用于 Sun Fire™ B1600 的 Sun™ SNMP Management Agent 指南

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054 U.S.A.
650-960-1300

部件号：817-2503-10
2003 年 4 月，修订版 01

请将有关本文档的意见或建议发送至：http://www.sun.com/hwdocs/feedback

请回收

Adobe PostScript

目录

图形列表

表格列表

代码示例

前言

本指南描述了用于 Sun Fire B1600 平台的 Sun SNMP Management Agent，它支持使用简单网络管理协议 (SNMP) 进行平台硬件管理。

Management Agent 可*监视*清单、配置、环境和故障报告。它还可以*控制*和*监视*服务指示器，以及处理器 blade 的电源、待机或复位状态。

本指南主要针对经验丰富的企业管理员和专业开发人员。

本指南分为以下两部分：

- 第一部分 （第一章到第八章）介绍 SNMP Management Agent 及其功能。
- 第二部分 （第九章到第十三章）介绍如何安装和配置本软件。

本书结构

本指南包括以下几章：

第一部分

第一章介绍 Sun SNMP Management Agent 软件的组件。

第二章简要介绍简单网络管理协议 (SNMP) 的主要特性。

第三章介绍 SNMPv3 Master Agent 的功能和特性。

第四章概述 SNMP 如何建立 Sun Fire B1600 的模型。

第五章介绍在 SNMP 界面中如何表示 Sun Fire B1600 受控对象及其关系。

第六章介绍 sunPlat 物理类的分层结构以及 SUN-PLATFORM-MIB 如何表示在 sunPlat 模型中定义的受控物理对象类。

第七章介绍 sunPlat 逻辑类的分层结构以及 SUN-PLATFORM-MIB 如何表示在 sunPlat 模型中定义的受控对象类。

第八章介绍 SUN-PLATFORM-MIB 中定义的 SunPlat 通知类及属性。

第二部分

第九章介绍组成 Sun Fire B1600 的管理软件的组件，以及安装 SNMP 软件前应执行的系统检查。

第十章介绍如何在 Sun Fire B1600 上安装管理软件。

第十一章 提供了有关用户配置文件的信息。

第十二章介绍安装后的缺省配置，并说明如何更改这些配置文件。

第十三章说明如何卸载本软件。

第十四章提供软件错误诊断的帮助。

附录 A 介绍如何安装 J2RE 使其与 J2SE 共存，以及如何修改启动脚本来定位安装。

排印约定

字体或符号	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。 `% You have mail.`
`AaBbCc123`	输入的内容，与计算机屏幕输出相区别	`%` **`su`** `Password:`
AaBbCc123	书名、新词或术语以及要强调的词。请用实际名称或值来替代命令行变量。	请阅读 *《用户指南》* 的第六章。这些称为*类*选项。 要删除文件，键入 **`rm`** *文件名*。

Shell 提示符

Shell	提示符
C shell	*机器名* %
C shell 超级用户	*机器名* %
Bourne shell 和 Korn shell	$
Bourne shell 和 Korn shell 超级用户	#

相关文档资料

应用	书名	部件号
SunMC	*《用于 Sun Fire B1600 的 SunMC 3.0 补充资料》*	817-2540
发行说明	*《SNMP Release Notes for the Sun Fire B1600》*	817-1006
Sun Fire B1600 平台	*《Sun Fire B1600 Blade System Chassis Hardware Installation Guide》*	816-7614
	《Sun Fire B1600 Blade System Chassis Software Setup Guide》	816-3361
	《Sun Fire B1600 Blade System Chassis Administration Guide》	816-4765
	《Sun Fire B1600 Blade System Chassis Switch Administration Guide》	816-3365

访问 Sun 文档资料

您可以查阅、打印或购买包括本地化版本在内的内容全面的精选 Sun 文档资料，网址为：

`http://www.sun.com/documentation`

联系 Sun 技术支持

如果有关于本产品的技术问题在此文档中未能找到答案，请访问以下网址：

`http://www.sun.com/service/contacting`

Sun 欢迎您提出宝贵意见

Sun 致力于提高文档资料质量，并十分乐意收到您的意见和建议。请访问以下网址提交您的意见或建议：

`http://www.sun.com/hwdocs/feedback`

请在反馈信息中包含本文档的书名和部件号，即：

《用于 Sun Fire B1600 的 Sun SNMP Management Agent 指南》，部件号 817-2503-10

第一部分　技术说明和功能

第一章

Sun SNMP Management Agent 补充资料

这一版本的 Sun™ SNMP Management Agent 可以对 Sun Fire™ B1600 搁板和 Sun Fire B100s blade 实行监视和控制。

