBR.h"
    :
```

5.2.7 ユーザ定義のクラス識別子・プロパティ識別子の定義

ここでは，ヘッダファイルを作成して GUID 値の実体を定義する方法について説明します。

なお，EDMCreateIds コマンドでも，GUID 値の実体を定義できます。EDMCreateIds コマンドの詳細については，マニュアル「DocumentBroker Version 3　システム導入・運用ガイド」を参照してください。また，ソースファイルが一つしかない場合や，ユーザが追加定義したサブクラスやプロパティが少ない場合は，ソースファイル内で define 文を使用して GUID 値を直接定義した方が簡単に定義できることもあります。適切な方法で定義してください。

この例では，次の条件でヘッダファイルを作成します。なお，サブクラスおよびプロパティの追加方法については，マニュアル「DocumentBroker Version 3　システム導入・運用ガイド」を参照してください。

条件

- 追加したサブクラスのクラス識別子は，usrClass_Document である。
- 追加したプロパティのプロパティ識別子は，usrProp_Name および usrProp_Author である。
- 追加したサブクラス，プロパティの GUID 値は，次のとおりである。

表 5-5　追加したサブクラス，プロパティの GUID 値

クラス識別子またはプロパティ識別子	GUID 値
usrClass_Document（サブクラス）	F5D8E3A2-284C-11d2-9177-0000E2130367
usrProp_Name（プロパティ）	F5D8E3A4-284C-11d4-9177-0000E2130367
usrProp_Author（プロパティ）	76fa162c-0406-11d2-b29a-0060b0ea4840

- 作成する定義ファイルの名称は，usrids.h および usrvar.h とする。
  なお，作成する定義ファイル名は任意です。

手順を示します。

1. **追加したサブクラスおよびプロパティの GUID 値を確認します。**
   GUID 値は，システム管理者に確認します。

   システム管理者による GUID 値の確認方法

   GUID 値は，EDMCrtSimMeta コマンドを実行することによって確認できます。設定した GUID 値をシステム管理者が忘れてしまった場合や，システムで GUID 値が自動的に割り振られる設定にしてある場合には，システム管理者が EDMCrtSimMeta コマンドを実行して GUID 値を確認してください。

   この例では，表 5-5 に示したように GUID 値は設定されていることとします。

2. **usrids.h を作成します。**
   このヘッダファイルには，GUID 値の実体を定義します。
   定義例を次に示します。

```
#define usrClass_DocumentVal¥
{ 0xf5d8e3a2,0x284c,0x11d2,¥
{ 0x91,0x77,0x0,0x0,0xe2,0x13,0x3,0x67 }}

#define usrProp_NameVal¥
{ 0xf5d8e3a4,0x284c,0x11d4,¥
{ 0x91,0x77,0x0,0x0,0xe2,0x13,0x3,0x67 }}

#define usrProp_AuthorVal¥
{ 0x76fa162c,0x0406,0x11d2,¥
{ 0xb2,0x9a,0x0,0x60,0xb0,0xea,0x48,0x40 }}
```

3. **usridvar.h を作成します。**
   このヘッダファイルには，USR_INIT_ID を定義してあるソースファイルに GUID 値の実体を定義し，それ以外のソースファイルには extern を宣言することを定義します。
   定義例を次に示します。

```
#include "usrids.h"

#if defined(USR_INIT_ID)
#   define USR_DECL_ID(x_) DMA_EXTERN_C DmaId x_=x_##Val;
#else
#   define USR_DECL_ID(x_) DMA_EXTERN_C DmaId x_;
#endif

USR_DECL_ID(usrClass_Document)
USR_DECL_ID(usrProp_Name)
```

```
USR_DECL_ID(usrProp_Author)
```

4. **ソースファイルに，**usidvar.h **をインクルードします。**

一つのソースファイルだけに，USR_INIT_ID を定義して，usridvar.h をインクルードします。これによって，そのソースファイルにユーザ定義のクラス識別子およびプロパティ識別子の GUID 値の実体が定義されます。USR_INIT_ID の定義は，DMA_INIT_ID および EDM_INIT_ID を定義したソースファイルと同じファイルにすることをお勧めします。なお，二つ以上のファイルに USR_INIT_ID を指定しないでください。
定義例を次に示します。

```
#define DMA_INIT_ID
#define EDM_INIT_ID
#define USR_INIT_ID

#include "DBR.h"
#include "usridvar.h"
    :
```

5.3 ファイル転送機能の使用

DocumentBroker サーバに接続するクライアントが異なるマシン上に存在する場合，サーバとクライアント間のファイル転送には，ファイル転送機能を使用する必要があります。例えば，DocumentBroker で管理している文書のファイルを異なるクライアントマシンに取得したり，異なるクライアントマシンに存在するファイルを DocumentBroker の文書として登録したりする場合に，ファイル転送が発生します。

ファイル転送機能は，ファイル転送サービスが提供しています。ファイル転送サービスを使用するための環境設定については，マニュアル「DocumentBroker Version 3　システム導入・運用ガイド」のファイル転送機能の使用についての説明を参照してください。

6 エラー処理

この章では，エラー処理について説明します。

6.1 エラーの種類

DocumentBroker で発生するエラーの種類について説明します。

6.1.1 major_code によるエラーの分類

ここでは，major_code によってエラーを分類する方法について説明します。エラーは，CdbrCore::GetLastError メソッドによって取得できる major_code によって分類できます。

major_code によって分類できるエラーには，次の 2 種類があります。

(1) データベースで発生したエラー（major_code が ERR_DB の場合）

データベースでエラーが発生した場合，major_code に「ERR_DB」が設定されます。

この場合は，CdbrCore::GetDBError メソッドをコールして，より詳細なエラー情報を取得してください。

(2) データベース以外で発生したエラー（major_code が ERR_DB 以外の場合）

エラーが，データベース以外で発生した場合は，major_code に「ERR_DBR」,「ERR_DMA」または「ERR_CORBA」が設定されます。

この場合は，CdbrCore::GetLastError メソッドによって取得した major_code および minor_code を基に，エラーの原因に対処してください。また，「ERR_CORBA」の場合は，ppErrorString に出力された情報を参照してください。

6.1.2 詳細な情報が取得できるエラー

DocumentBroker で発生するエラーには，さらに詳細な情報について示す詳細メッセージを取得できるものがあります。詳細メッセージの取得方法については，「6.3　詳細メッセージの取得」を参照してください。取得した詳細メッセージに対する対処については，マニュアル「DocumentBroker Version 3　メッセージ」を参照してください。

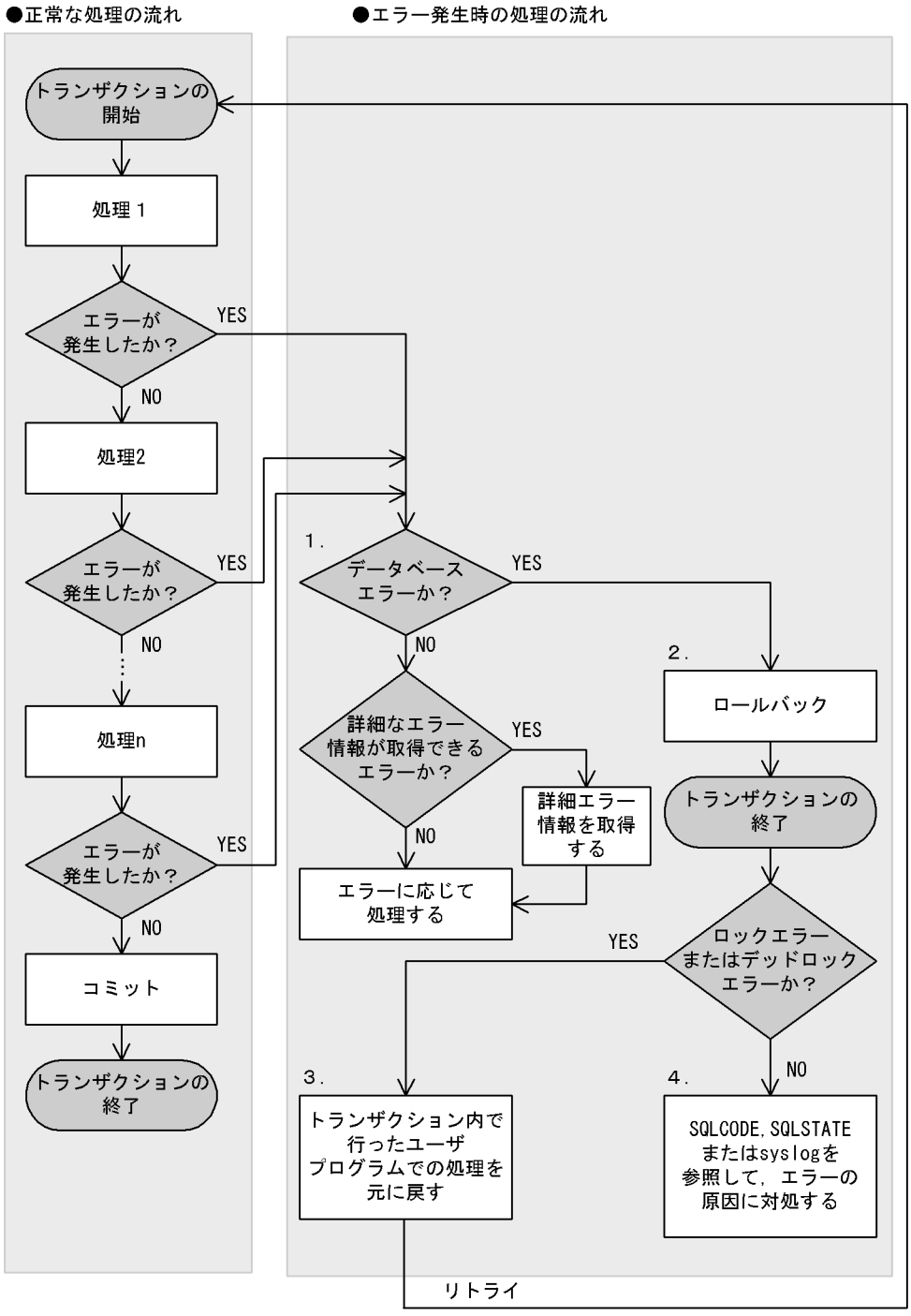
6.1.3 エラー確認後の処理

エラーの種類と情報の内容を確認したら，明示的に，CdbrSession::Rollback メソッドをコールして，ロールバックを実行してください。

6.2 エラー処理の流れ

ここでは，エラーが発生した場合の処理の流れについて説明します。次の図に，エラーが発生した場合の処理の流れを示します。

図 6-1 エラーが発生した場合の処理



エラーが発生した場合の処理手順について，図 6-1 の流れに沿って説明します。

1. エラーが発生したら，GetLastError メソッドによって，エラーの種類を判断します。

major_code が「ERR_DB」の場合はデータベースでエラーが発生していますので，操作 2 以降に従って，対処します。
それ以外の場合は，major_code および minor_code の内容に従って，エラーに対処します。
詳細なメッセージを取得する場合は，CdbrSession::GetLastDetailError メソッドをコールして取得してください。

2. データベースのエラーの場合（major_code が「ERR_DB」の場合）は，トランザクションをロールバックして終了させます。
   minor_code によって，操作 3 または操作 4 に進みます。
3. minor_code が「ERR_DB_LOCKED（ロックエラー）」または「ERR_DB_DEADLOCK_OCCURED（デッドロックエラー）」の場合は，リトライします。
   minor_code が「ERR_DB_LOCKED（ロックエラー）」または「ERR_DB_DEADLOCK_OCCURED（デッドロックエラー）」の場合は，ロックエラーまたはデッドロックエラーが発生しています。このエラーの場合は，リトライできます。トランザクション内にユーザプログラムで実行した処理を元に戻してから，リトライします。
4. minor_code が「ERR_DB_FAILED」の場合は，CdbrCore::GetDBError メソッドをコールします。
   データベースが起動していない場合などに，このエラーは発生します。
   出力された SQLCODE および SQLSTATE を参照して，原因を確認，対処します。
   また，GetDBError メソッドで取得する詳細なエラー情報は，システムのメッセージとしても出力されます。必要に応じて参照してください。
   なお，DocumentBroker のデフォルトの動作として，ロックを取得時にほかのトランザクションでロックが取得されている場合，そのロックが解放されるのをウエイトします。この場合は，タイムアウトしたときにだけ，ロックエラーが出力されますので，必要に応じてリトライしてください。
   なお，ロック取得時のウエイトは，CdbrDMA::ConnectObject メソッドの引数の設定によって，ウエイトしないように選択することもできます。

6.3 詳細メッセージの取得

アプリケーション開発でのデバッグ時および実行時のエラーに具体的に対処できるように，DocumentBroker では，詳細なエラー情報（詳細メッセージ）を提供しています。

この節では，詳細メッセージの出力先およびアクセス制御機能でエラーになったときに詳細メッセージが取得できる戻り値について説明します。

なお，詳細メッセージの内容については，マニュアル「DocumentBroker Version 3　メッセージ」を参照してください。

6.3.1 詳細メッセージの出力先

ここでは，詳細メッセージの出力先について説明します。

詳細メッセージは，詳細エラーログファイルまたは CdbrSession::GetLastDetailError メソッドの引数に出力されます。

(1) 詳細エラーログファイル

クライアント環境変数 DBR_DETAIL_ERRORLOG が ON のとき，DBR_DETAIL_ERRORLOG_DIR で指定されたディレクトリの下に「EDMErrTraceCLXXXXX_1.log」として出力されます。XXXX はプロセス ID を示します。詳細エラーログファイルには，エラーが発生しない場合もログが出力されますので，必要に応じて詳細エラーログファイルを削除するようにしてください。

クライアント環境変数については，「5.1.3　実行環境の設定」を参照してください。

なお，文書空間で使用する文字コード種別が UTF-8 の場合，メッセージテキストの %n に UTF-8 の文字列が出力されることがあります。UTF-8 の文字列の内容を確認するには，UTF-8 に対応したエディタなどで参照してください。

(2) CdbrSession::GetLastDetailError メソッドの引数

詳細メッセージは，それを取得するためのメソッドである CdbrSession::GetLastDetailError メソッドをコールした場合に，引数の値として出力されます。

このメソッドは，クラスライブラリのメソッドのコール時にエラーが発生した場合に，CdbrCore::GetLastError メソッドで major_code，minor_code を取得したあとで必要に応じてコールします。メソッドの詳細については，「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」の CdbrSession::GetLastDetailError メソッドを参照してください。

また，詳細メッセージを取得するためのコーディング例については，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。

6.3.2 アクセス制御機能でエラーが発生した場合に詳細メッセージが取得できる戻り値

ここでは，アクセス制御機能で発生するエラーのうち，詳細メッセージが取得できるエラーを表す戻り値について説明します。

次の戻り値が出力された場合は，詳細メッセージが取得できます。

表 6-1 詳細メッセージが取得できるアクセス制御機能の戻り値

major_code	minor_code
ERR_DBR	ERR_ACCESS_NOT_PERMITTED
ERR_DBR	ERR_BAD_PERMISSION
ERR_DBR	ERR_INVALID_PUBLICACL_COUNT
ERR_DBR	ERR_INVALID_ACE_COUNT
ERR_DBR	ERR_NONEXIST_PUBLICACL
ERR_DBR	ERR_PUBLICACL_NOT_BOUND

付録

付録 A　DMA オブジェクトの概要

クラスライブラリのオブジェクトは，DMA が提唱するオブジェクトモデルを基に作成された DMA オブジェクトを構成要素として作成されています。

ここでは，DMA オブジェクトモデルの概要として，次の項目について説明します。

- DMA オブジェクトの操作に使用する概念
- DocumentBroker で使用する DMA のクラス
- DMA オブジェクトで表す文書
- 文書のバージョン管理
- コンテインメントを利用した文書管理
- バージョン付き構成管理コンテナを利用した構成管理

なお，これらの説明内のオブジェクトは，すべて DMA オブジェクトです。

付録 A.1　DMA オブジェクトの操作に使用する概念

ここでは，オブジェクトを操作するために使用する次の概念について説明します。

● クラス

● プロパティ

● メソッド

● GUID と OIID

(1) クラス

クラスとは，オブジェクトの型を定義したひな形です。すべての DMA オブジェクトは，DMA クラスの一つを基にして作成します。あるクラスを基にして作成されたオブジェクトは，そのクラスのインスタンスです。したがって，同じクラスを基に作成されたオブジェクトは，すべて同じ型を持っています。

DocumentBroker では，用途に応じた DMA クラスを用意しています。クライアントアプリケーションは，これらから必要に応じた DMA クラスを選択してオブジェクトを作成する必要があります。

例えば，文書を表現するには，dmaClass_DocVersion クラスという DMA クラスを使用します。dmaClass_DocVersion クラスに基づいて作成する（インスタンス）オブジェクトは，DocVersion オブジェクトです。

(2) プロパティ

クラスには，プロパティが定義されています。**プロパティ**とは，オブジェクトの状態を表す属性です。

同じクラスから作成されたオブジェクトは，同じプロパティを持っています。DocumentBroker では，プロパティの値を取得したり，プロパティに値を設定したりすることによって，文書を検索したり更新したりします。

(3) メソッド

メソッドとは，クライアントアプリケーションが使用するコマンドで，オブジェクトの状態を参照したり更新したりするための手段として使用します。つまりメソッドは，プロパティに値を設定したり，プロパティから値を取得したりするために使用します。

(4) GUID と OIID

GUID は，クラスおよび検索に使用するオペレータ（問い合わせ演算子）を識別するために付けられる識別子です。メソッドで使用するデータ型である DmaId は，GUID です。

また，すべての DMA オブジェクトには，プロパティとして OIID が付けられています。OIID は，System オブジェクトの GUID，DocSpace オブジェクトの GUID，文書空間特有のオブジェクトの識別子から構成され，URL の形式で定義されています。この OIID を基に，DMA オブジェクトを探索できます。

OIID に使用する URL の文法については，マニュアル「DocumentBroker Version 3　クラスライブラリ　C++　リファレンス　基本機能編」を参照してください。

付録 A.2　DocumentBroker で使用する DMA クラス

ここでは，DMA クラスおよびプロパティについて説明します。

(1) クラス

ここでは，DMA クラスについて説明します。

(a) クラスの識別子

