Sun Java Enterprise System 5 监视指南

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

文件号码 820–1592
2007 年 3 月

070330@17039

目录

前言

本书介绍 Sun Java™ Enterprise System 5 (Java ES) 中新的监视功能。监视由 Sun Java System Monitoring Framework 2.0 和 Sun Java System Monitoring Console 1.0 实现。

本指南中的过程说明了如何为每个安装的组件配置并启用 Monitoring Framework，以及如何在 Monitoring Console 中查看所有受监视的数据。本指南未阐述此框架之外各个组件的日志文件及其他监视机制。

目标读者

本书的目标读者是：

- 需要为 Java ES 部署设计维护计划的软件架构师。
- 执行 Java ES 安装和配置的系统管理员。
- 监视和维护 Java ES 部署的系统管理员和技术人员。

阅读本书之前

您应熟悉下一节中介绍的 Java ES 文档集中的文档。您还应熟悉要监视的 Java ES 组件的设计和运行。

此外，如果计划安装和配置监视组件，必须首先完成所有其他组件的安装。执行任何安装或配置之前，应查阅《适用于 UNIX 的 Sun Java Enterprise System 5 发行说明》。

Java ES 文档集

Java ES 文档集介绍部署规划和系统安装。系统文档的 URL 是 `http://docs.sun.com/coll/1286.2` 和 `http://docs.sun.com/coll/1382.2`。有关 Java ES 的简介，请参阅下表中按顺序列出的书籍。

表 P–1 Java Enterprise System 文档

文档标题	内容
《适用于 UNIX 的 Sun Java Enterprise System 5 发行说明》 《Sun Java Enterprise System 5 Release Notes for Microsoft Windows》	含有有关 Java ES 的最新信息，包括已知问题。此外，各个组件具有独立的发行说明，它们在发行说明集合 (http://docs.sun.com/coll/1315.2 和 http://docs.sun.com/coll/1396.2) 中列出。
《Sun Java Enterprise System 5 技术概述》	介绍基本的 Java ES 技术和概念信息。描述组件、体系结构、过程和功能。
《Sun Java Enterprise System Deployment Planning Guide》	介绍如何基于 Java ES 规划和设计企业部署解决方案。介绍部署规划和设计的基本概念及原理，讨论解决方案的生命周期，并提供基于 Java ES 规划解决方案时使用的高级示例和策略。
《Sun Java Enterprise System 5 安装规划指南》	帮助您形成 Java ES 部署的硬件、操作系统和网络方面的实施规范。介绍在安装和配置规划中要解决的一些问题，如组件依赖性问题。
《适用于 UNIX 的 Sun Java Enterprise System 5 安装指南》 《Sun Java Enterprise System 5 Installation Guide for Microsoft Windows》	指导您完成安装 Java ES 的过程。还讲述了在安装后如何配置组件，及如何确定各组件运行正常。
《适用于 UNIX 的 Sun Java Enterprise System 5 安装参考》	提供有关配置参数的其他信息和配置计划中使用的工作表，并列出参考材料（如 Solaris 操作系统和 Linux 操作环境中的默认目录和端口号。）
适用于 UNIX 的 Sun Java Enterprise System 5 升级指南 《Sun Java Enterprise System 5 Upgrade Guide for Microsoft Windows》	提供从先前安装的版本升级到 Java ES 5 的说明。
《Sun Java Enterprise System 5 监视指南》	提供为每个产品组件设置 Monitoring Framework 以及使用 Monitoring Console 查看实时数据并创建监视规则的说明。
《Sun Java Enterprise System Glossary》	定义 Java ES 文档中使用的术语。

默认路径和文件名

下表说明了实现监视的 Java ES 组件的默认路径和文件名。

表 P–2 默认路径和文件名

占位符	说明	默认值
mfwk-base	表示自动安装 Monitoring Framework 共享组件的目录。此路径还用作配置目录的一部分。	Solaris 系统：`/opt/SUNWmfwk` Linux 系统：`/opt/sun/mfwk`
MConsole-base	表示为 Monitoring Console 选择的安装目录。	Solaris 系统：`/opt/SUNWjesmc` Linux 系统：`/opt/sun/jesmc`
WebConsole-base	表示自动安装 Web Console 共享组件的目录。	Solaris 系统：`/etc/webconsole/console` Linux 系统：`/etc/opt/webconsole/console`
AccessMgr-base	表示为 Sun Java System Access Manager 选择的安装目录。	Solaris 系统：`/opt/SUNWam` Linux 系统：`/opt/sun/identity`
AppServer-base	表示为 Sun Java System Application Server 选择的安装目录。	Solaris 系统：`/opt/SUNWappserver/appserver` Linux 系统：`/opt/sun/appserver`
CalServ-base	表示为 Sun Java System Calendar Server 选择的安装目录。	Solaris 系统：`/opt/SUNWics5` Linux 系统：`/opt/sun/calendar`
DirServ-base	表示为 Sun Java System Directory Server 选择的安装目录。	Solaris 系统：`/opt/SUNWdsee/ds6` Linux 系统：`/opt/sun/ds6`
IM-base	表示为 Sun Java System Instant Messaging 选择的安装目录。	Solaris 系统：`/opt/SUNWiim` Linux 系统：`/opt/sun/im`
MsgServ-base	表示为 Sun Java System Messaging Server 选择的安装目录。	Solaris 系统：`/opt/SUNWmsgsr` Linux 系统：`/opt/sun/messaging`
Portal-base	表示为 Sun Java System Portal Server 选择的安装目录。	Solaris 系统：`/opt/SUNWportal` Linux 系统：`/opt/sun/portal`
WebServer-base	表示为 Sun Java SystemWeb Server 选择的安装目录。	Solaris 系统：`/opt/SUNWwbsvr7` Linux 系统：`/opt/sun/webserver`

印刷约定

下表介绍了本书所采用的印刷约定。

表 P–3 印刷约定

字体	含义	示例
`AaBbCc123`	命令、文件和目录的名称；计算机屏幕输出	编辑 `.login` 文件。 使用 `ls -a` 列出所有文件。 `machine_name% you have mail.`
`AaBbCc123`	用户键入的内容，与计算机屏幕输出的显示不同	`machine_name%` **`su`** `Password:`
AaBbCc123	保留未译的新词或术语以及要强调的词。要使用实名或值替换的命令行变量。	要删除文件，请键入 `rm` *filename*。
新词术语强调	新词或术语以及要强调的词。	阅读《用户指南》的第 6 章。 **高速缓存**是本地存储的副本。 **切勿**保存文件。
《书名》	书名	阅读《用户指南》的第 6 章。

命令中的 Shell 提示符示例

下表给出了默认的系统提示符和超级用户提示符。

表 P–4 Shell 提示符

Shell	提示符
UNIX 和 Linux 系统上的 C shell	`machine_name%`
UNIX 和 Linux 系统上的 C shell 超级用户	`machine_name#`
UNIX 和 Linux 系统上的 Bourne shell 和 Korn shell	`$`
UNIX 和 Linux 系统上的 Bourne shell 和 Korn shell 超级用户	`#`
Microsoft Windows 命令行	`C:\`

符号约定

下表说明本书中可能用到的一些符号。

表 P–5 符号约定

符号	说明	示例	含义
[ ]	包含可选参数和命令选项。	ls [-l]	-l 选项不是必需的。
{ | }	包含所需命令选项的一组选择。	-d {y|n}	-d 选项要求您使用 y 参数或者 n 参数。
${ }	表示变量引用。	${com.sun.javaRoot}	引用变量 com.sun.javaRoot 的值。
-	结合同时发生的多个击键。	Control-A	按 A 键的同时按 Control 键。
+	结合相继发生的多个击键。	Ctrl+A+N	按 Control 键后松开，然后按后续各键。
→	表示图形用户界面中的菜单项选择。	“文件”→“新建”→“模板”	从“文件”菜单中选择“新建”。 从“新建”子菜单中选择“模板”。

文档、支持和培训

Sun Web 站点提供了有关以下附加资源的信息：

- 文档 (http://www.sun.com/documentation/)
- 支持 (http://www.sun.com/support/)
- 培训 (http://www.sun.com/training/)

搜索 Sun 产品文档

除了从 docs.sun.com℠ Web 站点搜索 Sun 产品文档以外，还可以使用搜索引擎进行搜索，方法是在搜索字段中键入以下语法：

search-term site:docs.sun.com

例如，要搜索 "broker"，请键入以下内容：

broker site:docs.sun.com

要在搜索中包含其他 Sun Web 站点（例如，java.sun.com、www.sun.com 和 developers.sun.com），请在搜索字段中使用 sun.com 替换 docs.sun.com。

第三方 Web 站点引用

本文档引用第三方 URL，并提供其他相关信息。

注 – Sun 对本文档中提到的第三方 Web 站点的可用性不承担任何责任。对于此类站点或资源中的（或通过它们获得的）任何内容、广告、产品或其他资料，Sun 并不表示认可，也不承担任何责任。对于因使用或依靠此类站点或资源中的（或通过它们获得的）任何内容、产品或服务而造成的或连带产生的实际或名义损坏或损失，Sun 概不负责，也不承担任何责任。