您可以根据不同的平台类型安装不同的代理：

- 运行在 Sun Fire B100s blade 上的域代理程序 （域硬件监视）

  在要监视的服务器本地安装该软件，并且只监视该服务器。对于 Sun Fire B1600，软件将分别监视每台 blade 服务器。

  Sun Fire B100 blade 的域硬件监视范围仅限于该 blade 服务器的硬件。它不监视其它搁板组件，例如服务指示器、 PSU、 SSC 和搁板本身的等同设备。

- 通过系统控制器代理的平台代理程序 （平台硬件监视）

  软件安装在一台远程 （平台代理程序）服务器上，该服务器通过系统控制器访问平台装置。这样您就可以监视所有由系统控制器管理的硬件。

  Sun Fire B1600 监视的平台硬件包括搁板、其等同设备、服务指示器及搁板所有的现场可更换部件 (FRU)。另外，它还会提供一些域硬件监视不能提供的 Sun Fire B100s blade 硬件信息 （尤其是电压监视）。

  图 1-1 显示两种硬件监视类型的示例。Sun Fire B1600 搁板 A 和 B 通过平台代理服务器连接到网络管理站 （平台硬件监视）。而 Sun Fire B1600 搁板 C 的情况是 Sun Fire B100s blade 直接连接到网络管理站 （域硬件监视）。

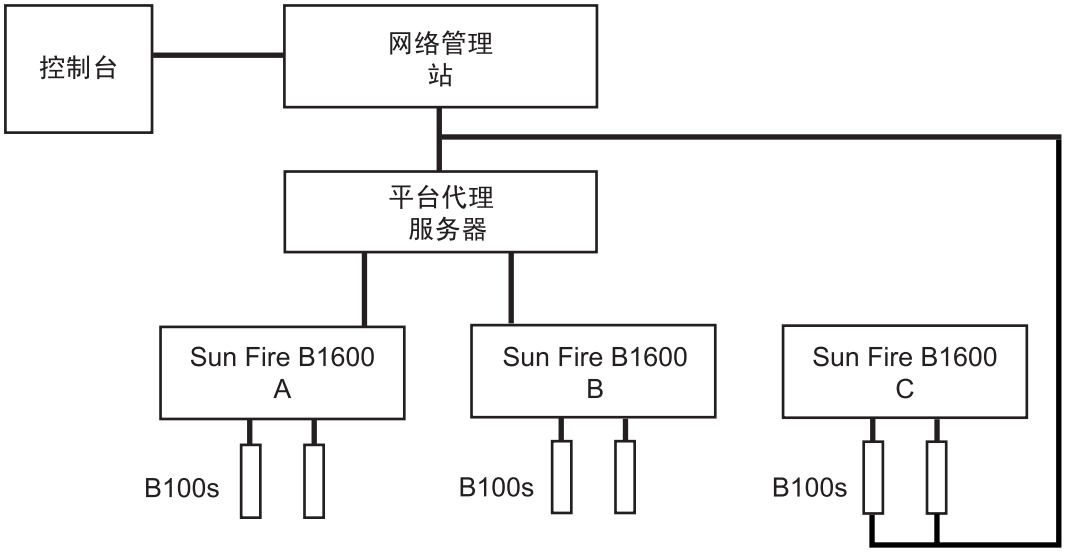

图 1-1 域和平台硬件监视示例

本软件包含许多软件包，可提供以下功能：

- SNMP 子代理程序

  缺省情况下， SNMP 子代理程序注册为 Solaris 主代理程序 snmpdx 的子代理程序。子代理程序亦称 *SNMP 中介器*。

- SNMPv3 主代理程序

  SNMPv3 主代理程序提供了一个单一安全的驻留点，可通过该点访问位于平台上的 SNMP 中介器。主代理程序充当代理服务器，将请求转发到 `snmpdx`。

- Sun Fire B1600 和 Sun Fire B100s 装置。

  这些软件包根据需要安装，取决于使用的是基于域的还是基于平台的硬件监视。

第二章

SNMP 简介

本章简要介绍了简单网络管理协议 (SNMP) 的重要特性。对于与 Sun Fire™ B1600 系统相关的特殊问题，本章并没有给出详尽的阐述。

本章包括以下各节：

SNMP 版本

SNMP 是一个管理联网设备 （系统）的开放式 Internet 标准。与其它 Internet 标准一样，它是通过 Internet 工程任务组 (IETF) 所发布的大量注解请求 (RFC) 来定义的。