DMA クラスは，次の識別子を付与して管理しています。

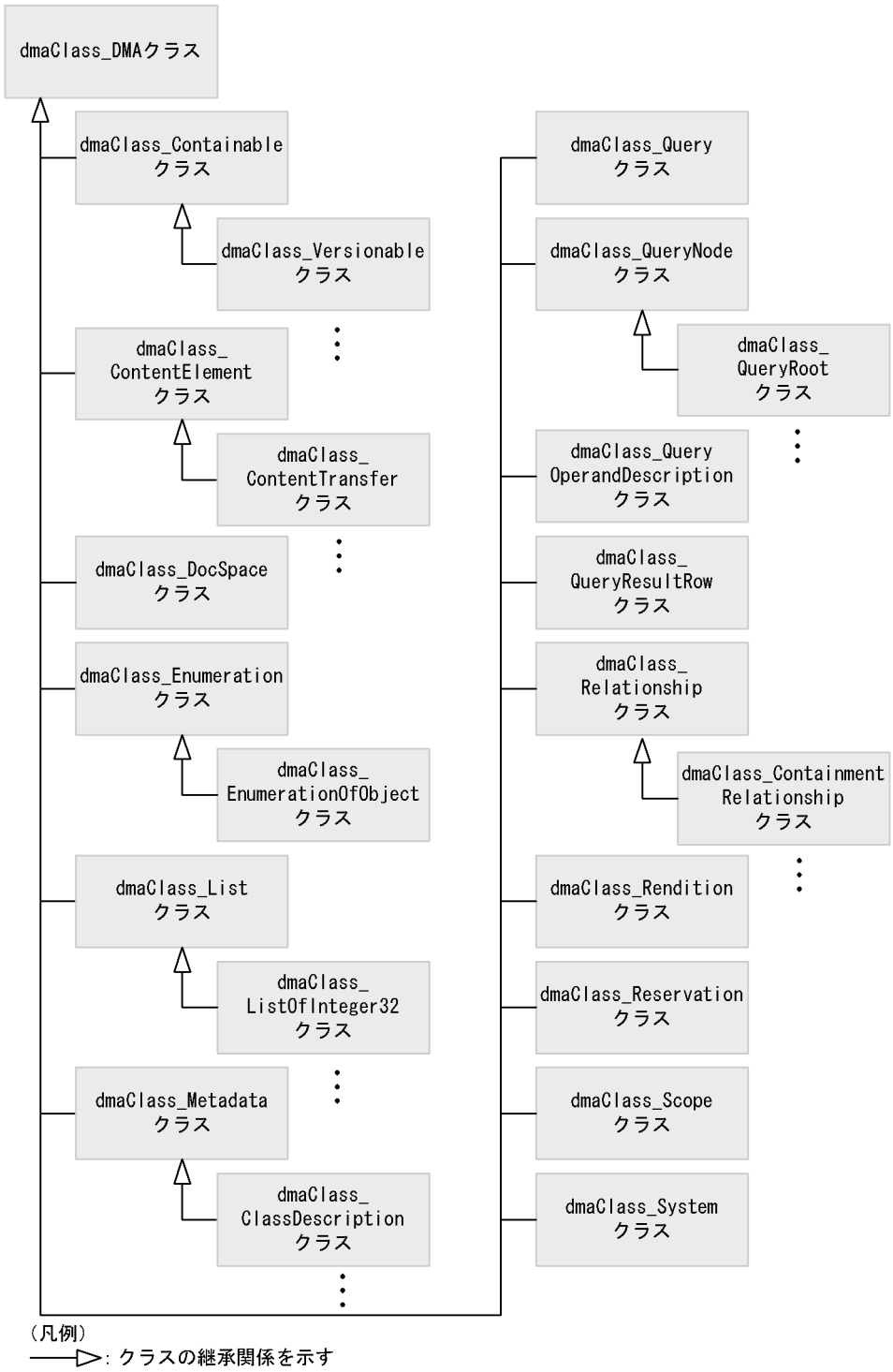
- **初期識別子**
  DocumentBroker によって割り当てられる GUID。
- **追加識別子**
  GUID の別名の識別子。追加識別子は，クライアントアプリケーションから任意の GUID の値で設定できます。ただし，クラスとプロパティに同じ GUID は設定できません。また，クラスと検索に使用するオペレータに同じ GUID を設定できません。さらに，DocumentBroker がすでに使用している GUID は重複して使用できません。

(b) クラスの継承関係

クラスは，その直上にあるクラスが持つプロパティを選択して継承します。

DMA クラスは，dmaClass_DMA クラスというクラスを基にしたツリー構造をしています。主な DMA クラスのツリー構造を，次の図に示します。

図 A-1　DMA クラスの階層構造



DocumentBroker が用意している DMA クラスは，dmaClass_DMA クラスが持つプロパティを受け継ぎ，各クラスで固有のプロパティが付加されたり，必要に応じて削除されたりしています。

また，DMA クラスのうち，下記のクラスについては，直下にサブクラスを定義できます。

● dmaClass_ConfigurationHistory クラス

● dmaClass_Container クラス

● edmClass_VersionTraceableContainer クラス

● edmClass_ContainerVersion クラス

● dmaClass_DocVersion クラス

● edmClass_ComponentDocVersion クラス

● edmClass_VersionTracedComponentDocVersion クラス

● edmClass_VersionTracedDocVersion クラス

● edmClass_IndependentPersistence クラス

サブクラスは，システム管理者が定義します。

(c) クラスのメタデータ

各 DMA クラスに関する詳細情報は，プロパティの一つとして保持されています。さらに，このプロパティの具体的な内容は，別のオブジェクトとして形成されています。このオブジェクトを，**メタデータオブジェクト**といいます。DMA クラスの詳細情報を記載したメタデータオブジェクトは，ClassDescription オブジェクトです。ClassDescription オブジェクト自身も，dmaClass_ClassDescription クラスを基に作成されるオブジェクトです。

ClassDescription オブジェクトに記載されている情報の一部を示します。

● DMA クラスの識別子の定義

● 直接のスーパークラスに関する定義

● サブクラスに関する定義

● DMA クラスのプロパティに関する定義

● 検索時に使用できるプロパティに関する定義

次の図に ClassDescription オブジェクトの例を示します。

図 A-2　ClassDescription オブジェクトの例

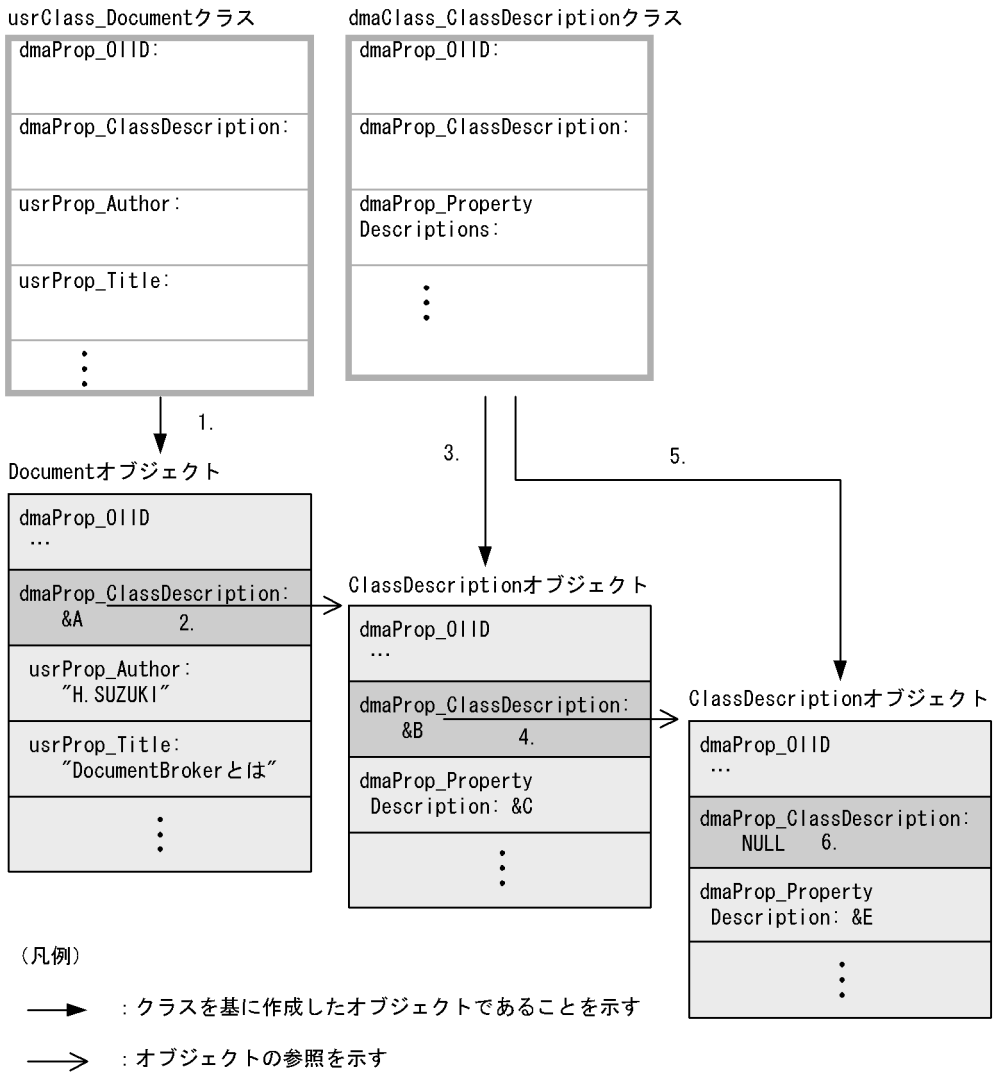


図 A-2 について説明します。

1. ここでは，usrClass_Document クラスという，dmaClass_DocVersion クラスのサブクラスを仮定します。この usrClass_Document クラスを基に Document オブジェクトを作成します。
2. Document オブジェクトの dmaProp_ClassDescription プロパティには，ClassDescription オブジェクトを参照するリファレンスが格納されています。リファレンスとは，独立なオブジェクトへの関連をたどる機能を提供して，Object 型のプロパティとして表現します。
3. ClassDescription オブジェクトには，usrClass_Document クラスの詳細情報が記載されています。ClassDescription オブジェクトは，dmaClass_ClassDescription クラスを基に作成されたオブジェクトです。
4. ClassDescription オブジェクトにも dmaProp_ClassDescription プロパティがあります。dmaProp_ClassDescription プロパティには，ClassDescription オブジェクトを参照するリファレンスが格納されています。
5. ClassDescription オブジェクトには，dmaClass_ClassDescription クラスの詳細情報が記載されています。この ClassDescription オブジェクトも，dmaClass_ClassDescription クラスを基に作成されています。

6. この ClassDescription オブジェクトにも dmaProp_ClassDescription プロパティがあります。しかし，これ以上参照先がないので，このプロパティには NULL 値が設定されます。

(d) メタデータ空間

メタデータ空間とは

各 DMA クラスを，継承関係に基づいて管理する空間を**メタデータ空間**といいます。したがって，メタデータ空間は，DMA クラスの集合です。

各 DMA クラスの定義情報は，そこに含まれる DMA クラスの ClassDescription オブジェクトを探索することによって得られます。

メタデータ空間には，そこに含まれる DMA クラスによって次の種類があります。

- System オブジェクトのメタデータ空間
- DocSpace オブジェクト（文書空間）のメタデータ空間
- Scope オブジェクトのメタデータ空間
- 問い合わせ結果のメタデータ空間

これらのメタデータ空間には，含まれる DMA クラスの定義が決められています。

なお，どのメタデータ空間にも，DMA クラスの継承の基になる dmaClass_DMA クラスが含まれます。

● System オブジェクトのメタデータ空間

dmaClass_System を含むメタデータ空間です。dmaClass_System クラスの定義情報は，System オブジェクトの ClassDescription オブジェクトを探索することによって得られます。

● DocSpace オブジェクトのメタデータ空間

dmaClass_DocSpace クラスおよび文書空間に含まれるすべての DMA クラスを含むメタデータ空間です。dmaClass_DocSpace クラスの定義情報は，DocSpace オブジェクトの ClassDescription オブジェクトを探索することによって得られます。また，文書空間に含まれるすべての DMA クラスの定義情報は，DocSpace オブジェクトの dmaProp_ClassDescriptions プロパティから探索することによって得られます。

● Scope オブジェクトのメタデータ空間

dmaClass_Scope クラスおよび検索に使用するすべての DMA クラスを含むメタデータ空間です。

dmaClass_Scope クラスの定義情報は，Scope オブジェクトの ClassDescription オブジェクトを探索することによって得られます。また，検索に使用する DMA クラスの定義情報は，Scope オブジェクトの次のプロパティから探索することによって得られます。

- dmaProp_SearchableClassDescriptions プロパティ
- dmaProp_QueryConstructionClassDescriptions プロパティ
- dmaProp_Operators プロパティ

● 問い合わせ結果のメタデータ空間

dmaClass_QueryResultSet クラスおよび dmaClass_QueryResultRow クラスを含むメタデータ空間です。dmaClass_QueryResultSet クラスおよび dmaClass_QueryResultRow クラスの定義情報は，QueryResultSet オブジェクトおよび QueryResultRow オブジェクトの dmaProp_ClassDescription プロパティを探索することによって得られます。

(2) プロパティ

ここでは，DMA クラスのプロパティの詳細について説明します。

(a) プロパティの識別子

プロパティには，クラスと同様に GUID が付けられて管理されます。クラスから継承されるプロパティの識別子は，サブクラスとスーパークラスで同じです。ただし，プロパティのデータ型と意味が同じであれば，直接的な継承関係を持たないクラスで同じプロパティ識別子が付けられます。

(b) プロパティの継承関係

DMA クラスのプロパティは，そのクラスのサブクラスに必要に応じて継承されます。

システム管理者がサブクラスを作成する場合，DocumentBroker によってスーパークラスに定義されているプロパティは，すべて継承する必要があります。

(c) プロパティのデータ型

ここでは，プロパティが持つデータ型について説明します。

● データ型の種類

プロパティは，値として次のデータ型を使用できます。

- Integer32 型
- String 型
- Boolean 型
- Id 型
- Object 型

● プロパティの基本単位

プロパティは，データ型に従った値を 1 個持つか複数個持つかが決められています。これを基本単位といいます。

基本単位には，次の種類があります。

- 値を 1 個持つプロパティ
  データ型に従った値を一つだけ持つプロパティです。この基本単位は Scalar 型です。
- 値を複数個持つプロパティ
  データ型に従った複数の値を一つの値として持つプロパティです。この基本単位は VariableArray 型です。
  VariableArray 型は，すべてのデータ型の値の一次元配列です。この一次元配列は可変長です。異なる種類のデータ型は混在できません。また DocumentBroker では，Object 型の値だけを配列要素にできます。したがって VariableArray 型のプロパティは，VariableArrayOfObject オブジェクトのようにデータ型に従った値を一次元配列として持つオブジェクトへのリファレンスを値として持ちます。
  実際の値は参照先の VariableArrayOfObject オブジェクトの一次元配列が保持します。一次元配列内の要素には順序性があり，ランダムにアクセスできます。また，要素の挿入，置換および削除ができます。
  VariableArray 型のプロパティを定義できるのは，次のサブクラスです。
  - dmaClass_Container クラスのサブクラス
  - dmaClass_DocVersion クラスのサブクラス
  - dmaClass_ConfigurationHistory クラスのサブクラス
  - edmClass_ContainerVersion クラスのサブクラス
  - edmClass_IndependentPersistence クラスのサブクラス

  VariableArray 型のプロパティから参照される VariableArrayOfObject オブジェクトの要素として格納できるのは，edmClass_Struct クラスのサブクラスのオブジェクトだけです。

(d) プロパティのメタデータ

プロパティに関する詳細情報は，データ型に従って dmaClass_PropertyDescription クラスにデータ型ごとにあるサブクラスのオブジェクトに記述されます。例えば，プロパティの値が String 型の場合は，PropertyDescriptionString オブジェクトにプロパティの詳細情報が記述され，プロパティの値が Integer32 型の場合は，PropertyDescriptionInteger32 オブジェクトにプロパティの詳細情報が記述されます。以降，dmaClass_PropertyDescription クラスのサブクラス（dmaClass_PropertyDescriptionString クラスや dmaClass_PropertyDescriptionInteger32 クラスなど）のオブジェクトを総称して PropertyDescription*DataType* オブジェクトと呼びます（*DataType* の部分は各データ型に対応します）。PropertyDescription*DataType* オブジェクトは，クラスの詳細情報を記述した ClassDescription オブジェクトの，dmaProp_PropertyDescriptions プロパティから参照されるメタデータオブジェクトです。PropertyDescription*DataType* オブジェクト自身も dmaClass_PropertyDescription*DataType* クラスを基に作成されるオブジェクトです。

PropertyDescription*DataType* オブジェクトに記載されている情報を一部示します。

- プロパティの識別子
- プロパティが持つデータ型
- プロパティの基本単位
- 検索条件として選択できるかどうかなどの情報

次の図に PropertyDescription*DataType* オブジェクトの例を示します。

図 A-3　PropertyDescription*DataType* オブジェクトの例

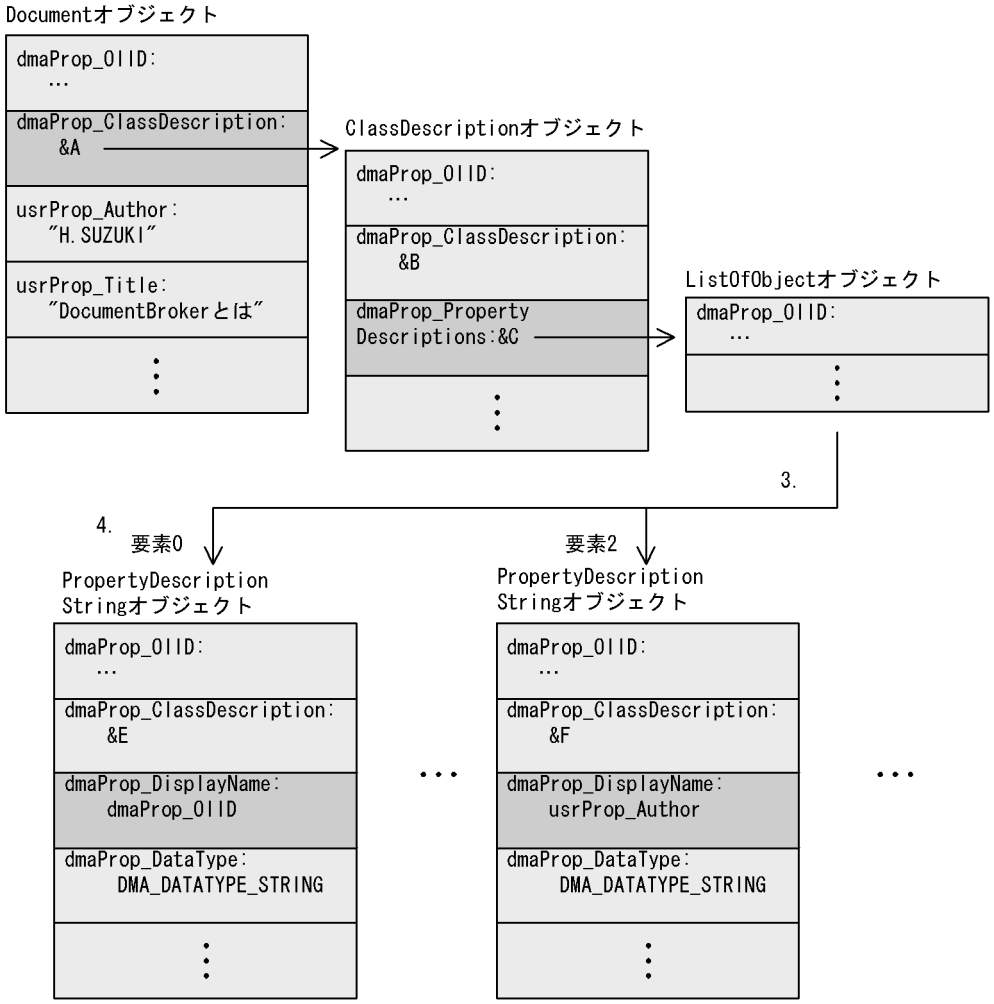


図 A-3 について説明します。

1. ここでは，Document オブジェクトを仮定して説明します。Document オブジェクトは，dmaClass_DocVersion クラスのサブクラス usrClass_Document クラスを基に作成されていると仮定します。Document オブジェクトの dmaProp_ClassDescription プロパティには，ClassDescription オブジェクトを参照するリファレンスが格納されています。
2. ClassDescription オブジェクトには，usrClass_Document クラスの詳細情報が記載されています。ClassDescription オブジェクトには，dmaProp_PropertyDescriptions プロパティがあります。dmaProp_PropertyDescriptions プロパティには，PropertyDescription*DataType* オブジェクトのリスト（ListOfObject オブジェクト）を参照するリファレンスが格納されています。
3. ListOfObject オブジェクトから PropertyDescription*DataType* オブジェクトの要素を取得します。
4. 要素として複数の PropertyDescription*DataType* オブジェクトがあります。ここでは，要素 0 の PropertyDescriptionString オブジェクトが，dmaProp_OIID プロパティの詳細情報を記述していることを示しています。同様に，要素 2 の PropertyDescriptionString オブジェクトが，usrProp_Author プロパティの詳細情報を記述していることを示しています。

付録 A.3　DMA オブジェクトで表す文書

ここでは，DMA オブジェクトで表す文書の概念について説明します。

(1) DocumentBroker での文書とは

DocumentBroker では，文書を DMA オブジェクトを使って表現します。例えば，「report.doc」という .doc ファイルで作成された「研究論文」という文書を仮定します。文書「研究論文」は，DMA オブジェクトを使うと次のように表現されます。

● 文書「研究論文」に相当する DMA オブジェクトである DocVersion オブジェクト
DocVersion オブジェクトは次のプロパティを保持し，文書の属性情報を管理します。

- dmaProp_Parent プロパティ
  この文書が格納されているフォルダを指し示すプロパティです。DocVersion オブジェクトで表現する文書は，0 個または 1 個のフォルダに格納できます。
- dmaProp_Containers プロパティ
  この文書に対してはり付けられている 0 個または複数個のインデクスが列挙されているプロパティです。
- dmaProp_VersionDescriptions プロパティ
  この文書がバージョン管理される場合，この文書が属している一連のバージョンを指し示す DMA オブジェクトが 0 個または複数個列挙されているプロパティです。
- dmaProp_Renditions プロパティ
  この文書が，どのような表現形式の文書なのか管理する DMA オブジェクトを指し示すプロパティです。

● 文書の種類である「Word 文書」を示す DMA オブジェクトである Rendition オブジェクト

● 文書を構成するコンテント「report.doc」を格納するための ContentTransfer オブジェクト

なお，DocumentBroker では，用途に応じて次に示す dmaClass_DocVersion クラスのサブクラスのオブジェクトを，文書を表す DMA オブジェクトとして DocVersion オブジェクトの代わりに使用できます。

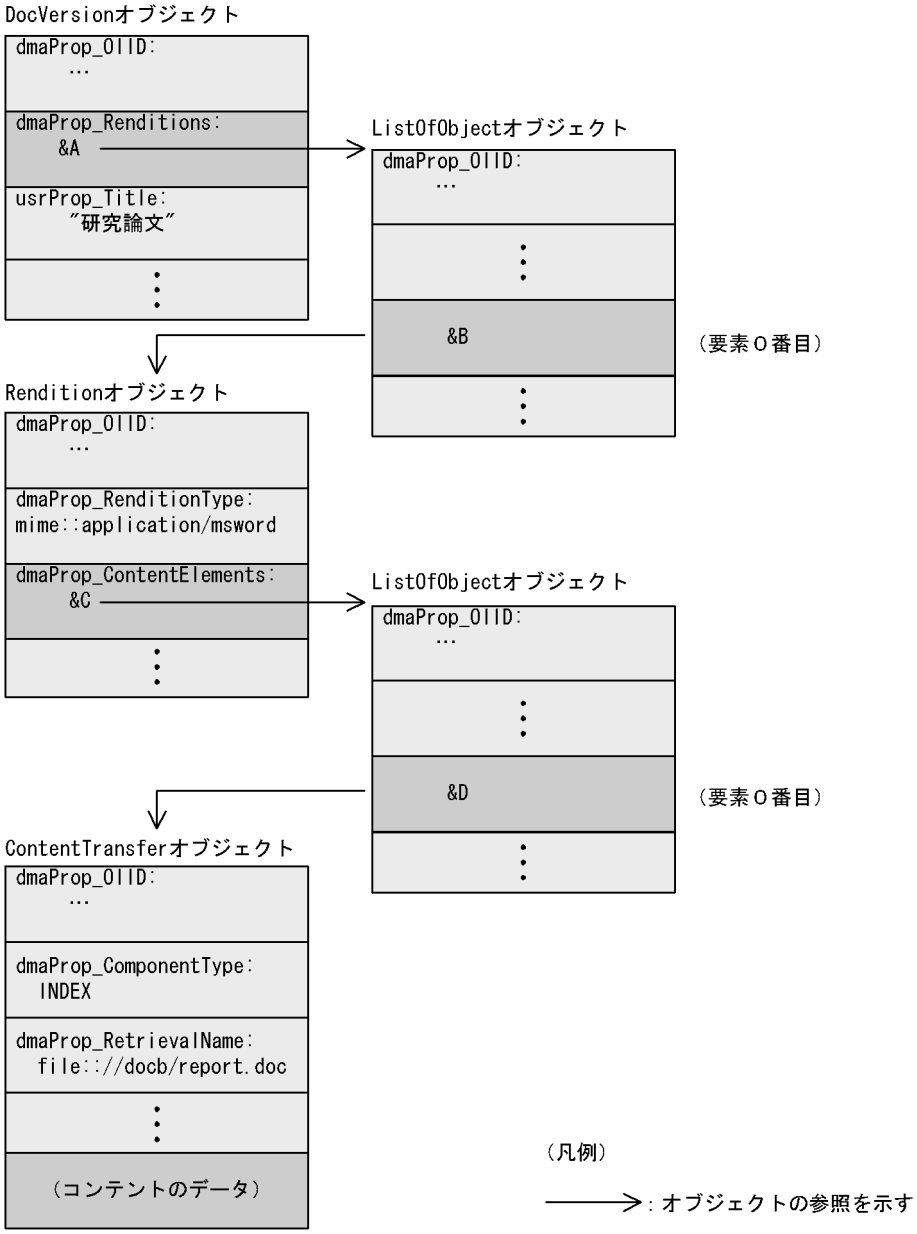
- VersionTracedDocVersion オブジェクト
- ComponentDocVersion オブジェクト
- VersionTracedComponentDocVersion オブジェクト

これらの DMA オブジェクトを使用すると，文書をバージョン管理したり，特定の一つのフォルダに格納して管理したり，文書に複数のインデクスをはり付けて管理したりするだけでなく，常に最新の文書を管理したり，バージョンを確定して文書を管理することができます。

ここでは，これらの DMA オブジェクトの基本となる DocVersion オブジェクトを「文書」として説明します。VersionTracedDocVersion オブジェクトについては「付録 A.5　コンテインメントを利用した文書管理」を参照してください。

