Sun Java Enterprise System 5 監視ガイド (UNIX 版)

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

Part No: 820–1590
2007 年 3 月

070830@18135

目次

はじめに

このマニュアルでは、Sun Java™ Enterprise System 5 (Java ES) の新しい監視機能について説明します。監視は Sun Java System Monitoring Framework 2.0 および Sun Java System Monitoring Console 1.0 によって実装されます。

このガイドで説明する手順では、インストールした各コンポーネントに対して Monitoring Framework を設定および有効化する方法と、すべての監視対象データを Monitoring Console で表示する方法を示します。このガイドでは、フレームワークに含まれない個別コンポーネントのログファイルやその他の監視機構については説明しません。

対象読者

このガイドは、次の対象読者向けに記述されています。

- Java ES 配備の保守計画を設計する必要があるソフトウェアアーキテクト。
- Java ES のインストールおよび設定を実行するシステム管理者。
- Java ES 配備を監視および保守するシステム管理者および技術者。

このマニュアルをお読みになる前に

次の節で説明する Java ES マニュアルセットのドキュメントの内容を理解しておくことをお勧めします。また、監視する Java ES コンポーネントの設計および機能についても理解しておくことをお勧めします。

さらに、監視コンポーネントをインストールおよび設定する場合は、まず、その他すべてのコンポーネントのインストールを完了する必要があります。インストールまたは設定を実行する前に、『Sun Java Enterprise System 5 リリースノート (UNIX 版)』を参照することをお勧めします。

Java ES ドキュメントセット

Java ES ドキュメントセットには、配備計画およびシステムインストールに関する情報が記載されています。システムマニュアルの URL は http://docs.sun.com/coll/1286.2 です。Java ES の概要を理解するため、次の表に紹介されているマニュアルを、記載されている順番に参照してください。

表 P–1　Java Enterprise System のマニュアル

マニュアルタイトル	内容
『Sun Java Enterprise System 5 リリースノート (UNIX 版)』 『Sun Java Enterprise System 5 リリースノート (Windows 版)』	既知の問題など、Java ES に関する最新の情報が記載されています。さらに、各コンポーネントのリリースノートが Release Notes Collection (http://docs.sun.com/coll/1315.2) にリストされています。
『Sun Java Enterprise System 5 技術の概要』	Java ES の技術的および概念的な基礎について説明します。コンポーネント、アーキテクチャー、プロセス、および機能について説明しています。
『Sun Java Enterprise System Deployment Planning Guide』	Java ES に基づく企業配備ソリューションの計画および設計に関する情報を提供します。配備の計画および設計に関する基本的概念と原則を示し、ソリューションのライフサイクルについて説明し、Java ES に基づくソリューションを計画する際に使用する高度な例と戦略を提供します。
『Sun Java Enterprise System 5 インストール計画ガイド』	Java ES の配備に関し、ハードウェア、オペレーティングシステム、およびネットワーク面の実装仕様の開発に役立つ情報を提供します。インストールおよび設定計画を遂行する上で注意すべきコンポーネントの依存関係などの問題について説明します。
『Sun Java Enterprise System 5 インストールガイド (UNIX 版)』 『Sun Java Enterprise System 5 インストールガイド (Windows 版)』	Java ES のインストールプロセスを段階的に説明します。また、インストール後にコンポーネントを設定する方法、および設定したコンポーネントが正常に機能するかどうかを確認する方法についても説明します。
『Sun Java Enterprise System 5 インストールリファレンス (UNIX 版)』	設定パラメータについての補足情報と、設定の計画時に使用するワークシートを提供します。また、Solaris オペレーティングシステムまたは Linux オペレーティング環境でのデフォルトディレクトリやポート番号などの参照情報を示します。
『Sun Java Enterprise System 5 アップグレードガイド (UNIX 版)』 『Sun Java Enterprise System 5 Upgrade Guide for Microsoft Windows 』	以前にインストールしたバージョンから Java ES 5 にアップグレードする手順を示します。

表 P–1 Java Enterprise System のマニュアル *(続き)*

マニュアルタイトル	内容
『Sun Java Enterprise System 5 監視ガイド (UNIX 版)』	各製品コンポーネントに対して Monitoring Framework を設定する手順と、Monitoring Console を使用してリアルタイムデータを参照したり、監視ルールを作成したりする方法を示します。
『Sun Java Enterprise System 用語集』	Java ES ドキュメントで使用される用語について説明します。

デフォルトのパスとファイル名

次の表は、監視機能を実装する Java ES コンポーネントのデフォルトのパスおよびファイル名についての説明です。

表 P–2 デフォルトのパスとファイル名

プレースホルダ	説明	デフォルト値
mfwk-base	Monitoring Framework の共有コンポーネントが自動的にインストールされるディレクトリを表します。このパスは、設定ディレクトリの一部としても使用されます。	Solaris システム: `/opt/SUNWmfwk` Linux システム: `/opt/sun/mfwk`
MConsole-base	Monitoring Console に対して選択されたインストールディレクトリを表します。	Solaris システム: `/opt/SUNWjesmc` Linux システム: `/opt/sun/jesmc`
WebConsole-base	Web コンソールの共有コンポーネントが自動的にインストールされるディレクトリを表します。	Solaris システム: `/etc/webconsole/console` Linux システム: `/etc/opt/webconsole/console`
AccessMgr-base	Sun Java System Access Manager に対して選択されたインストールディレクトリを表します。	Solaris システム: `/opt/SUNWam` Linux システム: `/opt/sun/identity`
AppServer-base	Sun Java System Application Server に対して選択されたインストールディレクトリを表します。	Solaris システム: `/opt/SUNWappserver/appserver` Linux システム: `/opt/sun/appserver`
CalServ-base	Sun Java System Calendar Server に対して選択されたインストールディレクトリを表します。	Solaris システム: `/opt/SUNWics5` Linux システム: `/opt/sun/calendar`

表 P–2 デフォルトのパスとファイル名 (続き)

プレースホルダ	説明	デフォルト値
DirServ-base	Sun Java System Directory Server に対して選択されたインストールディレクトリを表します。	Solaris システム: `/opt/SUNWdsee/ds6` Linux システム: `/opt/sun/ds6`
IM-base	Sun Java System Instant Messaging に対して選択されたインストールディレクトリを表します。	Solaris システム: `/opt/SUNWiim` Linux システム: `/opt/sun/im`
MsgServ-base	Sun Java System Messaging Server に対して選択されたインストールディレクトリを表します。	Solaris システム: `/opt/SUNWmsgsr` Linux システム: `/opt/sun/messaging`
Portal-base	Sun Java System Portal Server に対して選択されたインストールディレクトリを表します。	Solaris システム: `/opt/SUNWportal` Linux システム: `/opt/sun/portal`
WebServer-base	Sun Java System Web Server に対して選択されたインストールディレクトリを表します。	Solaris システム: `/opt/SUNWwbsvr7` Linux システム: `/opt/sun/webserver`

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–3 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。 `machine_name% you have mail.`
`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。	`machine_name%` **`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。	『コードマネージャー・ユーザーズガイド』を参照してください。

表 P–3 表記上の規則 (続き)

字体または記号	意味	例
「」	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。

コマンド例のシェルプロンプト

次の表は、デフォルトのシステムプロンプトとスーパーユーザープロンプトを示しています。

表 P–4 シェルプロンプト

シェル	プロンプト
UNIX および Linux システムの C シェル	`machine_name%`
UNIX および Linux システムの C シェルのスーパーユーザー	`machine_name#`
UNIX および Linux システムの Bourne シェルおよび Korn シェル	`$`
UNIX および Linux システムの Bourne シェルおよび Korn シェルのスーパーユーザー	`#`
Microsoft Windows のコマンド行	`C:\`

記号の規則

次の表は、この用語集で使用される記号の一覧です。

表 P–5 記号の規則

記号	説明	例	意味
`[ ]`	省略可能な引数やコマンドオプションが含まれます。	`ls [-l]`	`-l` オプションは必須ではありません。
`{ | }`	必須のコマンドオプションの選択肢を囲みます。	`-d {y|n}`	`-d` オプションには `y` 引数か `n` 引数のいずれかを使用する必要があります。
`${ }`	変数参照を示します。	`${com.sun.javaRoot}`	`com.sun.javaRoot` 変数の値を参照します。
-	同時に押すキーを示します。	Control-A	Control キーを押しながら A キーを押します。
+	順番に押すキーを示します。	Ctrl+A+N	Control キーを押してから放し、それに続くキーを押します。
→	グラフィカルユーザーインタフェースでのメニュー項目の選択順序を示します。	「ファイル」→「新規」→「テンプレート」	「ファイル」メニューから「新規」を選択します。「新規」サブメニューから「テンプレート」を選択します。

マニュアル、サポート、およびトレーニング

Sun のサービス	URL	内容
マニュアル	http://jp.sun.com/documentation/	PDF 文書および HTML 文書をダウンロードできます。
サポートおよびトレーニング	http://jp.sun.com/supportraining/	技術サポート、パッチのダウンロード、および Sun のトレーニングコース情報を提供します。

◆ ◆ ◆ 第 1 章

1 Java ES の監視機能の概要

このガイドでは、Sun Java™ Enterprise System (Java ES) の Monitoring Framework 2.0 および Monitoring Console 1.0 コンポーネントについて説明します。これらのコンポーネントは連携して、リリース 5 で導入された新しい監視機能を実装します。

このガイドで説明する手順では、インストールした各コンポーネントに対して Monitoring Framework を有効化する方法と、すべての監視対象データを Monitoring Console で表示する方法を示します。このガイドでは、個別のコンポーネントがフレームワークの外部で実装するログファイル、エラーメッセージ、およびその他の監視機構については説明しません。Monitoring Framework と Monitoring Console のどちらも、監視対象コンポーネントの管理機能は提供しません。コンポーネントの管理については、各製品の個別のマニュアルを参照してください。

この章では、監視の基本的な概念および Monitoring Framework のアーキテクチャーについて説明します。

この章の内容は次のとおりです。

Monitoring Framework および Monitoring Console コンポーネント

Sun Java System Monitoring Framework は、コンポーネントをインストゥルメントしたり、コンポーネントの属性を監視対象として公開したりするためのインフラストラクチャーを提供します。業界標準の CIM (Common Information Model) 仕様に基づいて、*Common Monitoring Model (CMM)* と呼ばれる監視対象オブジェクトの階層が定義されます。各製品コンポーネントは、そのコンポーネントの監視可能な属性を表すオブジェクトを公開します。ノードエージェントは、1 つのホスト上の複数コンポー

ネントのビューを集約します。Monitoring Framework は、運用統計を収集したり、ユーザー定義のしきい値に基づくアラームを定義したりするための機構も提供します。

Sun Java System Monitoring Console は、Java ES コンポーネントを監視するためのグラフィカルインタフェースです。Monitoring Console には、Java ES 配備内のすべてのノードエージェントに接続するマスターエージェントが含まれます。Monitoring Console は Sun Java System Web コンソールに依存する Web ベースのアプリケーションであり、HTTP 経由であらゆる場所からのアクセスが可能です。Monitoring Console のメイン画面には、生成されたアラームなど、すべての有効化されたコンポーネントのステータスの要約が表示されます。この画面から、各コンポーネント内の監視対象オブジェクトの階層にアクセスしたり、すべての監視対象属性の詳細なステータスやリアルタイムの値を確認したりできます。Monitoring Console のインタフェースでは、任意のアラームの詳細を表示して確認したり、任意の属性に基づいて新しい監視ルールを作成したりできます。

Java ES の監視のしくみ

監視とは、実行時データを収集して公開し、サービス基準の品質を計測することにより、システム管理者がパフォーマンスを評価したり、アラームの通知を受けたりできるようにするプロセス全般のことです。実行時の運用中、管理者はパフォーマンス統計の表示、自動監視のためのルールの作成、アラームの確認などの作業を、すべて Monitoring Console との対話操作により実行することができます。ただし、Monitoring Framework のアーキテクチャーや、Monitoring Framework が Monitoring Console に接続するしくみを理解しておくと、設定、トラブルシューティング、および高度な監視を実行する際に役立ちます。

Java ES での監視は、次の概念に基づいて行われます。

- Common Monitoring Model (CMM) は、比較可能な属性に関して、すべての Java ES コンポーネントが同じオブジェクトや値を公開することを保証します。
- CMM インタフェースによって定義される Java オブジェクトは、製品コンポーネントの標準化されたインストゥルメンテーションを提供します。
- ノードエージェントは、システムにインストールされたすべてのコンポーネントのすべての監視対象オブジェクトを公開し、それらのオブジェクトの統計、ルール、およびアラームを管理します。
- 独立したホスト上のマスターエージェントは、すべてのノードエージェントが管理するすべての監視対象オブジェクトを集約し、Monitoring Console にデータを提供します。

次の節では、監視アーキテクチャーのこれらの各概念についてさらに詳しく説明します。

Common Monitoring Model (CMM)

