Solaris WBEM 開発ガイド

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

Part No: 817–2465–10
2003 年 8 月

原典: *Solaris WBEM Developer's Guide*

Part No: 816-4813-10

Revision A

031007@6671

目次

表目次

図目次

例目次

はじめに

『Solaris WBEM 開発ガイド』では、CIM (Common Information Model) の概念について説明します。Solaris™ オペレーティング環境での WBEM (Web-Based Enterprise Management) サービスの管理方法についても説明します。

またこのマニュアルでは、WBEM SDK (Solaris Web-Based Enterprise Management ソフトウェア開発キット) についても説明します。開発者は、WBEM SDK を利用して、Solaris オペレーティング環境のリソースを管理する標準ベースのアプリケーションを作成できます。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 の構成要素は次のとおりです。

- Distributed Management Task Force (DMTF) の CIM (Common Information Model) により、リソースの記述および管理を行うクライアントアプリケーションプログラミングインタフェース (API)。
- 管理リソース (管理対象のリソース) 上の動的データを取得および設定するプロバイダ API。
- WBEM クライアントおよびプロバイダプログラムのサンプル。
- システム上に管理リソースを作成したり表示したりできる Java™ プログラミング言語で作成された CIM Workshop アプリケーション。

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 オブジェクトマネージャ (CIM Object Manager) に伝えるようなソフトウェアを作成します。

- システムおよびネットワークアプリケーション開発者 – CIM クラスおよびインスタンスに格納されている情報を管理するアプリケーションを作成します。 これらの開発者は、Solaris WBEM サービス (Solaris WBEM Services) API を使って、CIM インスタンスおよびクラスのプロパティを取得または設定します。

このマニュアルをお読みになる前に

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 プログラミング言語
- CIM の概念
- ネットワーク管理の概念

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。

- 『*The Java Programming Language*』第 2 版、Ken Arnold、James Gosling 共著、Addison-Wesley、ISBN 0–201–31006–6.
- 『*The Java Class Libraries*』第 2 版、第 1 巻、Patrick Chan、Rosanna Lee、Douglas Kramer 共著、Addison-Wesley、ISBN 0–201–31002–3
- Distributed Management Task Force の「*CIM Tutorial*」(`http://www.dmtf.org/education/cimtutorial.php`)

次に、WBEM 技術に携わる場合に有用な Web サイトを示します。

- 「CIM Tutorial Glossary」– `http://www.dmtf.org/education/cimtutorial/reference/glossary.php`
- 「Distributed Management Task Force (DMTF)」– `http://www.dmtf.org`

  このサイトには、CIM の最新情報、各種の作業グループについての情報、および CIM スキーマの拡張についての情報が掲載されています。
- 「Rational Software」– `http://www.rational.com/uml`

  このサイトでは、Unified Modeling Language (UML) および Rose CASE ツールの関連文書を入手できます。

このマニュアルの構成

第 1 章では、WBEM (Web-Based Enterprise Management)、CIM (Common Information Model)、Solaris WBEM SDK のアプリケーションプログラミングインタフェース (API)、CIM Workshop について簡単に説明します。

第 2 章では、CIM オブジェクトマネージャについて説明します。CIM オブジェクトマネージャの起動および停止方法、CIM オブジェクトマネージャリポジトリのアップグレード方法についても取り上げます。

第 3 章では、Solaris WBEM SDK 付属のサンプルプログラムについて説明します。

第 4 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 5 章では、WQL (WBEM Query Language) と Query API による照会の作成および処理方法について説明します。

第 6 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 7 章では、MOF コンパイラの使用方法について説明します。

第 8 章では、WBEM のセキュリティ機構と CIM オブジェクトマネージャの機能について説明します。

第 9 章では、ログデータの表示方法と、Solaris WBEM SDK のコンポーネントによって生成されたエラーメッセージの意味を説明します。

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

関連情報

次の関連マニュアルも参照してください。

- 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。

一般規則

- このマニュアルでは、英語環境での画面イメージを使っています。このため、実際に日本語環境で表示される画面イメージとこのマニュアルで使っている画面イメージが異なる場合があります。本文中で画面イメージを説明する場合には、日本語のメニュー、ボタン名などの項目名と英語の項目名が、適宜併記されています。

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。

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、管理テクノロジとインターネットテクノロジを組み合わせたものです。企業のコンピューティング環境の管理を統合します。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。

DMTF (Distributed Management Task Force) は、コンピュータならびに情報通信業界の企業を代表するグループです。デスクトップ環境、エンタープライズ環境、インターネットの管理に関する標準規格の開発および普及において主導的な立場にあります。DMTF の目的は、異なったプラットフォーム間で異なったプロトコルを使用するコンピュータおよびネットワークの統合的な管理方法を開発することです。その結果、完全な相互運用性を備えた、費用効果の高い製品を実現することができます。

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 の構成要素は次のとおりです。

- 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、次のモデルで構成されます。
  - 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します。クラスと関連のサブセットで構成されます。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します。
  - 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します。管理アプリケーションの開発基盤を提供します。
- 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 サービスについて

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

- 26 ページの「CIM オブジェクトマネージャ」
- 27 ページの「MOF コンパイラ」
- 27 ページの「Solaris スキーマ」
- 27 ページの「Solaris WBEM SDK」

Solaris WBEM サービスは、Solaris オペレーティング環境で、管理データのセキュリティ保護されたアクセスと操作などの、WBEM サービスを提供するソフトウェアです。製品には Solaris プロバイダが組み込まれているため、管理アプリケーションから Solaris オペレーティング環境の管理リソース (デバイスおよびソフトウェアなど) の情報にアクセスできます。

CIMOM (CIM オブジェクトマネージャ) は、Remote Method Invocation (RMI) プロトコルまたは XML over HTTP プロトコルを使用する管理アプリケーションからの接続を受け入れ、接続されたクライアントに次のようなサービスを提供します。

- 管理サービス – CIMOM の形式をとります。CIMOM は、CIM データのセマンティクスと構文をチェックし、複数のアプリケーション、CIM オブジェクトマネージャリポジトリ (CIM Object Manager Repository)、管理対象のリソースにデータを分配します。
- セキュリティサービス – これらのサービスは、Solaris 管理コンソールのユーザーツールによって WBEM に指定します。これらのサービスについては、『*Solaris* のシステム管理 *(*セキュリティサービス*)*』を参照してください。
- Sun™ WBEM ユーザーマネージャ (Sun WBEM User Manager) – このツールを使用して、WBEM サーバー上の特定の名前空間 (ネームスペース) のアクセス制御リスト (ACL) を確立します。Sun WBEM ユーザーマネージャでは、承認されたユーザーの追加と削除、承認されたユーザーに対するアクセス特権の設定、および WBEM 対応システム上の CIM オブジェクトに対するユーザー認証とアクセスの管理を実行できます。ACL ベースのセキュリティは、Solaris WBEM サービスによって提供される固有の機能です。
- ロギングサービス – このサービスを構成するクラスを使えば、開発者は、イベントデータを動的に記録したり取得したりできるアプリケーションを作成できます。管理者はこのデータを使ってイベントの原因を追跡したり、判定したりすることができます。ロギングサービスの詳細については、第 9 章を参照してください。
- XML サービス – XML データを CIM クラスに変換します。XML/HTTP ベースの WBEM クライアントが CIM オブジェクトマネージャと通信できるようにします。

WBEM 対応システムに接続されると、WBEM クライアントは、次のような WBEM 操作を要求できます。すなわち、CIM クラスおよびインスタンスの作成、表示、削除や、指定された値をもつプロパティの検索、指定されたクラス階層にあるインスタンスやクラスの列挙などです。

ソフトウェアのコンポーネント

Solaris WBEM サービスソフトウェアは、アプリケーション、管理、およびプロバイダという 3 つの層で 機能するソフトウェアコンポーネントで構成されます。これらのコンポーネントはオペレーティングシステムやハードウェアとデータを送受信します。次の図に、各ソフトウェアコンポーネントと、それぞれの送受信方法を示します。

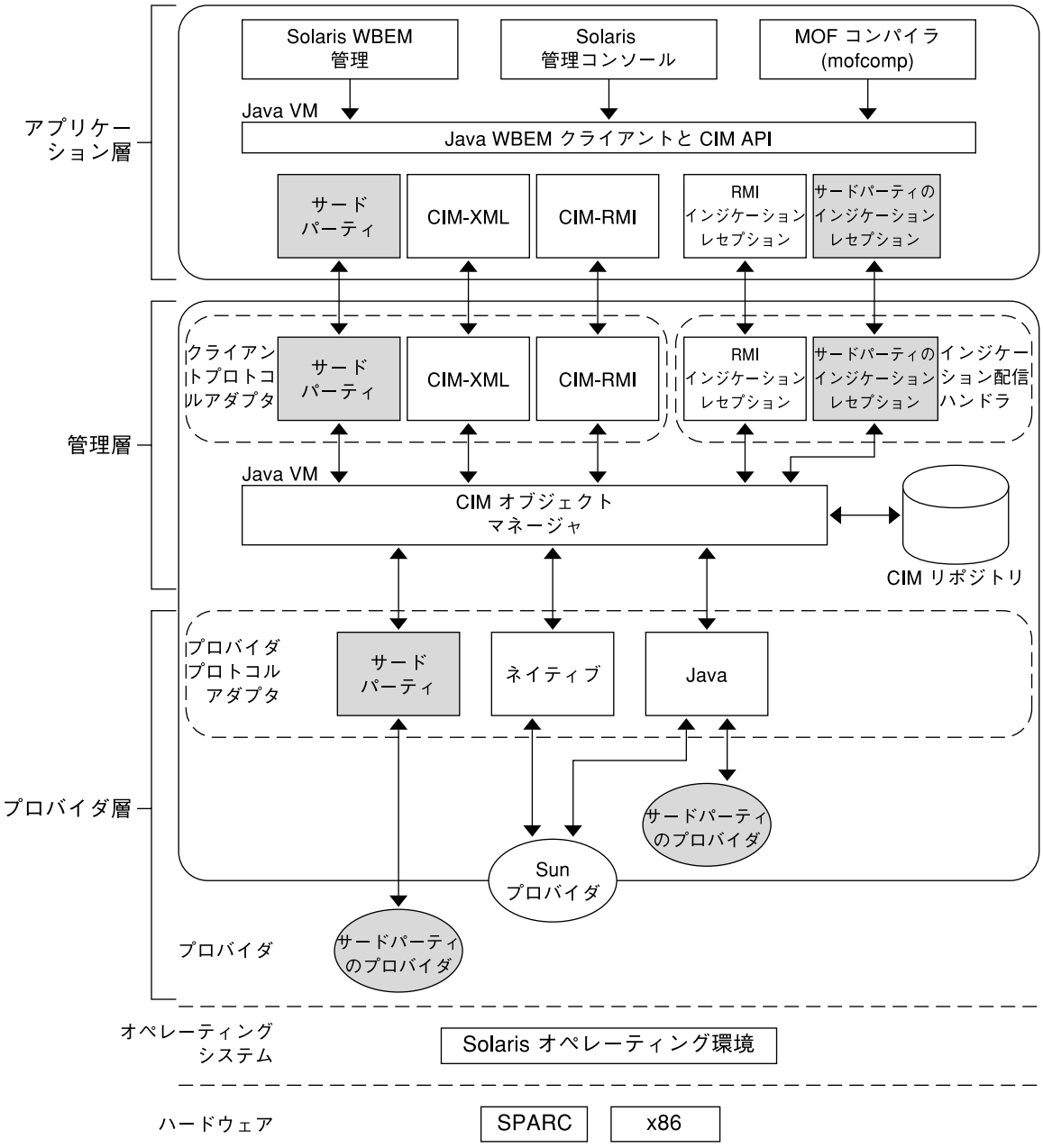

図 **1–1** Solaris WBEM サービスのアーキテクチャ

- アプリケーション層 – WBEM クライアントが管理リソースからのデータを処理したり、表示したりします。Solaris WBEM サービスには、次のアプリケーションがあります。
  - Sun WBEM ユーザーマネージャおよび Solaris 管理コンソールユーザーツール – これらのアプリケーションでは、システム管理者が、承認されたユーザーの追加や削除を実行したり、それらのユーザーの管理リソースへのアクセス特権を設定したりできます。
  - Solaris 管理コンソールログビューア – ログファイルを表示するアプリケーションです。ログに残ったコマンドを実行したユーザーの名前や、ログに残ったイベントが発生したクライアントコンピュータなど、ログレコードの詳細を表示できます。
  - Managed Object Format (MOF) コンパイラ – このプログラムは、MOF 文を含むファイルを解析し、そのファイルで定義されているクラスやインスタンスを Java クラスに変換し、その Java クラスを CIM オブジェクトマネージャリポジトリ (管理データを一元的に格納する場所) に追加します。

    MOF は、CIM のクラスやインスタンスを定義する言語です。MOF ファイルは、MOF 言語を使って CIM オブジェクトを記述する ASCII テキストファイルです。CIM オブジェクトは、プリンタ、ディスクドライブ、CPU などの管理リソースを表したモデルです。MOF ファイルは `/usr/sadm/mof` にあります。

    管理リソースの情報は MOF ファイルに格納されることがあります。MOF は Java に変換できるため、Java 仮想マシン (JVM) を持つシステムで動作するアプリケーションならこの情報の解釈や交換を行うことができます。さらに、インストールの後で、`mofcomp` コマンドを使って MOF ファイルをいつでもコンパイルできます。MOF については、DMTF の Web ページ `http://www.dmtf.org` を参照してください。
- 管理層 – この層のコンポーネントは、接続された WBEM クライアントに次のサービスを提供します。
  - Common Information Model (CIM) オブジェクトマネージャ – WBEM システム上の CIM オブジェクトを管理するソフトウェアです。CIM オブジェクトは内部的には Java クラスとして格納されます。CIM オブジェクトマネージャは WBEM クライアント、CIM オブジェクトマネージャリポジトリ、管理リソースとの間で情報を送受信します。
  - CIM オブジェクトマネージャリポジトリ – CIM のクラスやインスタンスの定義を一元的に格納する場所です。
  - クライアントおよび CIM アプリケーションプログラミングインタフェース (API) – WBEM クライアントアプリケーションは、これらの Java インタフェースを使って、管理リソースのクラスやインスタンスの作成および表示などの操作を CIM オブジェクトマネージャに要求します。
  - プロバイダインタフェース – プロバイダは、これらのインタフェースを使って管理リソースの情報を CIM オブジェクトマネージャに転送します。CIM オブジェクトマネージャは、プロバイダインタフェースを使って、ローカルにインストールされたプロバイダに情報を転送します。