Sun 欢迎您提出意见

Sun 致力于提高其文档的质量，并十分乐意收到您的意见和建议。要共享您的意见，请访问 http://docs.sun.com，然后单击 "Send Comments"（发送意见）。请在联机表单中提供完整的文档标题和文件号码。文件号码包含 7 位或 9 位数字，可以在书的标题页或文档的 URL 中找到该号码。例如，本书的文件号码是 820-1592。

◆ ◆ ◆ 第 1 章

Java ES 监视概述

本书介绍了 Sun Java™ Enterprise System (Java ES) 的 Monitoring Framework 2.0 和 Monitoring Console 1.0 组件。这些组件一起实现版本 5 中引入的新的监视功能。

本指南中的过程说明了如何为每个安装的组件启用 Monitoring Framework，以及如何在 Monitoring Console 中查看所有受监视的数据。本指南未阐述此框架之外各个组件可能实现的日志文件、错误消息及其他监视机制。Monitoring Framework 和 Monitoring Console 都不提供对受监视的组件进行管理的功能。有关管理某个组件的信息，请参见该产品附带的文档。

本章介绍监视概念并说明 Monitoring Framework 的体系结构。

本章包括以下各节：

Monitoring Framework 和 Monitoring Console 组件

Sun Java System Monitoring Framework 提供用于配置组件并公开其属性以便观察的基础结构。它根据行业标准通用信息模型 (Common Information Model, CIM) 规范定义受监视对象的分层结构，该结构称为**通用监视模型** *(Common Monitoring Model, CMM)*。每个产品组件公开了提供其可观察属性的对象，**节点代理**聚合一个主机上多个组件的视图。Monitoring Framework 还提供了收集操作统计信息以及基于用户定义的阈值定义报警的机制。

Sun Java System Monitoring Console 是用于监视 Java ES 组件的图形界面。它包括一个连接到 Java ES 部署中所有节点代理的**主代理**。Monitoring Console 是基于 Web 的应用程序，它依赖于可在任何位置通过 HTTP 访问的 Sun Java System Web Console。在主屏幕上，它提供所有已启用的组件的汇总状态，包括所有已触发的报警。这样，您可以访

问每个组件中受监视对象的分层结构，并且可以查看所有受监视属性的详细状态和实时值。通过 Monitoring Console 界面，可以显示任何报警的详细信息，确认该信息，或者基于任何属性创建新的监视规则。

Java ES 监视的工作方式

监视是收集运行时数据，公开这些数据，以及计算服务质量标准以便系统管理员可以评估性能并接收报警通知的整个过程。在运行时操作过程中，管理员仅需与 Monitoring Console 交互即可查看性能统计信息、创建自动监视规则以及确认报警。但是，了解 Monitoring Framework 的体系结构以及 Monitoring Framework 如何连接到 Monitoring Console 对执行配置、故障排除和高级监视是有所帮助的。

Java ES 中的监视基于以下概念：

- 通用监视模型 (Common Monitoring Model, CMM) 确保所有 Java ES 组件公开统一对象和可比较属性的值。
- CMM 接口定义的 Java 对象为产品组件提供标准化的配置。
- 节点代理公开系统上安装的所有组件的所有受监视对象，并管理这些对象的统计信息、规则和报警。
- 独立主机上的主代理聚合所有节点代理的所有受监视对象，并使数据可为 Monitoring Console 使用。

以下各节分别对监视体系结构的这些概念进行了更详细的介绍。

通用监视模型 (Common Monitoring Model, CMM)

标准化监视机制的基础是定义受监视的对象并在所有受监视的组件中应用这些对象。为此，监视体系结构将通用监视模型 (Common Monitoring Model, CMM) 定义为由分布式管理任务组 (Distributed Management Task Force, DMTF) 维护的通用信息模型 (Common Information Model, CIM) 的扩展。CMM 既是信息模型（指定受监视的对象，如计算机、应用程序等），也是数据模型（指定统一值，如操作状态值）。作为信息模型的一部分，CMM 还定义对象的属性（例如由服务处理的请求数）以及对象之间的关系（如在某台计算机上托管某个服务的事实）。

由于 CMM，一些概念（如应用程序、服务、访问点等）对于所有产品组件都是相同的，即使底层实现不同也如此。例如，Web Server 可能公开用于处理 HTTP 请求的服务，而 Directory Server 可能公开用于处理 LDAP 请求的服务。但是，标准对象会捕获这两种功能中的通用功能，例如度量已处理的请求数和给定时间段中响应请求的平均次数的功能，等等。

而且，某些数据值已标准化，这样其含义在整个系统中是统一的。例如，不论正在监视的是什么产品组件，操作状态 DEGRADED 始终意味着服务仍然可用，但性能已极大下降。

CMM 规范体现在用于配置的 Java 接口和类中，如附录 A 中所述。

CMM 配置

在 Monitoring Framework 中，配置是实现 CMM 定义的一组 Java 接口和类。为了实现 Java ES 中新的监视功能，产品组件已配置了其代码，这些代码用于实例化 CMM 对象，并通过受监视对象的属性公开运行时值。每个组件实现的 CMM 对象确定可以监视的内容，因此，一些组件公开的属性可能少于其他组件。附录 B 中给出了每个产品组件公开以便进行监视的对象和属性的列表。

节点代理

在监视术语中，节点是由唯一的全限定域名或 IP 地址标识的单个逻辑主机。节点可以是整个系统，也可以是配置为虚拟系统的 Solaris zone。节点代理与该主机上所有配置的组件通信，并公开这些节点的所有受监视的对象。节点代理还管理所有用来收集性能统计信息、监视规则中定义的阈值以及为节点代理包含的受监视对象生成报警的逻辑。

下图显示了具有三个 Java ES 产品组件实例的单个主机上的节点代理的内容。它还显示了如何在节点代理中实例化配置，以便公开由产品组件提供的值。

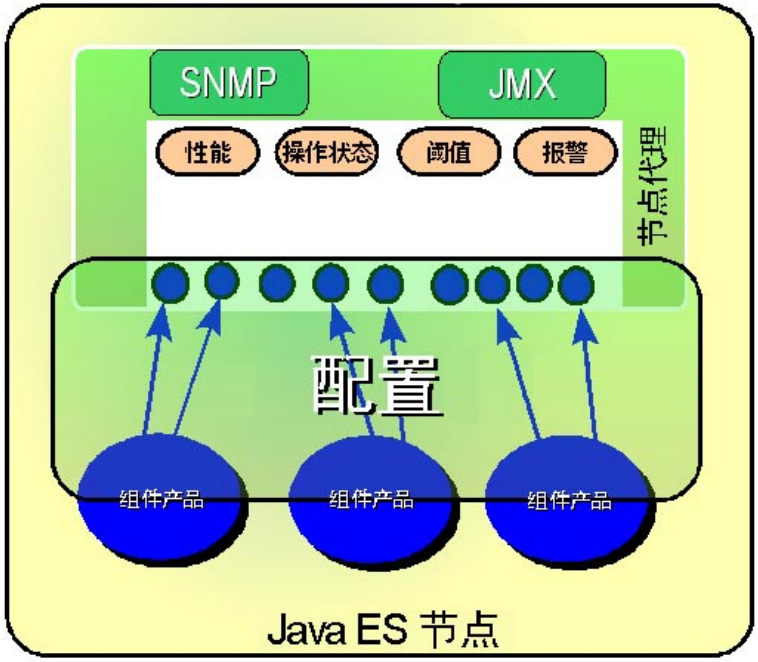

图 1–1 节点代理图

该节点代理被实现为加载到 Common Agent Container（本身为 Java 虚拟机）中的一个模块。节点代理的实现基于 Java Management Extensions (JMX)（用于监视和远程管理的标准 Java 扩展）。任何启用了 JMX 且理解 CMM 的监视应用程序都可以访问节点代理中的受监视对象。使用 JMX 功能时，节点代理还可以通过简单网络监视协议 (Simple Network Monitoring Protocol, SNMP) 公开某些受监视的对象。

主代理

作为 Monitoring Console 安装的一部分，主代理被部署在一台独立的计算机上。主代理中配置了所有节点的名称和地址，这样它可以聚合所有节点代理的受监视对象。主代理也基于 JMX，主代理使用 JMX（也被加载到主代理的本地 Common Agent Container 中）与节点代理通信。

下图显示了连接到两个节点的主代理。Monitoring Console 连接到主代理以监视每个节点上的三个组件。如果希望使用 SNMP 进行监视，必须分别连接到每个节点，因为主代理不聚合 SNMP 属性。主代理被设计为仅与 Monitoring Console 配合使用，不能通过其他监视应用程序访问主代理。