標準化された監視機構の基本は、監視されるオブジェクトを定義し、すべての監視対象コンポーネントを横断してこれらのオブジェクトを採用することです。監視アーキテクチャーではこの目的のために、Common Monitoring Model (CMM) を定義しています。CMM は、Distributed Management Task Force (DMTF) が管理する Common Information Model (CIM) の拡張です。CMM は、コンピュータやアプリケーションなどの監視対象オブジェクトの仕様を定める情報モデルであると同時に、運用ステータス値などの統一された値の仕様を定めるデータモデルでもあります。CMM では情報モデルの一部として、オブジェクトの属性も定義します。これには、サービスによって処理される要求の数や、あるサービスは特定のコンピュータ上でホストされる、といったオブジェクト間の関係などがあります。

CMM により、配下の実装が異なる場合も含めて、アプリケーション、サービス、アクセスポイントなどの概念がすべての製品コンポーネント間で共通化されます。たとえば、Web Server は HTTP 要求を処理するサービスを公開し、Directory Server は LDAP 要求を処理するサービスを公開します。CMM で定義される標準オブジェクトは、これら 2 つの機能に共通する要素 (処理される要求数を計測する能力、特定の期間中に要求に応答する平均時間など) を捕捉します。

また、値の意味がシステム全体を通して一貫するように、特定のデータ値が標準化されます。たとえば、監視対象の製品コンポーネントに関係なく、`DEGRADED` という運用ステータスは常に、「サービスはまだ利用できるがパフォーマンスが著しく低下している」ことを意味します。

CMM 仕様は、インストゥルメンテーションに使用される Java のインタフェースおよびクラスに組み込まれています。これらのインタフェースやクラスについては付録 A で説明しています。

CMM インストゥルメンテーション

Monitoring Framework で、インストゥルメンテーションとは、CMM 定義を実装する Java のインタフェースおよびクラスの集合のことです。Java ES の新しい監視機能のために、製品コンポーネントのコードがインストゥルメントされています。その目的は、CMM オブジェクトをインスタンス化し、監視対象オブジェクトの属性を通して実行時の値を公開できるようにすることです。各コンポーネントによって実装される CMM オブジェクトは、監視可能な要素を決定します。またこの理由のため、一部のコンポーネントはほかのコンポーネントよりも公開する属性が少なくなります。付録 B では、各製品コンポーネントによって監視対象として公開されるオブジェクトおよび属性の一覧を示しています。

ノードエージェント

監視に関する用語で、ノードとは、一意な完全指定ドメイン名または IP アドレスによって識別される、単一の論理ホストのことです。システム全体、または仮想システムとして構成された Solaris ゾーンのどちらかがノードとして機能できます。ノードエージェントは、配備先ホスト上のすべてのインストゥルメント済みコンポーネントと通信し、それらのコンポーネントの監視対象オブジェクトをすべて公開します。またノードエージェントは、エージェントに含まれる監視対象オブジェクトに関して、パフォーマンス統計を収集したり、ルールで定義されたしきい値を監視したり、アラームを生成したりするためのすべてのロジックを管理します。

次の図は、1 つのホスト上のノードエージェントの内容を表しています。このホストには、3 つの Java ES 製品コンポーネントのインスタンスが存在します。またこの図では、製品コンポーネントによって提供される値を公開するために、ノードエージェント内でインストゥルメンテーションがインスタンス化されるしくみも示しています。

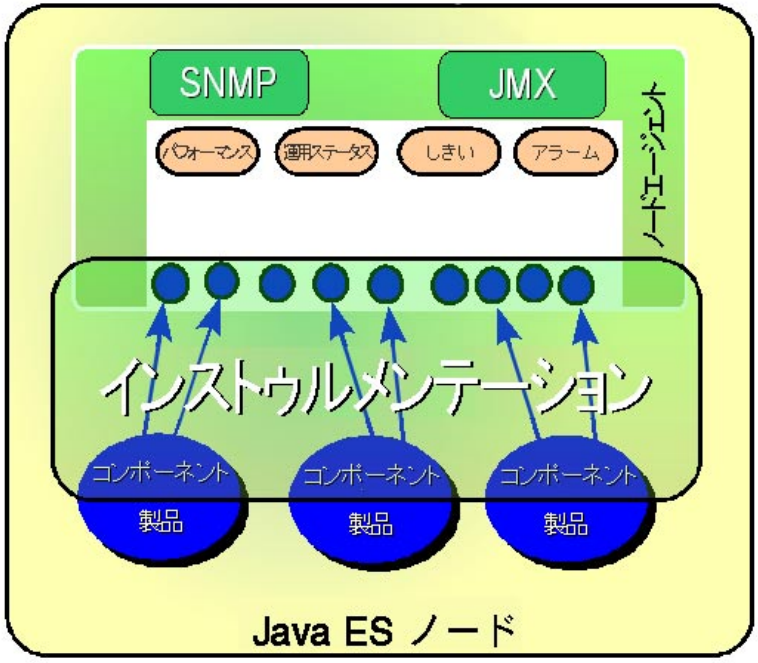


図 1–1　ノードエージェントの図

ノードエージェントは、それ自体が Java 仮想マシンである共通エージェントコンテナにロードされるモジュールとして実装されます。ノードエージェントの実装は、監視およびリモート管理のための標準 Java 拡張機能である Java Management Extensions (JMX) に基づいています。CMM を認識するすべての JMX 対応監視アプリ

ケーションが、ノードエージェント内の監視対象オブジェクトにアクセスできます。ノードエージェントは JMX の機能を利用して、特定の監視対象オブジェクトを Simple Network Monitoring Protocol (SNMP) 経由で公開することもできます。

マスターエージェント

マスターエージェントは、Monitoring Console インストールの一部として独立したマシン上に配備されます。マスターエージェントには、すべてのノードエージェント内の監視対象オブジェクトを集約できるように、すべてのノードの名前またはアドレスが設定されます。マスターエージェントも JMX に基づいています。マスターエージェントは JMX を利用してノードエージェントと通信し、そのローカル共通エージェントコンテナにロードされます。

次の図は、2 つのノードに接続されたマスターエージェントを表します。Monitoring Console はマスターエージェントに接続して、各ノード上の 3 つのコンポーネントを監視します。マスターエージェントは SNMP 属性を集約しないため、監視のために SNMP を使用する場合、各ノードに個別に接続する必要があります。マスターエージェントは Monitoring Console のみとの組み合わせで使用することを前提に設計されており、ほかの監視アプリケーションからのアクセスはできません。

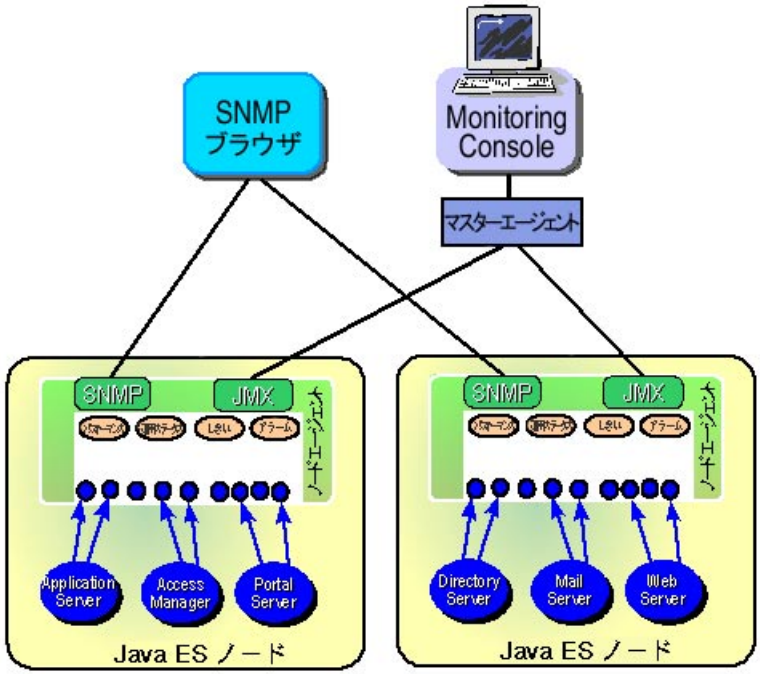


図 1–2 監視アーキテクチャー全体の図

推奨するインストール順序

Java ES の監視機能を評価または配備する場合、次の順序でインストールを実行するのが最も簡単な方法です。

1. 『Sun Java Enterprise System 5 インストールガイド (UNIX 版)』の推奨事項および手順説明に従って、すべてのコンポーネントを配備環境にインストールし、設定します。
2. 第 2 章の説明に従って、すべての監視対象コンポーネントに対して Monitoring Framework を有効化および設定します。
3. 第 3 章の説明に従って、独立したホスト上に Monitoring Console をインストールし、マスターエージェントを起動してから Web サーバーを起動します。これで、すべての監視対象コンポーネントが Monitoring Console から認識可能になり、アクティブに監視されるようになるはずです。

注 – このリリースでのノードエージェントおよびマスターエージェントの非互換性が原因で、Monitoring Console は、ほかの Java ES コンポーネントが存在しないホストにインストールする必要があります。詳細は、64 ページの「Monitoring Console のトラブルシューティング」を参照してください。

監視を有効化したあとで配備済みコンポーネントを変更するときは常に、29 ページの「Monitoring Framework のトラブルシューティング」の説明に従って、マスターエージェントのコンテナと Monitoring Console の Web サーバーを再起動する必要があります。

◆ ◆ ◆ 第 2 章

Monitoring Framework の有効化と設定

16 ページの「Java ES の監視のしくみ」で説明したように、Monitoring Framework は、すべての監視対象コンポーネントに必要なインストゥルメンテーションおよびノードエージェントを提供します。したがって Monitoring Framework は、Java Enterprise System インストーラを使用して監視対象コンポーネントをインストールするたびに自動的にインストールされる共有コンポーネントです。

ただし、監視対象コンポーネントの多くはデフォルトで監視が有効ではなく、一部のコンポーネントについては、追加の設定を行わないとノードエージェントに表示されません。インストールした個々の製品コンポーネントについて、この章で説明する手順を実行してください。

注 – この章で説明する手順を実行する前に、特定のホスト上で実行する予定のすべての製品コンポーネントをインストールおよび設定することをお勧めします。インストールまたは設定を実行する前に、『Sun Java Enterprise System 5 リリースノート (UNIX 版)』を参照することをお勧めします。

これらの手順では、通常は必要でないためまだドキュメント化されていない `mfwksetup` コマンドを使用します。

この章の内容は次のとおりです。

インストール済みディレクトリのレイアウト

共有コンポーネントである Monitoring Framework は、必要になるたびに自動的にインストールされます。各オペレーティングシステムでインストールされるパッケージの名前については、『Sun Java Enterprise System 5 インストールリファレンス (UNIX 版)』の第 5 章「インストール可能なパッケージの一覧」を参照してください。次の表は、Monitoring Framework パッケージ内のディレクトリについての説明です。9 ページの「デフォルトのパスとファイル名」で説明するように、デフォルトのインストールディレクトリ *mfwk-base* には次の意味があります。

- Solaris システム: `/opt/SUNWmfwk`
- Linux システム: `/opt/sun/mfwk`

表 2–1　Monitoring Framework によって使用されるディレクトリ

パス	内容の説明
mfwk-base`/config`	設定ファイルのテンプレート
Solaris システム: *mfwk-base*`/lib` Linux システム: *mfwk-base*`/share/lib`	Java アーカイブ (`.jar`) ファイル
Solaris システム: *mfwk-base*`/lib` Linux システム: *mfwk-base*`/share/lib`	32 bit 実行時ライブラリファイル (`.so`)
Solaris SPARC® システム: *mfwk-base*`/lib/sparcv9` Solaris x86 システム: *mfwk-base*`/amd64` Linux システム: *mfwk-base*`/lib64`	64 ビット実行時ライブラリファイル (`.so`)
mfwk-base`/bin`	公開スクリプトと非公開バイナリ
mfwk-base`/mib`	Monitoring Framework によってサポートされる SNMP MIB のテキストバージョン
mfwk-base`/xml`	(`mfwksetup` コマンドによって配備される) エージェントおよびマスターエージェント用の共通エージェントコンテナ記述子テンプレート
mfwk-base`/dtd`	OSS/J 機能用の DTD ファイル
`/etc/`*mfwk-base*`/config`	セキュリティー関連を含む設定ファイル
`/etc/`*mfwk-base*`/xml`	エージェントおよびサンプル用の共通エージェントコンテナ記述子

表 2–1　Monitoring Framework によって使用されるディレクトリ　(続き)

パス	内容の説明
/var/*mfwk-base*/logs	Monitoring Framework のログファイル
/var/*mfwk-base*/reports	監視ルールレポートのベースディレクトリ
/var/*mfwk-base*/alarms	アラームファイルのリポジトリ

Access Manager での Monitoring Framework の使用

Access Manager ではデフォルトで監視が有効ですが、制限により、監視対象オブジェクトが Monitoring Console に表示されません。

監視可能なオブジェクトおよび属性の一覧については、67 ページの「Access Manager のインストゥルメンテーション」を参照してください。

▼ Access Manager で監視を有効にするには

1 次のコマンドを使用して、**Access Manager** で一時的に監視を無効にします。

```
cacaoadm unregister-module com.sun.cmm.am.xml
cacaoadm restart
```

2 編集のために次の **Access Manager XML** 記述子ファイルを開きます。

```
vi /etc/AccessMgr-base/config/com.sun.cmm.am.xml
```

3 次の内容を含む行を探します。

```
<param-name>Product Name</param-name>
<param-value>Access Manager</param-value>
```