有三种 SNMP 版本可定义认可的标准：

- SNMPv1
- SNMPv2 （亦称 SNMPv2c，在本文档中即是如此）
- SNMPv3

1988 年首先定义了 SNMPv1。 1993 年引入了 SNMPv2，它试图通过增加更多的协议操作和数据类型以及提供安全性来解决 SNMPv1 的一些缺点。安全模型的限制产生了如今的 SNMPv2c 标准，它采用了基于安全性的新特性。同时还出现了实验版的 SNMPv2usec 和 SNMPv2*，但这些版本并未获得广泛采用，仍处于实验阶段。

1999 年引入的 SNMPv3 定义了 SNMP 管理框架，它支持可插拔组件，包括安全性。

有关这些标准的进一步信息，请参考 IETF 站点 的以下 RFC 信息 (`http://www.ietf.org/rfc.html`)：

- SNMPv1: RFC1155, RFC1157, RFC1212, RFC1215
- SNMPv2: RFC2578, RFC2579, RFC2580, RFC3416
- SNMPv3: RFC3410, RFC3411, RFC3412, RFC3413, RFC3414, RFC3415
- 标准之间的共存：RFC2576

SNMP 管理器和代理程序

SNMP 是一种网络协议，它允许通过网络管理站 (NMS) （通常亦称 *管理器*）远程管理设备。

设备必须具有一个与其相关联的 SNMP *代理程序*（SNMP 中介器）方可被管理。该中介器的用途是：

- 从管理器接收表示设备状态的数据请求，给予适当的响应
- 接受来自管理器的数据，启用对设备状态的控制
- 生成 SNMP *陷阱*，即主动发送到一个或多个选定管理器的消息，以通知有关该设备的重大事件

SNMP 管理信息库

要管理和监视设备，则必须使用一种代理程序和管理器均可识别的格式表示其特性。这些特性可表示设备的物理特性 （例如风扇速度）或服务 （例如路由表）。定义这些特性的数据结构称作 *管理信息库* (MIB)。此数据模型通常是表格的形式，但也可包含简单的值。路由表即为前者的示例，而后者的示例可为一个表明代理程序启动时间的时间戳。

MIB 是可通过 SNMP 访问的虚拟数据存储的定义。可使用如下的 `get` 和 `set` 操作从管理器访问其内容：

- 响应 `get` 操作，中介器会提供数据，这些数据可以是在本地维护的，或直接来自所管理的设备。
- 响应 `set` 操作，代理程序通常会执行一些操作，影响其自身或其管理的设备的状态。

为使 NMS 通过其代理程序管理设备，对应于由代理程序提交的数据的 MIB 必须加载到管理器。执行此操作的机制因采用的网络管理软件不同而异。此操作向管理器提供信息，以便正确寻址和解释代理程序提交的数据模型。

注意 – MIB 可能会引用其它 MIB 中的定义，因此使用某一指定 MIB 时，可能需要加载其它 MIB。

为寻址此虚拟数据存储中的内容，MIB 以*对象标识符* (OID) 定义。OID 由一组按分层结构排列的整数序列构成，它定义一个唯一的名称空间。每个指定的整数均有一个关联的文本名称。例如， OID `1.3.6.1` 对应于 OID 名称 `iso.org.dod.internet`， `1.3.6.1.4` 对应于 OID 名称 `iso.org.dod.internet.private`。

数字形式用于 SNMP 内部协议事务，而文本形式则用于用户界面中以便于阅读。引用这种 OID 表示的对象时，通常使用其名称的最后一部分作为简写形式。为避免这种约定引起的混淆，通常为所有由此定义的对象名称使用特定的 MIB 前缀，例如 *sunPlat*，从而使这种标识符在全局范围内唯一。

注意 – MIB 使用称为 ASN.1 的语言进行定义，其讨论超出本文档范围。作为参考，SNMPv2c 的 MIB 结构由其管理信息结构 (SMI) 定义 （在 RFC2578 中定义）。它定义 MIB 中可用的语法和基本数据类型。 RFC2579 中定义的文本约定 （类型定义）定义了附加的数据类型和枚举。

MIB 表

MIB 定义的大多数数据内容均为表格形式，并组织为由对象序列 （每个对象均有其自身的 OID）组成的数据项。例如，一个风扇特征表可包含多行，每个风扇一行，其中每列对应风扇的当前速度、期望速度及可接受的最低速度。