文書「研究論文」を表現する DMA オブジェクトの関連を，次の図に示します。なお，dmaClass_DocVersion クラスのサブクラスとして，usrClass_Document クラスを作成していることを前提とします。

図 A-4　文書を表現する DMA オブジェクトの関連（例）



次に，文書を表現するための各 DMA オブジェクトについて説明します。

(a) DocVersion オブジェクト

文書を表現する DMA オブジェクトです。文書をバージョン管理する場合は，文書の特定のバージョンを表現するために使用します。文書のバージョン管理については，「付録 A.4　文書のバージョン管理」を参照してください。

文書の実体（コンテントデータ）は，DocVersion オブジェクトの下位にある Rendition オブジェクトおよびその下位にある ContentTransfer オブジェクトを経由して参照および更新できます。

(b) Rendition オブジェクト

文書を構成する文書の種類を管理する DMA オブジェクトです。Rendition オブジェクトは，

ContentTransfer オブジェクト一つを保持できます。

Rendition オブジェクトは，DocVersion オブジェクトにバインドして永続化する従属オブジェクトです。

(c) ContentTransfer オブジェクト

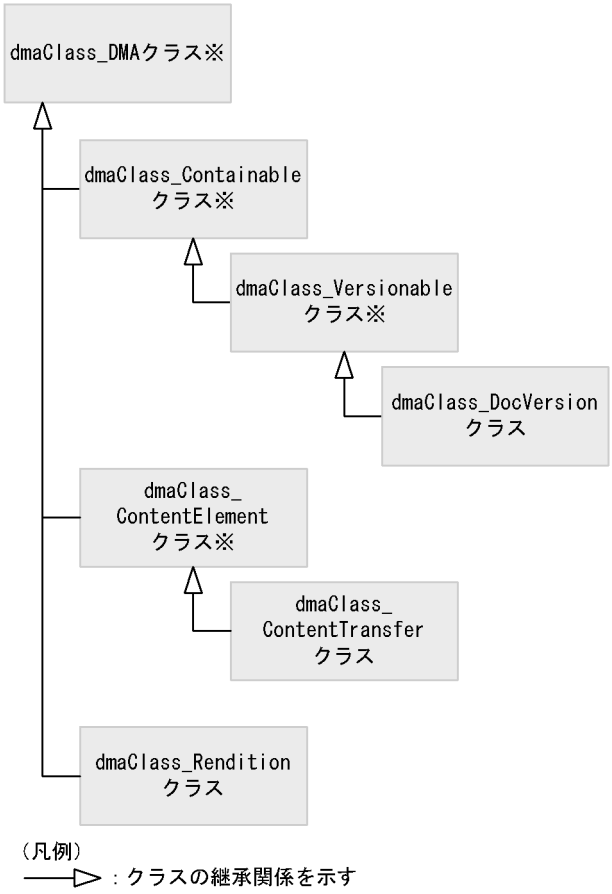
文書の実体であるコンテントデータを管理するための DMA オブジェクトです。一つのコンテントデータに対して，一つの ContentTransfer オブジェクトが存在します。ContentTransfer オブジェクトは，コンテントデータをデータベースに格納する場合に使用するオブジェクトです。

ContentTransfer オブジェクトは Rendition オブジェクトにバインドして永続化する従属オブジェクトです。

(2) 文書を表現するために使用する DMA クラス

文書を表現するために使用する DMA クラスを，継承関係に基づいて次の図に示します。

図 A-5　文書を表現するために使用するクラスの継承関係



注※　直接インスタンスを持たない抽象クラスです。

付録 A.4　文書のバージョン管理

ここでは，文書のバージョン管理について説明します。

(1) バージョンを利用した文書管理

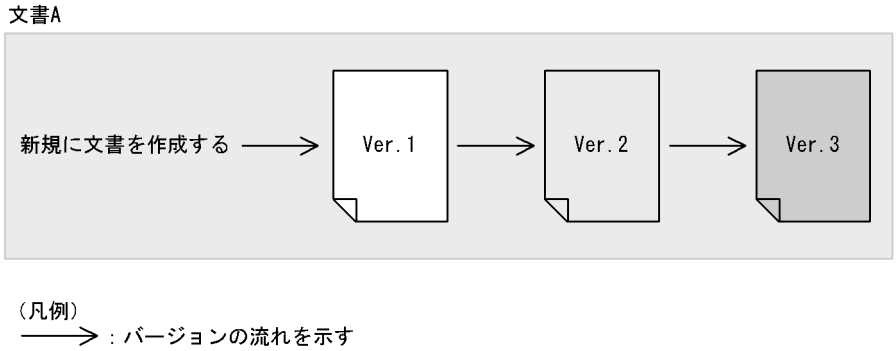
DocumentBroker では，文書に「第１版」「第２版」などの連続する版（**バージョン**）を付けて管理できます。文書にバージョンを付けて管理することによって，過去のある時点での文書を取り出したり，その文書を基にして新たな文書を作成することなどができます。

(2) バージョンの管理方法

バージョンは，文書を新規に作成して保存した時点から付けられます。保存されている文書を更新して，次に保存する時点で，バージョンを上げるかどうかを選択できます。また，文書に複数のバージョンが付けられている場合は，最新のバージョンを編集して，新しいバージョンとして保存できます。

文書のバージョン管理の概念を，次の図に示します。

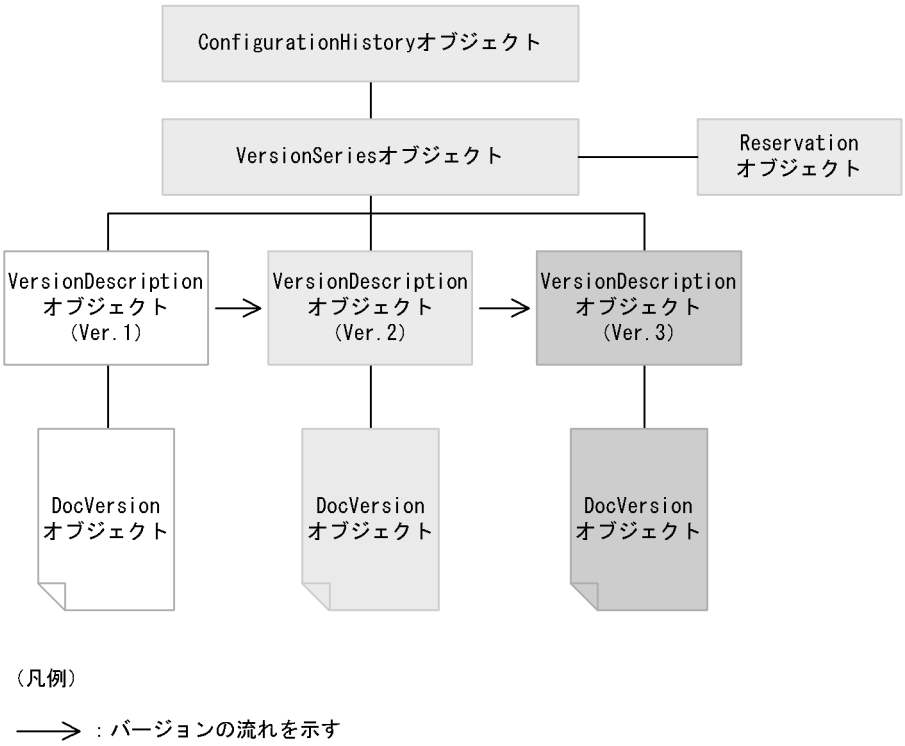
図 A-6　文書のバージョン管理の概念



なお，最後のバージョン番号が最新のバージョン（カレントバージョン）になります。

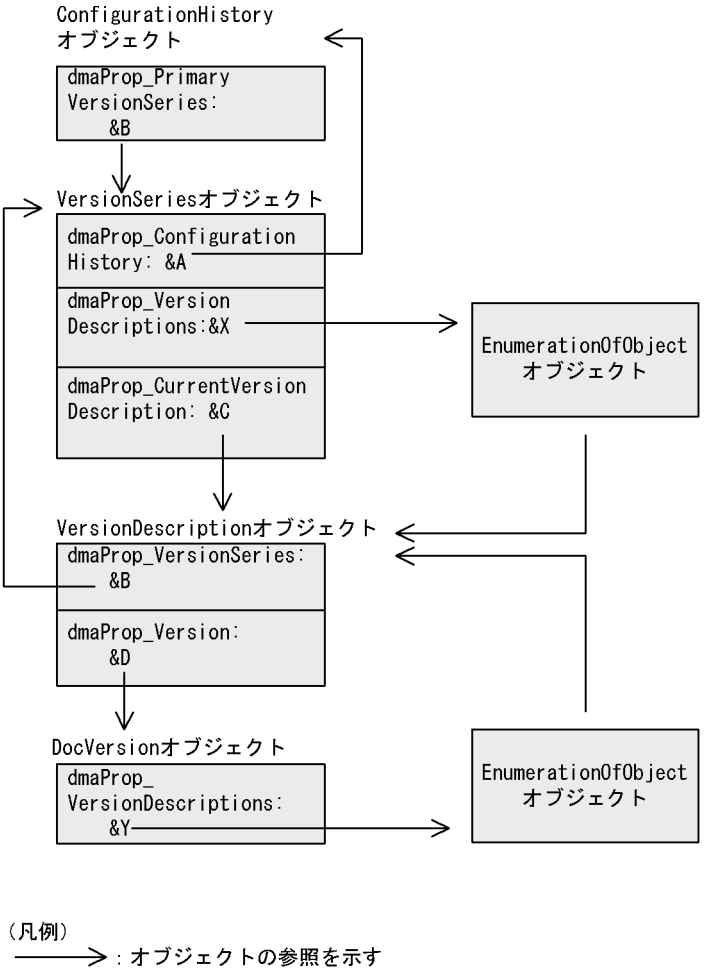
図 A-6 を，DMA オブジェクトの概念に置き換えて次の図に示します。

図 A-7　文書のバージョンを表現する DMA オブジェクトの関連（例）



文書をバージョン管理する場合の DMA オブジェクトの参照関係を次の図に示します。

図 A-8　バージョン管理での DMA オブジェクトの関係の例



次に，図 A-8 の説明を兼ねて文書のバージョン管理に使用する DMA オブジェクトについて説明します。

(a) ConfigurationHistory オブジェクト

文書の一連のバージョンを統括する DMA オブジェクトです。バージョン管理される文書の最上位に位置します。したがって，文書がバージョン管理されている場合，ConfigurationHistory オブジェクトが文書として表現されます。つまり，文書 A に第 1 版から第 3 版までのバージョンがある場合，ConfigurationHistory オブジェクトによって，第 1 版から第 3 版までの一連のバージョンが，まとめて文書 A として表現されます。

ConfigurationHistory オブジェクトは，図 A-8 に示したとおり，dmaProp_PrimaryVersionSeries プロパティに VersionSeries オブジェクトへのリファレンスを保持します。

(b) VersionSeries オブジェクト

バージョンの構成を管理するオブジェクトです。例えば，ある文書に対してバージョンが幾つあるか，最新のバージョンはどれか，などの情報を管理しています。VersionSeries オブジェクトは，ConfigurationHistory オブジェクトの直下に一つだけ存在します。

VersionSeries オブジェクトは，図 A-8 で示したとおり，dmaProp_VersionDescriptions プロパティに VersionDescriptions オブジェクトを要素に持つ EnumerationOfObject オブジェクトへのリファレンスを保持します。また，dmaProp_CurrentVersionDescription プロパティには，最新のバージョンの

VersionDescription オブジェクトへのリファレンスが格納されています。このプロパティによって EnumerationOfObject オブジェクトを介さずに最新のバージョンの文書を探索できます。

(c) VersionDescription オブジェクト

図 A-8 で示したとおり，VersionDescription オブジェクトは，dmaProp_VersionSeries プロパティに VersionSeries オブジェクトへのリファレンスを保持し，dmaProp_Version プロパティに DocVersion オブジェクトへのリファレンスを保持しています。このように，VersionDescription オブジェクトは，VersionSeries オブジェクトに DocVersion オブジェクトを結び付けて，VersionSeries オブジェクトで管理するバージョンに対応する文書（DocVersion オブジェクト）が何であるかを管理します。

(d) DocVersion オブジェクト

文書の特定のバージョンに相当する DMA オブジェクトです。バージョン管理されている DocVersion オブジェクトは図 A-8 のように，dmaProp_VersionDescriptions プロパティに，VersionDescription オブジェクトを要素に持つ EnumerationOfObject オブジェクトへのリファレンスを保持します。

DocVersion オブジェクトの詳細については，「付録 A.3　DMA オブジェクトで表す文書」を参照してください。

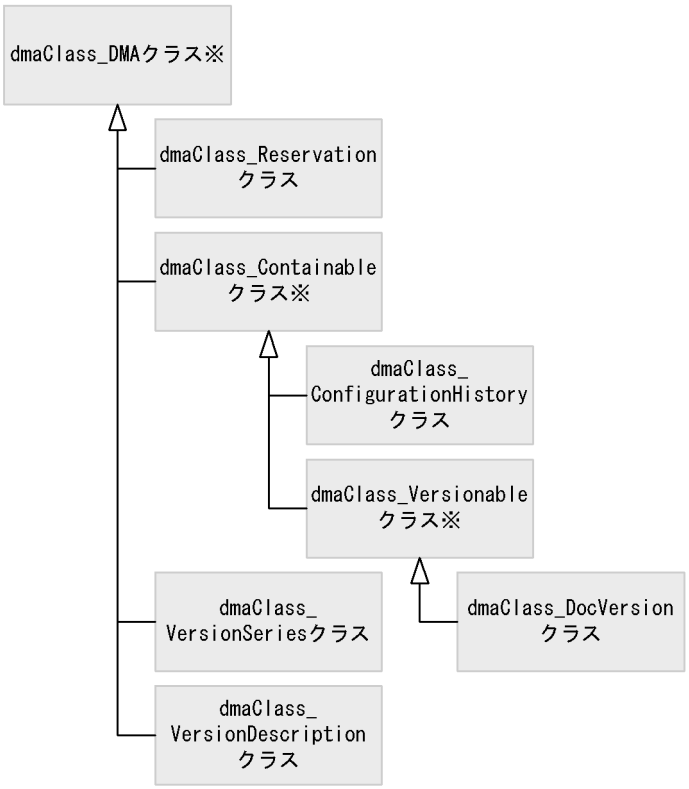
(e) Reservation オブジェクト

新しいバージョンを作成する権利を VersionSeries オブジェクトに予約する場合に使用する DMA オブジェクトです。新しいバージョンのチェックインが完了すると削除されます。チェックインについては，「付録 A.4(4)　バージョン管理でのオブジェクトの操作」を参照してください。

(3) 文書のバージョン管理で使用する DMA クラス

文書のバージョン管理に使用する DMA クラスを，継承関係に基づいて次の図に示します。

図 A-9　文書のバージョン管理に使用する DMA クラスの継承関係



注※　直接インスタンスを持たない抽象クラスです。

(4) バージョン管理でのオブジェクトの操作

文書に新しいバージョンを作成するための操作の概要について説明します。

(a) 新しいバージョンを作成するための権利の予約

文書に新しいバージョンを作成するには，VersionSeries オブジェクトに新しいバージョンを追加するための権利を予約する必要があります。

(b) DocVersion オブジェクトのチェックイン

最新のバージョンとする DocVersion オブジェクトを，VersionSeries オブジェクトに登録します。さらに，VersionSeries オブジェクトに対して接続する VersionDescription オブジェクトが通知されます。これを**チェックイン**といいます。

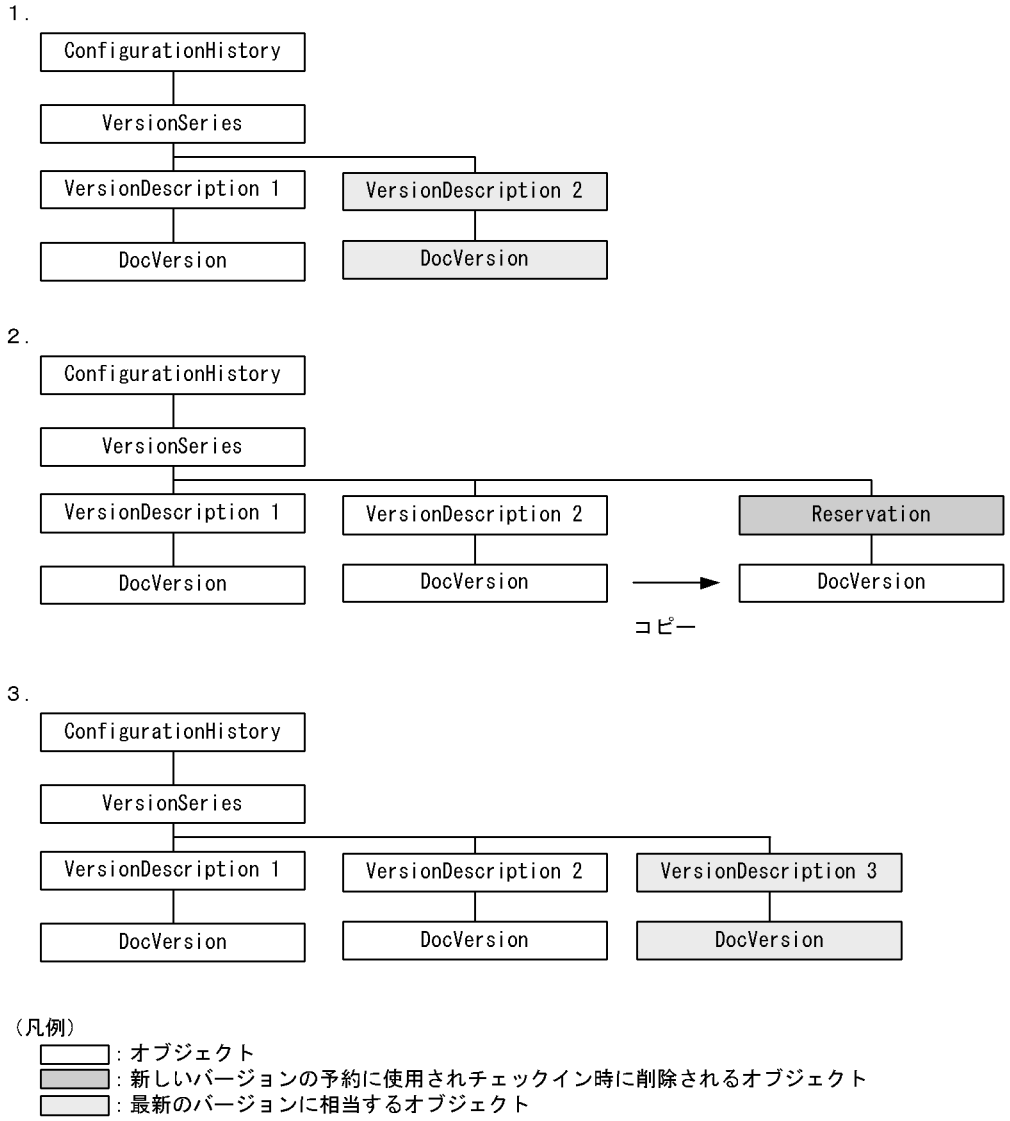
(c) 予約のキャンセル

(a) での操作をキャンセルすると，作成された Reservation オブジェクトは削除されます。

(d) バージョン管理でのオブジェクトの操作例

バージョン管理での DMA オブジェクトの操作例を，次の図に示します。

図 A-10　バージョン管理での DMA オブジェクトの操作例



付録 A.5　コンテインメントを利用した文書管理

ここでは，コンテインメントを利用して文書を管理する方法について説明します。

(1) コンテインメントの考え方

DocumentBroker では，**コンテインメント**（Containment：包含）というオブジェクト同士の関連を使って文書を管理できます。コンテインメントでは，ある二つのオブジェクトを想定して，一方を「包含するオブジェクト」，もう一方を「包含されるオブジェクト」として考えます。例えば，フォルダに文書を格納することを考えた場合，フォルダがオブジェクトを「包含するオブジェクト」，文書はオブジェクトに「包含されるオブジェクト」に相当します。なお，DocumentBroker では，オブジェクトを「包含するオブジェクト」を総称して**コンテナ**と呼びます。

コンテインメントでは，包含するオブジェクトと包含されるオブジェクトを関連づけるための DMA オブジェクトがあります。これを**関連オブジェクト**といいます。関連オブジェクトには，包含するオブジェクトと包含されるオブジェクトとの関係を表現するためのプロパティが用意されています。

なお，一つの関連オブジェクトが，複数のオブジェクト間の関係を示すために共用されることはありません。例えば，A オブジェクトに，B オブジェクトと C オブジェクトが包含される場合，A オブジェクトと B オブジェクトを関連づける関連オブジェクトと，A オブジェクトと C オブジェクトを関連づける関連オブジェクトの二つが必要です。このように，コンテインメントは「包含するオブジェクト」「関連オブジェクト」「包含されるオブジェクト」の三つのオブジェクトで表現されます。

(2) コンテインメントの種類

コンテインメントには，どの関連オブジェクトによって関連を表すかによって次の三つの種類があります。

● 直接型
関連オブジェクトとして DirectContainmentRelationship オブジェクトを使用して，包含するオブジェクトと包含されるオブジェクトを結び付けます。

● 参照型
関連オブジェクトとして ReferentialContainmentRelationship オブジェクトを使用して，包含するオブジェクトと包含されるオブジェクトを結び付けます。

● 構成管理型
関連オブジェクトとして VersionTraceableContainmentRelationship オブジェクトを使用して，包含するオブジェクトと包含されるオブジェクトを結び付けます。

次に各コンテインメントについて説明します。

(a) 直接型（DirectContainment）

直接型のコンテインメントでは，次のオブジェクトを使用してオブジェクト同士の関係を，親子の関係で示します。

包含するオブジェクト
直接型の関連オブジェクトを指し示す dmaProp_Children プロパティを保持する DMA オブジェクトは，すべて直接型で包含するオブジェクトとして使用できます。直接型で包含するオブジェクトとして使用できる代表的な DMA オブジェクトを示します。

- Container **オブジェクト**
  dmaClass_Container クラスを基にして作成する DMA オブジェクトです。
- ContainerVersion **オブジェクト**
  edmClass_ContainerVersion クラスを基にして作成する DMA オブジェクトで，構成管理型でオブジェクトを包含できる構成管理コンテナを表します。この DMA オブジェクトで表したコンテナは，バージョンを付けて管理することもできます。構成管理コンテナについては「付録 A.6　バージョン付き構成管理コンテナを利用した構成管理」を参照してください。

関連オブジェクト

- DirectContainmentRelationship **オブジェクト**
  dmaClass_DirectContainmentRelationship クラスを基にして作成する DMA オブジェクトです。包含するオブジェクトを指し示すプロパティと包含されるプロパティを指し示すプロパティを保持します。

包含されるオブジェクト
dmaClass_Containable クラスを継承し，直接型の関連オブジェクトを指し示す dmaProp_Parent プロパティを保持するサブクラスを基に作成した DMA オブジェクトであれば，すべて直接型で包含されます。直接型で包含される代表的な DMA オブジェクトを示します。

- Container **オブジェクト**
  dmaClass_Container クラスを基に作成する DMA オブジェクトです。一般的にフォルダにフォル

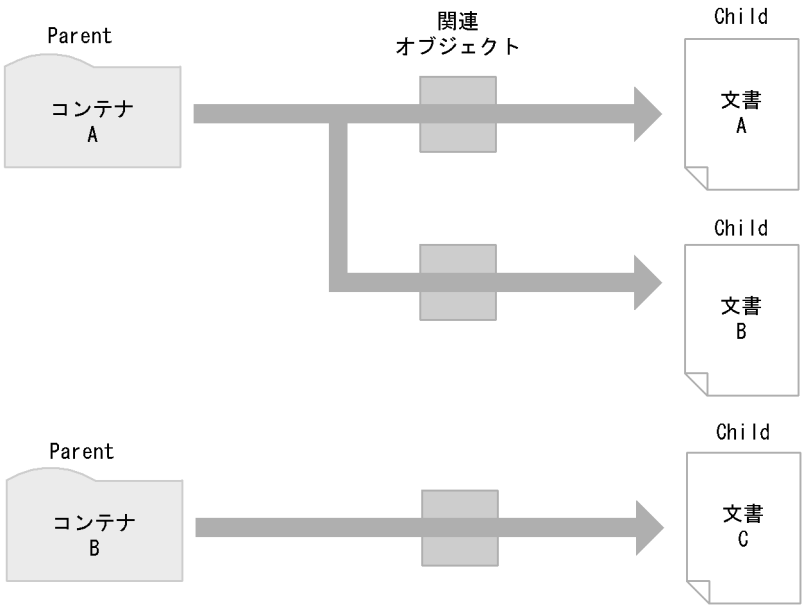
ダを格納するイメージで，コンテナにコンテナを包含することができます。

- ConfigurationHistory **オブジェクト**
- DocVersion **オブジェクト**

これらの DMA オブジェクトは，バージョン管理に使用できる DMA オブジェクトと同じです。したがって，バージョン管理とコンテインメントを組み合わせて文書を管理できます。

直接型での DMA オブジェクトの関係を次の図に示します。なお，コンテナ（Container オブジェクト）が直接型の関連オブジェクトを介して文書（DocVersion オブジェクト）を包含する場合を例にしています。

図 A-11　直接型でのオブジェクトの関係

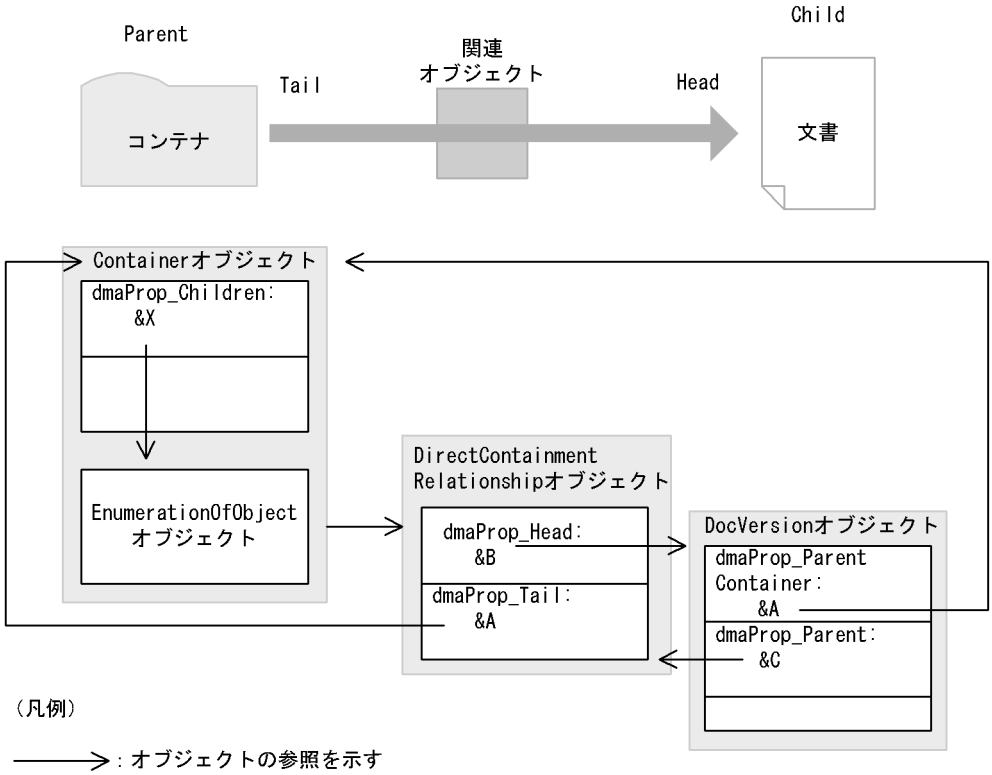


直接型では，オブジェクトを包含するオブジェクトのことを Parent，オブジェクトに包含されるオブジェクトのこと Child と呼びます。直接型に使用する関連オブジェクトは，DirectContainmentRelationship オブジェクトです。図 A-11 では，コンテナは文書の Parent に相当します。コンテナに包含される文書は，コンテナの Child に相当します。

Parent と Child の関係は 1 ：n（n は 1 以上の整数）です。つまり，Parent は複数の Child を包含できます。しかし，Child は一つの Parent だけに包含されます。図 A-11 のコンテナ A は複数の文書（文書 A および文書 B）を包含しています。しかし，文書 A および文書 B はコンテナ A にだけ包含され，文書 C はコンテナ B にだけ包含されています。例えば，文書 C がコンテナ A とコンテナ B に直接型で同時に包含されることはありません。

図 A-11 の関係を，DMA オブジェクトを使って次の図に示します。

図 A-12　直接型で使用する DMA オブジェクト



DirectContainmentRelationship オブジェクトは，dmaProp_Head および dmaProp_Tail プロパティを保持します。これらのプロパティは，包含するオブジェクト（図 A-12 ではコンテナ）から始まる矢印を描いた時の終点（Head）と始点（Tail）を示します。

dmaProp_Head プロパティには包含されるオブジェクト，dmaProp_Tail プロパティには包含するオブジェクトへのリファレンスが格納されます。図 A-12 では，dmaProp_Head プロパティには文書へのリファレンスが，dmaProp_Tail プロパティにはコンテナへのリファレンスが格納されています。

なお，Parent 内の Child の並びは保持されません。

(b) 参照型（ReferentialContainment）

参照型のコンテインメントでは，次の DMA オブジェクトを使用してオブジェクト同士の関係を示します。なお，関連オブジェクト以外の DMA オブジェクトについての説明は，「(a) 直接型（DirectContainment）」を参照してください。

包含するオブジェクト

参照型の関連オブジェクトを指し示す dmaProp_Containees プロパティを保持する DMA オブジェクトは，すべて参照型で包含するオブジェクトとして使用できます。参照型で包含するオブジェクトとして使用できる代表的な DMA オブジェクトを示します。

- Container **オブジェクト**
- ContainerVersion **オブジェクト**

関連オブジェクト

- ReferentialContainmentRelationship **オブジェクト**

  dmaClass_ReferentialContainmentRelationship クラスを基に作成する DMA オブジェクトです。

包含するオブジェクトを指し示すプロパティと包含されるオブジェクトを指し示すプロパティを保持します。

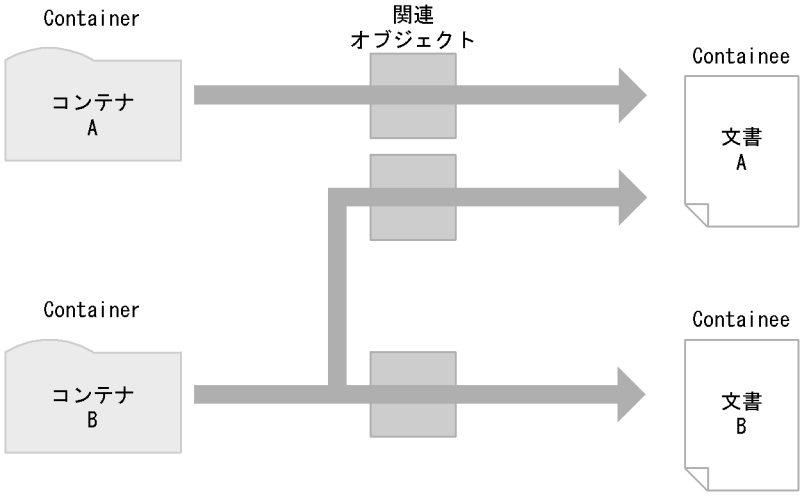
包含されるオブジェクト