2 行目を次のように修正します。

```
<param-value>Java ES Access Manager</param-value>
```

ファイルを保存し、エディタを終了します。

4 修正した **XML** モジュールを登録します。

```
mfwk-base/bin/mfwksetup -u /etc/AccessMgr-base/config/com.sun.cmm.am.xml
mfwk-base/bin/mfwksetup -r /etc/AccessMgr-base/config/com.sun.cmm.am.xml
```

5 共通エージェントコンテナを再起動します。

```
cacaoadm restart
```

注意事項　サードパーティー製 Web コンテナでの動作が未検証であることが理由で、Access Manager を WebSphere または WebLogic に配備したとき、監視はデフォルトで無効です。58 ページの「監視を選択的に無効化および再有効化するには」の説明に従って監視を有効化できますが、この構成はサポートされていません。

Application Server での Monitoring Framework の使用

監視可能なオブジェクトおよび属性の一覧については、67 ページの「Application Server のインストゥルメンテーション」を参照してください。

▼ Application Server で監視を有効にするには

1 `/var/`*AppServer-base*`/domains/domain1/config/domain.xml` ファイルを編集し、すべての `module-monitoring-level` 設定を `OFF` から `HIGH` に変更します。または、次の処理を行います。

a. `https://`*hostname*`:4849` で、**Application Server** 管理コンソールにログオンします。

b. 「設定」を選択し、「**server-config (Admin Config)**」を選択します。

c. 「監視」の値を `HIGH` に設定します。

d. ほかのすべての値を `HIGH` に設定します。

2 次のコマンドで、**Application Server** を再起動します。

```
cd AppServer-base/appserv/bin
asadmin stop-domain domain1
asadmin start-domain user myUser domain1
```

入力を求められたら、*myUser* のパスワードを入力します。

3 **Application Server** とともに **Portal Server** のインスタンスを配備および監視していた場合、**Application Server** を再起動するプロセスが **Portal Server** の監視に干渉します。**Portal Server** インスタンスが **Monitoring Console** に表示されるようにするには、ブラウザでポータルページを表示する必要があります。たとえば、`portalserv.example.com` というポータルを監視できるようにするには、ポータルページ `http://portalserv.example.com:8080/portal` を読み込みます。

注意事項　制限により、Application Server がクラッシュまたは停止したとき、Application Server の監視対象オブジェクトは Monitoring Framework から削除されます。この状況が発生すると、Application Server は Monitoring Console から表示が消え、監視できなくなります。

Calendar Server での Monitoring Framework の使用

監視可能なオブジェクトおよび属性の一覧については、67 ページの「Calendar Server のインストゥルメンテーション」を参照してください。

▼ Calendar Server で監視を有効にするには

1 ics.conf ファイルを編集します。

```
vi CalServ-base/cal/config/ics.conf
```

2 次の行を追加します。

```
local.mfagent.enable="yes"
```

3 **Calendar Server XML** モジュールを登録します。

```
mfwk-base/bin/mfwksetup -r /opt/SUNWics5/cal/lib/com.sun.cmm.cs.xml
```

4 LD_LIBRARY_PATH 環境変数を次のように設定します。

```
LD_LIBRARY_PATH=mfwk-base/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

5 **Calendar Server** を再起動します。

```
cd CalServ-base/cal/sbin/
./stop-cal
./start-cal
```

6 共通エージェントコンテナを再起動します。

```
cacaoadm restart
```

Directory Server での Monitoring Framework の使用

監視可能なオブジェクトおよび属性の一覧については、68 ページの「Directory Server のインストゥルメンテーション」を参照してください。

▼ Directory Server で監視を有効にするには

1 一時パスワードファイルを作成します。

```
echo –n password > /tmp/pwd
```

2 次のコマンドで、**Monitoring** プラグインを有効にします。

```
DirServ-base/ds6/bin/dscfg enable-plugin -e -p 389 -w /tmp/pwd  "Monitoring Plugin"
```

3 **Directory Server** を再起動します。

```
cd DirServ-base/ds6/bin
./dsadm restart /var/DirServ-base/DSinstance/
```

Instant Messaging での Monitoring Framework の使用

監視可能なオブジェクトおよび属性の一覧については、68 ページの「Instant Messaging のインストゥルメンテーション」を参照してください。

▼ Instant Messaging で監視を有効にするには

1 編集のために次の **Instant Messaging XML** 記述子ファイルを開きます。

```
vi /etc/IM-base/default/com.sun.cmm.im.xml
```

2 「**Install Location**」を *IM-base* から `/etc/IM-base/default` に変更します。

3 修正した **Instant Messaging XML** 記述子を登録します。

```
mfwk-base/bin/mfwksetup -r /etc/IM-base/default/com.sun.cmm.im.xml
```

4 ファイル *IM-base*`/config/iim.conf` に次の行を追加することにより、インストゥルメンテーションを有効にします。

```
iim_server.monitor.enable = true
```

5 次のコマンドで、**Instant Messaging** を再起動します。

```
cd IM-base/sbin
./imadmin stop
./imadmin start
```

6 共通エージェントコンテナを再起動します。

```
cacaoadm restart
```

Messaging Server での Monitoring Framework の使用

監視可能なオブジェクトおよび属性の一覧については、68 ページの「Messaging Server のインストゥルメンテーション」を参照してください。

▼ Messaging Server で監視を有効にするには

1. 次のコマンドで、インストゥルメンテーションを有効にします。

   MsgServ-base/sbin/configutil -o local.mfagent.enable -v 1

2. **Messaging Server XML** モジュールを登録します。

   mfwk-base/bin/mfwksetup -r *MsgServ-base*/lib/com.sun.cmm.ms.xml

3. **Messaging Server** を再起動します。

   ```
   cd MsgServ-base/sbin
   ./stop-msg
   ./start-msg
   ```

4. 共通エージェントコンテナを再起動します。

   ```
   cacaoadm restart
   ```

Portal Server での Monitoring Framework の使用

監視可能なオブジェクトおよび属性の一覧については、68 ページの「Portal Server のインストゥルメンテーション」を参照してください。

▼ Portal Server で監視を有効にするには

- **Portal Server** を有効にするには、次の **URL** にログオンする必要があります。

  http://*FullHostname*:8080/portal/dt

  これによりポータル JSP がコンパイルされ、監視の準備ができたポータルインスタンスが作成されます。

注意事項　Portal Server をホストしている Application Server が再起動されるたびに、この手順を使用して監視を手動で再び有効にする必要があります。

Web Server での Monitoring Framework の使用

監視可能なオブジェクトおよび属性の一覧については、68 ページの「Web Server のインストゥルメンテーション」を参照してください。

▼ Web Server で監視を有効にするには

1 次のコマンドで **Web Server** を起動します。

```
cd /var/WebServer-base/https-FullHostname/bin
./startserv
```

2 管理サーバーを起動します。

```
cd /var/WebServer-base/admin-server/bin
./startserv
```

共通エージェントコンテナの設定

もう 1 つの共有コンポーネントである共通エージェントコンテナは、Monitoring Framework がノードエージェントを実行するために依存するコンポーネントです。インストールの順序によっては、共通エージェントコンテナが停止され再起動が必要になる場合があります。また、共通エージェントコンテナはインストゥルメンテーション済みであり、同様に監視できます。監視対象オブジェクトについては、67 ページの「共通エージェントコンテナのインストゥルメンテーション」を参照してください。

共通エージェントコンテナおよびノードエージェントがすでに起動しているかどうかを確認するには、次のコマンドを実行します。

```
cacaoadm status
```

次のようなメッセージが表示された場合、ノードエージェントは実行中です。

```
default instance is DISABLED at system startup.
Smf monitoring process:
26996
Uptime: 0 day(s), 0:57
```

次のようなメッセージが表示された場合、ノードエージェントは実行中ではありません。

```
default instance is DISABLED at system startup.
default instance is not running.
```

▼ 共通エージェントコンテナの監視を有効にするには

共通エージェントコンテナは、監視を可能にするためのインストゥルメンテーションを備えた共有コンポーネントです。18 ページの「ノードエージェント」で説

明するように、ホスト上またはゾーン内のすべての Java ES コンポーネントは共通エージェントコンテナおよびノードエージェントを共有します。配備環境内の、共通エージェントコンテナを監視するすべての論理ホスト上で root ユーザーとしてこのタスクを実行します。

1. 共通エージェントコンテナが実行中の場合、次のコマンドで停止します。

```
cacaoadm stop
```

2. コンテナ自体のインストゥルメンテーションを有効にします。

```
cacaoadm set-param enable-instrumentation=true
```

3. 設定したパラメータの値を確認し、共通エージェントコンテナを再起動します。

```
cacaoadm get-param enable-instrumentation
cacaoadm start
```

4. キーパスワードを作成します。

```
echo –n password > /etc/mfwk-base/config/security/password.cacao
```

5. キーを生成します。

```
mfwk-base/bin/cpgenkey -n cacao -p /etc/mfwk-base/config/security/password.cacao
```

6. 共通エージェントコンテナ自体の監視モジュールを登録します。

```
cacaoadm register-module /usr/lib/cacao/ext/instrum/config/com.sun.cacao.instrum.xml
cacaoadm register-module /usr/lib/cacao/ext/instrum_jesmf/config/com.sun.cacao.instrum.jesmf.xml
cacaoadm register-module /usr/lib/cacao/ext/instrum_jesmf/config/com.sun.cacao.cmm.xml
```

Monitoring Framework のトラブルシューティング

『Sun Java Enterprise System 5 リリースノート (UNIX 版)』に記載されている既知の問題も参照してください。

HP-UX プラットフォームでの Monitoring Framework の使用

HP-UX 上の Java 仮想マシン (JVM) はデフォルトでは、Monitoring Framework で必要なタスクの多い処理のために調整されておらず、このことが原因で `OutOfMemory` 例外が発生する可能性があります。JVM を設定するには、`HPjconfig` ツールをダウンロードして実行します。ダウンロードは `http://h21007.ww2.hp.com/dspp/tech/tech_TechDocumentDetailPage_IDX/1,1701,1620,00.html` から可能です。

Microsoft Windows での Monitoring Framework の使用

Windows プラットフォーム上での、Monitoring Framework による Java ES コンポーネントの監視は完全にサポートされていますが、いくつかの違いがあります。たとえば、特定の既知の問題を回避するために Java 1.5 以降にアップグレードする必要があります。その他の既知の問題については、『Sun Java Enterprise System 5 リリースノート (Windows 版)』を参照してください。

▼ ノードエージェントを再起動するには

ノードエージェントのホストである共通エージェントコンテナを再起動する必要がある場合、そのノードエージェント経由で監視されているコンポーネントは、次の手順を実行するまでの間 Monitoring Console から参照できなくなります。

1. ノードエージェントのホストである共通エージェントコンテナを再起動します。

   ```
   cacaoadm restart
   ```

2. マスターエージェントのホストである共通エージェントコンテナを再起動します。マスターエージェントは、**Monitoring Console** をインストールしたホスト上または ゾーン内の **Monitoring Framework** 内で動作します。

   ```
   cacaoadm restart
   ```

   マスターエージェントは自動的に、以前に監視していたすべてのノードエージェントに再接続します。

3. **Monitoring Console** のホストである **Web** サーバーを再起動します。

   ```
   /usr/sbin/smcwebserver restart
   ```

mfwkadm コマンド

この節の内容は、マニュアルページのセクション 1M にある、保守コマンド `mfwkadm` のマニュアルページの内容を再構成したものです。このコマンドは、ノードエージェントの内容を管理するために使用します。管理する内容には、監視対象となっているコンポーネントのすべてのモジュールや、監視ジョブと呼ばれるこのノード上で定義した監視ルールが含まれます。マニュアルページの一部の用語や説明については、このガイドで使用されている用語や説明と一致するように変更が加えられています。

機能説明

mfwkadm --help

mfwkadm start

mfwkadm stop

mfwkadm restart

mfwkadm list-params

mfwkadm list-modules

mfwkadm info *runningInstance*

パフォーマンス監視

mfwkadm pm-job observable-classes

mfwkadm pm-job observable-objects [class=*objectClass*] [domain= *objectDomain*]

mfwkadm pm-job observable-attributes class=*objectClass*

mfwkadm pm-job list

mfwkadm pm-job info *jobName*

mfwkadm pm-job create *jobName* granularity=*integerValue* object=*objectName* [object=*objectName* ...]

mfwkadm pm-job delete *jobName*

mfwkadm pm-job suspend *jobName*

mfwkadm pm-job resume *jobName*

運用ステータス監視

mfwkadm opstat-job observable-classes

mfwkadm opstat-job observable-objects [class=*objectClass*] [domain= *objectDomain*]

mfwkadm opstat-job observable-attributes class= *objectClass*

mfwkadm opstat-job list

mfwkadm opstat-job info *jobName*

mfwkadm opstat-job create *jobName* granularity= *integerValue* object=*objectName* [object=*objectName* ...]

mfwkadm opstat-job delete *jobName*

mfwkadm opstat-job suspend *jobName*

mfwkadm opstat-job resume *jobName*

しきい値監視

mfwkadm thrsh-job observable-classes

mfwkadm thrsh-job observable-objects [class=*objectClass*] [domain= *objectDomain*]

mfwkadm thrsh-job observable-attributes class=*objectClass*

mfwkadm thrsh-job list

mfwkadm thrsh-job info *jobName*

mfwkadm thrsh-job create *jobName*
granularity=*integerValue* attributeName=*attributeName* attributeType=*attributeType*
thresholdValue=*thresholdValue* thresholdOffset=*offsetValue*
thresholdDirection=[RISING | FALLING] object=*objectName*

mfwkadm thrsh-job delete *jobName*

mfwkadm thrsh-job suspend *jobName*

mfwkadm thrsh-job resume *jobName*

説明

mfwkadm ユーティリティーは、ノードエージェントとも呼ばれる Monitoring Framework エージェントを管理するためのコマンド行インタフェースです。ノードエージェントは共通エージェントコンテナの内部で実行されます。mfwkadm ユーティリティーを使用して、ノードエージェントを停止および再起動したり、エージェントが実行する監視ジョブを管理したりできます。このコマンドは、ノードエージェントが動作しているホストと同じホストから実行することをお勧めします。このコマンドの引数は、ここで示されている順序どおりに指定する必要があります。

出力メッセージの言語を変更するには、使用する言語のロケールに対応した値を LC_MESSAGE 環境変数に設定します。mfwkadm コマンドは、lib/resources ディレクトリ内の JesmfMessages_*locale*.pm という名前のファイルに格納されているメッセージを使用します。指定されたロケールに対応するメッセージファイルが存在しないか、ロケールを指定しない場合、mfwkadm コマンドは JesmfMessages.pm ファイル内のデフォルトのメッセージセットを使用します。

mfwkadm ユーティリティーのサブコマンドを次に示します。アスタリスク (*) の付いたコマンドを使用するには、共通エージェントコンテナが実行中でノードエージェントがロードされている必要があります。

- start
- stop
- restart
- list-params (*)
- list-modules (*)
- info (*)
- pm-job (*)
- opstat-job (*)
- thrsh-job (*)

ノードエージェントが起動してから mfwkadm ユーティリティーが使用可能になるまでの間には、ロードする共通エージェントコンテナモジュールの数に応じて数秒から数分の遅延が発生します。この期間中は、コマンドを実行しようとしても明示的なメッセージを出して失敗します。

オプション

次のオプションがサポートされています。

--help　　使用方法の要約を表示します。

サブコマンド

start
　共通エージェントコンテナを停止せずに、Monitoring Framework ノードエージェントとその関連コンポーネント製品モジュールを起動します。

　この動作では、まずノードエージェントが配備されてから、関連するコンポーネント製品モジュールが共通エージェントコンテナに配備されます。この機能は、cacaoadm ユーティリティーの lock および undeploy サブコマンドのラッパーです。

　start サブコマンドでは、ノードエージェントと、Monitoring Framework に関連付けられた Java ES コンポーネントモジュールのみが起動されます。コンポーネントモジュールには com.sun.cmm というプレフィックスが付きます。

　セキュリティー:start サブコマンドを実行できるのは、共通エージェントコンテナを起動したユーザーに限られます。それ以外のユーザーが実行しようとした場合、次のようなエラーメッセージが表示されます。