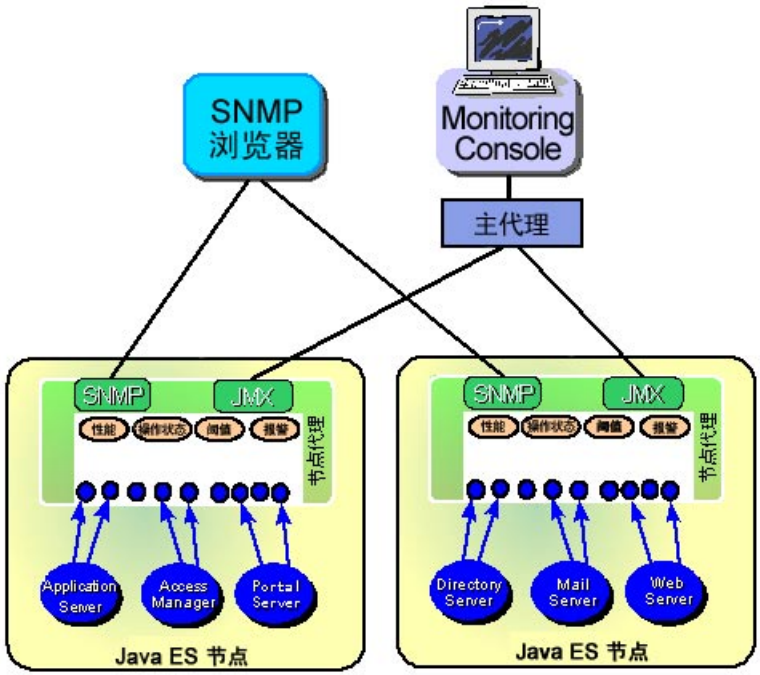

图 1–2 总体监视体系结构图

建议的安装顺序

如果选择评估或部署 Java ES 的监视功能，按以下顺序执行安装最简单：

1. 按照《适用于 UNIX 的 Sun Java Enterprise System 5 安装指南》中的建议和说明在部署中安装并配置所有组件。
2. 按照第 2 章中的说明为所有受监视的组件启用并配置 Monitoring Framework。
3. 按照第 3 章中的说明在单独的主机上安装 Monitoring Console，启动主代理，然后启动 Web Server。此时，所有受监视的组件都应在 Monitoring Console 中可见并受到监视。

注 – 由于此版本中的节点代理和主代理之间不兼容，因此必须将 Monitoring Console 安装在不包含任何其他 Java ES 组件的主机上。有关更多详细信息，请参见第 59 页中的“Monitoring Console 故障排除”。

启用监视后，只要修改部署的组件，就需要重新启动主代理的容器和 Monitoring Console 的 Web 服务器，如第 27 页中的“Monitoring Framework 故障排除”中所述。

◆ ◆ ◆ 第 2 章

2 启用和配置 Monitoring Framework

如第 14 页中的 “Java ES 监视的工作方式”中所述，Monitoring Framework 可为每个受监视的组件提供所需的配置和节点代理。因此，Monitoring Framework 是一个共享组件，在使用 Java Enterprise System 安装程序安装受监视的组件时，会自动安装它。

但是，默认情况下，许多受监视的组件没有启用监视功能，并且某些组件需要进一步配置才能显示在节点代理中。对于已安装的各个产品组件，请按照本章中的过程进行操作。

注 – 执行本章中的任何过程之前，建议安装和配置要在指定主机上运行的所有产品组件。进行任何安装或配置之前，应参阅《适用于 UNIX 的 Sun Java Enterprise System 5 发行说明》。

这些过程使用了 `mfwksetup` 命令（一般无需此命令，因而未记录）。

本章包括以下各节：

安装的目录布局

Monitoring Framework 是一种共享组件，只要需要，就可以自动安装。有关操作系统中安装的软件包的名称，请参见《适用于 UNIX 的 Sun Java Enterprise System 5 安装参考》中的第 5 章“可安装软件包列表”。下表介绍了 Monitoring Framework 软件包中的目录。默认安装目录 *mfwk-base* 具有以下含义，如第 9 页中的 “默认路径和文件名”中所述：

- Solaris 系统：`/opt/SUNWmfwk`
- Linux 系统：`/opt/sun/mfwk`

表 2–1 Monitoring Framework 使用的目录

路径	内容说明
mfwk-base`/config`	配置文件模板
Solaris 系统：*mfwk-base*`/lib`	Java 归档文件 (`.jar`)
Linux 系统：*mfwk-base*`/share/lib`	
Solaris 系统：*mfwk-base*`/lib`	32 位运行时库文件 (`.so`)
Linux 系统：*mfwk-base*`/share/lib`	
Solaris SPARC® 系统：*mfwk-base*`/lib/sparcv9`	64 位运行时库文件 (`.so`)
Solaris x86 系统：*mfwk-base*`/amd64`	
Linux 系统：*mfwk-base*`/lib64`	
mfwk-base`/bin`	公共脚本和专用二进制文件
mfwk-base`/mib`	Monitoring Framework 支持的 SNMP MIB 的文本版本
mfwk-base`/xml`	代理和主代理的 Common Agent Container 描述符模板（通过 `mfwksetup` 命令部署）
mfwk-base`/dtd`	OSS/J 功能的 DTD 文件
`/etc/`*mfwk-base*`/config`	配置文件，包括与安全性相关的文件
`/etc/`*mfwk-base*`/xml`	代理和示例的 Common Agent Container 描述符
`/var/`*mfwk-base*`/logs`	Monitoring Framework 日志文件
`/var/`*mfwk-base*`/reports`	用于监视规则报告的基目录
`/var/`*mfwk-base*`/alarms`	报警文件的系统信息库

在 Access Manager 中使用 Monitoring Framework

默认情况下，Access Manager 中会启用监视功能，但由于受到限制，受监视的对象无法显示在 Monitoring Console 中。

有关可以监视的对象和属性的列表，请参见第 63 页中的 “配置 Access Manager”。

▼ 在 Access Manager 中启用监视功能

1 **使用以下命令在 Access Manager 中临时禁用监视功能：**

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

2 **打开 Access Manager XML 描述符文件进行编辑：**

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

3 **找到包含以下内容的行：**

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

然后将第二行修改为：

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

保存该文件并退出编辑器。

4 **注册已修改的 XML 模块：**

```
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 **重新启动 Common Agent Container：**

```
cacaoadm restart
```

故障排除 由于未对第三方 Web 容器进行行为测试，因此在 Websphere 或 Weblogic 中部署 Access Manager 时，默认情况下会禁用监视功能。尽管不支持此配置，仍可以按照第 51 页中的 “有选择地禁用和重新启用监视”中所述启用监视功能。

在 Application Server 中使用 Monitoring Framework

有关可以监视的对象和属性的列表，请参见第 63 页中的 “配置 Application Server”。

▼ 在 Application Server 中启用监视功能

1 **编辑文件** `/var/`*AppServer-base*`/domains/domain1/config/domain.xml`，**并将所有** `module-monitoring-level` **设置从** `OFF` **更改为** `HIGH`。**或者：**

   a. **登录到位于** `https://`*hostname*`:4849` **的 Application Server 管理控制台**

   b. **选择“配置”，然后选择“服务器配置（管理配置）”**

   c. **将“监视”值设置为** `HIGH`

   d. **将所有其他值都设置为** `HIGH`

2 **使用以下命令重新启动 Application Server：**

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

出现提示后，输入 *myUser* 的密码。

3 **如果使用 Application Server 对 Portal Server 实例进行了部署和监视，重新启动 Application Server 将会影响 Portal Server 监视。要使 Portal Server 实例显示在 Monitoring Console 中，必须在浏览器中访问门户页面。例如，加载页面** `http://portalserv.example.com:8080/portal` **以便允许监视** `portalserv.example.com`。

故障排除 由于受到限制，当 Application Server 崩溃或关闭时，将从 Monitoring Framework 中删除 Application Server 的受监视对象。如果出现这种情况，Application Server 将从 Monitoring Console 中消失，从而无法再受到监视。

在 Calendar Server 中使用 Monitoring Framework

有关可以监视的对象和属性的列表，请参见第 63 页中的 “配置 Calendar Server”。

▼ 在 Calendar Server 中启用监视功能

1. **编辑 `ics.conf` 文件：**

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

2. **添加以下行：**

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

3. **注册 Calendar Server XML 模块：**

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

4. **按如下所示设置 `LD_LIBRARY_PATH` 环境变量：**

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

5. **重新启动 Calendar Server：**

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

6. **重新启动 Common Agent Container：**

   ```
   cacaoadm restart
   ```

在 Directory Server 中使用 Monitoring Framework

有关可以监视的对象和属性的列表，请参见第 64 页中的 “配置 Directory Server”。

▼ 在 Directory Server 中启用监视功能

1. **创建一个临时的密码文件：**

   ```
   echo –n password > /tmp/pwd
   ```

2. **使用以下命令启用监视插件：**

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

3. **重新启动 Directory Server：**

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

在 Instant Messaging 中使用 Monitoring Framework

有关可以监视的对象和属性的列表，请参见第 64 页中的 “配置 Instant Messaging”。

▼ 在 Instant Messaging 中启用监视功能

1. **打开 Instant Messaging XML 描述符文件进行编辑：**

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

2. **将安装位置从 *IM-base* 更改为 `/etc/IM-base/default`。**

3. **注册修改的 Instant Messaging XML 描述符：**

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

4. **通过在文件 *IM-base*`/config/iim.conf` 中添加以下行来启用配置：**

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

5. **使用以下命令重新启动 Instant Messaging：**

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

6. **重新启动 Common Agent Container：**

   ```
   cacaoadm restart
   ```

在 Messaging Server 中使用 Monitoring Framework

有关可以监视的对象和属性的列表，请参见第 64 页中的 “配置 Messaging Server”。

▼ 在 Messaging Server 中启用监视功能

1. **使用以下命令启用配置：**

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

2. **注册 Messaging Server XML 模块：**

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

3 **重新启动 Messaging Server：**

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

4 **重新启动 Common Agent Container：**