表内各行的编址可为：

- 简单的一维索引 （表内的行号，例如 “6”）
- 较复杂的多维实例说明符，例如 IP 地址和端口号 （如 `127.0.0.1, 1234`）

MIB 中的每个表定义均有一个 INDEX 子句，它定义用哪个实例说明符来选择指定数据项。上述两种情况下，用于定义所需行的索引的对象均必须本身已在 MIB 中定义。因此，简单的一维索引表通常包含一个供该表的 INDEX 子句引用的索引列。这样即可通过指定给出列前缀的 OID 来寻址表内指定的数据项。

例如，带有实例说明符后缀 （如前面示例中的 `127.0.0.1.1234`）的 `myFanTable.myFanEntry.myCurrentFanSpeed` 指定 `myFanTable.myFanEntry.myCurrentFanSpeed.127.0.0.1.1234`。

定义 MIB 语法的 SMI 提供了扩展表格的重要功能，通过向表中添加额外的列来有效地添加额外的数据项。此功能通过定义一个带有 INDEX 子句 （该子句为要扩展的表 INDEX 子句的副本）的表来实现。

还可定义另一种 MIB 表，不通过其中的对象来索引，而是通过从其它表格 *导入* 的对象 （可能在其它 MIB 中已定义）来索引。这种结构可以有效地为现有表格添加列。

注意 – SUN-PLATFORM-MIB 广泛采用这种机制扩展 ENTITY-MIB 中定义的表（请参见第五章）。

访问控制

MIB 中定义的所有可寻址对象均有相关联的最大访问权限，例如 *只读* 或 *读写*。这些定义决定了代理程序支持的最大访问权限，管理器可用其限制操作员所进行的操作。代理程序可根据需要应用较低的访问权限，也就是说，它可以拒绝对读写对象的写访问。这种拒绝基于以下原因：

- 对寻址对象的该操作的可行性 （例如，如果 MIB 定义的对象表示只能执行特定任务处理的计算机的状态）
- 安全性限制，限制一组受限管理器的特定操作

SMMPv1 中用于安全性访问权限通信的机制为 *团体字符串*。这些字符串是随 SNMP 数据请求传递的简单文本字符串，例如 *private* 和 *public*。由于 SNMPv1 和 SNMPv2 请求不加密，因此这种请求不安全。该机制用于定义代理程序应响应的团体字符串，及其来自哪个管理器。它取决于采用的代理程序，但通常基于访问控制表 (ACL)，即说明可用访问权限的文件。

关于如何配置 ACL 的说明，请参见第十一章。

SNMP 主代理程序

管理器通过发送 (UDP) 数据包到运行代理程序的系统的公认端口 161 与代理程序进行通信。如果某个系统中同时运行多个代理程序，且它们管理不同设备，则使用端口资源可能会发生冲突。解决方案之一是为每个代理程序指定不同的、非标准的端口编号。另一种方案是引入 *主代理程序* 的概念，即代表所有运行在指定系统上的代理程序接受 SNMP 请求，并相应地转发这些请求。这种方案的优点是允许管理器以一致的方式访问所有 SNMP 代理程序。 Sun Fire B1600 对以上两种方法均支持。

有关主代理程序的进一步信息，请参考第三章。

SNMP 中介器和 snmpdx

`snmpdx` 是标准的 Solaris™ SNMP 代理程序，并且是 SNMPv1 主代理程序。

缺省情况下， SNMP 中介器注册为 `snmpdx` 的子代理程序。这种配置中，尽管可发出 SNMPv2c 通知，但只支持 SNMPv1 的 `get` 和 `set` 请求。

`snmpdx` 和 SNMP 中介器之间的关系将在第十章和第十一章中进一步阐述。

另见手册页 `snmpdx`(1M)。

第三章

主代理程序

本章介绍 SNMPv3 主代理程序的功能和特性。

本章包括以下各节：

功能

SNMPv3 主代理程序提供唯一安全的驻留点，可通过该点访问 SNMP 管理信息。

SNMPv3 主代理程序绑定到 SNMP 服务端口 （缺省值为 161），将所有请求转发到 `snmpdx` （标准的 Solaris 主代理程序）。 snmpdx 接下来将这些请求转发到相应的注册子代理程序。 `snmpdx` 包含在标准的 Solaris 发行版中提供，但它只支持 SNMPv1，因此不能直接提供 SNMPv3 所提供的安全性。主代理程序转发所有请求到 SNMPv （无论其为 SNMPv1、 v2c 或 v3），以便 `snmpdx` 进行处理。

