Solaris WBEM SDK 開発ガイド

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 816–3984–10
2002 年 5 月

原典: *Solaris WBEM SDK Developer's Guide*

Part No:806-6828-10

Revision A

020319@2851

目次

はじめに **13**

1 Solaris WBEM の概要 19
WBEM について 19
CIM について 20
Solaris WBEM Services について 20
CIM Object Manager 21
MOF コンパイラ 21
Solaris スキーマ 21
Solaris WBEM SDK 22
CIM Workshop を使用した WBEM アプリケーションの開発 23
CIM Workshop マニュアル 24
CIM Workshop の実行 24

2 MOF コンパイラを使用した JavaBeans の作成 27
MOF コンパイラについて 27
`mofcomp` を使用した JavaBeans の生成 28
CIM を Java にマップする方法 29
JavaBeans の生成の例 33

3 クライアントプログラムの記述 47
概要 47
クライアントアプリケーションの処理手順 47
クライアント接続の開始と終了 48
名前空間について 48

クライアント接続の開始　49
クライアント接続の終了　50
基本的なクライアント操作の実行　50
インスタンスの作成　51
インスタンスの削除　51
インスタンスの取得と設定　53
プロパティの取得と設定　54
オブジェクトの列挙　55
関連の作成　60
メソッドの呼び出し　64
クラス定義の取得　65
例外の処理　66
名前空間の作成　67
名前空間の削除　68
基底クラスの作成　68
クラスの削除　68
アクセス制御の設定　70
`Solaris_UserAcl` クラス　70
`Solaris_NamespaceAcl` クラス　72
修飾子と修飾子のデータ型の処理　73
CIM 修飾子の取得と設定　73
クライアント要求のバッチ処理　74
CIM イベントの処理　76
インジケーションについて　77
予約について　79
CIM リスナーの追加　80
イベントフィルタの作成　80
イベントハンドラの作成　82
イベントフィルタとイベントハンドラのバインド　83
ログメッセージの読み取りと書き込み　84
ログファイルについて　84

4　プロバイダプログラムの作成　**91**
プロバイダについて　91
プロバイダのデータソース　92
プロバイダの種類　93
プロバイダインタフェースの実装　95

インスタンスプロバイダの作成　95
メソッドプロバイダの作成　98
アソシエータプロバイダの作成　99
インジケーションプロバイダの作成　101
ネイティブプロバイダの作成　104
プロバイダの作成　105
▼ プロバイダの CLASSPATH を設定する方法　105
▼ プロバイダを登録する方法　106

5　WBEM 照会の作成　109
WQL について　109
照会の記述　110
WQL キーワード　110
照会の構文解析　113
SELECT リスト　113
FROM 句　114
WHERE 句　114
照会を処理するプロバイダの記述　115

6　Solaris WBEM SDK サンプルプログラムの使用　117
サンプルプログラムについて　117
サンプルアプレット　118
▼ アプレットビューアを使用してサンプルアプレットを実行する方法　118
▼ Web ブラウザでサンプルアプレットを実行する方法　118
サンプルクライアントプログラム　119
サンプルクライアントプログラムの実行　120
サンプルプロバイダプログラム　121
▼ サンプルプロバイダプログラムの実行方法　121

A　WBEM のエラーメッセージ　123
WBEM のエラーメッセージ　123
エラーメッセージの構成　123
エラーメッセージの一覧　124

B　Solaris スキーマ　139
Solaris スキーマファイル　139

Solaris_Schema1.0.mof ファイル　141
Solaris_CIMOM1.0.mof ファイル　141
Solaris_Core1.0.mof ファイル　142
Solaris_Application1.0.mof ファイル　142
Solaris_System1.0.mof ファイル　143
Solaris_Device1.0.mof ファイル　144
Solaris_Acl1.0.mof ファイル　144
Solaris_Network1.0.mof ファイル　145
Solaris_Users1.0.mof ファイル　145
Solaris_Event1.0.mof ファイル　145
Solaris_SNMP1.0.mof ファイル　145
Solaris_VM1.0.mof ファイル　146
Solaris_Project1.0.mof ファイル　147
Solaris_Performance1.0.mof ファイル　147

索引　**149**

表目次

表 1–1 Solaris WBEM API 22
表 2–1 MOF ファイル要素 29
表 2–2 CIM 要素を Java 要素にマップする方法 29
表 2–3 CIM データ型を Java データ型にマップする方法 30
表 2–4 メタ修飾子 31
表 2–5 標準修飾子 31
表 2–6 MOF 要素を Java 要素にマップする方法 33
表 3–1 オブジェクトの列挙 56
表 3–2 関連メソッド 61
表 3–3 `TeacherStudent` メソッド 62
表 3–4 `invokeMethod` パラメータ 64
表 3–5 `CIM_Indication` クラス構造 77
表 3–6 `CIM_IndicationFilter` プロパティ 81
表 3–7 `CIM_IndicationHandler` プロパティ 82
表 3–8 ログメッセージの要素 85
表 4–1 プロバイダの種類 93
表 4–2 `EventProvider` メソッド 102
表 5–1 SQL の概念と WQL の対応 109
表 5–2 サポートされる WQL キーワード 110
表 5–3 `SELECT` 文の例 111
表 5–4 `WHERE` 句で使用できる WQL 演算子 113
表 6–1 サンプルクライアントプログラム 119
表 6–2 サンプルプロバイダプログラム 121
表 B–1 Solaris スキーマファイル 140

図目次

図 3–1　TeacherStudent 関連 1　60
図 3–2　TeacherStudent 関連 2　61

例目次

例 2–1 JavaBeans の生成 33
例 3–1 ルートアカウントへの接続 49
例 3–2 ユーザーアカウントへの接続 49
例 3–3 RBAC の役割 ID の認証 50
例 3–4 クライアント接続の終了 50
例 3–5 インスタンスの作成 51
例 3–6 インスタンスの削除 52
例 3–7 インスタンスの取得と設定 54
例 3–8 プロパティの取得 54
例 3–9 プロパティの設定 55
例 3–10 列挙クラス 56
例 3–11 クラスおよびインスタンスの列挙 57
例 3–12 クラス名の列挙 59
例 3–13 名前空間の列挙 59
例 3–14 インスタンスの引き渡し 64
例 3–15 メソッドの呼び出し 65
例 3–16 クラス定義の取得 66
例 3–17 名前空間の作成 67
例 3–18 クラスの削除 69
例 3–19 CIM 修飾子の設定 73
例 3–20 バッチ処理の例 75
例 3–21 CIM リスナーの追加 80
例 3–22 イベントフィルタの作成 82
例 3–23 イベントハンドラの作成 83
例 3–24 イベントフィルタとイベントハンドラのバインド 83
例 3–25 `Solaris_LogEntry` のインスタンス作成 85

例 3–26 ログ記録リストの表示 87
例 4–1 CIMInstance プロバイダ 96
例 4–2 メソッドプロバイダ 98
例 4–3 CIMAssociator プロバイダ 100
例 4–4 プロバイダの登録 107
例 5–1 照会を処理するプロバイダ 115
例 A–1 エラーメッセージの構成 123

はじめに

『*Solaris WBEM SDK* 開発ガイド』では、Solaris™ WBEM (Web-Based Enterprise Management) ソフトウェア開発キット (SDK) について説明します。このツールキットを使うことによってソフトウェア開発者は、Solaris™ オペレーティング環境のリソースを管理する標準規格のアプリケーションを開発することができます。また、プロバイダ (管理リソースと通信してデータにアクセスするプログラム) を作成することもできます。

Solaris WBEM SDK には、 Distributed Management Task Force (DMTF) に基づく Common Information Model (CIM) のリソースを記述および管理するためのアプリケーションプログラミングインタフェース (API)、および管理リソースの動的データの取得や設定を行うプロバイダ API が含まれています。また Solaris WBEM SDK には、システム上の管理リソースを作成および監視する Java アプリケーションの CIM Workshop と、サンプル WBEM クライアントおよびプロバイダプログラムも含まれています。

対象読者

このマニュアルは、次のようなソフトウェア開発者を対象としています。

- システム設計開発者 – ソフトウェアプロバイダを介して、標準のCIM 形式のデバイス情報を CIM Object Manager に伝えます。
- システムとネットワークアプリケーションの開発者 – CIM のクラスとインスタンスに格納される情報を管理するアプリケーションを開発します。また、Solaris WBEM Services API を使用して CIM インスタンスやクラスのプロパティの取得や設定を行います。

お読みになる前に

このマニュアルは、読者に次の知識があることを前提としています。

- オブジェクト指向プログラミングの概念
- Java プログラミング
- CIM の概念

知識が不十分な場合には、次の書籍を参考にすることをお勧めします。

- 『*Java:How to Program*』、H. M. Deitel 、P. J. Deitel 著、Prentice Hall 発行、ISBN 0–13–263401–5.
- 『*The Java Class Libraries*』第 2 版、第 1 巻、Patrick Chan、Rosanna Lee、Douglas Kramer、Addison-Wesley 著、ISBN 0–201–31002–3
- 『CIM Tutorial』、Distributed Management Task Force (DMTF) 提供

次の Web サイトは、WBEM 技術に関連する有用なサイトです。

- DMTF –このサイトには、CIM の最新の開発情報、各種の作業グループについての情報、CIM スキーマの拡張方法などが掲載されています。
- Rational Software – このサイトでは、Unified Modeling Language (UML) の関連文書を入手できます。

内容の紹介

第 1 章 では、WBEM、CIM、Solaris WBEM SDK API、および CIM Workshop について紹介します。

第 2 章 では、MOF コンパイラの使用方法について説明します。

第 3 章 では、クライアント API を使用してクライアントプログラムを記述する方法について説明します。

第 4 章 では、プロバイダ API を使用してプロバイダプログラムを記述する方法について説明します。

第 5 章 では、照会を記述および処理するために Query API および WBEM Query Language (WQL) を使用する方法について説明します。

第 6 章 では、Solaris WBEM SDK に付属するサンプルプログラムについて説明します。

付録 A では、Solaris WBEM SDK のコンポーネントによって生成されるエラーメッセージについて説明します。

付録 B では、Solaris WBEM SDK に含まれる MOF ファイルについて説明します。

関連マニュアル

次の関連マニュアルも参照してください。

- 『*Solaris WBEM Services* の管理』 – Solaris オペレーティング環境で WBEM サービスを管理する方法について説明しています。
- Javadoc™ リファレンスページ – WBEM API について説明しています。`/usr/sadm/lib/wbem/doc/index.html` を参照してください。
- CIM/Solaris スキーマ – CIM および Solaris スキーマについて説明しています。`/usr/sadm/lib/wbem/doc/mofhtml/index.html` を参照してください。
- DMTF 用語集 – CIM および WBEM 関連用語の総合的な用語集です。`http://www.dmtf.org/education/cimtutorial/reference/glossary.php` を参照してください。

Sun のオンラインマニュアル

docs.sun.com では、Sun が提供しているオンラインマニュアルを参照することができます。マニュアルのタイトルや特定の主題などをキーワードとして、検索を行うこともできます。URL は、`http://docs.sun.com` です。

表記上の規則

このマニュアルでは、次のような字体や記号を特別な意味を持つものとして使用します。

表 **P–1** 表記上の規則

字体または記号	意味	例
AaBbCc123	コマンド名、ファイル名、ディレクトリ名、画面上のコンピュータ出力、コード例を示します。	.login ファイルを編集します。 ls -a を使用してすべてのファイルを表示します。 system%
AaBbCc123	ユーザーが入力する文字を、画面上のコンピュータ出力と区別して示します。	system% **su** password:
AaBbCc123	変数を示します。実際に使用する特定の名前または値で置き換えます。	ファイルを削除するには、rm *filename* と入力します。
『 』	参照する書名を示します。	『コードマネージャ・ユーザーズガイド』を参照してください。
「 」	参照する章、節、ボタンやメニュー名、強調する単語を示します。	第 5 章「衝突の回避」を参照してください。 この操作ができるのは、「スーパーユーザー」だけです。
\	枠で囲まれたコード例で、テキストがページ行幅を超える場合に、継続を示します。	sun% **grep '^#define ** **XV_VERSION_STRING'**

コード例は次のように表示されます。

- C シェル

  machine_name% **command y|n** [*filename*]

- C シェルのスーパーユーザー

  machine_name# **command y|n** [*filename*]

- Bourne シェルおよび Korn シェル

  $ **command y|n** [*filename*]

- Bourne シェルおよび Korn シェルのスーパーユーザー

  # **command y|n** [*filename*]

[ ] は省略可能な項目を示します。上記の例は、*filename* は省略してもよいことを示しています。

| は区切り文字 (セパレータ) です。この文字で分割されている引数のうち 1 つだけを指定します。

キーボードのキー名は英文で、頭文字を大文字で示します (例: Shift キーを押します)。ただし、キーボードによっては Enter キーが Return キーの動作をします。

ダッシュ (-) は 2 つのキーを同時に押すことを示します。たとえば、Ctrl-D は Control キーを押したまま D キーを押すことを意味します。

第 1 章

Solaris WBEM の概要

この章では、Solaris WBEM (Web-Based Enterprise Management) の概要について説明します。内容は次のとおりです。

- 19 ページの「WBEM について」
- 20 ページの「CIM について」
- 20 ページの「Solaris WBEM Services について」
- 23 ページの「CIM Workshop を使用した WBEM アプリケーションの開発」

注 – この章では、WBEM および CIM の一般的な概要について説明します。WBEM および CIM の詳細については、DMTF の Web サイト (`http://www.dmtf.org`) を参照してください。

WBEM について

WBEM (Web-Based Enterprise Management) は、管理技術およびインターネット技術を組み合わせたもので、企業のコンピューティング環境を統合します。WBEM を使用すると、最新の Web 技術を活用した、標準規格の管理ツールの統合的なセットを提供できます。 WBEM 方式で管理アプリケーションを開発することにより、連携して動作する製品を低コストで作成できます。

コンピュータ業界とテレコミュニケーション業界の企業を代表するグループの 1 つである Distributed Management Task Force (DMTF) は、デスクトップ環境、企業規模のシステム、およびインターネットを管理するための標準規格の開発と普及において主導的な立場にあります。DMTF の目的は、さまざまなプラットフォームおよびプロトコルに渡ってコンピュータおよびネットワークを管理する統合的な手法を開発し、費用効率の高い相互運用性に優れた製品を提供することにあります。

CIM について

DMTF によって開発された CIM (Common Information Model) は、システムとネットワークの管理に使用される業界標準規格です。CIM は、ネットワーク環境の各部の分類と定義を行い、それらの統合方法を表現するための概念的な共通のフレームワークを提供します。CIM の概念は、技術の実装には依存せず、あらゆる管理領域に適用できます。

CIM は、次の要素で構成されます。

- CIM 仕様 – ほかの管理モデルとの統合に使用される言語および手法を定義します。
- CIM スキーマ – システム、アプリケーション、LAN、およびデバイスの実際のモデルの説明を提供します。CIM スキーマは、次の要素で構成されます。
  - コアモデル – 管理環境の基本となる一般的な前提事項を提供する。コアモデルは、クラスと関連のサブセットで構成される。このセットにより、管理システムを分析および説明する基本的な用語を提供する。
  - 共通モデル – 特定の技術や実装に依存せず、特定の管理領域に共通する概念を表す。管理アプリケーションの開発基盤を提供する。
- 拡張スキーマ – 共通モデルで使用される技術およびプラットフォーム固有の拡張を表します。拡張スキーマは、オペレーティングシステムなどの環境に固有のものです。たとえば、Solaris スキーマは拡張スキーマです。ベンダーは、オブジェクトのサブクラスを作成することにより製品のモデルを拡張します。次にアプリケーションは、標準モデルのオブジェクトインスタンスを表示して異機種システム混在環境の異なる製品を管理します。

Solaris WBEM Services について

Solaris WBEM Services は、WBEM および CIM 標準を Solaris に実装したものです。Solaris WBEM Services には、次のコンポーネントが含まれます。

- 21 ページの「CIM Object Manager」
- 21 ページの「MOF コンパイラ」
- 21 ページの「Solaris スキーマ」
- 22 ページの「Solaris WBEM SDK」

CIM Object Manager

CIM Object Manager は、WBEM 対応システムの CIM オブジェクトを管理します。WBEM クライアントアプリケーションが CIM オブジェクトの情報にアクセスすると、CIM Object Manager は、そのオブジェクトの適切なプロバイダまたは CIM Object Manager Repository のいずれかに接続します。WBEM クライアントアプリケーションが CIM Object Manager Repository では利用できない管理リソースのデータを要求した場合、CIM Object Manager はその要求をその管理リソースのプロバイダに転送します。プロバイダは、動的にその情報を取得します。

WBEM クライアントアプリケーションは、CIM Object Manager との接続を確立し、CIM クラスの作成または CIM インスタンスの更新などの WBEM 操作を実行します。WBEM クライアントアプリケーションが CIM Object Manager に接続すると、WBEM クライアントは CIM Object Manager に対する参照を取得します。WBEM クライアントは、取得した参照を使ってサービスを要求したり操作を実行したりします。

MOF コンパイラ

MOF (Managed Object Format) は、CIM スキーマを指定する言語です。管理者は、ASCII テキストを使用してクラスおよびインスタンスを定義してファイルに保存し、 MOF コンパイラ (`mofcomp(1M)`) に送ります。MOF コンパイラによって、ファイルの構文解析が行われ、ファイルに定義されたクラスおよびインスタンスが CIM Object Manager Repository に追加されます。MOF コンパイラを使用して MOF ファイルから自動的に JavaBeans を生成する手順については、第 2 章を参照してください。

MOF は、Java に変換できるので、MOF で開発されたアプリケーションは、Java をサポートするすべてのシステムあるいは環境で動作します。

注 – MOF 言語、ファイル、および構文についての詳細は `http://www.dmtf.org/education/cimtutorial/extend/spec.php` を参照してください。

Solaris スキーマ

Solaris スキーマは、共通モデルの拡張スキーマで、特に Solaris オペレーティング環境で実行される管理オブジェクトを記述するためのものです。

Solaris WBEM Services をインストールすると、CIM スキーマと Solaris スキーマを形成する MOF ファイルがディレクトリ `/usr/sadm/mof` に置かれます。これらのファイルは、CIM Object Manager の起動時に自動的にコンパイルされ実行されます。

ファイル名の中に CIM_ 接頭辞を含む CIM スキーマファイルが、標準の CIM オブジェクトになります。Solaris スキーマは、標準の CIM スキーマを拡張し、Solaris オブジェクトを記述しています。Solaris スキーマを構成する MOF ファイルのファイル名には、Solaris_ 接頭辞が含まれます。

注 – CIM スキーマおよび Solaris スキーマに関するドキュメントは /usr/sadm/lib/wbem/doc/mofhtml/index.html にインストールされます。

Solaris WBEM SDK

Solaris WBEM SDK は、API のセットで、管理アプリケーションを作成するために必須のコンポーネントが含まれています。管理アプリケーションによって、XML および HTTP 通信規格を使用して WBEM 対応の管理デバイスと通信することができます。

Solaris WBEM アプリケーションは、WBEM API を介して CIM Object Manager から情報およびサービスを要求します。これらの API では、CIM オブジェクトが Java クラスとして記述されます。プログラマは、これらのインタフェースを使用して管理対象オブジェクトを記述したり、特定のシステム環境内の管理対象オブジェクトの情報を取得したりすることができます。CIM を使用して管理対象オブジェクトをモデル化する場合の利点は、CIM に準拠するシステム間でそれらのオブジェクトを共有できることです。

注 – Solaris WBEM API のマニュアルは、Solaris のインストール時に Javadoc 形式で /usr/sadm/lib/wbem/doc/index.html にインストールされます。

Solaris WBEM API については、次の表で説明します。

表 **1–1** Solaris WBEM API

API	パッケージ名	説明
CIM	javax.wbem.cim	基本的な CIM 要素を表す共通クラスおよびメソッドを含む。CIM API は、オブジェクトをローカルシステムに作成する

表 1–1 Solaris WBEM API　（続き）

API	パッケージ名	説明
クライアント 第 3 章を参照	`java.wbem.client`	アプリケーションは `CIMClient` クラスを使用して CIM Object Manager に接続する。CIM Object Manager とのデータ転送には、ほかのクラスおよびメソッドを使用する バッチ処理可能な API (クライアント API のサブセット) を新たに使用すると、クライアントは複数の要求を 1 回のリモートコールでバッチ処理できる。これにより、複数のリモートメッセージ交換による遅延を短縮できる
プロバイダ 第 4 章を参照	`java.wbem.provider`	CIM Object Manager は、これらの API を使用して動的データのアプリケーション要求をプロバイダに渡す
照会 第 5 章を参照。	`java.wbem.query`	WQL を使って照会を表現したり処理したりするクラスおよびメソッドを含む

CIM Workshop を使用した WBEM アプリケーションの開発

Solaris WBEM SDK に含まれる GUI ベースの開発ツールである CIM Workshop を使用すると、WBEM アプリケーションを開発できます。CIM Workshop を使うと、次の作業を実行できます。

- クラスの表示、追加、削除、および検索
- 名前空間の表示、追加、および削除
- 新しいクラスへのプロパティ、修飾子、およびメソッドの追加
- インスタンスの作成
- インスタンス値の変更
- 関連の表示
- イベントの予約
- メソッドの実行

注 – CIM スキーマクラスまたは Solaris スキーマクラスの既存のプロパティ、メソッド、または修飾子の変更は、CIM ガイドラインによって禁止されています。また、継承されたプロパティ、メソッド、および修飾子の値は変更できません。

CIM Workshop マニュアル

CIM Workshop では、メインウィンドウ以外のすべてのダイアログボックスにコンテキストヘルプが表示されます。ダイアログボックスにあるテキスト入力フィールド、タブ、およびラジオボタンなどのインタフェースコンポーネントをクリックすると、ダイアログボックスの左側の情報区画に適切なヘルプが表示されます。

ヒント – 情報区画を閉じたり再表示したりするには、ダイアログボックスの左上のクエスチョンマーク (?) ボタンをクリックします。

CIM Workshop の実行

デフォルトでは、CIM Workshop は、Remote Method Invocation (RMI) プロトコルを使用して、ローカルホスト (デフォルトの名前空間は `root\cimv2`) 上の CIM Object Manager に接続します。CIM Object Manager を実行するリモートホストを指定することもできます。

▼ CIM Workshop を起動する方法

1. システムプロンプトで次のコマンドを入力します。

   ```
   % /usr/sadm/bin/cimWorkshop
   ```

   「ログイン (CIM Workshop Login) 」ダイアログボックスが表示されます。

2. コンテキストヘルプの指示に従って、「ログイン **(CIM Workshop Login)** 」ダイアログボックスのフィールドに必要な情報を入力します。「了解 **(OK)**」をクリックします。

   CIM Workshop メインウィンドウが表示されます。

▼ CIM Workshop を終了する方法

- **CIM Workshop** メインウィンドウで「**Workshop**」→「終了 **(Exit)**」を選択します。

  CIM Workshop が終了します。

メインウィンドウについて

CIM Workshop メインウィンドウは、次の 3 つの区画で構成されます。

- 左側の区画 – 現在の名前空間のクラス継承ツリーが表示される。
- 右側の区画 – 「プロパティ (Properties)」、「メソッド (Methods)」、および「イベント (Events)」タブが表示される。左側の区画のクラスを選択して、右側区画のタブをクリックすると、選択したクラスのプロパティ、メソッド、またはイベントの詳細情報を表示できる。

- 下部の区画 – 予約したイベントの発生を知らせる通知が表示される。

第 2 章

MOF コンパイラを使用した JavaBeans の作成

ここでは、MOF (Managed Object Format) コンパイラの概要および `mofcomp` コマンドで `-j` オプションを指定して JavaBeans を作成する方法について説明します。内容は次のとおりです。

- 27 ページの「MOF コンパイラについて」
- 29 ページの「CIM を Java にマップする方法」
- 33 ページの「JavaBeans の生成の例」

注 – MOF コンパイラの詳細については、`mofcomp(1M)` のマニュアルページを参照してください。

MOF コンパイラについて

MOF は、DMTF によって開発されたコンパイル言語です。MOF は、CIM および WBEM の静的および動的クラスとインスタンスを定義します。Solaris WBEM Services に含まれる CIM および Solaris MOF を使用でき、独自の MOF を作成することもできます。DMTF の MOF 言語を使用して独自の MOF を作成する方法の詳細については、DMTF Web サイト (`http://www.dmtf.org`) を参照してください。

MOF コンパイラ `mofcomp(1M)` は、MOF ファイルを構文解析し、クラスおよびインスタンスを Java クラスに変換します。そのあと、クラスをデフォルトの `root\cimv2` または他の指定された名前空間に追加します。MOF ファイルは簡単に Java に変換できるので、Java ベースのアプリケーションでは、Java 仮想マシン (JVM) を実行するコンピュータ上の MOF ファイルに含まれるデータを解釈および交換することができます。

Solaris のインストール時に MOF コンパイラにより、CIM スキーマおよび Solaris スキーマを記述する MOF ファイルが自動的にコンパイルされ、それらのファイルが CIM Object Manager Repository に追加されます。

mofcomp を使用した JavaBeans の生成

WBEM、JavaBeans™ または Bean のコンテキストで CIM クラスおよびデータ要素にアクセスし、操作するためのメソッドおよびインスタンスを定義します。mofcomp コマンドで -j オプションを指定すると、MOF ファイルで定義された CIM クラスを表す JavaBeans が自動的に生成されるので、開発作業を簡略化できます。これらの自動生成された JavaBeans によってインタフェースが定義されます。実装コードを追加するかどうかは、開発者によって異なります。

注 – プログラムが、基盤となる JavaBeans 実装に対して行った変更の影響を受けないようにするため、元の JavaBeans の代わりにインタフェースを使用します。

---

mofcomp コマンドに -j オプションを指定すると、CIMBean.java というベース Java インタフェースと CIMBeanImpl.java というインタフェースを実装するベース Bean が生成されます。CIMBeanImpl.java には、生成された Bean に共通するすべてのコードが含まれます。生成されたすべての Java インタフェースは CIMBean.java から継承され、生成されたすべての Bean は CIMBeanImpl.java から継承されるため、結果的にベース実装から継承されることになります。

MOF ファイルに定義されている各 CIM クラス用に MOF コンパイラの JavaBeans 生成機能は、次の要素で構成される Java インタフェースを生成します。

- MOF ファイルで定義されているプロパティのアクセスおよび変更用メソッド
- MOF ファイルで定義されている invokeMethods と同等のメソッド

Java インタフェースには、*CIMClass*Bean.java という名前が付けられます。これらの Java インタフェースを実装した Bean クラス名は、 *CIMClass*BeanImpl.java です。また CIM DisplayName、Units、および Version 修飾子を含むプロパティのアクセス用メソッドも生成されます。

CIM クラスの OUT 修飾子パラメータを含む invokeMethod 用に、メソッドの呼び出し生成出力を保持するコンテナインタフェースが生成されます。これらのインタフェースには、*CIMClass_ MethodName*Output.java という名前が付けられます。この *CIMClass_MethodName*Output.java コンテナインタフェースのインスタンスは、Bean メソッドの最後のパラメータとして必要です。Bean メソッドでパラメータとして使用される 1 つまたは複数の Object データ型は可変データ型ではないため、このコンテナインタフェースが必要になります。そのため、入力および出力データの両方を保持するためには使用できません。

MOF ファイル要素

-j オプションを利用するには、MOF ファイルに PACKAGE 要素を含める必要があります。次の形式で IMPORTS および EXCEPTIONS 要素を指定できます。

PACKAGE=*NameOfJavaPackage*
IMPORTS=*JavaClassName1* : *JavaClassName2* : ...
EXCEPTIONS=*Exception1* : *Exception2* : ...