```
cacaoadm restart
```

在 Portal Server 中使用 Monitoring Framework

有关可以监视的对象和属性的列表，请参见第 64 页中的 “配置 Portal Server”。

▼ 在 Portal Server 中启用监视功能

- **要启用 Portal Server，用户必须登录到**

  ```
  http://FullHostname:8080/portal/dt
  ```

  这将会编译门户 JSP，以创建可以监视的门户实例。

故障排除 每次重新启动托管 Portal Server 的 Application Server 时，都必须按照此过程手动重新启用监视功能。

在 Web Server 中使用 Monitoring Framework

有关可以监视的对象和属性的列表，请参见第 64 页中的 “配置 Web Server”。

▼ 在 Web Server 中启用监视功能

1 **使用以下命令启动 Web Server：**

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

2 **启动管理服务器：**

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

设置 Common Agent Container

Common Agent Container 是另一个共享组件，也是 Monitoring Framework 赖以运行节点代理的共享组件。Common Agent Container 可能会停止并需要重新启动，具体取决于安装顺序。另外，Common Agent Container 已经过配置，也可以受到监视。有关受监视对象的说明，请参见第 63 页中的 “配置 Common Agent Container” 。

要检查是否已启动 Common Agent Container 和节点代理，请运行以下命令：

```
cacaoadm status
```

如果显示类似以下内容的消息，则表明节点代理正在运行：

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

如果显示类似以下内容的消息，则表明节点代理没有运行：

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

▼ 启用对 Common Agent Container 的监视

Common Agent Container 是一个共享组件，具有允许监视的配置。如第 15 页中的 “节点代理” 中所述，主机或区域中的所有 Java ES 组件都共享 Common Agent Container 和节点代理。请以超级用户身份在部署中要监视 Common Agent Container 的每个逻辑主机上执行此任务。

1. **如果 Common Agent Container 正在运行，请使用以下命令将其停止：**

   ```
   cacaoadm stop
   ```

2. **启用该容器本身的配置：**

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

3. **检查刚设置的参数的值，然后重新启动 Common Agent Container：**

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

4. **创建密钥密码：**

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

5 **生成密钥：**

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

6 **注册 Common Agent Container 自己的监视模块：**

```
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 故障排除

另请参见《适用于 UNIX 的 Sun Java Enterprise System 5 发行说明》中列出的已知问题。

在 HP-UX 平台上使用 Monitoring Framework

默认情况下，不会为 Monitoring Framework 所需的大量任务处理优化 HP-UX 上的 Java 虚拟机 (Java Virtual Machine, JVM)，因此可能导致 `OutOfMemory` 异常。要配置 JVM，请从以下位置下载并运行 `HPjconfig` 工具：`http://h21007.ww2.hp.com/dspp/tech/tech_TechDocumentDetailPage_IDX/1,1701,1620,00.html`。

在 Microsoft Windows 上使用 Monitoring Framework

完全支持通过 Monitoring Framework 在 Windows 平台上监视 Java ES 组件，但与 HP-UX 平台相比，存在某些差异。例如，为避免某些已知问题，必须升级到 Java 1.5 或更高版本。有关其他已知问题，请参见《Sun Java Enterprise System 5 Release Notes for Microsoft Windows》。

▼ 重新启动节点代理

如果需要重新启动托管某节点代理的 Common Agent Container，则在执行以下过程之前，通过该节点代理监视的组件在 Monitoring Console 中将是不可见的：

1 **重新启动托管节点代理的 Common Agent Container：**

```
cacaoadm restart
```

2 **重新启动托管主代理的 Common Agent Container。主代理在主机的 Monitoring Framework 或安装了 Monitoring Console 的区域中运行。**

```
cacaoadm restart
```

主代理将自动与先前监视的所有节点代理重新连接。

3 **重新启动托管 Monitoring Console 的 Web 服务器：**

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

mfwkadm 命令

本节复制了 mfwkadm 命令（手册页 1M 一节中的一种维护命令）的手册页。使用此命令可以管理节点代理的内容，包括所监视组件的全部模块以及在此节点上定义的所有监视规则（也称为作业）。为与本文档中使用的术语和说明相符，已在此处对该手册页中的某些术语和说明进行了修改。

概要

mfwkadm --help

mfwkadm start

mfwkadm stop

mfwkadm restart

mfwkadm list-params

mfwkadm list-modules

mfwkadm info *runningInstance*

性能监视

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*

操作状态监视

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*

阈值监视

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*

说明

mfwkadm 实用程序是一种管理 Monitoring Framework 代理（也称为节点代理）的命令行界面。该节点代理在 Common Agent Container 中运行。mfwkadm 实用程序可用于停止并重新启动节点代理，以及管理所执行的监视作业。应从正在运行节点代理的同一主机上运行此命令。必须遵从此处显示的此命令的参数顺序。

要更改输出消息的语言，请将 LC_MESSAGE 环境变量设置为您使用的**语言环境**。mfwkadm 命令将使用 lib/resources 目录中名为 JesmfMessages_*locale*.pm 的文件包含的消息。如果语言环境没有对应的消息文件或者未指定语言环境，mfwkadm 命令将使用 JesmfMessages.pm 文件中默认的消息集。

mfwkadm 实用程序具有以下子命令。带有星号 (*) 标记的子命令需要运行 Common Agent Container 以及加载节点代理。

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

启动节点代理后，要经过几秒钟至几分钟的延迟才可以使用 mfwkadm 实用程序，具体取决于要装入的 Common Agent Container 模块的数量。如果在此期间使用命令，将会失败，并显示一条显式消息。

选项

支持以下选项：

--help　　显示用法摘要。

子命令

start

启动 Monitoring Framework 节点代理及其关联的组件产品模块，而不停止 Common Agent Container。

此操作首先在 Common Agent Container 中部署节点代理，然后部署关联的组件产品模块。此工具是一种包装器，其优先级高于 cacaoadm 实用程序的 lock 和 undeploy 子命令。

start 子命令仅启动与 Monitoring Framework 关联的节点代理和 Java ES 组件模块。组件模块的前缀为 com.sun.cmm。

安全性：start 子命令只能由启动了 Common Agent Container 的用户来运行。否则，将会显示类似以下内容的错误消息：

```
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`
停止 Common Agent Container 中的 Monitoring Framework 节点代理及其关联的 Java ES 组件模块。

此操作将首先停止 Common Agent Container 中部署的所有 Java ES 组件模块，然后停止节点代理。此工具是一种包装器，其优先级高于 `cacaoadm` 的 `lock` 和 `unlock` 子命令。

`stop` 子命令仅停止与 Monitoring Framework 关联的那些 Java ES 组件模块，然后停止节点代理本身。组件模块的前缀为 `com.sun.cmm`。

安全性：`stop` 子命令只能由启动了 Common Agent Container 的用户来运行。否则，将会显示类似以下内容的错误消息：

```
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`
重新启动 Common Agent Container 中的 Monitoring Framework 节点代理及其关联的 Java ES 组件模块。

此操作将尝试先停止 Common Agent Container 中的节点代理及其关联模块，然后再启动它们，其方式与 `stop` 和 `start` 子命令相同。