dmaClass_Containable クラスを継承し，参照型の関連オブジェクトを指し示す dmaProp_Containers プロパティを保持するサブクラスを基に作成したオブジェクトであれば，すべて参照型で包含されます。参照型で包含される代表的な DMA オブジェクトを示します。

- Container **オブジェクト**
- ConfigurationHistory **オブジェクト**
- DocVersion **オブジェクト**

参照型のコンテインメントでの DMA オブジェクトの関係を次の図に示します。なお，コンテナ（Container オブジェクト）が参照型の関連オブジェクトを介して文書（DocVersion オブジェクト）を包含する場合を例にしています。

図 A-13　参照型での DMA オブジェクトの関係

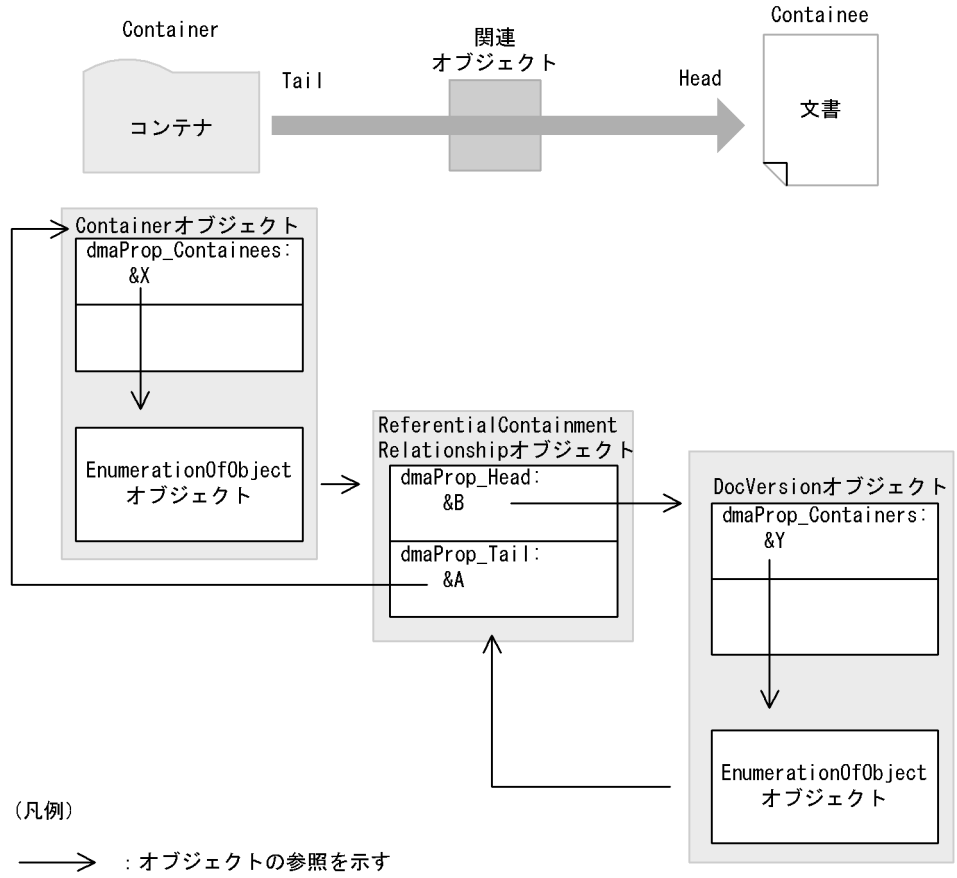


参照型では，オブジェクトを包含するオブジェクトを Container，オブジェクトに包含されるオブジェクトを Containee と呼びます。参照型に使用する関連オブジェクトは，ReferentialContainmentRelationship オブジェクトです。図 A-13 では，コンテナは文書の Container に相当します。コンテナに包含される文書は，コンテナの Containee に相当します。

Container と Containee の関係は n：m（n，m は 1 以上の整数）です。つまり，Container は複数の Containee を包含できます。Containee は，複数の Container に包含されることもできます。図 A-13 では，コンテナ B は複数の文書（文書 A および文書 B）を包含しています。文書 A は複数のコンテナ（コンテナ A およびコンテナ B）に同時に包含されています。

図 A-13 の関係を，DMA オブジェクトを使って次の図に示します。

図 A-14　参照型で使用する DMA オブジェクト



ReferentialContainmentRelationship オブジェクトも，DirectContainmentRelationship オブジェクトと同様に dmaProp_Head および dmaProp_Tail プロパティを保持します。

図 A-14 では，dmaProp_Head プロパティには文書へのリファレンスが，dmaProp_Tail プロパティにはコンテナへのリファレンスが格納されています。

なお，Container 内の Containee および Containee 内の Container の並びは保持されません。

(c) 構成管理型（VersionTraceableContainment）

構成管理型のコンテインメントでは，次のオブジェクトを使用してオブジェクト同士の関係を示します。

包含するオブジェクト

構成管理型の関連オブジェクトを指し示す edmProp_VTContainees プロパティを保持する DMA オブジェクトは，すべて構成管理型で包含するオブジェクトとして使用できます。構成管理型で包含するオブジェクトとして使用できる DMA オブジェクトを示します。

- ContainerVersion **オブジェクト**

  edmClass_ContainerVersion クラスを基にして作成するオブジェクトで，コンテナにバージョンを付けて構成管理型でオブジェクトを包含できるバージョン付き構成管理コンテナを表します。

関連オブジェクト

- VersionTraceableContainmentRelationship **オブジェクト**

  edmClass_VersionTraceableContainmentRelationship クラスを基に作成する DMA オブジェクトです。包含するオブジェクトを指し示すプロパティと包含されるプロパティを指し示すプロパティ

を保持します。また，特定のバージョンを固定して包含するか最新のバージョンを包含するかを指定できる edmProp_VTMode プロパティを保持します。

包含されるオブジェクト

dmaClass_Versionable クラスを継承するサブクラスのうち，構成管理型の関連オブジェクトを指し示す edmProp_VTContainers プロパティを保持する DMA クラスを基に作成した DMA オブジェクトであれば，すべて構成管理型で包含されます。

なお，dmaClass_Versionable クラスは，dmaClass_Containable クラスを継承するため，dmaProp_Parent および dmaProp_Containers プロパティを保持します。したがって，構成管理型で包含されるオブジェクトは，直接型および参照型でも包含されます。

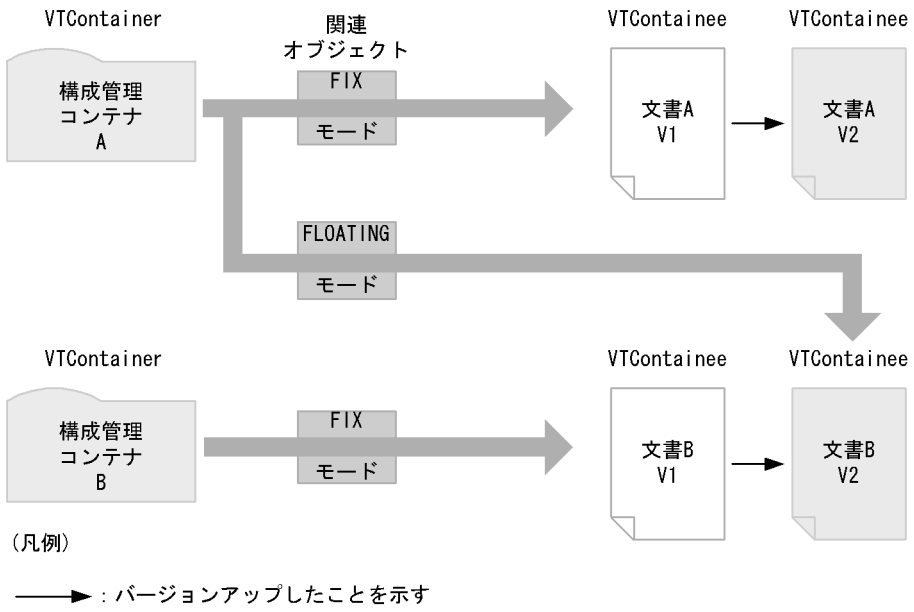
構成管理型で包含される DMA オブジェクトを示します。

- VersionTracedDocVersion **オブジェクト**
  dmaClass_DocVersion クラスのサブクラスの edmClass_VersionTracedDocVersion クラスを基に作成する DMA オブジェクトです。構成管理型で管理する文書を表します。
- ContainerVersion **オブジェクト**
- VersionTracedComponentDocVersion オブジェクト
  dmaClass_DocVersion クラスのサブクラスの edmClass_VersionTracedDocVersion クラスを基に作成する DMA オブジェクトです。

なお，構成管理型の場合，包含されるオブジェクトのことを特に**構成要素**と呼びます。構成要素となれるオブジェクトは，一つの直接型のコンテインメント，複数の参照型のコンテインメントおよび複数の構成管理型のコンテインメントを結べます。

構成管理型のコンテインメントでの DMA オブジェクトの関係を次の図に示します。なお，構成管理コンテナ（ContainerVersion オブジェクト）が構成管理型の関連オブジェクトを介して文書（VersionTracedDocVersion オブジェクト）を包含する場合を例にしています。

図 A-15　構成管理型での DMA オブジェクトの関係



構成管理型では，オブジェクトを包含するオブジェクトを VTContainer，オブジェクトに包含されるオブジェクトを VTContainee と呼びます。構成管理型に使用する関連オブジェクトは，

VersionTraceableContainmentRelationship オブジェクトです。図 A-15 では，構成管理コンテナは文書の VTContainer です。構成管理コンテナに包含される文書は，構成管理コンテナの VTContainee です。

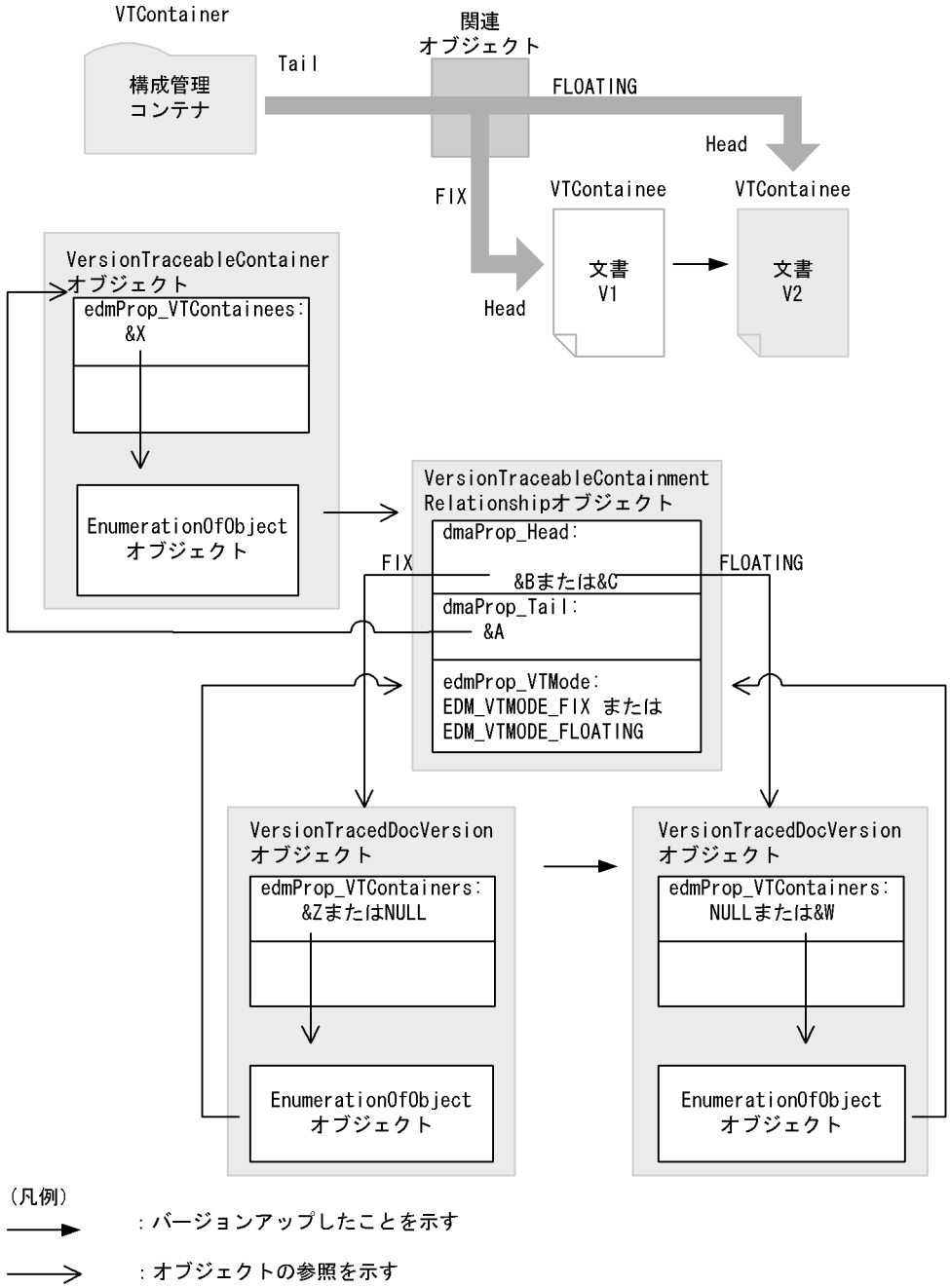
VTContainer は，バージョン管理されているオブジェクトのバージョンをたどって包含できるオブジェクトです。VTContainee は，バージョンをたどって包含されるオブジェクトです。

構成管理型では，最新のバージョンのオブジェクトを追跡してコンテインメントを結ぶ FLOATING **モード**と，バージョンを確定してコンテインメントを結ぶ FIX **モード**のどちらかを選択できます。FLOATING モードと FIX モードを総称して**構成管理モード**と呼びます。構成管理モードが FLOATING モードの場合，VTContainer は常に最新のオブジェクトを包含します。構成管理モードが FIX モードの場合，VTContainer は包含するオブジェクトがバージョンアップしても同じオブジェクトを包含し続けます。図 A-15 では，構成要素の文書 A と文書 B が共に V 1 から V 2 にバージョンアップしたときに構成管理モードによって構成管理コンテナ A および構成管理コンテナ B と文書のバージョンの関連を示しています。構成管理コンテナ A は，文書 A を FIX モードで，文書 B を FLOATING モードで包含しています。したがって，文書が V1 から V2 にバージョンアップされると，構成管理コンテナ A は，文書 A については V1 を包含し続け，文書 B については最新の V2 を包含します。

VTContainer と VTContainee の関係は参照型の Container と Containee の関係と同様に n：m（n，m は 1 以上の整数）で，VTContainer は複数の VTContainee を包含できます。VTContainee は，複数の VTContainer に包含されることもできます。

図 A-15 の関係を DMA オブジェクトを使って次の図に示します。

図 A-16　構成管理型で使用する DMA オブジェクト



注　VersionTracedDocVersionオブジェクトのバージョンを管理するオブジェクト群（ConfigurationHistoryオブジェクト，VersionSeriesオブジェクトおよびVersionDescriptionオブジェクト）は省略しています

VersionTraceableContainmentRelationship オブジェクトも，DirectContainmentRelationship オブジェクト，ReferentialContainmentRelationship オブジェクトと同様に dmaProp_Head および dmaProp_Tail プロパティを保持します。

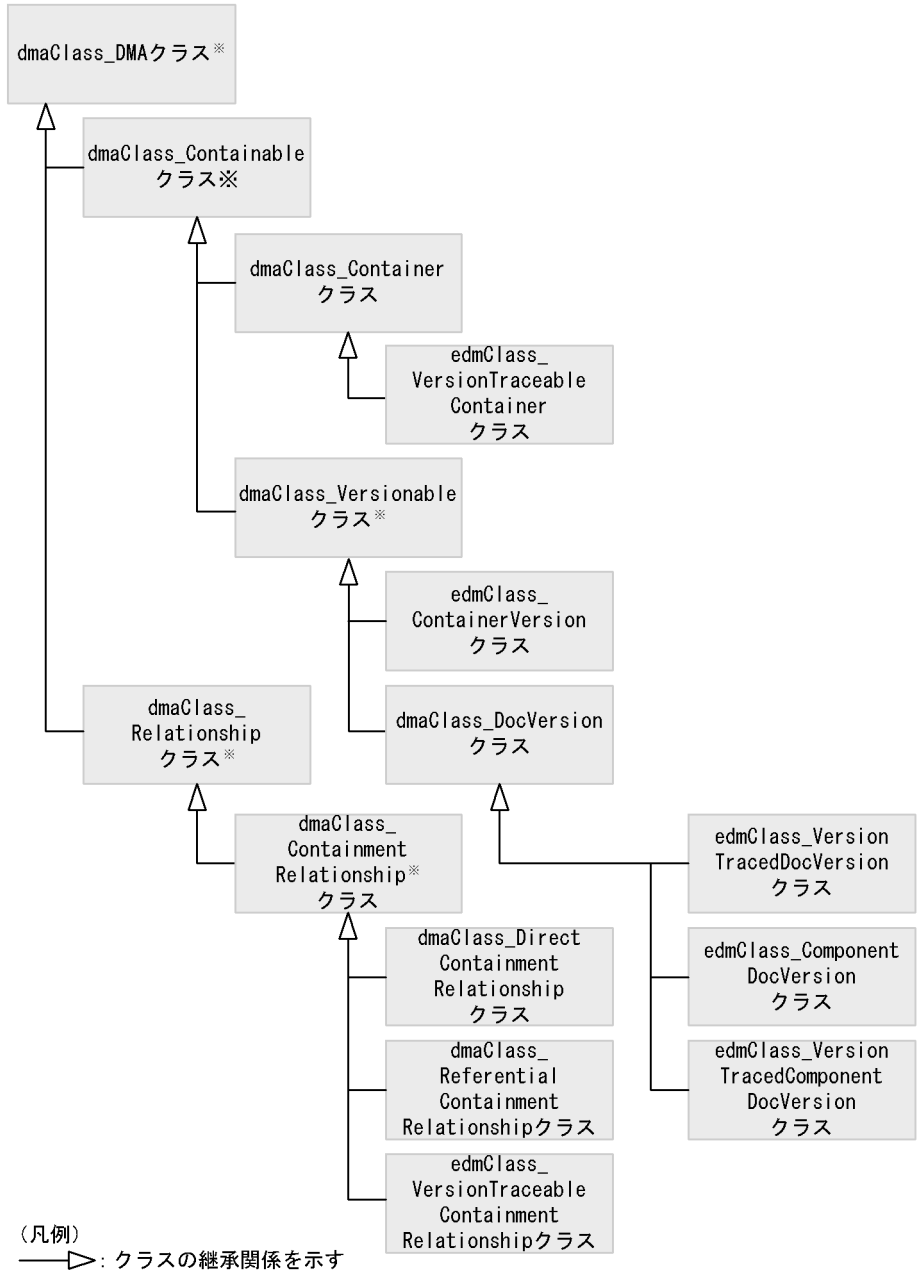
図 A-16 では，dmaProp_Head プロパティには文書へのリファレンスが，dmaProp_Tail プロパティには構成管理コンテナへのリファレンスが格納されています。なお，VTContainer 内の VTContainee および VTContainee 内の VTContainer の並びは保持されません。

VersionTraceableContainmentRelationship オブジェクトの edmProp_VTMode プロパティの値が EDM_VTMODE_FLOATING（FLOATING モード）の場合，参照先の DMA オブジェクトがバージョンアップすると dmaProp_Head プロパティの値が変化して最新のバージョンを指します。
edmProp_VTMode プロパティの値が EDM_VTMODE_FIX（FIX モード）の場合，参照先の DMA オブジェクトがバージョンアップしても dmaProp_Head プロパティの値は変わりません。

(3) コンテインメントに使用する DMA クラス

コンテインメントに使用する DMA クラスを，継承関係に基づいて次の図に示します。

図 A-17　コンテインメントに使用する DMA クラスの継承関係



注※　直接インスタンスを持たない抽象クラスです。

付録 A.6　バージョン付き構成管理コンテナを利用した構成管理

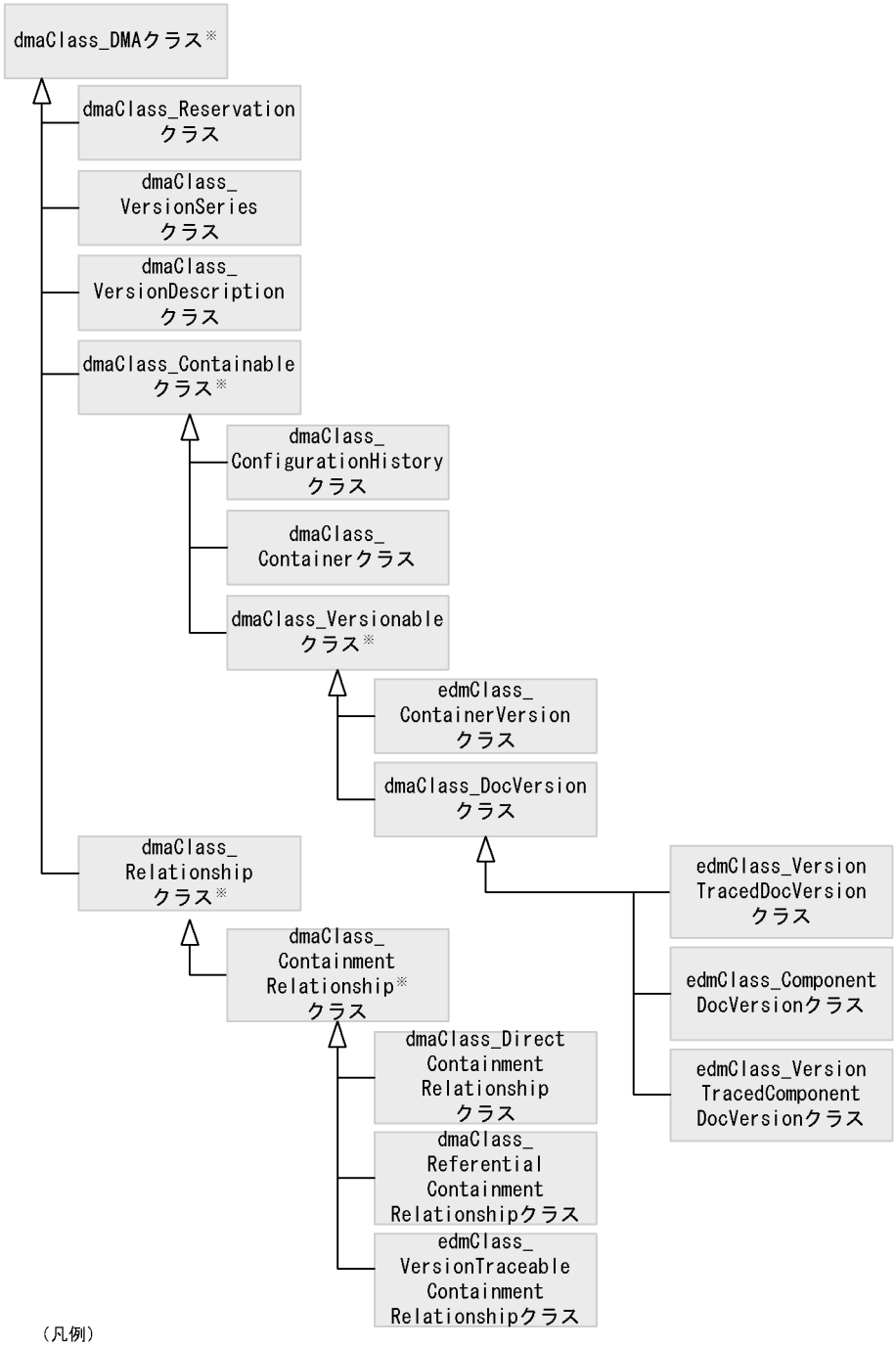
ここでは，関連づけて管理している文書の集合全体としてのバージョンを管理したり，要素ごとにバージョンを指定して集合全体の構成を管理したりする方法について説明します。

(1) バージョン付き構成管理コンテナを利用した構成管理で使用する DMA クラス

バージョン付き構成管理コンテナを利用した構成管理では，edmClass_ContainerVersion クラス，関連オブジェクトの DMA クラスおよびバージョン管理の DMA クラスを使用します。
edmClass_ContainerVersion クラスを基に作成する ContainerVersion オブジェクトは，コンテナにバージョンを付けて包含するオブジェクトを構成管理できるコンテナです。ContainerVersion オブジェクトのことを特に**バージョン付き構成管理コンテナ**と呼びます。

バージョン付き構成管理コンテナを利用した構成管理で使用する DMA クラスを継承関係に基づいて次の図に示します。

図 A-18　バージョン付き構成管理コンテナを利用した構成管理で使用する DMA クラス



注※　直接インスタンスを持たない抽象クラスです。

(2) バージョン付き構成管理コンテナの機能

バージョン付き構成管理コンテナ（ContainerVersion オブジェクト）には，次に示す機能があります。

● 包含するオブジェクトとしての機能

- 直接型または参照型で包含する**コンテナの機能**
- 構成管理型で包含する**構成管理コンテナの機能**

● バージョン管理されるオブジェクトとしての機能

● 包含されるオブジェクトとしての機能

- 直接型，参照型または構成管理型で包含されることができます。

(a) 包含するオブジェクトとしての機能

● バージョン付き構成管理コンテナが保持するコンテナ機能のプロパティ

edmClass_ContainerVersion クラスは，次に示すコンテナの機能を果たすためのプロパティを保持します。

コンテナの機能のためのプロパティ

- dmaProp_Children プロパティ
  直接型のコンテインメントで格納されるオブジェクトとの関連を表す DirectContainmentRelationship オブジェクトを要素とする EnumerationOfObject オブジェクトへのリファレンスが格納されます。
- dmaProp_Containees プロパティ
  参照型で格納されるオブジェクトとの関連を表す ReferentialContainmentRelationship オブジェクトを要素とする EnumerationOfObject オブジェクトへのリファレンスが格納されます。

したがって，ContainerVersion オブジェクトは，直接型または参照型でほかのオブジェクトを包含するコンテナの機能を持ちます。
また，ContainerVersion オブジェクトは，次に示す構成管理コンテナ機能を果たすためのプロパティを保持します。

構成管理コンテナ機能のためのプロパティ

- edmProp_VTContainees プロパティ
  構成管理されて格納される構成要素との関連を表す VersionTraceableContainmentRelationship オブジェクトを要素とする EnumerationOfObject オブジェクトへのリファレンスが格納されます。

したがって，ContainerVersion オブジェクトは，構成管理型でほかのオブジェクトを包含する構成管理コンテナの機能を持ちます。

● バージョン付き構成管理コンテナに包含されるオブジェクト

バージョン付き構成管理コンテナは，dmaProp_Parent プロパティ，dmaProp_Containers プロパティおよび edmProp_VTContainers プロパティを保持して直接型で包含されるオブジェクト，参照型で包含されるオブジェクトおよび構成管理型で包含されるオブジェクトを包含できます。バージョン付き構成管理コンテナに包含できる DMA オブジェクトを示します。

- DocVersion オブジェクト
  直接型および参照型で包含される文書です。
- VersionTracedDocVersion オブジェクト
  直接型，参照型および構成管理型で包含される文書です。
- ConfigurationHistory オブジェクト
  直接型および参照型で包含されるバージョン管理されている文書の一連のバージョンです。
- Container オブジェクト
  直接型および参照型で包含されるコンテナです。
- VersionTraceableContainer オブジェクト
  直接型および参照型で包含される構成管理コンテナです。
- ContainerVersion オブジェクト
  直接型，参照型および構成管理型で包含されるバージョン付き構成管理コンテナです。

(b) バージョン管理されるオブジェクトとしての機能

edmClass_ContainerVersion クラスは，dmaClass_Versionable クラスを継承します。したがって，文書をバージョン管理するのと同様に ConfigurationHistory オブジェクト，VersionSeries オブジェクトおよび VersionDescription オブジェクトを使用してバージョン管理できます。バージョン管理の方法の詳細については，「付録 A.4(2)　バージョンの管理方法」を参照してください。

(c) 包含されるオブジェクトとしての機能

edmClass_ContainerVersion クラスは，dmaClass_Containable クラスからオブジェクトに「包含されるオブジェクト」となれるプロパティ（dmaProp_Parent プロパティおよび dmaProp_Containers プロパティ）を継承しています。また，構成管理型で包含される構成要素となれるプロパティ（edmProp_VTContainers プロパティ）も保持します。したがって，バージョン付き構成管理コンテナは，すべての種類のコンテナ（Container オブジェクトおよび ContainerVersion オブジェクト）に包含されることができます。

(3) バージョン付き構成管理コンテナのバージョンの追加

バージョン付き構成管理コンテナは，文書のバージョンの追加と同じ方法でバージョンアップできます。

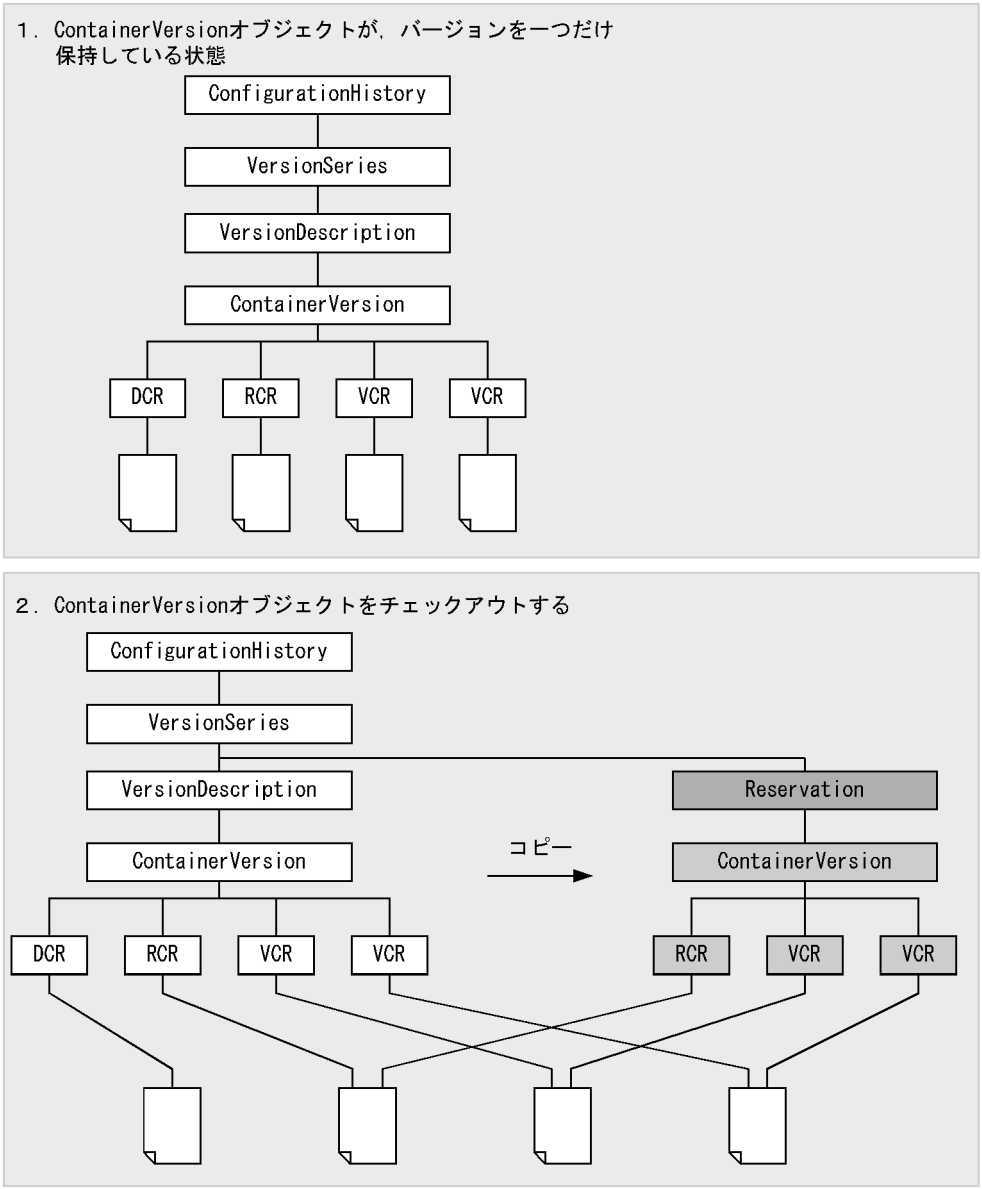
(a) チェックアウト時にコピーされる DMA オブジェクト

バージョン付き構成管理コンテナのバージョンをチェックアウトすると，次に示す DMA オブジェクトがコピーされて Reservation オブジェクトに接続されます。

- 最新の ContainerVersion オブジェクト
- ReferentialContainmentRelationship オブジェクトおよび VersionTraceableContainmentRelationship オブジェクト