- プロバイダ層 – プロバイダは、CIM オブジェクトマネージャと 1 つまたは複数の管理リソースとの間の仲介を行います。WBEM クライアントから CIM オブジェクトマネージャリポジトリに存在しないデータを要求されると、CIMOM は、要求を適切なプロバイダに転送します。
  - Solaris プロバイダ – Solaris オペレーティング環境内の管理リソースのインスタンスを、CIM オブジェクトマネージャに提供します。プロバイダは、管理デバイスに関する情報の取得および設定を行います。ネイティブプロバイダとは、管理対象デバイスで動作するように作成されたマシン固有のプログラムです。たとえば、Solaris オペレーティング環境を実行しているシステム上のデータにアクセスするプロバイダには、そのシステムに照会する C 関数が含まれているはずです。Java Native Interface は、JDK™ ソフトウェアの一部です。Java Native Interface を使ってプログラムを作成すれば、その Java プログラムコードはどのプラットフォームに移植しても確実に動作します。Java Native Interface を使うと、Java 仮想マシン内で動作する Java コードを、C、C++、アセンブラなど、他の言語で作成されたアプリケーションやライブラリとともに動作するようにできます。
  - Solaris スキーマ – Solaris オペレーティング環境内の管理対象オブジェクトを記述するクラスの集合です。CIM スキーマや Solaris スキーマのクラスは CIM オブジェクトマネージャリポジトリに格納されます。CIM スキーマは、どの管理環境にもある管理オブジェクトを表すためのクラス定義の集合です。

    Solaris スキーマは CIM スキーマを拡張したもので、一般的な Solaris オペレーティング環境の管理オブジェクトを表すクラス定義の集合です。ユーザーは、MOF コンパイラ (`mofcomp`) 使用して CIM スキーマ、Solaris スキーマ、あるいはその他のクラスを CIM オブジェクトマネージャリポジトリに追加することもできます。
- オペレーティングシステム層 – Solaris プロバイダを使えば、管理アプリケーションから Solaris オペレーティング環境にある管理リソース (デバイスおよびソフトウェア) の情報にアクセスできます。
- ハードウェア層 – 管理クライアントは、サポートされる任意の Solaris プラットフォームの管理データにアクセスできます。

CIM オブジェクトマネージャ

CIM オブジェクトマネージャは、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、CIMOM は、そのオブジェクトの適切なプロバイダまたは CIM オブジェクトマネージャリポジトリ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から CIM オブジェクトマネージャリポジトリに存在しない管理リソースを要求されると、CIMOM は、要求をその管理リソースのプロバイダに転送します。プロバイダは情報を動的に取得します。

WBEM クライアントアプリケーションは、CIM オブジェクトマネージャとの接続を確立します。この接続は、CIM クラスの作成、CIM インスタンスの更新といった WBEM 操作に使用されます。WBEM クライアントアプリケーションが CIM オブ

ジェクトマネージャに接続すると、WBEM クライアントは CIM オブジェクトマネージャへの参照を取得します。WBEM クライアントは、この参照を利用して、サービスを要求したり各種操作を実行したりできます。

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 オブジェクトマネージャリポジトリに追加されます。MOF コンパイラを使用して MOF ファイルから自動的に JavaBeans™ コンポーネントを生成する手順については、第 7 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 プラットフォーム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、および構文の詳細については、 `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す。特に、Solaris オペレーティング環境で実行されている管理オブジェクトを記述するためのものです。

Solaris WBEM サービス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、CIMOM の起動時に自動的にコンパイルされます。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、管理アプリケーションの作成に必要なコンポーネントを含む API のセットです。これらのアプリケーションは、XML および HTTP 通信標準に従って WBEM 対応の管理デバイスと通信します。

Solaris WBEM アプリケーションは、WBEM API を介して CIM オブジェクトマネージャ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
クライアント	`javax.wbem.client`	アプリケーションは、CIM オブジェクトマネージャとの通信に `CIMClient` クラスを使用する。CIM オブジェクトマネージャ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	`javax.wbem.provider`	CIM オブジェクトマネージャは、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	`javax.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 WBEM アプリケーションを開発できます。CIM Workshop の機能は次のとおりです。

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

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。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ヘルプテキストが表示されます。

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、RMI (Remote Method Invocation) 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 CIMOM に接続します。CIM オブジェクトマネージャ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 の起動方法

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 の終了方法

- **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ます。
- 右側の区画 – 「プロパティ (Properties)」、「メソッド (Methods)」、および「イベント (Events)」タブが表示されます。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ます。
- 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ます。

第 2 章

CIM オブジェクトマネージャの使用

Common Information Model (CIM) オブジェクトマネージャは、WBEM クライアントアプリケーションと管理リソースとの間で CIM データを送受信するソフトウェアです。

次の内容について説明します。

CIM オブジェクトマネージャについて

CIM オブジェクトマネージャ (CIMOM) は、WBEM 対応システムの CIM オブジェクトを管理します。CIM オブジェクトは、プリンタ、ディスクドライブ、CPU などの管理リソースを表したモデルです。CIM オブジェクトは、内部的には Java プログラミング言語のクラスとして格納されます。

WBEM クライアントアプリケーションがオブジェクト情報にアクセスすると、CIMOM は、そのオブジェクトに適したプロバイダか、CIM オブジェクトマネージャリポジトリに接続します。プロバイダとは、管理オブジェクトと通信してデータにアクセスするクラスです。WBEM クライアントアプリケーションから要求されたデータが、CIM オブジェクトマネージャリポジトリに含まれていない管理リソースのデータである場合、 CIM オブジェクトマネージャはその要求を該当する管理リソースのプロバイダに転送します。プロバイダは情報を動的に取得します。

CIM オブジェクトマネージャは、起動時に次の機能を実行します。

- ポート 5987 上で RMI 接続を、ポート 5988 上で XML over HTTP 接続を待機する
- CIM オブジェクトマネージャリポジトリとの接続を設定する

- 要求が入ってくるのを待つ

CIM オブジェクトマネージャは、次のことを行います。

- セキュリティ検査を実行して、ユーザーログインを認証し、名前空間へのアクセスを承認する
- CIM データ操作の構文と意味を検査して、最新の CIM 仕様に準拠していることを確認する
- 適切なプロバイダか CIM オブジェクトマネージャリポジトリに要求を転送する
- プロバイダや CIM オブジェクトマネージャリポジトリから受け取ったデータを WBEM クライアントアプリケーションに転送する

クライアント側で WBEM 操作を実行する必要があるとき、WBEM クライアントアプリケーションは CIMOM に接続します。たとえば、CIM クラスの作成や、CIM インスタンスの更新処理が必要なときです。WBEM クライアントアプリケーションが CIMOM に接続すると、クライアントアプリケーションは CIMOM への参照を取得します。クライアントアプリケーションは、この参照を利用して、サービスや操作を要求します。

`init.wbem` コマンド

`init.wbem` コマンドは、インストール時とシステムのリブート時に自動的に実行されます。`init.wbem` コマンドは、CIM オブジェクトマネージャと Solaris 管理コンソールサーバーが結合された単一プロセスを実行します。また、CIM オブジェクトマネージャおよび Solaris 管理コンソールサーバーを停止したり、サーバーからステータスを入手したりするためにも、`init.wbem` コマンドを使用できます。このコマンドに関する付加的な情報は、`init.wbem(1M)` のマニュアルページにあります。

一般には CIM オブジェクトマネージャを停止する必要はありません。ただし、既存のプロバイダを変更する場合には、変更後のプロバイダを使用する前に、CIM オブジェクトマネージャを停止してから再起動する必要があります。

`init.wbem` コマンドには、次の 3 つのオプションを指定できます。

- `start` – ローカルホスト上の CIM オブジェクトマネージャまたは Solaris 管理コンソールサーバーを起動する
- `stop` – ローカルホスト上の CIM オブジェクトマネージャおよび Solaris 管理コンソールサーバーを停止する
- `status` – ローカルホスト上の CIM オブジェクトマネージャおよび Solaris 管理コンソールサーバーのステータスを取得する

Solaris 管理コンソールサーバー

Solaris 管理コンソールソフトウェアは、ユーザー管理、ディスク管理、ログビューアなどの Solaris 管理アプリケーションを提供します。Solaris 管理コンソールサーバーは、コンソールを取り込むことが可能なツールを提供します。Solaris 管理コンソールサーバーは、コンソールとそのツール群を対象とした一般的なサービスも実行できます。たとえば、認証、承認、ロギング、メッセージング、持続性などです。

Solaris 管理コンソールについては、ほかの章で説明します。詳細は、『*Solaris* のシステム管理 (基本編)』を参照してください。

システムのブート

`init.wbem` コマンドは、 `/etc/init.d` ディレクトリにあります。init state 2 に入った時点 (通常はブート時) で、`/etc/rc2.d/S90wbem` ファイルが `start` オプションを指定して実行されます。`/etc/rc0.d/K36wbem`、`/etc/rc1.d/K36wbem`、および `/etc/rcS.d/K36wbem` の 3 ファイルは、init state 0、1、および S に入った時点で、`stop` オプションで実行されます。

CIM オブジェクトマネージャの停止と再起動

プロバイダを変更する場合は、変更後のプロバイダを使用する前に、CIM オブジェクトマネージャを停止し、再起動しなければなりません。

▼ CIM オブジェクトマネージャを停止する方法

1. スーパーユーザーになります。
2. **CIM** オブジェクトマネージャを停止します。

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

▼ CIM オブジェクトマネージャを再起動する方法

1. スーパーユーザーになります。
2. **CIM** オブジェクトマネージャを再起動します。

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

CIM オブジェクトマネージャリポジトリのアップグレード

以前のバージョンの Solaris を Solaris 9 にアップグレードした場合は、MOF (Managed Object Format) のカスタムデータを Solaris 9 の新しいリポジトリ形式に更新する必要があります。アップグレードを行うと、アップグレード前に変更を加えた CIM および Solaris MOF データがすべて破壊されます。そのため、アップグレード後に MOF ファイルを再コンパイルするか、WBEM データをマージする必要があります。

注意 – 変更を加えたデータの再コンパイルまたはマージ処理を正しく実行しないと、データは失われます。

Solaris 9 オペレーティング環境にアップグレードしたあと WBEM データを再コンパイルするかマージするかについては、次の表を参照してください。

表 **2–1** WBEM データを再コンパイルするかマージするかの判別

アップグレード前の環境	所有する **MOF (Managed Object Format)** ファイルを再コンパイルするか
Solaris 8 (Solaris WBEM サービス 2.0) Solaris 8 6/00 (WBEM サービス 2.0) Solaris 8 10/00 (WBEM サービス 2.2)	する
Solaris 8 1/01 (WBEM サービス 2.3) Solaris 8 4/01 (WBEM サービス 2.4) Solaris 8 7/01 (WBEM サービス 2.4) Solaris 8 10/01 (WBEM サービス 2.4) Solaris 9 5/02 (WBEM サービス 2.5) Solaris 9 9/02 (WBEM サービス 2.5) Solaris 9 12/02 (WBEM サービス 2.5)	しない。ただし、アップグレード済みリポジトリにデータをマージする必要がある

▼ MOF ファイルの再コンパイル方法

1. システムを **Solaris 9** オペレーティング環境にアップグレードします。
2. スーパーユーザーになります。

3. 所有する **MOF** ファイルが置かれているディレクトリに移動します。

4. **mofcomp** コマンドを使って、所有する個々の **MOF** ファイルをコンパイルします。

   # **/usr/sadm/bin/mofcomp root** *root-passwd MOF-filename*

   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。

5. **CIM** オブジェクトマネージャを停止します。

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

6. **CIM** オブジェクトマネージャを起動します。

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

   CIMOM は、/var/sadm/wbem/logr/ ディレクトリに、変換済みデータを含むリポジトリファイルを追加します。このディレクトリは、Solaris 9 オペレーティング環境へのアップグレード時に作成されたものです。

▼ WBEM データをマージする方法

1. システムを **Solaris 9** オペレーティング環境にアップグレードします。

2. スーパーユーザーになります。

3. **CIM** オブジェクトマネージャを停止します。

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

   注意 – wbemconfig convert コマンドを実行する前に CIM オブジェクトマネージャを停止しなかった場合、データが損傷を受ける場合があります。

4. 以前のバージョンの **Reliable Log** にある元のデータを、**Solaris 9** の **Reliable Log** 内のデータとマージします。

   # **/usr/sadm/lib/wbem/wbemconfig convert**

注 – `wbemconfig convert` コマンドでは、独自にカスタマイズした MOF データは正しく変換できますが、変更を加えた CIM または Solaris MOF データは変換できません。変更を加えた CIM または Solaris MOF データは壊れます。変更を加えた CIM または Solaris MOF データを新しいリポジトリに再コンパイルするには、`mofcomp` コマンドを使用して、クラス定義を含む MOF ファイルをコンパイルします。

例外メッセージ

CIM オブジェクトマネージャは、MOF の構文や意味が正しくない場合、例外メッセージを生成します。例外メッセージの詳細については、第 9 章を参照してください。

第 3 章

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。

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、プログラムファイルが `/usr/demo/wbem` 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 サービス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列挙する。ローカルまたはリモートシステム上の CIM オブジェクトマネージャに接続する
- クライアントプログラム – Java API を使用して CIM オブジェクトマネージャ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 オブジェクトマネージャ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、次のプログラム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

- 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には、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** ブラウザでサンプルアプレットを実行する場合は、**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 **3–1** 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 username password classname* [rmi|http]	1 回のバッチ処理で 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 オブジェクトマネージャに接続し、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 **3–1** 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で、**CLASSPATH** 環境変数を設定します。
  - C シェルを使用して、次のように入力します。

    ```
    % setenv 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。

    ```
    % export 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

ほとんどのサンプルクライアントプログラムは、CIM オブジェクトマネージャと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受け付けます。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 **3–2** サンプルプロバイダプログラム

ファイル名	目的
NativeProvider.java	CIM オブジェクトマネージャからの要求に応答し、この要求を 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。instanceProvider API および methodProvider API を実装する。また、インスタンスを列挙するメソッドや、Native_Example クラスのインスタンスを取得するメソッドを宣言する。このほか、「Hello World」という文字列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 オブジェクトマネージャ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答えるプロバイダと見なす。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、インスタンスを列挙したり、Native_Example クラスのインスタンスを取得したりする場合に、このプロバイダを呼び出す。Native_Example プロバイダは、API を使用してオブジェクトを列挙したり、オブジェクトインスタンスを作成したりする。Native_Example クラスはネイティブメソッドを宣言する。このメソッドは、native.c ファイル内の C 関数を呼び出し、システム固有の値を取得する。システム固有の値には、ホスト名、シリアル番号、リリース、マシン、アーキテクチャ、製造元などがある
native.c	Native_Example Java プロバイダのメソッドをネイティブ 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 プログラミング言語で作成されたネイティブメソッドの名前と、これらのメソッドを実行するネイティブ C 関数の対応関係を定義する
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、これらの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** オブジェクトマネージャを停止します。

   ```
   # /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 CIMOM の停止と再起動を行なってください。

   ---

6. **CIM** オブジェクトマネージャを起動します。

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

7. スーパーユーザー状態から抜けます。

8. **CIMOM** 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。

第 4 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`) を使ってクライアントプログラムを作成する方法について説明します。この章の内容は、次のとおりです。

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

クライアント API の概要

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 を使ってオブジェクトを構築したあと、そのオブジェクトのインスタンスを作成します。こうしたオブジェクトには、クラス、インスタンス、名前空間などがあります。アプリケーションは、クライアント API を使ってオブジェクトを CIM オブジェクトマネージャに渡し、WBEM 操作を要求します。たとえば、CIM クラスの作成、インスタンスの作成、名前空間の作成といった操作があります。

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` を使って CIMOM に接続します。クライアントアプリケーションは、WBEM 操作を実行する必要があるとき、毎回 CIMOM に接続します。WBEM 操作には、CIM クラスの作成、CIM インスタンスの更新などがあります。詳細は、44 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。次に、ほとんどのプログラムが実行する一般的な処理を示します。

   - 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
   - エラー処理

   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は、クライアントプログラムからも実行できます。詳細は、47 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 オブジェクトマネージャ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、44 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

クライアントが WBEM 操作を実行するためには、まずアプリケーションから CIMOM への接続を確立する必要があります。たとえば、CIM クラス、CIM インスタンス、CIM 修飾子型などを追加、変更、または削除する WBEM 作業があります。クライアントアプリケーションと CIM オブジェクトマネージャは、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 オブジェクトマネージャ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 CIMOM への接続を確立する場合、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、および修飾子型を含みます。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 ソフトウェア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 クラスが含まれます。

- root\security – セキュリティ関連のクラスが含まれます。
- root\snmp – SNMP アダプタクラスが含まれます。
- root\system – CIM オブジェクトマネージャを管理するクラスが含まれます。

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 オブジェクトマネージャに接続します。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。デフォルト値は、ローカルホスト上の root\cimv2 です。ローカルホストとは、クライアントアプリケーションを実行しているホストです。クライアントから CIMOM への接続が確立されたあと、後続の CIMClient 操作はすべて指定された名前空間内で行われます。
- *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。CIMOM は、ユーザー名からそのアクセス特権を確認し、CIM オブジェクトに許可されたアクセスの種類を決定します。
- *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)。CIMOM へのメッセージ送信に使用するプロトコル。有効な値は、 *RMI* (デフォルト値) または *HTTP* です。

例 **4–1** 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 オブジェクトマネージャに接続します。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。

```
{
   ...