安全性：`restart` 子命令只能由启动了 Common Agent Container 的用户来运行。否则，将会显示类似以下内容的错误消息：

```
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`
列出与 Monitoring Framework 节点代理相关的所有配置参数。

安全性：此命令不存在任何用户限制。

`list-modules`
显示实现通用监视模型 (Common Monitoring Model, CMM) 以及加载到 Common Agent Container 中的那些组件产品模块的列表。此子命令还将列出每个已安装的 Java ES 组件的所有正在运行的实例。每个组件都可以拥有零个、一个或多个正在运行的实例。

安全性：对于非启动 Common Agent Container 的用户，已安装的 Java ES 组件列表不包括组件实例。

`info` *runningInstance*
显示有关指定 *runningInstance* 的信息。*runningInstance* 必须与 `list-modules` 子命令输出中列出的正在运行的实例匹配。 显示的信息包括：

- 对于每种监视作业，显示与正在运行的实例关联的所有可观察对象（按其类名排序）。可观察对象即可以分别使用 `pm-job`、`opstat-job` 或 `thrsh-job` 子命令对其创建性能监视作业、操作状态作业或阈值监视作业的对象。
- 对于每类可观察对象，显示其所有可观察属性，包括每类对象的名称和类型。

安全性：对于非启动 Common Agent Container 的用户，不会显示任何信息。

性能监视

`pm-job observable-classes`
显示可以创建性能监视作业的对象的所有当前可观察类的列表。

`pm-job observable-objects` [class= *objectC lass*] [domain=*objectDomain*]
显示可以创建性能监视作业的所有当前可观察对象的列表。默认情况下，将会列出所有可观察类的所有对象以及每个域中的所有对象。对象列表按其类名排序。

class=*objectClass*
指定可选的 *objectClass* 可将输出限制为该特定类的可观察对象。*objectClass* 必须为通过 `pm-job observable-classes` 子命令列出的类之一。

domain=*objectDomain*
指定可选的 *objectDomain* 可将输出限制为该域中的可观察对象。对象的域为对象的名称中冒号 (":") 字符前面的字符串。

`pm-job observable-attributes class=`*objectClass*
显示指定的 *objectClass* 的所有可观察属性的列表。将显示属性及其名称和类型。*objectClass* 必须为支持性能监视作业的类（使用 `pm-job observable-classes` 子命令可以列出这些类）之一。

`pm-job list`
显示所有当前定义的性能监视作业的列表。作业将针对具有定义的性能作业的每个对象列出，并且对象按其类名排序。显示的有关每个作业的信息与通过 `pm-job info` 子命令显示的信息相同。

安全性：对于非启动 Common Agent Container 的用户，不会显示任何作业。

`pm-job info` *jobName*
显示有关名为 *jobName* 的性能监视作业的详细信息。*jobName* 必须为通过 `pm-job list` 子命令显示的作业。此子命令显示的信息如下：

- 性能监视作业的名称。
- 性能监视作业的类型（“按对象分类”或“按类分类”）。按对象分类的作业监视一个或多个指定的对象实例，而按类分类的作业监视对象类的每个实例。请注意，无法使用 `mfwkadm` 实用程序创建按类分类的作业。

- 性能监视作业的状态：活动在工作、活动未工作或已暂停。活动在工作的作业当前已预定运行并正在收集数据。活动未工作的作业正在运行，但没有收集数据，因为当前时间不在其工作调度内。已暂停的作业没有运行，也没有收集任何数据。使用 `pm-job suspend` 和 `pm-job resume` 子命令可以更改性能监视作业的运行状态。
- 性能监视作业的粒度（以秒为单位）。这是此作业的数据收集时间间隔。
- 监视作业的报告周期。报告周期乘以粒度等于通知频率。例如，如果粒度周期为 10 秒，报告周期为 6，按事件报告的作业将每 10 秒收集一次数据，并且每 60 秒 (10*6) 发送一个包含 6 份报告的通知。如果该作业同时按文件报告，则还会每 60 秒发送一个包含 6 个所生成文件的位置的事件。
- 性能监视作业是否在按事件进行报告。这意味着，性能监视作业的结果将以通知方式发送至注册的客户机。
- 性能监视作业是否在按文件进行报告。这意味着，性能监视作业的报告将写入本地文件，并且包含文件名的通知将发送至注册的客户机。
- 性能监视作业的报告格式，该格式始终为 `XML`。
- 性能监视作业的调度。该调度指定在什么日期和时间，作业处于活动在工作状态或活动未工作状态（收集数据或不收集数据）。 对于按对象分类的作业：
- 被观察对象的列表，按名称排序。
- 如果仅指定了可观察属性的子集，将按名称和类型列出被观察对象的被观察属性。 对于按类分类的作业：
- 被观察类的列表，按名称排序。
- 如果仅指定了可观察属性的子集，将按名称和类型列出被观察类的被观察属性。这些属性为所有类公用。

安全性：对于非启动 Common Agent Container 的用户，不会显示任何信息。

`pm-job create` *jobName* granularity= *integerValue* object=*objectName* [object= *objectName* ...]
在一个或多个对象上创建新的性能监视作业。`mfwkadm` 命令无法创建按类分类的作业。创建性能监视作业时，可以设置以下参数：

jobName
唯一标识性能监视作业的字符串。*jobName* 不能已被任何其他性能监视作业使用。

granularity=*integerValue*
当作业处于活动在工作状态时，连续两次启动测量数据收集之间的指定时间（以秒为单位）。粒度周期示例可以是 300 秒（5 分钟）、900 秒（15 分钟）、1800 秒（每半小时）或 3600 秒（每小时）。在大多数情况下，300 秒的粒度周期就足够了。对于某些测量，采用更大的粒度周期收集数据可能更有意义。

object=*objectName* [object= *objectName* ...]
性能监视作业将收集其数据并进行报告的一个或多个可观察对象。*objectName* 必须为通过 `pm-job list` 或 `pm-job observable-objects` 子命令显示的对象。指定多个 object= *objectName* 参数将会创建一个监视多个对象的性能监视作业。

安全性：此子命令只能由启动了 Common Agent Container 的用户来运行。

`pm-job delete` *jobName*
删除名为 *jobName* 的性能监视作业。*jobName* 必须为通过 `pm-job list` 子命令显示的作业。

安全性：此子命令只能由启动了 Common Agent Container 的用户来运行。

`pm-job suspend` *jobName*
暂停名为 *jobName* 的性能监视作业。无论已暂停的作业的调度如何，该作业都处于非活动状态并不再收集数据。但是，该作业将保持已定义状态，并且可以通过 `pm-job resume` 子命令重新激活。*jobName* 必须为通过 `pm-job list` 子命令显示的作业之一。

安全性：此子命令只能由启动了 Common Agent Container 的用户来运行。

`pm-job resume` *jobName*
恢复名为 *jobName* 的性能监视作业。恢复的作业将根据其调度开始收集数据并发送报告。*jobName* 必须为通过 `pm-job list` 子命令显示的作业之一。此子命令与 `pm-job suspend` 子命令对应。

安全性：此子命令只能由启动了 Common Agent Container 的用户来运行。

操作状态监视

`opstat-job observable-classes`
显示可以为其创建操作状态监视作业的对象的所有当前可观察类的列表。

`opstat-job observable-objects` [class= *objectClass*] [domain=*objectDomain*]
显示可以创建操作状态监视作业的所有当前可观察对象的列表。默认情况下，将会列出所有可观察类的所有对象以及每个域中的所有对象。对象列表按其类名排序。

class=*objectClass*
指定可选的 *objectClass* 可将输出限制为该特定类的可观察对象。*objectClass* 必须为通过 `opstat-job observable-classes` 子命令列出的类之一。

domain=*objectDomain*
指定可选的 *objectDomain* 可将输出限制为该域中的可观察对象。对象的域为对象的名称中冒号 (“:”) 字符前面的字符串。

`opstat-job observable-attributes class=` *objectClass*
显示指定的 *objectClass* 的所有可观察属性的列表。将显示属性的名称和类型。*objectClass* 必须为通过 `opstat-job observable-classes` 子命令列出的类之一。

`opstat-job list`
显示所有当前定义的操作状态监视作业的列表。作业将针对具有定义的操作状态作业的每个对象列出，并且对象按其类名排序。显示的有关每个作业的信息与通过 `opstat-job info` 子命令显示的信息相同。

安全性：对于非启动 Common Agent Container 的用户，不会显示任何作业。

`opstat-job info` *jobName*
显示有关名为 *jobName* 的操作状态监视作业的详细信息。*jobName* 必须为通过 `opstat-job list` 子命令显示的作业。此子命令显示的信息如下：

- 操作状态监视作业的名称。
- 操作状态监视作业的类型（“按对象分类”或“按类分类”）。按对象分类的作业监视指定的对象实例，而按类分类的作业监视对象类的每个实例。请注意，无法使用 `mfwkadm` 实用程序创建按类分类的作业。
- 操作状态监视作业的状态：活动在工作、活动未工作或已暂停。活动在工作的作业当前已预定运行并正在收集数据。活动未工作的作业正在运行，但未收集数据，因为当前时间不在其工作调度内。已暂停的作业没有运行，也没有收集任何数据。使用 `opstat-job suspend` 和 `opstat-job resume` 子命令可以更改操作状态监视作业的运行状态。
- 操作状态监视作业的粒度（以秒为单位）。这是此作业的数据收集时间间隔。