```
Error occured in mfwkadm
Problem running /usr/sbin/cacaoadm unlock com.sun.mfwk 2>&1.
Stdout/Stderr: This command must be run by user: [root].
```

`stop`
: Monitoring Framework ノードエージェントと、それに関連付けられた共通エージェントコンテナ内の Java ES コンポーネントモジュールを停止します。

  この動作ではまず、共通エージェントコンテナに配備されているすべての Java ES コンポーネントのモジュールが停止され、続いてノードエージェントが停止されます。この機能は、`cacaoadm` ユーティリティーの `lock` および `unlock` サブコマンドのラッパーです。

  `stop` サブコマンドは、Monitoring Framework と関連付けられている Java ES コンポーネントモジュールのみを停止し、続いてノードエージェント自体を停止します。コンポーネントモジュールには `com.sun.cmm` というプレフィックスが付きます。

  セキュリティー:`stop` サブコマンドを実行できるのは、共通エージェントコンテナを起動したユーザーに限られます。それ以外のユーザーが実行しようとした場合、次のようなエラーメッセージが表示されます。

```
Error occured in mfwkadm
Problem running /usr/sbin/cacaoadm unlock com.sun.mfwk 2>&1.
Stdout/Stderr: This command must be run by user: [root].
```

`restart`
: Monitoring Framework ノードエージェントと、それに関連付けられた共通エージェントコンテナ内の Java ES コンポーネントモジュールを再起動します。

  この動作は、`stop` および `start` サブコマンドと同じように、ノードエージェントと、それに関連付けられた共通エージェントコンテナ内のモジュールの停止および起動を試みます。

  セキュリティー:`restart` サブコマンドを実行できるのは、共通エージェントコンテナを起動したユーザーに限られます。それ以外のユーザーが実行しようとした場合、次のようなエラーメッセージが表示されます。