次の表では、これらの要素について説明します。

表 **2–1** MOF ファイル要素

要素	説明
PACKAGE	必須。MOF コンパイラによって生成されたソースファイルを含む Java パッケージ名を指定する
IMPORTS	任意。生成されたソースファイルにインポートする Java クラス名をコロン (:) で区切って指定する。必要な数の Java クラスを必要なだけの行にわたって指定できる
EXCEPTIONS	任意。生成されたソースファイルに含める Java 例外名をコロン (:) で区切って指定する。必要な数の Java クラス例外を必要なだけの行にわたって指定できる 注 – EXCEPTIONS を指定する場合、IMPORTS を指定する必要がある

CIM を Java にマップする方法

次の表では、CIM 要素を Java 要素にマップする方法を説明します。

表 **2–2** CIM 要素を Java 要素にマップする方法

CIM 要素	**Java** 要素
クラス	生成された Java ソースファイル名のベースとして、CIM クラス名が使用される。生成された Java クラスは、MOF のクラスとサブクラスの関係に定義された継承に従う
プロパティ	各 CIM プロパティにアクセス用メソッドおよび変更メソッドが作成される。関連するアクセス用メソッドおよび変更メソッドのベースとして、CIM プロパティ名が使用される
メソッド	各 CIM メソッドに対応する Java メソッドが作成される。関連した Java メソッド名のベースとして CIM メソッド名が使用される。戻り値は同様の Java データ型にマップされる。入力および出力パラメータが Java メソッドの引数として使用される。出力パラメータは、直接メソッドシグニチャに含まれないが、出力コンテナオブジェクトにカプセル化されて、メソッドパラメータとして含められる
修飾子	修飾子については、表 2–4 および 表 2–5 を参照
関連	特定の要素は要求されていない

表 **2–2** CIM 要素を Java 要素にマップする方法　*(続き)*

CIM 要素	**Java 要素**
インジケーション	特定の要素は要求されていない
参照方法	それぞれの CIM 参照に対して、生成された Java インタフェースへの参照が作成される
トリガー	特定の要素は要求されていない
スキーマ	特定の要素は要求されていない

次の表では、CIM データ型を Java データ型にマップする方法を説明します。

表 **2–3** CIM データ型を Java データ型にマップする方法

CIM データ型	**Java** データ型	アクセス用メソッド	変更メソッド
`uint8 X`	`UnsignedInt8`	`UnsignedInt8 getX();`	`void setX (UnsignedInt8 x);`
`sint8 X`	`Byte`	`Byte getX();`	`void setX(Byte x);`
`uint16 X`	`UnsignedInt16`	`UnsignedInt16 getX() ;`	`void setX (UnsignedInt16 x);`
`sint16 X`	`Short`	`Short getX();`	`void setX(Short x);`
`uint32 X`	`UnsignedInt32`	`UnsignedInt32 getX() ;`	`void setX (UnsignedInt32 x);`
`sint32 X`	`Integer`	`Integer getX();`	`void setX(Integer x) ;`
`uint64 X`	`UnsignedInt64`	`UnsignedInt64 getX() ;`	`void setX (UnsignedInt64 x);`
`sint64 X`	`Long`	`Long getX();`	`void setX(Long x);`
`String X`	`String`	`String getX();`	`void setX(String x);`
`Boolean X`	`Boolean`	`Boolean isX();`	`void setX(Boolean x) ;`
`real32 X`	`Float`	`Float getX();`	`void setX(Float x);`
`real64 X`	`Double`	`Double getX();`	`void setX(Double x);`
`DateTime X`	`CIMDateTime`	`CIMDateTime getX();`	`void setX (CIMDateTime x);`
`Reference X`	`CIMObjectPath`	`CIMObjectPath getX() ;`	`void setX (CIMObjectPath x);`

表 **2–3** CIM データ型を Java データ型にマップする方法　(続き)

CIM データ型	**Java** データ型	アクセス用メソッド	変更メソッド
`char16 X`	`Character`	`Character getX();`	`void setX(Character x);`

次の表にモデルのメタ構造体の定義を改良するメタ修飾子のリストを示します。これらの修飾子は、互いに排他的で、MOF 構文のオブジェクトクラスまたはプロパティ宣言の実際の使用方法を改良します。

表 **2–4** メタ修飾子

修飾子	スコープ	型	意味
関連	クラス	Boolean	マップに影響しない
インジケーション	クラス	Boolean	クラスは abstract (抽象) クラス

次の表では、標準修飾子と CIM オブジェクトを Bean にマップする際に与える影響について説明します。オプションの修飾子はサポートされていません。Javadoc は、このマップに基づく各インタフェースおよびクラスで作成されます。

表 **2–5** 標準修飾子

修飾子	スコープ	意味
`ABSTRACT`	クラス、関連、インジケーション	クラスは abstract クラスなので、Java インタフェースに影響を与えない
`DESCRIPTION`	任意	提供される情報によって、ソースファイルに Javadoc コメントが生成される
`DISPLAYNAME`	プロパティ	表示名のアクセス用メソッドが作成される `public String displayNameFor` *`Property`*`();`
`IN`	パラメータ	メソッドシグニチャーを決定する
`OUT`	パラメータ	メソッドパラメータシグニチャーおよび戻り値を決定する
`TERMINAL`	クラス	クラスまたはインタフェースは、final

表 **2–5** 標準修飾子　(続き)

修飾子	スコープ	意味
UNITS	プロパティ、メソッド、パラメータ	ほかのアクセス用メソッドが作成される。 `public String get` *property*`Units();`
VALUMAP	プロパティ、メソッド、パラメータ	Bean には、CIM ValueMap または Values 修飾子が設定されている CIM クラスの各プロパティに生成された定数が含まれる。これらのクラス変数を生成するために、定数名および定数値を取得する方法は、プロパティのデータ型およびプロパティがどの修飾子を処理するかによって異なる **注 –** CIM 仕様で定義されている ValueMap および Values 修飾子は、修飾子名から予想される意味とは異なる場合がある。ValueMap ではプロパティの正当な値セットを定義し、Values では整数値と文字列を変換する。
VALUES	プロパティ、メソッド、パラメータ	Bean には、CIM ValueMap または Values 修飾子が設定されている CIM クラスの各プロパティの生成された定数が含まれる。これらのクラス変数を生成するために、定数名および定数値を取得する方法は、プロパティのデータ型およびプロパティがどの修飾子を処理するかによって異なる **注 –** CIM 仕様で定義されている ValueMap および Values 修飾子は、修飾子名から予想される意味とは異なる場合がある。ValueMap ではプロパティの正当な値セットを定義し、Values では整数値と文字列を変換する
VERSION	クラス、スキーマ、関連、インジケーション	クラスには、`getClassVersion()` メソッドが設定されている

次の表では、MOF 要素を Java 要素にマップする方法を説明します。

表 **2–6** MOF 要素を Java 要素にマップする方法

MOF 要素	**Java 要素**
説明修飾子	クラス、プロパティ、またはメソッドの説明
クラスの完全な MOF 表現	Java インタフェースと実装される Bean を指定するクラス Javadoc の説明

JavaBeans の生成の例

ここでは、mofcomp コマンドに -j オプションを指定した場合に生成される JavaBeans の例を示します。

mofcomp コマンドは、スーパーユーザーとして実行するか、コンパイルを実行中の名前空間に書き込み権を持つユーザーとして実行する必要があります。

注 – コマンド行に直接パスワードを入力する必要があるので、mofcomp コマンドの実行時には、-u (ユーザー) と -p (パスワード) オプションを同時に指定しないようにしてください。代わりに、暗号化されたパスワードを入力するプロンプトが表示されるように -u オプションだけを指定します。

例 **2–1** JavaBeans の生成

```
/usr/sadm/bin/mofcomp -u root -p mypassword -o /tmp
-j /tmp/bean.txt /usr/sadm/mof/Simple.mof
```

/usr/sadm/mof/Simple.mof の内容は、次のとおりです。

```
/usr/sadm/mof/Simple.mof
--------------------
#pragma include ("CIM_Core26.mof")

class Simple_Class {

      [Key, Description ("Name of the class.") ]
   string Name;

      [Description ("Method to print the contents of the class.") ]
   string printClass();

};
```

/tmp/bean.txt の内容は、次のとおりです。

```
/tmp/bean.txt
----------
PACKAGE=foo.com
```

例 **2–1** JavaBeans の生成　(続き)

```
IMPORTS=java.lang.Exception
EXCEPTIONS=Exception
```

CIMBean.java の内容は、次のとおりです。

```
package foo.com;

import javax.wbem.cim.CIMException;
import javax.wbem.client.CIMOMHandle;
import javax.wbem.cim.CIMInstance;

/**
 * このインタフェースは、CIMBeanImpl とそのサブクラスによって
 *  実装されるメソッドを定義する。CIMBeanImpl は、'mofcomp -j'
 *  によって生成される Java ソースの基底クラスで構成される
 */
public interface CIMBean {

    /**
     * このメソッドは、CIMBean の CIMOMHandle を返す
     *
     * @return    CIMOMHandle    CIMOM のハンドル
     */
    public CIMOMHandle getCIMOMHandle();

    /**
     * このメソッドは、CIMBean の CIMOMHandle を指定値に設定する
     *
     * @param    CIMOMHandle    CIMOM のハンドル
     */
    public void setCIMOMHandle(CIMOMHandle handle);

    /**
     * このメソッドは、CIMBean の CIMInstance を返す
     *
     * @return    CIMInstance    管理されている CIMInstance のハンドル
     */
    public CIMInstance getCIMInstance();

    /**
     * このメソッドは、CIMBean の CIMInstance を指定値に設定する
     *
     * @param    CIMInstance    管理されている CIMInstance のハンドル

     */
    public void setCIMInstance(CIMInstance instance);

    /**
     * このメソッドは、リモート呼び出しを実行して
     * CIMOM の CIMInstance を更新する
     */
    public void update() throws CIMException;
```

例 **2–1** JavaBeans の生成　*(続き)*

```
    /**
     * このメソッドは、リモート呼び出しを実行して
     * CIMOM の CIMInstance の指定された CIMProperty を更新する
     *
     * @param    String    CIMInstance では、このプロパティ名が更新される
     * @param    Object    CIMProperty では、このプロパティ値が更新される
     */
    public void update(String propName, Object value) throws CIMException;

    /**
     * このメソッドは、リモート呼び出しを実行して
     * CIMOM の CIMInstance を削除する
     */
    public void delete() throws CIMException;

    /**
     * このメソッドは、CIMInstance のキー修飾プロパティ
     * 名の文字列型の配列を返す。これは、CIMInstance に
     * 修飾子情報が含まれない場合に CIMInstance の
     * CIMObjectPath を作成するために必要
     *
     * @return String Version 修飾子の値または "-1" (値が
     * 指定されていない場合)
     */
    public String[] getBeanKeys();

    /**
     * このメソッドは、CIM クラスの Version 修飾子の値または
     * "-1"(この修飾子に値が設定されていない場合) を返す
     *
     * @return    String    キー修飾プロパティ名
     */
    public String getBeanVersion();

    /**
     * このメソッドは、CIMBean の文字列表現を返す。
     * このメソッドは、デバッグ用。文字列の
     * 形式は、実装ごとに異なる場合がある。空白文字列が
     * 返されても、NULL 文字ではない場合がある
     *
     * @return    String     Bean の文字列表現
     */
    public String toString();

} // CIMBean インタフェース
```

CIMBeanImpl.java の内容は、次のとおりです。

```
package foo.com;

import java.io.Serializable;
import java.util.*;
import javax.wbem.client.CIMOMHandle;
```

例 **2–1** JavaBeans の生成　(続き)

```
import javax.wbem.cim.CIMException;
import javax.wbem.cim.CIMInstance;
import javax.wbem.cim.CIMObjectPath;
import javax.wbem.cim.CIMValue;
import javax.wbem.client.CIMOMHandle;

/**
 * このクラスは、CIMBean インタフェースを実装する。このクラスは、
 * mofcomp -j によって生成された Java ソースコードの基底クラス
 */
public class CIMBeanImpl implements CIMBean, Serializable {

    private CIMInstance    cimInstance = null;
    private CIMOMHandle    cimomHandle = null;

    /**
     * このデフォルトコンストラクタにはパラメータがなく、
     * CIMBeanImpl の空のインスタンスを作成する
     */
    public CIMBeanImpl() {

    super();

    } // コンストラクタ

    /**
     * このコンストラクタには、指定された CIMOMHandle および
     * CIMInstance を設定する。
     * またこのコンストラクタは CIMBeanImpl を作成する
     *
     * @param    CIMOMHandle    CIMOM へのハンドル
     * @param    CIMInstance    管理されている CIMInstance へのハンドル
     */
    public CIMBeanImpl(CIMOMHandle handle, CIMInstance instance) {

    super();
    cimomHandle = handle;
    cimInstance = instance;

    } // constructor

    /**
     * このメソッドは、CIMBean の CIMOMHandle を返す
     *
     * @return    CIMOMHandle    CIMOM へのハンドル
     */
    public CIMOMHandle getCIMOMHandle() {

    return (cimomHandle);

    } // getCIMOMHandle

    /**
```

例 **2–1** JavaBeans の生成　*(続き)*

```
     * このメソッドは、CIMBean の CIMOMHandle を指定値に設定する
     *
     * @param    CIMOMHandle    CIMOM へのハンドル
     */
    public void setCIMOMHandle(CIMOMHandle handle) {

    this.cimomHandle = handle;

    } // setCIMOMHandle

    /**
     * このメソッドは、CIMBean の CIMInstance を返す
     *
     * @return    CIMInstance    管理されているCIMInstance へのハンドル
     */
    public CIMInstance getCIMInstance() {

    return (cimInstance);

    } // getCIMInstance

    /**
     * このメソッドは、CIMBean の CIMInstance を
     * 指定値に設定する
     *
     * @param    CIMInstance    管理されている CIMInstance へのハンドル
     */
    public void setCIMInstance(CIMInstance instance) {

    this.cimInstance = instance;

    } // setCIMInstance

    /**
     * このメソッドは、リモート呼び出しを実行して
     * CIMOM の CIMInstance を更新する
     */
    public void update() throws CIMException {

    cimomHandle.setInstance(getObjectPath(), cimInstance);

} // update

    /**
     * このメソッドは、リモート呼び出しを実行してCIMOM の
     *  CIMInstance の指定した CIMProperty を更新する
     *
     * @param    String    CIMInstance でこのプロパティ名を更新する
     * @param    Object    CIMProperty でこのプロパティ値を更新する
     */
    public void update(String propName, Object value) throws CIMException {

    cimomHandle.setProperty(getObjectPath(), propName, new CIMValue(value));
```

例 **2–1** JavaBeans の生成　(続き)

```
} // update

/**
 * このメソッドは、リモート呼び出しを実行し、
 * CIMOM の CIMInstance を削除する
 */
public void delete() throws CIMException {

cimomHandle.deleteInstance(getObjectPath());

} // delete

/**
 * サブクラスは、この簡易メソッドを使用して指定された CIMProperty の
 *  CIMValue に含まれるオブジェクトを取得する
 * 注: 返されるオブジェクトは NULL 文字の場合がある
 *
 * @param    String    このプロパティ名の値を取得する必要がある
 * @return    Object    CIMProperty の CIMValue に含まれるオブジェクト
 */
protected Object getProperty(String propName) {

try {

    return (cimInstance.getProperty(propName).getValue().getValue());

} catch (NullPointerException npe) {
}
return ((Object)null);

} // getProperty

/**
 * サブクラスはこの簡易メソッドを使用して指定された
 * CIMProperty の CIMValue に含まれる Vector に等しい
 * String[] を取得する
 * 注: 返される String[] は NULL である場合がある
 *
 * @param    String    このプロパティ名の値を取得する
 * @param    String[]    プロパティ Values 修飾子データ
 * @param    Object[]    プロパティ ValueMap 修飾子データ
 * @return    String[]    プロパティ値の定数のコンテナ
 */
protected String[] getArrayProperty(String propName, String[]
valueArr,    Object[] valueMapArr) {

List propList = null;
try {

    propList =
    ((List)cimInstance.getProperty(propName).getValue().getValue());
```

例 **2–1** JavaBeans の生成　*(続き)*

```
    } catch (NullPointerException npe) {
    }

    if (propList != null) {

        String[] returnArr;
        returnArr = new String[propList.size()];
        ListIterator listIterator = propList.listIterator();
        int counter = 0;
        while (listIterator.hasNext()) {

        returnArr[counter] = valueArr[getArrayIndex(valueMapArr,
            listIterator.next())];
        counter++;

        }
        return (returnArr);

    }
    return ((String[])null);

    } // getArrayProperty

    /**
     * このメソッドは、プロパティ値（指定されたオブジェクトパスなど）が
     *  参照する CIMInstance を取得し、それを指定した Bean に設定する。
     * このメソッドは、Association プロパティのアクセス用メソッド
     * により使用される
     *
     * @param    CIMObjectPath    CIMInstance のオブジェクトパス
     * @param    CIMBeanImpl    取得した CIMInstance の Bean コンテナ
     */
    protected void getAssociationProperty(CIMObjectPath cop,
    CIMBeanImp    bean) throws CIMException {

    cop.setNameSpace("");
    CIMInstance ci = cimomHandle.getInstance(cop, false, true, true,
        (String[])null);
    bean.setCIMInstance(ci);
    bean.setCIMOMHandle(cimomHandle);

    } // getAssociationProperty

    /**
     * サブクラスは、この簡易メソッドを使用して名前が
     * 指定された CIMProperty の指定された Object 値を含む
     * CIMValue を設定する
     *
     * @param    String    このプロパティ名に新しい値を設定する
     * @param    Object    CIMInstance で更新するプロパティ値
     */
    protected void setProperty(String propName, Object propValue)throws
    IllegalArgumentException {
```

例 **2–1** JavaBeans の生成　(続き)

```
cimInstance.setProperty(propName, new CIMValue(propValue));

} // setProperty

/**
 * サブクラスは、この簡易メソッドを使用して、名前が
 * 指定された CIMProperty の指定された String[] に等しい
 *  Vector を含む CIMValue を設定する
 *
 * @param    String    このプロパティ名の値を取得する
 * @param    String[]    プロパティ Values の修飾子データ
 * @param    Object[]    プロパティ ValueMap の修飾子データ
 * @param    String[]    CIMInstance に設定するプロパティ値
 */
protected void setArrayProperty(String propName, String[] valueArr,
Object[] valueMapArr, String[] propValues) {

Vector vPropValue = new Vector(propValues.length);
for (int i = 0; i < propValues.length; i++) {

    vPropValue.addElement(valueMapArr[getArrayIndex(valueArr,
    propValues[i])]);

}
setProperty(propName, vPropValue);

} // setArrayProperty

/**
 * このメソッドは、CIMInstance の Key 修飾プロパティ
 * 名の文字列配列を返す。これは、修飾子情報が含ま
 * れない場合、CIMInstance の CIMObjectPath を構築する
 * ために必要
 *
 * @return    String[]    Key 修飾プロパティ名の配列
 */
public String[] getBeanKeys() {

return ((String[])null);

} // getBeanKeys

/**
 * このメソッドは、クラスの CIMInstance の CIMObjectPath を返す。
 *
 * @return    CIMObjectPath    CIMInstance のオブジェクトパス
 */
protected CIMObjectPath getObjectPath() {

CIMObjectPath cop = new CIMObjectPath(cimInstance.getClassName());
Vector vKeys = cimInstance.getKeyValuePairs();
if ((vKeys != null) && (vKeys.size()> 0)) {
```

例 **2–1** JavaBeans の生成　*(続き)*

```
        cop.setKeys(vKeys);

    } else {

        String[] keyArr = getBeanKeys();
        if (keyArr != null) {

        String keyProperty;
        for (int i = 0; i < keyArr.length; i++) {

            keyProperty = keyArr[i];
            cop.addKey(keyProperty,
            (cimInstance.getProperty(keyProperty)).getValue());

        }

        }

    }
    return (cop);

    } // getObjectPath

    /**
     * この簡易メソッドは、指定した配列に指定したオブジェクトの
     * インデックスを返す。
     * 配列にオブジェクトが含まれない場合は、-1 を返す
     *
     * @param  Object[]  オブジェクトのインデックスを検索する
     *                   オブジェクト配列
     * @param  Object    オブジェクト配列でこのオブジェクトのインデックスを
     *                   検索する
     * @return int       オブジェクト配列のオブジェクトのインデックス
     */
    protected int getArrayIndex(Object[] objArr, Object obj) {

    List arrList = Arrays.asList(objArr);
    return (arrList.indexOf(obj));

    } // getArrayIndex

    /**
     * このメソッドは、CIM クラスの Version 修飾子値を返す。
     * この修飾子が指定されていない場合は、"-1" を返す
     *
     * @return    String    Version 修飾子の値。
     * 修飾子が指定されていない場合は、-1。
     */
    public String getBeanVersion() {

    return ("-1");
```

例 **2–1** JavaBeans の生成　(続き)

```
    } // getBeanVersion

    /**
     * このメソッドは、CIMBean の文字列表現を返す。
     * このメソッドはデバッグ用。文字列の形式は、実装ごと
     * に異なる場合がある。
     * 返される文字列が空でも NULL 文字でない場合がある
     *
     * @return    String    Bean の文字列表現
     */
    public String toString() {

    return (cimInstance.toString());

    } // toString

} // CIMBeanImpl クラス
```

Simple_ClassBean.java の内容は次のとおりです。

```
package foo.com;

import javax.wbem.client.*;
import javax.wbem.cim.*;
import java.util.*;
import java.lang.Exception;

/**
 * このインタフェースには、CIM class Simple_Class で定義された
 * すべてのプロパティのアクセス用メソッドおよび変更メソッドが含まれる。
 * またこのクラスに定義された invokeMethods と等価のメソッドも含まれる。
 * このインタフェースは、Simple_ClassBeanImpl によって実装される。
 * CIM class Simple_Class は、次のように記述される
 */
public interface Simple_ClassBean extends CIMBean {

    /**
     * このメソッドは、Simple_Class.Name プロパティ値を返す。
     * このプロパティは、次のように記述される
     *
     * クラス名
     *
     * @return    String    現在の Name プロパティ値
     * @exception    Exception
     */
    public String getName() throws Exception;

    /**
     * このメソッドは、Simple_Class.Name プロパティ値を設定する。
     * このプロパティは、次のように記述する
     *
     * クラス名
     *
```

例 **2–1** JavaBeans の生成　*(続き)*

```
     * @param    String    新しい Name プロパティ値
     * @exception    Exception
     */
    public void setName(String name) throws Exception;


    /**
     * このメソッドは、Simple_Class.printClass() メソッドを呼び出す。
     * このメソッドは、次のように記述する
     *
     * クラスのコンテンツを出力するメソッド
     *
     * @return    String    printClass() 呼び出しの戻り値
     * @exception    Exception
     */
    public String printClass() throws Exception, CIMException;

} // Simple_ClassBean インタフェース
```

Simple_ClassBeanImpl.java の内容は、次のとおりです。

```
package foo.com;

import javax.wbem.client.*;
import javax.wbem.cim.*;
import java.util.*;
import java.lang.Exception;

/**
 * このインタフェースには、CIM class Simple_Class で定義された
 * すべてのプロパティのアクセス用メソッドおよび変更メソッドが含まれる。
 * またこのクラスに定義された invokeMethods と等価のメソッドも含まれる。
 * このクラスは、Simple_ClassBeanImpl インタフェースを実装する。
 * CIM class Simple_Class は、次のように記述される
 */
public class Simple_ClassBeanImpl extends CIMBeanImpl implements
    Simple_ClassBean {

     private CIMOMHandle cimomHandle = null;
    private CIMInstance cimInstance = null;
    private final static String[] keysArr = {"Name"};

    /**
     * このコンストラクタは、Simple_ClassBean インタフェースを
     * 実装する Simple_ClassBeanImpl Class を作成し、
     * Java Bean に CIM class Simple_Class をカプセル化する。
     * CIM class Simple_Class は次のように記述される
     *
     * @param    CIMOMHandle    CIMOM のハンドル
     * @param    CIMInstance    管理されている CIMInstance のハンドル
     */
    public Simple_ClassBeanImpl(CIMOMHandle handle, CIMInstance instance)
    {
```

例 **2–1** JavaBeans の生成　(続き)

```
 super(handle, instance);

this.cimomHandle = handle;
this.cimInstance = instance;

} // コンストラクタ

/**
 * このメソッドは、CIM クラスに定義された Key
 *  修飾プロパティの名前で文字列の配列を返す。
 * Key 修飾子が CIMInstance に含まれない場合、この
 * メソッドを使用して Bean によって管理される
 *  CIMInstance の CIMObjectPath を構築する
 *
 * @return    String[]    Key 修飾プロパティ名の配列
 */
public String[] getBeanKeys() {

return keysArr;

} // getBeanKeys

/**
 * このメソッドは、Simple_Class.Name プロパティ値を返す。
 * このプロパティは次のように記述される
 *
 * クラス名
 *
 * @return    String    現在の Name プロパティ値
 * @exception    Exception
 */
public String getName() throws Exception {

return (String)getProperty("Name");

} // getName

/**
 * このメソッドは、Simple_Class.Name プロパティ値を設定する。
 * このプロパティは、次のように記述される
 *
 * クラス名
 *
 * @param    String    新しい Name プロパティ値
 * @exception    Exception
 */
public void setName(String name) throws Exception {

setProperty("Name", name);

} // setName
```

例 **2–1** JavaBeans の生成　*(続き)*

```
    /**
     * このメソッドは、Simple_Class.printClass() メソッドを呼び出す。
     * このメソッドは、次のように記述される
     *
     * クラスのコンテンツを出力するメソッド
     *
     * @return    String    printClass() 呼び出しの戻り値
     * @exception    Exception
     */
    public String printClass() throws Exception, CIMException {

    Vector vInParams = new Vector();
    Vector vOutParams = new Vector();

    CIMValue cv = cimomHandle.invokeMethod(cimInstance.getObjectPath(),
 "printClass", vInParams, vOutParams);
    return (String)cv.getValue();

    } // printClass
} // Simple_ClassBeanImpl クラス
```

第 3 章

クライアントプログラムの記述

この章では、Solaris WBEM SDK クライアント API (`javax.wbem.client`) を使用して、クライアントプログラムを記述する方法を説明します。次の内容が含まれます。

- 47 ページの「概要」
- 48 ページの「クライアント接続の開始と終了」
- 50 ページの「基本的なクライアント操作の実行」
- 70 ページの「アクセス制御の設定」
- 73 ページの「修飾子と修飾子のデータ型の処理」
- 74 ページの「クライアント要求のバッチ処理」
- 76 ページの「CIM イベントの処理」
- 84 ページの「ログメッセージの読み取りと書き込み」

注 – WBEM クライアント API (`javax.wbem.client`) の詳細は、`/usr/sadm/lib/wbem/doc/index.html` を参照してください。

概要

WBEM クライアントアプリケーションは、`javax.wbem.client` API を使用して CIM オブジェクトを操作します。クライアントアプリケーションは、CIM API を使用してオブジェクト(クラス、インスタンス、名前空間など) を構築し、続いてそのオブジェクトの初期化またはインスタンス化を実行します。クライアントアプリケーションはクライアント API を使用してオブジェクトを CIM Object Manager に渡し、CIM のクラス、インスタンス、名前空間の作成などの WBEM 操作を要求します。

クライアントアプリケーションの処理手順

クライアントアプリケーションは通常、次の手順で処理を行います。

1. `CIMClient` を使用して CIM Object Manager に接続します。クライアントアプリケーションは、WBEM の操作 (CIM クラスの作成や CIM インスタンスの更新など) を実行する必要があるたびに、CIM Object Manager に接続します。詳細は、48 ページの「クライアント接続の開始と終了」を参照してください。
2. クライアントAPI を使用して、操作の要求およびプログラミング作業を実行します。アプリケーションの機能セットは、どの処理を要求すべきかを決定します。ほとんどのプログラムが実行するタスクには、インスタンスの作成、削除、更新、オブジェクトの列挙、メソッドの呼び出し、クラス定義の取得、およびエラーの処理が含まれます。クライアントプログラムから、クラスの作成と削除、名前空間の作成と削除、および修飾子の使用を実行することも可能です。詳細は、50 ページの「基本的なクライアント操作の実行」を参照してください。
3. `CIMClient` を使用して CIM Object Manager へのクライアント接続を閉じて、クライアントセッションが使用しているリソースをすべて解放します。詳細は、48 ページの「クライアント接続の開始と終了」を参照してください。

クライアント接続の開始と終了

クライアントアプリケーションは、最初に CIM Object Manager を使用して接続を確立してから、CIM クラス、CIM インスタンス、CIM 修飾子タイプの追加、変更、削除などの WBEM 操作を実行します。クライアントアプリケーションと CIM Object Manager は、同一のホスト上でも、別のホスト上でも実行可能です。また、複数のクライアントが 1 つの CIM Object Manager に接続を確立することもできます。

名前空間について

アプリケーションが CIM Object Manager への接続を行う場合、名前空間への接続も行う必要があります。後続の操作はすべてこの名前空間上で行われます。名前空間は、ディレクトリに似た構造を持ち、内部にクラス、インスタンス、および修飾子タイプを含みます。名前空間内のオブジェクト名は、すべて一意にする必要があります。Solaris WBEM SDK をインストールすると、次の 4 つの名前空間が作成されます。

- `root\cimv2` – デフォルトの名前空間。Solaris WBEM Services がインストールされたシステムのオブジェクトを表す CIM クラスが含まれる。
- `root\security` – セキュリティ関連のクラスが含まれる。
- `root\snmp`– SNMP アダプタクラスが含まれる。
- `root\system`– CIM Object Manager を管理するクラスが含まれる。

クライアント接続の開始

クライアント接続を開始する場合、CIMClient クラスを使用して CIM Object Manager に接続します。CIMClient クラスは、次の 4 つの引数を取ります。

- *name* – 必須。クライアント接続に使用されるホストおよび名前空間の名前を含む CIMNameSpace オブジェクトのインスタンス。デフォルトは、ローカルホスト (クライアントアプリケーションが稼働するホスト) 上の root\cimv2。CIM Object Manager への接続が完了すると、後続の CIMClient 操作はすべて指定された名前空間内で発生する
- *principal* – 必須。有効な Solaris ユーザーアカウント名を含む UserPrincipal オブジェクトのインスタンス。CIM Object Manager は、ユーザー名のアクセス権を検査して、CIM オブジェクトに許可されるアクセスの種類を判断する
- *credential* – 必須。UserPrincipal Solaris アカウントの有効なパスワードを含む PasswordCredential オブジェクトのインスタンス
- *protocol* – 任意 (文字列)。CIM Object Manager へのメッセージ送信に使用するプロトコル (RMI (デフォルト) または HTTP)

例 **3–1** ルートアカウントへの接続

次の例では、アプリケーションは、デフォルトの名前空間のローカルホストで稼働する CIM Object Manager に接続します。アプリケーションは、デフォルトの名前空間のすべての CIM オブジェクトに読み取り権および書き込み権を持つ、ルートアカウント用 UserPrincipal オブジェクトを作成します。

```
{
    ...

  /* 2 つの NULL 文字を使用して初期化された名前空間オブジェクトを作成する。
     2 つの NULL 文字はデフォルトのホスト (ローカルホスト) とデフォルトの
     名前空間 (root\cimv2) を表す */

    CIMNameSpace cns = new CIMNameSpace("", "");

    UserPrincipal up = new UserPrincipal("root");
    PasswordCredential pc = new PasswordCredential("root_password");
    /* root パスワードを使用し、スーパーユーザーとして名前空間に接続する */

    CIMClient cc = new CIMClient(cns, up, pc);
    ...
}
```