   /* 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) を示す 2 つの NULL
   文字列で初期化された名前空間オブジェクトを作成する */

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

   UserPrincipal up = new UserPrincipal("root");
   PasswordCredential pc = new PasswordCredential("root-password");
   /* root ユーザーとして root パスワードを使って名前空間に
   接続する */

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

例 **4–1** ルートアカウントへの接続　(続き)

例 **4–2** 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、CIMOM への接続を確立するため、CIMClient クラスを使って、資格情報 (ホスト名、名前空間、ユーザー名、およびパスワード) を渡します。

```
{
    ...
    /* ホスト happy 上の名前空間
    A で初期化された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 **4–3** 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 **4–4** 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

この節では、javax.wbem.client API を使った操作の要求方法と、一般的なプログラミング作業の実行方法について説明します。

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。これらの修飾子は、インスタンス、またはそのインスタンスの特定のプロパティに対して定義できます。クラス宣言内に修飾子を記述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 **4–5** 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 パッケージを作成します。

```
...
{
/* ローカルホスト上の名前空間 root\cimv2
で CIM オブジェクトマネージャに接続する。
root\cimv2 にオブジェクトへの書き込み権を
持つアカウントのユーザー名とパスワードを指定する */

 UserPrincipal up = new UserPrincipal("root");
 PasswordCredential pc = new PasswordCredential("root-password");
 /* root ユーザーとして、root パスワードを使って名前空間に接続
 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で生成された Solaris_Package
 クラスの新しいインスタンスを作成する。
 クラスのプロバイダがデフォルト値を指定しない場合、
 プロパティの値は NULL になるため、明示的に設定する
 必要がある */
```

例 **4–5** インスタンスの作成　(続き)

```
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 **4–6** インスタンスの削除

この例は、次のことを実行します。

- クライアントアプリケーションを CIMOM に接続する
- `CIMObjectPath` を使って、削除されるオブジェクトの CIM オブジェクトパスを含むオブジェクトを構築する
- `enumerateInstance` を呼び出して、指定のインスタンスとそのサブクラスのすべてのインスタンスを取得する
- `deleteInstance` を呼び出して各インスタンスを削除する

```
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;

/**
 * 指定のクラスのインスタンスをすべて返す。
 * この例は、ホスト名 (args[0])、ユーザー名 (args[1])、パスワード
 * (args[2])、名前空間 (args[3])、およびクラス名 (args[4]) という
 * 5 つの引数を取る。指定のクラス名のインスタンスをすべて削除
 * する。指定のユーザー名は、指定の名前空間に対する書き込み権
 * 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は使用法を表示して終了
        if (args.length != 5) {
```

例 **4–6** インスタンスの削除　*(続き)*

```
            System.out.println("Usage: DeleteInstances host username " +
                               "password namespace classname ");
            System.exit(1);
        }
        try {
            // args[0] にはホスト名、args[3] には名前空間が含まれる。
            // 指定のホスト上の指定の名前空間を指す
            // CIMNameSpace (cns) を作成する
            CIMNameSpace cns = new CIMNameSpace(args[0], args[3]);

            // args[1] と args[2] にはユーザー名とパスワードが含まれる。
            // このユーザー名で UserPrincipal (up) を作成し、
            // このパスワードで PasswordCredential を作成する
            UserPrincipal up = new UserPrincipal(args[1]);
            PasswordCredential pc = new PasswordCredential(args[2]);

            // CIM オブジェクトマネージャに接続し、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);

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

            // クラスおよびその全サブクラスのすべてのインスタンス
            // オブジェクトパスの列挙を取得する。インスタンス
            // オブジェクトパスは、CIM オブジェクトマネージャが
            // インスタンスを検索するときに使用する参照
            Enumeration e = cc.enumerateInstanceNames(cop);

            // 列挙内のインスタンスオブジェクトパスを反復する。
            // オブジェクトを構築し、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、このインスタンスを出力
            // して、削除する
            while (e.hasMoreElements()) {
                CIMObjectPath  op = (CIMObjectPath)e.nextElement();
                System.out.println(op);
                cc.deleteInstance(op);
            } // 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

        // セッションの終了
        if (cc != null) {
            cc.close();
        }
    }
}
```

例 **4–6** インスタンスの削除　(続き)

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は、一般に、`getInstance` メソッドを使って CIMOM から CIM インスタンスを取得します。クラスのインスタンスが作成されると、クラスはクラス階層内の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の非継承プロパティだけを返します。非継承プロパティは、インスタンス内で定義されます。
- *localOnly* が false の場合、クラス内のすべてのプロパティが返されます。たとえば、インスタンス内で定義されたプロパティや、クラス階層内のすべての親クラスから継承されたすべてのプロパティが返されます。

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 **4–7** インスタンスの取得と設定

この例は次のことを行います。

- 列挙内のオブジェクトパスのインスタンスを取得する
- 各インスタンス内の b のプロパティ値を 10 に更新する
- 更新されたインスタンスを CIMOM に渡す

```
...
{
    // 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 オブジェクトマネージャに渡す */

    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、1 組の関数と見なすことができます。一方の関数はプロパティ値を「取得」し、もう一方の関数はプロパティ値を「設定」します。

例 **4–8** プロパティの取得

次の例は、enumerateInstanceNames を使って Solaris プロセッサのすべてのインスタンスの名前を返します。この例は、getProperty を使っ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 を使ってこの値を出力します。

```
...
{
/* 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 オブジェクトマネージャは、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 */

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

例 **4–9** 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 を使って Solaris_User Template のすべてのインスタンスの名前を取得します。また、setProperty を使って各インスタンスの初期シェルの値を設定します。

```
...
{
    /* 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 オブジェクトマネージャは、
    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 のインスタンスごとに /usr/bin/sh に
```

例 **4–9** プロパティの設定　(続き)

```
    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

列挙とは一度にオブジェクトを 1 つずつ取り出すことができるオブジェクトの集合です。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 **4–1** オブジェクトの列挙

ソート方法	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され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され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

表 **4–1** オブジェクトの列挙　(続き)

ソート方法	引数なし	*deep*	*localOnly*
`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
`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 **4–10** クラスの列挙

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、列挙される
        CIM クラスの名前 (myclass) で初期化する */

    CIMObjectPath cop = new CIMObjectPath(myclass);

    /* この列挙には、列挙されたクラス内のクラスとサブクラス
        が含まれる (deep=true)。この列挙は、各クラスおよびサブクラスの継承
        されないメソッドおよびプロパティだけを返す (localOnly
         が true) */

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

例 **4–11** 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;
```

例 **4–11** クラスおよびインスタンスの列挙　*(続き)*

```
/**
 * この例は、クラスとインスタンスを列挙する。コマンド行から
 * 渡されるクラス上で deep 列挙と shallow 列挙を行う
 */
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);

           // コマンド行からクラス名を取得
           cop = new CIMObjectPath(args[3]);
           // クラスの deep 列挙を実行
           Enumeration e = cc.enumerateClasses(cop, true, true, true,
                                               true);
           // クラスのサブクラスをすべて出力
           while (e.hasMoreElements()) {
               System.out.println(e.nextElement());
           }
           System.out.println("+++++");
           // クラスの shallow 列挙を実行
           e = cc.enumerateClasses(cop, false, true, true, true);
           // 第 1 レベルのサブクラスを出力
           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
           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

           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
           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
           while (e.hasMoreElements()) {
               System.out.println(e.nextElement());
           }
           System.out.println("+++++");
```

例 **4–11** クラスおよびインスタンスの列挙　*(続き)*

```
            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);
        }
        // セッションの終了
        if (cc != null) {
            cc.close();
        }
    }
}
```

例 **4–12** 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 **4–13** 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;
```

例 **4–13** 名前空間の列挙　(続き)

```
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
        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 オブジェクトマネージャに接続して
            // 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

       // セッションの終了
       if (cc != null) {
```

例 **4–13** 名前空間の列挙　　(続き)

```
            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。実際のオブジェクト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。

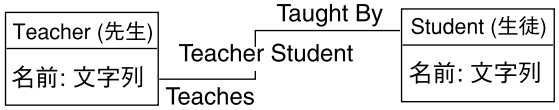

図 **4–1** TeacherStudent 関連 1

前出の図には、`Teacher` と `Student` という 2 つのクラスが示されています。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 **4–2** 関連メソッド

ソート方法	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

表 **4–2** 関連メソッド　(続き)

ソート方法	説明
`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、単一の必須引数 `CIMObjectPath` を取ります。`CIMObjectPath` は、検索する関連、関連付けられたクラス、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です。CIMOM は、関連、関連付けられたクラス、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。

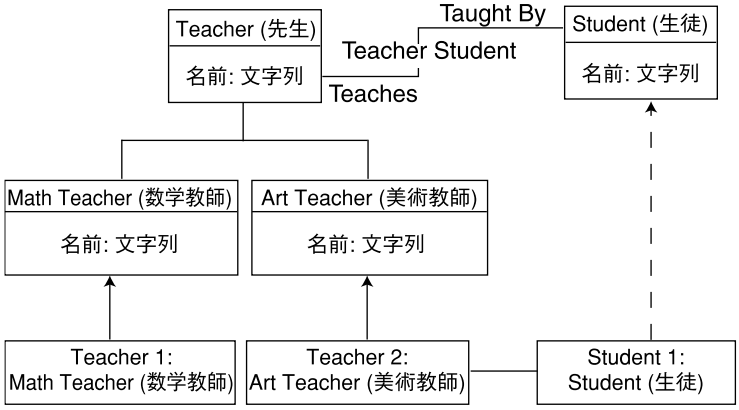

図 **4–2** TeacherStudent 関連 2

前出の図の `associators` および `associatorNames` メソッドは、`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 **4–3** 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。MathTeacher と ArtTeacher 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
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。これらの引数は、インスタンスまたはクラスの名前だけを返し、その内容を返さないメソッドには不必要です。

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、サンプルモデルパスを示します。

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

このモデルパスは、次の値を指定します。

- \\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 **4–14** インスタンスの引き渡し

この例は、op クラスとそのサブクラスのインスタンスを列挙します。while ループを使って個々のインスタンスを CIMObjectPath (op) にキャストし、各インスタンスを associators メソッドの第 1 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。最初の 4 つのパラメータは、返されるクラスおよびインスタンスをフィルタします。

- assocClass
- resultClass
- resultRole
- role

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 **4–4** 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
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 **4–15** メソッドの呼び出し

この例では、CIM_Service クラスのインスタンスを取得します。これらのインスタンスは、デバイスやソフトウェアの機能を管理するサービスを表します。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 オブジェクトマネージャに渡す。
    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 オブジェクトマネージャは 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);
              }
}
```

例 **4–15** メソッドの呼び出し　*(続き)*

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。クラスが作成されると、そのクラスは、クラス階層内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 **4–16** 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 オブジェクトマネージャ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 オブジェクトマネージャ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);
        }
```

例 **4–16** クラス定義の取得　*(続き)*

```
        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。CIMOM は、Java の例外処理により、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 オペレーティング環境は、標準 CIM MOF (Managed Object Format) ファイルをコンパイルして、デフォルトの名前空間を生成します。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 **4–17** 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 オブジェクトマネージャ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 オブジェクトマネージャに接続して名前空間オブジェクトを渡します。CIM オブジェクトマネージャは、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 */
```

例 **4–17** 名前空間の作成　(続き)

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

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

    /* CIM オブジェクトマネージャに接続し、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 オブジェクトマネージャに、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、
    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 構文に慣れている場合は、テキストエディタで MOF ファイルを作成します。次に、MOF コンパイラでファイルをコンパイルし、Java クラスを生成します。詳細は、第 7 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 (抽象) 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個別に定義します。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 (抽象) 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。このメソッドは、クラスを削除し、`CIMException` をスローします。

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 **4–18** 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、次の 4 つの必須文字列引数を取ります。

- *hostname*
- *classname*
- *username*
- *password*

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アクセス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;

/**
 * コマンド行で指定されたクラスを削除する。
```

例 **4–18** クラスの削除　(続き)

```
 * 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]);
            // 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 – 指定の名前空間内の CIM オブジェクトに対するユーザーの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

名前空間内の CIM オブジェクトに対するアクセス制御は、ユーザーごとに設定できます。Solaris_UserACL クラスのインスタンスを作成し、そのアクセス権を変更します。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。

Solaris_NameSpaceACL クラスを使ってこの 2 つのクラスの使用方法を組み合わせ、まずは、名前空間内のオブジェクトに対するすべてのユーザーのアクセスを制限します。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* プロパティには、次の表に示すアクセス特権の値を設定できます。

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

次の表に、Solaris_UserAcl クラスが定義するキープロパティを示します。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。

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

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

// root ユーザーとして root\security 名前空間に接続する
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。

```
...
/* 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。

```
...
// 更新されたインスタンスを CIM オブジェクトマネージャに渡す
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あり、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。

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

// root ユーザーとして root\security 名前空間に接続する
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。

```
...
/* root\molly 名前空間のアクセス権 (capability)
を読み取り/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。

```
// 更新されたインスタンスを CIM オブジェクトマネージャに渡す
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、パラメータのいずれか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。この修飾子で説明できるのは、プロパティとオブジェクトへの参照だけです。
`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、修飾子を派生クラスおよびインスタンスに継承できるかどうかを指定する規則について説明します。派生クラスや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規則で決定することもできます。