- 操作状态监视作业是否在按事件进行报告。这意味着，操作状态监视作业的结果将以通知方式发送至注册的客户机。
- 操作状态监视作业是否在按文件进行报告。这意味着，操作状态监视作业的报告将写入本地文件，并且包含文件名的通知将发送至注册的客户机。
- 操作状态监视作业的报告格式，该格式始终为 `XML`。
- 操作状态监视作业的调度。该调度指定在什么日期和时间，作业处于活动在工作状态或活动未工作状态（收集数据或不收集数据）。
- 对于按对象分类的作业，显示被观察对象的列表，按名称排序。
- 对于按类分类的作业，显示被观察类的列表，按名称排序。

安全性：对于非启动 Common Agent Container 的用户，不会显示任何信息。

`opstat-job create` *jobName* granularity= *integerValue* object=*objectName* [object= *objectName* ...]
在一个或多个对象上创建新的操作状态监视作业。`mfwkadm` 命令无法创建按类分类的作业。创建性能监视作业时，可以设置以下参数：

jobName
唯一标识操作状态监视作业的字符串。*jobName* 不能已被任何其他操作状态监视作业使用。

granularity=*integerValue*
当作业处于活动在工作状态时，连续两次启动测量数据收集之间的指定时间（以秒为单位）。

object=*objectName* [object= *objectName* ...]
操作状态监视作业将收集其数据并报告的一个或多个可观察对象。*objectName* 必须为通过 `opstat-job list` 或 `opstat-job observable-objects` 子命令显示的对象。指定多个 object= *objectName* 参数将会创建一个监视多个对象的操作状态作业。

安全性：此子命令只能由启动了 Common Agent Container 的用户来运行。

`opstat-job delete` *jobName*
删除名为 *jobName* 的操作状态监视作业。*jobName* 必须为通过 `opstat-job list` 子命令显示的作业。

安全性：此子命令只能由启动了 Common Agent Container 的用户来运行。

`opstat-job suspend` *jobName*
暂停名为 *jobName* 的操作状态监视作业。无论已暂停的作业的调度如何，该作业都处于非活动状态并将不再收集数据。但是，该作业将保持已定义状态，并且可以通过 `opstat-job resume` 子命令重新激活。*jobName* 必须为通过 `opstat-job list` 子命令显示的作业。

安全性：此子命令只能由启动了 Common Agent Container 的用户来运行。

`opstat-job resume` *jobName*
恢复名为 *jobName* 的操作状态监视作业。恢复的作业将根据其调度开始收集数据并发送报告。*jobName* 必须为通过 `opstat-job list` 子命令显示的作业。此子命令与 `opstat-job suspend` 子命令对应。

安全性：此子命令只能由启动了 Common Agent Container 的用户来运行。

阈值监视

`thrsh-job observable-classes`
显示可以为其创建阈值监视作业的对象的所有当前可观察类的列表。

`thrsh-job observable-objects` [class= *objectClass*] [domain=*objectDomain*]
显示可以为其创建阈值监视作业的所有当前可观察对象的列表。默认情况下，将会列出所有可观察类的所有对象以及每个域中的所有对象。对象列表按其类名排序。

class=*objectClass*
指定可选的 *objectClass* 可将输出限制为该特定类的可观察对象。*objectClass* 必须为通过 `thrsh-job observable-classes` 子命令列出的类之一。

domain=*objectDomain*
指定可选的 *objectDomain* 可将输出限制为该域中的可观察对象。对象的域为对象的名称中冒号 (“:”) 字符前面的字符串。

`thrsh-job observable-attributes class=`*objectClass*
显示指定的 *objectClass* 的所有可观察属性的列表。将显示属性的名称和类型。*objectClass* 必须为通过 `thrsh-job observable-classes` 子命令列出的类之一。

`thrsh-job list`
显示所有当前定义的阈值监视作业的列表。作业将针对具有定义的阈值作业的每个对象列出，并且对象按其类名排序。显示的有关每个作业的信息与通过 `thrsh-job info` 子命令显示的信息相同。

安全性：对于非启动 Common Agent Container 的用户，不会显示任何作业。

`thrsh-job info` *jobName*

显示有关名为 *jobName* 的阈值监视作业的详细信息。*jobName* 必须为通过 `thrsh-job list` 子命令显示的作业。此子命令显示的信息如下：

- 阈值监视作业的名称。
- 阈值监视作业的多样性。在此版本中，仅允许使用监视一个对象的一种属性的简单阈值作业。
- 阈值监视作业的状态：活动在工作、活动未工作或已暂停。活动在工作的作业当前已预定运行并正在收集数据。活动未工作的作业正在运行，但未收集数据，因为当前时间不在其工作调度内。已暂停的作业没有运行，也没有收集任何数据。使用 `thrsh-job suspend` 和 `thrsh-job resume` 子命令可以更改阈值监视作业的运行状态。
- 阈值监视作业的粒度（以秒为单位）。这是此作业的数据收集时间间隔。
- 阈值监视作业的调度。该调度指定在什么日期和时间，作业处于活动在工作状态或活动未工作状态（收集数据或不收集数据）。
- 阈值监视作业的报警配置。此报警将在受监视的属性的被观察值超出定义的阈值时触发。显示内容包括报警的类型和严重性。
- 阈值监视作业的被观察对象。
- 对其应用阈值的属性名称。
- 触发报警的阈值。
- 在达到阈值时将触发报警的值的变化方向（`RISING` 或 `FALLING`）。
- 阈值的容错偏移。如果方向为 `RISING`，则直到被观察属性的值小于 *thresholdValue-offsetValue* 时才会再次触发报警。如果方向为 `FALLING`，则直到被观察属性的值大于 *thresholdValue+offsetValue* 时才会再次触发报警。即使该偏移为零，此行为也适用。

安全性：对于非启动 Common Agent Container 的用户，不会显示任何信息。

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

创建监视一个对象的一种属性的新阈值监视作业。创建阈值作业时，可以设置以下参数：

jobName

唯一标识阈值监视作业的字符串。*jobName* 不能已被任何其他阈值监视作业使用。

object=*objectName*

阈值监视作业将收集其属性值以便与阈值进行比较的可观察对象。*objectName* 必须为通过 `thrsh-job list` 或 `thrsh-job observable-objects` 子命令显示的对象。

granularity=*integerValue*
当作业处于活动在工作状态时，连续两次启动属性值观察之间的指定时间（以秒为单位）。

attributeName=*attributeName*
阈值监视作业将收集其值并与阈值进行比较的属性的名称。*attributeName* 必须通过 `thrsh-job info` 或 `thrsh-job observable-attributes` 子命令列出。

attributeType=*attributeType*
要监视的可观察属性的类型。*attributeType* 必须通过 `thrsh-job info` 或 `thrsh-job observable-attributes` 子命令列出。

thresholdValue=*thresholdValue*
导致此阈值作业在按 `thresholdDirection` 指定的方向超出时触发报警的受监视属性的值。

thresholdOffset=*offsetValue*
offsetValue 确定在触发连续的报警时阈值作业的容错。*offsetValue* 必须为零或正值。触发某个报警事件后，直到受监视的属性的值超出 *offsetValue* 和 `thresholdDirection` 定义的范围时才会触发新的报警事件。

thresholdDirection=`[RISING|FALLING]`
如果方向为 `RISING`，则直到被观察属性的值小于 *thresholdValue*-*offsetValue* 时才会再次触发报警事件。如果方向为 `FALLING`，则直到被观察属性的值大于 *thresholdValue*+*offsetValue* 时才会再次触发报警事件。即使 *offsetValue* 为零，此行为也适用。

安全性：此子命令只能由启动了 Common Agent Container 的用户来运行。

`thrsh-job delete` *jobName*
删除名为 *jobName* 的阈值监视作业。*jobName* 必须为通过 `thrsh-job list` 子命令显示的作业。

安全性：此子命令只能由启动了 Common Agent Container 的用户来运行。

`thrsh-job suspend` *jobName*
暂停名为 *jobName* 的阈值监视作业。无论已暂停的作业的调度如何，该作业都处于非活动状态并将不再收集数据。但是，该作业将保持已定义状态，并且可以通过 `thrsh-job resume` 子命令重新激活。*jobName* 必须为通过 `thrsh-job list` 子命令显示的作业。

安全性：此子命令只能由启动了 Common Agent Container 的用户来运行。

`thrsh-job resume` *jobName*
恢复名为 *jobName* 的阈值监视作业。恢复的作业将根据其调度开始收集数据并发送报告。*jobName* 必须为通过 `thrsh-job list` 子命令显示的作业。此子命令与 `thrsh-job suspend` 子命令对应。

安全性：此子命令只能由启动了 Common Agent Container 的用户来运行。

示例

以下假定方案说明如何使用 mfwkadm 实用程序及其选项和子命令。

list-modules 子命令显示当前主机上的 Java ES 组件实例及其在 Common Agent Container 中的对应模块。以下示例列出了两个安装的组件：没有任何运行实例的 Directory Server 和具有一个运行实例的 Web Server。