例 **3–2** ユーザーアカウントへの接続

次の例では、アプリケーションは、最初に CIMNameSpace、UserPrincipal、および PasswordCredential オブジェクトのインスタンスを作成します。次に、CIMClient クラスを使用して CIM Object Manager に接続し、ホスト名、名前空間、ユーザー名、およびパスワード資格を CIM Object Manager に渡します。

```
{
    ...
    /* ホスト happy 上の A (名前空間名)
```

例 **3–2** ユーザーアカウントへの接続　(続き)

```
        によって初期化される名前空間オブジェクトを作成する */
    CIMNameSpace cns = new CIMNameSpace("happy", "A");
    UserPrincipal up = new UserPrincipal("Mary");
    PasswordCredential pc = new PasswordCredential("marys_password");
    CIMClient cc = new CIMClient(cns, up, pc);
    ...
    ...
}
```

例 **3–3** RBAC の役割 ID の認証

`SolarisUserPrincipal` および `SolarisPasswordCredential` クラスを使用して、ユーザーの役割 ID を認証します。次の例では、`Mary` の役割を `Admin` として認証します。

```
{
...
CIMNameSpaceRole cns = new CIMNameSpace("happy", "A");
SolarisUserPrincipal sup = new SolarisUserRolePrincipal("Mary", "Admin");
SolarisPswdCredential spc = new
        SolarisPswdCredential("marys_password", "admins_password");
CIMClient cc = new CIMClient(cns, sup, spc);
```

クライアント接続の終了

`CIMClient` クラスの `close` メソッドを使用して、クライアント接続を閉じ、セッションが使用したサーバーリソースを解放します。

例 **3–4** クライアント接続の終了

次の例では、クライアント接続を閉じます。インスタンス変数 *cc* は、クライアント接続を表します。

```
...
cc.close();
...
```

基本的なクライアント操作の実行

このセクションでは、`javax.wbem.client` API を使用して、操作の要求および一般的なプログラミング作業を実行する方法を説明します。

インスタンスの作成

既存のクラスのインスタンスを作成するには、newInstance メソッドを使用します。既存のクラスがキープロパティを保持する場合、アプリケーションはそのプロパティを一意の値に設定する必要があります。インスタンスは、必要に応じてそのクラスに定義されていない別の修飾子を定義することもできます。それらの修飾子をインスタンスまたは特定のインスタンスプロパティ用に定義できますが、クラス宣言内で定義する必要はありません。

アプリケーションは、クラスに定義されている一連の修飾子を getQualifiers メソッドを使用して取得できます。

例 **3–5** インスタンスの作成

次の例では、newInstance メソッドを使用して、 CIM インスタンスを表す Java クラスを作成します (たとえば、 Solaris_Package クラスから Solaris パッケージを作成する)。

```
...
{
/* ローカルホストの root\cimv2 名前空間の
   CIM Object Manager に接続。root\cimv2 名前空間内の
   オブジェクトに対する書き込み権を持つアカウントのユーザー名とパスワードを
   指定する */

    UserPrincipal up = new UserPrincipal("root");
    PasswordCredential pc = new PasswordCredential("root_password");
    /* root パスワードを使用して、スーパーユーザーとして名前空間に
       接続する */

    CIMClient cc = new CIMClient(cns, up, pc);
...

// Solaris_Package クラスを取得
cimclass = cc.getClass(new CIMObjectPath("Solaris_Package"),
                                          true, true, true, null);

/* プロパティのデフォルト値を使用して生成された Solaris_Package
   クラスの新しいインスタンスを作成する。このクラスのプロバイダが
   デフォルト値を指定しないと、プロパティの値は NULL 文字になるため、
   明示的に設定する必要がある */

 CIMInstance ci = cc.createInstance (new CIMObjectPath("Solaris_Package"),
 ci);
}
...
```

インスタンスの削除

インスタンスの削除には、deleteInstance メソッドを使用します。

例 **3–6** インスタンスの削除

次の例では、クライアントアプリケーションを CIM Object Manager に接続し、CIMObjectPath を使用して削除するオブジェクトの CIM オブジェクトパスを含むオブジェクトを構築し、enumerateInstance を使用してインスタンスおよびサブクラスのすべてのインスタンスを取得し、 deleteInstance を使用して各インスタンスを削除します。

```
import java.rmi.*;
import java.util.Enumeration;

import javax.wbem.cim.CIMClass;
import javax.wbem.cim.CIMException;
import javax.wbem.cim.CIMInstance;
import javax.wbem.cim.CIMNameSpace;
import javax.wbem.cim.CIMObjectPath;

import javax.wbem.client.CIMClient;
import javax.wbem.client.PasswordCredential;
import javax.wbem.client.UserPrincipal;

/**
 * 指定されたクラスのインスタンスをすべて返す。
 * このプログラム例では、5 つの必須コマンド行引数、hostname (args[0])、
 * username (args[1])、password (args[2]) namespace (args[3])、
 * および classname (args[4]) を取り、
 * 指定されたクラス名のインスタンスをすべて削除する。
 * 指定されたユーザー名は、
 * 指定された名前空間への書き込み権を持つ必要がある
 */
public class DeleteInstances {
    public static void main(String args[]) throws CIMException {
        CIMClient cc = null;
        // 5 つの引数が指定されない場合、使用法を表示して終了する
        if (args.length != 5) {
           System.out.println("Usage: DeleteInstances host username " +
                              "password namespace classname ");
           System.exit(1);
           }
       try {
          // args[0] にはホスト名が、args[3] には名前空間
          // が含まれる。指定されたホスト上の指定された
          // 名前空間を指す CIMNameSpace (cns) を作成する
          CIMNameSpace cns = new CIMNameSpace(args[0], args[3]);

         // args[1] と args[2] にはユーザー名およびパスワードが含まれる。
         // ユーザー名を使用して UserPrincipal (up) を作成し、
         // パスワードを使用して PasswordCredential を作成する
         UserPrincipal up = new UserPrincipal(args[1]);
         PasswordCredential pc = new PasswordCredential(args[2]);

        // CIM Object Manager に接続して、
        // 作成した CIMNameSpace、UserPrincipal、および
        // PasswordCredential オブジェクトを渡す
```

例 **3–6** インスタンスの削除　*(続き)*

```
        cc = new CIMClient(cns, up, pc);

        // クラス名 (args[4]) を取得し、CIMObjectPath を作成する
        CIMObjectPath cop = new CIMObjectPath(args[4]);

        // クラスおよびそのサブクラスすべてのインスタンスオブジェクトパス
        // すべての列挙を取得する。インスタンスオブジェクトパスは、
        // CIM Object Manager がインスタンス検出に
        // 使用する参照を指す
        Enumeration e = cc.enumerateInstanceNames(cop);


        // 列挙されたインスタンスオブジェクトパスを繰り返し処理する。
        // オブジェクトを作成して列挙された各インスタンスのオブジェクトパス
        // を格納し、インスタンスの出力後に削除する
        while (e.hasMoreElements()) {
           CIMObjectPath  op = (CIMObjectPath)e.nextElement();
           System.out.println(op);
           cc.deleteInstance(op);
        } // while 終了
  } catch (Exception e) {
       // 例外が発生した場合はそれを出力する
       System.out.println("Exception: "+e);
  } // catch 終了

 // セッションを閉じる
 if (cc != null) {
     cc.close();
        }
    }
}
```

インスタンスの取得と設定

クライアントアプリケーションが CIM Object Manager から CIM インスタンスを取得する場合には、通常 `getInstance` メソッドが使用されます。クラスのインスタンスが作成されるときに、インスタンスはその派生元クラスとそのクラス階層にあるすべての親クラスのプロパティを継承します。`getInstance` メソッドはブール値引数 *localOnly* を受け取ります。*localOnly* が true の場合、getInstance メソッドは指定されたインスタンスによって継承されたプロパティ以外のプロパティだけを返します。

- これらのプロパティは、そのインスタンス自体によって定義されたものです。
- *localOnly* が false の場合、そのクラスのすべてのプロパティ (インスタンス内で定義されたプロパティおよびクラス階層内のすべての親クラスから継承されたプロパティすべて) が返されます。

既存のインスタンスを更新する場合は、`setInstance` メソッドを使用します。

例 **3–7** インスタンスの取得と設定

次の例では、列挙されたオブジェクトパスのインスタンスを取得し、各インスタンス内で b のプロパティ値を 10 に更新し、更新したインスタンスを CIM Object Manager に渡します。

```
...
{
    // オブジェクトパス、myclass という CIM 名を含むオブジェクト
    // を作成する
    CIMObjectPath cop = new CIMObjectPath("myclass");

    /* 列挙内の各インスタンスオブジェクトパスのインスタンスを取得し、
       各インスタンス内で b のプロパティ値を 10 に更新し、
       更新したインスタンスを CIM Object Manager に渡す */

    while(e.hasMoreElements()) {
           CIMInstance ci = cc.getInstance((CIMObjectPath)(e.nextElement
                                          ()),true, true, true, null);
           ci.setProperty("b", new CIMValue(new Integer(10)));
           cc.setInstance(new CIMObjectPath(),ci);
           }
}
...
```

プロパティの取得と設定

CIM プロパティは、CIM クラスの特性を記述する値です。プロパティは、プロパティ値を取得する機能と、プロパティ値を設定する機能の組み合わせとして考えることができます。

例 **3–8** プロパティの取得

次の例では、enumerateInstanceNames を使用して Solaris プロセッサのすべてのインスタンス名を返し、getProperty を使用して各インスタンスの現在のクロック速度の値を取得し、println を使用して出力します。

```
...
{
/* オブジェクト CIMObjectPath を作成して、
Solaris_Processor クラスの名前を格納する */

CIMObjectPath cop = new CIMObjectPath("Solaris_Processor");

/* CIM Object Manager は、Solaris_Processor クラスのインスタンス名を
   含む列挙を返す */

Enumeration e = cc.enumerateInstanceNames(cop);

/* インスタンスオブジェクトパスの列挙を繰り返し処理する。
   getProperty メソッドを使用して、Solaris プロセッサごとの
   現在のクロック速度の値を取得する */
```

例 **3–8** プロパティの取得　(続き)

```
while(e.hasMoreElements()) {
        CIMValue cv = cc.getProperty(e.nextElement(CIMObjectPath),
                                     "CurrentClockSpeed");
        System.out.println(cv);
}
...
}
```

例 **3–9** プロパティの設定

次の例では、すべての Solaris_UserTemplate インスタンスの初期シェル 値を設定します。このコードセグメントは、enumerateInstanceNames を使用して Solaris_UserTemplate のすべてのインスタンス名を取得し、setProperty を使用して各インスタンスの初期シェル値を設定します。

```
...
{
    /* オブジェクト (CIMObjectPath) を作成して
    Solaris_Processor クラスの名前を格納する */

    CIMObjectPath cop = new CIMObjectPath("Solaris_UserTemplate");

    /* CIM Object Manager は、Solaris_UserTemplate クラスおよび
       そのサブクラスすべてのインスタンス名を含む列挙を返す */

    Enumeration e = cc.enumerateInstanceNames(cop);

    /* インスタンスオブジェクトパスの列挙を繰り返し
処理する。setProperty メソッドを使用して、初期シェル値を
Solaris_UserTemplate インスタンスごとに /usr/bin/sh に
設定する */

   for (; e.hasMoreElements(); cc.setProperty(e.nextElement(),
        "/usr/bin/sh", new CIMValue(new Integer(500))));

...
}
```

オブジェクトの列挙

列挙とはオブジェクトの集合です。列挙は一度に 1 つずつ取り出すことができます。クラス、クラス名、インスタンス、インスタンス名、および名前空間を列挙できます。次の表に示すように、列挙の結果は使用するメソッドや引数によって異なります。

オブジェクトの列挙

表 **3–1** オブジェクトの列挙

メソッド	引数なし	*deep*	*localOnly*
`enumerateClasses`	*path* に指定されたクラスの内容を返す	**true** の場合:指定されたクラスのサブクラスの内容を返す。ただし、クラス自体は返さない	**true** の場合:指定されたクラスの継承しないプロパティおよびメソッドのみを返す
		false の場合:指定されたクラスの直接のサブクラスの内容を返す	**false** の場合:指定されたクラスのプロパティをすべて返す
`enumerateInstances`	*path* に指定されたクラスのインスタンスを返す	**true** の場合:指定されたクラスおよびそのサブクラスのインスタンスを返す	**true** の場合:指定されたクラスのインスタンスの継承しないプロパティのみを返す
		false の場合:指定されたクラスおよびそのサブクラスのインスタンスを返すサブクラスのプロパティは、フィルタ処理される	**false** の場合:指定されたクラスのインスタンスのプロパティをすべて返す
`enumerateClassNames`	*path* に指定されたクラスの名前を返す	**true** の場合:指定されたクラスから派生したすべてのクラスの名前を返す	なし
		false の場合:指定されたクラスの第 1 レベルの子の名前のみを返す	なし
`enumerateInstanceNames`	*path* に指定されたクラスのインスタンスの名前を返す	なし	なし
		なし	なし
`enumNameSpace`	*path* に指定された名前空間内の名前空間のリストを返す	**true** の場合:指定された名前空間内の名前空間階層全体を返す	なし
		false の場合:指定された名前空間の第 1 レベルの子のみを返す	なし

例 **3–10** 列挙クラス

次のプログラム例は、クラスおよびそのサブクラスの内容を返します。

```
...
{
    /* CIMObjectPath オブジェクトを作成し、
    列挙する CIM クラスの名前 (myclass) を使用して
初期化する */
```

例 **3–10** 列挙クラス　*(続き)*

```
    CIMObjectPath cop = new CIMObjectPath(myclass);

    /* この列挙には、列挙されたクラス内のクラスとサブクラスが
       含まれる (deep=true)。この列挙は、クラスおよびサブクラスごとに
       継承されないメソッドおよびプロパティのみを返す
       (localOnly は true)*/

    Enumeration e = cc.enumerateClasses(cop, true, true);
}
...
```

例 **3–11** クラスおよびインスタンスの列挙

次のプログラム例は、クラスおよびインスタンスの列挙で deep および shallow (*deep=false*) が指定された場合の動作を示します。 *localOnly* フラグは、クラスおよびインスタンスの名前ではなく、クラスおよびインスタンスの内容を返します。

```
import java.rmi.*;
import java.util.Enumeration;

import javax.wbem.client.CIMClient;
import javax.wbem.cim.CIMClass;
import javax.wbem.cim.CIMException;
import javax.wbem.cim.CIMInstance;
import javax.wbem.cim.CIMNameSpace;
import javax.wbem.cim.CIMObjectPath;

import javax.wbem.client.UserPrincipal;
import javax.wbem.client.PasswordCredential;


/**
 * このプログラム例では、クラスとインスタンスを列挙する。
コマンド行から渡されるクラスについて、deep および
 * shallow 列挙を実行する
 */
public class ClientEnum {

    public static void main(String args[]) throws CIMException {
        CIMClient cc = null;
        CIMObjectPath cop = null;
        if (args.length < 4) {
            System.out.println("Usage: ClientEnum host user passwd " +
                               "classname");
            System.exit(1);
   }
   try {
       CIMNameSpace cns = new CIMNameSpace(args[0]);
       UserPrincipal up = new UserPrincipal(args[1]);
       PasswordCredential pc = new PasswordCredential(args[2]);
       cc = new CIMClient(cns, up, pc);
```

例 **3–11** クラスおよびインスタンスの列挙　*(続き)*

```
        // コマンド行からクラス名を取得する
        cop = new CIMObjectPath(args[3]);
        // クラスの deep 列挙を実行する
        Enumeration e = cc.enumerateClasses(cop, true, true, true,
                                            true);
        // クラスのサブクラスをすべて出力する
        while (e.hasMoreElements()) {
            System.out.println(e.nextElement());
        }
        System.out.println("+++++");
        // クラスの shallow 列挙を実行する
        e = cc.enumerateClasses(cop, false, true, true, true);
        // 第 1 レベルのサブクラスを出力する
        while (e.hasMoreElements()) {
            System.out.println(e.nextElement());
        }
        System.out.println("+++++");
        // クラスのインスタンスの deep 列挙を実行する
        e = cc.enumerateInstances(cop, false, true, true, true, null);
        // クラスおよびそのサブクラスのインスタンスを
        // すべて出力する
        while (e.hasMoreElements()) {
            System.out.println(e.nextElement());
        }
        System.out.println("+++++");
        // クラスのインスタンスの shallow 列挙を実行する
        e = cc.enumerateInstances(cop, false, false, true, true, null);
        // クラスのインスタンスをすべて出力する
        while (e.hasMoreElements()) {
            System.out.println(e.nextElement());
        }
        System.out.println("+++++");

        e = cc.enumerateInstanceNames(cop);
        while (e.hasMoreElements()) {
            System.out.println(e.nextElement());
        }
        System.out.println("+++++");

        e = cc.enumerateInstanceNames(cop);
        while (e.hasMoreElements()) {
            CIMObjectPath opInstance = (CIMObjectPath)e.nextElement();
            CIMInstance ci = cc.getInstance(opInstance, false,
                                            true, true, null);
            System.out.println(ci);
        }
        System.out.println("+++++");

    }
    catch (Exception e) {
        System.out.println("Exception: "+e);
```

例 **3–11** クラスおよびインスタンスの列挙　(続き)

```
  }

 // セッションを閉じる
 if (cc != null) {
     cc.close();
  }
 }
}
```

例 **3–12** クラス名の列挙

次のプログラム例は、クラス名およびサブクラス名のリストを返します。

```
...
{
    /* CIMObjectPath オブジェクトを作成し、
    列挙する CIM クラスの名前 (myclass) を使用して初期化する */
    CIMObjectPath cop = new CIMObjectPath(myclass);

    /* この列挙には、列挙されたクラス内のクラスおよびサブクラスの
       名前が含まれる */
    Enumeration e = cc.enumerateClassNames(cop, true);
}
...
```

例 **3–13** 名前空間の列挙

このプログラム例は、CIMClient クラスの enumNameSpace メソッドを使用して、名前空間とその中に含まれるすべての名前空間の名前を出力します。

```
import java.rmi.*;
import java.util.Enumeration;

import javax.wbem.cim.CIMClass;
import javax.wbem.cim.CIMException;
import javax.wbem.cim.CIMInstance;
import javax.wbem.cim.CIMNameSpace;
import javax.wbem.cim.CIMObjectPath;

import javax.wbem.client.CIMClient;
import javax.wbem.client.PasswordCredential;
import javax.wbem.client.UserPrincipal;


/**
 *
 */
public class EnumNameSpace {
    public static void main(String args[]) throws CIMException {
        CIMClient cc = null;
        // 4 つの引数が指定されない場合、使用法を表示して終了する
```

例 **3–13** 名前空間の列挙　(続き)

```
        if (args.length < 4) {
            System.out.println("Usage: EnumNameSpace host username " +
                               "password namespace");
            System.exit(1);
            }
        try {
            // args[0] にはホスト名が含まれる。指定されたホスト上の
            // 指定された名前空間を指す。
            // CIMNameSpace (cns) を作成する
            CIMNameSpace cns = new CIMNameSpace(args[0], "");

           // args[1] と args[2] にはユーザー名およびパスワードが含まれる。
           // ユーザー名を使用して UserPrincipal (up) を、
           // パスワードを使用して PasswordCredential を作成する
           UserPrincipal up = new UserPrincipal(args[1]);
           PasswordCredential pc = new PasswordCredential(args[2]);

           // CIM Object Manager に接続して
           // 作成した CIMNameSpace、UserPrincipal、および
           // PasswordCredential オブジェクトを渡す
           cc = new CIMClient(cns, up, pc);

           // 名前空間 (args[3]) を使用して CIMObjectPath を作成する
           CIMObjectPath cop = new CIMObjectPath("", args[3]);

           // 名前空間を列挙する
           Enumeration e = cc.enumNameSpace(cop);
           while (e.hasMoreElements()) {
               System.out.println((CIMObjectPath)e.nextElement());
           } // while の終了

           } catch (Exception e) {
              // 例外が発生した場合はそれを出力する
              System.out.println("Exception: "+ e);
           } // catch の終了

          // セッションを閉じる
          if (cc != null) {
              cc.close();
        }
    }
}
```

関連の作成

関連とは、コンピュータやそのハードディスクなどの管理される複数のリソース間の関係を表したものです。この関係は、関連修飾子を含む特殊なクラス型である関連クラス内で抽象化されます。オブジェクト自体に影響を与えることなく、関連クラスを追加または変更できます。

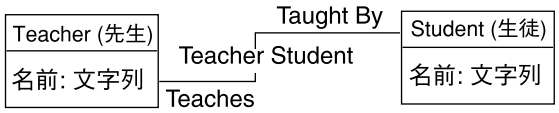

図 **3–1** TeacherStudent 関連 1

図 3–1 に、`Teacher` と `Student` という 2 つのクラスを示します。両方のクラスは、`TeacherStudent` 関連によってリンクされています。`TeacherStudent` 関連には、次の 2 つの参照が存在します。

- `Teaches` は、`Teacher` クラスのインスタンスを参照するプロパティ
- `TaughtBy` は、 `Student` クラスのインスタンスを参照するプロパティ

関連メソッド

`CIMClient` 内の関連メソッドは、クライアントとインスタンス間の関連 (関係) に関する情報を返します。次の表では、これらのメソッドについて説明します。

表 **3–2** 関連メソッド

メソッド	説明
`associators`	指定された CIM クラスやインスタンスに関連付けられている CIM クラスやインスタンスを取得する
`associatorNames`	指定された CIM クラスやインスタンスに関連付けられている CIM クラスやインスタンスの名前を取得する
`references`	指定された CIM クラスやインスタンスを参照する関連クラスやインスタンスを取得する
`referenceNames`	指定された CIM クラスやインスタンスを参照する関連クラスやインスタンスの名前を取得する

これらのメソッドは、1 つの必須引数 `CIMObjectPath` を取ります。この引数には、検索する関連や、関連付けされたクラスまたはインスタンスを持つソースの CIM クラスまたは CIM インスタンスの名前を指定します。CIM Object Manager は、関連や関連づけされたクラスまたはインスタンスを検出できない場合、何も返しません。

- `CIMObjectpath` がクラスの場合、このメソッドは関連付けられたクラスとそのクラスのサブクラスを返します。
- `CIMObjectpath` がインスタンスの場合、このメソッドは関連付けされたクラスとそのサブクラスのインスタンスを返します。

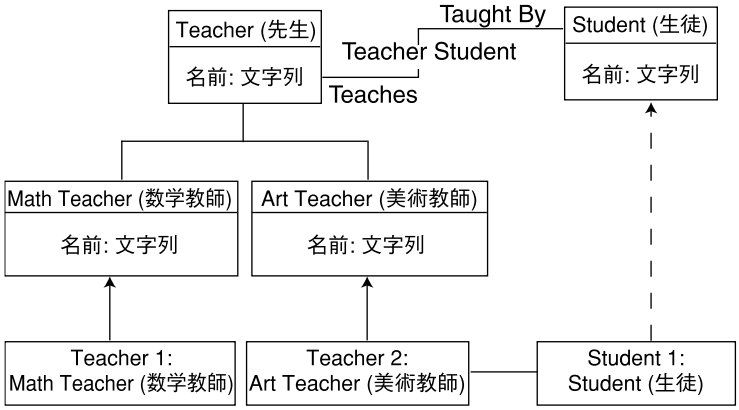

図 **3–2** TeacherStudent 関連 2

図 3–2では、`associators` および `associatorNames` メソッドが、`Teacher` および `Student` クラスに関連付けられたクラスの情報を返します。`references` および `referenceNames` メソッドは、`Teacher` および `Student` クラス間の関連に関する情報を返します。

表 **3–3** `TeacherStudent` メソッド

例	出力	説明
`associators(Teacher, null, null, null, null, false, false, null)`	`Student` クラス	関連付けられているクラスを返す。`Student` は、`TeacherStudent` 関連によって `Teacher` とリンクされている
`associators(MathTeacher, null, null, null, null, ,false, false, null)`	`Student`	関連付けられているクラスを返す。`Teacher` は、`TeacherStudent` 関連によって `Student` とリンクされている。`Math Teacher` と `Art Teacher` は、`Teacher` から `TeacherStudent` 関連を継承する
`associatorNames(Teacher, null, null, null, null)`	`Student` クラスの名前	関連付けられているクラスの名前を返す。`Student` は、`TeacherStudent` 関連によって `Teacher` とリンクされている
`references(Student , null, null. false, false, null)`	`TeacherStudent`	`Student` が関与している関連を返す

表 **3–3** TeacherStudent メソッド　(続き)

例	出力	説明
references(Teacher , null,null. false,false, null)	TeacherStudent	Teacher が関与している関連を返す
references(Teacher , null,null,false,false, null)	TeacherStudent	Teacher が関与している関連を返す
referenceNames(Teacher , null,null)	TeacherStudent クラスの名前	Teacher が関与している関連の名前を返す
referenceNames(Teacher , null,null)	TeacherStudent クラスの名前	Teacher が関与している関連の名前を返す

注 – associatorNames および referenceNames メソッドは、引数 *includeQualifiers*、*includeClassOrigin*、および *propertyList* を取りません。インスタンスまたはクラスの名前だけ (内容全体ではなく) を返すメソッドには、これらの引数は関係ないからです。

関連メソッドへのクラスの引き渡し

クラス名を指定する場合、そのモデルパスを指定します。モデルパスには、クラスの名前空間、クラス名、およびキーが含まれます。キーは、管理リソースを一意に識別するプロパティまたはプロパティのセットです。キープロパティは、key 修飾子によって示されます。次にモデルパスを示します。

```
\\myserver\\root\cimv2\Solaris_ComputerSystem.Name=
mycomputer: CreationClassName=Solaris_ComputerSystem
```

このモデルパスには、以下が指定されています。

- \\myserver\root\cimv2 – ホスト myserver 上のデフォルトの CIM 名前空間
- Solaris_ComputerSystem – インスタンスの派生元クラスの名前
- Name=mycomputer、CreationClassName=Solaris_ComputerSystem – 2 つのキープロパティ (key property=value 形式)

関連メソッドへのインスタンスの引き渡し

enumerateInstances メソッドを使用して、所定のクラスのすべてのインスタンスを返し、ループ構造を使用して各インスタンスを処理します。ループでは、各インスタンスを関連メソッドに渡すことができます。

例 **3–14** インスタンスの引き渡し

次の例では、op クラスとそのサブクラス内のインスタンスを列挙し、while ループを使って、各インスタンスを CIMObjectPath (op) にキャストし、 associators メソッドへの最初の引数として渡します。

```
{
    ...
    Enumeration e = cc.enumerateInstances(op, true);
    while (e.hasMoreElements()) {
        op = (CIMObjectPath)e.nextElement();
        Enumeration e1 = cc.associators(op, null, null,
                null, null, false, false, null);
    ...
    }
```

関連メソッドでのオプション引数の使用

関連メソッドでオプション引数を使用して、返されるクラスやインスタンスをフィルタ処理できます。すべてのパラメータが処理されるまで、オプションの各パラメータ値は、その結果をフィルタ処理のために次のパラメータに渡します。

任意の 1 つのオプションパラメータ、またはその組み合わせた値を渡すことができます。各パラメータには値または null を指定する必要があります。返されるクラスやインスタンスをフィルタ処理するパラメータには、assocClass、 resultClass、role、 resultRole があります。これらの引数を使用すると、これらのパラメータに指定された値と一致するクラスやインスタンスだけが返されます。返されるクラスやインスタンスに含まれている情報をフィルタ処理するパラメータには、includeQualifiers、includeClassOrigin、propertyList があります。

メソッドの呼び出し

プロバイダによってサポートされるクラス内のメソッドを呼び出すには、invokeMethod インタフェースを使用します。メソッドのシグニチャーを取得するには、最初にそのメソッドが属するクラスの定義を取得する必要があります。invokeMethod メソッドは CIMValue を返します。呼び出したメソッドが戻り値を定義していない場合には、戻り値は null です。

invokeMethod インタフェースは、次の表に示す 4 つの引数を受け取ります。

表 **3–4** invokeMethod パラメータ

パラメータ	データ型	説明
name	CIMObjectPath	インスタンス名。このインスタンスでメソッドを呼び出す

表 **3–4** invokeMethod パラメータ　(続き)

パラメータ	データ型	説明
methodName	String	呼び出すメソッド名
inParams	Vector	メソッドに渡す入力パラメータ
outParams	Vector	メソッドから受け取る出力パラメータ

例 **3–15** メソッドの呼び出し

次の例では、CIM_Service クラスのインスタンス (デバイスやソフトウェアの機能を管理するサービス) を取得してから、invokeMethod メソッドを使って各サービスを停止します。

```
{
    ...
    /* CIM_Service クラスの
CIM オブジェクトパスを CIM Object Manager に渡す。
このクラスで定義されたメソッドを呼び出す */

    CIMObjectPath op = new CIMObjectPath("CIM_Service");

    /* CIM Object Manager は インスタンスオブジェクトパスの列挙、
CIM_Service クラスのインスタンス名を]
返す */

    Enumeration e = cc.enumerateInstanceNames (op, true);

    /* インスタンスオブジェクトパスの列挙を繰り返し処理する */

    while(e.hasMoreElements()) {
                // インスタンスを取得する
                CIMObjectPath op = (CIMObjectPath) e.nextElement();
                //Stop Service メソッドを呼び出して CIM サービスを停止する
                cc.invokeMethod("StopService", null, null);
              }
}
```

クラス定義の取得

CIM クラスを取得するには getClass メソッドを使用します。クラスを作成すると、その派生元クラスとそのクラス階層のすべての親クラスのメソッドとプロパティを継承します。getClass メソッドは、ブール値引数 *localOnly* を受け取ります。

- *localOnly* が true の場合、getClass は継承されていないプロパティおよびメソッドを返す
- *localOnly* が false の場合、 getClass はクラス内のすべてのプロパティを返す