```
Error occured in mfwkadm
Problem running //usr/sbin/cacaoadm unlock com.sun.mfwk 2>&1.
Stdout/Stderr: This command must be run by user: [root].
```

`list-params`
: Monitoring Framework ノードエージェントに関係するすべての設定パラメータを一覧表示します。

  セキュリティー:このコマンドにはユーザー制限はありません。

`list-modules`
Common Monitoring Model (CMM) を実装し、共通エージェントコンテナにロードされるコンポーネント製品モジュールの一覧を表示します。このサブコマンドは、インストールされた各 Java ES コンポーネントのすべての実行中インスタンスも一覧表示します。各コンポーネントには、0 個、1 個、または 2 個以上の実行中インスタンスが存在する可能性があります。

セキュリティー:共通エージェントコンテナを起動したユーザー以外のユーザーが実行した場合、インストールされた Java ES コンポーネントの一覧にはコンポーネントインスタンスは含まれません。

`info` *runningInstance*
runningInstance で指定された実行中インスタンスについての情報を表示します。*runningInstance* は、`list-modules` サブコマンドの出力で表示された実行中インスタンスと一致する必要があります。

表示される情報には次のものが含まれます。

- 監視ジョブのタイプごとに、実行中のインスタンスと関連付けられたすべての監視可能オブジェクトがクラス名でソートされて表示されます。監視可能なオブジェクトとは、`pm-job` サブコマンドでパフォーマンス監視ジョブを、`opstat-job` サブコマンドで運用ステータスジョブを、または `thrsh-job` サブコマンドでしきい値監視ジョブを作成できるオブジェクトのことです。
- 監視可能オブジェクトのクラスごとに、各クラスの名前やタイプなど、クラスの監視可能属性のすべてが表示されます。

セキュリティー:共通エージェントコンテナを起動したユーザー以外によって実行された場合、情報は表示されません。

パフォーマンス監視

`pm-job observable-classes`
パフォーマンス監視ジョブを作成できるオブジェクトについて、現在監視可能なすべてのクラスの一覧を表示します。

`pm-job observable-objects` [class= *objectC lass*] [domain=*objectDomain*]
パフォーマンス監視ジョブを作成できる、現在監視可能なすべてのオブジェクトの一覧を表示します。デフォルトでは、すべての監視可能クラスの、またすべてのドメイン内のすべてのオブジェクトが一覧表示されます。オブジェクトの一覧はクラス名でソートされます。

class=*objectClass*
オプションの *objectClass* を指定すると、その特定クラスの監視可能オブジェクトのみに出力が限定されます。*objectClass* は、`pm-job observable-classes` サブコマンドによって一覧表示されるいずれかのクラスである必要があります。

domain=*objectDomain*
オプションの *objectDomain* を指定すると、そのドメイン内の監視可能オブジェクトのみに出力が限定されます。オブジェクトのドメインは、オブジェクトの名前のうちコロン (:) 記号よりも前の文字列の部分です。

`pm-job observable-attributes class=`*objectClass*
objectClass で指定されたオブジェクトクラスのすべての監視可能属性の一覧を表示します。属性はその名前および型とともに表示されます。*objectClass* は、`pm-job observable-classes` サブコマンドによって一覧表示されるパフォーマンス監視ジョブをサポートするクラスのいずれかである必要があります。

`pm-job list`
現在定義されているすべてのパフォーマンス監視ジョブの一覧を表示します。定義されているパフォーマンス監視ジョブが存在するオブジェクトごとにジョブが一覧表示され、オブジェクトはそのクラス名によってソートされます。各ジョブについて表示される情報は、`pm-job info` サブコマンドによって表示されるものと同じです。

セキュリティー:共通エージェントコンテナを起動したユーザー以外によって実行された場合、ジョブは表示されません。

`pm-job info` *jobName*

jobName で指定されたパフォーマンス監視ジョブについての詳細情報を表示します。*jobName* は、`pm-job list` サブコマンドによって表示されるジョブである必要があります。このサブコマンドは次の情報を表示します。

- パフォーマンス監視ジョブの名前。
- パフォーマンス監視ジョブのタイプ。「by object」(オブジェクト別) または「by class」(クラス別) のいずれか。オブジェクト別ジョブは、1 つ以上の名前付きオブジェクトインスタンスを監視します。クラス別ジョブは、オブジェクトクラスのすべてのインスタンスを監視します。`mfwkadm` ユーティリティーではクラス別ジョブを作成できないことに注意してください。
- パフォーマンス監視ジョブの状態。「active on-duty」(アクティブで監視中)、「active off-duty」(アクティブのみ)、または「suspended」(停止中) のいずれか。「アクティブで監視中」のジョブは、その時点でスケジュールに従って実行中であり、かつデータを収集中です。「アクティブのみ」のジョブは実行中ですが、その時点で作業スケジュール外であるためデータは収集していません。「停止中」のジョブは実行されておらず、データも収集していません。パフォーマンス監視ジョブの実行状態を変更するには、`pm-job suspend` および `pm-job resume` サブコマンドを使用します。
- パフォーマンス監視ジョブの粒度 (秒数)。これは、このジョブによるデータ収集の間隔です。
- 監視ジョブのレポート期間。レポート期間と粒度の積により、通知頻度が算出されます。たとえば、粒度が 10 秒でレポート期間が 6 の場合、イベント別のジョブレポートは 10 秒おきにデータを収集し、6 つのレポートを含む通知を 60

秒 (10*6) おきに送信します。ジョブがファイル別のレポートも行っている場合、このジョブは 6 つの生成されたファイルの場所を含むイベントを 60 秒おきに送信します。

- パフォーマンス監視ジョブがイベント別のレポートを行っているかどうか。これは、パフォーマンス監視ジョブの結果が、登録済みクライアントに通知として送信されることを意味します。
- パフォーマンス監視ジョブがファイル別のレポートを行っているかどうか。これは、パフォーマンス監視ジョブのレポートがローカルファイルに書き込まれ、ファイル名を含む通知が登録済みクライアントに送信されることを意味します。
- パフォーマンス監視ジョブのレポート形式。これは常に XML です。
- パフォーマンス監視ジョブのスケジュール。スケジュールは、ジョブが「アクティブで監視中」(データを収集する) または「アクティブのみ」(データは収集しない) の状態になる日時を指定します。

オブジェクト別ジョブについては、次の情報が表示されます。

- 名前でソートされた監視対象オブジェクトの一覧。
- 監視可能属性のうち一部のみが指定されている場合、監視対象オブジェクトの監視対象属性が名前および型別に一覧表示されます。

クラス別ジョブについては、次の情報が表示されます。

- 名前でソートされた監視対象クラスの一覧。
- 監視可能属性のうち一部のみが指定されている場合、監視対象クラスの監視対象属性が名前および型別に一覧表示されます。これらの属性はすべてのクラスに共通です。

セキュリティー:共通エージェントコンテナを起動したユーザー以外によって実行された場合、情報は表示されません。

`pm-job create` *jobName* granularity= *integerValue* object=*objectName* [object= *objectName* ...]

1 つ以上のオブジェクトを対象とした新しいパフォーマンス監視ジョブを作成します。`mfwkadm` コマンドでは、クラス別ジョブは作成できません。パフォーマンス監視ジョブを作成するとき、次のパラメータを設定できます。

jobName

パフォーマンス監視ジョブを一意に識別する文字列。*jobName* には、ほかのパフォーマンス監視ジョブによってすでに使用されている文字列は指定できません。

granularity=*integerValue*

ジョブが「アクティブで監視中」のときに、2 回の連続する測定データ収集の開始間隔を秒数で指定します。粒度期間の指定例は、300 秒 (5 分)、900 秒 (15 分)、1800 秒 (30 分おき)、3600 秒 (1 時間おき) などです。ほとんどの場合、300

秒の粒度期間で十分です。一部の測定値については、より大きな粒度期間でデータを収集すると効果的な場合があります。

object=*objectName* [object= *objectName* ...]
パフォーマンス監視ジョブがデータを収集してレポートする、1 つ以上の監視対象オブジェクト。*objectName* は、`pm-job list` または `pm-job observable-objects` サブコマンドによって表示されるオブジェクトである必要があります。「object=*objectName*」パラメータを複数指定すると、複数のオブジェクトを監視する 1 つのパフォーマンス監視ジョブが作成されます。

セキュリティー:このサブコマンドを実行できるのは、共通エージェントコンテナを起動したユーザーに限られます。

`pm-job delete` *jobName*
jobName で指定されたパフォーマンス監視ジョブを削除します。*jobName* は、`pm-job list` サブコマンドによって表示されるジョブである必要があります。

セキュリティー:このサブコマンドを実行できるのは、共通エージェントコンテナを起動したユーザーに限られます。

`pm-job suspend` *jobName*
jobName で指定されたパフォーマンス監視ジョブを停止します。停止されたジョブは、そのスケジュールに関係なく、実行されずデータも収集しません。ただし、ジョブの定義は残り、`pm-job resume` サブコマンドによってそのジョブを再びアクティブにすることができます。*jobName* は、`pm-job list` サブコマンドによって表示されるジョブである必要があります。

セキュリティー:このサブコマンドを実行できるのは、共通エージェントコンテナを起動したユーザーに限られます。

`pm-job resume` *jobName*
jobName で指定されたパフォーマンス監視ジョブを再開します。再開されたジョブは、そのスケジュールに従ったデータ収集およびレポート送信を開始します。*jobName* は、`pm-job list` サブコマンドによって表示されるジョブである必要があります。これは、`pm-job suspend` サブコマンドとペアで機能するコマンドです。

セキュリティー:このサブコマンドを実行できるのは、共通エージェントコンテナを起動したユーザーに限られます。

運用ステータス監視

`opstat-job observable-classes`
運用ステータス監視ジョブを作成できるオブジェクトについて、現在監視可能なすべてのクラスの一覧を表示します。

`opstat-job observable-objects` [class= *objectClass*] [domain=*objectDomain*]
運用ステータス監視ジョブを作成できる、現在監視可能なすべてのオブジェクトの一覧を表示します。デフォルトでは、すべての監視可能クラスの、またすべてのドメイン内のすべてのオブジェクトが一覧表示されます。オブジェクトの一覧はクラス名でソートされます。

class=*objectClass*
オプションの *objectClass* を指定すると、その特定クラスの監視可能オブジェクトのみに出力が限定されます。*objectClass* は、`opstat-job observable-classes` サブコマンドによって一覧表示されるいずれかのクラスである必要があります。

domain=*objectDomain*
オプションの *objectDomain* を指定すると、そのドメイン内の監視可能オブジェクトのみに出力が限定されます。オブジェクトのドメインは、オブジェクトの名前のうちコロン (:) 記号よりも前の文字列の部分です。

`opstat-job observable-attributes class=` *objectClass*
objectClass で指定されたオブジェクトクラスのすべての監視可能属性の一覧を表示します。属性はその名前および型とともに表示されます。*objectClass* は、`opstat-job observable-classes` サブコマンドによって一覧表示されるいずれかのクラスである必要があります。

`opstat-job list`
現在定義されているすべての運用ステータス監視ジョブの一覧を表示します。定義されている運用ステータス監視ジョブが存在するオブジェクトごとにジョブが一覧表示され、オブジェクトはそのクラス名によってソートされます。各ジョブについて表示される情報は、`opstat-job info` サブコマンドによって表示されるものと同じです。

セキュリティー:共通エージェントコンテナを起動したユーザー以外によって実行された場合、ジョブは表示されません。

`opstat-job info` *jobName*

jobName で指定された運用ステータス監視ジョブについての詳細情報を表示します。*jobName* は、`opstat-job list` サブコマンドによって表示されるジョブである必要があります。このサブコマンドは次の情報を表示します。

- 運用ステータス監視ジョブの名前。
- 運用ステータス監視ジョブのタイプ。「by object」(オブジェクト別) または「by class」(クラス別) のいずれか。オブジェクト別ジョブは、1 つの名前付きオブジェクトインスタンスを監視します。クラス別ジョブは、オブジェクトクラスのすべてのインスタンスを監視します。`mfwkadm` ユーティリティーではクラス別ジョブを作成できないことに注意してください。
- 運用ステータス監視ジョブの状態。「active on-duty」(アクティブで監視中)、「active off-duty」(アクティブのみ)、または「suspended」(停止中) のいずれか。「アクティブで監視中」のジョブは、その時点でスケジュールに従って実

行中であり、かつデータを収集中です。「アクティブのみ」のジョブは実行中ですが、その時点で作業スケジュール外であるためデータは収集していません。「停止中」のジョブは実行されておらず、データも収集していません。運用ステータス監視ジョブの実行状態を変更するには、`opstat-job suspend` および `opstat-job resume` サブコマンドを使用します。

- 運用ステータス監視ジョブの粒度 (秒数)。これは、このジョブによるデータ収集の間隔です。
- 運用ステータス監視ジョブがイベント別のレポートを行っているかどうか。これは、運用ステータス監視ジョブの結果が、登録済みクライアントに通知として送信されることを意味します。
- 運用ステータス監視ジョブがファイル別のレポートを行っているかどうか。これは、運用ステータス監視ジョブのレポートがローカルファイルに書き込まれ、ファイル名を含む通知が登録済みクライアントに送信されることを意味します。
- 運用ステータス監視ジョブのレポート形式。これは常に `XML` です。
- 運用ステータス監視ジョブのスケジュール。スケジュールは、ジョブが「アクティブで監視中」(データを収集する) または「アクティブのみ」(データは収集しない) の状態になる日時を指定します。
- オブジェクト別ジョブの場合、名前でソートされた監視対象オブジェクトの一覧。
- クラス別ジョブの場合、名前でソートされた監視対象クラスの一覧。

セキュリティー**:**共通エージェントコンテナを起動したユーザー以外によって実行された場合、情報は表示されません。

`opstat-job create` *jobName* granularity= *integerValue* object=*objectName* [object= *objectName* ...]

1 つ以上のオブジェクトを対象とした新しい運用ステータス監視ジョブを作成します。`mfwkadm` コマンドでは、クラス別ジョブは作成できません。パフォーマンス監視ジョブを作成するとき、次のパラメータを設定できます。

jobName
運用ステータス監視ジョブを一意に識別する文字列。*jobName* には、ほかの運用ステータス監視ジョブによってすでに使用されている文字列は指定できません。

granularity=*integerValue*
ジョブが「アクティブで監視中」のときに、2 回の連続する測定データ収集の開始間隔を秒数で指定します。

object=*objectName* [object= *objectName* ...]
運用ステータス監視ジョブがデータを収集してレポートする、1 つ以上の監視対象オブジェクト。*objectName* は、`opstat-job list` または `opstat-job observable-objects` サブコマンドによって表示されるオブジェクトである必要

があります。「object=*objectName*」パラメータを複数指定すると、複数のオブジェクトを監視する 1 つの運用ステータス監視ジョブが作成されます。

セキュリティー:このサブコマンドを実行できるのは、共通エージェントコンテナを起動したユーザーに限られます。

`opstat-job delete` *jobName*
jobName で指定された運用ステータス監視ジョブを削除します。*jobName* は、`opstat-job list` サブコマンドによって表示されるジョブである必要があります。

セキュリティー:このサブコマンドを実行できるのは、共通エージェントコンテナを起動したユーザーに限られます。

`opstat-job suspend` *jobName*
jobName で指定された運用ステータス監視ジョブを停止します。停止されたジョブは、そのスケジュールに関係なく、実行されずデータも収集しません。ただし、ジョブの定義は残り、`opstat-job resume` サブコマンドによってそのジョブを再びアクティブにすることができます。*jobName* は、`opstat-job list` サブコマンドによって表示されるジョブである必要があります。

セキュリティー:このサブコマンドを実行できるのは、共通エージェントコンテナを起動したユーザーに限られます。

`opstat-job resume` *jobName*
jobName で指定された運用ステータス監視ジョブを再開します。再開されたジョブは、そのスケジュールに従ったデータ収集およびレポート送信を開始します。*jobName* は、`opstat-job list` サブコマンドによって表示されるジョブである必要があります。これは、`opstat-job suspend` サブコマンドとペアで機能するコマンドです。

セキュリティー:このサブコマンドを実行できるのは、共通エージェントコンテナを起動したユーザーに限られます。

しきい値監視

`thrsh-job observable-classes`
しきい値監視ジョブを作成できるオブジェクトについて、現在監視可能なすべてのクラスの一覧を表示します。

`thrsh-job observable-objects` [class= *objectClass*] [domain=*objectDomain*]
しきい値監視ジョブを作成できる、現在監視可能なすべてのオブジェクトの一覧を表示します。デフォルトでは、すべての監視可能クラスの、またすべてのドメイン内のすべてのオブジェクトが一覧表示されます。オブジェクトの一覧はクラス名でソートされます。

class=*objectClass*
オプションの *objectClass* を指定すると、その特定クラスの監視可能オブジェクトのみに出力が限定されます。*objectClass* は、`thrsh-job observable-classes` サブコマンドによって一覧表示されるいずれかのクラスである必要があります。

domain=*objectDomain*
オプションの *objectDomain* を指定すると、そのドメイン内の監視可能オブジェクトのみに出力が限定されます。オブジェクトのドメインは、オブジェクトの名前のうちコロン (:) 記号よりも前の文字列の部分です。

`thrsh-job observable-attributes class=`*objectClass*
objectClass で指定されたオブジェクトクラスのすべての監視可能属性の一覧を表示します。属性はその名前および型とともに表示されます。*objectClass* は、`thrsh-job observable-classes` サブコマンドによって一覧表示されるいずれかのクラスである必要があります。

`thrsh-job list`
現在定義されているすべてのしきい値監視ジョブの一覧を表示します。定義されているしきい値監視ジョブが存在するオブジェクトごとにジョブが一覧表示され、オブジェクトはそのクラス名によってソートされます。各ジョブについて表示される情報は、`thrsh-job info` サブコマンドによって表示されるものと同じです。

セキュリティー:共通エージェントコンテナを起動したユーザー以外によって実行された場合、ジョブは表示されません。

`thrsh-job info` *jobName*

jobName で指定されたしきい値監視ジョブについての詳細情報を表示します。*jobName* は、`thrsh-job list` サブコマンドによって表示されるジョブである必要があります。このサブコマンドは次の情報を表示します。

- しきい値監視ジョブの名前。
- しきい値監視ジョブの多重度。このリリースでは、1 つのオブジェクトの 1 つの属性を監視する単純なしきい値監視ジョブのみを使用できます。
- しきい値監視ジョブの状態。「active on-duty」(アクティブで監視中)、「active off-duty」(アクティブのみ)、または「suspended」(停止中) のいずれか。「アクティブで監視中」のジョブは、その時点でスケジュールに従って実行中であり、かつデータを収集中です。「アクティブのみ」のジョブは実行中ですが、その時点で作業スケジュール外であるためデータは収集していません。「停止中」のジョブは実行されておらず、データも収集していません。しきい値監視ジョブの実行状態を変更するには、`thrsh-job suspend` および `thrsh-job resume` サブコマンドを使用します。
- しきい値監視ジョブの粒度 (秒数)。これは、このジョブによるデータ収集の間隔です。
- しきい値監視ジョブのスケジュール。スケジュールは、ジョブが「アクティブで監視中」(データを収集する) または「アクティブのみ」(データは収集しない) の状態になる日時を指定します。
- しきい値監視ジョブのアラーム設定。これは、監視対象属性の監視対象値が、定義されたしきい値を超えたときに生成されるアラームです。表示にはアラームの種類および重大度が含まれます。

- しきい値監視ジョブの監視対象オブジェクト。
- しきい値が適用される属性名。
- アラームを生成するしきい値の値。
- しきい値に達したときにアラームを生成する値の進行方向。RISING (上昇) または FALLING (下降)。
- しきい値の許容範囲オフセット。方向が RISING の場合、監視対象属性の値が *thresholdValue-offsetValue* を下回るまでアラームは再生成されません。方向が FALLING の場合、監視対象属性の値が *thresholdValue+offsetValue* を上回るまでアラームは再生成されません。この動作は、オフセットがゼロの場合でも適用されます。

セキュリティー:共通エージェントコンテナを起動したユーザー以外によって実行された場合、情報は表示されません。

thrsh-job create *jobName* object= *objectName* granularity=*integerValue* attributeName= *attributeName* attributeType=*attributeType* thresholdValue= *thresholdValue* thresholdOffset=*offsetValue* thresholdDirection= [RISING|FALLING]

1 つのオブジェクトの 1 つの属性を監視する新しいしきい値監視ジョブを作成します。しきい値ジョブを作成するとき、次のパラメータを設定できます。

jobName
: しきい値監視ジョブを一意に識別する文字列。*jobName* には、ほかのしきい値監視ジョブによってすでに使用されている文字列は指定できません。

object=*objectName*
: しきい値監視ジョブが、しきい値との比較目的で属性値を収集する対象の監視可能オブジェクト。*objectName* は、thrsh-job list または thrsh-job observable-objects サブコマンドによって表示されるオブジェクトである必要があります。

granularity=*integerValue*
: ジョブが「アクティブで監視中」のときに、2 回の連続する属性値監視の開始間隔を秒数で指定します。

attributeName=*attributeName*
: しきい値監視ジョブがその値を収集し、しきい値と比較する対象の属性の名前。*attributeName* は、thrsh-job info または thrsh-job observable-attributes サブコマンドによって一覧表示される属性名である必要があります。

attributeType=*attributeType*
: 監視される監視可能属性の型。*attributeType* は、thrsh-job info または thrsh-job observable-attributes サブコマンドによって一覧表示される属性の型である必要があります。

thresholdValue=*thresholdValue*
: thresholdDirection で指定された方向にしきい値を通過したときに、このしきい値監視ジョブがアラームを生成する監視対象属性の値。

thresholdOffset=*offsetValue*
offsetValue は、後続のアラームを生成する際のしきい値監視ジョブの許容範囲を決定します。*offsetValue* はゼロまたは正の値である必要があります。アラームイベントが生成されたあと、*offsetValue* および `thresholdDirection` によって定義された範囲を監視対象属性の値が超えるまで、新しいアラームイベントは生成されません。

thresholdDirection=`[RISING|FALLING]`
方向が `RISING` の場合、監視対象属性の値が *thresholdValue-offsetValue* を下回るまでアラームイベントは再生成されません。方向が `FALLING` の場合、監視対象属性の値が *thresholdValue+offsetValue* を上回るまでアラームイベントは再生成されません。この動作は、*offsetValue* がゼロの場合にも適用されます。

セキュリティー:このサブコマンドを実行できるのは、共通エージェントコンテナを起動したユーザーに限られます。

`thrsh-job delete` *jobName*
jobName で指定されたしきい値監視ジョブを削除します。*jobName* は、`thrsh-job list` サブコマンドによって表示されるジョブである必要があります。

セキュリティー:このサブコマンドを実行できるのは、共通エージェントコンテナを起動したユーザーに限られます。

`thrsh-job suspend` *jobName*
jobName で指定されたしきい値監視ジョブを停止します。停止されたジョブは、そのスケジュールに関係なく、実行されずデータも収集しません。ただし、ジョブの定義は残り、`thrsh-job resume` サブコマンドによってそのジョブを再びアクティブにすることができます。*jobName* は、`thrsh-job list` サブコマンドによって表示されるジョブである必要があります。

セキュリティー:このサブコマンドを実行できるのは、共通エージェントコンテナを起動したユーザーに限られます。

`thrsh-job resume` *jobName*
jobName で指定されたしきい値監視ジョブを再開します。再開されたジョブは、そのスケジュールに従ったデータ収集およびレポート送信を開始します。*jobName* は、`thrsh-job list` サブコマンドによって表示されるジョブである必要があります。これは、`thrsh-job suspend` サブコマンドとペアで機能するコマンドです。

セキュリティー:このサブコマンドを実行できるのは、共通エージェントコンテナを起動したユーザーに限られます。

例

次の架空のシナリオは、`mfwkadm` ユーティリティーと、そのオプションおよびサブコマンドの使用方法の例を示します。

list-modules サブコマンドは、現在のホスト上の Java ES コンポーネントインスタンスと、それらに対応する共通エージェントコンテナ内のモジュールを示します。次の例は、2 つのインストール済みコンポーネントを一覧表示します。Directory Server には実行中のインスタンスがなく、Web Server には 1 つの実行中インスタンスがあります。