```
$ 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
```

以下 info 子命令显示了 Web Server 实例的可观察对象以及在每种类型的作业中这些对象的类型和可观察属性。

```
$ 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]
        ...
```

以下命令显示了定义的性能监视作业的列表。在此示例中，包含一个称为 `myPerfJob` 的用于监视一个对象的性能作业：

```
$ 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.
```

以下命令将创建一个与通过 `opstat-job info` 或 `opstat-job observable-objects` 子命令获取的两个可观察对象相关的操作状态监视作业：

```
$ 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
```

以下命令将暂停上面创建的作业：

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

以下命令显示了潜在的阈值监视作业的可观察类：

```
$ 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
```

以下命令显示了用于监视上例中 com.sun.cmm.statistics.CMM_SWRQueueStats 类的对象的阈值作业的可观察属性：

```
$ 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]
```

以下命令是另一个作业创建示例，此处将创建一个阈值作业：

```
$ 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
```

以下示例说明了上例中创建的阈值监视作业的 thrsh-job info 子命令的输出：

```
$ 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
```

退出状态

返回的退出值如下：

0　成功完成

1　发生错误

属性

属性类型	属性值
可用性	SUNWmfwk
界面稳定性	Contract Private

另请参见

cacao.5、cacaoadm.1m

◆ ◆ ◆ 第 3 章

3 安装和使用 Monitoring Console

Monitoring Console 是基于 Web 的应用程序，它显示通过配置收集的所有监视数据。该应用程序依赖于主代理来聚合所有来自每个节点代理的值和报警通知。

安装 Monitoring Console 之后，您可以在任何主机上通过一个简单的浏览器窗口安全地访问它，甚至可以通过 Internet 进行访问（如果防火墙已配置为允许该访问）。使用图形界面，可以实时查看受监视的值，查看并确认报警以及创建触发自定义报警的规则。

注 – 执行任何安装或配置之前，应查阅《适用于 UNIX 的 Sun Java Enterprise System 5 发行说明》。

本章包括以下各节：

安装 Monitoring Console

由于此版本中主代理的限制，不能在作为节点代理的同一主机上安装主代理。因此，不能将 Monitoring Console 安装在已安装了任何其他受监视的 Java ES 组件的同一主机上。除非已配置了 Solaris zone，否则必须在独立的主机上安装 Monitoring Console。有关更多信息，请参见下面第 46 页中的 “在 Solaris zone 中安装 Monitoring Console”

安装 Monitoring Console 时，还会将 Monitoring Framework 作为共享组件相关项进行安装。控制台需要使用框架和 Common Agent Container 来加载主代理，但与节点代理不同，主代理不可由用户配置。特别是不应在安装 Monitoring Console 的主机或区域中使用 `mfwkadm` 命令。

▼ 使用 Java ES 安装程序安装 Monitoring Console

由于此 Beta 版中的限制，必须在未安装任何其他 Java ES 组件的主机或 Solaris zone 中安装 Monitoring Console。因此，Monitoring Console 是要在此过程中安装的唯一组件。

此过程使用安装程序的图形界面。有关如何在其他模式下运行安装程序的信息，请参见《适用于 UNIX 的 Sun Java Enterprise System 5 安装指南》中的第 4 章 “使用基于文本的界面安装”以及《适用于 UNIX 的 Sun Java Enterprise System 5 安装指南》中的第 5 章 “以无提示模式安装”。

1. **在 Java ES 发行版中从与您的平台对应的目录启动 `installer` 应用程序。有关更多详细信息，请参见《 适用于 UNIX 的 Sun Java Enterprise System 5 安装指南 》中的“开始安装”。**
2. **在出现欢迎屏幕并接受许可证后，选择 “ 升级或安装 ” ，再选择 “ 安装新软件 ” ，然后单击 “ 下一步 ” 。**
3. **在组件选择屏幕上，仅选择要安装的 Sun Java System Monitoring Console 。单击 “ 下一步 ” 。**
4. **此时安装程序将检查共享组件需要的升级。完成后，单击 “ 下一步 ” 。**
5. **此时安装程序将检查系统要求。如果操作系统需要修补程序，请 “ 取消 ” 安装，添加系统必需的修补程序，然后重新启动此过程。否则单击 “ 下一步 ” 。**
6. **在配置类型选择屏幕上，选择 “ 立即配置 ” ，并在接下来的自定义配置屏幕上单击 “ 下一步 ” 。**
7. **此时安装程序已准备好安装 Monitoring Console ，单击 “ 下一步 ” 开始安装。在安装过程中，如果尚未对 Java ES 的部署进行注册，可以打开产品注册窗口。**
8. **安装完成后，可以查看安装摘要和日志，然后单击 “ 安装完成 ” 退出安装程序。**

接下来的操作 现在应继续阅读第 47 页中的 “配置 Monitoring Console” 。

▼ 在 Solaris zone 中安装 Monitoring Console

使用 Solaris zone，可以在同一台物理主机上将 Monitoring Console 作为 Java ES 的其他组件进行安装。这些组件将处于全局区域 (global zone) 中，您需要创建一个稀疏根本地区域作为 Monitoring Console 的逻辑主机。按以下顺序进行操作。

1. **在全局区域中安装并配置除 Monitoring Console 之外的所有 Java ES 组件。在全局区域中完成选定组件的所有安装后配置，以便所有服务器实例都在运行。**

2. 作为全局区域中的安装的一部分，Monitoring Framework 将作为共享组件安装在全局区域中。执行第 2 章中适用于所安装组件的所有过程。

3. 在同一台主机上，创建一个稀疏根本地区域作为 Monitoring Console 的逻辑主机。因为是稀疏根区域 (sparse root zone)，所以安装在 *mfwk-base* 中的 Monitoring Framework 应该是可见的（请参见第 9 页中的“默认路径和文件名”）。

4. 按照过程第 46 页中的“使用 Java ES 安装程序安装 Monitoring Console”在稀疏根本地区域中安装 Monitoring Console。

5. 使用以下命令在稀疏根区域中配置 Monitoring Framework：

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

   此命令将使用全局区域中的文件在本地区域中创建必要的 Monitoring Framework 配置文件。

接下来的操作 现在应继续阅读第 47 页中的“配置 Monitoring Console”。

▼ 配置 Monitoring Console

此过程介绍如何在独立的物理主机上配置 Monitoring Console。如果在由 Solaris zone 创建的逻辑主机上安装了 Monitoring Console，则配置命令与在稀疏根区域中进行配置的命令相同，但必须在该区域的文件系统中运行这些命令。

1. 使用 Monitoring Framework 通过以下命令初始化主代理：

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

2. 使用以下命令重新启动 Common Agent Container (cacao)：

   ```
   cacaoadm restart
   ```

▼ 取消配置 Monitoring Console

如果在希望安装其他组件的主机上安装和配置 Monitoring Console，将无法监视这些组件，因为 Monitoring Console 与 Monitoring Framework 存在冲突。要使用节点代理监视这些新组件，必须取消配置 Monitoring Console 的主代理。

- 以超级用户身份运行以下命令以取消配置 Monitoring Console：

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

安装的目录布局

有关操作系统上安装的软件包的名称，请参见《适用于 UNIX 的 Sun Java Enterprise System 5 安装参考》中的第 5 章“可安装软件包列表”。下表介绍了 Monitoring Console 软件包中的目录。默认安装目录 *MConsole-base* 具有以下含义，如第 9 页中的 “默认路径和文件名”中所述：

- Solaris 系统：`/opt/SUNWjesmc`
- Linux 系统：`/opt/sun/jesmc`

表 3–1 Monitoring Console 使用的目录和文件

路径	内容说明
MConsole-base`/WEB-INF/classes`	Web 应用程序 servlet 类
MConsole-base`/WEB-INF/lib`	Web 应用程序 JAR 相关项
MConsole-base`/WEB-INF/*.xml`	Web 应用程序描述符
MConsole-base`/css`	样式表文件
MConsole-base`/html`	HTML 文件
MConsole-base`/images`	用户界面中使用的 GIF 图像文件
MConsole-base`/js`	JavaSript™ 文件
MConsole-base`/*.jsp`	JavaServer Pages™ 文件
WebConsole-base`/prereg/jesmc/*.reg`	Monitoring Console 的 Web Console 文件

启动 Monitoring Console

Monitoring Console 是可以通过任何浏览器使用的 Web 应用程序，只要该浏览器能够连接到安装了该应用程序的主机。您将通过同一主机上自动安装的 Web Console 访问 Monitoring Console。以下过程说明了如何访问 Monitoring Console 以及如何查看受监视的组件。