例 **3–16** クラス定義の取得

このプログラム例は、次のメソッドを使用してクラス定義を取得します。

- `CIMNameSpace` – 新しい名前空間を作成する
- `CIMClient` – CIM Object Manager への新しいクライアント接続を作成する
- `CIMObjectPath` – オブジェクトパス (取得するクラス名を含むオブジェクト) を作成する
- `getClass` – CIM Object Manager からクラスを取得する

```
import java.rmi.*;
import javax.wbem.client.CIMClient;
import javax.wbem.cim.CIMInstance;
import javax.wbem.cim.CIMValue;
import javax.wbem.cim.CIMProperty;
import javax.wbem.cim.CIMNameSpace;
import javax.wbem.cim.CIMObjectPath;
import javax.wbem.cim.CIMClass;
import javax.wbem.cim.CIMException;
import java.util.Enumeration;
/**
 * コマンド行で指定されたクラスを取得する。作業を
 * デフォルトの名前空間 root\cimv2 で行う
 */
public class GetClass {
    public static void main(String args[]) throws CIMException {
      CIMClient cc = null;
      try {
         CIMNameSpace cns = new CIMNameSpace(args[0]);
         UserPrincipal up = new UserPrincipal("root");
         PasswordCredential pc = new PasswordCredential("root_password");
         cc = new CIMClient(cns);
         CIMObjectPath cop = new CIMObjectPath(args[1]);
             // 指定されたクラスに対してローカルなメソッド
             // とプロパティのみを返す (localOnly は true)
         cc.getClass(cop, true);
      } catch (Exception e) {
         System.out.println("Exception: "+e);
      }
      if(cc != null) {
         cc.close();
        }
    }
}
```

例外の処理

各 `CIMClient` メソッドは、`CIMException`、つまりエラー状態をスローします。CIM Object Manager は、Java の例外処理を使用して WBEM 固有の例外の階層を作成します。`CIMException` クラスは、CIM 例外の基底クラスです。`CIMException` 以外の CIM 例外クラスは、`CIMException` クラスのサブクラスです。

CIM 例外の各クラスは、API コードが処理する特定のエラー状態を定義します。CIMException は、エラーコードおよび例外に関連したパラメータを取得するメソッドを保持します。CIMException クラスの詳細は、/usr/sadm/lib/wbem/doc/index.html を参照してください。

名前空間の作成

Solaris WBEM SDK のインストールにより、標準の CIM (MOF) ファイルがデフォルトの名前空間にコンパイルされます。新しい名前空間を作成する場合は、その名前空間にオブジェクトを作成する前に適切な CIM .mof ファイルを新しい名前空間にコンパイルする必要があります。たとえば、標準の CIM 要素を使用するクラスを作成する場合は、名前空間に CIM コアスキーマをコンパイルします。CIM アプリケーションスキーマを拡張するクラスを作成する場合は、名前空間に CIM アプリケーションをコンパイルします。

例 **3–17** 名前空間の作成

次の例では、2 段階の処理を実行して、既存の名前空間内に名前空間を作成します。

1. 名前空間の作成時に、CIMNameSpace メソッドが CIM Object Manager に渡すパラメータを含む名前空間オブジェクトを作成します。
2. CIMClient クラスは、CIM Object Manager に接続して名前空間オブジェクトを渡します。CIM Object Manager は、名前空間オブジェクトに含まれるパラメータを使用して名前空間を作成します。

```
{
    ...
    /* クライアント上で名前空間オブジェクトを作成し、コマンド行から
渡されるパラメータを格納する。args[0] にはホスト名 (たとえば myhost) が、
args[1] には親の名前空間 (たとえば最上位ディレクトリ) が含まれる */

    CIMNameSpace cns = new CIMNameSpace (args[0], args[1]);

    UserPrincipal up = new UserPrincipal("root");
    PasswordCredential pc = new PasswordCredential("root_password");

    /* CIM Object Manager に接続し、3 つのパラメータを渡す。
3 つのパラメータとは、ホスト名 (args[0]) および親の名前空間名 (args[1])
を含む名前空間オブジェクト (cns)、ユーザー名文字列 (args[3])、
およびパスワード文字列 (args[4]) を指す */

    CIMClient cc = new CIMClient (cns, up, pc);

    /* CIM Object Manager に、NULL 文字列 (ホスト名) と args[2]の
子名前空間の名前 (たとえば secondlevel) を含む
別の名前空間オブジェクトを渡す */

    CIMNameSpace cop = new CIMNameSpace("", args[2]);

    /* myhost の最上位の名前空間に args[2] として渡された名前で、
```

例 **3–17** 名前空間の作成　(続き)

```
新しい名前空間を作成する /*

    cc.createNameSpace(cop);
    ...
}
```

名前空間の削除

名前空間を削除するには、`deleteNameSpace` メソッドを使用します。

基底クラスの作成

注 – MOF 言語を使用して基底クラスを作成することもできます。MOF の構文に慣れている場合は、テキストエディタを使用して MOF ファイルを作成し、次に MOF コンパイラを使用してそのファイルを Java クラスにコンパイルします。詳細は、第 2 章を参照してください。

CIM クラスを表す Java クラスを作成するには、`CIMClass` クラスを使用します。ほとんどの基底クラスは、クラス名およびキープロパティまたは abstract 修飾子を指定するだけで宣言できます。ただし、ほとんどのクラスには、クラスのデータを表すプロパティが含まれます。プロパティを宣言するには、プロパティのデータ型、名前、およびオプションのデフォルト値を含めます。プロパティのデータ型は、`CIMDataType` のインスタンスである必要があります。

プロパティには、キープロパティであることを示すキー修飾子を指定できます。キープロパティは、クラスのインスタンスを一意に定義します。インスタンスを持てるのは、キーが指定されたクラスだけです。そのため、キープロパティが定義されないクラスは、abstract クラスとしてしか使用できません。新しい名前空間内でクラスにキープロパティを定義する場合は、最初にコア MOF ファイルをその名前空間にコンパイルする必要があります。コア MOF ファイルには、標準の CIM 修飾子 (キー修飾子など) の宣言が含まれます。

クラス定義は、別名、修飾子、修飾子フレーバなどの MOF 機能が含まれることにより複雑化します。

クラスの削除

クラスを削除するには、`CIMClient` の `deleteClass` メソッドを使用します。クラスを削除すると、 `CIMException` がスローされます。

注 – 基底クラスを削除する場合、最初に既存のサブクラスまたはインスタンスをすべて削除しておく必要があります。

例 **3–18** クラスの削除

このプログラム例は、deleteClass メソッドを使用して、デフォルトの名前空間 root\cimv2 にあるクラスを削除します。このプログラムは、4 つの必須文字列引数 (*hostname*、*classname*、*username*、および *password*) を取ります。このプログラムを実行するユーザーは、root\cimv2 名前空間への書き込み権を持つアカウントのユーザー名とパスワードを指定する必要があります。

```
import javax.wbem.cim.CIMClass;
import javax.wbem.cim.CIMException;
import javax.wbem.cim.CIMNameSpace;
import javax.wbem.cim.CIMObjectPath;
import javax.wbem.client.CIMClient;
import javax.wbem.client.UserPrincipal;
import javax.wbem.client.PasswordCredential;

import java.rmi.*;
import java.util.Enumeration;

/**
 * コマンド行字で指定されたクラスを削除する。
 * デフォルトの名前空間 root\cimv2 で作業を行う
 */
public class DeleteClass {
    public static void main(String args[]) throws CIMException {
       CIMClient cc = null;
       // 4 つの引数が指定されていない場合、使用法を表示して終了する
       if (args.length != 4) {
           System.out.println("Usage: DeleteClass host className " +
                              "username password");
           System.exit(1);
       }
       try {
          // args[0] にはホスト名が含まれる。指定されたホスト上の
          // デフォルトの名前空間を指す CIMNameSpace (cns) を作成する
          CIMNameSpace cns = new CIMNameSpace(args[0]);

          // args[2] と args[3] には、ユーザー名およびパスワードが含まれる。
          // ユーザー名を使用して UserPrincipal (up) を、
          // パスワードを使用して PasswordCredential を作成する
          UserPrincipal up = new UserPrincipal(args[2]);
          PasswordCredential pc = new PasswordCredential(args[3]);

          cc = new CIMClient(cns, up, pc);

          // クラス名 (args[4]) を取得し、CIMObjectPath を作成する
          CIMObjectPath cop = new CIMObjectPath(args[1]);
```

例 **3–18** クラスの削除　(続き)

```
            // クラスを削除する
            cc.deleteClass(cop);
    }
    catch (Exception e) {
            System.out.println("Exception: "+e);
    }
    if (cc != null) {
        cc.close();
        }
     }
}
```

アクセス制御の設定

ユーザーまたは名前空間ごとに、アクセス制御を設定できます。次のアクセス制御クラスは、root\security 名前空間に格納されます。

- Solaris_Acl – Solaris アクセス制御リスト (ACL) の基底クラス。このクラスは、文字列プロパティ *capability* を定義し、そのデフォルト値を r (読み取り専用) に設定する
- Solaris_UserAcl – ユーザーが、指定された名前空間内の CIM オブジェクトに対して保持するアクセス制御を示す
- Solaris_NamespaceAcl – 名前空間に対するアクセス制御を示す

Solaris_UserACL クラスのインスタンスを作成し、そのインスタンスのアクセス権を変更することにより、名前空間内の CIM オブジェクトへの各ユーザーのアクセス制御を設定できます。同様に、Solaris_NameSpaceACL クラスのインスタンスを作成し、createInstance メソッドを使用してそのインスタンスへのアクセス権を設定することにより、名前空間へのアクセス制御を設定できます。

これら 2 つのクラスを結び付けて使用する場合、Solaris_NameSpaceACL クラスを最初に使用して、名前空間内のオブジェクトへのすべてのユーザーのアクセスを制限するのは効果的な方法です。次に、Solaris_UserACL クラスを使用して、選択したユーザーに名前空間へのアクセスを許可します。

Solaris_UserAcl クラス

Solaris_UserAcl クラスは、Solaris_Acl 基底クラスを拡張したクラスです。Solaris_UserAcl クラスは、 Solaris_Acl 基底クラスから文字列プロパティ *capability* をデフォルト値 r (読み取り専用) で継承します。*capability* プロパティを次のいずれかの値に設定することで、アクセス権を指定できます。

アクセス権	説明
r	読み取り
rw	読み取りおよび書き込み
w	書き込み
なし	アクセス権なし

Solaris_UserAcl クラスは、次のキープロパティを定義します。名前空間内には、名前空間とユーザー名 ACL のペアのインスタンスを 1 つだけ入れることができます。

プロパティ	データ型	目的
nspace	string	ACL を適用する名前空間を示す
username	string	ACL を適用するユーザーを示す

▼ ユーザーのアクセス制御設定

1. Solaris_UserAcl クラスのインスタンスを作成します。次に例を示します。

```
...
/* root\security
(名前空間の名前) を使用して初期化した名前空間オブジェクトを
ローカルホスト上に作成する */

CIMNameSpace cns = new CIMNameSpace("", "root\security");

// root\security 名前空間にスーパーユーザーとして接続する
cc = new CIMClient(cns, user, user_passwd);

// Solaris_UserAcl クラスを取得する
cimclass = cc.getClass(new CIMObjectPath("Solaris_UserAcl");

// Solaris_UserAcl の新しいインスタンスを作成する
class ci = cimclass.newInstance();
...
```

2. *capability* プロパティを目的のアクセス権に設定します。次に例を示します。

```
...
/* root\molly 名前空間内のオブジェクトに対するユーザー Guest の
アクセス権 (capability) を読み取りおよび書き込みに変更する */
ci.setProperty("capability", new CIMValue(new String("rw"));
ci.setProperty("nspace", new CIMValue(new String("root\molly"));
ci.setProperty("username", new CIMValue(new String("guest"));
...
```

3. インスタンスを更新します。次に例を示します。

```
...
// 更新されたインスタンスを CIM Object Manager に渡す
cc.createInstance(new CIMObjectPath(), ci);
...
```

Solaris_NamespaceAcl クラス

Solaris_NamespaceAcl クラスは、 Solaris_Acl 基底クラスを拡張したクラスです。Solaris_UserAcl クラスは、 Solaris_Acl 基底クラスから文字列プロパティ *capability* をデフォルト値 r (すべてのユーザーに対し読み取り専用) で継承します。Solaris_NamespaceAcl クラスは、次のキープロパティを定義します。

プロパティ	データ型	目的
nspace	string	アクセス制御リストを適用する名前空間を示す。名前空間内には、名前空間 ACL のインスタンスを 1 つだけ指定できる

▼ 名前空間のアクセス制御設定

1. Solaris_namespaceAcl クラスのインスタンスを作成します。次に例を示します。

```
...
/* root\security （名前空間の名前） を使用して
初期化した名前空間オブジェクトをローカルホスト上に作成する */
CIMNameSpace cns = new CIMNameSpace("", "root\security");

// root\security 名前空間にスーパーユーザーとして接続する
cc = new CIMClient(cns, user, user_passwd);

// Solaris_namespaceAcl クラスを取得する
cimclass = cc.getClass(new CIMObjectPath("Solaris_namespaceAcl");

// Solaris_namespaceAcl の新しいインスタンスを作成する
class ci = cimclass.newInstance();
...
```

2. *capability* プロパティを目的のアクセス権に設定します。次に例を示します。

```
...
/* root\molly 名前空間へのアクセス権 (capability) を
読み取りおよび書き込みに変更する */
ci.setProperty("capability", new CIMValue(new String("rw"));
ci.setProperty("nspace", new CIMValue(new String("root\molly"));
...
```

3. インスタンスを更新します。次に例を示します。

```
// 更新されたインスタンスを CIM Object Manager に渡す
cc.createInstance(new CIMObjectPath(), ci);
```

修飾子と修飾子のデータ型の処理

CIM 修飾子は、CIM クラス、インスタンス、プロパティ、メソッド、またはパラメータの特性を示す要素です。修飾子の属性は、次のとおりです。

- 型
- 値
- 名前

MOF 構文では、各 CIM 修飾子は、定義された CIM 修飾子データ型を 1 つ持ちます。修飾子には、その修飾子を使用可能な CIM 要素を示すスコープ属性はありません。スコープを定義できるのは、修飾子のデータ型宣言内だけです。修飾子内でスコープを変更することはできません。

次に、CIM 修飾子のデータ型を宣言する MOF 構文を示します。この文は、ブール型 (デフォルト値は false) を使用して修飾子のデータ型 `key` を定義します。このデータ型が記述できるのは、プロパティと、オブジェクトに対する参照だけです。`DisableOverride` フレーバは、このキー修飾子がそれらの値を変更できないことを意味します。

```
Qualifier Key : boolean = false, Scope(property, reference),
                    Flavor(DisableOverride);
```

次のコード例は、CIM 修飾子の MOF 構文を示します。このサンプル MOF ファイルでは、`key` と説明は、プロパティ `a` の修飾子です。プロパティのデータ型は、プロパティ名 `a` を持つ整数です。

```
{
[key, Description("test")]
int a;
};
```

CIM 修飾子の取得と設定

修飾子フレーバは、修飾子の使用を制御するフラグです。フレーバは、派生クラスとインスタンスに修飾子を継承できるかどうか、および派生クラスまたはインスタンスが修飾子の元の値をオーバーライドできるかどうかを指定する規則を記述します。

例 **3–19** CIM 修飾子の設定

次のコード例は、新しいクラスの CIM 修飾子のリストをそのスーパークラス内の修飾子に設定します。

例 **3–19** CIM 修飾子の設定　(続き)

```
{

 try {
     cimSuperClass = cimClient.getClass(new CIMObjectPath(scName));
        Vector v = new Vector();
        for (Enumeration e = cimSuperClass.getQualifiers().elements();
                            e.hasMoreElements();) {
CIMQualifier qual = (CIMQualifier)((CIMQualifier)e.nextElement()).clone();
        v.addElement(qual);
        }
        cimClass.setQualifiers(v);
 } catch (CIMException exc) {
          return;
        }
    }
}
...
```

クライアント要求のバッチ処理

複数の CIMClient API 呼び出しを単一のリモート呼び出しとしてバッチ処理することにより、複数のリモートメッセージ交換による遅延を軽減できます。BatchCIMClient クラスのインスタンスを使用して、バッチ要求内で実行する操作のリストを作成できます。次に、CIMClient クラスの performBatchOperations メソッドを使用して、CIM Object Manager に操作のリストを送信します。

注 – バッチ操作は、単一のトランザクションを意味するわけではありません。各操作は、バッチに含まれる他の操作とは独立しており、前に実行される操作が正常終了したかまたは失敗したかには影響されません。

BatchCIMClient クラスには、非バッチモードと同じ CIM 操作を実行できるメソッドが含まれます。これらのメソッドは、BatchCIMClient メソッドが CIMClient クラス内の等価の型と同じ型を返さないことを除き (バッチ操作の完了後に値がリストとして返されるため)、CIMClient メソッドと類似しています。このメソッドは、整数の操作 ID を返します。この ID を使用して、操作の結果をあとで取得できます。BatchCIMClient のメソッドが呼び出されると、BatchCIMClient オブジェクトは、後で実行される CIMOperation オブジェクトのリストを作成します。

クライアントが、CIMClient の performBatchOperations メソッドを呼び出すことによって、バッチ操作リストを実行すると、バッチ操作の結果を含む BatchResult オブジェクトが返されます。次に、クライアントは操作 ID を

BatchResult クラスの getResult に渡して、操作の結果を取得します。リスト内の操作により例外が発生すると、BatchResult オブジェクトに例外オブジェクトが埋め込まれます。失敗した操作 ID を使用する getResult メソッドを起動すると、getResult メソッドにより例外がスローされます。

例 **3–20** バッチ処理の例

次の例は、バッチ処理 API を使用して、1 つのリモート呼び出し内で複数の操作を実行する方法を示します。この例では、3 つの操作 (enumerateInstanceNames、getClass、および enumerateInstances) が単独のバッチ操作として実行されます。

```
import java.util.Enumeration;
import java.util.ArrayList;
import java.util.Vector;
import java.lang.String;

import javax.wbem.cim.*;
import javax.wbem.client.*;
import javax.wbem.client.UserPrincipal;
import javax.wbem.client.PasswordCredential;


public class TestBatch {
    public static void main(String args[]) throws CIMException {
           CIMClient cc = null;
           CIMObjectPath cop = null;
           String protocol = CIMClient.CIM_RMI;
           if (args.length < 4) {
               System.out.println("Usage: TestBatch host user passwd
                                  classname " + "[rmi|http]");
               System.exit(1);
               }
           try {
               CIMNameSpace cns = new CIMNameSpace(args[0]);

               UserPrincipal up = new UserPrincipal(args[1]);
               PasswordCredential pc = new PasswordCredential(args[2]);
               if (args.length == 5 && args[4].equalsIgnoreCase("http")) {
                       protocol = CIMClient.CIM_XML;
               }
               cc = new CIMClient(cns, up, pc, protocol);

               CIMObjectPath op = new CIMObjectPath(args[3]);

               BatchCIMClient bc = new BatchCIMClient();
               int[] ids = new int[3];

               ids[0] = bc.enumerateInstanceNames(op);
               ids[1] = bc.getClass(op, false, true, true, null);
               ids[2] = bc.enumerateInstances(op, true, false, false,
                                              false, null);

               BatchResult br = cc.performBatchOperations(bc);
```

例 **3–20** バッチ処理の例　(続き)

```
            Enumeration instanceNames = (Enumeration)br.getResult
                                        (ids[0]);
                CIMClass cl = (CIMClass)br.getResult(ids[1]);
                Enumeration instances = (Enumeration)br.getResult(ids[2]);

                while (instanceNames.hasMoreElements()) {
                    System.out.println((CIMObjectPath)instanceNames.
                                       nextElement());
                }

                System.out.println(cl.toMOF());

                while (instances.hasMoreElements()) {
                    System.out.println((CIMInstance)instances.
                                       nextElement());
                }

            }
            catch (Exception e) {
                e.printStackTrace();
                System.out.println("Exception: "+e);
            }

            // セッションを閉じる
            if (cc != null) {
                cc.close();
            }
    }
}
```

CIM イベントの処理

注 – CIM インジケーション、および CIM インジケーションを使用してイベントの発生を通知する方法については、 http://www.dmtf.org/education/whitepapers.php の CIM Schema White Papers を参照してください。

「イベント」とは、1 つの発生した事象です。「インジケーション」とは、イベントの発生通知です。CIM (Common Information Model) では、発行されるのはインジケーションであり、イベントではありません。イベントの発生時に、プロバイダはインジケーションを生成します。

インジケーションは、状態の変化を認識する 0 以上の「トリガー」を保持します。WBEM は、トリガーを表す明示的なオブジェクトを保持しません。その代わり、トリガーは以下により暗黙的に示されます。

- システムの基本オブジェクトに関する操作 — クラスの作成、削除、変更、アクセスや、インスタンスの変更またはアクセス
- 管理された環境内で発生するイベント

たとえば、サービスが終了してトリガーが機能すると、このイベントによりサービスの終了を通知するインジケーションが生成されます。

Solaris WBEM サービススキーマ内の関連する CIM イベントクラスは、`/usr/sadm/lib/wbem/doc/mofhtml/index.html` で確認できます。クラスは、次のように構築されます。

表 **3–5** `CIM_Indication` クラス構造

ルートクラス	スーパークラス	サブクラス
`CIM_Indication`	`CIM_ClassIndication`	`CIM_ClassCreation`、`CIM_ClassDeletion`、 `CIM_ClassModification`
	`CIM_InstIndication`	`CIM_InstCreation`、 `CIM_InstDeletion`、`CIM_InstMethodCall`、 `CIM_InstModification`、`CIM_InstRead`
	`CIM_ProcessIndication`	`CIM_AlertIndication`、`CIM_AlertInstIndication`、`CIM_ThresholdIndication`、`CIM_SNMPTrapIndication`

インジケーションについて

CIM イベントは、ライフサイクルまたはプロセスとして分類できます。「ライフサイクルイベント」は、データの変更に対応して発生する組み込みの CIM イベントであり、その内部で、クラスの作成、変更、削除、クラスインスタンスの作成、変更削除、読み取り、またはメソッド呼び出しが行われます。「プロセスイベント」は、ライフサイクルイベントに含まれないユーザー定義のイベントです。

イベントプロバイダは、CIM Object Manager により作成された要求に応答してインジケーションを生成します。CIM Object Manager は予約要求を分析し、`EventProvider` や `CIMIndicationProvider` インタフェース、あるいはその両方を使用してプロバイダと通信し、適切なインジケーションの生成を要求します。プロ

バイダがインジケーションを生成すると、CIM Object Manager は CIM _IndicationHandler インスタンスにより指定された宛先にインジケーションを配信します。これらのインスタンスは、予約者により作成されます。

イベントプロバイダは、インスタンスプロバイダと同じ方法で検出されます。インスタンスのライフサイクルインジケーション (CIM_InstIndication のサブクラス) に属する予約の場合、CIM Object Manager が予約の有効範囲内のクラスを判別すると、これらのクラスのインスタンスプロバイダに対し通信を行います。プロセスインジケーションの場合、CIM Object Manager は Provider 修飾子を使用して、適切なプロバイダと通信を行います。

CIM Object Manager および CIM Object Manager Repository は、次の条件下でインジケーションを処理します。

- プロバイダがインジケーションをサポートしないか、CIM Object Manager にポーリングを指示した場合、CIM Object Manager は、CIM_InstMethodCall、 CIM _InstModification、CIM_InstDeletion、および CIM_InstCreation イベントを処理する
- CIM Object Manager Repository は、プロバイダを保持しないクラスインジケーションおよびライフサイクルインジケーションをすべて処理する。これらのクラスには、CIM_ClassCreation、CIM_ClassDeletion、 CIM _ClassModification、CIM_InstCreation、 CIM _InstModification、CIM_InstDeletion、および CIM_InstRead が含まれる

上記の場合、プロバイダはインジケーションを生成しないか、EventProvider インタフェースを実装します。また、プロバイダは、イベントの生成機能を CIM Object Manager に委託することもできます。CIM Object Manager は、プロバイダに対して enumerateInstances を呼び出して、以前の状態のスナップショットを現在の状態と比較し、インスタンスが作成、変更、または削除されたかどうかを判断します。

注 – ポーリングを使用すると大幅なオーバーヘッドが発生するため、ほとんどの場合、プロバイダが独自のインジケーションを処理する必要があります。インジケーションを生成する場合、プロバイダ自体がポーリングを行う必要があります。このとき、プロバイダはタスクを CIM Object Manager に委託できます。

プロバイダが EventProvider インタフェースを実装する場合、CIM Object Manager はインタフェース内のメソッドを呼び出し、その応答に応じて操作を実行します。特定のプロバイダが予約要求に参加する必要があると CIM Object Manager が判断すると、次の順序でメソッドが呼び出されます。

1. mustPoll - CIM_InstCreation、CIM_InstDeletion、および CIM _InstModification に対応して CIM Object Manager により呼び出され、CIM Object Manager によるポーリングをプロバイダが望んでいるかどうかを判断します。プロバイダが EventProvider インタフェースを実装していない場合、CIM Object Manager はデフォルトでポーリングを実行するとみなします。

2. `authorizeFilter` - プロバイダが `Authorizable` インタフェースを実装する場合、CIM Object Manager によりこのメソッドが呼び出され、予約が承認されるかどうかを判断します。プロバイダは、インジケーションハンドラの所有者のユーザー ID (インジケーションを受信するユーザー)、または予約を作成したユーザーに基づいて、決定を行います。

   プロバイダが `Authorizable` インタフェースを実装しない場合、CIM Object Manager は、名前空間に対してデフォルトの読み取り承認検査を実行します。

   プロバイダが `EventProvider` インタフェースを実装せず、CIM Object Manager がポーリングを試みる場合、プロバイダに対する `enumerateInstances` が正常終了すると、承認が正しく行われます。

3. `activateFilter` - 承認が正しく行われ、プロバイダがポーリングを要求しない場合に、CIM Object Manager により呼び出されます。
4. `deActivateFilter` - 予約が予約者または CIM Object Manager により削除される場合 (宛先ハンドラが正常に機能しない場合など) に呼び出されます 。

予約について

クライアントアプリケーションでは、CIM イベントが通知されるように予約することができます。「予約」は、1 つまたは複数の一連のインジケーションを宣言することによって行います。現在は、プロバイダがイベントインジケーションを予約することはできません。

CIM イベントのインジケーションを予約するアプリケーションには、 次の指定を行います。

- 予約対象のインジケーション
- CIM Object Manager がインジケーションを送信するハンドラ

イベントの発生は、`CIM_Indication` クラスのいずれかのサブクラスのインスタンスとして表されます。インジケーションは、そのイベントがクライアントによって予約されているときだけ生成されます。

▼ 予約の作成

アプリケーションは、1 つまたは複数のイベントフィルタと 1 つまたは複数のイベントハンドラを作成できます。イベントインジケーションは、アプリケーションがイベントの予約を作成するまで送信されません。

1. `CIM_Listener` のインスタンスを作成します。詳細は、「**CIM** リスナーの追加」を参照してください。
2. `CIM_IndicationFilter` のインスタンスを作成します。詳細は、「イベントフィルタの作成」を参照してください。
3. `CIM_IndicationHandler` のインスタンスを作成します。詳細は、「イベントハンドラの作成」を参照してください。

4. CIM_IndicationFilter を CIM_IndicationHandler にバインドします。詳細は、「イベントフィルタとイベントハンドラのバインド」を参照してください。

CIM リスナーの追加

CIM イベントのインジケーションを受信するには、最初に CIMClient に対して addCIMListener メソッドを呼び出して、CIMListener のインスタンスを CIMClient に追加します。

注 – CIMListener インタフェースは、indicationOccured メソッドを実装する必要があります。このメソッドは、引数として CIMEvent を取ります。インジケーションが送信可能な場合、このメソッドが呼び出されます。

例 **3–21** CIM リスナーの追加

```
// CIM Object Manager に接続する
cc = new CIMClient();

// CIM リスナーを登録する
cc.addCIMListener(
new CIMListener() {
    public void indicationOccured(CIMEvent e) {
    }
});
```

イベントフィルタの作成

イベントフィルタでは、送信するイベントの種類と、どのような条件下で送信するかを指定します。CIM_IndicationFilter クラスのインスタンスを作成し、そのプロパティの値を定義することによって、イベントフィルタを作成します。各イベントフィルタは、そのフィルタが属する名前空間に属するイベントに対してのみ有効です。

CIM_IndicationFilter クラスは、一意のフィルタ識別、照会文字列の指定、照会文字列を構文解析する照会言語の指定が可能な文字列プロパティを保持します。現在は、WQL (WBEM Query Language) だけがサポートされます。

表 3–6 CIM_IndicationFilter プロパティ

プロパティ	説明	必須/任意
SystemCreationClassName	このフィルタを作成するクラスがあるシステム名、またはこのクラスが適用されるシステム名	任意。値は、CIM Object Manager により決定される
SystemName	このフィルタがあるシステム名、またはこのフィルタが適用されるシステム名	任意。このキープロパティのデフォルトは、CIM Object Manager が動作しているシステム名
CreationClassName	このフィルタの作成に使用するクラス名またはサブクラス名	任意。CIM Object Manager は、このキープロパティのデフォルトとして CIM_IndicationFilter を割り当てる
Name	フィルタの固有名	任意。CIM Object Manager は一意の名前を割り当てる
SourceNamespace	CIM インジケーション生成元であるローカル名前空間へのパス	任意。デフォルトは null
Query	インジケーションをどのような条件のときに生成するかを定義する照会式。現在は、Level 1 の WQL 式だけがサポートされる。WQL 照会式の詳細は、第 5 章を参照	必須
QueryLanguage	照会を表現する言語	必須。デフォルトは WQL