例 **4–19** 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。

```
{

 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;
```

例 **4–19** CIM 修飾子の設定　(続き)

```
 }
}
...
```

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 オブジェクトマネージャ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。バッチ内の個々の操作間に依存関係はありません。前の操作の成功または失敗が、後続の操作に影響を及ぼすことはありません。

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、CIMClient メソッドとよく似ています。ただし、BatchCIMClient メソッドによって返される型は、CIMClient クラス内の同等のメソッドが返す型とは異なります。これは、バッチ操作の完了後に、リストとして値が返されるからです。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。

クライアントは、CIMClient の performBatchOperations メソッドを呼び出すことにより、バッチ操作を実行します。バッチ操作の結果は、BatchResult オブジェクト内に返されます。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 **4–20** 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。 enumerateInstanceNames、 getClass、enumerateInstances という 3 つの操作が、単一のバッチ処理として実行されます。

```
import java.util.Enumeration;
import java.util.ArrayList;
```

例 **4–20** バッチ処理の例　(続き)

```
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);

             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());
```

例 **4–20** バッチ処理の例　*(続き)*

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

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

            // セッションの終了
            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、トリガーは以下のアクション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対する操作。クラスの作成、削除、変更、クラスへのアクセスなどを実行できます。インスタンスの変更、インスタンスへのアクセスも可能です。
- 管理された環境内で発生するイベント

たとえば、サービスの終了によってトリガーが作動したとき、このイベントは、サービスが終了したことを通知するインジケーションになります。

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。次の表に、クラスの構造を示します。

表 **4–5** 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、データ内の特定の変更によって発生する組み込み型の CIM イベントです。ライフサイクルイベントをトリガーする変更には、次のものがあります。

- クラスの作成、変更、または削除
- 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

「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、CIMOM 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。CIMOM は、予約要求を分析します。CIMOM は、EventProvider または CIMIndicationProvider インタフェースを使ってプロバイダと通信を行い、適切なインジケーションを生成するように要求します。プロバイダがインジケーションを生成すると、CIMOM 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。CIM_InstIndication のサブクラスなど、インスタンスのライフサイクルインジケーションに属する予約の場合、CIMOM は、特定の手順に従います。CIMOM は、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と通信します。プロセスインジケーションの場合、CIMOM 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 オブジェクトマネージャおよび CIM オブジェクトマネージャリポジトリは、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場合、またはプロバイダによって CIMOM からのポーリングが禁じられている場合、CIMOM は次のイベントを処理します。
  - CIM_InstMethodCall
  - CIM_InstModification
  - CIM_InstDeletion
  - CIM_InstCreation
- CIM オブジェクトマネージャリポジトリは、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します。次のクラスがあります。
  - CIM_ClassCreation
  - CIM_ClassDeletion
  - CIM_ClassModification
  - CIM_InstCreation
  - CIM_InstModification
  - CIM_InstDeletion
  - CIM_InstRead

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 オブジェクトマネージャに委託することもできます。CIM オブジェクトマネージャは、プロバイダ上の enumerateInstances を呼び出します。CIMOM は、以前の状態のスナップショットと現在の状態のスナップショットを比較して、インスタンスの作成、変更、または削除が行われたかどうかを特定します。

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は、プロバイダ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 CIMOM 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、CIMOM 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 CIMOM 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 オブジェクトマネージャ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するため、CIM オブジェクトマネージャにより、CIM_InstCreation、CIM_InstDeletion、および CIM_InstModification に対して呼び出されます。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 オブジェクトマネージャ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、予約が承認されるかどうかを判断するため、CIMOM により、このメソッドが呼び出されます。プロバイダは、インジケーションハンドラの所有者(インジケーションの受信者) のユーザーID か、予約を作成したユーザーのユーザーID に基づいて決定を行うことができます。

   プロバイダが `Authorizable` インタフェースを実装しない場合、CIM オブジェクトマネージャは、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、CIMOM 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、CIMOM により呼び出されます。
4. `deActivateFilter`– 予約者または CIMOM によって予約が削除された場合に呼び出されます。たとえば、宛先ハンドラが正しく機能しない場合などです。

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は、 次の情報を提供します。

- アプリケーションが予約するインジケーション
- CIMOM 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。「**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 **4–21** CIM リスナーの追加

```
// CIM オブジェクトマネージャに接続する
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 クラスは文字列型のプロパティを持ちます。これらのプロパティの設定により、フィルタを一意に識別したり、照会文字列を指定したり、照会文字列の構文解析を行う照会言語を指定したりできます。現在は、WQL (WBEM Query Language) だけがサポートされます。

表 **4–6** 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 オブジェクトマネージャにより決定される
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 オブジェクトマネージャ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 オブジェクトマネージャは、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 オブジェクトマネージャ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_solarisdiskdrive"
```

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** インスタンス内にプロパティ値を設定して、次の情報を識別できるようにします。
   - フィルタ名
   - CIM イベントを選択するフィルタ文字列
   - 照会文字列の構文解析を行う照会言語 (WQL)

```
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 **(filter)** を作成します。このインスタンスを **CIM** オブジェクトマネージャリポジトリに格納します。

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

例 **4–22** イベントフィルタの作成

```
CIMClass cimfilter = cc.getClass(new CIMObjectPath
                                ("CIM_IndicationFilter"), true);
CIMInstance ci = cimfilter.newInstance();
//名前空間に test_a クラスが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 クラスを作成することにより、CIM_IndicationHandler クラスを拡張します。このサブクラスは、RMI プロトコルを使用するクライアントアプリケーションに、CIM イベントのインジケーションを送信します。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 **4–7** 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 オブジェクトマネージャにより指定される
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 オブジェクトマネージャ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 オブジェクトマネージャは、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 **4–23** 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、filter プロパティの値として 78 ページの「イベントフィルタの作成」で作成した filter オブジェクトパスを指定します。また、handler プロパティの値として、例 4–23 で作成した delivery オブジェクトパスを指定します。

例 **4–24** 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);
ci = filterdelivery.newInstance():

// filter インスタンスを参照する filter プロパティを作成
ci.setProperty("filter", new CIMValue(filter));

// delivery インスタンスを参照する handler プロパティを作成
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 ファイルには、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次のような状態を読み取ることができます。

- システムがシリアルポートにアクセスできない
- システムが正常にファイルシステムをマウントしている
- システム上で許可されている数より多くのプロセスが実行されている

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。`Solaris_LogServiceProperties` クラスの `singleton` インスタンスでは、次のプロパティを操作できます。

- ログファイル名
- ログファイルが格納されているディレクトリ
- ログファイルのサイズの制限
- 格納できるログファイル数
- メッセージを `syslogd(1M)` に転送するかどうか

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_Device.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 **4–8** 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

次の例では、ログを作成し、その内容を表示します。

例 **4–25** 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));
```

例 **4–25** Solaris_LogEntry のインスタンス作成　(続き)

```
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? ");
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()));
```

例 **4–25** Solaris_LogEntry のインスタンス作成　(続き)

```
                keys.addElement(logsvcKey);
                logsvcKey = new CIMProperty("SummaryMessage");
                System.out.println("Please enter a summary message:");
                logsvcKey.setValue(new CIMValue(d.readLine()));
                keys.addElement(logsvcKey);
                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();
            }

            // セッションの終了
            if (cc != null) {
                cc.close();
            }
        }
    }
```

例 **4–26** 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) {
```

例 **4–26** ログ記録リストの表示　(続き)

```
        System.out.println("Usage: ReadLog host username password " +
                           "[rmi|http]");
        System.exit(1);
    }

    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!");
```

例 **4–26** ログ記録リストの表示　(続き)

```
                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 : " +
    ((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 5 章

WBEM 照会の作成

この章では、WQL (WBEM Query Language) と照会 API を使った照会の作成方法を説明します。この章の内容は、次のとおりです。

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、標準 ANSI SQL (ANSI 構造化照会言語、American National Standards Institute Structured Query Language) のサブセットです。WQL には、Solaris 環境で WBEM をサポートできるように、意味的な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 サービスの実装では、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、CIMOM 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。

次の例では、`Storage_Capacity` プロパティに特定の値を持つ `Solaris_DiskDrive` クラスのインスタンスをすべて検索します。

```
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、OR 演算子 (論理和演算子) は AND 演算子 (論理積演算子) より後に評価される
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。返されるインスタンスには、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 を持つすべてのインスタンスを選択する
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 を持つ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、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るためのものです。この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。

次に、SELECT 文のあとに追加する WHERE 句の構文の例を示します。

```
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 文の後に追加するには、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す。
- 演算子は、有効な WQL 演算子です。
- 演算子のどちらかが、プロパティ名または定数になります。
- 任意の算術式を指定することはできません。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ます。

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

- WHERE 句内では、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ます。各グループは、演算子 AND、OR、または NOT で結合されている必要があります。

  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 が示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 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されます。

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。これらの式は、その他の AND、OR、および NOT 式と、バイナリ関係で構成されます。

葉ノードは、BinaryRelQueryExp で、*property operator constant* 形式の式で表されます。この形式は、プロパティと定数値の 2 項関係を表します。 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 になります。

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 コマンドで表示される、API 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 を参照してください。

照会を処理するプロバイダの記述

次の例のプロバイダプログラムは、照会 API を使って、 execQuery メソッドからプロバイダに渡された WQL 文字列の構文解析を行います。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)) {
```

例 **5–1** 照会を処理するプロバイダ　(続き)

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

第 6 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。

注 – `javax.wbem.provider` 内の WBEM プロバイダ API の詳細については、`/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

ディスクドライブや CPU といった管理リソース (管理対象のリソース) と通信してデータにアクセスする特別なクラスのことを「プロバイダ」と呼びます。プロバイダは、統合および解釈の目的で、Solaris WBEM サービスの主要 WBEM エージェントである CIMOM (CIM オブジェクトマネージャ) にデータを転送します。また、WBEM リソースのサブセットの管理作業を引き受けることにより、CIMOM の負荷を軽減します。データ転送には、`javax.wbem.provider` API を使用します。アプリケーションから CIM オブジェクトマネージャリポジトリにないデータを要求されると、CIMOM は、プロバイダインタフェースを使って、この要求を適切なプロバイダに転送します。

Solaris ソフトウェアプロバイダは、さまざまな領域で使用されます。たとえば、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、`cron` ツール、ネットワーク構成、製品レジストリで使用されます。さらに、デバイスやシステムのパフォーマンスの監視にも使用されます。

プロバイダは、インスタンスのテンプレートとなるクラスではなく、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存在しますが、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とメモリーを持ちながら、CIMOM から委託された作業を行います。WBEM を正しく機能させるためには、CIMOM に各プロバイダの位置を通知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、CIMOM 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、次のタスクを実行します。

- 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から CIM オブジェクトマネージャリポジトリで使用できない管理リソースのデータを要求された場合、CIMOM は、この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 CIMOM 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、そのデータを Java CIM クラスにマップしてから CIMOM に渡します。
- 管理リソースを制御する – CIMOM は、管理リソースを制御するため、管理アプリケーションから受け取った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、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てからデータを渡します。

注 – プロバイダと CIMOM 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、CIMOM を停止して再起動すると消去される
- CIM オブジェクトマネージャリポジトリ – この永続的なメモリーは、Solaris WBEM サービスソフトウェアをアンインストールすると消去される。プロバイダは、CIMOM ハンドルと内部プロバイダを使って、CIMOM からこのメモリーにアクセスする必要がある
- 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、システム呼び出しにより、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、処理できる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、CIMOM と通信し、クライアント API 経由で WBEM にアクセスします。CIMOM 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異なる場合があります。`/usr/sadm/lib/wbem/doc/index.html` を参照してください。

- CIM オブジェクトマネージャリポジトリにデータを格納している場合、プロバイダは、CIMOM へのハンドルを使ってこのリポジトリにアクセスします。これらのハンドルは、クライアント API のメソッドを呼び出します。99 ページの「プロバイダインタフェースの実装」を参照してください。
- CIM オブジェクトマネージャリポジトリ内にインスタンスまたは関連を作成する必要がある場合は、内部プロバイダを使用します。プロバイダは、インスタンスのメソッドか、WBEM 内部のアソシエータ (関連) プロバイダのメソッドを呼び出します。99 ページの「プロバイダインタフェースの実装」を参照してください。

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 **6–1** 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 CIMOM 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 1 種類以上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、CIMOM は、Java 言語で作成されたプロバイダのみをサポートします。

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 クラスでなければなりません。すなわち、接頭辞、続いて下線 (_)、最後に文字列の順で構成されていなければなりません。

  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 – プロバイダ修飾子の前には、必ず「`java:`」を付けます。これは、プロバイダが Java 言語で作成されていることを CIMOM に認識させるためです。

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、次の情報を示します。

- `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` – CIMOM を実装する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*) – CIM オブジェクトマネージャリポジトリにデータを格納する場合、プロバイダは、渡された CIMOM ハンドルを、CIMOM との通信に使用する CIMOM ハンドルに指定する必要があります。次に例を示します。

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

  CIM オブジェクトマネージャリポジトリ内にインスタンスを作成する必要がある場合、またはこのリポジトリ内の関連を操作する必要がある場合、プロバイダはまず、渡された CIMOM ハンドルをサブクラス ProviderCIMOMHandle にキャストする必要があります。次に、内部インスタンスまたは関連プロバイダを取得します。次に例を示します。

```
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、CIMOM 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、91 ページの「照会の構文解析」を参照してください。

例 **6–1** 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:
      * 名前だけではなくインスタンス全体を返す
      */
     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を含める
             ci = ci.filterProperties(propertyList, includeQualifier,
                                                    includeClassOrigin);
             instances.addElement(ci);
             loop += 1;
         } else {
             ci.setProperty("First", new CIMValue("red"));
             ci.setProperty("Last", new CIMValue("apple"));
             // 要求されたプロパティのみを含める
```

例 **6–1** CIMInstance プロバイダ　*(続き)*

```
            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を含める
            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();
                // 取得したいインスタンスを一意に識別するため
                // オブジェクトパス内の渡された項目からキーを取得することが
                // 必要である
                java.util.Vector keys = cop.getKeys();
                // これは例なので、キーをインスタンス内に配置するだけで
                // 終了する
                ci.setProperties(keys);
                // キープロパティ以外のプロパティがある場合はここに追加する

                // 要求されたプロパティのみを含める
                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 {
```

例 **6–1** CIMInstance プロバイダ　(続き)

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

     public CIMObjectPath createInstance(CIMObjectPath op,
                                         CIMInstance ci)
                                         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が Solaris WBEM プロバイダのメソッドを呼び出すためには、invokeMethod メソッドが必要です。これは、組み込み済みのプロバイダと開発者によって追加されたプロバイダの両方に当てはまります。

- 組み込み済みプロバイダ – 「プラットフォームに依存しない」CIM_* プロバイダ、または Solaris プラットフォーム固有の Solaris_* プロバイダ
- 開発者によって追加されたプロバイダ – 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