```
$ mfwkadm list-modules

Installed products and their running instances:
===============================================

Instances for installed product: com.sun.cmm.ds:collectionID=/opt/SUNWdsee/ds6,
name=Sun Java(TM) System Directory Server,type=CMM_InstalledProduct
------------------------------

No instance.

Instances for installed product: com.sun.cmm.ws:collectionID=/var/opt/SUNWwbsvr7,
name=WebServer,type=CMM_InstalledProduct
------------------------------

/wsPrefix/com.sun.cmm.ws:name=https-hostname.example.com,type=CMM_ApplicationSystem
```

次の info サブコマンドは、Web Server インスタンス内の監視可能オブジェクトと、それらのオブジェクトのクラスおよび属性をジョブのタイプ別に表示します。

```
$ mfwkadm info /wsPrefix/com.sun.cmm.ws:name=https-hostname.example.com,\\
type=CMM_ApplicationSystem

Information about running instance: /wsPrefix/com.sun.cmm.ws:
name=https-hostname.example.com,type=CMM_ApplicationSystem
==================================

Observable objects for performance jobs:
----------------------------------------

+ Objects of class: com.sun.cmm.settings.CMM_ApplicationSystemSetting

        /wsPrefix/com.sun.cmm.ws:name=https-hostname.example.com-setting,
type=CMM_ApplicationSystemSetting

        Observable attributes:

        Caption [STRING]
        ConfigurationDirectory [STRING]
        CreationClassName [STRING]
        Description [STRING]
        DirectoryName [STRING]
```

```
        ElementName [STRING]
        InstanceID [STRING]
        Name [STRING]
        URL [STRING]

+ Objects of class: com.sun.cmm.settings.CMM_KeepAliveSetting

        /wsPrefix/com.sun.cmm.ws:name=process-1-keepalive-setting,
type=CMM_KeepAliveSetting

        Observable attributes:

        AllocationUnit [STRING]
        Caption [STRING]
        ConnectionsUpperBound [LONG]
        CreationClassName [STRING]
        Description [STRING]
        ElementName [STRING]
        InputUnit [STRING]
        InstanceID [STRING]
        LowerAllocationLimit [LONG]
        LowerInputLimit [LONG]
        LowerOutputLimit [LONG]
        Name [STRING]
        OtherAllocationUnit [STRING]
        OtherInputUnit [STRING]
        OtherLowerAllocationLimit [LONG]
        OtherLowerInputLimit [LONG]
        OtherLowerOutputLimit [LONG]
        OtherOutputUnit [STRING]
        OtherUpperAllocationLimit [LONG]
        OtherUpperInputLimit [LONG]
        OtherUpperOutputLimit [LONG]
        OutputUnit [STRING]
        QueuedUpperBound [LONG]
        SecondsTimeout [LONG]
        TimeoutUpperBound [LONG]
        UpperAllocationLimit [LONG]
        UpperInputLimit [LONG]
        UpperOutputLimit [LONG]
        ...
```

次のコマンドは、定義済みのパフォーマンス監視ジョブの一覧を表示します。この例では、1 つのオブジェクトを監視する、`myPerfJob` という名前の 1 つのパフォーマンス監視ジョブが存在します。

```
$ mfwkadm pm-job list

BY_OBJECTS performance jobs:
```

```
==========================

Performance job information for: myPerfJob
------------------------------

Type:                BY_OBJECTS
State:               ACTIVE_ON_DUTY
Granularity period:  30
Reporting period:    1
By event:            EVENT_SINGLE
By file:             EVENT_SINGLE
Report format:       XML
Schedule:
        Global start time: Immediately
        Global stop time: Forever
        Weekly schedule: Everyday
        Daily schedule: All day
Observed objects:
                /wsPrefix/com.sun.cmm.ws:name=virtualServer-hostname.example.com-
webApp-/-stats,type=CMM_VirtualServerWebModuleStats
Observed attributes:
                All available

BY_CLASSES performance jobs:
==========================

No jobs found.
```

次のコマンドは、opstat-job info または opstat-job observable-objects サブコマンドで取得された 2 つの監視可能オブジェクトに関係する運用ステータス監視ジョブを作成します。

```
$ mfwkadm opstat-job create myOpStatJob granularity=60 \\
object=/wsPrefix/com.sun.cmm.ws:name=process-1,type=CMM_UnixProcess \\
object=/wsPrefix/com.sun.cmm.ws:name=process-1-DNSCache1,type=CMM_DnsCache
```

次のコマンドは、前の例で作成したジョブを停止します。

```
$ mfwkadm opstat-job suspend myOpStatJob
```

次のコマンドは、潜在的なしきい値監視ジョブの監視可能クラスを示します。

```
$ mfwkadm thrsh-job observable-classes

Threshold jobs observable classes:
================================

com.sun.cmm.cim.CIM_ScopedSettingData
```

```
com.sun.cmm.cim.CIM_SettingData
com.sun.cmm.cim.CIM_StatisticalData
com.sun.cmm.cim.statistics.CIM_EthernetPortStatistics
com.sun.cmm.cim.statistics.CIM_NetworkPortStatistics
com.sun.cmm.cim.statistics.j2ee.CIM_J2eeJVMStats
com.sun.cmm.cim.statistics.j2ee.CIM_J2eeStatistic
com.sun.cmm.settings.CMM_ApplicationSystemSetting
com.sun.cmm.settings.CMM_KeepAliveSetting
com.sun.cmm.settings.CMM_QueueTimeoutSetting
com.sun.cmm.settings.CMM_RFC2788ApplicationSystemSetting
com.sun.cmm.settings.CMM_ScopedSettingData
com.sun.cmm.settings.CMM_SoftwareResourceSetting
com.sun.cmm.settings.CMM_SWRBufferSetting
com.sun.cmm.settings.CMM_SWRLimitSetting
com.sun.cmm.settings.CMM_SWRQueueSetting
com.sun.cmm.settings.CMM_VirtualServerSetting
com.sun.cmm.statistics.CMM_ApplicationSystemStats
com.sun.cmm.statistics.CMM_ApplicationSystemWatchdogStats
com.sun.cmm.statistics.CMM_ConnectionQueueStats
com.sun.cmm.statistics.CMM_DnsCacheStats
com.sun.cmm.statistics.CMM_EthernetPortStats
com.sun.cmm.statistics.CMM_FileCacheStats
com.sun.cmm.statistics.CMM_HTTPResponsesStats
com.sun.cmm.statistics.CMM_JVMJSR174ExtStats
com.sun.cmm.statistics.CMM_JVMJSR174Stats
com.sun.cmm.statistics.CMM_JVMStats
com.sun.cmm.statistics.CMM_NetworkPortStats
com.sun.cmm.statistics.CMM_OperatingSystemStats
com.sun.cmm.statistics.CMM_ProcessorStats
com.sun.cmm.statistics.CMM_ProcessStats
com.sun.cmm.statistics.CMM_QueueTimeoutStats
com.sun.cmm.statistics.CMM_RFC2788ApplicationTableStats
com.sun.cmm.statistics.CMM_ServiceStats
com.sun.cmm.statistics.CMM_SoftwareResourceStats
com.sun.cmm.statistics.CMM_SolarisEthernetPortStats
com.sun.cmm.statistics.CMM_SolarisNetworkPortStats
com.sun.cmm.statistics.CMM_SolarisOperatingSystemStats
com.sun.cmm.statistics.CMM_SolarisProcessorStats
com.sun.cmm.statistics.CMM_SolarisProcessorSysinfoStats
com.sun.cmm.statistics.CMM_SolarisProcessorVmStats
com.sun.cmm.statistics.CMM_Statistic
com.sun.cmm.statistics.CMM_SWRBufferStats
com.sun.cmm.statistics.CMM_SWRCacheStats
com.sun.cmm.statistics.CMM_SWRLimitStats
com.sun.cmm.statistics.CMM_SWRQueueStats
com.sun.cmm.statistics.CMM_UnixOperatingSystemStats
com.sun.cmm.statistics.CMM_UnixProcessStats
com.sun.cmm.statistics.CMM_VirtualServerWebModuleStats
com.sun.cmm.statistics.CMM_WebModuleStats
```

次のコマンドは、前の例で見つかったクラス com.sun.cmm.statistics.CMM_SWRQueueStats のオブジェクトを監視するしきい値監視ジョブについて、監視可能属性を示します。

```
$ mfwkadm thrsh-job observable-attributes \\
class=com.sun.cmm.statistics.CMM_SWRQueueStats

Threshold jobs observable attributes:
=====================================

Class: com.sun.cmm.statistics.CMM_SWRQueueStats

Attributes:

BufferSize [LONG]
EntriesCount [LONG]
EntriesHighWaterMark [LONG]
EntriesLowWaterMark [LONG]
EntriesTotal [LONG]
ErrorCount [INTEGER]
FailedOperations [LONG]
LowerLimit [LONG]
OperationsCount [LONG]
OtherLowerLimit [LONG]
OtherUpperLimit [LONG]
OverflowsCount [LONG]
QueuedCount [LONG]
QueuedHighWater [LONG]
SampleInterval [LONG]
TotalQueuedCount [LONG]
UpperLimit [LONG]
```

次のコマンドは、ジョブ作成のもう 1 つの例です。ここではしきい値監視ジョブを作成します。

```
$ mfwkadm thrsh-job create myThreshJob granularity=30 \\
object=/wsPrefix/com.sun.cmm.ws:name=process-1-threadPool-NativePool-stats,\\
type=CMM_SWRQueueStats attributeName=EntriesCount attributeType=LONG \\
thresholdValue=1000 thresholdOffset=10 thresholdDirection=RISING
```

次の例は、前の例で作成したしきい値監視ジョブに対する thrsh-job info サブコマンドの出力を示します。