▼ イベントフィルタの作成

1. CIM_IndicationFilter クラスのインスタンスを作成します。

```
CIMClass cimfilter = cc.getClass
        (new CIMObjectPath("CIM_IndicationFilter"),
         true, true, true, null);
CIMInstance ci = cimfilter.newInstance();
```

2. イベントフィルタ名を指定します。

```
Name = "filter_all_new_solarisdiskdrives"
```

3. **WQL** 文字列を作成し、返されるイベントインジケーションを指定します。

```
String filterString = "SELECT *
        FROM CIM_InstCreation WHERE sourceInstance
        ISA Solaris_DiskDrive";
```

4. cimfilter インスタンスの各プロパティ値に、フィルタ名、**CIM** イベントを選択するフィルタ文字列、照会文字列を解析する照会言語 **(WQL)** を設定します。

```
ci.setProperty(“Name”, new
        CIMValue("filter_all_new_solarisdiskdrives”));
ci.setProperty("Query", new CIMValue(filterString));
ci.setProperty("QueryLanguage", new CIMValue("WQL");)
```

5. cimfilter インスタンスを filter という名前で作成し、**CIM Object Manager Repository** 内に格納します。

```
CIMObjectPath filter = cc.createInstance(new
                                                CIMObjectPath(), ci);
```

例 **3–22** イベントフィルタの作成

```
CIMClass cimfilter = cc.getClass(new CIMObjectPath
                                 (“CIM_IndicationFilter”), true);
CIMInstance ci = cimfilter.newInstance();
// test_a クラスが名前空間内に存在するものとする
String filterString = "select * from CIM_InstCreation where sourceInstance
                       isa test_a"

ci.setProperty("query", new CIMValue(filterString));
CIMObjectPath filter = cc.createInstance(newCIMObjectPath(), ci);
```

イベントハンドラの作成

イベントハンドラは、CIM_IndicationHandler クラスのインスタンスです。 CIM_IndicationHandler クラスのインスタンス内でプロパティを設定して、ハンドラに一意の名前を付け、その所有者の UID を示します。CIM イベント MOF には、HTTP プロトコルを使ってクライアントアプリケーションに送信されるインジケーションの宛先を指定する CIM_IndicationHandlerCIMXML クラスが定義されています。Solaris イベント MOF は、Solaris_JAVAXRMIDelivery クラスを作成して CIM_IndicationHandler クラスを拡張し、RMI プロトコルを使用して CIM イベントインジケーションをクライアントアプリケーションへ送信します。RMI クライアントは、Solaris_JAVAXRMIDelivery クラスのインスタンスを作成して、RMI 送信の場所を設定する必要があります。

表 **3–7** CIM_IndicationHandler プロパティ

プロパティ	説明	必須/任意
SystemCreationClassName	このハンドラを作成するクラスがあるシステム名、またはこのクラスが適用されるシステム名	任意。CIM Object Manager により指定される

表 **3–7** CIM_IndicationHandler プロパティ　(続き)

プロパティ	説明	必須/任意
SystemName	このハンドラがあるシステム名、またはこのハンドラが適用されるシステム名。	任意。このキープロパティのデフォルト値は、CIM Object Manager が動作しているシステム名。
CreationClassName	このハンドラの作成に使用するクラスまたはサブクラス	任意。CIM Object Manager は、適切なクラス名をこのキープロパティのデフォルトとして割り当てる
Name	このハンドラの固有名	任意。クライアントアプリケーションは固有名を指定する必要がある
Owner	このハンドラを作成した、または保持するエンティティ名。プロバイダは、この値を検査して、インジケーションの受信をハンドラに承認するかどうかを判断できる	任意。デフォルトは、このインスタンスを作成するユーザーの Solaris ユーザー名

例 **3–23** イベントハンドラの作成

```
// Solaris_JAVAXRMIDelivery クラスのインスタンスを作成するか、
// ハンドラの適切なインスタンスを取得する
CIMInstance ci = cc.getIndicationHandler(null);

// 新しいインスタンス (delivery) を
// rmidelivery インスタンスから作成する
CIMObjectPath delivery = cc.createInstance(new CIMObjectPath(), ci);
```

イベントフィルタとイベントハンドラのバインド

CIM_IndicationSubscription クラスのインスタンスを作成することにより、イベントフィルタとイベントハンドラをバインドします。このクラスのインジケーションを作成すると、イベントフィルタにより指定されたイベントのインジケーションが送信されます。

次の例では、予約 (filterdelivery) を作成し、81 ページの「イベントフィルタの作成」で作成した filter オブジェクトパスに filter プロパティを定義します。また、例 3–23 で作成した delivery オブジェクトパスに handler プロパティを定義します。

例 **3–24** イベントフィルタとイベントハンドラのバインド

```
CIMClass filterdelivery = cc.getClass(new
        CIMObjectPath("CIM_IndicationSubscription"),
        true, true, true, null);
```

例 **3–24** イベントフィルタとイベントハンドラのバインド　(続き)

```
ci = filterdelivery.newInstance():

// フィルタインスタンスを参照する filter という名のプロパティを作成する
ci.setProperty("filter", new CIMValue(filter));

// 送信インスタンスを参照する handler という名のプロパティを作成する
ci.setProperty("handler", new CIMValue(delivery));

CIMObjectPath indsub = cc.createInstance(new CIMObjectPath(), ci);
```

ログメッセージの読み取りと書き込み

Solaris MOF には、ログクラスが含まれます。クライアントは、これらのクラスを使用してエラー、警告、および情報メッセージをログに記録し、読み取ることができます。たとえば、ログメッセージから、システムがシリアルポートにアクセスできなくなった日時、システムがファイルシステムを正常にマウントした日時、またシステムで稼働するプロセス数が許可された数を超えた日時などを知ることができます。

ログクラスの基盤となるプロバイダは、ログ要求を `syslog` デーモン (Solaris オペレーティング環境のデフォルトログシステム) に転送できます。詳細は、`syslogd(1M)` のマニュアルページを参照してください。

ログファイルについて

WBEM ログメッセージは、`/var/sadm/wbem/log` ディレクトリ内の個々のログファイルに格納されます。ログファイル名、ログファイルの格納ディレクトリ、ログファイルのサイズ制限、格納するログファイルの数、メッセージを `syslogd(1M)` に転送するかどうかは、`Solaris_LogServiceProperties` クラスの `singleton` インスタンスを使用して指定するプロパティです。

各ログエントリの形式は、`CIM_LogRecord` のサブクラスである `Solaris_LogEntry` クラスによって定義されます。`Solaris_LogEntry` は `Solaris_Device1.0.mof` 内に、`CIM_LogRecord` は `CIM_Device26.mof` 内に存在します。

ログメッセージには、次の要素が含まれます。

表 **3–8** ログメッセージの要素

要素	説明
Category	メッセージの種類 – アプリケーション、システム、またはセキュリティ
Severity	状況の重大性 – 警告またはエラー
Application	ログメッセージを書き込んだアプリケーション (またはプロバイダ) の名前
User	ログメッセージの生成時にアプリケーションを使用していたユーザー名
Client Machine	ログメッセージの生成時にユーザーが使用していたシステム名および IP アドレス
Server Machine	ログメッセージの原因となったイベントが発生したシステム名
Summary Message	イベントの概要説明
Detailed Message	イベントの詳細説明
Data	イベントをより把握するための状況説明
SyslogFlag	メッセージを syslogd(1M) に送信するかどうかを指定するブール値のフラグ

例 **3–25** Solaris_LogEntry のインスタンス作成

このプログラム例では、Solaris_LogEntry のインスタンスを作成し、そのインスタンスを設定します。

```
public class CreateLog {
    public static void main(String args[]) throws CIMException {

        // 渡されたコマンド行引数が不足している場合は、
        // 使用法を表示する
        if (args.length < 3) {
            System.out.println("Usage: CreateLog host username password
                               " + "[rmi|http]");
            System.exit(1);
        }

        String protocol = CIMClient.CIM_RMI;
        CIMClient cc = null;
        CIMObjectPath cop = null;
        BufferedReader d = new BufferedReader(new InputStreamReader
                                              (System.in));

        String input_line = "";

        // 作成するレコード数をユーザーに問い合わせる
        System.out.print("How many log records do you want to write? ");
```

例 **3–25** Solaris_LogEntry のインスタンス作成　(続き)

```
int num_recs = 0;

try {
        num_recs = Integer.parseInt(d.readLine());
} catch (Exception ex) {
        ex.printStackTrace();
        System.exit(1);
}

// try-catch ブロックのオーバーアーチ
try {
    CIMNameSpace cns = new CIMNameSpace(args[0]);
    UserPrincipal up = new UserPrincipal(args[1]);
    PasswordCredential pc = new PasswordCredential(args[2]);

    // トランスポートプロトコルをデフォルトの RMI に設定する
    if (args.length == 4 && args[3].equalsIgnoreCase("http")) {
        protocol = CIMClient.CIM_XML;
    }

    cc = new CIMClient(cns, up, pc, protocol);

        Vector keys = new Vector();
        CIMProperty logsvcKey = null;


        // ログ記録の作成に必要な関連情報の
        // 入力をユーザーに求める

        System.out.println("Please enter the record Category: ");
        System.out.println("\t(0)application, (1)security,
                                              (2)system");
        logsvcKey = new CIMProperty("category");
        input_line = d.readLine();
        logsvcKey.setValue(new CIMValue(Integer.valueOf
                                    (input_line)));
        keys.addElement(logsvcKey);
        System.out.println("Please enter the record Severity:");
        System.out.println("\t(0)Informational, (1)Warning,
                                               (2)Error");
        logsvcKey = new CIMProperty("severity");
        input_line = d.readLine();
        logsvcKey.setValue(new CIMValue(Integer.valueOf
                        (input_line)));
        keys.addElement(logsvcKey);
        logsvcKey = new CIMProperty("Source");
        System.out.println("Please enter Application Name:");
        logsvcKey.setValue(new CIMValue(d.readLine()));
        keys.addElement(logsvcKey);
        logsvcKey = new CIMProperty("SummaryMessage");
        System.out.println("Please enter a summary message:");
        logsvcKey.setValue(new CIMValue(d.readLine()));
        keys.addElement(logsvcKey);
```

例 **3–25** Solaris_LogEntry のインスタンス作成　(続き)

```
                logsvcKey = new CIMProperty("DetailedMessage");
                System.out.println("Please enter a detailed message:");
                logsvcKey.setValue(new CIMValue(d.readLine()));
                keys.addElement(logsvcKey);
                logsvcKey = new CIMProperty("RecordData");
                logsvcKey.setValue(
                        new CIMValue("0xfe 0x45 0xae 0xda random data"));
                keys.addElement(logsvcKey);
                logsvcKey = new CIMProperty("SyslogFlag");
                logsvcKey.setValue(new CIMValue(new Boolean(true)));
                keys.addElement(logsvcKey);
                CIMObjectPath logreccop =
                        new CIMObjectPath("Solaris_LogEntry", keys);
                CIMClass logClass = cc.getClass(logreccop);
                CIMInstance ci = logClass.newInstance();
                ci.setClassName("Solaris_LogEntry");
                ci.setProperties(keys);
                // System.out.println(ci.toString());

                // 要求された数のレコードインスタンスを作成する
                for (int i = 0; i < num_recs; i++) {
                        cc.createInstance(logreccop, ci);
                }
        } catch (Exception e) {
            System.out.println("Exception: "+e);
                e.printStackTrace();
        }

        // セッションを閉じる
        if (cc != null) {
            cc.close();
        }
    }
}
```

例 **3–26** ログ記録リストの表示

このプログラム例では、ログ記録のリストを表示します。

```
public class ReadLog {
   public static void main(String args[]) throws CIMException {

        String protocol = CIMClient.CIM_RMI;

        // 渡されたコマンド行引数が不足している場合、
        // 使用法を表示する
        if (args.length < 3) {
            System.out.println("Usage: ReadLog host username password " +
                               "[rmi|http]");
            System.exit(1);
        }
```

例 **3–26** ログ記録リストの表示　(続き)

```
        CIMClient cc = null;
        CIMObjectPath cop = null;
        CIMObjectPath serviceObjPath = null;
        Vector inVec = new Vector();
        Vector outVec = new Vector();

        // try-catch ブロックのオーバーアーチ
        try {
            CIMNameSpace cns = new CIMNameSpace(args[0]);
            UserPrincipal up = new UserPrincipal(args[1]);
            PasswordCredential pc = new PasswordCredential(args[2]);

            // トランスポートプロトコルをデフォルトの RMI に設定する
            if (args.length == 4 && args[3].equalsIgnoreCase("http")) {
                protocol = CIMClient.CIM_XML;
            }

            cc = new CIMClient(cns, up, pc, protocol);

            cop = new CIMObjectPath("Solaris_LogEntry");

            //  Solaris_LogEntry クラスのインスタンスリストを列挙する
            Enumeration e = cc.enumerateInstances(cop, true, false,
                                                  false, false, null);

            // リストを繰り返し処理して、各プロパティを出力する
            for (; e.hasMoreElements(); ) {
                System.out.println("--------------------------------");
                CIMInstance ci = (CIMInstance)e.nextElement();
                System.out.println("Log filename : " +
                    ((String)ci.getProperty("LogName").getValue().
                                                        getValue()));
                int categ =
    (((Integer)ci.getProperty("Category").getValue().getValue()).
       intValue());
                if (categ == 0)
                    System.out.println("Category : Application Log");
                else if (categ == 1)
                    System.out.println("Category : Security Log");
                else if (categ == 2)
                    System.out.println("Category : System Log");
                int severity =
    (((Integer)ci.getProperty("Severity").getValue().getValue()).
       intValue());
                if (severity == 0)
                    System.out.println("Severity : Informational");
                else if (severity == 1)
                    System.out.println("Severity : Warning Log!");
                else if (severity == 2)
                    System.out.println("Severity : Error!!");
                System.out.println("Log Record written by :" +
    ((String)ci.getProperty("Source").getValue().getValue()));
                System.out.println("User : " +
```

例 **3–26** ログ記録リストの表示　(続き)

```
    ((String)ci.getProperty("UserName").getValue().getValue()));
                System.out.println("Client Machine : " +
    ((String)ci.getProperty("ClientMachineName").getValue().getValue()));
                System.out.println("Server Machine : " +
    ((String)ci.getProperty("ServerMachineName").getValue().getValue()));
                System.out.println("Summary Message : " +
    ((String)ci.getProperty("SummaryMessage").getValue().getValue()));
                System.out.println("Detailed Message : " +
    ((String)ci.getProperty("DetailedMessage").getValue().getValue()));
                System.out.println("Additional data : " +
    ((String)ci.getProperty("RecordData").getValue().getValue()));
                boolean syslogflag =
((Boolean)ci.getProperty("SyslogFlag").getValue().getValue()).
  booleanValue();
                if (syslogflag == true) {
                    System.out.println("Record was written to syslog");
                } else {
                    System.out.println("Record was not written to syslog");
                }
                System.out.println("--------------------------------");
            }
        } catch (Exception e) {
            System.out.println("Exception: "+e);
            e.printStackTrace();
        }

        // セッションを閉じる
        if (cc != null) {
            cc.close();
        }
    }
}
```

第 4 章

プロバイダプログラムの作成

この章では、プロバイダプログラムの作成方法について説明します。内容は次のとおりです。

- 91 ページの「プロバイダについて」
- 95 ページの「プロバイダインタフェースの実装」
- 105 ページの「プロバイダの作成」

注 – WBEM プロバイダ API (`javax.wbem.provider`) についての詳細は、`/usr/sadm/lib/wbem/doc/index.html` を参照してください。

プロバイダについて

プロバイダは、データにアクセスするために、ディスクドライブおよび CPU などの管理リソースと通信する特別なクラスです。プロバイダは、統合と解釈を行うためにデータを CIM Object Manager に送ります。WBEM リソースの固有のサブセットの管理作業を引き継ぐことにより、Solaris WBEM Services を調整する主 WBEM エージェントである CIM Object Manager を取得できます。プロバイダは、`javax.wbem.provider` API を使用してこのデータを転送します。CIM Object Manager Repository で使用できないアプリケーションからデータの要求を受け取ると、CIM Object Manager は、プロバイダインタフェースを使用して、その要求を適切なプロバイダに送ります。

Solaris ソフトウェアプロバイダは、ユーザー、グループ、別名、役割、ファイルシステム、ディスク、プロセス、複製ツール、ネットワーク構成、プロダクト登録、デバイスおよびシステム性能モニターなど、さまざまな領域に存在します。

プロバイダでは、クラス (インスタンスのテンプレートとして使用する) ではなく、「インスタンス」を作成、変更、および削除します。インスタンスは、永続的な記憶領域に設定できますが、動的に使用することもできます。

プロバイダは、独自のプロセスおよびメモリーを持ち、CIM Object Manager に委託された作業を実行しますが、CIM Object Manager では、WBEM の調整作業を実行するために各プロバイダの位置を把握する必要があります。MOF ファイルに新規または変更されたプロバイダを含めることにより、CIM Object Manager にプロバイダの情報を知らせることができます。MOF ファイルは、プロバイダがサポートするクラスおよびインスタンスを定義します。MOF ファイルを登録するには、`mofcomp (1M)` コマンドを使用します。

プロバイダは、次の処理を実行します。

- 管理アプリケーションにデータを提供する – 管理アプリケーションにより、CIM Object Manager Repository で使用できない管理リソースについてのデータが要求された場合、CIM Object Manager は要求をプロバイダに転送します。プロバイダは、管理リソースからデータにアクセスし、そのデータを CIM Object Manager に渡します。管理リソースから受け取ったデータがネイティブ形式 (C コードなど) である場合、プロバイダは、CIM Object Manager に渡す前にそのデータを Java CIM クラスにマップします。
- 管理リソースを制御する – 管理リソースを制御するために、管理アプリケーションがデータを CIM Object Manager に送ると、CIM Object Manager は、データを適切なプロバイダに渡します。管理リソースがネイティブ形式のデータを必要とする場合、データを渡す前にプロバイダは CIM クラスをリソースのネイティブ形式にマップします。

注 – プロバイダと CIM Object Manager は、同じコンピュータに置く必要があります。

プロバイダのデータソース

プロバイダは、次のソースからデータを取得します。

- 永続的でないデータ – プロバイダのメソッドが実行している場合にのみ存在するプロバイダクラスのローカル変数
- 永続的なメモリー (プロバイダに対してローカル) – プロバイダクラスの大域変数を作成するために使用する。このプロバイダメモリーは、CIM Object Manager を停止して再起動すると消去される。
- CIM Object Manager Repository – この永続的なメモリーは、Solaris WBEM Services をアンインストールすると消去される。CIM Object Manager からこのメモリーにアクセスするには、プロバイダが CIM Object Manager ハンドルおよび内部プロバイダを使用する必要がある。
- プロバイダによって管理されるファイルおよびデータベース (つまり動的なデータ) – プロバイダは、システムからデータを取得することにより、データを動的に生成できる。たとえば、プロバイダはシステム呼び出しを行なって、現在実行中のプロセス数を取得できる。

プロバイダの種類

プロバイダは扱うことができるサービス要求の種類によって分類されます。クライアントプログラムは、クライアント API を介して CIM Object Manager と通信 (および WBEM データにアクセス) します (`/usr/sadm/lib/wbem/doc/index.html` を参照)。CIM Object Manager は、プロバイダメソッドをクライアント API の対応するクライアントメソッドにマップします。ただし、引数リストおよび対応するメソッドの戻り値は次のように異なる場合があります。

- プロバイダがデータを CIM Object Manager Repository に格納する場合、CIM Object Manager のハンドル (95 ページの「プロバイダインタフェースの実装」を参照) を使用して CIM Object Manager Repository にアクセスします。CIM Object Manager はクライアント API のメソッドを呼び出します。
- プロバイダで、CIM Object Manager Repository にインスタンスまたは関連を作成する必要がある場合、内部プロバイダ (95 ページの「プロバイダインタフェースの実装」を参照) を使用します。内部プロバイダは、インスタンスのメソッドまたは WBEM の内部にあるアソシエータプロバイダを呼び出します。

使用するメソッドおよびクラスの引数リストおよび戻り値が正しいかどうか確認してください。

次の表に Solaris WBEM SDK プロバイダの種類を示します。

表 **4–1** プロバイダの種類

種類	クラス名	説明
インスタンス	`CIMInstanceProvider`	所定のクラスの動的なインスタンスを提供する。インスタンスの取得、列挙、変更、削除をサポートする
メソッド	`MethodProvider`	1 つ以上のクラスのメソッドを供給する
アソシエータ	`CIMAssociatorProvider`	動的な関連クラスのインスタンスを供給する
インジケーション	`EventProvider`	CIM イベントのインジケーションを処理する
承認	なし	マーカーインタフェースは、プロバイダが独自に承認検査を行うことを CIM Object Manger に知らせる

関連するメソッドを登録および実装することにより、単一のプロバイダを前述の 1 つ以上の種類のプロバイダとして使用できます。

プロバイダ名の命名規約

プロバイダは、単一の Java クラスに含めることができます。また各プロバイダをそれぞれのクラスに分けて保存することもできます。プロバイダ名は、クラスのプロバイダとなる Java クラスを識別します。現時点では、CIM Object Manager は、Java 言語で記述されたプロバイダのみサポートします。

プロバイダ名およびクラス名は、次の規約に従う必要があります。

- クラス名は、有効な CIM クラスである必要があります。つまり、接頭辞、アンダースコア、文字列の順に名前を指定します。

  たとえば、`green_apples` および `red_apples` は、有効な CIM クラス名ですが、`apples`、 `apples_`、および `_apples` は、有効なクラス名ではありません。
- MOF ファイルに指定するプロバイダ名は、プロバイダクラスファイルの名前と一致する必要があります。

  たとえば、`SimpleCIMInstanceProvider` はプロバイダ名、`Ex_SimpleCIMInstanceProvider` はクラス名です。

注 – Java 言語でプロバイダが記述されていることを CIM Object Manager に通知するために、「`java:`」をすべてのプロバイダ修飾子の前に付ける必要があります。

標準の Java クラスおよびパッケージ命名規則に従って、プロバイダ名を作成します。パッケージ名の接頭辞は、小文字の ASCII 文字で最上位のドメイン名 (`com`、`edu`、`gov`、 `mil`、`net`、`org`) 、または ISO 標準 3166、1981 で指定されている英語 2 文字の国名コードにする必要があります。

パッケージ名のあとに続く名前は、組織内部の命名規則によって異なります。組織内部の規則では、ディレクトリ名のコンポーネントとして、部名、課名、プロジェクト名、マシン名、あるいはログイン名などを指定します。たとえば、プロバイダ名 `java:com.sun.wbem.cimom` は、次の要素を示します。

- `java:` – プロバイダを記述した言語
- `com` – 最上位のドメイン名
- `sun` – 会社名
- `wbem` – 製品名
- `cimom` – CIM Object Manager を実装したクラスファイルの種類

プロバイダインタフェースの実装

プロバイダの作成時には、プロバイダがサポートするインタフェースを指定する必要があります。プロバイダがサポートする各インタフェースのすべてのメソッドを実装してください。またすべてのプロバイダには、 `CIMProvider` インタフェースを実装する必要があります。プロバイダインタフェースには、次の 2 つのメソッドがあります。

- `initialize(CIMOMHandle` *cimom*) – プロバイダが CIM Object Manager Repository にデータを保存すると、渡された CIM Object Manager ハンドルを CIM Object Manager ハンドルに割り当てる必要があります。プロバイダはこのハンドルを使用して CIM Object Manager と通信します。たとえば、次のようになります。

```
        private CIMOMHandle cimom = null;
              ...
            public void initialize(CIMOMHandle cimom) throws CIMException
 {
              this.cimom = (CIMOMHandle) cimom;
```

  プロバイダは、インスタンスを作成したり、CIM Object Manager Repository の関連を操作したりします。プロバイダは、渡された CIM Object Manager ハンドルをサブクラス `ProviderCIMOMHandle` にキャストしたあと、内部インスタンスまたは関連プロバイダを取得します。たとえば、次のようになります。

```
        private ProviderCIMOMHandle cimom = null;
              private CIMAssociatorProvider ap = null;
              ...
            public void initialize(CIMOMHandle cimom) throws CIMException
 {
              this.cimom = (ProviderCIMOMHandle) cimom;
              ap = pcimom.getInternalProvider();
```

  注 – `initialize` コマンドは、CIM Object Manager の再起動後、プロバイダが初期化されるたびに自動的に実行されます。

- `cleanup()` – 現時点では、可変部分として使用します。

インスタンスプロバイダの作成

次のコード例は、 `Ex_SimpleCIMInstanceProvider` クラスの `enumerateInstances` および `getInstance` インタフェースを実装します。わかりやすくするために、この例では、`CIMException` をスローすることによって `deleteInstance`、`createInstance`、`setInstance`、`execQuery` の各インタフェースを実装します。

注 – execQuery メソッドの実装についての詳細は、113 ページの「照会の構文解析」を参照してください。

例 **4–1** CIMInstance プロバイダ

```
/*
 * "@(#)SimpleCIMInstanceProvider.java"
 */
import javax.wbem.cim.*;
import javax.wbem.client.*;
import javax.wbem.provider.CIMProvider;
import javax.wbem.provider.CIMInstanceProvider;
import javax.wbem.provider.MethodProvider;
import java.util.*;
import java.io.*;

public class SimpleCIMInstanceProvider implements CIMInstanceProvider{
     static int loop = 0;
     public void initialize(CIMOMHandle cimom) throws CIMException {
     }
     public void cleanup() throws CIMException {
     }
     public CIMObjectPath[] enumerateInstanceNames(CIMObjectPath op,
                                                   CIMClass cc)
     throws CIMException {
               return null;
     }
     /*
      * enumerateInstances:
      * 名前だけでなくインスタンス全体が返される
      */
     public CIMInstance[] enumerateInstances(CIMObjectPath op,
                          boolean localOnly,boolean includeQualifiers,
                          boolean includeClassOrigin,String[]
                          propertyList, CIMClass cc) throws CIMException
     {
     if (op.getObjectName().equalsIgnoreCase\
                            ("Ex_SimpleCIMInstanceProvider"))
             {
             Vector instances = new Vector();
             CIMInstance ci = cc.newInstance();
                 if (loop == 0){
                     ci.setProperty("First", new CIMValue("red"));
                     ci.setProperty("Last", new CIMValue("apple"));
                     // 要求されたプロパティのみ含める
       ci = ci.filterProperties(propertyList, includeQualifier,
       includeClassOrigin);
                     instances.addElement(ci);
                     loop += 1;
                 } else {
                     ci.setProperty("First", new CIMValue("red"));
                     ci.setProperty("Last", new CIMValue("apple"));
```

例 **4–1** CIMInstance プロバイダ　*(続き)*

```
                    // 要求されたプロパティのみ含める
        ci = ci.filterProperties(propertyList, includeQualifier,
        includeClassOrigin);
                    instances.addElement(ci);
                    ci = cc.newInstance();
                    ci.setProperty("First", new CIMValue("green"));
                    ci.setProperty("Last", new CIMValue("apple"));
                    // 要求されたプロパティのみ含める
      ci = ci.filterProperties(propertyList, includeQualifier,
      includeClassOrigin);
                    instances.addElement(ci);
                }
             return (CIMInstance[])instances.toArray();
             }
        throw new CIMException(CIM_ERR_INVALID_CLASS);
    }

    public CIMInstance getInstance(CIMObjectPath op, boolean localOnly,
    boolean includeQualifiers, boolean includeClassOrigin,
    String[] propertyList, CIMClass cc) ) throws CIMException {
            if (op.getObjectName().equalsIgnoreCase
                            ("Ex_SimpleCIMInstanceProvider"))
            {
                CIMInstance ci = cc.newInstance();
// passed in オブジェクトパスからキーを取得する必要がある
// 取得したいインスタンスを一意に識別する
java.util.Vector keys = cop.getKeys();
// この例は一般的でないので、単にキーをインスタンスに
// 配置して実行する
                ci.setProperties(keys);
// ほかにキー以外のプロパティがある場合には、ここに追加する

                // 要求されたプロパティのみ含める
ci = ci.filterProperties(propertyList, includeQualifiers,
includeClassOrigin);
                return ci;
            }
           throw new CIMException(CIM_ERR_INVALID_CLASS);
      }

    public CIMInstance[] execQuery(CIMObjectPath op, \
                           String query, String ql, CIMClass cc)
          throws CIMException {
          throw(new CIMException(CIMException.CIM_ERR_NOT_SUPPORTED));
    }

    public void setInstance(CIMObjectPath op, CIMInstance ci, boolean
                            includeQualifiers, String[] propertyList)
          throws CIMException {
          throw(new CIMException(CIMException.CIM_ERR_NOT_SUPPORTED));
    }

    public CIMObjectPath createInstance(CIMObjectPath op, CIMInstance ci)
```

例 **4–1** CIMInstance プロバイダ　(続き)

```
            throws CIMException {
        throw(new CIMException(CIMException.CIM_ERR_NOT_SUPPORTED));
    }

     public void deleteInstance(CIMObjectPath cp) throws CIMException {
         throw(new CIMException(CIMException.CIM_ERR_NOT_SUPPORTED));
     }
 }
```

メソッドプロバイダの作成

クライアントプログラムで Solaris WBEM プロバイダのメソッドを呼び出すには、メソッド invokeMethod を使用する以外には他の方法はありません。次のいずれかのプロバイダを使用します。

- 組み込み型 – プラットフォームに依存しない CIM_* プロバイダまたは Solaris 固有の Solaris_* プロバイダ
- 開発者によって追加 – たとえば、MethodProvider インタフェースを実装すると、プロバイダまたは WBEM 以外のメソッドを提供するメソッドプロバイダが作成される

次のコード例は、CIM Object Manager からの要求を 1 つ以上の特化されたプロバイダに配信する Solaris_ComputerSystem プロバイダクラスを作成します。これらの特化されたプロバイダは、特定の管理対象オブジェクトの動的データの要求に対するサービスを行います。たとえば、Solaris_Package プロバイダは、Solaris_Package クラスのメソッドを実行する要求に対応します。