次のコード例では、CIMOM からの要求を 1 つ以上の特別な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を処理します。たとえば、Solaris_Package プロバイダは、Solaris_Package クラス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、システムのシャットダウン、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 **6–2** メソッドプロバイダ

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

    public void cleanup() throws CIMException {
```

例 **6–2** メソッドプロバイダ　*(続き)*

```
    }

    public CIMValue invokeMethod(CIMObjectPath op, String methodName,
            Vector inParams, Vector outParams)  throws CIMException {
        if (op.getObjectName().equalsIgnoreCase("solaris_computersystem")) {
            if (methodName.equalsIgnoreCase("reboot")) {
                // ここに表示されていないヘルパー関数を呼び出す
                return new CIMValue(rebootSystem());
            }
            if (methodName.equalsIgnoreCase("shutdown")) {
                // ここに表示されていないヘルパー関数を呼び出す
                return new CIMValue(shutdownSystem());
            }
        }
        if (op.getObjectName().equalsIgnoreCase("solaris_serialport")) {
            if (methodName.equalsIgnoreCase("disableportservice")) {
                // ここに表示されていないヘルパー関数を呼び出す
                return new CIMValue(deletePort(op));
            }
        }
        // ここに到達した場合はエラーを返す
        throw new CIMException(CIMException.CIM_ERR_NOT_SUPPORTED,
            "The requested function does not exist");
    }
    // 以下にヘルパー関数を定義する
    ...
}
```

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によって呼び出される個々の関連メソッド内の `objectName` 引数、すなわち `CIMObjectPath` は、クラスではなく「インスタンス」のオブジェクトパスになっていなければなりません。

CIMOM がインスタンスのオブジェクトパスを認識しない場合は、クライアントが CIM オブジェクトマネージャリポジトリ内の関連のクラス定義を求めているものと見なします。関連のクラス定義には、その関連のメンバーインスタンスを派生するテンプレートが含まれます。したがって、CIMOM は、プロバイダの関連メソッド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。CIMOM 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処理します。

例 **6–3** CIMAssociator プロバイダ

```
...

public CIMInstance[] associators(CCIMObjectPath assocName, CIMObjectPath
                objectName, String resultClass, String role, String
                resultRole, boolean includeQualifiers, boolean
```

例 **6–3** CIMAssociator プロバイダ　*(続き)*

```
                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"))) {
            // 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、次のタスクを実行する必要があります。

- `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 CIMOM 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。
   次に例を示します。

```
public class sampleEventProvider implements
        InstanceProvider EventProvider{

    // プロバイダが CIM オブジェクトマネージャ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 **6–2** に示すそれぞれのメソッドを実行します。

3. 作成、変更、および削除のインスタンスイベント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** オブジェクトマネージャに送信します。

```
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。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場合、CIMOM は、このインタフェースに含まれるメソッドを使ってプロバイダに通知します。このメソッドは、ク

ライアントが CIM イベントの予約を取り消した場合にも使用されます。さらにこれらのメソッドにより、CIMOM 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も指定します。

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 **6–2** `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、CIMOM 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
`authorizeFilter`	クライアントが予約を作成すると、CIMOM 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、CIMOM 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、CIMOM 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

CIMOM 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 句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型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、CIMOM 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する場合、CIMOM は、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。プロバイダは、`activateFilter` メソッド

を呼び出すことにより、この通知を行います。また、CIMOM 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、CIMOM の要求に応答します。通常、プロバイダは、CIMOM が activateFilter メソッドを呼び出すときに設定されるフラグ変数を定義します。このフラグの設定は、CIMOM が deActivateFilter メソッドを呼び出すときにクリアされます。そのあと、プロバイダは、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、作成された 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ます。プロバイダは、deliverEvent メソッドを使って、CIMOM にインジケーションを返します。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。CIMOM 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。CIMOM は、この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送信されるインジケーションを受信すること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

Java Native Interface は、JDK ソフトウェアの一部です。Java Native Interface を使ってプログラムを作成すれば、その Java プログラムコードはどのプラットフォームにも移植可能です。Java コードは、Java Native Interface により、Java 以外の言語 (C、C++、アセンブラなど) で作成されたアプリケーションやライブラリを使用できます。

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 オブジェクトマネージャに `.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。

次に例を示します。

```
/* 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。

   次に例を示します。

```
/* 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/")));
```

   次の表に、標準的な URL 形式を示します。

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。

   次に例を示します。

```
// 更新されたインスタンスを CIM オブジェクトマネージャに渡す
cc.createInstance(new CIMObjectPath(), ci);
```

▼ プロバイダを登録する方法

新しいプロバイダ、または変更されたプロバイダを CIM オブジェクトマネージャに登録して、プロバイダがサポートするデータおよび操作に関する情報を送受信できるようにします。また、CIMOM にプロバイダの場所を通知するためにプロバイダを登録します。CIM オブジェクトマネージャは、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 ファイルの詳しい作成方法については、DMTF の 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、**CIMOM** のプロバイダの種類と場所を指定します。

   次に例を示します。

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

   この修飾子は、次の情報を示します。

   - `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 **6–4** 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 7 章

MOF コンパイラを使用した JavaBeans コンポーネントの作成

この章では、MOF (Managed Object Format) コンパイラの概要を説明します。また、`mofcomp` コマンドで `-j` オプションを使用して JavaBeans™ コンポーネントを作成する方法についても説明します。この章の内容は次のとおりです。

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。

MOF コンパイラについて

MOF (Managed Object Format) は、Distributed Management Task Force (DMTF) によって開発されたコンパイラ言語です。MOF 言語は、CIM と WBEM の静的クラス、動的クラス、およびインスタンスを定義します。Solaris WBEM サービス付属の CIM および Solaris MOF ファイルを使用できます。また、独自の MOF ファイル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 ファイルを作成する方法の詳細については、DMTF の Web サイト (`http://www.dmtf.org`) を参照してください。

MOF コンパイラ (`mofcomp(1M)`) は、次のタスクを実行します。

- MOF ファイルの構文解析
- クラスおよびインスタンスの Java プログラミング言語への変換
- デフォルトの名前空間 (`root\cimv2`) または指定された名前空間内の CIM オブジェクトマネージャリポジトリへのクラスの追加

MOF ファイルは、簡単に Java プログラミング言語に変換できます。その結果、Java テクノロジベースのアプリケーションは、Java 仮想マシンを実行する任意のマシン上で MOF ファイル内のデータを解釈したり、交換したりできます。

Solaris のインストール時に、MOF コンパイラは、CIM スキーマおよび Solaris スキーマを記述する MOF ファイルを自動的にコンパイルし、CIM オブジェクトマネージャリポジトリに追加します。

mofcomp を使用した JavaBeans コンポーネントの生成

JavaBeans コンポーネント (Bean) は、WBEM のコンテキストで、CIM クラスおよびデータにアクセスしたり、これらを操作したりするためのメソッドを定義します。開発作業の効率化のため、`mofcomp` コマンドに `-j` オプションを指定して、MOF ファイル内の CIM クラスから Bean を生成できます。これらの自動生成された Bean はインタフェースを定義します。実装コードは追加する必要があります。

注 – プログラムが、基盤となる JavaBeans 実装に対して行なった変更の影響を受けないようにするため、元の JavaBeans コンポーネント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` オプションを指定した場合、Java インタフェース `CIMBean.java` と、このインタフェースを実装する Bean、`CIMBeanImpl.java` 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。生成されるすべての Java インタフェースは、`CIMBean.java` から拡張されます。生成されるすべての Bean は、`CIMBeanImpl.java` から拡張され、元となる実装を継承します。

MOF コンパイラの JavaBeans 生成機能では、MOF ファイルに定義された各 CIM クラスに対して、次のメソッドを含む Java プログラミング言語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。オブジェクトデータ型や Bean メソッドのパラメータのデータ型は変更できないのでこのコンテナインタフェースは必須です。よってこれらのデータ型は、データの入出力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。また次の形式で IMPORTS および EXCEPTIONS 要素も指定できます。

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 **7–1** 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 クラスの名前を指定する。複数のクラスを指定する場合はコロン (:) で区切る。必要な数の Java クラスを複数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追加する Java 例外の名前を指定する。複数の例外を指定する場合はコロン (:) で区切る。必要な数の Java クラス例外を複数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 プログラミング言語にマップする方法

次の表に、CIM 要素を Java プログラミング言語の要素にマップする方法を示します。

表 **7–2** 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。出力パラメータはメソッドシグニチャーに直接追加されずに、メソッドパラメータとして追加される出力コンテナオブジェクトにカプセル化される
修飾子	修飾子については、表 7–4 および 表 7–5 を参照
関連	特定の要素は要求されていない
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 **7–3** 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);`

表 7–3 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
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);
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 7–4 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 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。このマッピングに基づいて、個々のインタフェースおよびクラスの Javadoc API ドキュメンテーションが生成されます。

表 **7–5** 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 プログラミング言語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
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();`
VALUEMAP	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。定数の名前と値を取得してこれらのクラス変数を生成する方法は、プロパティとその修飾子のデータ型によって異なる 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 はプロパティの正当な値セットを定義する。Values は、整数値と文字列の相互変換を行う

表 7–5 標準修飾子　(続き)

修飾子	スコープ	意味
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。定数の名前と値を取得してこれらのクラス変数を生成する方法は、プロパティとその修飾子のデータ型によって異なる 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 はプロパティの正当な値セットを定義する。Values は、整数値と文字列の相互変換を行う
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 7–6 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 Javadoc クラスの説明

JavaBeans コンポーネントの生成例

次の例では、`mofcomp` コマンドに `-j` オプションを使用したときに生成される JavaBeans 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 – mofcomp コマンドに -u (ユーザー) オプションと -p (パスワード) オプションの両方を指定することは避けてください。コマンド行に直接パスワードを入力する代わりに、-u オプションだけを指定して、暗号化されたパスワードの入力プロンプトが表示されるようにすることができます。

例 **7–1** JavaBeans コンポーネントの生成

/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
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
     *
```

例 **7–1** JavaBeans コンポーネントの生成　*(続き)*

```
     * @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;

    /**
     * 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
```

例 **7–1** JavaBeans コンポーネントの生成　*(続き)*

```
     *
     * @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。空の文字列が
     * 返されるかもしれないが、NULL 文字は返されない
     *
     * @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;
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。'mofcomp -j'
 * によって生成される 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はパラメータを取らず、
     * CIMBeanImpl の空のインスタンスを作成する
     */
    public CIMBeanImpl() {
```

例 **7–1** JavaBeans コンポーネントの生成　*(続き)*

```
    super();

    } // コンストラクタ

    /**
     * このコンストラクタは、指定の CIMOMHandle と CIMInstance を
     * 取り、CIMBeanImpl を作成する
     *
     * @param      CIMOMHandle は CIMOM のハンドル
     * @param      CIMInstance は管理対象の CIMInstance のハンドル
     *
     */
    public CIMBeanImpl(CIMOMHandle handle, CIMInstance instance) {

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

    } // コンストラクタ

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

    return (cimomHandle);

    } // getCIMOMHandle

    /**
     * このメソッドは CIMBean の CIMOMHandle に指定の値を設定する
     *
     * @param CIMOMHandle は CIMOM 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 は管理対象の CIMInstance のハンドル
     */
    public CIMInstance getCIMInstance() {

    return (cimInstance);

    } // getCIMInstance
```

例 **7–1** JavaBeans コンポーネントの生成　*(続き)*

```
    /**
     * このメソッドは CIMBean の CIMInstance に指定の値を設定する
     *
     * @param CIMInstance は管理対象の CIMInstance のハンドル
     */
    public void setCIMInstance(CIMInstance instance) {

    this.cimInstance = instance;

    } // setCIMInstance

    /**
    * このメソッドは、リモート呼び出しにより、CIMOM 内の
    * CIMInstance を更新する
    */
    public void update() throws CIMException {

    cimomHandle.setInstance(getObjectPath(), cimInstance);

} // 更新

    /**
     * このメソッドは、リモート呼び出しにより、CIMOM 内の
     * CIMInstance の指定の CIMProperty を更新する
     *
     * @param String は CIMInstance 内の更新されるプロパティの名前
     * @param Object は CIMProperty 内の更新される値
     */
    public void update(String propName, Object value) throws CIMException {

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

    } // 更新

    /**
     * このメソッドは、リモート呼び出しにより、CIMOM 内の
     * CIMInstance を削除する
     */
    public void delete() throws CIMException {

    cimomHandle.deleteInstance(getObjectPath());

    } // 削除

    /**
     * サブクラスが指定の CIMProperty の CIMValue 内のオブジェクトを
     * 取得するために使用できる簡易メソッド
     * 注: 返されるオブジェクトが NULL の場合もある
     *
     * @param    String はプロパティ名。このプロパティの値が取得される
     * @return    Object は CIMProperty の CIMValue に含まれるオブジェクト
     */
```

例 **7–1** JavaBeans コンポーネントの生成　*(続き)*

```
    protected Object getProperty(String propName) {

    try {

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

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

    } // getProperty

    /**
     * サブクラスが指定の CIMProperty の CIMValue 内の Vector と
     * 等価の String[] を取得するために使用できる簡易メソッド
     * 注: 返される String[] が NULL の場合もある
     *
     * @param String はプロパティ名。このプロパティの値が取得される
     * @param String[] はプロパティ Values の修飾子のデータ
     * @param Object[] はプロパティ ValueMap の修飾子のデータ
     * @return String[] は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());

    } 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
```

例 **7–1** JavaBeans コンポーネントの生成　*(続き)*

```
/**
 * このメソッドは、プロパティ値によって参照される CIMInstance
 * (指定のオブジェクトパス) を取得し、指定の Bean 内に設定する。
 * このメソッドは、Association プロパティのアクセスメソッドに
 * よって使用される
 *
 * @param    CIMObjectPath は CIMInstance のオブジェクトパス
 * @param    CIMBeanImpl は取得される 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

/**
 * サブクラスが、指定の名前の CIMProperty 内に指定の Object 値
 * を持つ CIMValue を設定するために使用できる簡易メソッド
 *
 * @param    String    は新しい値を設定されるプロパティの名前
 * @param    Object    は CIMInstance 内の更新されるプロパティ値
 */
protected void setProperty(String propName, Object propValue)throws
IllegalArgumentException {

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

} // setProperty

/**
 * サブクラスが、指定の名前の CIMProperty 内に指定の String[] と
 * 等価の Vector を含む CIMValue を設定するために使用できる
 * 簡易メソッド
 *
 * @param String は値を取得されるプロパティの名前
 * @param String[] はプロパティ Values の修飾子のデータ
 * @param Object[] はプロパティ ValueMap の修飾子のデータ
 * @param String[] は CIMInstance 内に設定されるプロパティ値
 *
 */
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++) {
```

例 **7–1** JavaBeans コンポーネントの生成　*(続き)*

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

    }
    setProperty(propName, vPropValue);

    } // setArrayProperty

    /**
     * このメソッドは、CIMInstance 内の Key 修飾プロパティ名
     * の文字配列を返す。CIMInstance に修飾子の情報が含まれていない
     * 場合、その CIMObjectPath を構築するために必要
     *
     * @return    String[] はキー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 は 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)) {