▼ 启动 Monitoring Console

1. **必须首先重新启动 Web Console 的 Web 服务器。在安装了 Monitoring Console 的主机或区域中运行以下命令：**

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

2. **等待 Web Console 启动。使用以下命令查看 Web Console 是否已启动：**

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

可能需要运行此命令多次才能看到以下消息：

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

3 **使用以下 URL 通过任何可以连接到 Monitoring Console 主机的浏览器打开 Web Console。如果将 Monitoring Console 安装在 Solaris zone 中，则 *MC-host* 是指定给该区域的逻辑主机名：**

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

4 **您可能会看到有关不可信证书的消息，这取决于浏览器的配置。需要信任该证书才能访问 Web Console。**

5 **出现提示时，在 Monitoring Console 主机上使用超级用户密码以 `root` 身份登录 Web Console。**

登录后，Web Console 会列出其提供的所有服务。

6 **要打开 Monitoring Console 的主窗口，请单击标题“其他”下的 Sun Java System Monitoring Console，如以下屏幕捕获中所示。**

▼ 连接到节点代理

首次启动 Monitoring Console 时，必须指明托管受监视的组件的位置。在 Java ES 部署中指定每个节点代理的位置，控制台将自动显示每个节点代理中的所有组件。如果以后通过在新主机中安装 Java ES 组件来向部署中添加这些组件，还需要重复此过程。

添加节点代理后，每次访问控制台时，Monitoring Console 将重新连接到该节点代理，直到显式删除该节点代理。如果先前添加的节点代理不响应，请执行过程第 27 页中的“重新启动节点代理”。

1 **在安装 Monitoring Console 的逻辑主机与包含要监视的节点代理和 Java ES 组件的主机之间同步日期和时间。无论是自动同步还是手动同步，每台主机的时间之间的差值必须不能超过大约 10 分钟。**

2 **如有必要，单击 Monitoring Console 左侧的分层结构根目录上的“部署”链接，导航到“部署”级别屏幕。现在选择右侧窗格中的“主机”选项卡，然后单击“添加”。**

3 **在显示的“添加主机”对话框中，按以下屏幕捕获中所示输入必要的信息：**

- 主机名：输入配置了受监视组件的节点代理的全限定主机名。
- 端口：11164，除非在节点代理所驻留的主机上另外配置了 Monitoring Framework。
- 远程主机超级用户密码：输入节点代理所驻留的系统的超级用户密码。

4. **单击“ 测试连接性 ”。**

   如果连接信息正确，且主机代理已进行了配置并正在运行，对话框将指明已连接到主机代理。

5. **单击“ 确定 ”完成“ 添加主机 ”对话框，新名称将显示在“ 主机 ”列表中。该主机的节点代理中的所有受监视组件现在也显示在左列中。**

6. **对 Java ES 部署中安装了受监视组件的每台主机重复此过程。**

接下来的操作 现在可以浏览左列中列出的组件，以查看其操作状态、这些组件公开的受监视属性以及它们已触发的所有报警。

使用 Monitoring Console

本节中的过程介绍如何与 Monitoring Console 交互。

▼ 有选择地禁用和重新启用监视

Java ES 监视机制设计为轻量机制，以便不影响产品系统的性能。但是在某些情况下，可能需要停止收集监视值，以便配置仅对性能造成极小的影响。Monitoring Console 提供了分别在每台主机上执行此操作的方法，如以下过程中所述。

1. **如有必要，单击 Monitoring Console 左侧的分层结构根目录上的“ 部署 ”链接，导航到“ 部署 ”级别屏幕。然后单击右侧窗格中的“ 主机 ”选项卡。**

“主机”选项卡上的表列出了所有包含由 Monitoring Console 监视的 Java ES 组件的主机。

2 **使用表的左列中的复选框选择所有要停止监视的主机。单击“ 主机 ”表顶部的“ 禁用 ”。**

更多信息

结果

禁用对主机的监视时，该主机下的分层结构中的所有监视对象都将被禁用。在禁用状态下，受监视的对象不再被更新，但它们可能仍然包含最新的值。依赖于禁用对象的监视规则被暂停。要启用已禁用的主机，请使用“主机”表顶部的“启用”按钮执行此过程。

▼ 创建新的监视规则

监视规则也称为监视作业，是一组受监视值的条件，这些条件由用户定义以触发报警。Monitoring Console 中的监视规则向导将帮助您定义要监视的条件。

1 **如有必要，单击 Monitoring Console 左侧的分层结构根目录上的“ 部署 ”链接，导航到“ 部署 ”级别屏幕。现在选择以下屏幕捕获中所示的右侧窗格中的“ 监视规则 ”选项卡，然后单击监视规则表中的“ 新建 ”：**

2 为新的监视规则指定名称并选择要监视的服务器的类型。

3 选择要监视的组件产品的实例。如果在独立主机上安装的同一产品存在两个实例，则某些实例可能在此表中具有相同的名称。在此情况下，这些实例的顺序可能与显示在左侧窗格的分层结构中的顺序相同，但不能确定是否是这样。您可能需要对这两个实例创建相同的监视规则，以确保定义了规则。

4 选择包含要监视的属性的对象：

5 现在，可以最后指定要监视的属性以及将生成报警的值。

6 输入规则的开始日期和结束日期。与用于确定规则活动或不活动的调度不同，开始日期和结束日期用来确定什么时候规则是被定义为存在的。如果开始时间为过去的时间（默认值经常出现这种情况），则与此规则关联的监视将立即开始。

7 可选择使用控件来创建一个或多个时间范围，在该时间范围内规则将进行监视，也可以选择一周中的某些天来创建每周调度。

8 在规则向导的此最后步骤中，查看您的输入，然后单击“ 完成 ”以创建新规则。

完成规则向导后，应再次查看“监视规则”选项卡，此时新规则已在规则表中。

Monitoring Console 故障排除

另请参见《适用于 UNIX 的 Sun Java Enterprise System 5 发行说明》中列出的已知问题。

如果主代理与节点代理冲突，请检查以下情况：

- 如果使用 Solaris zone，请确保 Monitoring Console 安装在稀疏根本地区域中。
- 确保主机或区域中不存在先前安装的受监视组件。
- 在任何一种情况下，都需要卸载 Monitoring Console 和所有组件，更正问题，然后重新安装 Monitoring Console。

如果卸载 Monitoring Console 并在同一台主机上重新安装它，Monitoring Console 将无法初始化并且不会显示在 Web Console 中。在此情况下，可在 Monitoring Console 主机上运行 `masetup -i` 命令来初始化主代理。然后执行过程第 48 页中的 “启动 Monitoring Console”。

监视规则有一个限制，即只有当规则处于活动状态时才允许被禁用。如果要禁用的规则的调度使该规则在当前时间处于非活动状态，则必须暂时更改该规则的调度使该规则处于活动状态，或者彻底删除该规则。

由于 Windows 平台的限制，主机统计信息中的 `handleCount` 和 `threadCount` 值始终为 0（零）。

◆ ◆ ◆ 附 录 A

CMM 对象参考

通用监视模型 (Common Monitoring Model, CMM) 是使用 Java 编程语言实现的通用信息模型 (Common Information Model, CIM) 的扩展。CIM 体现在 com.sun.cmm.cim.* 软件包的 Java 接口中。CMM 体现在扩展 CIM 接口的 `com.sun.cmm.*` 软件包的接口中。受监视的对象在节点代理中由实现 CMM 接口的类表示。下表显示了每类对象的可以监视的属性。

CMM 对象概述

CMM 基于一组有限的核心接口，这些接口定义该类型的受监视对象可以公开的属性。下表显示了表示大量类型的由 CMM 定义的受监视对象的类，并给出了这些类的一些关键属性：

`CMM_InstalledProduct`	视为一个整体的 Java ES 组件产品。例如，Java ES Directory Server。
`CMM_ApplicationSystem`	已安装和配置的 Java ES 组件产品的实例。此实例可能在运行，也可能未运行。此对象的典型属性是管理员的联系信息、系统的操作状态以及应用程序的启动或停止时间。
`CMM_Service`	组件产品的特定功能，例如 Java ES Directory Server 验证服务。典型属性是服务的操作状态。
`CMM_SoftwareResource`	在高速缓存、线程池等环境中软件实体的表示。典型属性是高速缓存大小。
`CMM_LogicalComponent`	一个实体，由服务处理并对最终用户可见，但不表示实际物理资源或软件功能。例如，软件实例的一组配置参数，而不是实例本身。
`CMM_ServiceAccessURI`	可以使用服务的位置。典型属性是端口号或统一资源标识符 (Uniform Resource Identifier, URI)。

CMM_RemoteServiceAccessPoint	远程连接的访问和寻址信息。典型属性是 URI 或连接的操作状态（打开或关闭）。
CMM_Process	运行程序的一个实例。典型属性是内存或 CPU 的使用情况。
CMM_UnitaryComputerSystem	Java ES 部署使用的单个主机，例如桌面计算机或服务器。典型属性是可用处理器的数量或物理内存的大小。
CMM_OperatingSystem	使主机的硬件可用的软件或固件。典型属性是系统中可用虚拟内存的大小。
CMM_JVM	Java ES 服务器使用的 Java 虚拟机。示例属性是 Java 虚拟机的版本号。
CMM_DatabaseService	代表数据库执行的任务，例如提供用户访问。典型属性是与数据库的连接的最大允许数量。
CMM_CommonDatabase	给定类型数据库中的通用属性。典型属性是最近备份的日期。

◆ ◆ ◆ 附 录 B

每个组件公开的受监视对象

本附录中的各节列出了支持监视的各个产品组件中已配置的 CMM 对象。仅配置了对象属性的子集时，也将列出这些属性。

配置 Common Agent Container

尚未记录。

配置 Access Manager

尚未记录。

配置 Application Server

尚未记录。

配置 Calendar Server

尚未记录。

配置 Directory Server

尚未记录。

配置 Instant Messaging

尚未记录。

配置 Messaging Server

尚未记录。

配置 Portal Server

尚未记录。

配置 Web Server

尚未记录。

索引