コピーされた ReferentialContainmentRelationship オブジェクトおよび VersionTraceableContainmentRelationship オブジェクトは，ContainerVersion オブジェクトが永続化されるときに一緒に永続化されます。バージョン付き構成管理コンテナのバージョンのチェックアウト時の DMA オブジェクトの関係を次の図に示します。なお，バージョン付き構成管理コンテナが文書を包含している場合を例にしています。

図 A-19　バージョン付き構成管理コンテナのバージョンのチェックアウト時の DMA オブジェクトの関係



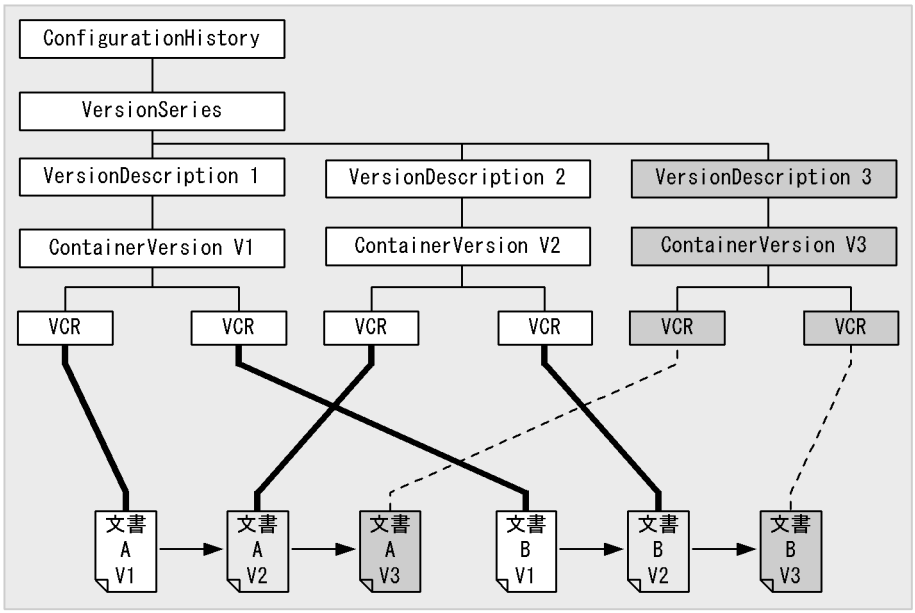
ReferentialContainmentRelationship オブジェクトまたは VersionTraceableContainmentRelationship オブジェクトを介して包含されるオブジェクトは，複数のコンテナに同時に包含されることができます。しかし，DirectContainmentRelationship オブジェクトを介して包含されるオブジェクトは，複数のコンテナに直接型で同時に包含されることはありません。そのため，DirectContainmentRelationship オブジェクトは，チェックアウト時にコピーされません。

(4) バージョン付き構成管理コンテナのバージョンの追加と構成管理

バージョン付き構成管理コンテナのバージョン管理機能と構成管理コンテナ機能を併用すると，バージョン付き構成管理コンテナを一つの概念文書のように扱って，コンテナの内の構成管理および構成要素集合のバージョン管理ができます。

バージョン付き構成管理コンテナに新規バージョンが追加されたあと，構成要素のバージョンが追加されると，構成管理モードが FIX モードか FLOATING モードかによって，次の図のようにバージョン付き構成管理コンテナに包含されるオブジェクトが決まります。

図 A-20　バージョン付き構成管理コンテナの構成要素のバージョンの追加の例



（凡例）

：VersionTracedDocVersionオブジェクト

：最新のバージョンに相当するオブジェクト

：FIXモードで包含する文書のバージョンが固定されていることを示す

：FLOATINGモードで最新のバージョンを包含していることを示す

：バージョンの流れを表す

VCR ： VersionTraceableContainmentRelationshipオブジェクト

注　文書をバージョン管理するためのオブジェクト群（ConfigurationHistoryオブジェクト，VersionSeriesオブジェクトおよびVersionDescriptionオブジェクト）は省略しています

図 A-20 は，バージョン付き構成管理コンテナが構成管理型で文書（VersionTracedDocVersion）を包含しており，構成要素が V1 から V3 までバージョンアップしている例です。ContainerVersion オブジェクトの V1 は，文書 A と文書 B の V1 を FIX モードで包含しています。ContainerVersion オブジェクトの V2 は，文書 A と文書 B の V2 を FIX モードで包含しています。ContainerVersion の V3 は，文書 A と文書 B を FLOATING モードで包含しており，現在 V3 が最新バージョンなので，V3 の文書を包含しています。このように，複数の文書を構成管理型でバージョン付き構成管理コンテナに格納すると，コンテナごとに構成要素のバージョンを確定し，コンテナをバージョンアップすることによって，構成要素集合のバージョンを管理できます。

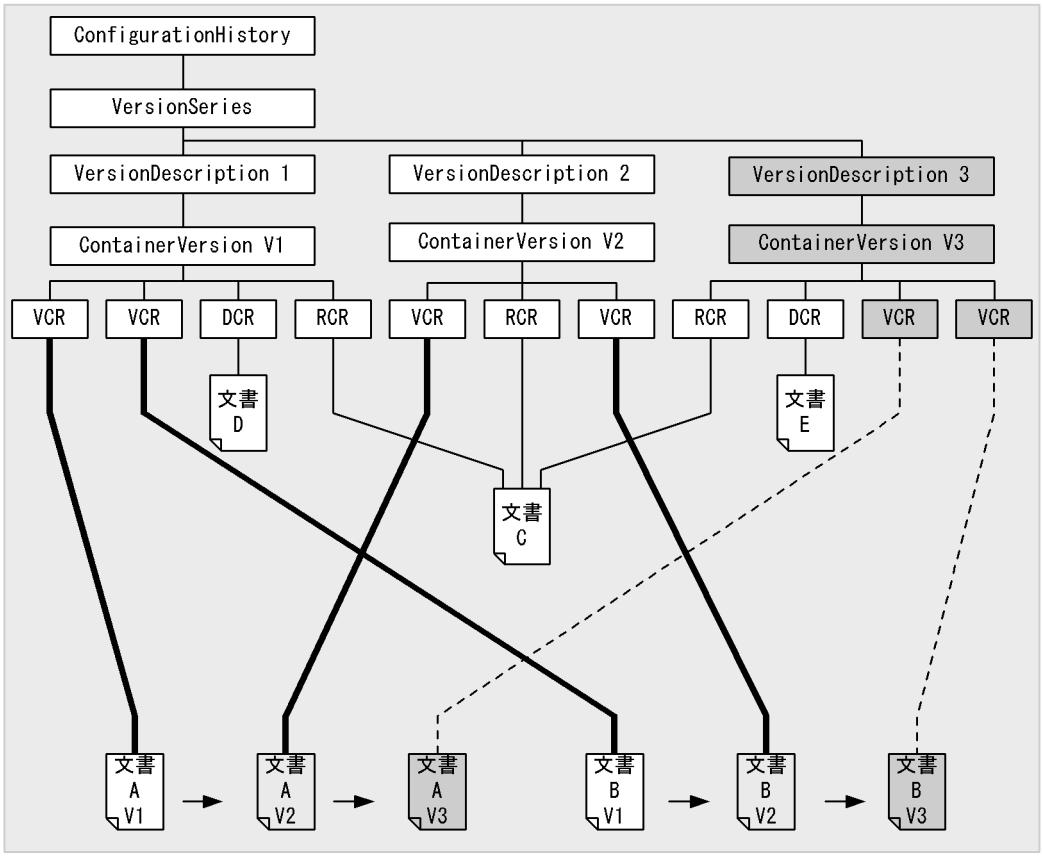
別のバージョン付き構成管理コンテナ（ContainerVersion オブジェクト）も同様にバージョン付き構成管

理コンテナで構成管理できます。

(5) バージョン付き構成管理コンテナに格納されたオブジェクトの管理例

バージョン付き構成管理コンテナには，複数の異なる関連オブジェクトを使用して文書を格納できます。関連オブジェクトをどのように使い分けて文書を管理するか，例を次の図に示します。

図 A-21　バージョン付き構成管理コンテナに格納された文書の管理例



（凡例）

：最新のバージョンに相当するオブジェクト

：DocVersionオブジェクトまたはdmaClass_DocVersionクラスのサブクラスのオブジェクト

：バージョンの流れを表す

：FIXモードでバージョンが固定されていることを示す

：FLOATINGモードで最新バージョンを包含していることを示す

DCR：DirectContainmentRelationshipオブジェクト
RCR：ReferentialContainmentRelationshipオブジェクト
VCR：VersionTraceableContainmentRelationshipオブジェクト

注　文書をバージョン管理するためのオブジェクト群（ConfigurationHistoryオブジェクト，VersionSeriesオブジェクトおよびVersionDescriptionオブジェクト）は省略しています

文書 A と文書 B は VersionTraceableContainmentRelationship オブジェクトを介してバージョン付き構成管理コンテナのすべてのバージョンに渡って包含され，構成管理される文書です。文書 C は ReferentialContainmentRelationship オブジェクトを介してすべてのバージョン付き構成管理コンテナのバージョンに渡って包含される文書です。文書 D と文書 E は DirectContainmentRelationship オブジェ

クトを介して特定のバージョンのバージョン付き構成管理コンテナにだけ包含される文書です。

ある研究論文をバージョン付き構成管理コンテナで管理する例を図 A-21 に当てはめて説明します。次のような三つのバージョン付き構成管理コンテナのバージョンがあるとします。

- ContainerVersion オブジェクト V1
  セミナーで発表するまでのバージョンの文書を格納します。
- ContainerVersion オブジェクト V2
  研究論文として完成したバージョンを格納します。
- ContainerVersion オブジェクト V3
  学会での発表のためにさらに修正したバージョンを格納します。

そして，研究論文が次のような複数の文書で構成されているとします。

- 本論 .doc
  研究論文の本論。
- 結論 .doc
  研究論文の結論部分。
- テーマ .doc
  研究論文のテーマについての概要説明。
- 発表 .doc
  セミナー，学会での発表用のレジュメ。

「本論 .doc」および「結論 .doc」は，すべてのバージョンに渡って包含される内容の文書であり，セミナー発表と論文完成および学会発表のタイミングでバージョンを確定して管理する文書です。したがって，図 A-21 の文書オブジェクト A と B のように VersionTraceableContainmentRelationship オブジェクトを介して格納します。

「テーマ .doc」は研究論文の概要説明です。すべてのバージョンのバージョン付き構成管理コンテナに渡って包含される内容の文書で，一度作成してしまえば更新する必要のない文書です。したがって，図 A-21 の文書オブジェクト C のように ReferentialContainmentRelationship オブジェクトを介して格納します。

「発表 .doc」は，セミナーと学会があるときだけに臨時に作成した文書で，V1 と V3 のバージョン付き構成管理コンテナにだけ必要な文書です。したがって，図 A-21 の文書 D と E のように DirectContainmentRelationship オブジェクトを介してそれぞれのバージョン付き構成管理コンテナに格納します。

付録 B　edmSQL の構文解析エラー情報

ここでは，edmSQL を使用した検索で構文解析エラーになった場合に出力される，構文解析エラーの内容について説明します。

edmSQL の文法については，「4.5　edmSQL の文法」を参照してください。

なお，edmSQL の構文解析でエラーが発生した場合，メソッドの戻り値は次の値になります。

```
major_code：ERR_DBR
minor_code：ERR_EQL_BAD_STATEMENT
```

付録 B.1　記述形式

このマニュアルでの構文解析エラーの記述形式について説明します。

＜メッセージタイプ＞＜メッセージ ID ＞

メッセージテキスト

パラメタの内容

埋め込みパラメタがある場合の，パラメタの説明です。埋め込みパラメタは，% で表記します。

説明

エラーの内容と対処方法についての説明です。

付録 B.2　構文解析エラー

Error 1001

トークン "%1"(%2 バイト目) で構文エラーを検出しました。

A syntax error was detected in the token "%1" (byte number %2).

パラメタの内容

- %1：構文解析エラーを検出したトークンの文字列
- %2：%1 のトークンが出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

入力した edmSQL 文で，表示されたトークンまたはそれよりも前に出現するトークンが，構文規則に合っていません。
正しい構文でトークンを指定してください。

付録 B.3　字句解析エラー

Error 2001

ID **文字列 "%1"(%2 バイト目) は，長さ，文字又はハイフン位置のどれかが不正です。**

The ID character string "%1" (byte number %2) includes an invalid length, character, or hyphen position.

パラメタの内容

- %1：指定した ID 文字列の先頭からの内容（64 バイトまで出力）
- %2：%1 の ID 文字列が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

ID 文字列に次の不正のどれかを検出しました。

- ID 文字列では使用できない文字が指定されています。
- ID 文字列の形式よりも長い文字列が指定されています。
- ハイフンの位置が正しくありません。

内容を確認して，ID 文字列の形式に従って修正してください。

Error 2002

数値で始まる文字列 "%1"(%2 バイト目) は，数値定数の形式に対して不正です。

The character string "%1" (byte number %2) beginning with a numeric has an invalid format for a numeric constant.

パラメタの内容

- %1：指定した数値定数の先頭からの内容（64 バイトまで出力）
- %2：%1 の数値定数が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

数値として指定できない文字を指定しています。例えば，次のような指定をしています。

- 識別子を数字で開始しています。
- 数値定数と識別子を区切らずに並べて指定しています。

内容を確認して，次の方法で対処してください。

- 整数定数の形式に従って修正してください。
- 数字で始まる識別子は，先頭をアルファベットにするか，区切られた識別子の形式に修正してください。
- 区切り文字を入れて，数値定数と分離して指定してください。

Error 2003

記号 "%1" (%2 バイト目) は未サポートです。

Symbol "%1" (byte number %2) is not supported.

パラメタの内容

- %1：記号文字列
- %2：%1 の記号が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

指定した記号は，使用できません。この記号は，edmSQL の文法の演算子などとして予約されている記号ですが，構文では使用できません。

Error 2004

アンダースコア "_" (%1 バイト目) は，識別子の先頭や演算子などの記号としては使用できません。

An underscore (byte number %1) cannot be used at the beginning of an ID or as a symbol of an operator etc.

パラメタの内容

- %1：アンダースコアが出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

アンダースコアは，識別子の先頭や演算子の記号としては使用できません。

Error 2005

文字列定数の終端を検知する前に文が終了しました。指定内容 "%1" (%2 バイト目)

The statement ended before the end of a character string constant was detected. Specification "%1" (byte number %2).

パラメタの内容

- %1：指定した文字列定数（64 バイトまで出力）
- %2：%1 の文字列定数が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

文字列定数の終端記号である「'」が検知される前に，edmSQL 文が終了しました。
文字列定数の終端に「'」を挿入してください。

Error 2006

区切られた識別子の終端を検知する前に文が終了しました。指定内容 "%1" (%2 バイト目)

The statement ended before the end of a delimited ID was detected. Specification "%1" (byte number %2).

パラメタの内容

- %1：区切られた識別子（64 バイトまで出力）
- %2：%1 の区切られた識別子が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

区切られた識別子の終端記号である「"」が検知される前に，edmSQL 文が終了しました。
区切られた識別子の終端に，「"」を挿入してください。

Error 2007

文字 "%1" (%2 バイト目) は，文字列定数以外では使用できません。

Character "%1" (byte number %2) can be used in character string constants only.

パラメタの内容

- %1：指定した文字
- %2：%1 で指定した文字が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

指定した文字は，edmSQL 文で使用できる文字ではありません。

付録 B.4　意味解析エラー

(1) プロパティ表現に関するエラー

Error 3001

プロパティ "%1"(%2 バイト目) は，永続プロパティでないため，edmSQL 文に指定できません。

The property "%1" (byte number %2) cannot be specified for the edmSQL statement because it is not a persistent property.

パラメタの内容

- %1：プロパティ名
- %2：%1 のプロパティ名が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

検索対象となるプロパティは，永続プロパティ（データベース上に存在するプロパティ）である必要があります。
指定したプロパティが永続プロパティであるかどうか確認してください。

Error 3002

プロパティ "%1"(%2 **バイト目) は，選択可能** (Selectable) **でないため，選択項目に指定できません。**

The property "%1" (byte number %2) cannot be specified as a selection item because it is not selectable.

パラメタの内容

- %1：プロパティ名
- %2：%1 のプロパティ名が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

DMA の検索モデルに基づき，選択可能（Selectable）でないプロパティは選択項目（SELECT 句）には指定できません。
選択可能なプロパティだけを選択項目に指定してください。必要に応じてメタ情報を修正してください。

Error 3003

プロパティ "%1"(%2 **バイト目) は，検索可能** (Searchable) **でないため，検索条件に指定できません。**

The property "%1" (byte number %2) cannot be specified as a search condition because it is not searchable.

パラメタの内容

- %1：プロパティ名
- %2：%1 のプロパティ名が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

DMA の検索モデルに基づき，検索可能 (Searchable) でないプロパティは条件式 (WHERE 句や ON 条件) には指定できません。
検索可能なプロパティだけを条件式に指定してください。必要に応じてメタ情報を修正してください。

Error 3004

プロパティ "%1"(%2 **バイト目) は，基本単位が** VariableArray **型でない** Object **型のプロパティであるため，選択項目に指定できません。**

The property "%1" (byte number %2) cannot be specified as a selection item because the basic unit is an Object-type property that is not a Variable array property.