```
$ mfwkadm thrsh-job info myThreshJob

Threshold job information for: myThreshJob
------------------------------
```

```
Type:                SIMPLE
State:               ACTIVE_ON_DUTY
Granularity period:  30
Schedule:
        Global start time: Immediately
        Global stop time: Forever
        Weekly schedule: Everyday
        Daily schedule: All day
Alarm configuration:
               Type: QualityOfServiceAlarm
               Severity: INDETERMINATE
Threshold definition(s):
               Object: /wsPrefix/com.sun.cmm.ws:name=process-1-threadPool-
NativePool-stats,type=CMM_SWRQueueStats
                       Attribute: EntriesCount [LONG]
                       Value: 1000
                       Direction: RISING
                       Offset: 10
```

終了ステータス

次の終了値が返されます。

0　正常終了

1　エラーが発生した。

属性

属性タイプ	属性値
有効な Solaris のバージョン	SUNWmfwk
インタフェースの安定性	Contract Private

関連項目

cacao.5、cacaoadm.1m

◆ ◆ ◆ 第 3 章

Monitoring Console のインストールと使用

Monitoring Console は、インストゥルメンテーションによって収集されたすべての監視データを表示する Web ベースのアプリケーションです。Monitoring Console はマスターエージェントを利用して、個々のノードエージェントからすべての値およびアラーム通知を集約します。

インストールされた Monitoring Console には、任意のホスト上のブラウザウィンドウから安全にアクセスできます。ファイアウォールの設定で許可されている場合は、インターネット経由でのアクセスも可能です。グラフィカルインタフェースを使用して、監視対象の値をリアルタイムで参照したり、アラームを表示および確認したり、カスタムアラームを生成するためのルールを作成したりできます。

注 – インストールまたは設定を実行する前に、『Sun Java Enterprise System 5 リリースノート (UNIX 版)』を参照することをお勧めします。

この章の内容は次のとおりです。

Monitoring Console のインストール

このリリースでのマスターエージェントの制限により、ノードエージェントと同じホスト上にマスターエージェントを作成することはできません。結果として、Java ES のほかの監視対象コンポーネントと同じホスト上には Monitoring Console をインストールできません。Solaris ゾーンを設定した場合を除き、Monitoring Console は

その専用ホストにインストールする必要があります。詳細は、この章の53 ページの「Solaris ゾーンに Monitoring Console をインストールするには」を参照してください。

Monitoring Console をインストールすると、依存する共有コンポーネントとして Monitoring Framework もインストールされます。コンソールは、マスターエージェントをロードするためにフレームワークおよび共通エージェントコンテナを必要としますが、ノードエージェントと異なり、マスターエージェントはユーザーによる設定はできません。特に、Monitoring Console をインストールするホスト上またはゾーン内では `mfwkadm` コマンドを使用しないことをお勧めします。

▼ Java ES インストーラを使用して Monitoring Console をインストールするには

Monitoring Console は、ほかの Java ES コンポーネントがインストールされていないホストまたは Solaris ゾーンにインストールする必要があります。結果として、Monitoring Console はこの手順でインストールするただ 1 つのコンポーネントです。

この手順では、インストーラのグラフィカルインタフェースを使用します。その他のモードでのインストーラの実行については、『Sun Java Enterprise System 5 インストールガイド (UNIX 版)』の第 4 章「テキストベースのインタフェースによるインストール」および『Sun Java Enterprise System 5 インストールガイド (UNIX 版)』の第 5 章「サイレントモードでのインストール」を参照してください。

1. **Java ES** リリース内の、使用しているプラットフォームに対応するディレクトリから `installer` アプリケーションを起動します。詳細は、『**Sun Java Enterprise System 5** インストールガイド **(UNIX** 版**)**』の「インストールを開始するには」を参照してください。

2. 開始画面から次に進み、ライセンス条項を承諾したあとで、「アップグレードまたはインストールの選択」で「新しいソフトウェアのインストール」を選択して「次へ」をクリックします。

3. コンポーネント選択画面で、**Sun Java System Monitoring Console** のみをインストール対象として選択します。「次へ」をクリックします。

4. 共有コンポーネントに必要なアップグレードがあるかどうかがチェックされます。チェックが終了したら「次へ」をクリックします。

5. システム要件がチェックされます。オペレーティングシステムにパッチが必要な場合、インストールを取り消して、必要なパッチをシステムに追加してからこの手順を再開します。パッチを追加する必要がない場合は、「次へ」をクリックします。

6 設定タイプの選択画面で「今すぐ設定」をクリックし、次のカスタム設定画面で「次へ」をクリックします。

7 **Monitoring Console** をインストールする準備ができたら、「次へ」をクリックしてインストールを開始します。**Java ES** の配備をまだ登録していない場合、インストール中に製品登録ウィンドウが開くことがあります。

8 インストールが完了したら、インストールの概要とログを確認し、「インストールが完了しました」をクリックしてインストーラを終了します。

次の手順 54 ページの「Monitoring Console を設定するには」に進みます。

▼ Solaris ゾーンに Monitoring Console をインストールするには

Solaris ゾーンを使用すると、Java ES のほかのコンポーネントと同じ物理ホスト上に Monitoring Console をインストールできます。これらのコンポーネントは大域ゾーン内に配置され、Monitoring Console 用の論理ホストにする疎ルートローカルゾーンを作成します。次の順序で作業を行います。

1 **Monitoring Console** を除くすべての **Java ES** コンポーネントを大域ゾーンにインストールし、設定します。大域ゾーン内の選択したコンポーネントのインストール後設定をすべて完了し、すべてのサーバーインスタンスを実行中の状態にします。

2 大域ゾーンへのインストールの過程で、**Monitoring Framework** が共有コンポーネントとして大域ゾーンにインストールされます。**第 2 章**の手順のうち、インストールしたコンポーネントに該当するものをすべて実行します。

3 同じホスト上に、**Monitoring Console** 用の論理ホストとして疎ルートローカルゾーンを作成します。これは疎ルートゾーンであるため、*mfwk-base* にインストールされた **Monitoring Framework** は認識可能となるはずです **(9 ページの「デフォルトのパスとファイル名」を参照)**。

4 **52 ページの「Java ES インストーラを使用して Monitoring Console をインストールするには」**の手順に従って、疎ルートローカルゾーンに **Monitoring Console** をインストールします。

5 次のコマンドで、疎ルートゾーン内の **Monitoring Framework** を設定します。

```
cd mfwk-base/bin
./mfwksetup -i
```

このコマンドは大域ゾーン内のファイルを使用して、必要な Monitoring Framework 設定ファイルをローカルゾーンに作成します。

次の手順　54 ページの「Monitoring Console を設定するには」に進みます。

▼ Monitoring Console を設定するには

ここでは、独立した物理ホスト上の Monitoring Console を設定する方法について説明します。Solaris ゾーンによって作成された論理ホスト上に Monitoring Console をインストールした場合、コマンドは同じですが、そのゾーンのファイルシステムの内部でコマンドを実行する必要があります。

1. 次のコマンドで、**Monitoring Framework** を使用してマスターエージェントを初期化します。

```
cd mfwk-base/bin
./masetup -i
```

2. 次のコマンドで、共通エージェントコンテナ (cacao) を再起動します。

```
cacaoadm restart
```

▼ Monitoring Console を構成解除するには

ほかのコンポーネントをインストールするホストに Monitoring Console をインストールして設定する場合、Monitoring Framework での競合が原因でそれらのコンポーネントを監視できません。ノードエージェントを使用して新しいコンポーネントを監視するには、Monitoring Console のマスターエージェントを構成解除する必要があります。

- **root** ユーザーとして次のコマンドを実行し、**Monitoring Console** を構成解除します。

```
cacaoadm stop
cacaoadm unregister-module com.sun.mfwk.masteragent.xml
cacaoadm register-module /etc/mfwk-base/xml/com.sun.mfwk.xml
cacaoadm restart
```

インストール済みディレクトリのレイアウト

各オペレーティングシステムでインストールされるパッケージの名前については、『Sun Java Enterprise System 5 インストールリファレンス (UNIX 版)』の第 5 章「インストール可能なパッケージの一覧」を参照してください。次の表は、Monitoring Console パッケージ内のディレクトリについての説明です。9 ページの「デフォルトのパスとファイル名」で説明するように、デフォルトのインストールディレクトリ *MConsole-base* には次の意味があります。

- Solaris システム: `/opt/SUNWjesmc`

- Linux システム: `/opt/sun/jesmc`

表 3–1 Monitoring Console によって使用されるディレクトリおよびファイル

パス	内容の説明
MConsole-base/WEB-INF/classes	Web アプリケーションサーブレットクラス
MConsole-base/WEB-INF/lib	Web アプリケーションの依存 JAR ファイル
MConsole-base/WEB-INF/*.xml	Web アプリケーション記述子
MConsole-base/css	スタイルシートファイル
MConsole-base/html	HTML ファイル
MConsole-base/images	ユーザーインタフェースで使用される GIF 画像ファイル
MConsole-base/js	JavaSript™ ファイル
MConsole-base/*.jsp	JSP (JavaServer Pages™) ファイル
WebConsole-base/prereg/jesmc/*.reg	Monitoring Console 用の Web コンソールファイル

Monitoring Console の起動

Monitoring Console は Web アプリケーションであり、Monitoring Console をインストールしたホストに接続できる任意のブラウザから利用できます。Monitoring Console へのアクセスは、同じホストに自動的にインストールされる Web コンソール経由で行います。次の手順では、Monitoring Console にアクセスして監視対象コンポーネントを参照する方法について説明します。

▼ Monitoring Console を起動するには

1 まず、**Web** コンソール用の **Web** サーバーを再起動する必要があります。**Monitoring Console** をインストールしたホスト上またはゾーン内で、次のコマンドを実行します。

```
/usr/sbin/smcwebserver restart
```

2 **Web** コンソールの起動を待ちます。次のコマンドを使用して、**Web** コンソールの準備ができたかどうかを確認します。

```
/usr/sbin/smcwebserver status
```

次のメッセージが表示されるまで、このコマンドを何回か実行しなければならない場合があります。