        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);
```

例 **7–1** JavaBeans コンポーネントの生成　*(続き)*

```
    } // getObjectPath

    /**
     * 指定の配列内の指定のオブジェクトのインデックスか、
     * 配列内にオブジェクトが存在しない場合は '-1' を返す
     * 簡易メソッド
     *
     * @param    Object[] はオブジェクトのインデックスの検索先と
     * なるオブジェクト配列
     * @param    Object はインデックスを検索するオブジェクト配列内の
     * オブジェクト
     * @return    int はオブジェクト配列内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 修飾子の値か、この
    * 修飾子が存在しない場合は '-1' を返す
     *
     * @return    String は Version 修飾子の値か、この修飾子が存在
     * しない場合は'-1'
     */
    public String getBeanVersion() {

    return ("-1");

    } // 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、
     * デバッグの目的で使用される。文字列の形式は、実装によって
     * 異なる。空の文字列が返されるかもしれないが、NULL 文字は
     * 返されない
     *
     * @return    String は Bean の文字列表現
     */
    public String toString() {

    return (cimInstance.toString());

    } // toString

} // Class CIMBeanImpl
```

Simple_ClassBean.java の内容は次のとおりです。

```
package foo.com;
```

例 **7–1** JavaBeans コンポーネントの生成　*(続き)*

```
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
     *
     * @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。

例 **7–1** JavaBeans コンポーネントの生成　*(続き)*

```
package foo.com;

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

/**
 * このクラスには、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_ClassBean 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、
     * 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)
    {

     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;
```

例 **7–1** JavaBeans コンポーネントの生成　*(続き)*

```
    } // 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


    /**
     * 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
```

例 **7–1** JavaBeans コンポーネントの生成　*(続き)*

```
} // Simple_ClassBeanImpl クラス
```

第 8 章

セキュリティの管理

この章では、WBEM のセキュリティ機構と CIMOM (CIM オブジェクトマネージャ) が実施する機能について説明します。

この章では、次の内容について説明します。

WBEM のセキュリティ機構

WBEM は、そのデータへのセキュリティ保護されたアクセスを確保するために、次のようないくつかの機構を採用しています。

- 認証 – WBEM サーバーにクライアントのユーザー ID を示し、クライアントの資格情報を使ってクライアントを検証するプロセス。
- 役割の引き受け – WBEM サーバーが承認検査を行う際に Solaris の役割によるアクセス制御 (RBAC) の役割 ID が WBEM サーバーによって使用されることを認めるプロセス。
- セキュリティ保護されたメッセージング – それぞれのクライアント要求メッセージにセキュリティ保護されたメッセージオーセンティケータを追加するプロセス。このオーセンティケータにより、WBEM サーバーはメッセージの送信元を検査し、配信時にメッセージが変更されなかったかどうかを判別します。
- 承認 – 認証されたユーザーまたは役割 ID が、WBEM によって管理されているデータへのアクセス権を許可されていることを確認するプロセス。承認の管理には、Solaris 管理コンソールのユーザーツールや、Sun WBEM ユーザーマネージャを使用します。

- 監査 – WBEM サーバーによって実行された特定の操作の監査レコードを書き込むプロセス。これらのレコードは、認証されたユーザーが WBEM サーバーシステム上の管理データに加えた変更を追跡します。
- ロギング – 特定のセキュリティ関連イベントを WBEM ログに書き込みます。WBEM ログは、Solaris 管理コンソールのログビューアを使って参照することができます。

ここに挙げたそれぞれの機構について、このあと詳しく説明します。

クライアントの認証

クライアントアプリケーションが CIMOM サーバーに接続するときには、WBEM サーバー上の CIMOM によってクライアントのユーザー ID が認証されなければなりません。ユーザーの WBEM クライアントは、Solaris ユーザー ID とそのログインパスワードを提供する必要があります。ID と資格情報は、クライアントと WBEM サーバー間のセキュリティ認証交換で使用されます。この交換により、クライアントが WBEM サーバーシステムへのログインを許可された有効な Solaris ユーザーであるかどうかを検証します。

WBEM サーバーがユーザーの ID と資格情報を検証できない場合や、ユーザー ID が無効な場合、WBEM サーバーは CIM セキュリティ例外を返します。この例外には、`NO_SUCH_PRINCIPAL` エラーが含まれます。

WBEM サーバーがユーザーの ID と資格情報を検証できない場合や、ユーザーのパスワードが無効な場合、WBEM サーバーは CIM セキュリティ例外を返します。この例外には、`INVALID_CREDENTIAL` エラーが含まれます。

役割の引き受け

役割 ID を引き受けられるのは、WBEM ユーザーが Remote Method Invocation (RMI) プロトコルを選択した場合だけです。役割の引き受けは、XML over HTTP プロトコルではサポートされません。

Solaris での WBEM 実装は、クライアントが WBEM サーバー上の CIMOM によって認証されるときに、クライアントが Solaris の役割 ID を引き受ける機能をサポートします。RBAC 承認を確認する際、WBEM サーバーは、役割を引き受けるユーザーの ID に許可されたアクセス権ではなく、役割に許可されたアクセス権を使用します。

RBAC の役割については、『*Solaris* のシステム管理 *(*セキュリティサービス*)*』の「役割によるアクセス制御 (概要)」を参照してください。

クライアントは、接続しようとするときに、Solaris のユーザー ID とパスワードに加えて、Solaris の役割 ID とパスワードを入力する必要があります。

WBEM サーバーが Solaris の役割 ID を検証できない場合、WBEM サーバーは、`NO_SUCH_ROLE` エラーで CIM セキュリティ例外を返します。

引き受けた役割 ID に対して役割のパスワードが無効な場合、WBEM サーバーは、INVALID_CREDENTIAL エラーで CIM セキュリティ例外を返します。

役割 ID と役割のパスワードが有効であっても、ユーザーがその役割を引き受けることを許可されていない場合、WBEM サーバーは例外を返します。CANNOT_ASSUME_ROLE エラーで CIM セキュリティ例外を返します。

セキュリティ保護されたメッセージング

CIM RMI プロトコルでは、クライアントから WBEM サーバーに渡される個々の要求に、メッセージデータから構築されたメッセージオーセンティケータが含まれます。認証交換の間に確立されたセッションキーと共に、一方向ダイジェストも作成されます。

WBEM サーバーは、このメッセージオーセンティケータを検証します。これにより、要求が認証済みのクライアントからのものであり、メッセージがサーバーに届くまでの間に変更または再送信されていないことが保証されます。

元のクライアント以外のクライアントによって変更、再送信、または作成されたメッセージを検出すると、WBEM サーバーは CIM セキュリティ例外を返します。この例外には、CHECKSUM_ERROR エラーが含まれます。さらに、WBEM サーバーは、ログメッセージを WBEM ログに書き込みます。

承認

WBEM サーバーが接続したあとは、WBEM サーバーは CIM クライアントとのそれ以降の操作におけるすべての承認検査について、認証されたユーザーまたは役割 ID を使用します。

WBEM は、次の機構により、2 種類の承認検査をサポートします。

- アクセス制御リスト (ACL)。これは、特定の名前空間について WBEM サーバーによって維持される
- RBAC 承認。これは、Solaris オペレーティング環境の一部として構成される

WBEM がどの承認検査機構を使用するかは、MOF クラスプロバイダがどのように実装されているかに応じて決まります。WBEM が特定の MOF クラス操作で使用する承認検査は、次の要因によって決まります。

- WBEM が実行する操作
- MOF クラスの実装方法

Solaris_Acl.mof で定義されているクラスは、WBEM の ACL ベースのセキュリティを実装します。WBEM の ACL ベースのセキュリティは、Solaris WBEM サービスにデフォルトの承認スキーマを提供します。WBEM ベースのセキュリティは、特定の状況で、特定の CIM 操作セットに適用されます。ACL ベースのセキュリティは、Solaris WBEM サービスによって提供される固有の機能です。

WBEM サーバー上の特定の名前空間について ACL を確立するには、Sun WBEM ユーザーマネージャ (`wbemadmin`) を使用します。Sun WBEM ユーザーマネージャを使って、名前空間の ACL に、ユーザー名または役割名を追加できます。ユーザーごとに読み取り権や書き込み権を割り当てることもできます。Sun WBEM ユーザーマネージャについては、137 ページの「Sun WBEM ユーザーマネージャを使ってアクセス制御を設定する」 および `wbemadmin(1M)` のマニュアルページを参照してください。

書き込み権のあるユーザーは、クラスのメタデータの変更、その名前空間にある MOF クラスのインスタンスの変更、およびインスタンスに対する呼び出しメソッドの発行が可能です。ローカル WBEM サーバーの `root` ユーザー ID には、いつでも、サーバー上の名前空間すべてに対する書き込み権が許可されます。明示的な ACL 項目のない、すべての認証されたユーザーには、デフォルトで読み取り権が許可されます。

MOF クラスのメタデータのアクセスを含む操作 (たとえば、`getClass`) は、WBEM の ACL を使用します。これらの操作には、認証されたユーザーに、MOF クラスを含む名前空間についての ACL による許可されたアクセス権の検査が含まれます。ACL 項目に RBAC の役割を設定することもできますが、ACL 項目はいつもユーザー ID に対して検査され、役割 ID に対しては検査されません。つまり、ACL に役割名を設定することはできますが、CIMOM は実行時にその役割名を検査しないということです。

MOF クラスのインスタンスが関係する操作には、WBEM ACL か RBAC 承認のいずれかの検査が含まれる可能性があります。

また、ユーザー、または役割 ID にアクセス権を許可して、プロバイダが RBAC 承認を使用している MOF クラスのインスタンスをそのユーザーがアクセスおよび変更できるようにすることもできます。これらのアクセス権を許可するには、Solaris 管理コンソールのユーザーツールに含まれる権利ツールを使用します。ユーザーへのアクセス権の許可については、『*Solaris* のシステム管理 (セキュリティサービス)』の「権利プロファイルの作成または変更」を参照してください。

MOF クラスのインスタンスが WBEM の永続的なデータストアに格納されている場合、CIMOM は、その MOF クラスを含む名前空間についての WBEM ACL を検査します。MOF クラスプロバイダの実装は、次の条件下では、ほぼ確実に RBAC 承認検査を行います。

- MOF クラスプロバイダの実装がプロバイダのデータストアにアクセスする場合
- MOF クラスプロバイダの実装が Solaris オペレーティング環境のシステムデータにアクセスする場合

一般に、MOF クラス定義にプロバイダ修飾子が含まれている場合、プロバイダの実装は通常、RBAC 承認検査を行います。MOF クラス定義にプロバイダ修飾子が含まれていない場合、CIMOM は次のことを実行します。

- MOF クラスのインスタンスを WBEM の永続的なデータストアに格納する
- MOF クラスの名前空間へのアクセスを制御する ACL を検査して、アクセスが許可されていることを確認する

監査

WBEM サーバーは、処理中に特定のイベントについて監査レコードを書き込みます。たとえば、クライアント認証が成功または失敗したとき、およびユーザー情報を変更する操作が実行されたときにはいつでも、WBEM サーバーは監査レコードを書き込みます。

WBEM サーバーは、基本となる Solaris Basic Security Module (BSM) を使用して監査レコードを書き込みます。監査情報が確実に記録されるようにするには、WBEM サーバー上の Solaris オペレーティング環境で BSM の監査機構 (`bsmconv`) を有効にする必要があります。 `bsmconv` コマンドについては、`bsmconv(1M)` のマニュアルページを参照してください。

注 – Trusted Solaris™ ソフトウェアを使用している場合は、BSM の監査機構を有効にする必要はありません。

ロギング

WBEM サーバーは、特定のセキュリティイベントが発生すると、WBEM ログにログレコードを書き込みます。たとえば、クライアントの認証済みセッションが確立された場合や承認が失敗した場合などです。WBEM ログは、Solaris 管理コンソールのログビューアで調べることができます。ログビューアについては、第 9 章を参照してください。

セキュリティ関連のログイベントは、カテゴリが「セキュリティ (Security)」のログによって見分けることができます。カテゴリは、「カテゴリ (Category)」列に表示されます。セキュリティ関連のログメッセージだけを表示するには、ログビューアのログフィルタダイアログボックスで「セキュリティ (Security)」というカテゴリを選択します。ほとんどのセキュリティログメッセージには、クライアントのユーザー ID とクライアントホストの名前が含まれています。

Sun WBEM ユーザーマネージャを使ってアクセス制御を設定する

Sun WBEM ユーザーマネージャ (`wbemadmin`) では、ユーザー自身や、特権を持つその他のユーザーが、次の操作を実行できます。

- 承認されたユーザーの追加および削除
- 承認されたユーザーのためのアクセス特権の設定

- ユーザー認証の管理と、WBEM 対応のシステム上の CIM オブジェクトへのユーザーアクセス

注 – アクセス制御を指定するユーザーは、Solaris ユーザーアカウントを持っていなければなりません。

Sun WBEM ユーザーマネージャで実行できることと実行できないこと

アクセス特権は、個々の名前空間、またはユーザーと名前空間の組み合わせに対して設定できます。ユーザーを追加し、名前空間を選択すると、そのユーザーには、選択した名前空間内の CIM オブジェクトに対する読み取り権がデフォルトで許可されます。

注 – ユーザーのアクセス権と名前空間のアクセス権を効果的に組み合わせたい場合は、まず名前空間へのアクセスを制限します。次に、個々のユーザーに、その名前空間に対する読み取り権、読み取り/書き込み権、または書き込み権を許可します。

個々の管理対象オブジェクトにアクセス権を設定することはできません。しかし、ある名前空間内のすべての管理対象オブジェクトにアクセス権を設定したり、ユーザーごとにアクセス権を設定したりすることはできます。

`root` としてログインすると、CIM オブジェクトに対して次のような種類のアクセス権を設定できます。

- 読み取り専用 – CIM スキーマオブジェクトへの読み取り専用のアクセス権を許可する。このアクセス権を持つユーザーは、インスタンスとクラスを取得することはできるが、CIM オブジェクトの作成、削除、および変更はできない
- 読み取り/書き込み – すべての CIM クラス、インスタンス、および呼び出しメソッドに対する完全な読み取り、書き込み、および削除のアクセス権を許可する
- 書き込み – すべての CIM クラスおよびインスタンスに対する書き込みおよび削除のアクセス権を許可するが、読み取り権は許可しない
- アクセス権なし – CIM クラスおよびインスタンスに対するアクセス権を許可しない

Sun WBEM ユーザーマネージャの使用方法

この節では、Sun WBEM ユーザーマネージャを起動して使用する方法を説明します。

▼ Sun WBEM ユーザーマネージャを起動する方法

1. スーパーユーザーになります。
2. コマンドウィンドウで、次のコマンドを入力します。

   ```
   # /usr/sadm/bin/wbemadmin
   ```

   Sun WBEM ユーザーマネージャが起動し、「ログイン (Login)」ダイアログボックスが表示されます。

   注 – 「ログイン (Login)」ダイアログボックスのいずれかのフィールドをクリックすると、そのコンテキストヘルプ情報が「コンテキストヘルプ (Context Help)」パネルに表示されます。