パラメタの内容

- %1：プロパティ名
- %2：%1 のプロパティ名が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

VariableArray 型プロパティ以外の Object 型のプロパティ（オブジェクトリファレンス）を直接選択項目に指定することはできません。
VariableArray 型プロパティ以外の Object 型のプロパティは，oiidstr 関数の引数として選択項目に指定してください。これによって Object 型のプロパティを OIID 文字列に変換した値が取得できます。

Error 3005

プロパティ "%1"(%2 **バイト目) は，特殊なプロパティであるため，文法で規定した位置以外には指定できません。**

The property "%1" (byte number %2) is a special property, so it can be specified at a syntax-defined position only.

パラメタの内容

- % 1：プロパティ名
- %2：%1 のプロパティ名が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

全文検索インデクス用プロパティなどの特殊な意味を持つプロパティは，edmSQL の文法で規定された位置以外には使用できません。
文法で規定されている位置を確認して，正しい位置に指定してください。

Error 3006

プロパティ "%1"(%2 バイト目) の修飾子に対応するクラス名又は相関名が見付かりません。

A class or correlation name is missing for the qualifier of the property "%1" (byte number %2).

パラメタの内容

- %1：プロパティ名
- %2：%1 のプロパティ名が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

プロパティの修飾子として指定したクラス名または相関名が，該当するプロパティが存在するシンボル空間に見つかりません。FROM 句に指定したクラス名や相関名と異なっている可能性があります。
指定したプロパティを修飾するために参照できる FROM 句を確認して，修飾子として指定したクラス名や相関名を修正してください。または，FROM 句に指定したクラス名や相関名を修正してください。

Error 3007

プロパティの修飾子 "%1"(%2 バイト目) に対応するクラス名又は相関名が見付かりません。

A class or correlation name is missing for the qualifier "%1" (byte number %2) of the property.

パラメタの内容

- %1：クラス名または相関名
- %2：%1 のクラス名または相関名が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

プロパティの修飾子（クラス名または相関名）に対応するクラス名または相関名が，FROM 句の指定から見つかりません。
FROM 句に指定したクラス名もしくは相関名，または指定したプロパティ修飾子を確認して，修正してください。

Error 3008

プロパティ "%1"(%2 バイト目) は，修飾子 "%3" に対応したクラスにありません。

The property "%1" (byte number %2) does not exist in the class for the qualifier "%3".

パラメタの内容

- %1：プロパティ名
- %2：%1 のプロパティ名が出現する入力 edmSQL 文の先頭からの位置（バイト数）
- %3：クラス名または相関名

説明

プロパティの修飾子として指定したクラス名または相関名に対応するクラスに，該当するプロパティが定義されていません。
プロパティ名または指定したクラスが不正な可能性があります。

メッセージに修飾子（%3）として空文字列が出力された場合は，入力 edmSQL 文で修飾子を省略している場合です。この場合は，FROM 句に指定したただ一つのクラスに，指定したプロパティが存在しないことになります。
メタ情報を確認して，正しいプロパティ名または正しいクラスを指定してください。必要に応じてプロパティを定義してください。

Error 3009

プロパティの修飾子 "%1"(%2 バイト目) の指定があいまいです。

The specification of the qualifier "%1" (byte number %2) of the property is ambiguous.

パラメタの内容

- %1：クラス名または相関名
- %2：%1 のクラス名または相関名が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

プロパティのクラスを明示するための修飾子（クラス名または相関名）があいまいで，FROM 句に指定されたクラスを特定できません。
FROM 句に指定したクラス，またはプロパティに指定した修飾子を確認して，修正してください。

（2） VariableArray 型プロパティの要素参照，フィールド参照に関するエラー

Error 3101

VariableArray **型プロパティ "%1"(%2 バイト目) の要素に，プロパティ "%3" はありません。**

The property "%3" is not an element of a Variable array property "%1" (byte number %2).

パラメタの内容

- %1：VariableArray 型プロパティのプロパティ名
- %2：%1 のプロパティ名が出現する入力 edmSQL 文の先頭からの位置（バイト数）
- %3：フィールド参照を指定した VariableArray 型プロパティの要素のプロパティ名

説明

VariableArray 型プロパティの要素に対してフィールド参照を指定した場合に，その要素を定義するクラスにフィールド参照で指定したプロパティが存在しません。
フィールド参照をしたプロパティ名を確認して，正しく修正してください。必要に応じてメタ情報を修正してください。

Error 3102

VariableArray **型プロパティ "%1"(%2 バイト目) の要素のプロパティ "%3" は，永続プロパティでないため，** edmSQL **文に指定できません。**

The property "%3" that is an element of the Variable array property "%1" (byte number %2) cannot be specified for the edmSQL statement because it is not a persistent property.

パラメタの内容

- %1：VariableArray 型プロパティのプロパティ名
- %2：%1 のプロパティ名が出現する入力 edmSQL 文の先頭からの位置（バイト数）
- %3：フィールド参照を指定した VariableArray 型プロパティの要素のプロパティ名

説明

検索の対象となる VariableArray 型プロパティの要素のプロパティは，永続プロパティ（データベース上に存在するプロパティ）である必要があります。

指定した VariableArray 型プロパティの要素のプロパティが，永続プロパティかどうかを確認してください。

Error 3103

VariableArray 型プロパティ "%1"(%2 バイト目) の要素のプロパティ "%3" は，検索可能 (Searchable) でないため，検索条件に指定できません。

The property "%3" that is an element of the Variable array property "%1" (byte number %2) cannot be specified as a search condition because it is not searchable.

パラメタの内容

- %1：VariableArray 型プロパティのプロパティ名
- %2：%1 のプロパティ名が出現する入力 edmSQL 文の先頭からの位置（バイト数）
- %3：フィールド参照を指定した VariableArray 型プロパティの要素のプロパティ名

説明

VariableArray 型プロパティの要素に対してフィールド参照したプロパティが，検索可能（Searchable）ではありません。このため，検索条件には指定できません。
必要に応じてメタ情報を修正してください。

Error 3104

プロパティ "%1" (%2 バイト目) は，VariableArray 型プロパティでないため，要素参照に指定できません。

The property "%1" (byte number %2) cannot be specified for element reference because it is not a Variable array property.

パラメタの内容

- %1：プロパティ名
- %2：%1 のプロパティ名が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

フィールド参照を指定したプロパティが，VariableArray 型プロパティではありません。フィールド参照は VariableArray 型プロパティだけに指定できます。
プロパティ名を確認して，修正してください。

Error 3105

プロパティ "%1" (%2 バイト目) は，VariableArray 型プロパティであるため，副問い合わせの選択項目に指定できません。

The property "%1" (byte number %2) cannot be specified as a selection item of a sub-query because it is a Variable array property.

パラメタの内容

- %1：VariableArray 型プロパティのプロパティ名
- %2：%1 のプロパティ名が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

副問い合わせの選択項目に，VariableArray 型プロパティは指定できません。

Error 3106

プロパティ "%1"(%2 バイト目) は，VariableArray 型プロパティであるため，検索対象の結合条件に指定できません。

The property "%1" (byte number %2) cannot be specified as a join condition for searching because it is a Variable array property.

パラメタの内容

- %1：VariableArray 型プロパティのプロパティ名
- %2：%1 のプロパティ名が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

ON 条件に VariableArray 型プロパティを指定することはできません。

（3） 関数（ルーチン起動，数値関数，集合関数）に関するエラー

Error 3201

%1 関数 (%2 バイト目) は，使用可能な関数として定義していないため，edmSQL **文に指定できません。**

The function %1 (byte number %2) is not defined as a usable function so it cannot be specified for the edmSQL statement.

パラメタの内容

- %1：関数名
- %2：%1 の関数名が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

edmSQL の文法で規定されていない関数名を指定しました。
指定した関数の関数名を確認して，修正してください。

Error 3202

%1 関数 (%2 バイト目) は，文法で規定した位置以外には指定できません。

The function %1 (byte number %2) can be specified in a syntax-defined position only.

パラメタの内容

- %1：関数名
- %2：%1 の関数名が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

関数を規定された位置以外に指定しました。
指定した関数の属性（選択可能または検索可能）を確認して，選択可能であれば選択項目に指定してください。検索可能であれば検索条件に指定してください。

Error 3203

%1 関数 (%2 バイト目) の引数の数が不正です。

The number of arguments of the function %1 (byte number %2) is invalid.

パラメタの内容

- %1：関数名
- %2：%1 の関数名が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

関数に指定した引数の個数が不正です。
関数に指定できる引数の個数を確認して，修正してください。

Error 3204

%1 関数 (%2 バイト目) の第 %3 引数のデータ型が不正です。

The data type of argument %3 of the function %1 (byte number %2) is invalid.

パラメタの内容

- %1：関数名
- %2：%1 の関数名が出現する入力 edmSQL 文の先頭からの位置（バイト数）
- %3：引数の出現番号（1 以上）

説明

関数で，該当する引数に指定したデータ型が不正です。
関数の引数に指定できるデータ型確認して，修正してください。

Error 3205

%1 関数の第 %2 引数には，文法で規定した特殊なプロパティだけが指定できます。

Only a syntax-defined special property can be specified for argument %2 of the function %1.

パラメタの内容

- %1：関数名
- %2：引数の出現番号（1 以上）

説明

全文検索や概念検索などの関数の，edmSQL の文法で規定した特殊なプロパティだけが指定できる引数に，そのプロパティ以外を指定しています。
引数に指定した内容を確認して，修正してください。

Error 3206

%1 関数 (%2 バイト目) の第 %3 引数には，文法で規定した特殊なプロパティだけが指定できます。

Only a syntax-defined special property can be specified for argument %3 of the function %1 (byte number %2).

パラメタの内容

- %1：関数名
- %2：%1 の関数名が出現する入力 edmSQL 文の先頭からの位置（バイト数）
- %3：引数の出現番号（1 以上）

説明

全文検索や概念検索などの関数の，edmSQL の文法で規定した特殊なプロパティだけが指定できる引数に，そのプロパティ以外を指定しています。
引数に指定した内容を確認して，修正してください。

Error 3207

%1 関数 (%2 バイト目) の第 %3 引数には，文字列定数又は ? パラメタだけが指定できます。

Only character string constant or the ? parameter can be specified for argument %3 of the function %1 (byte number %2).

パラメタの内容

- %1：関数名
- %2：%1 の関数名が出現する入力 edmSQL 文の先頭からの位置（バイト数）
- %3：引数の出現番号（1 以上）

説明

値または？パラメタだけが指定できる関数の引数に，値または？パラメタ以外を指定しています。
引数に指定した内容を確認して，修正してください。

Error 3208

%1 関数 (%2 バイト目) の第 %3 引数には，? パラメタだけが指定できます。

Only the ? parameter can be specified for argument %3 of the function %1 (byte number %2).

パラメタの内容

- %1：関数名
- %2：%1 の関数名が出現する入力 edmSQL 文の先頭からの位置（バイト数）
- %3：引数の出現番号（1 以上）

説明

？パラメタだけが指定できる関数の引数に，？パラメタ以外を指定しています。
引数に指定した内容を確認して，修正してください。

Error 3209

%1 関数の第 %2 引数には，VariableArray **型プロパティ** "%3"(%4 **バイト目) を指定できません。**

The Variable array property "%3" (byte number %4) cannot be specified for argument %2 of the function %1.

パラメタの内容

- %1：関数名
- %2：引数の出現番号（1 以上）
- %3：VariableArray 型プロパティのプロパティ名
- %4：%3 のプロパティ名が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

関数で，該当する引数には指定できない VariableArray 型プロパティを指定しています。
関数の引数に指定できる内容を確認して，修正してください。

Error 3210

%1 関数 (%2 バイト目) の第 %3 引数には，VariableArray **型プロパティの要素を指定できません。**

An element of a Variable array property cannot be specified for argument %3 of the function %1 (byte number %2).

パラメタの内容

- %1：関数名
- %2：%1 の関数名が出現する入力 edmSQL 文の先頭からの位置（バイト数）
- %3：引数の出現番号（1 以上）

説明

関数呼び出しで，該当する引数には指定できない VariableArray 型プロパティの要素のプロパティを指定しています。
関数の引数に指定できる内容を確認して，修正してください。

Error 3211

oiidstr **関数の第 %1 引数には，基本単位が** VariableArray **型でない** Object **型のプロパティだけが指定できます。**

Only the Object-type property whose basic unit is not Variable array property can be specified for argument %1 of the oiidstr function.

パラメタの内容

- %1：引数の出現番号（1 以上）

説明

oiidstr 関数の引数には，VariableArray 型プロパティ以外の Object 型のプロパティだけが指定できます。
指定した引数の内容を確認して，修正してください。

Error 3212

%1 関数 (%2 バイト目) の第 %3 引数には，OIID 文字列だけが指定できます。

Only the OIID character string can be specified for argument %3 of the function %1 (byte number %2).

パラメタの内容

- %1：関数名
- %2：%1 の関数名が出現する入力 edmSQL 文の先頭からの位置（バイト数）
- %3：引数の出現番号（1 以上）

説明

関数で，OIID 文字列だけが指定できる引数に，OIID 文字列以外を指定しています。
引数に指定した内容を確認して，修正してください。

Error 3213

%1 関数 (%2 バイト目) の第 %3 引数の OIID 文字列の長さが不正です。

The length of the OIID character string of argument %3 of the function %1 (byte number %2) is invalid.

パラメタの内容

- %1：関数名
- %2：%1 の関数名が出現する入力 edmSQL 文の先頭からの位置（バイト数）
- %3：引数の出現番号（1 以上）

説明

関数で，OIID 文字列を指定する引数に，不正な長さの OIID 文字列を指定しています。
引数に指定した OIID 文字列を確認して，修正してください。

Error 3214

%1 関数 (%2 バイト目) の第 %3 引数の ? パラメタに，AS< データ型指定 > がありません。

The AS <data-type-specification> is missing in the ? parameter of argument %3 of the function %1 (byte number %2).

パラメタの内容

- %1：関数名
- %2：%1 の関数名が出現する入力 edmSQL 文の先頭からの位置（バイト数）
- %3：引数の出現番号（1 以上）

説明

ルーチンの起動に指定する関数の引数に ? パラメタを指定する場合は，必ず AS< データ型指定 > を指定して，? パラメタのデータ型を明確にしてください。
指定した引数のデータ型を確認して，? パラメタに対応する AS< データ型指定 > を指定してください。

Error 3215

関数の引数の？パラメタ以外に，AS< データ型指定 >(%1 バイト目) があります。

The AS <data-type-specification> (byte number %1) exists in a position other than the ? parameter of the function arguments.

パラメタの内容

- %1：AS< データ型指定 > が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

関数の引数の？パラメタ以外に，AS< データ型指定 > を指定しています。AS< データ型指定 > は，関数の引数の？パラメタに対してだけ指定できます。
AS< データ型指定 > を削除してください。

Error 3216

COUNT 関数 (%1 バイト目) は，アクセス制御が有効であるため，主問い合わせの選択項目に指定できません。

The COUNT function (byte number %1) cannot be specified as a selection item of a main query because of the effect of access control.

パラメタの内容

- %1：関数名が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

アクセス制御が有効な場合，COUNT 関数は指定できません。
COUNT 関数を指定する場合は，アクセス制御を無効にしてください。

Error 3217

%1 関数 (%2 バイト目) は，副問い合わせの選択項目に指定できません。

The function %1 (byte number %2) cannot be specified as a selection item of a sub-query.

パラメタの内容

- %1：関数名
- %2：%1 の関数名が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

副問い合わせ中に指定できない関数を，副問い合わせ中に指定しています。
指定した関数の仕様を確認して，修正してください。

(4) 値式（四則演算，符号，文字列連結），論理演算子に関するエラー

Error 3301

演算子 "%1"(%2 バイト目) は，検索対象の結合条件に指定できません。

The operator "%1" (byte number %2) cannot be specified as a join condition of search targets.

パラメタの内容

- %1：演算子
- %2：%1 の演算子が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

指定した演算子は，ON 条件には指定できません。演算子には ON 条件に指定できないものがあります。

指定した演算子の制限事項を確認して，修正してください。

Error 3302

演算子 "%1"(%2 バイト目) の被演算子のデータ型が不正です。

The operand data type of the operator "%1" (byte number %2) is invalid.

パラメタの内容

- %1：演算子
- %2：%1 の演算子が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

被演算子同士，または被演算子と演算子が要求するデータ型が一致しません。
例えば，文字列と数値の比較を指定した場合などに発生するエラーです。
演算子が要求するデータ型と被演算子のデータ型を確認して，修正してください。

(5) 集合指定子（DISTINCT）に関するエラー

Error 3401

重複排除 "DISTINCT"(%1 バイト目) は，アクセス制御が有効であるため，主問い合わせの SELECT 句に指定できません。

The duplicate exclusion DISTINCT (byte number %1) cannot be specified for main-query SELECT because of the effect of access control.

パラメタの内容

- %1："DISTINCT" が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

アクセス制御が有効な場合の検索は，暗黙にオブジェクトが識別される検索になるため，重複排除は実行できません。
重複排除 "DISTINCT" 指定を削除してください。または，アクセス制御を無効にして，重複排除を指定してください。

Error 3402

VariableArray 型プロパティ "%1"(%2 バイト目) は，重複排除 "DISTINCT" を SELECT 句に指定しているため，選択項目に指定できません。

The Variable array property "%1" (byte number %2) cannot be specified as a selection item because the duplicate exclusion DISTINCT is specified for the SELECT clause.

パラメタの内容

- %1：VariableArray 型プロパティのプロパティ名
- %2：%1 のプロパティ名が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

VariableArray 型プロパティが選択項目に指定されている場合，SELECT 句の集合指定子に重複排除 "DISTINCT" は指定できません。
重複排除 "DISTINCT" は指定しないでください。または，VariableArray 型プロパティを選択項目から削除してください。

Error 3403

extracts 関数 (%1 バイト目) は，重複排除 "DISTINCT" を SELECT 句に指定しているため，選択項目に指定できません。

The extracts function (byte number %1) cannot be specified as a selection item because the duplicate exclusion DISTINCT is specified for the SELECT clause.

パラメタの内容

- %1：extracts 関数が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

extracts 関数は，重複排除できないデータ型の値を返却するため，この返却値に対する重複排除は実行できません。
重複排除 "DISTINCT" の指定を削除してください。

(6) クラス（クラス表現，相関名）に関するエラー

Error 3501

クラス "%1"(%2 バイト目) は，文書空間にないため，検索対象に指定できません。

The class "%1" (byte number %2) cannot be specified to be searched because it is not in the document space.

パラメタの内容

- %1：クラス名
- %2：%1 のクラス名が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

検索対象として指定したクラスが文書空間に存在しません。クラス名が不正な可能性があります。
文書空間に定義しているクラス名を確認して，正しいクラスを指定してください。

Error 3502

クラス "%1"(%2 バイト目) は，文書空間に検索可能 (Searchable) なクラスとして定義していないため，検索対象に指定できません。

The class "%1" (byte number %2) cannot be specified to be searched because it is not defined as a searchable class in the document space.

パラメタの内容

- %1：クラス名
- %2：%1 のクラス名が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

指定したクラスは，文書空間には定義されていますが，検索可能なクラスとして文書空間のスコープに登録されていません。
メタ情報定義を確認して，必要であれば検索可能なクラスとして登録してください。

Error 3503

クラス名又は相関名 "%1"(%2 バイト目) が重複しています。

The class or correlation name "%1" (byte number %2) is duplicated.

パラメタの内容

- %1：重複しているクラス名または相関名
- %2：%1 のクラス名または相関名が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

FROM 句の指定で，クラス名と相関名，または相関名同士を同じ名前で指定しています。または，同一のクラスを結合する場合に，相関名で明確に区別していません。
それぞれの検索対象が明確に区別できるように，相関名を付けてください。

Error 3504

クラス "%1"(%2 **バイト目**) **に指定した相関名が重複しています。**

The correlation name specified for the class "%1" (byte number %2) is duplicated.

パラメタの内容

- %1：クラス名
- %2：%1 のクラス名が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

プロパティのクラスを明示するための修飾子（クラス名または相関名）が，FROM 句の指定から見つかりません。
FROM 句のクラスの指定またはプロパティに指定したプロパティ修飾子を確認して，修正してください。

(7) 結合種別に関するエラー

Error 3601

一つの FROM **句には，二種類以上の結合種別は指定できないため，結合種別** "%1"(%2 **バイト目**) **は指定できません。**

The join type "%1" (byte number %2) cannot be specified because two or more join types cannot be specified for one FROM clause.

パラメタの内容

- %1：結合種別
- %2：%1 の結合種別が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

LEFT OUTER 結合と INNER 結合は，同一の FROM 句に混在することはできません。結合種別は統一してください。

(8) 述語（比較述語，論理述語，In 述語，Like 述語（Like および Xlike），Null 述語，Exist 述語）に関するエラー

Error 3701

論理述語 "%1"(%2 **バイト目**) **は，検索対象の結合条件に指定できません。**

The logical predicate "%1" (byte number %2) cannot be specified as a join condition of a search target.

パラメタの内容

- %1：論理述語
- %2：%1 の論理述語が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

指定した論理述語は，ON 条件には指定できません。論理述語には ON 条件に指定できないものがあります。
論理述語の制限事項を確認して，修正してください。

Error 3702

述語 "%1"(%2 バイト目) は，検索対象の結合条件に指定できません。

The predicate "%1" (byte number %2) cannot be specified as a join condition of a search target.

パラメタの内容

- %1：述語
- %2：%1 の述語が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

指定した述語は，ON 条件には指定できません。述語には ON 条件に指定できないものがあります。
述語の制限事項を確認して，修正してください。

Error 3703

比較述語 "%1"(%2 バイト目) の被演算子のデータ型が不正です。

The data type of an operand of the comparison predicate "%1" (byte number %2) is invalid.

パラメタの内容

- %1：比較述語
- %2：%1 の比較述語が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

被演算子同士，または被演算子と述語が要求するデータ型が一致しません。
述語が要求するデータ型と被演算子のデータ型を確認して，修正してください。

Error 3704

論理述語 "%1"(%2 バイト目) の被演算子のデータ型が不正です。

The data type of an operand of the logical predicate "%1" (byte number %2) is invalid.

パラメタの内容

- %1：論理述語
- %2：%1 の論理述語が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

被演算子同士，または被演算子と述語が要求するデータ型が一致しません。
述語が要求するデータ型と被演算子のデータ型を確認して，修正してください。

Error 3705

述語 "%1"(%2 バイト目) の被演算子のデータ型が不正です。

The data type of an operand of the predicate "%1" (byte number %2) is invalid.

パラメタの内容

- %1：述語
- %2：%1 の述語が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

被演算子同士，または被演算子と述語が要求するデータ型が一致しません。
述語が要求するデータ型と被演算子のデータ型を確認して，修正してください。

Error 3706

一つの In 述語には，同じデータ型の値しか指定できないため，%1 番目の値 (%2 バイト目) は指定できません。

The number %1 value (byte number %2) cannot be specified for one In predicate because only a value of the same data type can be specified.

パラメタの内容

- %1：In 述語の列挙された項目の出現番号
- %2：%1 の項目が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

In 述語で列挙する項目として，異なるデータ型の項目を指定しています。
In 述語では，同じデータ型の項目だけを列挙して指定できます。
指定した項目のデータ型を確認して，修正してください。

Error 3707

Null 述語 (%1 バイト目) の値式には，プロパティだけが指定できます。

Only properties can be specified for the expression of a Null predicate (byte number %1).

パラメタの内容

- %1：Null 述語が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

Null 述語にプロパティ以外の値式を指定しています。Null 述語の値式には，プロパティだけが指定できます。
Null 述語の内容を確認して，修正してください。

Error 3708

比較述語 "%1"(%2 バイト目) の両辺の被演算子には，VariableArray 型プロパティの要素を同時に指定できません。

Elements of a Variable array property cannot be simultaneously specified for the operands on both sides of the comparison predicate "%1" (byte number %2).

パラメタの内容

- %1：比較述語
- %2：%1 の比較述語が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

VariableArray 型プロパティのフィールド参照同士を，比較述語で比較することはできません。

Error 3709

論理述語 "%1"(%2 バイト目) の値式には，論理型を戻り値とする関数だけが指定できます。

Only a function whose return value is a logical type can be specified for the expression of the logical predicate "%1" (byte number %2).

パラメタの内容

- %1：論理述語
- %2：%1 の論理述語が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

論理述語の値式には，論理型を戻り値とする関数以外は指定できません。

Error 3710

アスタリスク（*）(%1 バイト目) は，Exists 述語の副問い合わせ以外の選択項目に指定できません。

An asterisk (byte number %1) can be specified only for a selection item of the sub-query of the Exists predicate.