```
Sun Java(TM) Web Console is running.
```

3 **Monitoring Console** ホストに接続できる任意のブラウザから、次の **URL** を指定して **Web** コンソールを開きます。**Solaris** ゾーン内に **Monitoring Console** をインストールした場合、*MC-host* はそのホストに付与した論理ホスト名です。

`https://`*MC-host*`.`*domain*`:6789`

4 ブラウザの設定によっては、証明書が信頼できないというメッセージが表示される場合があります。**Web** コンソールにアクセスするには、証明書を信頼する必要があります。

5 ログイン画面で、`root` ユーザーとして、**Monitoring Console** ホスト上の **root** パスワードを使用して **Web** コンソールにログインします。

ログインすると、Web コンソールが提供するすべてのサービスが一覧表示されます。

6 **Monitoring Console** のメインウィンドウを開くには、次の画面例に示すように、「その他」というヘッダーの下の「**Sun Java System Monitoring Console**」をクリックします。

▼ ノードエージェントに接続するには

Monitoring Console を最初に起動するとき、監視対象コンポーネントのホストの場所を指定する必要があります。Java ES 配備内の各ノードエージェントの場所を指定すると、各ノードエージェント内のすべてのコンポーネントが自動的にコンソールに

表示されます。また、新しいホストに Java ES コンポーネントをインストールすることによってあとから配備にコンポーネントを追加した場合、この手順を繰り返す必要があります。

ノードエージェントを追加すると、Monitoring Console はそのエージェントが明示的に削除されるまで、ユーザーがコンソールにアクセスするたびにそのエージェントに再接続します。以前に追加したノードエージェントが応答していない場合は、30 ページの「ノードエージェントを再起動するには」の手順に従います。

1 **Monitoring Console** をインストールする論理ホストと、ノードエージェントおよび監視対象の **Java ES** コンポーネントが含まれているホストの間で日付と時刻を同期します。同期を自動または手動のどちらで行うかに関係なく、各ホスト上の時刻のずれは約 **10** 分以内である必要があります。

2 必要であれば、**Monitoring Console** の左側に表示された階層のルートにある「配備」リンクをクリックすることにより、配備レベルの表示に切り替えます。ここで、右区画内の「ホスト」タブを選択して「追加」をクリックします。

3 表示された「ホストの追加」ダイアログで、次の画面例で示されているように必要な情報を入力します。

- 「ホスト名」: 監視対象コンポーネントを設定したノードエージェントの完全指定ホスト名を入力します。

- 「ポート」: 11164。ただし、ノードエージェントが存在するホスト上で、ほかのポートを使用するように Monitoring Framework を設定した場合を除きます。
- 「遠隔ホストの root パスワード」: ノードエージェントが存在するシステムの root ユーザーのパスワードを入力します。

4 「接続のテスト」をクリックします。

接続情報が正しく、ホストエージェントが設定済みで実行中であれば、接続に成功したというメッセージがダイアログに表示されます。

5 「了解」をクリックして「ホストの追加」ダイアログを閉じます。新しい名前がホストの一覧に表示されます。また、そのホストのノードエージェント内のすべての監視対象コンポーネントが左側の列に表示されます。

6 **Java ES** 配備内の、監視対象コンポーネントがインストールされているすべてのホストに対してこの手順を繰り返します。

次の手順 これで、左側の列に一覧表示されたコンポーネントを参照して、各コンポーネントの運用ステータス、コンポーネントが公開する監視対象属性、コンポーネントが生成したアラームなどを確認できるようになります。

Monitoring Console の使用

この節では、Monitoring Console の操作方法について説明します。

▼ 監視を選択的に無効化および再有効化するには

Java ES の監視機構は、本番システムのパフォーマンスに影響を及ぼさないよう、軽量に動作するように設計されています。ただし、状況によっては、監視値の収集を停止し、インストゥルメンテーションがパフォーマンスにほとんど影響を及ぼさないようにすることが望ましい場合があります。次の手順で説明するように、Monitoring Console には、ホスト単位でこの対処を行うための手段が用意されています。

1 必要であれば、**Monitoring Console** の左画面の階層のルートにある「配備」リンクをクリックすることにより、配備レベルの表示に切り替えます。次に、右区画で「ホスト」タブをクリックします。

「ホスト」タブのテーブルには、Monitoring Console によって監視される Java ES コンポーネントを含むすべてのホストが一覧表示されます。

2 テーブルの左側の列にあるチェックボックスを使用して、監視を停止するすべてのホストを選択します。「ホスト」テーブルの上部にある「無効化」をクリックします。

参考 結果

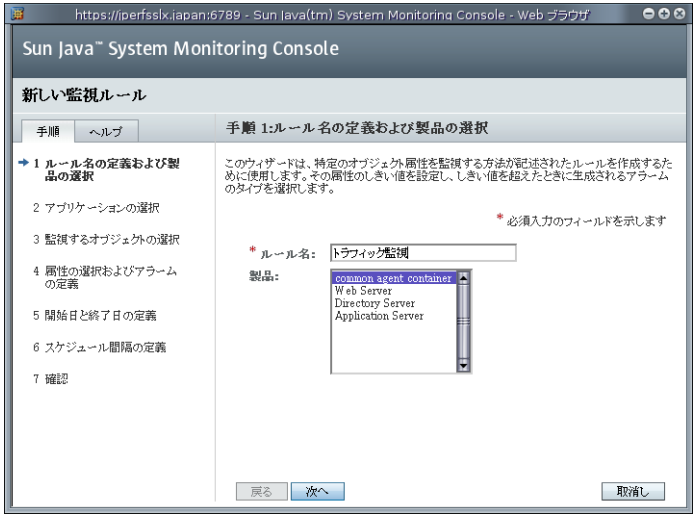
ホストの監視を無効にすると、階層内でそのホストの下にあるすべての監視オブジェクトが無効化されます。無効化された状態の監視対象オブジェクトは、更新はされなくなりますが、無効化する直前の値がまだ含まれている場合があります。無効化されたオブジェクトに依存する監視ルールは停止されます。無効化されたホストを有効にするには、「ホスト」テーブルの上部にある「有効化」ボタンを使用し、同様の手順で設定します。

▼ 新しい監視ルールを作成するには

監視ジョブとも呼ばれる監視ルールは、監視対象の値の条件をセットにしたものであり、アラームを生成する目的でユーザーが定義します。Monitoring Console の監視ルールウィザードを使用すると、監視する条件を簡単に定義できます。

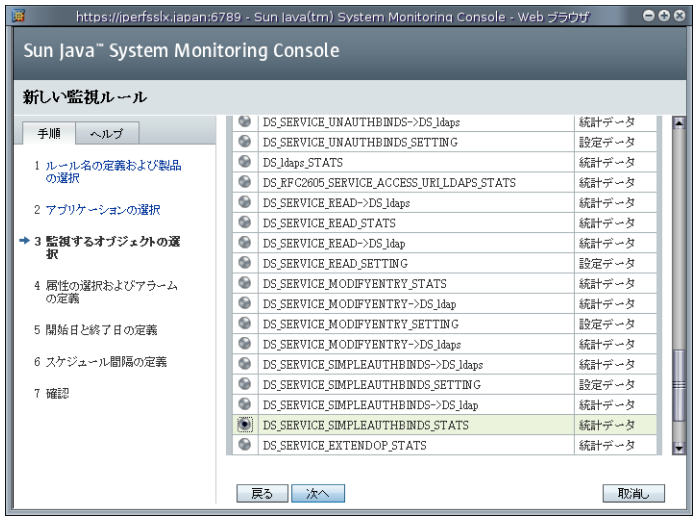
1 必要であれば、**Monitoring Console** の左画面の階層のルートにある「配備」リンクをクリックすることにより、配備レベルの表示に切り替えます。ここで、次の画面例に示すように、右区画にある「監視ルール」タブを選択してから、監視ルールのテーブル内で「新規」をクリックします。

2 新しい監視ルールに名前を付け、監視するサーバーのタイプを選択します。

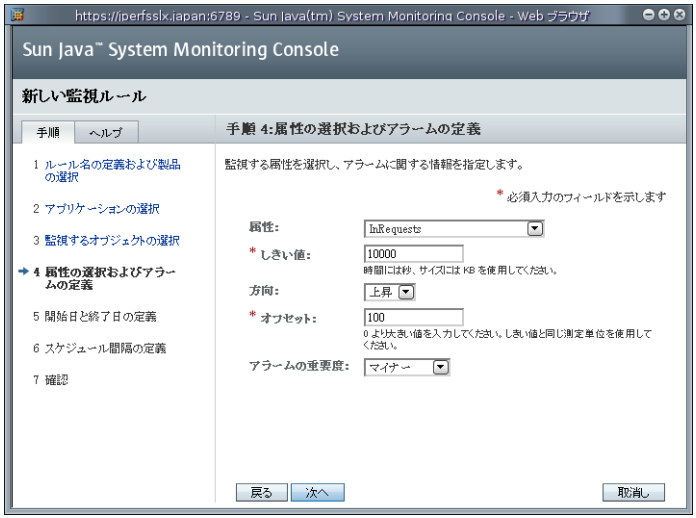


3 監視するコンポーネント製品のインスタンスを選択します。同じ製品の 2 つのインスタンスが別々のホストにインストールされている場合、このテーブル内の一部のインスタンス間で名前が重複する場合があります。そのような場合、左区画の階層内でそれらのインスタンスは同じ順序で表示される可能性がありますが、それらを確実に区別する方法はありません。ルールを確実に定義するために、両方のインスタンスに対して同じ監視ルールを作成しなければならない場合があります。

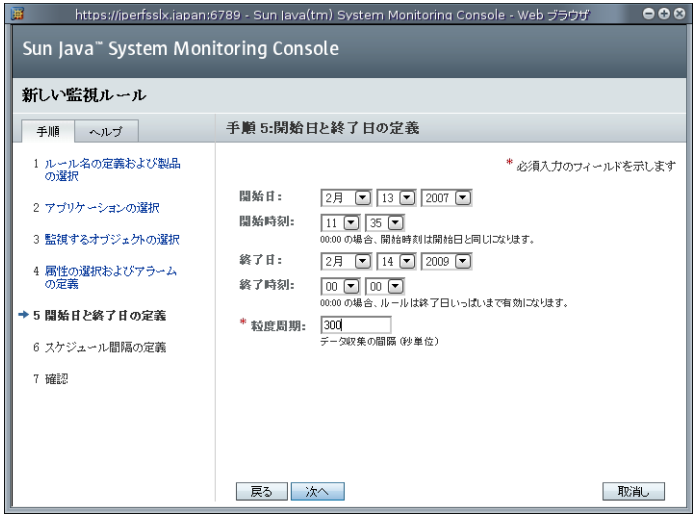
4 監視する属性を含むオブジェクトを選択します。



5 ここで最終的に、監視する属性と、アラームを生成する値を指定できます。



6 ルールの開始日と終了日を入力します。ルールがアクティブかそうでないかを決定するスケジュールとは違い、開始日と終了日は、ルールが存在すると定義される期間の始まりと終わりを決定します。開始時期が過去の場合、そのルールに関連付けられた監視はただちに開始します。これは、すべてのデフォルト値に該当します。



7 コントロールを使用して、ルールがアクティブになり監視を実行する 1 つの以上の期間を必要に応じて作成します。曜日を選択して週次のスケジュールを作成することもできます。

8 ルールウィザードのこの最終ステップでは、入力した内容を確認し、「完了」をクリックして新しいルールを作成します。

ルールウィザードが完了すると、再び「監視ルール」タブが表示され、新しいルールがルールのテーブルに表示されます。

Monitoring Console のトラブルシューティング

『Sun Java Enterprise System 5 リリースノート (UNIX 版)』に記載されている既知の問題も参照してください。

マスターエージェントがノードエージェントと競合している場合、次の条件を確認してください。

- Solaris ゾーンを使用する場合、疎ルートローカルゾーンに Monitoring Console をインストールしたことを確認します。
- 監視対象コンポーネントの以前のインストールが、ホスト上またはゾーン内に残っていないことを確認します。
- いずれの場合にも、Monitoring Console および該当するコンポーネントをすべてアンインストールし、問題を修正してから Monitoring Console のインストールをやり直す必要があります。

Monitoring Console をアンインストールして同じホストに再インストールすると、Monitoring Console は初期化に失敗し、Web コンソールに表示されません。その場合は、Monitoring Console ホスト上で `masetup –i` コマンドを実行してマスターエージェントを初期化します。次に、55 ページの「Monitoring Console を起動するには」の手順に従います。

監視ルールには、ルールがアクティブ状態のときにしかルールを無効にできないという制限があります。スケジュールにより現在アクティブ状態でないルールを無効にする場合、ルールをただちにアクティブにするようにそのスケジュールを変更するか、またはルールを完全に削除する必要があります。

Windows プラットフォームの制限により、ホスト統計の `handleCount` および `threadCount` の値は常に 0 (ゼロ) です。

◆ ◆ ◆ 付 録 A

CMM オブジェクトリファレンス

Common Monitoring Model (CMM) は、Java プログラミング言語で実装される Common Information Model (CIM) の拡張機能です。CMM は com.sun.cmm.cim.* パッケージの Java インタフェースに組み込まれています。CMM は、CIM インタフェースを拡張する com.sun.cmm.* パッケージのインタフェースに組み込まれています。監視対象オブジェクトはノードエージェント内で、CMM インタフェースを実装するクラス別に表現されます。次の表は、監視可能な属性をオブジェクトのクラス別に示したものです。

CMM オブジェクトの概要

CMM は、そのタイプの監視対象オブジェクトが公開できる属性を定義する、コアインタフェースの限られた集合に基づいています。次の一覧は、CMM によって定義される各タイプの監視対象オブジェクトを表すクラスと、各クラスの主な属性の一部を示します。

CMM_InstalledProduct	Java ES コンポーネント製品全体。たとえば、Java ES Directory Server などです。
CMM_ApplicationSystem	Java ES コンポーネント製品の、インストールおよび設定されたインスタンス。このインスタンスは、実行中であっても実行中でなくてもかまいません。このオブジェクトの一般的な属性には、管理者の連絡先情報、システムの運用ステータス、アプリケーションの起動および停止時刻などがあります。
CMM_Service	コンポーネント製品の特定の機能。たとえば、Java ES Directory Server 認証サービスなど。一般的な属性には、サービスの運用ステータスなどがあります。

CMM_SoftwareResource	キャッシュやスレッドプールなどの、環境内のソフトウェアエンティティーの表現。一般的な属性には、キャッシュサイズなどがあります。
CMM_LogicalComponent	サービスによって操作され、エンドユーザーが参照できるが、実際の物理リソースやソフトウェア機能は表現しないエンティティー。たとえば、ソフトウェアインスタンス自体ではなく、インスタンスの設定パラメータの集合などがこれに該当します。
CMM_ServiceAccessURI	サービスを公開して使用できるようにするポイント。一般的な属性には、ポート番号や URI (Uniform Resource Identifier) などがあります。
CMM_RemoteServiceAccessPoint	リモート接続のためのアクセスおよびアドレス情報。一般的な属性には、URI や、接続の運用ステータス (オープンまたはクローズ) などがあります。
CMM_Process	実行中プログラムの単一のインスタンス。一般的な属性には、メモリーまたは CPU の使用状況などがあります。
CMM_UnitaryComputerSystem	デスクトップマシンやサーバーなどの、Java ES 配備によって使用される単一のホスト。一般的な属性には、使用可能なプロセッサ数や物理メモリーの容量などがあります。
CMM_OperatingSystem	ホストマシンのハードウェアを使用できるようにするソフトウェアまたはファームウェア。一般的な属性には、システム上で利用可能な仮想メモリーの容量などがあります。
CMM_JVM	Java ES サーバーが使用する Java 仮想メモリー。一般的な属性には、Java 仮想マシンのバージョン番号などがあります。
CMM_DatabaseService	ユーザーアクセスの提供など、データベースに代わって実行されるタスク。一般的な属性には、データベース接続の最大許容数などがあります。
CMM_CommonDatabase	特定のタイプのデータベースで共通のプロパティー。一般的な属性には、最新のバックアップの日付などがあります。

◆ ◆ ◆ 付録 B

各コンポーネントが公開する監視対象オブジェクト

この節では、監視機能をサポートする各製品コンポーネントの、インストゥルメント済み CMM オブジェクトの一覧を示します。オブジェクトの属性の一部のみがインストゥルメントされる場合、それらの属性の一覧も示します。

共通エージェントコンテナのインストゥルメンテーション

まだドキュメント化されていません。

Access Manager のインストゥルメンテーション

まだドキュメント化されていません。

Application Server のインストゥルメンテーション

まだドキュメント化されていません。

Calendar Server のインストゥルメンテーション

まだドキュメント化されていません。

Directory Server のインストゥルメンテーション

まだドキュメント化されていません。

Instant Messaging のインストゥルメンテーション

まだドキュメント化されていません。

Messaging Server のインストゥルメンテーション

まだドキュメント化されていません。

Portal Server のインストゥルメンテーション

まだドキュメント化されていません。

Web Server のインストゥルメンテーション

まだドキュメント化されていません。

索引