   ---

3. 「ログイン **(Login)**」ダイアログボックスの各フィールドに情報を入力します。

   a. 「ユーザー名 **(User Name)**」フィールドに、ユーザー名を入力します。

      注 – ログインするには、`root\security` 名前空間に対する読み取り権が必要です。デフォルトでは、Solaris ユーザーはゲスト特権を持っているため、デフォルトの名前空間に対する読み取り権が許可されています。読み取り権を持つユーザーは、ユーザー特権を表示することはできますが、変更することはできません。

      ユーザーにアクセス権を許可するためには、`root` としてログインするか、`root\security` 名前空間に対する書き込み権を持つユーザーとしてログインする必要があります。

      ---

   b. 「パスワード **(Password)**」フィールドに、ユーザーアカウントのパスワードを入力します。

4. 「了解」をクリックします。

   「ユーザーマネージャ (User Manager)」ダイアログボックスが開きます。このダイアログボックスには、ユーザー名と、現在のホスト上の名前空間内の WBEM オ

ブジェクトに対するアクセス権の一覧が表示されます。

▼ ユーザーにデフォルトのアクセス権を許可する方法

1. **Sun WBEM** ユーザーマネージャを起動します。
2. ダイアログボックスの「ユーザーアクセス **(Users Access)**」の部分で、「追加 **(Add)**」をクリックします。
   使用できる名前空間の一覧を表示したダイアログボックスが開きます。
3. 「ユーザー名 **(User Name)**」フィールドに、**Solaris** ユーザーアカウントの名前を入力します。
4. 表示された名前空間の中から名前空間を **1** つ選択します。
5. 「了解 **(OK)**」をクリックします。
   このユーザー名が「ユーザーマネージャ (User Manager)」ダイアログボックスに追加されます。
6. 変更を保存し、「ユーザーマネージャ **(User Manager)**」ダイアログボックスを閉じるには、「了解 **(OK)**」をクリックします。変更を保存し、ダイアログボックスを開いたままにするには、「適用 **(Apply)**」をクリックします。
   指定したユーザーに、選択した名前空間内の CIM オブジェクトに対する読み取り権が許可されます。

▼ ユーザーのアクセス権を変更する方法

1. **Sun WBEM** ユーザーマネージャを起動します。
2. アクセス権を変更するユーザーを選択します。
3. ユーザー特権を設定します。ユーザーに読み取り専用のアクセス権を許可するには、「読み取り **(Read)**」チェックボックスをクリックします。ユーザーに書き込み権を許可するには、「書き込み **(Write)**」チェックボックスをクリックします。
4. 変更を保存し、「ユーザーマネージャ **(User Manager)**」ダイアログボックスを閉じるには、「了解 **(OK)**」をクリックします。変更を保存し、ダイアログボックスを開いたままにするには、「適用 **(Apply)**」をクリックします。

▼ ユーザーのアクセス権を削除する方法

1. **Sun WBEM** ユーザーマネージャを起動します。

2. ダイアログボックスの「ユーザーアクセス **(Users Access)**」の部分で、アクセス権を削除するユーザー名を選択します。

3. 名前空間に対するこのユーザーのアクセス権を削除するために、「削除 **(Delete)**」をクリックします。

   確認を求めるダイアログボックスが開きます。このダイアログボックスでは、ユーザーのアクセス権を本当に削除してよいかどうか確認を求められます。

4. 削除を確定するには、「了解 **(OK)**」をクリックします。

5. 変更を保存し、「ユーザーマネージャ **(User Manager)**」ダイアログボックスを閉じるには、「了解 **(OK)**」をクリックします。変更を保存し、ダイアログボックスを開いたままにするには、「適用 **(Apply)**」をクリックします。

▼ 名前空間のアクセス権を設定する方法

1. **Sun WBEM** ユーザーマネージャを起動します。

2. ダイアログボックスの「ネームスペースへのアクセス **(Namespace Access)**」の部分で、「追加 **(Add)**」をクリックします。

   ダイアログボックスが開きます。このダイアログボックスには、使用可能な名前空間の一覧が表示されます。

3. アクセス権を設定する名前空間を選択します。

   注 – デフォルトでは、ユーザーは、名前空間に対する読み取り専用のアクセス権を許可されます。

   - 名前空間に対してアクセス権をまったく許可しない場合は、「読み取り (Read)」および「書き込み (Write)」の各チェックボックスをどちらも選択しません。
   - 書き込み権を許可するには、「書き込み (Write)」を選択します。
   - 読み取り権を許可するには、「読み取り (Read)」を選択します。

4. 変更を保存し、「ユーザーマネージャ **(User Manager)**」ダイアログボックスを閉じるには、「了解 **(OK)**」をクリックします。変更を保存し、ダイアログボックスを開いたままにするには、「適用 **(Apply)**」をクリックします。

▼ 名前空間のアクセス権を削除する方法

1. **Sun WBEM** ユーザーマネージャを起動します。

2. ダイアログボックスの「ネームスペースへのアクセス **(Namespace Access)**」の部分で、アクセス制御を削除する名前空間を選択して、「削除 **(Delete)**」をクリックします。

名前空間からアクセス制御が削除され、「Sun WBEM ユーザーマネージャ (Sun WBEM User Manager)」ダイアログボックスの名前空間の一覧からその名前空間が削除されます。

3. 変更を保存し、「ユーザーマネージャ **(User Manager)**」ダイアログボックスを閉じるには、「了解 **(OK)**」をクリックします。変更を保存し、ダイアログボックスを開いたままにするには、「適用 **(Apply)**」をクリックします。

Solaris WBEM SDK API を使用してアクセス制御を設定する方法

WBEM SDK のアプリケーションプログラミングインタフェース (SDK API) を使って、名前空間、またはユーザー単位でアクセス制御を設定できます。これらのセキュリティクラスは、`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することにより、名前空間内の CIM オブジェクトに対して、個々のユーザーのアクセス制御を設定できます。次に、API を使って、そのインスタンスのアクセス権を変更します。同様に、最初に `Solaris_NameSpaceACL` クラスのインスタンスを作成することにより、名前空間のアクセス制御を設定できます。次に、`createInstance` メソッドなどの API を使って、そのインスタンスのアクセス権を設定します。

この 2 つのクラスを効果的に組み合わせたい場合は、まず `Solaris_NameSpaceACL` クラスを使用して、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します。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` クラスから、デフォルト値 `r` (読み取り専用) の文字列型プロパティ *capability* を継承します。

capability プロパティを次のいずれかの値に設定することで、アクセス権を設定できます。

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 2 つ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を組み合わせた ACL 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の設定方法

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 オブジェクトマネージャに渡す
cc.createInstance(new CIMObjectPath(), ci);
...
```

Solaris_NamespaceAcl クラス

Solaris_NamespaceAcl は、Solaris_Acl クラスから、デフォルト値 -r (すべてのユーザーに読み取り専用アクセスを適用) の文字列型プロパティ *capability* を継承します。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の設定方法

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。

```
...
/* ローカルホスト上の root\security (名前空間の名前)
で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("", "root\security");

// root ユーザーとして root\security 名前空間に接続する
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 名前空間のアクセス権 (capability)
```

```
を読み取り/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 オブジェクトマネージャに渡す
cc.createInstance(new CIMObjectPath(), ci);
```

WBEM のセキュリティに関する問題の障害追跡

この節では、次のような問題が発生した場合の対処方法を説明します。

- クライアント (ユーザー) が WBEM サーバー上の CIMOM によって認証されない
- WBEM サーバーが役割を引き受けない
- ACCESS_DENIED エラーが発生する

クライアント (ユーザー) が WBEM サーバー上の CIMOM によって認証されない場合

クライアントが WBEM サーバー上の CIMOM によって正常に認証されない場合、WBEM サーバーは CIM セキュリティ例外を返します。この例外は、サーバーがクライアントアプリケーション内に CIM クライアントハンドルを作成しようとしたときに返されます。この例外には、認証の試行が失敗した理由を示すエラーコードが含まれています。

WBEM サーバーがユーザーの ID と資格情報を検証できない場合や、ユーザー ID が無効な場合、WBEM サーバーは CIM セキュリティ例外を返します。この例外には、NO_SUCH_PRINCIPAL エラーが含まれます。WBEM サーバーがユーザーの ID と資格情報を検証できない場合や、ユーザー ID に対するパスワードが無効な場合、WBEM サーバーは CIM セキュリティ例外を返します。この例外には、INVALID_CREDENTIAL エラーが含まれます。

WBEM サーバーが Solaris の役割 ID を検証できない場合、WBEM サーバーは、NO_SUCH_ROLE エラーの入った CIM セキュリティ例外を返します。

引き受けた役割 ID に対して役割のパスワードが無効な場合、WBEM サーバーは、INVALID_CREDENTIAL エラーの入った CIM セキュリティ例外を返します。

役割 ID とパスワードが有効でも、そのユーザーが役割を引き受けることを許可されていなければ、WBEM サーバーは、 CANNOT_ASSUME_ROLE エラーの入った CIM セキュリティ例外を返します。

これらの CIM セキュリティ例外については、次の表で詳しく説明します。

エラー	考えられる原因	解決法
NO_SUCH_PRINCIPAL	指定されたユーザー ID が WBEM サーバー上の Solaris オペレーティング環境で無効であるか、そのユーザーのユーザーアカウントがパスワードを持っていない、またはロックされている	ユーザー ID が有効であることを確認してください。ユーザー ID が有効であれば、ユーザーは、WBEM サーバーマシン上の Solaris オペレーティング環境にログインできます。場合によっては、ネームサービステーブルも確認する必要があります。これにより、Solaris WBEM サーバーが、サーバー上に設定されたネームサービスのユーザー ID を使用しているかどうかを判別できます。
INVALID_CREDENTIAL	指定されたユーザーのパスワード、または引き受けられた役割のパスワードが、WBEM サーバー上の Solaris オペレーティング環境のユーザーに対して無効である	ユーザーのパスワードが正しいかどうか確認してください。

エラー	考えられる原因	解決法
NO_SUCH_ROLE	WBEM サーバーに対する認証で使用された役割 ID が、WBEM サーバー上の Solaris オペレーティング環境で有効な RBAC 役割ではない	役割 ID は、サーバー上の passwd テーブルでは有効ですが、この ID でサーバーにログインすることはできません。 Solaris ソフトウェアは、役割 ID に直接ログインすることを許可しません。passwd テーブルで役割 ID を確認し、user_attr テーブルでその役割がユーザーの役割の種類として定義されていることを確認してください。user_attr テーブル内の役割 ID には、*type=role* という構文の属性が含まれています。 Solaris 管理コンソールのユーザーツールを使って、有効なユーザーまたは有効な役割 ID を確認することもできます。ユーザーアカウントツールではユーザーの確認、管理役割ツールでは役割の確認が可能です。ただし、ユーザーツールを使用するときは、CIMOM サーバー上のテーブルの正しいソースを把握していなければなりません。なぜなら、CIMOM サーバーが NIS などのネームサービスを使用している場合は、そのネームサービスのマスターサーバーにアクセスしなければならないからです。
CANNOT_ASSUME_ROLE	役割 ID は有効だが、認証交換で指定されたユーザー ID は、その役割を引き受けるように構成されていない	Solaris 管理コンソールユーザーツールコレクションに含まれている管理役割ツールを使って、ユーザーに役割を明示的に割り当てます。管理役割ツールについては、『*Solaris* のシステム管理 (セキュリティサービス)』の「役割プロパティの変更」を参照してください。

そのほかの CIM セキュリティ例外エラーが起きる場合

WBEM サーバーは、その他のエラーインジケーションを CIM セキュリティ例外に入れて返すことがあります。しかし、これらのインジケーションの多くは、認証交換におけるシステム障害を示すものです。認証交換のセキュリティオプションについて、WBEM クライアントの構成が WBEM サーバーの構成と互換性がない可能性が考えられます。

これらのエラーインジケーションが発生した場合は、クライアントマシン上の WBEM インストールで、`WbemClient.properties` にセキュリティの適切な構成プロパティ値が入っているかどうか確認します。このファイルは通常、WBEM のインストールディレクトリ内のベンダー拡張サブディレクトリ `/usr/sadm/lib/wbem/extension` にあります。

さらに、クライアントアプリケーションの `CLASSPATH` に `sunwbem.jar` と拡張ディレクトリのパス名が含まれていることを確認します。

承認検査に失敗した場合

クライアントが、WBEM サーバーへの要求に関連付けられたデータのアクセスまたは変更を許可されていない場合、WBEM サーバーは CIM セキュリティ例外を返します。この例外には、`ACCESS_DENIED` エラーが含まれます。

`ACCESS_DENIED` エラーは、ユーザーまたは役割が、要求によって管理されているデータへのアクセスを許可されていないため、その要求を完了できなかったことを示します。

WBEM ログのセキュリティメッセージで、失敗した要求を確認してください。ログデータの表示方法については、151 ページの「ログビューアを使ってログデータを調べる」を参照してください。承認障害のログメッセージは、「概要 (Summary)」列に「アクセスを拒否しました (`Access denied`)」 と示されています。「ユーザー (User)」列には、検査に使用された認証されたユーザーまたは役割の名前が表示されます。「ソース (Source)」列には、検査を実行しているプロバイダの名前が表示されます。この列のプロバイダ名は、プロバイダ実装クラスの名前ではなく、ユーザーにわかりやすいプロバイダ名です。

詳細なメッセージの中に、検査されているアクセス権の名前や、ユーザーまたは役割に許可されていなかったアクセス権の名前が含まれています。

アクセス権が *namespace*: *right* という形式で表示されている場合、承認検査は名前空間 ACL を使用したことを示しています。認証されたユーザーは、その名前空間に対してそのアクセス権 (読み取りまたは書き込み) を許可されていません。

Sun WBEM ユーザーマネージャ (`wbemadmin`) を使用して、ユーザーに適切なアクセス権を許可してください。Sun WBEM ユーザーマネージャについては、137 ページの「Sun WBEM ユーザーマネージャを使ってアクセス制御を設定する」を参照してください。

アクセス権が solaris.*application .right* という形式で表示されている場合、承認検査 は RBAC 承認を使用したことを示します。

Solaris 管理コンソールユーザーツールコレクションに含まれている管理役割ツールを使って、ユーザーまたは役割に必要なアクセス権を許可してください。この手順については、『*Solaris* のシステム管理 (セキュリティサービス)』の「役割プロパティの変更」を参照してください。

第 9 章

問題発生時の解決方法

この章の内容は次のとおりです。

- ログデータを表示する方法
- WBEM エラーメッセージを読み取る方法
- エラーメッセージの一覧

ログビューアを使ってログデータを調べる

WBEM ロギングサービスは、アプリケーション開発者やプロバイダの作成者がログファイルにログメッセージを書き込むことを可能にします。WBEM ログファイルで、管理サブシステムから生成されるエラーメッセージ、警告メッセージ、および情報メッセージを追跡します。たとえば、次のような状態の時にログメッセージを書き込みます。

- システムがシリアルポートにアクセスできない場合
- システムが正常にファイルシステムをマウントした場合
- システム上で許可されている数より多くのプロセスが実行されている場合

ログレコードを作成したあとは、Solaris 管理コンソールアプリケーションとログビューアを起動することができます。ログレコードは、Solaris 管理コンソールソフトウェアを起動した時に自動的に作成されます。

ログレコードの詳細のすべてを Solaris 管理コンソールのログビューアで見ることができます。

▼ Solaris 管理コンソールアプリケーションとログビューアを起動する方法

1. **Solaris** 管理コンソールを起動するには、次のコマンドを入力します。

   ```
   $ smc
   ```

2. 「ナビゲーション **(Navigation)**」パネルで、「このコンピュータ **(This Computer)**」をダブルクリックするか、その横にある展開 **/** 圧縮のアイコンをクリックします。

   「このコンピュータ (This Computer)」の下に、コマンドのツリーが表示されます。

3. 「システムステータス **(System Status)**」をダブルクリックします。

   「ログビューア (Log Viewer)」アイコンが表示されます。

4. 「ログビューア **(Log Viewer)**」アイコンをクリックします。

   ログビューアが起動します。

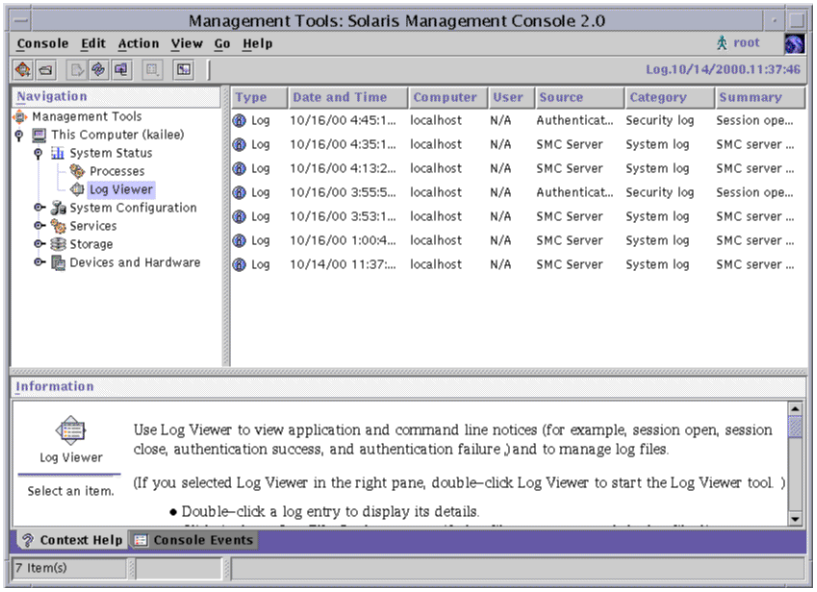

図 **9–1** Solaris 管理コンソールアプリケーションで「ログビューア (Log Viewer)」を選択した状態

WBEM のエラーメッセージについて

CIM オブジェクトマネージャ (CIMOM) によって、 MOF (Managed Object Format) 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 **9–1** 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。