パラメタの内容

- %1：* が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

選択項目に「*」（アスタリスク）を指定できるのは，Exists 述語の副問い合わせの選択項目だけです。

（9）データ操作（ソート，ソート種別）に関するエラー

Error 3801

ソートキー "%1"(%2 **バイト目**) **に対応する選択項目が見付かりません。**

A selection item is missing for the sort key "%1" (byte number %2).

パラメタの内容

- %1：ソートキー（プロパティ名または数値）
- %2：%1 のソートキーが出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

指定したソートキーに対応する選択項目がありません。
選択項目を確認して修正してください。

Error 3802

ソートキー "%1"(%2 **バイト目**) **に対応した選択項目は，ソート可能** (Orderable) **でないため，ソートできません。**

A selection item for the sort key "%1" (byte number %2) cannot be sorted because it is not orderable.

パラメタの内容

- %1：ソートキー（プロパティ名または数値）
- %2：%1 のソートキーが出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

ソートキーとして指定した選択項目はソートに使用できません。ソートに使用できる選択項目のデータ型は，Integer32 型および String 型だけです。
選択項目のデータ型を確認して，修正してください。必要に応じてメタ情報を修正してください。

（10）グループ操作に関するエラー

Error 3901

アクセス制御が有効であるため，GROUP BY **句** (%1 **バイト目**) **は指定できません。**

The GROUP BY clause (byte number %1) cannot be specified because of the effect of access control.

パラメタの内容

- %1：GROUP BY 句が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

アクセス制御が有効な場合，GROUP BY 句は指定できません。
GROUP BY 句を指定する場合は，アクセス制御を無効にしてください。

Error 3902

GROUP BY **句で指定したプロパティでないため，選択項目** "%1"(%2 **バイト目**) **は指定できません。**

The selection item "%1"(byte number %2) cannot be specified. Because the property isn't specified in GROUP BY clause.

パラメタの内容

- %1：選択項目
- %2：%1 の選択項目が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

GROUP BY 句を用いた問い合わせ時には，選択項目は GROUP BY 句で指定したプロパティに限定されます。
GROUP BY 句で指定したプロパティを，選択項目に指定してください。

Error 3903

プロパティ "%1"(%2 バイト目) は GROUP BY 句では指定できません。

The property "%1" (byte number %2) cannot be specified in GROUP BY clause.

パラメタの内容

- %1：値式
- %2：%1 の値式が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

GROUP BY 句ではオブジェクト型，バイナリ型のプロパティ，特殊なプロパティ，関数を指定できません。
オブジェクト型，バイナリ型のプロパティ，特殊なプロパティ，関数を値式から削除してください。

Error 3904

プロパティ "%1"(%2 バイト目) が重複しています。

The value expression "%1" (byte number %2) is duplicated.

パラメタの内容

- %1：プロパティ
- %2：%1 のプロパティが出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

同一のプロパティを GROUP BY 句に指定することはできません。
GROUP BY 句から重複しているプロパティを削除してください。

Error 3906

述語 "%1"(%2 バイト目) は HAVING 句に直接指定できません。

The predicate "%1" (byte number %2) cannot be specified in HAVING clause.

パラメタの内容

- %1：述語
- %2：%1 の述語が出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

NULL 述語および LIKE 述語は HAVING 句に直接指定できません。
HAVING 句の探索条件から，直接指定している NULL 述語および LIKE 述語を削除してください。

Error 3907

プロパティ "%1"(%2 バイト目) は，HAVING 句に指定できません。

The property "%1" (byte number %2) cannot be specified in HAVING clause.

パラメタの内容

- %1：プロパティ名
- %2：%1 のプロパティが出現する入力 edmSQL 文の先頭からの位置（バイト数）

説明

HAVING 句に指定可能なプロパティは，GROUP BY 句で指定したプロパティ，外側の問い合わせの FROM 句に指定したクラスのプロパティまたは集合関数の引数に指定したプロパティだけです。GROUP BY 句で指定したプロパティ，外側の問い合わせの FROM 句に指定したクラスのプロパティ，または集合関数の引数に指定したプロパティを HAVING 句に指定してください。

(11) 副問い合わせに関するエラー

Error 4001

副問い合わせでは複数の選択項目を指定できないため，選択項目 "%1"(%2 バイト目) は指定できません。

The selection item "%1" (byte number %2) cannot be specified. Because multi selection items cannot be specified in the sub-query.

パラメタの内容

- %1：選択項目
- %2：2 番目の選択項目の edmSQL 文の先頭からの位置（バイト数）

説明

副問い合わせには複数の選択項目を指定できません。副問い合わせに複数の選択項目を指定していないか，見直してください。

付録 B.5　そのほかのエラー

Error 9001

構文解析用のスタックがオーバーフローしました。

The stack for syntax analysis has overflowed.

説明

構文解析用の作業領域が制限値に達しました。
または，制限値に達してはいませんが，データベースの制限値を超えました。
edmSQL 文が複雑過ぎるため，構文が解析できません。単純な構文に変更してください。
構文が複雑になる要素としては，次のようなものが考えられます。

- 副問い合わせのネストが深い
- 演算子の入れ子指定が多い
- 演算順序を指定する括弧の数が多い

複雑な構文の edmSQL 文を指定すると，検索処理性能も低下する可能性があります。
単純な構文にするためには，次のような点を考慮してください。

- 演算順序を考慮して，式をまとめる括弧を外す
- 処理が再帰しないようにする

次に，複雑な edmSQL 文を単純にする例を示します。

複雑な edmSQL 文の例

```
SELECT  myProp_Foo
FROM    myClass
WHERE   (((myProp_Foo1 = 1)
            AND (myProp_Foo2 = 2))
                 AND (myProp_Foo3 = 3))
```

単純な edmSQL 文に置き換えた例

```
SELECT  myProp_Foo
FROM    myClass
WHERE   myProp_Foo1 = 1
          AND  myProp_Foo2 = 2
               AND  myProp_Foo3 = 3
```

このように，不要な括弧を外して構文を単純にすることで，スタックの消費量は少なくできます。

付録 C　HiRDB のシステム共通定義のオペランド pd_max_access_tables の見積もり方法

ここでは，HiRDB のシステム共通定義のオペランドである，pd_max_access_tables の見積もり方法について説明します。

なお，HiRDB のシステム共通定義の詳細については，マニュアル「HiRDB　システム定義」を参照してください。

pd_max_access_tables に指定する値は，次に示す式で求めた値を目安としてください。

＜目安の値＞＝50×N＋M1×L

次に，N，M1 および L に代入する値について説明します。

N：

１トランザクションの間にコールするメソッドの個数です。ただし，次のメソッドおよび関数は除きます。

- 各クラスのコンストラクタメソッドおよびデストラクタメソッド
- CdbrCompound クラスのメソッド
- CdbrCore クラスのメソッド
- CdbrDMA クラスの次に示すメソッド
  GetOIID，GetType，ReleaseObject，SetOIID
- CdbrEqlStatement クラスのメソッド
- CdbrSession::GetClassType メソッド以外の CdbrSession クラスのメソッド
- CdbrVariableArray クラスのメソッド
- dbrDelete 関数

M1：

検索を CdbrEqlStatement クラスの機能によって実現している場合，この値を使用します。

CdbrEqlStatement::Execute メソッドをコールすることによって発行される SQL が参照する HiRDB の表数です。

CdbrEqlStatement::Execute メソッドに指定する edmSQL 文ごとに，次の式で値を求め，その中でいちばん大きな値を M1 としてください。

```
M1＝主問い合せのクラス結合数＋副問い合せのクラス結合数
    ＋SELECT句に指定しているextracts関数の数
    ＋SELECT句に指定した別表のVariableArray型プロパティの数＋10
```

L：

1 トランザクションの間にコールする CdbrEqlStatement::Execute メソッドの数です。

付録 D　用語解説

DocumentBroker で使用する用語について説明します。

（英字）

ACE

Access Control Element の略です。アクセス制御エレメントのことです。

ACFlag

Access Control Flag の略です。アクセス制御フラグのことです。

ACL

Access Control List の略です。アクセス制御リストのことです。

AND-NOT 検索

検索オペレータに「AND-NOT」を使用する検索方法です。二つの検索条件を AND-NOT でつないで，左側のオペランドに指定した検索条件は成立するが，右側のオペランドに指定した検索条件は成立しない文書を検索します。例えば，「著者が『日立太郎』であるが，所属は『日立製作所』ではない文書を検索する」というような場合に使用できます。

AND 検索

検索オペレータに「AND」を使用して，検索条件同士の論理積を求める検索方法です。例えば，「著者が『日立太郎』で，文書中に『コンピュータ』という文字列を含む文書を検索する」というような場合に使用できます。

API（Application Programming Interface）

アプリケーションプログラムとのインターフェースを指します。

CdbrCompound オブジェクト

複合データを表すオブジェクトです。複合データとは，複数の異なる型によって表されるデータです。主に，VariableArray 型のプロパティとして設定されている Object 型の要素の値を参照，設定するときに使用します。

CdbrConfiguratedReferentialContainer オブジェクト

バージョン付き構成管理コンテナに相当するオブジェクトです。CdbrConfiguratedReferentialContainer クラスを基に作成します。

CdbrDocument オブジェクト

（バージョンなし）文書に相当するオブジェクトです。CdbrDocument クラスを基に作成します。CdbrVersionableDocument オブジェクトの個々のバージョンとしても操作できます。

CdbrEqlStatement オブジェクト

SQL に基づいた文法で検索条件が記述できる edmSQL 検索を実行するためのオブジェクトです。CdbrEqlStatement クラスを基に作成します。

CdbrIndependentPersistence オブジェクト

クラスライブラリで提供するクラスの階層構造に属さない，独立データを表すオブジェクトです。CdbrIndependentPersistence クラスを基に作成します。

CdbrPublicACL オブジェクト

パブリック ACL を表すオブジェクトです。CdbrPublicACL クラスを基に作成します。

CdbrReferentialContainer オブジェクト

フォルダや分類（インデクス）を使用した文書管理を実現する，（バージョンなし）コンテナに相当するオブジェクトで

す。CdbrReferentialContainer クラスを基に作成します。

CdbrSession オブジェクト

文書空間とクライアント環境のセッションを確立するオブジェクトです。CdbrSession クラスを基に作成します。

CdbrVariableArray オブジェクト

可変長配列を表すオブジェクトです。主に，VariableArray 型のプロパティの値を参照，設定するときに使用します。

CdbrVersionableDocument オブジェクト

DocumentBroker で扱うバージョン付き文書に相当するオブジェクトです。このオブジェクトによって表される文書は，バージョン管理ができます。CdbrVersionableDocument クラスを基に作成します。

CdbrVersionTraceableContainer オブジェクト

バージョンなし構成管理コンテナに相当するオブジェクトです。CdbrVersionTraceableContainer クラスを基に作成します。CdbrConfiguratedReferentialContainer オブジェクトの個々のバージョンとしても操作できます。

CdbrXmlTranslatorFactory オブジェクト

XML プロパティマッピング機能や XML インデクスデータ作成機能を実行するための操作環境を管理するオブジェクトです。
また，CdbrXmlTranslator オブジェクトを作成します。

CdbrXmlTranslator オブジェクト

XML プロパティマッピング機能および XML インデクスデータ作成機能を実行するオブジェクトです。

Child

直接型のコンテインメント（DirectContainment）の場合，オブジェクト（Parent）に包含されるオブジェクトを指します。

ConfigurationHistory オブジェクト

バージョン管理に使用する最上位の DMA オブジェクトです。文書のバージョンを管理する VersionSeries オブジェクトを管理します。

Containable オブジェクト

コンテナの包含要素になるオブジェクトです。CdbrContainable クラスのサブクラスを基に作成されたオブジェクトを指します。

Containee

参照型のコンテインメント（ReferentialContainment）の場合，オブジェクト（Container）に包含されるオブジェクトを指します。

Container

参照型のコンテインメントの場合，オブジェクト（Containee）を包含するオブジェクトを指します。

ContainerVersion オブジェクト

包含しているオブジェクト全体を一つの概念的なオブジェクトとして管理する DMA オブジェクトです。また，構成管理型のコンテインメントではオブジェクトを包含するオブジェクトとして利用できます。ContainerVersion オブジェクトは，バージョンを保持できるオブジェクトです。したがって，ContainerVersion オブジェクトのバージョンを上げることで，ある時点でのオブジェクトのまとまりを管理できます。

Container オブジェクト

コンテインメントを利用してオブジェクトを管理する場合に，オブジェクトを包含する DMA オブジェクトの一つです。dmaClass_Container クラスおよびユーザが定義した dmaClass_Container クラスのサブクラスを基に作成します。

ContentElement（dmaClass_ContentElement クラス）

文書のコンテントにアクセスするために使われる DMA オブジェクトの抽象クラスです。ContentElement の四つのサブクラス（edmClass_ContentFileLink クラス，dmaClass_ContentTransfer クラス，edmClass_ContentTransfers クラス，および dmaClass_ContentReference クラス）は，コンテントの格納とアクセス機能を提供します。

ContentFileLink オブジェクト

文書のコンテントを管理するために使用する DMA オブジェクトです。File Link 連携機能を使用する場合に使用します。dmaClass_ContentElement クラスのサブクラスである edmClass_ContentFileLink クラスを基に作成します。

ContentReference オブジェクト

文書のコンテントを管理するために使用する DMA オブジェクトです。コンテントをデータベースに格納しないで，位置情報を永続化して管理する場合に使用します。dmaClass_ContentElement クラスのサブクラスである dmaClass_ContentReference クラスを基に作成します。

ContentTransfers オブジェクト

文書のコンテントを管理するために使用する DMA オブジェクトです。一つの文書のコンテントとして複数のファイルをデータベースに格納（永続化）して管理する場合に使用します。dmaClass_ContentElement クラスのサブクラスである edmClass_ ContentTransfers クラスを基に作成します。

ContentTransfer オブジェクト

文書のコンテントを管理するために使用する DMA オブジェクトです。コンテントをデータベースに格納（永続化）して管理する場合に使用します。dmaClass_ContentElement クラスのサブクラスである dmaClass_ContentTransfer クラスを基に作成します。

CORBA（Common Object Request Broker Architecture）

OMG（Object Management Group）が提唱するオブジェクト間の通信メカニズムを提供する ORB（Object Request Broker）の標準アーキテクチャです。

DCD（Document Content Description）

W3C で定義している，タグセットを記述するためのタグセットです。マッピング元 XML タグ定義はこの DCD の形式で記述します。DTD と同じように XML 文書の構造を記述しますが，DTD と異なり，DCD 自体が XML のタグセットとして定義されています。また，タグおよびその属性について，データの型を指定できるといった特長を持ちます。

DMA（Document Management Alliance）

文書管理インターフェースの標準化を図る団体 AIIM（Association for Information and Image Management International）によって定義される共通インターフェースです。

DMA URL

DMA オブジェクトの OIID（Object Instance Identifier）を定義する URL です。

DMA オブジェクト

DMA が規定するオブジェクトモデルに基づいたオブジェクトです。DMA で規定されたクラス（クラス名が dmaClass_ で始まるクラス），DocumentBroker で拡張したクラス（クラス名が edmClass_ で始まるクラス），およびこれらのクラスからユーザが定義したサブクラスを基に作成されます。

DocumentSpace 構成定義ファイル（docspace.ini）

文書空間の構成を定義するために使用するファイルです。

DocVersion オブジェクト

DocumentBroker で扱う文書に相当する DMA オブジェクトです。dmaClass_DocVersion クラスおよびユーザが定義した dmaClass_DocVersion クラスのサブクラスを基に作成します。

edmClass_Relationship クラス

文書間リレーションを表す DMA クラスです。単独では作成できない，文書に従属する Relationship オブジェクトの基となるクラスです。

edmSQL

DocumentBroker のオブジェクトを検索するための検索条件式を表現するための文法です。SQL の文法に基づいています。

edmSQL 検索

検索条件に，SQL ライクの文法で記述できる edmSQL 文を指定して実行する検索のことです。

FAM（File Access Module）

HiRDB File Link で使用するファイルサーバ上で，コンテントを管理するためのプログラムです。

File Link 文書

HiRDB File Link で管理しているファイルサーバに格納されている一つのファイル，または一つのディレクトリおよびそのサブディレクトリ下のすべてのファイルをコンテントとして持つ文書のことです。データベースでは，コンテントへのリンク情報を管理しています。CdbrDocument クラスまたは CdbrVersionableDocument クラスを使用して操作します。

File Link 連携機能

HiRDB File Link と連携してコンテントを管理する機能です。

GUID

Globally Unique Identifier の略です。DMA のクラス，プロパティ，検索オペレータなどに与えるユニークな識別子です。GUID は，「X」を 0 ～ 9 および a ～ f（小文字）で表される 16 進数とした「XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX（8 けた -4 けた -4 けた -4 けた -12 けた）」の形式で表されます。

ID ファイル

オブジェクト操作ツールを使用して複数のオブジェクトを一括して操作する場合，操作対象となるオブジェクトの OIID などを記述するファイルです。

LDAP

Lightweight Directory Access Protocol の略です。TCP/IP 上で動作する解放型 DAP を提供し，X.500 のデータモデルを保持しています。

NOT 条件

指定したキーワードとの不一致を求める検索条件です。例えば，「作成者が『日立』ではない文書を検索する」というような場合に使用できます。

OIID

Object Instance Identifier の略です。文書空間での永続オブジェクトの存在や格納位置などを明確にするために使用する識別子です。OIID は DMA URL として定義されます。

OR 検索

検索オペレータに「OR」を使用して，検索条件同士の論理和を求める検索方法です。例えば，「作成者が『日立太郎』であるオブジェクトか，作成者の所属が『日立製作所』であるオブジェクトを検索する」というような場合に使用できます。

Parent

直接型のコンテインメントの場合，オブジェクト（Child）を包含するオブジェクトを指します。

QueryResultRow オブジェクト（検索結果行オブジェクト）

QueryResultSet オブジェクトを構成する要素の一つ（行）として生成される DMA オブジェクトです。

dmaProp_Selections プロパティで指定されたプロパティを含みます。

QueryResultSet オブジェクト（検索結果集合オブジェクト）

検索結果の集合として生成される DMA オブジェクトです。QueryResultSet オブジェクトは検索条件を満たす 0 または 1 以上の QueryResultRow オブジェクトによって構成されます。

Query オブジェクト（問い合わせ）

検索に使用する問い合わせを表現するオブジェクトツリーの最上位の DMA オブジェクトを指します。

RD エリア

データの格納単位の一つで，1 ～ 16 個の HiRDB ファイルから構成されます。RD エリアには，システム用 RD エリア，ユーザ用 RD エリア，データディクショナリ LOB 用 RD エリア，およびユーザ LOB 用 RD エリアがあります。システム用 RD エリアとは，マスタディレクトリ用 RD エリア，データディクショナリ用 RD エリア，およびデータディレクトリ用 RD エリアの総称です。また，ユーザ用 RD エリアとは，表とインデクスを格納するための RD エリアで，公用 RD エリアと私用 RD エリアがあります。データディクショナリ LOB 用 RD エリアとは，ストアドプロシジャ，ストアドファンクションを使用する場合に，ストアドプロシジャ，ストアドファンクションの定義ソース，およびオブジェクトを格納するための RD エリアです。ユーザ LOB 用 RD エリアとは，ユーザが作成した文書，画像，音声などの長大なデータを格納するための RD エリアで，公用 RD エリアと私用 RD エリアがあります。

Rendition オブジェクト

文書の表現形式（HTML，PDF など）を管理する DMA オブジェクトです。

Reservation オブジェクト

新しいバージョンを作成する権利を VersionSeries オブジェクトに予約する DMA オブジェクトです。

Scalar 型

プロパティの基本単位の一つです。基本単位が Scalar 型であるプロパティは，データ型に従った値を一つだけ持ちます。

SystemManager オブジェクト

System オブジェクトを管理する DMA オブジェクトです。

System オブジェクト

サーバ内の文書空間を管理する DMA オブジェクトです。

UOC（User Own Coding）

ユーザによって作成されたプログラムのことです。DocumentBroker では，ユーザ認証や，アクセス制御機能のためのユーザ情報取得に UOC を使用できます。

VariableArray 型

プロパティの基本単位の一つです。基本単位が VariableArray 型であるプロパティは，データ型に従った複数の値を可変長な一次元配列として持ちます。また配列の要素は，構造体で管理できます。

Versionable オブジェクト

バージョン管理の対象になるオブジェクトです。CdbrVersionable クラスのサブクラスを基に作成されたオブジェクトを指します。

VersionDescription オブジェクト

文書のバージョン（更新履歴）を管理するために使用する DMA オブジェクトです。VersionSeries オブジェクトと Versionable オブジェクトを接続するために使用します。

VersionSeries オブジェクト

文書のバージョン（更新履歴）を管理するために使用する DMA オブジェクトです。連続的な履歴を持つバージョンの

構成を保持するオブジェクトです。

VersionTracedComponentDocVersion オブジェクト

直接型，参照型および構成管理型のコンテインメントでコンテナに包含される対象となるためのプロパティを持つ DMA オブジェクトです。dmaClass_DocVersion クラスのサブクラスの edmClass_VersionTracedComponentDocVersion クラスを基に作成します。

VersionTracedDocVersion オブジェクト

直接型，参照型および構成管理型のコンテインメントで，コンテナに包含される対象となるためのプロパティを持つ DMA オブジェクトです。dmaClass_DocVersion クラスのサブクラスの edmClass_VersionTracedDocVersion クラスを基に作成します。

VTContainee

構成管理型のコンテインメントの場合，オブジェクト（VTContainer）に包含されるオブジェクトを指します。

VTContainer

構成管理型のコンテインメントの場合，オブジェクト（VTContainee）を包含するオブジェクトを指します。

W3C（World Wide Web Consortium）

HTML や XML など，WWW に関する技術の標準化を推進する非営利団体です。

XML インデクスデータ作成機能

XML 文書の文書ファイルからプレーンテキスト形式または構造指定検索用の全文検索インデクスデータを作成する機能です。XML 形式のファイルの構文解析もあわせて実行します。

XML プロパティマッピング機能

XML 形式のファイルを構文解析して，DocumentBroker サーバのプロパティマッピング定義に従って XML 形式のファイル内に記述されているタグ間の文字列または属性値を，XML 文書のプロパティに割り当てて設定する機能です。

（ア行）

アクセス権

オブジェクトを作成したり，すでに作成されているオブジェクトにアクセスしたりする権利です。

アクセス制御エレメント（ACE：Access Control Element）

アクセス制御リスト（ACL）の要素です。一つのサブジェクトと一つのパーミッションの組で構成され，指定されたサブジェクトに対して指定されたパーミッションの範囲のアクセス権を与えることを示す情報です。

アクセス制御機能

DocumentBroker の文書空間でのオブジェクトの作成や，管理されている文書やコンテナなどのオブジェクトに対する操作を，ユーザやグループごとに許可または制限する機能です。

アクセス制御機能付き検索

アクセス制御機能を利用した文書空間で検索を実行した場合に，ユーザにアクセス権がないオブジェクトは検索結果として取得しない検索です。

アクセス制御情報

アクセス制御されている文書空間で，ユーザがメソッドを発行する際に，アクセス権の判定に使用される情報です。

アクセス制御情報変更権