メソッドプロバイダは、invokeMethod という単一のメソッドを実装します。このメソッドは、システムのリブート、システムの停止、またはシリアルポートの削除のいずれかの処理を実行する適切なプロバイダを呼び出します。

例 **4–2** メソッドプロバイダ

```
...
public class Solaris_ComputerSystem implements MethodProvider {
    ProviderCIMOMHandle pch = null;
    public void initialize(CIMOMHandle ch) throws CIMExcepiton {
        pch = (ProviderCIMOMHandle)ch;
    }

    public void cleanup() throws CIMException {
    }

    public CIMValue invokeMethod(CIMObjectPath op, String methodName,
            Vector inParams, Vector outParams)  throws CIMException {
        if (op.getObjectName().equalsIgnoreCase("solaris_computersystem")) {
            if (methodName.equalsIgnoreCase("reboot")) {
                // ヘルパー関数を呼び出す (ここには表示されていない)
```

例 **4–2** メソッドプロバイダ　(続き)

```
                return new CIMValue(rebootSystem());
            }
            if (methodName.equalsIgnoreCase("shutdown")) {
                // ヘルパー関数を呼び出す (ここには表示されていない)
                return new CIMValue(shutdownSystem());
            }
        }
        if (op.getObjectName().equalsIgnoreCase("solaris_serialport")) {
            if (methodName.equalsIgnoreCase("disableportservice")) {
                // ヘルパー関数を呼び出す (ここには表示されていない)
                return new CIMValue(deletePort(op));
            }
        }
        // エラー
        throw new CIMException(CIMException.CIM_ERR_NOT_SUPPORTED,
            "The requested function does not exist");
    }
    // 以下にヘルパー関数が定義される
    ...
}
```

アソシエータプロバイダの作成

注 – クライアントプログラム (`CIMObjectPath`) によって呼び出される関連メソッドの `objectName` 引数は、クラスではなくインスタンスのオブジェクトパスである必要があります。

インスタンスのオブジェクトパスが指定されていない場合、CIM Object Manager は、クライアントが CIM Object Manager Repository の関連 (関連のメンバーインスタンスが派生するテンプレート) のクラス定義を必要としているとみなします。プロバイダではなく、クライアント API の関連メソッドを使用します。

関連を設計およびコーディングする場合に最も重要な部分は、関連クラスそのものです。作成する関連は、関連クラスのコンテンツ以上には複雑になりません。関連のメンバー数は、関連クラスの参照数に等しくなります。役割は、さらに複雑な関連をモデル化する場合に使用できます。次に関連クラスの例を示します。

- 非対称ペア関連。教師と生徒のように 1 対 1 関係では、2 つの役割 (`teaches` と `taughtby`) が定義されます。

```
class TeacherStudent
    {
        Teacher REF teaches;
        Student REF taughtby;
    };
```

- 1 対多関係

```
class Classroom
    {
        Teacher REF teaches;
        Student1 REF taughtby;
        Student2 REF taughtby;
        Student3 REF taughtby;
        Student4 REF taughtby;
    };
```

- 多対多関係

```
class TeachingAssistants
    {
        Assistant1 REF assists;
        Assistant2 REF assists;
        Student1 REF assistedby;
        Student2 REF assistedby;
        Student3 REF assistedby;
        Student4 REF assistedby;
        Student5 REF assistedby;
    };
```

- 互いに対等な 2 つ以上のメンバーの関連

```
class Club
    {
        Member1 REF;
        Member2 REF;
        Member3 REF;
    };
```

次のコード例では associators メソッドを実装します。CIM Object Manager は、associatorNames、objectName、role、resultRole、includeQualifiers、includeClassOrigin、および propertyList のそれぞれの値を関連プロバイダに渡します。また CIM 関連クラスの名前と、返される関連オブジェクトが属する CIM クラスまたはインスタンスを出力します。このプロバイダは、example_teacher クラスと example_student クラスのインスタンスを扱います。

例 **4–3** CIMAssociator プロバイダ

```
...

public CIMInstance[] associators(CCIMObjectPath assocName, CIMObjectPath
                objectName, String resultClass, String role, String
                resultRole, boolean includeQualifiers, boolean
                includeClassOrigin, String[] propertyList)
                throws CIMException {
        System.out.println("Associators "+assocName+" "+objectName);
      if (objectName.getObjectName()equalsIgnoreCase("example_teacher")) {
      Vector v = new Vector();
      if ((role != null)  && (!role.equalsIgnoreCase("teaches"))) {
        // Teacher は、teaches という役割だけを担う
            return v;
        }
        if ((resultRole != null)  && (!resultRole.equalsIgnoreCase
                                    ("taughtby"))) {
```

例 **4–3** CIMAssociator プロバイダ　(続き)

```
        // Teacher は、taughtby という役割によってのみ得られる
        return v;
        }
        // Teacher のアソシエータを取得する
        CIMProperty nameProp = (CIMProperty)objectName.getKeys().elementAt
                               (0);
        String name = (String)nameProp.getValue().getValue();
        // Student のクラスを取得する
        CIMObjectPath tempOp = new CIMObjectPath("example_student");
        tempOp.setNameSpace(assocName.getNameSpace());
        CIMClass cc = cimom.getClass(tempOp, false);
        // objectName によって渡されたインスタンス名をテストし、Student
        // クラスの関連インスタンスを返す
        if(name.equals("teacher1")) {
                // teacher1 の Student (複数) を取得する
                CIMInstance ci = cc.newInstance();
                ci.setProperty("name", new CIMValue("student1"));
                v.addElement(ci.filterProperties(propertyList,
                             includeQualifiers,
                        includeClassOrigin));
                ci = cc.newInstance();
                ci.setProperty("name", new CIMValue("student2"));
                v.addElement(ci.filterProperties(propertyList,
                        includeQualifiers, includeClassOrigin));
                return v;
        }
}
```

インジケーションプロバイダの作成

CIM イベントのインジケーションを生成するには、次の処理を実行します。

- EventProvider インタフェースのメソッドを使用して、CIM イベントインジケーションの送信をいつ開始および停止するかを検出する
- CIM_Indication クラスの 1 つまたは複数のサブクラスのインスタンスを作成し、発生した CIM イベントの情報を格納する
- ProviderCIMOMHandle インタフェースの deliverEvent メソッドを使用して、インジケーションを CIM Object Manager に配信する

▼ イベントインジケーションを生成する方法

1. EventProvider インタフェースを実装します。
   たとえば、次のようになります。

```
public class sampleEventProvider implements
        InstanceProvider EventProvider{
```

```
    // プロバイダが CIM Object Manager に接続するための参照
    private ProviderCIMOMHandle cimom;
}
```

2. プロバイダが処理するインスタンスインジケーションに対して、表 **4–2** に示すそれぞれのメソッドを実行します。
3. 作成、変更、および削除インスタンスイベントのそれぞれの種類に対してインジケーションを作成します。

   以下に `createInstance` メソッドの例を示します。

```
public CIMObjectPath createInstance(CIMObjectPath op,
        CIMInstance ci)
    throws CIMException {
        CIMObjectpath newop = ip.createInstance(op, ci);
        CIMInstance indication = new CIMInstance();
        indication.setClassName("CIM_InstCreation");
        CIMProperty cp = new CIMProperty();
        cp.setName("SourceInstance");
        cp.setValue(new CIMValue(ci));
        Vector v = new Vector();
        v.addElement(cp);
        indication.setProperties(v);
        ...
    }
```

4. イベントインジケーションを **CIM Object Manager** に配信します。

```
cimom.deliverEvent(op.getNameSpace(), indication);
```

イベントプロバイダメソッド

イベントプロバイダは、`EventProvider` インタフェースを実装します。CIM Object Manager は、このインタフェースのメソッドを使って、クライアントが CIM イベントのインジケーションを予約したり、CIM イベントの予約を取り消したことをプロバイダに知らせます。さらにこれらのメソッドによって、プロバイダは、CIM Object Manager が特定のイベントインジケーションのポーリングを行うべきかどうか、インジケーションをハンドラに返すことを承認するかどうかを指定します。

次の表に、イベントプロバイダで実装する必要のある `EventProvider` インタフェースのメソッドを示します。

表 **4–2** `EventProvider` メソッド

メソッド	説明
`activateFilter`	クライアントが予約を作成すると、CIM Object Manager は、このメソッドを呼び出して CIM イベントの検査をプロバイダに依頼する

表 **4–2** EventProvider メソッド　(続き)

メソッド	説明
authorizeFilter	クライアントが予約を作成すると、CIM Object Manager は、このメソッドを呼び出して指定されたフィルタ式が許可されているかを検査する
deActivateFilter	クライアントが予約を削除すると、CIM Object Manager は、このメソッドを呼び出して指定されたイベントフィルタの停止をプロバイダに依頼する
mustPoll	クライアントが予約を作成すると、CIM Object Manager は、このメソッドを呼び出して、指定されたフィルタ式をプロバイダが許可するかどうか、そのフィルタ式のポーリングが必要かどうかを検査する

CIM Object Manager は、すべてのメソッドに次の引数の値を渡します。

- *filter* – インジケーションを生成する必要がある CIM イベントを指定する SelectExp 型
- *eventType* – CIM イベントの種類を指定する String 型。これは、select 式の FROM 節から抽出することもできる
- *classPath* – このイベントを必要とするクラス名を指定する CIMObjectPath 型

さらに、activateFilter メソッドは、これがこのイベントの種類の最初のフィルタであることを示すブール値 firstActivation を受け取ります。deActivateFilter メソッドは、これが最後のフィルタであることを示すブール値 lastActivation を受け取ります。

インジケーションの作成と配信

クライアントアプリケーションが CIM_IndicationSubscription クラスのインスタンスを作成して CIM イベントのインジケーションを予約すると、CIM Object Manager はこの要求を適切なプロバイダに転送します。プロバイダが EventProvider インタフェースを実装していれば、CIM Object Manager は、プロバイダの activateFilter メソッドを呼び出して、指定するイベントのインジケーションの送信をいつ開始するかをプロバイダに通知します。また、CIM Object Manager は、プロバイダの deActivateFilter メソッドを呼び出して、指定するイベントのインジケーションの送信をいつ停止するかをプロバイダに通知します。

プロバイダは、インスタンスを作成、変更、削除するたびに、インジケーションを作成、配信して、CIM Object Manager の要求に応答します。通常、プロバイダは、CIM Object Manager が activateFilter メソッドを呼び出した時に設定し、deActivateFilter メソッドを呼び出した時にクリアされるフラグ変数を定義します。そのあと、インスタンスを作成、変更、または削除するメソッドの中で、動作中のフラグの状態を検査します。フラグが設定されている場合、プロバイダは、作

成した CIM インスタンスオブジェクトを含むインジケーションを作成し、deliverEvent メソッドを使用してこのインジケーションを CIM Object Manager に返します。フラグが設定されていない場合、プロバイダは、イベントインジケーションの作成や配信を行いません。

プロバイダは、activateFilter メソッドが呼び出されると、インジケーションの配信を開始します。プロバイダは、CIM_Indication の concrete (具象) サブクラスのインスタンスを作成し、ProviderCIMOMHandled.deliverIndication メソッドを起動します。CIM Object Manager は、インジケーションを受信し、そのインジケーションを適切なインジケーションハンドラに配信します。プロバイダは、複数の種類のイベントを処理できます。たとえば、サイクルインジケーションの場合、プロバイダは CIM_InstCreation、CIM_InstDeletion、および CIM_InstModification を処理できます。

予約者が設定した種類を監視する場合、プロバイダは activateFilter および deActivateFilter 呼び出しにそれぞれ渡された firstActivation および lastActivation フラグを使用できます。firstActivation フラグは、特定のイベントの種類をはじめて予約した場合には、true になります。同様に lastActivation は、特定のイベントの種類の最後の予約を削除すると、true になります。これらのフラグを検査すると、プロバイダは、指定したイベントの種類を監視するために簡単にリソースを割り当てたり、割り当てを解除したりすることができます。

承認について

機密データを扱うプロバイダは、インジケーションの要求に対する承認を検査することができます。その場合、プロバイダは Authorizable インタフェースを実装して、そのプロバイダが承認を検査することを示す必要があります。また、プロバイダは authorizeFilter メソッドを実装する必要があります。CIM Object Manager は、authorizeFilter メソッドを呼び出して、イベントハンドラの所有者 (UID) がフィルタ式の評価に基づいて返されるインジケーションの受信を承認されているかどうかを検査します。イベントの宛先の所有者 (イベントハンドラ) の UID は、フィルタを動作中にするように要求するクライアントアプリケーションの所有者と同じである必要はありません。

ネイティブプロバイダの作成

プロバイダは、管理対象デバイスに関する情報の取得と設定を行います。ネイティブプロバイダは、特定の管理対象デバイスに特化して作成されたプログラムです。たとえば、Solaris システム上のデータにアクセスするプロバイダには、通常、システムを照会するために C 関数が含まれます。

ネイティブプロバイダは、一般に次のような理由で作成されます。

- 効率 – 速度が重視されるコードの一部を下位レベルのプログラミング言語 (アセンブラなど) で実装したあと、Java アプリケーションでそれらの機能を呼び出すと便利な場合がある

- プラットフォーム固有の機能にアクセスする必要がある – 標準の Java クラスライブラリが、アプリケーションに必要なプラットフォームに固有の機能をサポートしていない場合がある
- レガシーコード – レガシーコードを Java プロバイダと共に継続して使用したい場合がある

JDK の一部である JNI (Java Native Interface) は、Java のネイティブプログラミングインタフェースです。JNI を使用してプログラムを作成すると、ほとんどのプラットフォームで完全に移植可能です。Java 仮想マシン (JVM) で動作する Java コードで JNI を使用すると、そのコードは C、C++、アセンブラのようなほかの言語で作成されたアプリケーションおよびライブラリで実行できます。

Java プログラムの作成、および Java プログラムとネイティブメソッドの統合についての詳細は、Java Web サイト `http://java.sun.com` を参照してください。

プロバイダの作成

次の手順に従ってプロバイダを作成します。

1. プロバイダプログラムを作成または編集します。
2. Java プログラムをコンパイルしてクラスファイルを作成します。
3. 共有オブジェクトファイル (`.so`) を `/usr/sadm/lib/wbem` にコピーします。
4. `CLASSPATH` を `.class` および `.jar` ファイルがある場所に設定します。
5. プロバイダを登録します。

▼ プロバイダの CLASSPATH を設定する方法

プロバイダの `CLASSPATH` を設定して、CIM Object Manager に `.class` および `.jar` ファイルの保存場所を知らせます。

1. `Solaris_ProviderPath` クラスのインスタンスを作成します。

たとえば、次のようになります。

```
/* root\system （名前空間名） で初期化された名前空間オブジェクトを
ローカルホスト上に作成する  */
CIMNameSpace cns = new CIMNameSpace("", "root\system");

// root\system 名前空間にスーパーユーザーとして接続する
cc = new CIMClient(cns, "root", "root_password");

// Solaris_ProviderPath クラスを取得する
cimclass = cc.getClass(new CIMObjectPath("Solaris_ProviderPath");
```

```
// Solaris_ProviderPath の新しいインスタンスを作成する
class ci = cimclass.newInstance();
```

2. 標準的な **URL** 形式を使用して、*pathurl* プロパティにファイルの場所を設定します。
   たとえば、次のようになります。

```
/* プロバイダの CLASSPATH に /myhome/myproviders を設定する*/
ci.setProperty("pathurl", new CIMValue(new String
                          ("file:///myhome/myproviders/")));
```

   標準的な URL 形式を次の表に示します。

プロバイダの CLASSPATH	標準 **URL** 形式
ディレクトリへの絶対パス	`file:///a/b/c/`
`.jar` ファイルへの絶対パス	`file:///a/b/my.jar`

3. インスタンスを作成します。
   たとえば、次のようになります。

```
// 更新されたインスタンスを CIM Object Manager に渡す
cc.createInstance(new CIMObjectPath(), ci);
```

▼ プロバイダを登録する方法

サポートするデータと操作についての情報を通信するため、またプロバイダの位置を CIM Object Manager に通知するため、CIM Object Manager に新しいプロバイダまたは変更されたプロバイダを登録します。CIM Object Manager は、この情報を使用してプロバイダの読み込みと初期化、および特定のクライアント要求に適切なプロバイダを決定します。

1. プロバイダがサポートするクラスを定義するファイルを作成します。

注 – MOF ファイルの作成方法の詳細については、DMTF Web サイト `http://www.dmtf.org` を参照してください。

---

2. **MOF** ファイルにプロバイダ修飾子を含めて、**CIM Object Manager** のプロバイダの種類と場所を指定します。
   たとえば、次のようになります。

```
[Provider("java:com.sun.providers.myprovider")]
Class_name {
...
};
```

   この修飾子は、次の要素を示します。

- java: – Java 言語で記述され、javax.wbem.provider インタフェースを実装するプロバイダ
- com.sun.providers.myprovider – プロバイダを実装する Java クラス名

3. mofcomp(1M) コマンドを使用して、**MOF** ファイルをコンパイルします。

例 **4–4** プロバイダの登録

次の MOF ファイルには、Ex_SimpleCIMInstanceProvider クラスのプロバイダとして SimpleCIMInstanceProvider が宣言されています。

```
// ==========================================================
// タイトル:       SimpleCIMInstanceProvider
// ファイル名:    SimpleCIMInstanceProvider.mof
// 説明:
// ==========================================================================

// ==========================================================================
// Pragmas
// ==========================================================================
#pragma Locale ("en-US")

// ==========================================================================
//   SimpleCIMInstanceProvider
// ==========================================================================
[Provider("java:SimpleCIMInstanceProvider")]
class Ex_SimpleCIMInstanceProvider
{
   // プロパティ
      [Key, Description("First Name of the User")]
   string First;
      [Description("Last Name of the User")]
   string Last;
};
```

第 5 章

WBEM 照会の作成

ここでは、WQL および照会 API を使用して照会を作成する方法について説明します。内容は次のとおりです。

- 109 ページの「WQL について」
- 110 ページの「照会の記述」
- 113 ページの「照会の構文解析」

注 – WBEM 照会 API (`javax.wbem.query`) の詳細については、`/usr/sadm/lib/wbem/doc/index.html` を参照してください。

WQL について

WQL (WBEM Query Language) は、ANSI SQL (ANSI 構造化照会言語、ANSI Structured Query Language) のサブセットです。WQL には、Solaris 環境で WBEM をサポートするために意味上の変更が加えられています。

次の表に SQL の概念と WQL の対応を示します。

表 5–1 SQL の概念と WQL の対応

SQL の概念	**WQL** での表現
テーブル	CIM クラス
行	CIM インスタンス
列	CIM プロパティ

注 – SQL のように、WQL 文でも単一引用符 (' ') を使用します。

Solaris WBEM Services 実装では、WQL は検索専用の言語です。WQL を使用すると、CIM データモデルを使って格納されたデータを照会できます。CIM モデルでは、オブジェクトの情報は CIM クラスや CIM インスタンスに格納されています。CIM インスタンスには、名前、データ型、値からなるプロパティを持つことができます。

照会の記述

WBEM クライアントでは、WQL を使ってデータを照会したり、フィルタを適用したりします。データが特定のプロバイダによって提供された場合、CIM Object Manager によりクライアント照会が適切なプロバイダに渡されます。ユーザーは、特定のクラスまたは特定の名前空間内のすべてのクラスにおいて、指定された照会と一致するインスタンスを検索できます。

たとえば、`Storage_Capacity` プロパティに特定の値を持つ `Solaris_DiskDrive` クラスのすべてのインスタンスを検索できます。

```
select * from Solaris_DiskDrive where Storage_Capacity = 1000
```

WQL キーワード

Solaris WBEM SDK では、Level 1 WBEM SQL がサポートされます。Level 1 WBEM SQL では、join のない簡単な select 操作を行うことができます。次の表に、Sun WBEM SDK でサポートされる WQL キーワードを示します。

表 **5–2** サポートされる WQL キーワード

キーワード	説明
`AND`	2 つのブール式を結合し、両方の式が True であれば、True を返す
`FROM`	`SELECT` 文にリストされているプロパティを持つクラスを指定する
`NOT`	NULL 文字と共に使用される比較演算子
`OR`	2 つの条件を結合する。1 つの文に複数の論理演算子が使用されていると、AND 演算子 (論理積演算子)が評価されたあとで OR 演算子 (論理和演算子) が評価される
`SELECT`	照会で使用されるプロパティを指定する

表 **5–2** サポートされる WQL キーワード　*(続き)*

キーワード	説明
WHERE	照会のスコープを狭める
LIKE	提供された最低限の情報に基づいて結果セットを生成する

SELECT 文

SELECT 文を使用して単一のクラスとそのサブクラスのインスタンスを取得します。取得するプロパティおよび満たす必要がある条件を指定することもできます。

注 – 現時点では、join 操作はサポートされません。

SELECT 文の構文は、次のとおりです。

SELECT *list* FROM *class* WHERE *condition*

次の表に、SELECT 文の検索を改良する引数の使用例を示します。

表 **5–3** SELECT 文の例

照会例	説明
SELECT * FROM *class*	指定されたクラスとそのすべてのサブクラスのすべてのインスタンスを選択する。返されたインスタンスには、すべてのプロパティが含まれる
SELECT PropertyA FROM *class*	指定されたクラスとそのすべてのサブクラスのうち、PropertyA を持つクラスまたはサブクラスのインスタンスだけを選択する
SELECT PropertyA, PropertyB FROM *class* WHERE PropertyB=20	指定されたクラスとそのすべてのサブクラスのうち、PropertyB=20 をもつクラスとサブクラスを選択する。返されたインスタンスには、PropertyA と PropertyB だけが含まれる

FROM 句

FROM 句では、照会文字列に一致するインスタンスが含まれているクラスを指定します。WQL FROM 句では、join 以外の式だけがサポートされます。したがって、WQL FROM 句には 1 つのクラスしか指定できません。

FROM 句は、abstract クラス fromExp によって表されます。現時点では、fromExp の直接のサブクラスは NonJoinExp だけです。NonJoinExp サブクラスは、1 つのテーブル (CIM クラス) だけを指定した FROM 句を表しています。SELECT 操作はこのテーブルに対して行われます。

WHERE 句

WHERE 句は、照会のスコープを狭めます。WHERE 句には条件式が含まれています。これらの条件式には、プロパティまたはキーワード、演算子、定数が含まれます。

SELECT 文のあとに追加する WHERE 句の構文の例を次に示します。

SELECT *CIMinstance* FROM *CIMclass* WHERE *conditional_expression*

WHERE 句の *conditional_expression* は、次の形式です。

property operator constant

expression には、プロパティまたはキーワード、演算子、定数を指定します。WHERE 句を SELECT 文のあとに追加するために、次のどちらかの形式を使用します。

SELECT *instance* FROM *class* [WHERE *constant operator property*]

WHERE 句は次の規則に従う必要があります。

- 定数の値は、プロパティに対して適切なデータ型である
- 演算子は、有効な WQL 演算子である
- 演算子のどちらかが、プロパティ名または定数である
- 任意の算術式を指定することはできないたとえば、次の照会では、ready 状態のプリンタを持つ Solaris_Printer クラスのインスタンスだけが返される

  ```
  SELECT * FROM Solaris_Printer WHERE Status = 'ready'
  ```

- WHERE 句中には、論理演算子やカッコ式を使用して、プロパティ、演算子、定数からなる複数のグループを結合することができる。各グループは、AND、OR、NOT 演算子で結合されている必要がある。

  次の例では、Name プロパティに home か files が設定されている Solaris_FileSystem クラスのすべてのインスタンスを取得します。

  ```
  SELECT * FROM Solaris_FileSystem WHERE Name= 'home' OR Name= 'files'
  ```

  次の例では、名前が home か files のディスクのうち、一定の使用可能な容量が残っており、 Solaris ファイルシステムを持つディスクを取得します。

  ```
  SELECT * FROM Solaris_FileSystem WHERE (Name = 'home' OR
  Name = 'files') AND AvailableSpace> 2000000 AND FileSystem = 'Solaris'
  ```

WHERE 句で使用できる標準の *WQL* 演算子

SELECT 文の WHERE 句でのバイナリ式では、次の標準の WQL 演算子を使用できます。

表 **5–4** WHERE 句で使用できる WQL 演算子

演算子	説明
=	等しい
<	より小さい
>	より大きい
<=	以下
>=	以上
<>	等しくない

照会の構文解析

javax.wbem.query パッケージに含まれるユーティリティクラスを使用すると、WQL 照会を構文解析できます。メインクラスは SelectExp で、そのコンストラクタは、WQL 照会文字列を取り込みます。 SelectExp は文字列を構文解析し、それを 3 つの部分に分割します。それぞれの部分は、次の表に示すように対応するアクセス用メソッドを使って取得できます。

照会部分	アクセス用メソッド
SELECT リスト	getSelectList
FROM 句	getFromClause
WHERE 句	getWhereClause

構文解析すると、次に示すように SELECT リストに PropertyA および PropertyB が含まれます。FROM 句には test_class が含まれ、WHERE 句には条件式の構文解析ツリーが含まれます。

```
select PropertyA, PropertyB from test_class where
       PropertyA> 20 and PropertyB < 30
```

SELECT リスト

各 SelectExp の getSelectList メソッドが返す SELECT リストは、SelectList クラスのインスタンスです。SELECT リストは、選択したインスタンスに含める必要があるプロパティを指定し、AttributeExp インスタンスを構成します。SelectList メソッドの要素を使用すると、これらの AttributeExp インス

タンスを取得できます。それぞれの属性は、WQL の CIMInstance プロパティに対応する列名を表します。 AttributeExp の apply メソッドを CIMInstance に渡すと、 AttributeExp が表すプロパティ値が返されます。SelectList の apply メソッドを CIMInstance に渡すと、SelectList AttributeExp インスタンスが示すプロパティだけを含む CIMInstance が返されます。

FROM 句

現時点では、FROM 句では join 以外の式だけが処理されます。 SelectExp で getFromClause メソッドが呼び出されると、NonJoinExp のインスタンスが返されます。NonJoinExp は、select 操作が実行されるクラス名を表します。

WHERE 句

WHERE 句は、abstract クラスの QueryExp によって表現されます。concrete サブクラスは、AndQueryExp、OrQueryExp、NotQueryExp、および BinaryRelQueryExp です。これらの式のインスタンスは、元の条件式を表す構文解析ツリーの形式で結合することができます。

このツリーの内部ノードは、AndQueryExp、OrQueryExp、および NotQueryExp インスタンスで構成されます。これらのインスタンスは、AND、OR、および NOT 式を表します。また同様にほかの AND、OR、NOT 式、およびバイナリ関連でも構成されます。

葉ノードは、BinaryRelQueryExp で、*property operator constant* 形式の式で表されます。これは、プロパティと定数値の 2 項関係で表されます。getLeftValue、getRightValue、および getOperator メソッドを使用して、*property operator constant* を取得します。

各 QueryExp の apply メソッドを CIMInstance に渡すと、ブール値が返されます。QueryExp で表される条件式が、CIMInstance で true になると、ブール値が true になります。true でない場合は、ブール値は false を返します。

QueryExp には、ほかにも 2 つの有用なメソッドである canonizeDOC と canonizeCOD があります。 canonizeDOC と canonizeCOD を使用すると、処理をする際の条件式を簡略化することができます。 canonizeDOC メソッドは、構文木を複数の AND および複数の OR の任意結合から標準的な論理積の論理和 (複数の AND の OR) 形式に変換します。canonizeCOD メソッドは、構文木を複数の AND および複数の OR の任意結合から標準的な論理和の論理積 (複数の OR の AND) 形式に変換します。これらのクラスおよびメソッドは、入力照会に基づいてインスタンスにフィルタを適用するプロバイダによって使用されます。

注 – これらのクラスの詳細については、Javadoc リファレンスページを検索してください。/usr/sadm/lib/wbem/doc/index.html を参照します。

照会を処理するプロバイダの記述

次のプロバイダプログラム例では、execQuery メソッドによって渡された WQL 文字列を、照会 API を使用して構文解析します。このプログラムは、照会文字列中の Select 式を構文解析し、クラスを詳細列挙します。次に、列挙した各インスタンスを 1 つずつ処理して、各インスタンスを照会式および select リストと比較します。最後にプログラムは、照会文字列と一致するインスタンスの列挙が含まれるベクトルを返します。

例 **5–1** 照会を処理するプロバイダ

```
/*
 * execQuery メソッドは、部分的なキー照合に基づく限られた照会
 * のみサポートする。照会によって選択されたエントリがないと、
 * 空の Vector を返す
 *
 * @param  op 返される CIM インスタンスの CIM オブジェクトパス
 * @param  query CIM 照会式
 * @param  ql CIM 照会言語のインジケータ
 * @param  cc CIM クラス参照
 *
 * @return  CIM オブジェクトインスタンスのベクトル
 *
 * @version    1.19 01/26/00
 * @author     Sun Microsystems, Inc.
 */
public CIMInstance[] execQuery(CIMObjectPath op,
                       String query,
                       String  ql,
                       CIMClass cc)
           throws CIMException {

   Vector result = new Vector();
   try {
       SelectExp q = new SelectExp(query);
       SelectList attrs = q.getSelectList();
       NonJoinExp from = (NonJoinExp)q.getFromClause();
       QueryExp where = q.getWhereClause();

       CIMInstance[] v = enumerateInstances(op, false, true,
                                            true, null, cc);

       // インスタンスをフィルタリングする
       for (int i = 0; i < v.length; i++) {
               if ((where == null) || (where.apply(v[i]) == true)) {
                   result.addElement(attrs.apply(v[i]));
```