```
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

WBEM のエラーメッセージ

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。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: インスタンスを作成しようとしましたが、指定したクラスは abstract クラスであり、abstract クラスはインスタンスを持てません。

対処方法: concrete クラスのインスタンスを作成してください。

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、CIMOM が無効なチェックサムを受信した場合に表示されます。ネットワーク経由で渡されたデータパケット内のビット数が、チェックサムになります。データの送信者と受信者は、この数値を使って、データが送信時に損傷を受けたり改ざんされたりしていないことを確認します。この数値は、データ通信がセキュリティ保護されていることを確認する目的でも使用されます。

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 サービスのセキュリティ機能を使ってメッセージを再送信してください。Solaris WBEM サービスのセキュリティ機能については、第 8 章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、処理を実行するために必要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: システム管理者または CIMOM の管理者に、処理を実行する特権を要求してください。

CIM_ERR_ALREADY_EXISTS

例 1: CIM_ERR_ALREADY_EXISTS

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と同じ名前のクラスがすでに存在しています。

対処方法: CIM Workshop で既存のクラスを検索して、使用中のクラス名を確認します。そしてクラス名が一意になるようにクラスを作成します。

例 2: CIM_ERR_ALREADY_EXISTS

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と同じ名前のインスタンスがすでに存在しています。

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と同じ名前の名前空間がすでに存在しています。

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、このパラメータは重複した修飾子型の名前に置換されています。

原因: 指定されたプロパティに、作成しようとした修飾子型と同じ名前の修飾子型がすでに存在しています。

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、一意の名前で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、CIMOM 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: インスタンスを持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条件と、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: これは、さまざまなエラー条件で表示される汎用エラーメッセージです。

対処方法: 解決方法はエラー条件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} は、照会の無効部分に置換されます。
- {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: 例外の詳細情報に従ってエラーを修正します。さらに、照会文字列と照会言語が一致することを確認します。

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。そのスーパークラスとサブクラスが変更された可

能性もあります。つまり、指定されたスーパークラスが、実際には指定されたクラスのサブクラスになっている可能性があります。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: CIMOM 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の名前に置換されます。
- {1} は、指定されたクラスの名前に置換されます。

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: 名前空間名を再入力してください。名前空間を正しく入力したかどうか、スペルを確認してください。

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。プロパティとして参照を持つことができるのは、関連クラスだけです。

対処方法: 結合修飾子を追加するか、参照を削除します。

INVALID_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 に次のディレクトリおよびファイルが含まれていることを確認してください。
  /usr/sadm/lib/wbem/extension:/usr/sadm/lib/wbem/sunwbem.jar

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_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オーバーライドする concrete クラス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。オーバーライド済みの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 CIMOM 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: 指定されたホスト上で CIMOM が実行されていません。

対処方法: 接続しようとするホストで CIMOM が動作していることを確認します。そのホストで CIMOM が動作していない場合は、CIMOM 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: CIMOM の CLASSPATH 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。CIMOM 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_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、CIMOM の CLASSPATH に、次の項目のうち 1 個以上が欠けていることを示します。

- 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: CIMOM の CLASSPATH 環境変数を設定します。CIMOM 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、CIMOM の CLASSPATH に、次の項目のうち 1 個以上が欠けていることを示します。

- 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: CIMOM の CLASSPATH を設定します。CIMOM 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をオーバーライドしようとしています。しかし、オーバーライドしようとしているメソッドが定義されておらず、クラス階層内に存在しません。

メソッドをオーバーライドすると、その実装とシグニチャー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をオーバーライドしようとしています。しかし、メソッドが定義されていないため、オーバーライドしようとしているプロパティがクラス階層内に存在しません。

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、CIMOM の CLASSPATH に、次の項目のうち 1 個以上が欠けていることを示します。

- 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: CIMOM の CLASSPATH を設定します。接続しようとするホストで CIMOM が動作していることを確認します。そのホストで CIMOM が動作していない場合は、CIMOM 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です。

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 仕様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、指定されたクラスの名前に置換されます。

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のスペルが正しいことを確認します。

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。この修飾子は、CIM スキーマの一部か、拡張スキーマの一部として定義する必要があります。この定義がないと、特定のクラスのプロパティまたはメソッドの有効な修飾子として認識されません。

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 CIM 修飾子と CIM スキーマでの修飾子の使用方法については、 `http://www.dmtf.org` の DMTF CIM 仕様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。この修飾子は、CIM スキーマの一部か、拡張スキーマの一部として定義する必要があります。この定義がないと、特定のクラスのプロパティまたはメソッドの有効な修飾子として認識されません。

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 CIM 修飾子と CIM スキーマでの修飾子の使用方法については、 `http://www.dmtf.org` の DMTF CIM 仕様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 以外が指定されました。

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: すでに終了したセッションが引き続き使用されています。

対処方法: セッションを閉じないでください。

`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: プロバイダが正しくプロバイダ実装を登録していることを確認します。

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 (CIMOM 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 サービスのセキュリティ機能については、第 8 章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 インタフェースを指定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、呼び出されます。
- {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 class <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検出に使用されます。
- {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、コマンド `public class <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、CIMOM 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 サービスのセキュリティ機能については、第 8 章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の名前に置換されます。
- `{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。オーバーライド済みの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。たとえば、メソッドを変更するために [READ] 修飾子が指定された場合、 SCOPE_ERROR メッセージが返されます。

注 – CIM (Common Information Model) 仕様は、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修飾子に指示するスコープを持ちます。多くの修飾子は、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修飾子に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 仕様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が指定されたホストが使用できないか、または見つかりません。このメッセージは、次のいずれかの条件で表示されます。

- ホスト名のスペルが正しくない
- ホストコンピュータがほかのドメインに移された
- ホスト名がこのドメインに登録されていない
- システム状況により、ホストが一時的に使用できない

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を確認します。ホストのシステム状況を確認します。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 CIMOM のバージョン番号に置換されています。

原因: CIMOM 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 A

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 オブジェクトマネージャで使用できます。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。`/usr/sadm/mof/` 内の MOF ファイルは、システムのインストールクラスタと、インストールされているパッケージによって異なります。

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。

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 **A–1** Solaris スキーマファイル

Solaris スキーマファイル	説明
Solaris_Acl.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_Application.mof	Solaris パッケージおよびパッチを CIM でモデル化する
Solaris_CIMOM.mof	CIM オブジェクトマネージャの構成情報が含まれる
Solaris_Core.mof	コンピュータシステムの情報や統計情報を格納したコアクラスのクラス定義が含まれる
Solaris_Device.mof	CIM オブジェクトマネージャ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_Event.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は、管理クライアントへのインジケーションの配信を円滑に行う。この配信に使用されるプロトコルは、Sun Microsystems の CIM RMI (Remote Method Invocation) の実装である
Solaris_Network.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_Performance.mof	個々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用途やパフォーマンスに関連するクラスを定義する
Solaris_Project.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_Schema.mof	Solaris スキーマのすべての MOF ファイルを一覧表示し、MOF ファイルの読み取りとコンパイルの順番を指定する

表 A–1 Solaris スキーマファイル　(続き)

Solaris スキーマファイル	説明
Solaris_SNMP.mof	SNMP プロバイダの構成に使用するクラスと、SNMP プロバイダとその他のシステム上の SNMP エージェントの通信の構成に使用するクラスを含む
Solaris_System.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_Users.mof	ユーザーアカウントを使用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
WBEMServices.mof	クライアントとプロバイダの両方について、CIM オブジェクトマネージャとそのプロトコルアダプタのクラスパスを構成するクラスを含む

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_Acl.mof ファイル

Solaris_Acl.mof ファイルは、Solaris WBEM サービス内のセキュリティクラスを指定します。このファイルでは、アクセス制御リスト (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_Application.mof ファイル

Solaris_Application.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_Application.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_Application.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_CIMOM.mof ファイル

Solaris_CIMOM.mof ファイルには、CIM オブジェクトマネージャ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_CIMOM.mof ファイルでは、次のクラスが定義されます。

- CIM_ObjectManager
- CIM_ObjectManagerCommunicationMechanism
- CIM_WBEMCommunicationMechanism
- Solaris_CIMOM
- Solaris_ObjectManagerClientProtocolAdapter
- Solaris_ObjectManagerProtocolAdapter
- Solaris_ObjectManagerProviderProtocolAdapter
- Solaris_ProviderPath

さらに、Solaris_CIMOM.mof ファイルでは、関連クラスの CIM_CommMechanismForManager が定義されます。

Solaris_Core.mof ファイル

Solaris_Core.mof ファイルは、Solaris_Schema.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_Core.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_Core.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_Device.mof ファイル

Solaris_Device.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_Device.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_Event.mof ファイル

Solaris_Event.mof ファイルには、Solaris プラットフォーム固有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。これらの Solaris インジケーションハンドラは、CIM_IndicationHandler のサブクラスです。これらのサブクラスには、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_Event.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_Network.mof ファイル

Solaris_Network.mof ファイルでは、ネットワークドメイン、IP サブネット、ネーミングサービス関連のクラスが定義されます。これらのネーミングサービスには、NIS、NIS+、LDAP、DNS、およびサーバーの /etc ファイルが含まれます。Solaris_Network.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_Performance.mof ファイル

Solaris_Performance.mof ファイルでは、コンピューティングリソースの測定値関連のクラスが定義されます。これらのクラスは、個々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用途とパフォーマンスに関連しています。Solaris_Performance.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_Performance.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

Solaris_Project.mof ファイル

Solaris_Project.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_Project.mof ファイルでは、Solaris_Project クラスが定義されます。また、関連クラス Solaris_ProjectGroup および Solaris_ProjectUser が定義されます。

Solaris_Schema.mof ファイル

Solaris_Schema.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。このファイルには、MOF ファイルがコンパイルされる順番で一覧表示されます。

コンパイルのたびに Java クラスが生成され、CIMOM に送信されます。CIMOM では、クラスはイベントとして定義されるか、オブジェクトとして CIM オブジェクトマネージャリポジトリに送信され、保存されます。次の Solaris_Schema.mof ファイルリストには、Include 文がコンパイルされる順番で一覧表示されています。

```
/*
Solaris Schema
Copyright   (c) 2002 Sun Microsystems, Inc. All Rights Reserved.
*/
#pragma Include ("Solaris_Core.mof")
#pragma Include ("Solaris_Application.mof")
#pragma Include ("Solaris_System.mof")
#pragma Include ("Solaris_Device.mof")
#pragma Include ("Solaris_Network.mof")
#pragma Include ("Solaris_Users.mof")
#pragma Include ("Solaris_Project.mof")
#pragma Include ("Solaris_Event.mof")
#pragma Include ("Solaris_CIMOM.mof")
#pragma Include ("Solaris_SNMP.mof")

// 最後のインクルード。ここで CIM 名前空間を変更する
#pragma Include ("Solaris_Acl.mof")
```

コンパイラは、Solaris_Schema.mof ファイルの行の構文解析を行い、Include 文に指定されたファイルをコンパイルします。その後、Solaris_Schema.mof ファイルの中の次の行の構文解析を行います。インクルードされたすべてのファイルのコンパイルが完了するまで、このプロセスが繰り返されます。

Solaris_SNMP.mof ファイル

Solaris_SNMP.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_SNMP.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_System.mof ファイル

Solaris_System.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_System.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_Users.mof ファイル

Solaris_Users.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_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

また 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_LocalFSResidesOnExtent
- 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

WBEMServices.mof ファイル

WBEMServices.mof ファイルには、クライアントとプロバイダの両方について、CIM オブジェクトマネージャとそのプロトコルアダプタのクラスパスを構成するクラスが含まれます。WBEMServices.mof ファイルでは、次のクラスが定義されます。

- WBEMServices_CIMXMLObjectManagerClientProtocolAdapter
- WBEMServices_Classpath
- WBEMServices_ClientProtocolAdapterForManager
- WBEMServices_ObjectManager
- WBEMServices_ObjectManagerClientProtocolAdapter
- WBEMServices_ObjectManagerProtocolAdapter
- WBEMServices_ObjectManagerProviderProtocolAdapter
- WBEMServices_ProtocolAdapterForManager
- WBEMServices_ProviderProtocolAdapterForManager
- WBEMServices_RMIObjectManagerClientProtocolAdapter

索引

W

あ

い

え

き

く

こ

し

す

せ