实际上，主代理程序充当 SNMP 的防火墙，提供对所有现有子代理程序的安全访问。

配置概述

本节介绍了配置 SNMP 使其包含主代理程序功能的方法。这一主题将在第十一章中进一步阐述，其中包括本节其他部分提到的配置文件的完整说明和示例。

SNMP 中介器使用自动分配的端口号，注册为 snmpdx 的子代理程序。

注意 – SNMP 代理程序在本指南中称作 SNMP 中介器。

主代理程序启用后， `snmpdx` 自动启动变为 “禁用”状态，并且主代理程序注册于端口 161。 `snmpdx` 分配到一个新的端口号，它在主代理程序的启动文件控制下启动。

可通过以下方式进行配置：

- SNMPv3 安全性文件 `spama.uacl` 和 `spama.security`
- SNMPv1/v2 访问控制表 (ACL) 文件 `spama.acl`
- 配置文件 `spama.conf`
- 启动脚本，即 `spama`

不能在主代理程序正在运行时进行动态配置。

SNMP 中介器也需要进行配置，这部分内容将在第十一章中进行说明。

第四章

平台管理模型

本章概述了 SNMP 如何使用 Sun 平台 SNMP 模型 (sunPlat) 建立 Sun Fire B1600 系统模型。

本章包括以下各节：

为 Sun Fire B1600 平台 建立模型

Sun Fire B1600 表示一组嵌套在机箱内的*硬件资源*。一些资源 （例如主板）可以直接嵌套在机箱内。另一些资源则嵌套在其它资源内。例如，主板可包含处理器。这种从机箱内扩展的关系构成了硬件资源的*分层结构*，每一层均包含在其物理*父层*内。我们用表示不同硬件资源的*受控对象* 之间的*关系* 来模拟此分层结构。

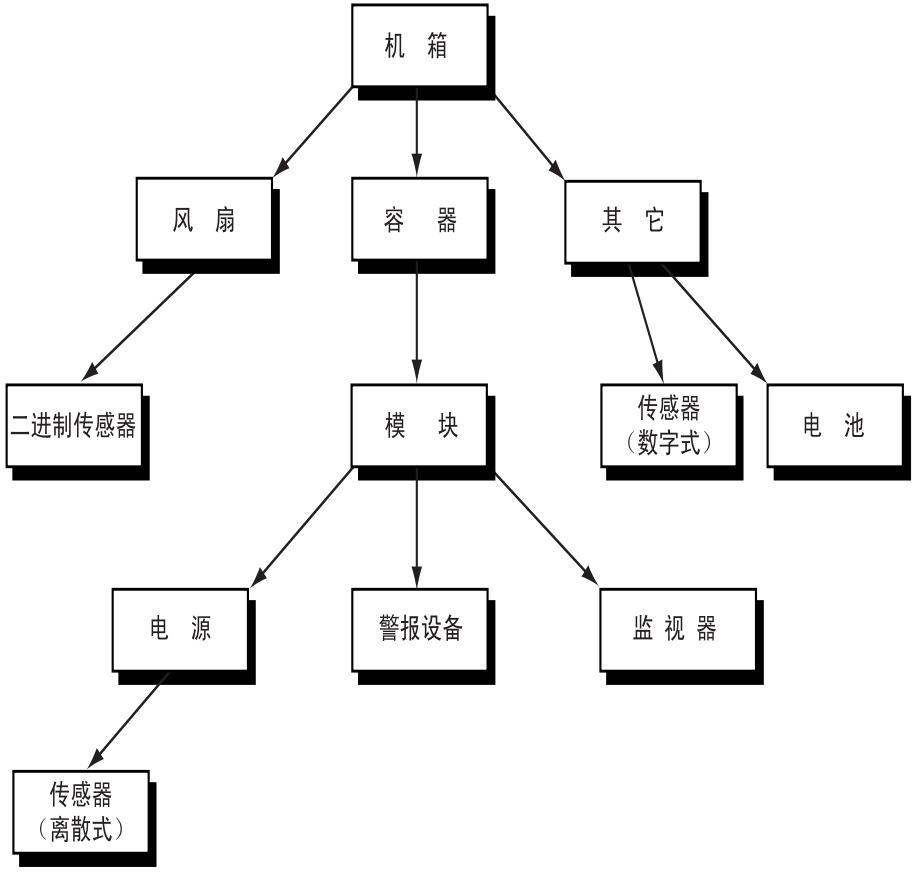

图 4-1 硬件资源分层结构示例

受控对象

sunPlat 模型提供了一组表示基本硬件资源的常用平台构造块。这些平台构造块的实例称作*受控对象*。若一个硬件资源可被监视，或可提供有用的配置信息，则表示为受控对象。