例 **5–1** 照会を処理するプロバイダ　(続き)

```
                }
            }
        } catch (Exception e) {
            throw new CIMException(CIMException.CIM_ERR_FAILED, e.toString());
        }
        return (CIMInstance[])result.toArray();
    } // execQuery
}
```

第 6 章

Solaris WBEM SDK サンプルプログラムの使用

この章では、Solaris WBEM SDK に付属のサンプルプログラムについて説明します。内容は次のとおりです。

- 117 ページの「サンプルプログラムについて」
- 118 ページの「サンプルアプレット」
- 119 ページの「サンプルクライアントプログラム」
- 121 ページの「サンプルプロバイダプログラム」

サンプルプログラムについて

Solaris WBEM SDK をインストールすると、Java アプレットおよびいくつかのプログラムが `/usr/demo/wbem` にインストールされます。これらのサンプルは、ユーザー独自のアプレットおよびプログラムを開発するためのベースとして使用できます。

注 – アプレットおよびサンプルプログラムを使用するには、`/usr/java` が JDK 1.2.2 以上で Solaris WBEM SDK ファイルが `/usr` ディレクトリにインストールされている必要があります。

次のサンプルプログラムが提供されます。

- アプレット – Solaris WBEM Services が動作するシステムにインストールされた Solaris ソフトウェアパッケージのリストを表示して説明する。ローカルまたはリモートシステム上の CIM Object Manager に接続する
- クライアントプログラム – Java API を使用して CIM Object Manager に要求を出すプログラム
- プロバイダプログラム – データにアクセスするために管理対象オブジェクトと通信を行うプログラム

サンプルアプレット

注 – このアプレットの詳細については、/usr/demo/wbem/applet/README を参照してください。

アプレットは、CIM Object Manager にネットワークからアクセスしているコンピュータで実行する必要があります。またそのコンピュータで、次のソフトウェアを実行する必要があります。

- JDK 1.2 アプレットビューア
- JDK 1.2.2 またはそれ以降のリリースの Java 実行環境を使用する Web ブラウザ、または JDK 1.2.2 またはそれ以降のリリースの Java Plug-in 製品がインストールされている Web ブラウザ

JDK アプレットビューアまたは Java 実行環境の詳細については、http://java.sun.com を参照してください。Java Plug-in の詳細については、『*Solaris Java Plug-in* ユーザーズガイド』を参照してください。

▼ アプレットビューアを使用してサンプルアプレットを実行する方法

- 次のコマンドを入力します。

```
% appletviewer -JD \
java.security.policy=/usr/demo/wbem/applet/applet.policy \
/usr/demo/wbem/applet/GetPackageInfoAp.html
```

▼ Web ブラウザでサンプルアプレットを実行する方法

- **Web** ブラウザで次のファイルを開きます。

  /usr/demo/wbem/applet/GetPackageInfoAp.html

サンプルクライアントプログラム

サンプルクライアントプログラムは、/usr/demo/wbem/client のサブディレクトリにあります。次の表でプログラムについて説明します。

表 **6–1** サンプルクライアントプログラム

ディレクトリ	プログラム	目的
./batching	./TestBatch *host_name user_name password classname* [rmi|http]	バッチ処理を 1 回呼び出すだけで enumerateInstanceName, getClass および enumerateInstances を実行する
./enumeration	./ClientEnum *host username password classname* [rmi|http]	指定されたホスト上のデフォルトの名前空間 root\cimv2 にある指定されたクラスのサブクラスとインスタンスを列挙する
./events	./Subscribe *host username password classname*	指定されたクラスのライフサイクルイベントを予約し、予約してから 1 分以内に発生したイベントを出力する。そのあと、イベントの予約を解除する
./logging	./CreateLog *host root_username root_password* [rmi|http]	指定されたホスト上にログレコードを作成する
	./ReadLog *host root_username root_password* [rmi|http]	指定されたホスト上のログレコードを読み取る
./misc	./DeleteClass *host classname root_username root_password* [rmi|http]	指定されたホスト上のデフォルトの名前空間 root\cimv2 にある指定されたクラスを削除する
	./DeleteInstances *host classname root_username root_password* [rmi|http]	指定されたホスト上のデフォルトの名前空間 root\cimv2 にある指定されたクラスのインスタンスを削除する
./namespace	./CreateNameSpace *host parentNS childNS root_username root_password* [rmi|http]	指定されたユーザーとして CIM Object Manager に接続し、指定されたホスト上に名前空間を作成する
	./DeleteNameSpace *host parentNS childNS root_username root_password* [rmi|http]	指定されたホスト上の指定された名前空間を削除する

表 **6–1** サンプルクライアントプログラム　(続き)

ディレクトリ	プログラム	目的
./query	./ExampleQuery *host username password* [rmi|http] *WQL_query*	サンプルインスタンスでテストクラスを作成し、そのクラスで照会を実行する
	./TestQuery *host username password* [rmi|http] *WQL_query*	指定された WQL 照会を実行する
./systeminfo	./SystemInfo *host username password* [rmi|http]	指定されたホストの Solaris プロセッサとシステムの情報を個別のウィンドウに表示する

サンプルクライアントプログラムの実行

クライアントプログラムを実行する前に、CLASSPATH に必要な .jar ファイルを設定する必要があります。

▼ CLASSPATH の設定方法

- 次のいずれかの方法を使用します。
  - C シェルを使用して次のように入力します。

```
% setenv CLASSPATH .:/usr/sadm/lib/wbem.jar:/usr/sadm/lib/xml.jar
:/usr/sadm/lib/wbem/sunwbem.jar:/usr/sadm/lib/wbem/extension
```

  - Bourne シェルを使用して次のように入力します。

```
$ set CLASSPATH=.:/usr/sadm/lib/wbem.jar:/usr/sadm/lib/xml.jar
:/usr/sadm/lib/wbem/sunwbem.jar:/usr/sadm/lib/wbem/extension
```

▼ サンプルクライアントプログラムの実行方法

ほとんどのクライアントサンプルプログラムでは、CIM Object Manager の接続に使用するプロトコルを指定するオプションパラメータを使用できます。 RMI は、デフォルトプロトコルです。

- 次の形式を使用してサンプルクライアントプログラムを実行します。

```
% java program_name parameters
```

たとえば、次のスクリプトは、 HTTP プロトコルを使用して *secret* パスワードで *root* ユーザーとして *myhost* に接続し SystemInfo プログラムを実行します。

```
% java SystemInfo myhost root secret  http
```

サンプルプロバイダプログラム

サンプルプロバイダプログラムは、/usr/demo/wbem/provider サブディレクトリにあります。次の表では、このプログラムについて説明します。

表 **6–2** サンプルプロバイダプログラム

ファイル名	目的
NativeProvider.java	CIM Object Manager からの要求に応答し、それらを Native_Example プロバイダに配信する最上位のプロバイダプログラム。このプログラムは、instanceProvider API と method Provider API を実装し、Native_Example クラスのインスタンスを列挙するメソッドと、取得するメソッドを宣言するこのプログラムは、文字列「Hello World」を出力するメソッドを呼び出すメソッドも宣言する
Native_Example.mof	NativeProvider プロバイダを CIM Object Manager に登録するクラスを作成する。この MOF ファイルは、NativeProvider を、Native_Example クラスの動的データ要求に対してサービスを行うプロバイダとして識別する。また NativeProvider によって実装されるプロパティとメソッドの宣言も行う
Native_Example.java	NativeProvider プログラムは、このプロバイダを呼び出して、Native_Example クラスのインスタンスを列挙するメソッドと取得するメソッドを実装する。Native_Example プロバイダは、API を使用してオブジェクトの列挙とオブジェクトインスタンスの作成を行うNative_Example クラスは、native.c ファイル内の C 関数を呼び出してシステム固有の値 (ホスト名、シリアル番号、リリース、マシン、アーキテクチャ、メーカーなど) を取得するネイティブメソッドの宣言も行う
native.c	Native_Example Java プロバイダからの呼び出しをネイティブ C コードで実装する C プログラム
Native_Example.h	Native_Example クラスに対して自動的に生成されるヘッダーファイル。Java ネイティブメソッド名とそれらのメソッドを実行するネイティブ C 関数間の対話を定義する
libnative.so	native.c ファイルからコンパイルされるバイナリネイティブ C コード

▼ サンプルプロバイダプログラムの実行方法

サンプルプロバイダプログラムを実行する前に環境を設定する必要があります。

1. LD_LIBRARY_PATH 環境変数を、プロバイダクラスファイルの位置に設定します。

- C シェルを使用して次のように入力します。

  ```
  % setenv LD_LIBRARY_PATH /usr/sadm/lib/wbem
  ```

- Bourne シェルを使用して次のように入力します。

  ```
  $ LD_LIBRARY_PATH=/usr/sadm/lib/wbem; export LD_LIBRARY_PATH
  ```

2. libnative.so 共有ライブラリファイルを、LD_LIBRARY_PATH 環境変数によって指定されているディレクトリにコピーします。

   ```
   % cp libnative.so /usr/sadm/lib/wbem
   ```

3. プロバイダクラスファイルを、それらのファイルが定義されているパッケージと同じパスに移動します。

   ```
   % mv *.class /usr/sadm/lib/wbem
   ```

4. スーパーユーザーになります。

5. LD_LIBRARY_PATH 環境変数を設定したシェルと同じシェルの **CIM Object Manager** を停止します。

   ```
   # /etc/init.d/init.wbem stop
   ```

   注 – シェルに LD_LIBRARY_PATH 環境変数を設定する場合は、設定した新しい値を認識させるために、そのシェルで CIM Object Manager の停止と再起動を行なってください。

   ---

6. **CIM Object Manager** を起動します。

   ```
   # /etc/init.d/init.wbem start
   ```

7. スーパーユーザー状態から抜けます。

8. **CIM Object Manager** の適切なクラスを読み込んでプロバイダを識別するためにプログラムに関連した .mof ファイルをコンパイルします。

   ```
   % mofcomp -u root -p root_password Native_Example.mof
   ```

9. **CIM Workshop** を起動します。

   ```
   % /usr/sadm/bin/cimworkshop
   ```

10. **CIM Workshop** ツールバーの「クラスを検索 **(Find Class)** 」アイコンをクリックします。

11. 「入力 **(Input)** 」ダイアログボックスで、表示したいクラス名を入力して「了解 **(OK)** 」をクリックします。

    CIM Workshop にクラスが表示されます。

付録 A

WBEM のエラーメッセージ

この付録では、Solaris WBEM Services と Solaris WBEM SDK のコンポーネントが生成するエラーメッセージについて説明します。内容は次のとおりです。

- 123 ページの「WBEM のエラーメッセージ」
- 124 ページの「エラーメッセージの一覧」

WBEM のエラーメッセージ

CIM Object Manager によって、MOF コンパイラと CIM Workshop で使用されるエラーメッセージが生成されます。MOF コンパイラにより、`.mof` ファイルのどこでエラーが発生したかを示す行番号が、エラーメッセージに追加されます。

注 – MOF コンパイラの詳細については、`mofcomp(1M)` のマニュアルページを参照してください。

エラーメッセージの構成

エラーメッセージは、次の要素から構成されます。

- 一意の識別子 – エラーメッセージを識別するための文字列。Javadoc 参照ページで一意の識別子を検索すると、エラーメッセージの内容についての説明を参照できる
- パラメータ – 例外メッセージに示される特定のクラス、メソッド、および修飾子の可変部分

例 **A–1** エラーメッセージの構成

MOF コンパイラは、次のようなエラーメッセージを返します。

例 **A–1** エラーメッセージの構成　（続き）

```
REF_REQUIRED = Association class CIM_Docked needs at least two refs.
Error in line 12.
```

- REF_REQUIRED は一意の識別子を示す
- CIM_Docked はパラメータを示す
- line 12 は、エラーが発生した .mof ファイルの行番号を示す

エラーメッセージの一覧

この節では、WBEM エラーメッセージを、一意の識別子順に説明します。

ABSTRACT_INSTANCE
: 説明**:** このエラーメッセージはパラメータ {0} を使用しますが、このパラメータは abstract クラスの名前に置換されています。

  原因**:** 指定されたクラスにインスタンスが作成されましたが、このクラスは abstract クラスです。abstract クラスは、インスタンスを持つことができません。

  対処方法**:** concrete クラスにインスタンスを作成します。

CANNOT_ASSUME_ROLE
: 説明**:** このエラーメッセージは、次の 2 つのパラメータを使用します。

  - {0} は、ユーザー名に置き換えられます。
  - {1} は、役割名に置き換えられます。

  原因**:** 指定された主体は、指定された役割を担えません。

  対処方法**:** ユーザーが指定された役割に適切な権限を持っているかどうかを確認します。ユーザーが適切な権限を持っていない場合、担当のシステム管理者にお問い合わせください。

CHECKSUM_ERROR
: 説明**:** このエラーメッセージは、パラメータを使用しません。

  原因**:** メッセージは、壊れているため送信できませんでした。この損傷は、送信中に誤って生じたか、あるいは第三者によって故意に壊された可能性があります。

注 – このエラーメッセージは、CIM Object Manager が無効なチェックサムを受ける場合に表示されます。チェックサムは、ネットワーク上で転送されるデータパケットのビット数です。この数は、伝送が安全であり、送信中にデータの破損や意図的な変更がなかったことを情報の送信側と受信側が確認するために使用されます。

送信前に、データに対してアルゴリズムが実行されます。この実行により生成されたチェックサムがデータに含められ、データパケットのサイズを示します。メッセージを受信すると、受信者はチェックサムを再計算し、送信者のチェックサムと比較できます。チェックサムが一致すれば、送信は安全に行われ、データの破損や変更が起きなかったと言えます。

対処方法**:** Solaris WBEM Services のセキュリティ機能を使用してメッセージを再送信します。Solaris WBEM Services のセキュリティについての詳細は、『*Solaris WBEM Services* 管理ガイド』の「セキュリティの管理 (手順)」を参照してください。

`CIM_ERR_ACCESS_DENIED`

説明**:** このエラーメッセージは、パラメータを使用しません。

原因**:** このエラーメッセージは、動作を完了するための適切な特権およびアクセス権がユーザーにない場合に表示されます。

対処方法**:** 担当のシステム管理者または CIM Object Manager の管理者に、処理を行うための特権を要求します。

`CIM_ERR_ALREADY_EXISTS`

例 *1: CIM_ERR_ALREADY_EXISTS*

説明**:** この場合には、エラーメッセージはパラメータ `{0}` を使用しますが、このパラメータは重複したクラス名に置換されています。

原因**:** 作成しようとしたクラスに、既存のクラスと同じ名前が使用されています。

対処方法**:** CIM Workshop で既存のクラスを検索して使用されているクラス名を確認し、一意の名前でクラスを作成します。

例 *2: CIM_ERR_ALREADY_EXISTS*

説明**:** この場合はパラメータ `{0}` を使用しますが、このパラメータは重複したインスタンス名に置換されています。

原因**:** 作成しようとしたクラスのインスタンスに、既存のインスタンスと同じ名前が使用されています。

対処方法**:** CIM Workshop で既存のインスタンスを検索して、使用中の名前を確認し、一意の名前でインスタンスを作成します。

例 *3: CIM_ERR_ALREADY_EXISTS*

説明**:** この場合は、パラメータ {0} を使用しますが、このパラメータは重複した名前空間名に置換されています。

原因**:** 作成しようとした名前空間に、既存の名前空間と同じ名前を使用します。

対処方法**:** CIM Workshop で既存の名前空間を検索して使用されている名前を確認します。次に一意の名前を使用して名前空間を作成します。

例 *4: CIM_ERR_ALREADY_EXISTS*

説明**:** この場合はパラメータ {0} を使用しますが、このパラメータは重複した修飾子型の名前に置換されています。

原因**:** 作成しようとした修飾子型に、変更されるプロパティの既存の修飾子型と同じ名前を使用します。

対処方法**:** CIM Workshop でプロパティの既存の修飾子型を検索して使用されている名前を確認します。次に一意の名前を使用して修飾子型を作成します。

CIM_ERR_CLASS_HAS_CHILDREN
説明**:** このエラーメッセージはパラメータ {0} を使用しますが、このパラメータはクラス名に置換されています。

原因**:** この例外は、スーパークラスを削除することによってサブクラスが無効にならないように CIM Object Manager によってスローされます。クライアントは、まず明示的にサブクラスを削除する必要があります。クラスインスタンスの検査前にサブクラスが検査されます。

対処方法**:** 指定したクラスのサブクラスを削除します。

CIM_ERR_CLASS_HAS_INSTANCES
説明**:** このエラーメッセージはパラメータ {0} を使用しますが、このパラメータはクラス名に置換されています。

原因**:** インスタンスをもつクラスを削除しようとすると、この例外がスローされます。

対処方法**:** 指定したクラスのインスタンスを削除します。

CIM_ERR_FAILED
説明**:** このエラーメッセージはパラメータ {0} を使用しますが、このパラメータはエラー状態と、考えられるその原因を説明したメッセージに置換されています。

原因**:** このエラーメッセージは、さまざまなエラー状況に対して表示される一般的なメッセージです。

対処方法**:** 解決方法は、エラー状況によって異なります。

CIM_ERR_INVALID_PARAMETER
説明: このエラーメッセージは、パラメータ {0} を使用しますが、このパラメータは無効なパラメータ名に置換されています。

原因: パラメータ名またはメソッド名が無効です。

対処方法: パラメータを修正します。

CIM_ERR_INVALID_QUERY
説明: このエラーメッセージは、次の 2 つのパラメータを使用します。

- {0} が照会の無効部分に置換されています。
- {1} は、照会の実際のエラーなどの追加情報に置換されています。

原因: 指定された照会には、構文エラーまたは意味上のエラーのいずれかが含まれます。

対処方法: 例外の詳細に基づいてエラーを修正します。また照会文字列と照会言語が一致することを確認してください。

CIM_ERR_INVALID_SUPERCLASS
説明: このエラーメッセージは、次の 2 つのパラメータを使用します。

- {0} は、指定されたスーパークラスの名前に置換されます。
- {1} は、指定されたサブクラスが存在しないクラスの名前に置換されます。

原因: あるスーパークラスのサブクラスに属するクラスが指定されましたが、そのスーパークラスは存在しません。指定されたスーパークラスにスペルミスがあるか、あるいは意図したスーパークラス名の代わりに存在しないスーパークラス名が指定されたことが考えられます。また、そのスーパークラスとサブクラスに変更があった可能性もあります。つまり、指定されたスーパークラスは、実際は指定されたクラスのサブクラスであるかもしれません。この例では、CIM_Container のスーパークラスとして CIM_Chassis が指定されていますが、CIM_Chassis は CIM_Container のサブクラスです。

対処方法: スーパークラスのスペルと名前が正しいか確認し、名前空間内にそのスーパークラスが存在することを確認します。

CIM_ERR_LOW_ON_MEMORY
説明: このエラーメッセージは、パラメータを使用しません。

原因: CIM Object Manager のメモリー不足です。

対処方法: クラス定義および静的インスタンスの一部を削除してメモリーを解放します。

CIM_ERR_NOT_FOUND
例 *1: CIM_ERR_NOT_FOUND*

説明**:** このインスタンスはパラメータ {0} を使用しますが、このパラメータは存在しないクラス名に置換されています。

原因**:** 指定されたクラスが存在しません。指定されたクラスにスペルミスがあるか、あるいは意図したクラス名の代わりに誤って存在しないクラス名が指定されたことが考えられます。

対処方法**:** クラスのスペルと名前が正しいか確認し、名前空間内にそのクラスが存在することを確認します。

例 *2: CIM_ERR_NOT_FOUND*

説明**:** このインスタンスは、次の2 つのパラメータを使用します。

- {0} は、指定されたインスタンス名に置換されます。
- {1} は、指定されたクラス名に置換されます。

原因**:** インスタンスが存在しません。

対処方法**:** インスタンスを作成します。

例 *3: CIM_ERR_NOT_FOUND*

説明**:** このインスタンスはパラメータ {0} を使用しますが、このパラメータは指定された名前空間名に置換されています。

原因**:** 指定された名前空間が見つかりません。このエラーは、入力ミスまたはスペルミスのために入力された名前空間名が正しくない場合に発生します。

対処方法**:** 名前空間名を入力し直し、入力とスペルが正しいことを確認します。

CIM_ERR_QUERY_LANGUAGE_NOT_SUPPORTED
説明**:** このエラーメッセージは、パラメータ {0} を使用しますが、このパラメータは無効な照会言語文字列に置換されています。

原因**:** 要求された照会言語は、CIM によって認識されません。

対処方法**:** サポートされる照会言語を使用してください。

CLASS_REFERENCE
説明**:** このエラーメッセージは、次の 2 つのパラメータを使用します。

- {0} パラメータは、参照関係を定義されたクラス名に置換されます。
- {1} パラメータは、参照名に置換されます。

原因**:** あるクラスに、そのクラスが参照を持つことを示すプロパティが定義されました。ただし、そのクラスは関連の一部ではありません。クラスが参照をプロパティとして持つことを定義できるのは、別のクラスとの関連がある場合だけです。

対処方法**:** 結合修飾子を追加するか、参照を削除します。

INVALID_DATA
説明: このエラーメッセージは、パラメータを使用しません。

原因: セキュリティ認証データが無効か、使用中のセキュリティメカニズムと互換性がありません。

対処方法: 使用するセキュリティモジュールが正しく構成されているかどうか確認します。

INVALID_CREDENTIAL
説明: このエラーメッセージは、パラメータを使用しません。

原因: このエラーメッセージは、無効なパスワードが入力された場合、またはクライアントアプリケーションの認証検査を含めるように CLASSPATH が設定されていない場合に表示されます。

対処方法:

- 正しいパスワードを使用してください。
- CLASSPATH に次のファイルが含まれていることを確認します。
  /usr/sadm/lib/wbem/extension:/usr/sadm/lib/wbem/sunwbem.jar

INVALID_QUALIFIER_NAME
説明: このエラーメッセージはパラメータ {0} を使用しますが、このパラメータは空の修飾子名を表す MOF 表記に置換されています。

原因: プロパティの修飾子が作成されましたが、修飾子の名前が指定されませんでした。

対処方法: 修飾子を含めます。

KEY_OVERRIDE
説明: このエラーメッセージは、次の 2 つのパラメータを使用します。

- {0} パラメータは、1 つまたは複数のキー修飾子を持つクラスを無効にする abstract クラス以外の名前に置換されます。
- {1} パラメータは、キー修飾子を持つ concrete クラス名に置換されます。

原因: 非 abstract クラス、つまり concrete クラスによって、1 つまたは複数のキー修飾子を持つ concrete クラスをオーバーライドするようになっています。CIM では、すべての concrete クラスは 1 つ以上のキー修飾子を必要とし、キークラス以外のクラスはキーを持つクラスをオーバーライドできません。

対処方法: 非キークラスにキー修飾子を作成します。

KEY_REQUIRED
説明: このエラーメッセージはパラメータ {0} を使用しますが、このパラメータはキーを必要とするクラス名に置換されています。

原因**:** concrete クラスにキー修飾子が指定されませんでした。CIM では、concrete クラスと呼ばれる非 abstract クラスにはすべて、1 つ以上の修飾子が必要です。

対処方法**:** クラスにキー修飾子を作成します。

`METHOD_OVERRIDDEN`
説明**:** このエラーメッセージは、次の 3 つのパラメータを使用します。

- `{0}` は、パラメータ `{1}` で示されるメソッドのオーバーライドを試みているメソッド名に置換されます。
- `{1}` は、パラメータ `{2}` で示されるメソッドによってすでにオーバーライドされているメソッド名に置換されます。
- `{2}` は、パラメータ `{1}` をオーバーライドしたメソッド名に置換されます。

原因**:** 3 つ目のメソッドによってすでにオーバーライドされている別のメソッドのオーバーライドを試みるメソッドが指定されました。いったんオーバーライドしたメソッドを再度オーバーライドすることはできません。

対処方法**:** オーバーライドするには別のメソッドを指定します。

`NEW_KEY`
説明**:** このエラーメッセージは、次の 2 つのパラメータを使用します。

- `{0}` は、キーの名前に置換されます。
- `{1}` は、新しいキーを定義しようとするクラス名に置換されます。

原因**:** あるクラスが新しいキーを定義しようとしていますが、スーパークラス内にキーがすでに定義されています。スーパークラスにいったんキーが定義されると、サブクラスに新しいキーを設定することはできません。

対処方法**:** 新しいキーを定義しないでください。

`NO_CIMOM`
説明**:** このエラーメッセージは、パラメータ `{0}` を使用しますが、このパラメータは CIM Object Manager の実行ホストに指定されたホスト名に置換されています。

原因**:** CIM Object Manager が指定されたホストで動作していません。

対処方法**:** 接続しようとするホストで CIM Object Manager が動作していることを確認します。そのホストで CIM Object Manager が動作していない場合は、CIM Object Manager が動作しているホストに接続します。

`NO_EVENT_PROVIDER`
説明**:** イベントプロバイダが見つかりません。

原因**:** プロパティプロバイダクラスが見つかりません。

対処方法: CIM Object Manager のクラスパスにプロバイダクラスのパラメータ、プロバイダが定義された指示クラス、および Java プロバイダクラス名が含まれることを確認します。CIM Object Manager の Solaris プロバイダが設定されており、プロバイダ修飾子が正しいことを確認します。

`NOT_EVENT_PROVIDER`
説明: このエラーメッセージは、パラメータを使用しません。

原因: クラスパスに存在するプロバイダクラスが `EventProvider` インタフェースを実装していません。

対処方法: プロバイダが正しいことを確認し、プロバイダ実装を登録します。

`NO_INSTANCE_PROVIDER`
説明: このエラーメッセージは、次の 2 つのパラメータを使用します。

- `{0}` は、インスタンスプロバイダが見つからないクラス名に置換されます。
- `{1}` は、指定されたインスタンスプロバイダ名に置換されます。

原因: 指定されたインスタンスプロバイダの Java クラスが見つかりません。このエラーメッセージは、CIM Object Manager のクラスパスに以下の 1 つまたは複数の項目が含まれていないことを示します。

- プロバイダクラス名
- プロバイダクラスのパラメータ
- プロバイダが定義される CIM クラス

対処方法: CIM Object Manager のクラスパスを設定します。CIM Object Manager の Solaris プロバイダが設定されており、プロバイダ修飾子が正しいことを確認します。

`NO_METHOD_PROVIDER`
説明: このエラーメッセージは、次の 2 つのパラメータを使用します。

- `{0}` は、メソッドプロバイダが見つからないクラス名に置換されます。
- `{1}` は、指定されたメソッドプロバイダクラスの名前に置換されます。

原因: 指定されたメソッドプロバイダの Java クラスが見つかりません。このエラーメッセージは、CIM Object Manager のクラスパスに以下の 1 つまたは複数の項目が含まれていないことを示します。

- プロバイダクラス名
- プロバイダクラスのパラメータ
- プロバイダが定義される CIM クラス

対処方法: CIM Object Manager のクラスパスを設定します。CIM Object Manager の Solaris プロバイダが設定されており、プロバイダ修飾子が正しいことを確認します。

NO_OVERRIDDEN_METHOD
説明: このエラーメッセージは、次の 2 つのパラメータを使用します。

- {0} は、{1} で示されるメソッドをオーバーライドしたメソッド名に置換されます。
- {1} は、オーバーライドされたメソッド名に置換されます。

原因: サブクラスのメソッドがスーパークラスのメソッドのオーバーライドを試みていますが、そのスーパークラスのメソッドはクラス階層に定義されていないため存在しません。

メソッドをオーバーライドすると、その実装と署名もオーバーライドされます。

対処方法: スーパークラス内にそのメソッドが存在することを確認します。

NO_OVERRIDDEN_PROPERTY
説明: このエラーメッセージは、次の 2 つのパラメータを使用します。

- {0} は、{1} をオーバーライドしたプロパティ名に置換されます。
- {1} は、プロパティをオーバーライドする名前に置換されます。

原因: サブクラスのプロパティがスーパークラスのプロパティのオーバーライドを試みていますが、スーパークラスのプロパティはクラス階層に定義されていないため、存在しません。

対処方法: スーパークラスにそのプロパティが存在することを確認します。

NO_PROPERTY_PROVIDER
説明: このエラーメッセージは、次の 2 つのパラメータを使用します。

- {0} は、プロパティプロバイダが見つからないクラス名に置換されます。
- {1} は、指定されたプロパティプロバイダの名前に置換されます。

原因: 指定されたプロパティプロバイダの Java クラスが見つかりません。このエラーメッセージは、CIM Object Manager のクラスパスに以下の 1 つまたは複数の項目が含まれていないことを示します。

- プロバイダクラス名
- プロバイダクラスのパラメータ
- プロバイダが定義される CIM クラス

対処方法: CIM Object Manager クラスパスを設定します。接続しようとするホストで CIM Object Manager が動作していることを確認します。そのホストで CIM Object Manager が動作していない場合、CIM Object Manager が動作しているホストに接続します。

NO_QUALIFIER_VALUE
説明: このエラーメッセージは、次の 2 つのパラメータを使用します。

- {0} は、要素 {1} を変更する修飾子の名前に置換されます。

- {1} は、修飾子の参照先である要素です。{1} は、修飾子に応じて、クラス、プロパティ、メソッド、または参照のいずれかになります。

原因: プロパティまたはメソッドに修飾子が指定されましたが、修飾子に値が含まれていません。たとえば、修飾子 VALUES には文字列配列を指定する必要があります。必要な文字列配列なしで VALUES 修飾子が指定されると、NO_QUALIFIER_VALUE のエラーメッセージが表示されます。

対処方法: 修飾子に必要なパラメータを指定します。各修飾子に必要な属性については、http://www.dmtf.org の DMTF CIM Specification を参照してください。