オブジェクトに設定されているアクセス制御情報（ACFlag および ACL）を変更する権利です。また，パブリック ACL をアクセス制御対象オブジェクトにバインドすることを許可する権利も含みます。なお，パブリック ACL のアクセス情報変更権には，パブリック ACL のユーザ定義プロパティを変更する権利を含みます。

アクセス制御フラグ（ACFlag：Access Control Flag）

オブジェクトの所有者，プライマリグループおよび全ユーザという区分でパーミッションを設定できるアクセス制御情報の一つです。

アクセス制御モデル

アクセス制御機能を利用して運用されている文書管理モデルです。

アクセス制御リスト（ACL：Access Control List）

任意のユーザまたはグループにパーミッションを設定できるアクセス制御情報の一つです。アクセス制御エレメント（ACE）のリストで構成されます。

アドレス呼び出し

文書空間内に構成されている独立した永続オブジェクトに付けられた ID（OIID）を利用して，そのオブジェクトを探索することです。

異表記展開検索

全文検索条件として指定する検索タームまたは検索タームの異表記を含む文書を検索する方法です。例えば，検索タームとして「バイオリン」を指定した場合に，「ヴァイオリン」という検索タームの異表記を含む文書も検索できます。

インデクス情報ファイル

ユーザが追加するプロパティにインデクスを定義する場合に，定義するインデクスの情報を記述するファイルです。

永続オブジェクト

データベースに格納されたオブジェクトを指します。

永続プロパティ

データベースに存在するプロパティを指します。

オブジェクト作成権

オブジェクト作成権限で，文書空間にオブジェクトを作成する権利を与えるパーミッションです。ユーザ権限定義ファイルで使用します。

オブジェクト作成権限

文書空間にオブジェクトを作成する権限で，ユーザ権限の一つです。ユーザ権限定義ファイルで定義します。オブジェクト作成権限を与えられたユーザおよびグループに属するユーザは，オブジェクトを作成するメソッド（CreateObject）を実行できます。

オブジェクト操作権限

文書空間内のすべてのオブジェクトを，与えられた権限の範囲で操作する権利で，ユーザ権限の一つです。ユーザ権限定義ファイルで定義します。例えば，オブジェクト操作権限としてプロパティ参照権を与えられたユーザおよびグループに属するユーザは，文書空間内のすべてのオブジェクトのプロパティを参照できます。

オブジェクト操作ツール

DocumentBroker が提供するクラスライブラリを使用したクライアントアプリケーションです。クラスライブラリを使用してユーザがクライアントアプリケーションを開発する場合に，必要であると思われる機能をコマンド群として提供します。

オブジェクトリファレンス

DMA オブジェクトへのリファレンスを示す Object 型プロパティの値です。例えば，dmaProp_ParentContainer プロパティの値がこれに当たります。

（カ行）

概念検索

ユーザが任意に指定した文章や文字列を手がかりにして，その条件と似た概念を持つ文書を検索する方法です。全文検索の一種です。概念検索で指定する条件のことを種文章といいます。

仮のバージョン識別子

チェックアウト中の仮のバージョンを識別するための識別子です。
チェックアウト中のオブジェクトを参照・更新する時に使用します。この識別子はチェックアウト時に DocumentBroker によって設定される識別子であり，仮のバージョンに該当するオブジェクトの OIID とは異なります。

関連オブジェクト

コンテインメントの型（直接型，間接型および構成管理型）を定義するための DMA オブジェクトです。
直接型のコンテインメントの場合，オブジェクトを包含するオブジェクト（Parent）とオブジェクトに包含されるオブジェクト（Child）を DirectContainmentRelationship オブジェクトを使用して関連づけます。参照型のコンテインメントの場合，オブジェクトを包含するオブジェクト（Container）とオブジェクトに包含されるオブジェクト（Containee）を ReferentialContainmentRelationship オブジェクトを使用して関連づけます。
構成管理型のコンテインメントの場合，オブジェクトを包含するオブジェクト（VTContainer）とオブジェクトに包含されるオブジェクト（VTContainee）を VersionTraceableContainmentRelationship オブジェクトを使用して関連づけます。

基本コンテント更新権

基本パーミッションの一つで，オブジェクトのコンテントを更新する権利を与えるパーミッションです。基本プロパティ参照権を含みます。文書に対して設定する場合は，全文検索インデクスを作成，削除する権利を含みます。

基本コンテント参照権

基本パーミッションの一つで，オブジェクトのコンテントを参照する権利を与えるパーミッションです。基本プロパティ参照権を含みます。文書に対して設定する場合は，全文検索を実行する権利を含みます。

基本削除権

基本パーミッションの一つで，オブジェクトを削除する権利を与えるパーミッションです。基本プロパティ参照権を含みます。

基本単位

DMA で規定されているプロパティは，データ型に従った値を 1 個持つか複数個持つかが決められています。これを基本単位といいます。

基本バージョン管理権

基本パーミッションの一つで，バージョン管理されているオブジェクトのバージョンを追加，削除する権利を与えるパーミッションです。基本プロパティ参照権を含みます。

基本パーミッション

ユーザおよびグループがオブジェクトに対して実行できる操作の範囲を定めるパーミッションの基本単位です。オブジェクト操作権限，アクセス制御フラグおよびアクセス制御エレメントで，ユーザおよびグループに許可する操作の範囲を定めるときに使用します。例えば，あるユーザに対して，文書の更新と削除を許可する場合は，更新と削除を許可するために，基本コンテント更新権と基本削除権という二つのパーミッションを設定します（一つのパーミッションで一つの権利を与える）。基本パーミッションには，基本プロパティ参照権，基本プロパティ更新権，基本コンテント参照権，基本コンテント更新権，基本リンク権，基本バージョン管理権および基本削除権があります。

基本プロパティ更新権

基本パーミッションの一つで，オブジェクトのプロパティを更新する権利を与えるパーミッションです。基本プロパティ参照権を含みます。

基本プロパティ参照権

基本パーミッションの一つで，オブジェクトのプロパティを参照する権利を与えるパーミッションです。そのほかのすべての基本パーミッションに含まれます。コンテナに対して設定する場合は，関連オブジェクトのユーザ定義プロパティの参照と管理されている要素を参照する権利を含みます。

基本リンク権

基本パーミッションの一つで，コンテインメントの設定，変更および関連オブジェクトのユーザ定義プロパティを変更する権利を与えるパーミッションです。基本プロパティ参照権を含みます。

近傍条件検索

検索オペレータに「Prox」を使用する拡張検索（全文検索）です。指定する検索タームが同時に存在する場合に，その検索ターム間の距離を条件として検索します。例えば，「『文書管理』という検索タームと『ドキュメント』という検索タームを含み，これらの検索タームがこのとおりの順序で出現し，かつ検索ターム間に入る文字が 5 文字以内である文書を検索する」というような場合に使用できます。

組み合わせパーミッション

基本パーミッションを複数組み合わせた権利を与えるパーミッションの単位です。アクセス制御フラグおよびアクセス制御エレメントでユーザおよびグループに許可する操作の範囲を定めるときに使用します。組み合わせパーミッションには，プロパティ参照権，参照権，プロパティ更新権，参照更新権，削除権，リンク権，バージョン管理権およびフルコントロールがあります。例えば，あるユーザに対して，ある文書の参照更新権という組み合わせパーミッションを設定すると，そのユーザは基本プロパティ参照権，基本プロパティ更新権，基本コンテント参照権および基本コンテント更新権が設定されたのと同じ範囲の操作を，その文書に対して実行できます。

クラス定義情報ファイル

DocumentBroker サーバで定義されている DMA オブジェクトのクラスまたはそのサブクラスのクラス名またはプロパティ名から GUID，データ型，プロパティの基本単位などの情報を取得するために使用するファイルです。
EDMCrtSimMeta コマンドでも作成できます。オブジェクト操作ツールを使用する場合に必要です。

継承（Inheritance）

既存のクラスを利用して新しいクラスを定義するオブジェクト指向の技術です。

検索結果構造体

検索結果が格納される構造体です。DMA オブジェクトの QueryResultRow オブジェクトの列挙によって構成されています。

構成管理型のコンテインメント（VersionTraceableContainment）

コンテナを利用した文書管理の方法です。構成管理型のコンテインメントで使われるコンテナは，バージョンを作成できるオブジェクトの構成を管理します。

構成管理コンテナ

構成管理型のコンテインメントで使われるコンテナです。
包含されるオブジェクトのバージョンを固定して管理したり，常に最新のバージョンを追跡して管理したりすることで，包含されるオブジェクト全体の構成を管理するために使用するコンテナです。CdbrVersionTraceableContainer に相当します。DMA オブジェクトでは，ContainerVersion オブジェクトに相当します。

構造指定検索

XML 文書を管理している場合に，文書の論理構造（エレメント）やエレメントの属性をキーとして検索する方法です。例えば，「タイトルに『コンピュータ』という単語が含まれ，章に『XML』という単語が含まれる文書を検索する」というような場合に使用します。また，「『document』というエレメントに設定されている属性『status』の属性値が『public』である文書を検索する」というような場合にも使用できます。

コンテインメント（包含）

コンテナを使用したオブジェクトの包含関係を示します。コンテインメントには直接型（DirectContainment），参照型（ReferentialContainment）および構成管理型（VersionTraceableContainment）の 3 種類があります。

コンテナ
コンテインメントで，オブジェクトを包含できるオブジェクトの総称です。CdbrReferentialContainer オブジェクト，CdbrConfiguratedReferentialContainer オブジェクトおよび CdbrVersionTraceableContainer オブジェクトがこれに当たります。また，DMA オブジェクトとしては，Container オブジェクト，VersionTraceableContainer オブジェクトおよび ContainerVersion オブジェクトがあります。

コンテント
一般的には，属性に対する文書のデータ部分を指します。DMA では，ある文書に関連づけられていて，DMA で規定されている Content モデルに従ってアクセスされるオブジェクトの集合を指します。また，アクセスされるオブジェクトの実体（例えば，report.doc，document.htm など）をコンテントデータといいます。

コンテント格納先パス
リファレンスファイル管理機能を使用する場合に，コンテント格納先ベースパスを基点とする相対パスのことです。

コンテント格納先ベースパス
リファレンスファイル管理機能を使用する場合に，コンテント格納先の基点となるディレクトリパスのことです。

コンテント種別
レンディションのコンテントが，シングルファイル文書のコンテントか，マルチファイル文書のコンテントか，リファレンスファイル文書のコンテントか，または File Link 文書のコンテントかを示します。

コンテントロケーション
リファレンスファイル管理機能を使用する場合に，コンテントの格納先を示す情報のことです。

（サ行）

削除権
組み合わせパーミッションの一つです。基本削除権と同じ操作を許可するパーミッションです。

サブクラス
あるクラスから派生するクラスのことです。または，それ自身がサブクラスとして参照されているクラスのことです。

サブジェクト（Subject）
アクセス権を与えるユーザまたはグループです。

サブジェクト種別（SubjectType）
アクセス権を与えるサブジェクトが，ユーザなのか，グループなのかまたはシステムなのかを識別するための情報です。

サブレンディション
マルチレンディション文書に，追加登録されたレンディションのことです。マスタレンディション以外のレンディションを指します。なお，サブレンディションは，登録後にマスタレンディションに変更できます。

参照型のコンテインメント（ReferentialContainment）
コンテナを利用した文書管理の方法です。参照型のコンテインメントで使われるコンテナは，文書にはり付けるインデクスの働きをします。

参照権
組み合わせパーミッションの一つです。基本コンテント参照権と同じ操作を許可するパーミッションです。

参照更新権
組み合わせパーミッションの一つです。基本プロパティ参照権，基本プロパティ更新権，基本コンテント参照権および基本コンテント更新権を組み合わせたパーミッションです。すなわち，参照更新権を設定することで，プロパティを参照，更新する権利とコンテントを参照，更新する権利を設定できます。

システム管理者
DocumentBroker の運用管理者です。

実行環境制御ファイル
オブジェクト操作ツールの実行環境を定義するファイルです。

状態フラグ
マルチレンディション文書の，マスタレンディションに対するサブレンディションのコンテントの状態を表すフラグです。マスタレンディションとサブレンディションのコンテントの状態が一致している，マスタレンディションのコンテントが更新されたのに対してサブレンディションのコンテントが更新されていない，またはサブレンディションのコンテントが存在しない，という 3 種類の状態が表されます。dbrProp_RenditionStatus プロパティの下位 2 バイトに設定されます。

所有者（Owner）
オブジェクトの所有者として設定されているユーザです。アクセス制御フラグでパーミッションを与えられます。所有者に設定されているユーザは，そのオブジェクトのアクセス制御フラグで所有者に与えられたパーミッションの範囲の操作をそのオブジェクトに対して実行できます。また，そのオブジェクトの所有者およびセキュリティ ACL の値を変更できます。

シングルファイル文書
データベースに登録されている一つのファイルをコンテントとして持つ文書のことです。CdbrDocument クラスまたは CdbrVersionableDocument クラスを使用して操作します。

スーパークラス
あるクラスのクラス定義に使われたクラスを，派生したクラスのスーパークラスといいます。

セキュリティ ACL
オブジェクトに設定されたアクセス制御情報へのアクセスを制御するためのアクセス制御リストです。任意のユーザまたはグループにアクセス制御情報変更権を設定できます。

セキュリティ運用者
DocumentBroker のアクセス制御の運用情報の管理者です。セキュリティ定義ファイルを保守します。

セキュリティ管理者
アクセス制御機能を利用した文書空間で，アクセス判定を受けることなく，すべてのオブジェクトに自由にアクセスする特権を持ち，文書空間のすべてのオブジェクトを保守するユーザです。セキュリティ定義ファイルに定義します。

セキュリティ定義ファイル
アクセス制御の運用情報を定義するファイルです。セキュリティ管理者，ユーザ権限定義ファイル名およびオブジェクト作成時に，アクセス制御フラグにデフォルトで設定されるパーミッションを定義します。

全文検索
文書中に含まれるキーワードを条件（全文検索条件）として，キーワードを含む文書を検索する方法です。

全文検索インデクス
全文検索の対象になるテキストデータに対応するプロパティです。edmProp_TextIndex プロパティ，edmProp_StIndex プロパティ，edmProp_ConceptTextIndex プロパティ，edmProp_ConceptStIndex プロパティおよび edmProp_Content プロパティに相当します。

全文検索機能付き文書クラス
全文検索の対象となる文書を作成するための dmaClass_DocVersion クラスのサブクラスです。メタ情報ファイルを新規に作成したり，既存のメタ情報ファイルを編集したりして，全文検索のためのプロパティを追加したユーザ定義のクラスです。

（タ行）

チェックアウト（check-out）

文書またはコンテナにバージョンを追加するために，次バージョンの追加を予約して，最新バージョンのコピーを要求することです。

DMA オブジェクトでは，VersionSeries オブジェクトに新規 DocVersion オブジェクト（文書）を作成するための権利を予約して，新規 DocVersion オブジェクトの追加対象となる VersionSeries オブジェクトで管理している最新の DocVersion オブジェクトのコピーを要求することです。

チェックイン（check-in）

文書またはコンテナのバージョンの追加を確定することです。

DMA オブジェクトでは，VersionSeries オブジェクトに，新規 DocVersion オブジェクト（文書）を追加することです。クライアントは特定の VersionSeries オブジェクトに対して，Reservation オブジェクトを保持しておく必要があります。

直接型のコンテインメント（DirectContainment）

コンテナを利用した文書管理の方法です。直接型のコンテインメントで使われるコンテナは，文書を格納するフォルダの働きをします。

定義情報ファイル

サブクラスおよびプロパティを追加するときに，追加するオブジェクトの定義情報を記述するファイルです。

ディレクトリサービス

ネットワーク上にあるユーザや組織の情報などの資源とその属性を記憶し，検索できるようにしたシステムです。

DocumentBroker では，Active Directory や Sun Java System Directory Server などの製品を使用した LDAP 対応のディレクトリサービスと連携できます。

LDAP 対応のディレクトリサービスとして使用できる製品の詳細については，マニュアル「DocumentBroker Version 3 システム導入・運用ガイド」を参照してください。

同義語展開検索

全文検索条件として指定する検索タームまたは検索タームの同義語を含む文書を検索する方法です。例えば，検索タームとして「パソコン」を指定した場合に，「電子計算機」，「パーソナルコンピュータ」，「PC」など，検索タームと同じ意味を持つ単語を含む文書も検索できます。

動作環境メタ情報ファイル

DocumentBroker を起動するときに参照するメタ情報です。実行環境ディレクトリ /etc/meta_files に格納されます。

独立データ

CdbrIndependentPersistence オブジェクトに相当します。ほかのオブジェクトに依存しない，独立したオブジェクトです。

特権

アクセス制御機能を利用した文書空間で，アクセス判定を受けることなく，すべてのオブジェクトに自由にアクセスする権利です。セキュリティ定義ファイルにセキュリティ管理者として定義されたユーザに与えられます。特権の有無は，ログイン時にセキュリティ定義ファイルが参照され，ログインユーザごとに作成されるユーザ情報に保持されます。

トップオブジェクト

クラスライブラリのオブジェクトを構成する複数の DMA オブジェクトのうち，最上位に位置するオブジェクトです。例えば，CdbrDocument オブジェクトの場合は，DMA オブジェクトの DocVersion オブジェクトがトップオブジェクトです。

（ナ行）

ナビゲーション

ある独立した永続オブジェクトから，別の関連する独立した永続オブジェクトに対する横断的な参照関係を指します。

（ハ行）

バージョン管理権

組み合わせパーミッションの一つです。基本プロパティ参照権，基本プロパティ更新権，基本コンテント参照権，基本コンテント更新権および基本バージョン管理権を組み合わせたパーミッションです。

バージョン付き構成管理コンテナ

バージョンを保持できる構成管理コンテナです。CdbrConfiguratedReferentialContainer オブジェクトに相当します。DMA オブジェクトでは，ConfigurationHistory オブジェクトに相当します。

バージョンなし構成管理コンテナ

バージョンを保持しない構成管理コンテナです。CdbrVersionTraceableContainer オブジェクトに相当します。DMA オブジェクトでは，ContainerVersion オブジェクトに相当します。

パーミッション

オブジェクトの作成，オブジェクトのプロパティ参照，オブジェクトのコンテント更新などの実行可能な操作の範囲を表す値です。オブジェクト作成権限を与えるパーミッション，オブジェクトの操作の範囲を定めるパーミッションがあります。オブジェクトの操作の範囲を定めるパーミッションには，基本パーミッションと組み合わせパーミッションがあります。

パブリック ACL

文書空間にオブジェクトとして存在するアクセス制御リスト（ACL）です。複数のオブジェクトが共有できます。

ファイルサーバ

HiRDB File Link で構築したファイルサーバを指します。

フィルタリング定義ファイル（TFD）

Preprocessing Library for Text Search の正規化機能によって構造指定検索用データ（ESIS-B データ）から不要なタグを除去したり，不要なタグ間のテキストをタグごと除去したりする場合に必要な定義ファイルです。

プライマリグループ

アクセス制御フラグ（ACFlag）でパーミッションを与えるグループです。

フルコントロール

組み合わせパーミッションの一つです。すべての基本パーミッションを組み合わせたパーミッションです。オブジェクトに対するすべての操作を許可します。

プロパティ更新権

組み合わせパーミッションの一つです。基本プロパティ更新権と同じ操作を許可するパーミッションです。

プロパティ参照権

組み合わせパーミッションの一つです。基本プロパティ参照権と同じ操作を許可するパーミッションです。

プロパティマッピング定義ファイル（DPM）

XML 文書中のタグ名とその内容をマッピングするクラス名，およびプロパティ名の対応関係の定義（プロパティマッピング定義）を記述したファイルです。この定義ファイルを基に，XML 定義ファイルの追加／更新／削除コマンド（EDMXmlMap）によってマッピング定義ファイルを生成します。

プロパティマッピング定義は，登録時にマッピング定義名を付けて登録します。プロパティマッピングの実行時には，

このマッピング定義名を指定して使用するプロパティマッピング定義を選択します。

文書

CdbrVersionableDocument オブジェクトおよび CdbrDocument オブジェクトに相当します。
CdbrVersionableDocument オブジェクトを，「バージョン付き文書」と呼びます。CdbrDocument オブジェクトを，「バージョンなし文書」と呼びます。
DMA オブジェクトでは，dmaClass_DocVersion クラスおよびそのサブクラスを基に作成するオブジェクトのことです。

文書空間

DMA オブジェクトモデルを実装するリポジトリです。

変換フラグ

マルチレンディション文書の，サブレンディションのコンテントを，レンディション変換の対象にするかどうかを表すフラグです。DocumentBroker Rendering Option を使用してレンディション変換を実行する場合に使用します。また，DocumentBroker Rendering Option によるレンディション変換でエラーが発生した場合には，エラーを示すフラグとしても使われます。dbrProp_RenditionStatus プロパティの上位 2 バイトに設定されます。

（マ行）

マスタレンディション

マルチレンディション文書に，最初に登録されたレンディションのことです。マルチレンディション文書を参照・更新するときには，レンディション形式を指定しますが，レンディション形式を指定しない場合は，マスタレンディションが対象になります。なお，マスタレンディションとして扱うレンディションは，登録後に変更できます。

マッピングセット定義ファイル（XMS）

HiRDB Adapter for XML が使用する，登録されたマッピング定義の一覧を管理するファイルです。

マッピング定義ファイル（XMP）

HiRDB Adapter for XML が使用できる文法に変換したマッピング定義が記述されているファイルです。ユーザ作成のプロパティマッピング定義ファイル（DPM）とマッピング元 XML タグ定義ファイル（DCD）から生成します。

マッピング元 XML タグ定義ファイル（DCD）

登録する XML 文書の構造を DCD の形式で記述したファイルです。

マルチファイル管理機能

一つの文書に，複数のファイルを登録して管理する機能です。

マルチファイル文書

データベースに登録されている複数のファイルをコンテントとして持つ文書のことです。CdbrDocument クラスまたは CdbrVersionableDocument クラスを使用して操作します。

マルチレンディション機能

一つの文書に，同一内容の複数の異なる形式のコンテントを登録する機能です。

マルチレンディション文書

複数のレンディションを登録している文書のことです。一つの同じ内容を表す複数の形式のコンテントを保持する文書です。CdbrDocument クラスまたは CdbrVersionableDocument クラスを使用して操作します。

メタ情報ファイル

DocumentBroker が利用する DMA のクラス，プロパティ，検索オペレータなどの詳細情報を定義したファイルです。クラスおよびプロパティを追加，変更する場合に使用できます。

メタデータ（metadata）
クラス，プロパティおよびオペレータに関する詳細情報を定義するデータです。

メタデータ空間
単一の継承関係を形成するクラスの集合のことです。

（ヤ行）

ユーザ権限
文書空間にオブジェクトを作成する権利（オブジェクト作成権限）と，文書空間内のすべてのオブジェクトに対する操作の範囲（オブジェクト操作権限）をユーザまたはグループ単位で定めるアクセス制御情報の一つです。ユーザ権限定義ファイルに定義します。ユーザ権限の内容は，ログイン時にユーザ権限定義ファイルが参照され，ログインユーザごとに作成されるユーザ情報に保持されます。

ユーザ権限定義ファイル
ユーザ権限（オブジェクト作成権限およびオブジェクト操作権限）を定義するためのファイルです。

ユーザ情報
ログインユーザのユーザ識別子，所属グループ，特権およびユーザ権限を表す情報です。ログイン時にユーザごとに生成され，アクセス権の判定に使用されます。

（ラ行）

ランキング検索
全文検索条件に対する適応度をスコアとして算出して，スコアを基に検索結果集合の要素（文書）をソートして出力する検索です。

リファレンスファイル管理機能
DocumentBroker サーバが存在するマシンから接続可能なファイルシステムの任意のディレクトリで文書のコンテントを管理し，文書のプロパティおよびコンテントの格納先の情報をデータベースで管理する機能です。

リファレンスファイル文書
DocumentBroker サーバが存在するマシンから接続可能なファイルシステムの任意のディレクトリに格納されているファイルをコンテントとして持つ文書のことです。データベースでは，文書のプロパティおよびコンテントの格納先の情報を管理しています。CdbrDocument クラスまたは CdbrVersionableDocument クラスを使用して操作します。

リレーション
DMA オブジェクトの Relationship オブジェクトに相当します。文書と文書の間の参照関係を表すオブジェクトです。

リレーション先文書
文書間リレーションを設定される，参照先になる文書です。

リレーション識別子
文書間に設定されたリレーションを識別するための識別子です。
リレーションを削除したり，リレーションのプロパティを更新したりする時に使用します。この識別子は，リレーションを設定した時に DocumentBroker によって設定される識別子です。同じ文書から同じ文書に対して二つのリレーションを設定した場合は，それぞれ異なるリレーション識別子が設定されます。

リレーション元文書
文書間リレーションを設定する元になる文書です。

リンク
DMA オブジェクトの DirectContainmentRelationship オブジェクト，ReferentialContainmentRelationship オブジェ

クトまたは VersionTraceableContainmentRelationship オブジェクトによって表される，文書とコンテナの関連づけです。

リンク権

組み合わせパーミッションの一つです。基本リンク権と同じ操作を許可するパーミッションです。

リンク識別子

コンテナオブジェクトへのリンク（関連づけ）を識別するための識別子です。
リンクを解除したり，リンクのプロパティの参照または更新したり，構成管理モードを変更したりする時に使用します。
この識別子は関連づけをした時に DocumentBroker によって設定される識別子です。同じコンテナオブジェクトに，同じ要素を 2 度関連づけした場合は，それぞれ異なるリンク識別子が設定されます。

レンディション

文書のコンテントの形式およびそのコンテントをあわせてレンディションと呼びます。DMA オブジェクトの Rendition オブジェクトおよび ContentTransfer オブジェクトに相当します。

レンディションタイプ

Word などのアプリケーションで編集したファイル，HTML 形式のファイル，GIF などの画像データのファイルのように，登録した文書のコンテントのファイルの形式を表す文字列です。レンディションごとに設定できます。
DocumentBroker では，レンディションタイプとして，MIME 名を指定することを推奨しています。

レンディションのコンテント種別変換機能

レンディションのコンテントの格納先を，最適な格納先へ変換することができる機能です。

レンディション変換

マルチレンディション文書の，マスタレンディションのコンテントの文書形式を変換して，サブレンディションのコンテントを作成，登録することです。

ローカル ACL

オブジェクトごとに設定できるアクセス制御リスト（ACL）です。VariableArray 型のプロパティとして設定されます。

ロケール（Locale）

言語や使用する文字コードの種別，特定の国や地域で特別な意味を持つ属性などの定義のことです。ロケールは地域化した形でアプリケーションの拡張性を提供するために使われます。

索引

記号

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

U

V

W

X

あ

い

え

お

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

つ

て

と

な

は

ひ

ふ

へ

ほ

ま

め

も

ゆ

よ

ら

り

る

れ

ろ