NO_SUCH_METHOD

説明: このエラーメッセージは、次の 2 つのパラメータを使用します。

- {0} は、指定されたメソッド名に置換されます。
- {1} は、指定されたクラス名に置換されます。

原因: 指定されたクラスにメソッドが定義されなかったことが考えられます。指定されたクラスにメソッドが定義されている場合には、定義内でスペルミスまたは入力ミスにより別のメソッドが指定されている可能性があります。

対処方法: 指定されたクラスのメソッドを定義します。メソッド名とクラス名が正しく入力されているかを確認します。

NO_SUCH_PRINCIPAL

説明: このエラーメッセージはパラメータ {0} を使用しますが、このパラメータは主体 (ユーザーアカウント) の名前に置換されています。

原因: 指定されたユーザーアカウントが見つかりません。ユーザー名の入力にスペルミスがあったか、あるいはそのユーザーにユーザーアカウントが設定されていません。

対処方法: ログイン時にユーザー名を正しく入力します。そのユーザーにユーザーアカウントが設定されていることを確認します。

NO_SUCH_QUALIFIER1

説明: このエラーメッセージはパラメータ {0} を使用しますが、このパラメータは未定義の修飾子の名前に置換されています。

原因: 新しい修飾子が指定されましたが、この修飾子は拡張スキーマの一部として定義されていません。特定のクラスのプロパティまたはメソッドに有効な修飾子として認識されるように、この修飾子を CIM スキーマまたは拡張スキーマの一部として定義する必要があります。

対処方法: この修飾子を拡張スキーマの一部として定義するか、あるいは標準の CIM 修飾子を使用します。標準の CIM 修飾子と CIM スキーマの修飾子の使用法については、http://www.dmtf.org の DMTF CIM Specification を参照してください。

NO_SUCH_QUALIFIER2
説明: このエラーメッセージは、次の 2 つのパラメータを使用します。

- {0} は、修飾子が変更する、クラス、プロパティ、またはメソッドの名前に置換されます。
- {1} は、見つからない修飾子の名前に置換されます。

原因: 特定のクラスのプロパティまたはメソッドを変更するために新しい修飾子が指定されましたが、その修飾子は拡張スキーマの一部として定義されていません。特定のクラスのプロパティまたはメソッドに有効な修飾子として認識されるように、この修飾子を CIM スキーマまたは拡張スキーマの一部として定義する必要があります。

対処方法: この修飾子を拡張スキーマの一部として定義するか、あるいは標準の CIM 修飾子を使用します。標準の CIM 修飾子と CIM スキーマの修飾子の使用法については、http://www.dmtf.org の DMTF CIM Specification を参照してください。

NO_SUCH_ROLE
説明: このエラーメッセージはパラメータ {0} を使用しますが、このパラメータは役割名に置換されています。

原因: 指定された役割が見つからないか、役割 ID ではありません。

対処方法: 入力した役割が存在するかどうか確認してください。役割が必要な場合は、担当のシステム管理者にお問い合わせください。

NO_SUCH_SESSION
説明: このエラーメッセージはパラメータ {0} を使用しますが、このパラメータはセッション識別子に置換されています。

原因: セッションが閉じているか、すぐあとに使用されます。

対処方法: セッションを閉じないでください。

NOT_HELLO
説明: このエラーメッセージは、パラメータを使用しません。

原因: このエラーメッセージは、hello メッセージ (CIM Object Manager に送信される最初のメッセージ) 内のデータが壊れている場合に表示されます。

対処方法: このエラーメッセージに対しての解決方法はありません。Solaris WBEM Services のセキュリティ機能についての詳細は、『*Solaris WBEM Services* 管理ガイド』の「セキュリティの管理 (手順)」を参照してください。

NOT_INSTANCE_PROVIDER
説明: このエラーメッセージは、次の 2 つのパラメータを使用します。

- {0} は、InstanceProvider インタフェースを定義しようとするインスタンス名に置換されます。

- {1} は、InstanceProvider インタフェースを実装していない Java プロバイダクラスの名前に置換されます。指定されたクラスのインスタンスをすべて列挙するには、InstanceProvider インタフェースを実装する必要があります。

原因: CLASSPATH 環境変数で指定されている Java プロバイダクラスのパスに、InstanceProvider インタフェースが実装されていません。

対処方法: プロバイダが正しくプロバイダ実装を登録していることを確認します。

NOT_METHOD_PROVIDER

説明: このエラーメッセージは、次の 2 つのパラメータを使用します。

- {0} は、MethodProvider インタフェースを定義しようとするメソッド名に置換されます。MethodProvider インタフェースが定義されると、指定されたメソッドがプログラム内で実装され、実行されます。
- {1} は、MethodProvider インタフェースを実装していない Java プロバイダクラスの名前に置換されます。

原因: クラスパスに存在する Java プロバイダクラスが MethodProvider インタフェースを実装していません。

対処方法: クラスパスに存在する Java プロバイダクラスが MethodProvider インタフェースを実装していることを確認します。プロバイダを宣言するには、クラス定義に public Solaris implements MethodProvider を使用します。

NOT_PROPERTY_PROVIDER

説明: このエラーメッセージは、次の 2 つのパラメータを使用します。

- {0} は、PropertyProvider インタフェースを定義しようとするメソッド名に置換されます。PropertyProvider インタフェースは、指定されたプロパティの値の取得に使用されます。
- {1}は、PropertyProvider インタフェースを実装していない Java プロバイダクラスの名前に置換されます。

原因: クラスパスに存在する Java プロバイダクラスが PropertyProvider インタフェースを実装していません。

対処方法: クラスパスに存在する Java プロバイダクラスが PropertyProvider インタフェースを実装していることを確認します。プロバイダを宣言するには、クラス定義に public Solaris implements PropertyProvider を使用します。

NOT_RESPONSE

説明: このエラーメッセージは、パラメータを使用しません。

原因: このエラーメッセージは、CIM Object Manager からの最初の応答メッセージにあるデータが壊れている場合に表示されます。

対処方法: このエラーメッセージに対しての解決方法はありません。Solaris WBEM Services のセキュリティ機能についての詳細は、『*Solaris WBEM Services* 管理ガイド』の「セキュリティの管理 (手順)」を参照してください。

PROPERTY_OVERRIDDEN
説明: このエラーメッセージは、次の 3 つのパラメータを使用します。

- {0} は、パラメータ {1} によって示されるプロパティをオーバーライドしようとするプロパティ名に置換されます。
- {1} は、すでにオーバーライドされているプロパティ名に置換されます。
- {2} は、パラメータ {1} で示されるプロパティをオーバーライドしたプロパティ名に置換されます。

原因: 別のプロパティによってすでにオーバーライドされているプロパティをオーバーライドしようとするプロパティが指定されました。いったんオーバーライドしたプロパティを再度オーバーライドすることはできません。

対処方法: オーバーライドする別のプロパティを指定します。

QUALIFIER_UNOVERRIDABLE
説明: このエラーメッセージは、次の 2 つのパラメータを使用します。

- {0} は、DisableOverride フレーバが設定されている修飾子の名前に置換されます。
- {1} は、{0} によって無効になるように設定されている修飾子の名前に置換されます。

原因: オーバーライドされた修飾子には、DisableOverride フレーバが設定されています。

対処方法: この修飾子の特性を、EnableOverride または Override=True に設定し直します。

REF_REQUIRED
説明: このエラーメッセージはパラメータ {0} を使用しますが、このパラメータはクラス名に置換されています。

原因: 関連を持つようにあるクラスが定義されましたが、参照が指定されていません。CIM では、関連は 1 つまたは複数の参照を含む必要があります。

対処方法: 参照を追加するか、関連修飾子を削除します。

SCOPE_ERROR
説明: このエラーメッセージは、次の 3 つのパラメータを使用します。

- {0} は、指定された修飾子が変更するクラス名に置換されます。
- {1} は、指定された修飾子の名前に置換されます。
- {2} は、修飾子が変更する属性の種類に置換されます。

原因: 修飾子の種類の定義に準拠しない方法で修飾子が指定されました。[READ] 修飾子のスコープは、プロパティを変更するように [READ] 修飾子に指示する定義です。[READ] 修飾子がそのスコープの指示以外の方法で使用されると (たとえばメソッドを変更するように指定されるなど)、SCOPE_ERROR メッセージが返されます。

注 – CIM は、CIM 修飾子が変更できる CIM 要素の種類を定義しています。修飾子の使用方法についてのこの定義は、修飾子のスコープと呼ばれます。ほとんどの修飾子は、プロパティまたはメソッド、あるいはこの両方の変更を指示するスコープを持ちます。また、ほとんどの修飾子は、パラメータ、クラス、関連、インジケーション、またはスキーマの変更を指示するスコープを持ちます。

対処方法: 指定された修飾子のスコープを確認します。CIM 修飾子の標準の定義については、http://www.dmtf.org の DMTF CIM Specification を参照してください。別の修飾子を使用するか、あるいは CIM 定義に従って修飾子を使用するようにプログラムを変更します。

TYPE_ERROR

説明: このエラーメッセージは、次の 5 つのパラメータを使用します。

- {0} は、指定された要素 (プロパティ、メソッド、修飾子など) の名前に置換されます。
- {1} は、指定された要素が属するクラス名に置換されます。
- {2} は、要素に定義されたデータ型に置換されます。
- {3} は、割り当てられた値のデータ型に置換されます。
- {4} は、割り当てられた実際の値に置換されます。

原因: プロパティまたはメソッドのパラメータ値と、定義されたそのデータ型が一致しません。

対処方法: プロパティまたはメソッドの値を、定義されたそのデータ型に一致させます。

UNKNOWNHOST

説明: このエラーメッセージはパラメータ {0} を使用しますが、このパラメータはホスト名に置換されています。

原因: 指定されたホストが呼び出されました。指定されたホストが使用できないか、あるいはこのホストが置かれていません。ホスト名にスペルミスがあるか、このホストコンピュータが別のドメインに移されたか、あるいはドメインに属するホスト名のリストに登録されていないか、システム状況が原因でそのホストが一時的に使用できない可能性があります。

対処方法**:** ホスト名のスペルに入力ミスがないか確認します。`ping` コマンドを使用して、そのホストコンピュータが応答していること、そのホストのシステム状況を確認します。また、そのホストが指定されたドメインに属していることを確認します。

`VER_ERROR`

説明**:** このエラーメッセージはパラメータ `{0}` を使用しますが、このパラメータは動作中のCIM Object Manager のバージョン番号に置換されています。

原因**:** CIM Object Manager は、接続しようとしているクライアントのバージョンをサポートしていません。

対処方法**:** サポートされているバージョンをインストールします。

付録 B

Solaris スキーマ

デフォルトでは、Solaris スキーマおよび CIM スキーマは、CIM Object Manager で使用できます。Solaris スキーマおよび CIM スキーマをコンパイルした MOF ファイルは、`/usr/sadm/mof/` で確認できます。CIM のコアモデルおよび共通モデルを実装した CIM スキーマファイルのファイル名には、「CIM」が使用されています。Solaris スキーマファイルのファイル名には、「Solaris」が使用され、Sun が CIM 用に定めた一意の拡張子が付けられます。

この章に一覧を表示した Solaris プロバイダのマニュアルは、指定されたプロバイダの MOF ファイルに含まれています。

- 141 ページの「`Solaris_Schema1.0.mof` ファイル」
- 141 ページの「`Solaris_CIMOM1.0.mof` ファイル」
- 142 ページの「`Solaris_Core1.0.mof` ファイル」
- 142 ページの「`Solaris_Application1.0.mof` ファイル」
- 143 ページの「`Solaris_System1.0.mof` ファイル」
- 144 ページの「`Solaris_Device1.0.mof` ファイル」
- 144 ページの「`Solaris_Acl1.0.mof` ファイル」
- 145 ページの「`Solaris_Network1.0.mof` ファイル」
- 145 ページの「`Solaris_Users1.0.mof` ファイル」
- 145 ページの「`Solaris_Event1.0.mof` ファイル」
- 145 ページの「`Solaris_SNMP1.0.mof` ファイル」
- 146 ページの「`Solaris_VM1.0.mof` ファイル」
- 147 ページの「`Solaris_Project1.0.mof` ファイル」
- 147 ページの「`Solaris_Performance1.0.mof` ファイル」

Solaris スキーマファイル

次の表では、`/usr/sadm/mof` の Solaris スキーマファイルの概要について説明します。

表 **B–1** Solaris スキーマファイル

Solaris スキーマファイル	説明
`Solaris_Schema1.0.mof`	`#pragma Include` 文で Solaris スキーマのすべての MOF ファイルを一覧表示する。MOF ファイルの読み取りおよびコンパイルの順序を指定する
`Solaris_CIMOM1.0.mof`	CIM Object Manager の構成情報が含まれる
`Solaris_Core1.0.mof`	WBEM コア機能の実装を可能にする。ロケール、修飾子、およびプロバイダの設定を可能にする
`Solaris_Application1.0.mof`	Solaris パッケージおよびパッチを CIM でモデル化する
`Solaris_System1.0.mof`	オペレーティングシステムおよびシステムプロセスなど、システムの Solaris スキーマコンポーネントをモデル化する。`Solaris_Process` および `Solaris_OperatingSystem` クラスの定義を使用して、CIM スキーマの定義を拡張する
`Solaris_Device1.0.mof`	CIM Object Manager がコンピュータで動作するように、システムのプロセッサ、シリアルポート、出力デバイス、および時間設定について説明する
`Solaris_Acl1.0.mof`	WBEM アクセス制御リスト (ACL) セキュリティのクラスを含む
`Solaris_Network1.0.mof`	ネットワークドメイン、IP サブネット、およびネームサービス (NIS、NIS+、LDAP、DNS、およびサーバー `/etc` ファイルなど) に関連するクラスを定義する
`Solaris_Users1.0.mof`	ユーザーアカウントを使用するクラスを定義する
`Solaris_Event1.0.mof`	一意の Solaris インジケーションハンドラを定義する。このファイルに定義されたクラスを使用すると、Sun が実装する CIM の RMI プロトコルを介して、インジケーションの管理クライアントへの配信が容易になる
`Solaris_SNMP1.0.mof`	SNMP デバイスの構成情報に関連するクラスを定義する
`Solaris_VM1.0.mof`	記憶デバイスに関連するクラスを定義する
`Solaris_Project1.0.mof`	Solaris プロジェクトデータベースをモデル化するクラスを定義する
`Solaris_Performance1.0.mof`	コンピューティングリソースの基準値に関連するクラス、つまりそれぞれのユーザーおよびプロジェクトのコンピューティングリソースの使用と性能に関連したクラスを定義する

次の節では、各スキーマの内容について詳しく説明します。

Solaris_Schema1.0.mof ファイル

Solaris_Schema1.0.mof ファイルは、Solaris スキーマを構成する、他のすべての MOF ファイルのハイレベルコンテナです。このファイルでは、MOF ファイルをコンパイルの必要な順序で一覧表示します。

各コンパイルの結果生成された Java クラスは、CIM Object Manager に送信されます。ここで Java クラスは、イベントとして動作するか、CIM Object Manager Repository に送信されてオブジェクトとして保存されます。次の Solaris_Schema1.0.mof ファイルリストには、Include 文がコンパイルに必要な順序で一覧表示されます。

```
/*
タイトル         Solaris Master MOF 1.0
説明     ほかの MOF すべての pragma に含める
日付         03/10/01
バージョン      1.1
Copyright   (c) 2000 Sun Microsystems, Inc. All Rights Reserved.
*/

#pragma Include ("Solaris_Core1.0.mof")
#pragma Include ("Solaris_Application1.0.mof")
#pragma Include ("Solaris_System1.0.mof")
#pragma Include ("Solaris_Device1.0.mof")
#pragma Include ("Solaris_Network1.0.mof")
#pragma Include ("Solaris_Users1.0.mof")
#pragma Include ("Solaris_Project1.0.mof")
#pragma Include ("Solaris_Event1.0.mof")
#pragma Include ("Solaris_CIMOM1.0.mof")
#pragma Include ("Solaris_SNMP1.0.mof")

// これは CIM 名前空間を変更するため、最後に含める必要がある。
#pragma Include ("Solaris_Acl1.0.mof")
```

コンパイラは、Solaris_Schema1.0.mof ファイルの行を構文解析して、Include 文に指定されているファイルをコンパイルします。次に Solaris_Schema1.0.mof ファイルの次の行を構文解析します。指定されたすべてのファイルをコンパイルするまでこの処理が続行されます。

Solaris_CIMOM1.0.mof ファイル

Solaris_CIMOM1.0.mof ファイルには、CIM Object Manager が使用するすべてのシステムプロパティが含まれます。

Solaris_CIMOM1.0.mof ファイルは、次のクラスを定義します。

- CIM_ObjectManager

- CIM_ObjectManagerCommunicationMechanism
- CIM_WBEMCommunicationMechanism
- Solaris_CIMOM
- Solaris_ObjectManagerClientProtocolAdapter
- Solaris_ObjectManagerProtocolAdapter
- Solaris_ObjectManagerProviderProtocolAdapter
- Solaris_ProviderPath

さらに、Solaris_CIMOM1.0.mof ファイルは、関連クラスの CIM _CommMechanismForManager を定義します。

Solaris_Core1.0.mof ファイル

Solaris_Core1.0.mof ファイルは、Solaris_Schema1.0.mof ファイルの次に最初にコンパイルされる Solaris スキーマファイルです。このファイルによって、Solaris プロバイダの Solaris_ComputerSystem クラスを定義できます。

Solaris_Core1.0.mof ファイルでは、次のクラスが定義されます。

- Solaris_ComputerSystem
- Solaris_LogRecord
- Solaris_LogService
- Solaris_Product
- Solaris_SystemDownStatisticalInformation
- Solaris_SystemUpStatisticalInformation

また Solaris_Core1.0.mof ファイルでは、次の関連クラスが定義されます。

- Solaris_ProductParentChild
- Solaris_ProductProductDependency
- Solaris_SystemSetting

Solaris_Application1.0.mof ファイル

Solaris_Application1.0.mof ファイルを使用すると、Solaris スキーマを拡張するパッケージやパッチをアプリケーションに設定できます。

Solaris_Application1.0.mof ファイルでは、次のクラスが定義されます。

- Solaris_InstalledSoftwareElement
- Solaris_Package
- Solaris_Patch
- Solaris_RegistrySoftwareElement
- Solaris_SoftwareElement
- Solaris_SoftwareFeature

また Solaris_Application1.0.mof ファイルでは、次の関連クラスが定義されます。

- Solaris_PatchPackageDependency
- Solaris_PatchToPatchDependency
- Solaris_ProductSoftwareElementDependency
- Solaris_ProductSoftwareElements
- Solaris_ProductSoftwareFeatureDependency
- Solaris_ProductSoftwareFeatures
- Solaris_RegistryElementDependency
- Solaris_SoftwareElementDependency
- Solaris_SoftwareElementProductDependency
- Solaris_SoftwareElementSoftwareFeatureDependency
- Solaris_SoftwareFeatureDependency
- Solaris_SoftwareFeatureParentChild
- Solaris_SoftwareFeatureProductDependency
- Solaris_SoftwareFeatureSoftwareElementDependency
- Solaris_SoftwareFeatureSoftwareElements

Solaris_System1.0.mof ファイル

Solaris_System1.0.mof ファイルでは、次のクラスが定義されます。

- Solaris_CpuSysinfo
- Solaris_CpuUtilizationInformation
- Solaris_CpuVminfo
- Solaris_DataFile
- Solaris_DiskIOInformation
- Solaris_DisklessClient
- Solaris_Eeprom
- Solaris_EepromSetting
- Solaris_InstalledOS
- Solaris_JobScheduler
- Solaris_JobScheduler_Cron
- Solaris_OperatingSystem
- Solaris_OSProcess
- Solaris_OsService
- Solaris_Process
- Solaris_RunningOS
- Solaris_ScheduledJob
- Solaris_ScheduledJob_Cron

また Solaris_System1.0.mof ファイルでは、次の関連クラスが定義されます。

- Solaris_EepromElementSetting
- Solaris_HostedJobScheduler
- Solaris_OwningJobScheduler
- Solaris_SystemDevice

Solaris_Device1.0.mof ファイル

Solaris_Device1.0.mof ファイルでは、次のクラスが定義されます。

- Solaris_Environment
- Solaris_EthernetAdapter
- Solaris_Keyboard
- Solaris_LogEntry
- Solaris_LogServiceProperties
- Solaris_LogServiceSetting
- Solaris_MessageLog
- Solaris_MessageLogRecord
- Solaris_MessageLogSetting
- Solaris_Printer
- Solaris_PrintJob
- Solaris_PrintQueue
- Solaris_PrintSAP
- Solaris_PrintService
- Solaris_Processor
- Solaris_SerialPort
- Solaris_SerialPortConfiguration
- Solaris_SerialPortSetting
- Solaris_SoundDevice
- Solaris_SyslogRecord
- Solaris_TimeZone

また Solaris_Device1.0.mof ファイルでは、次の関連クラスが定義されます。

- Solaris_CpuSysinfoPerformanceMonitor
- Solaris_CpuUtilizationPerformanceMonitor
- Solaris_CpuVminfoPerformanceMonitor
- Solaris_LogInDataFile
- Solaris_OwningPrintQueue
- Solaris_PrinterServicingQueue
- Solaris_QueueForPrintService
- Solaris_RecordInLog
- Solaris_SystemTimeZone

Solaris_Acl1.0.mof ファイル

Solaris_Acl1.0.mof ファイルでは、Solaris WBEM Services セキュリティクラスが指定されます。このファイルでは、ACL、ユーザー、および名前空間の次の基底クラスが定義されます。

- Solaris_Acl
- Solaris_NamespaceAcl
- Solaris_UserAcl

Solaris_Network1.0.mof ファイル

Solaris_Network1.0.mof ファイルでは、ネットワークドメイン、IP サブネット、およびネーミングサービス (NIS、NIS+、LDAP、DNS、およびサーバー /etc ファイルなど) に関連するクラスが定義されます。Solaris_Network1.0.mof ファイルでは、次のクラスが定義されます。

- Solaris_AdminDomain
- Solaris_DnsAdminDomain
- Solaris_IPProtocolEndpoint
- Solaris_IPSubnet
- Solaris_LdapAdminDomain
- Solaris_NisAdminDomain
- Solaris_NisplusAdminDomain
- Solaris_SystemAdminDomain

Solaris_Users1.0.mof ファイル

Solaris_Users1.0.mof ファイルでは、次のクラスが定義されます。

- Solaris_AuthorizationAttribute
- Solaris_EmailAlias
- Solaris_ExecutionProfile
- Solaris_MailBox
- Solaris_ProfileAttribute
- Solaris_ShellSAP
- Solaris_UserAccount
- Solaris_UserGroup
- Solaris_UserHomeDirectory
- Solaris_UserTemplate

Solaris_Event1.0.mof ファイル

Solaris_Event1.0.mof ファイルには、Solaris に一意のインジケーションハンドラを処理するクラスが含まれます。この Solaris インジケーションハンドラは、CIM_IndicationHandler のサブクラスです。このサブクラスには、Solaris_RMIDelivery および Solaris_JAVAXRMIDelivery が含まれます。クライアント RMI プロトコルでは、Solaris_JAVAXRMIDelivery ハンドラが使用されます。Solaris_Event1.0.mof ファイルには、以前のバージョンの WBEM と互換性を保つために Solaris_RMIDelivery が含まれます。

Solaris_SNMP1.0.mof ファイル

Solaris_SNMP1.0.mof ファイルでは、SNMP デバイスの構成情報に関連するクラスが定義されます。Solaris_SNMP1.0.mof ファイルでは、次のクラスが定義されます。

- Solaris_SNMPGroupConf
- Solaris_SNMPSystem
- Solaris_SNMPSystemConf

Solaris_VM1.0.mof ファイル

Solaris_VM1.0.mof ファイルでは、次のような記憶デバイスに関連するクラスが定義されます。

- スライス内の状態データベースの複製
- データ用に使用可能な記憶エクステントのエクステント範囲
- ストライプ
- 連結ストライプ
- ミラー
- RAID Level 5 デバイス
- UFS ログファイルシステム
- スペアプール
- ディスクセット
- 記憶装置ボリューム

Solaris_VM1.0.mof ファイルでは、次のクラスが定義されます。

- Solaris_Directory
- Solaris_DiskDrive
- Solaris_DiskPartition
- Solaris_HSFS
- Solaris_LocalFileSystem
- Solaris_MediaPresent
- Solaris_NFS
- Solaris_UFS
- Solaris_VMConcat
- Solaris_VMDiskSet
- Solaris_VMExtent
- Solaris_VMHotSparePool
- Solaris_VMMirror
- Solaris_VMRaid5
- Solaris_VMSoftPartition
- Solaris_VMStateDatabase
- Solaris_VMStorageVolume
- Solaris_VMStripe
- Solaris_VMTrans

Solaris_VM1.0.mof ファイルでは、次の関連クラスが定義されます。

- Solaris_DiskIOPerformanceMonitor
- Solaris_HSFSMount
- Solaris_Mount
- Solaris_NFSExport

- Solaris_NFSMount
- Solaris_UFSMount
- Solaris_VMConcatComponent
- Solaris_VMDriveInDiskSet
- Solaris_VMExtentBasedOn
- Solaris_VMExtentInDiskSet
- Solaris_VMHostInDiskSet
- Solaris_VMHotSpareInUse
- Solaris_VMHotSpares
- Solaris_VMMirrorSubmirrors
- Solaris_VMRaid5Component
- Solaris_VMSoftPartComponent
- Solaris_VMStatistics
- Solaris_VMStripeComponent
- Solaris_VMTransLog
- Solaris_VMTransMaster
- Solaris_VMUsesHotSparePool
- Solaris_VMVolumeBasedOn

Solaris_Project1.0.mof ファイル

Solaris_Project1.0.mof ファイルでは、Solaris プロジェクトデータベースを表すクラスが定義されます。

Solaris_Project1.0.mof ファイルでは、クラス Solaris_Project が定義されます。また Solaris_Project1.0.mof ファイルでは、次の関連クラスが定義されます。

- Solaris_ProjectGroup
- Solaris_ProjectUser

Solaris_Performance1.0.mof ファイル

Solaris_Performance1.0.mof ファイルでは、コンピューティングリソースの基準値に関連するクラス、つまりそれぞれのユーザーおよびプロジェクトのコンピューティングリソースの使用と性能に関連したクラスを定義します。

Solaris_Performance1.0.mof ファイルでは、次のクラスが定義されます。

- Solaris_ActiveProject
- Solaris_ActiveUser
- Solaris_ProcessStatisticalInformation
- Solaris_ProjectProcessAggregateStatisticalInformation
- Solaris_UserProcessAggregateStatisticalInformation

また Solaris_Performance1.0.mof ファイルでは、次の関連クラスが定義されます。

- Solaris_ActiveProjectProcessAggregateStatistics
- Solaris_ActiveUserProcessAggregateStatistics
- Solaris_ProcessStatistics
- Solaris_ProjectProcessStatistics
- Solaris_UserProcessStatistics

索引

C

CIM Object Manager
  Repository, 141
  エラーメッセージ, 123
CIM Workshop
  起動, 24
CIM 修飾子, 設定, 73
CIM スキーマ, ファイル, 139
Solaris WBEM Services エラーメッセージ
  「エラーメッセージ」を参照

D

Distributed Management Task Force, 19

J

Java
  Solaris WBEM SDK サンプルプログラム, 117
  Java プログラムとネイティブメソッドの統合, 105
  インスタンスの削除, 52
  インスタンスの作成, 51
  インスタンスの設定, 54

Solaris WBEM SDK エラーメッセージ
  「エラーメッセージ」を参照
Solaris WBEM SDK プログラム例
  プログラム例を参照

M

Managed Object Format, 説明, 21
Managed Object Format (MOF), Solaris スキーマ, 139
MOF (Managed Object Format), 基底クラスの作成, 68
`mofcomp` コマンド, セキュリティ上の注意事項, 33
MOF ファイル, コンパイルに関するセキュリティ上の注意事項, 33

S

Solaris WBEM SDK, プログラミング作業, 48
Solaris スキーマ, ファイル, 139

W

WBEM, 定義, 19

Workshop
CIM Workshopを参照

あ
アクセス制御
設定
名前空間, 72
ユーザー, 71
アプリケーションプログラミングインタフェース (API)
CIM 修飾子の設定, 73
インスタンスの削除, 52
インスタンスの作成, 51
インスタンスの設定, 54
概要, 22
クラスの削除, 69
クラスの取得, 66
名前空間の作成, 67
名前空間の列挙, 59
プログラミング作業, 48
メソッドの呼び出し, 65

い
インスタンス
削除, 51
作成, 51
取得と設定, 53

え
エラーメッセージ, 123

き
基底クラス, 作成, 68

く
クラス
CIMClass, 68
deleteClass, 68
クラス (続き)
削除, 69
作成, 68
取得, 66
セキュリティ, 70

し
修飾子
型宣言の例, 73
キー, 68
定義, 73

す
スキーマ
CIM
ファイル, 139
Solaris
ファイル, 139
Solaris スキーマ, 139

せ
セキュリティ名前空間, 48

た
単一のプロバイダ, 93

て
デフォルト名前空間, 48

な
名前空間
アクセス制御の設定, 70
作成, 67
デフォルト, 49
列挙, 59

ふ

プロバイダ
  CIM Object Manager による登録, 106
  インタフェース, 94
  ネイティブプロバイダの作成, 104

め

メソッド
  deleteInstance, 51
  enumNameSpace, 59
  getClass, 65
  getInstance, 53
  invokeMethod, 64
  名前空間の削除, 68
  呼び出し, 65
メソッド、createInstance, 70

れ

例
  CIM 修飾子の設定, 73
  インスタンスの削除, 52
  インスタンスの作成, 51
  インスタンスの設定, 54
  クラスの削除, 69
  クラスの取得, 66
  名前空間の作成, 67
  名前空間の列挙, 59
  メソッドの呼び出し, 65
例外
  「エラーメッセージ」を参照