附加受控对象用于表示管理界面的其它特性。例如，硬件资源可发送同步状态报告（*通知*），以响应问题（*警报*）或配置中的更改（*事件*）。

受控对象根据受控对象*类*进行定义。硬件资源的特征通过受控对象的*特性*体现。称为*子类*的新类根据现有类定义。子类*继承*其*父类*的所有特征，但可以通过添加新特性来表现其自身特征。

图 4-2 所示为 sunPlat 模型定义的硬件构造块的类继承性分层结构。

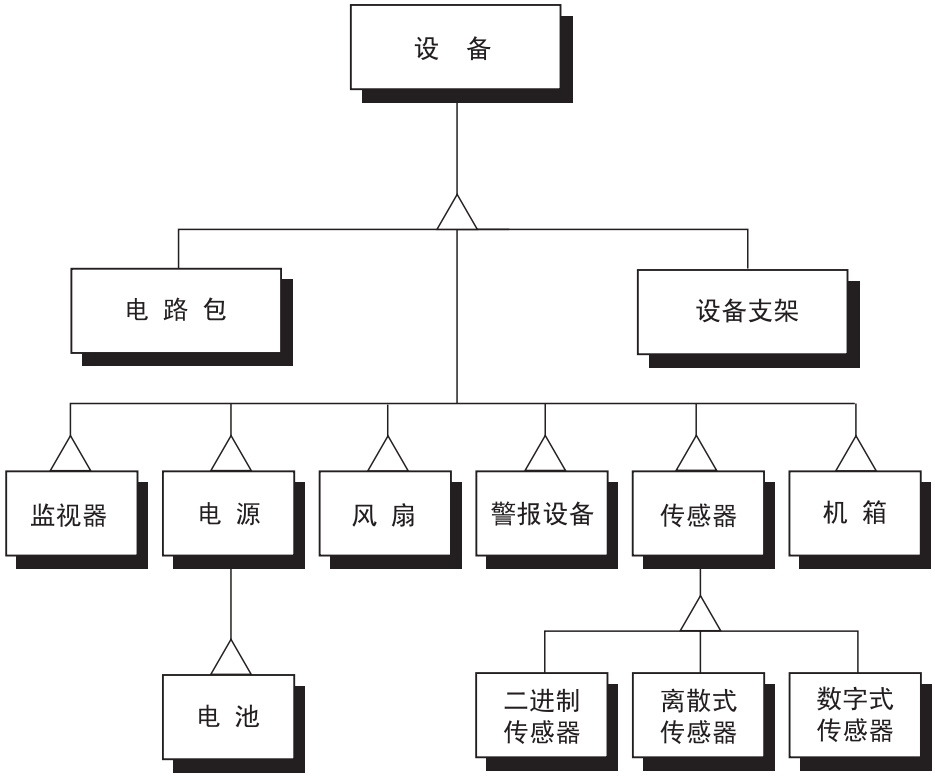

图 4-2 sunPlat 受控对象继承性图表

sunPlat 类的衍生

sunPlat 类基于工业标准管理概念。 Sun Fire B1600 系统使用选自其所代表的硬件框架结构的 ITU-T 通用网络信息模型的子集。这提供了一个强大的可扩展框架，在电信管理网 (TMN) 中支持统一的故障和配置管理。

分布式管理任务组 (DMTF) 通用信息模型 (CIM) 图解模拟了物理环境以及事件定义和处理，并为通用模型提供了系统专用的扩展。

第五章

Sun Fire B1600 MIB

本章说明 SNMP 界面如何表示 Sun Fire B1600 受控对象及其关系。

本章包括以下各节：

模型的 SNMP 表示法

SNMP 中介器支持巡回检测管理和基于事件的管理两种方式。 Sun Fire B1600 系统的物理组件同时也是管理域内的逻辑表示。它们由 ENTITY-MIB 提供，通过 RFC 2737 定义，并由 SUN-PLATFORM-MIB 扩展。

注意 – 虽然 MIB 中定义的许多对象都有 `read-write` 的 `MAX-ACCESS` 权限，但只有操作适合于建立模型的组件时，这些对象才是可写的。

ENTITY-MIB 包含以下各组，它们描述受控系统的物理和逻辑元素：

entityPhysical 组

`entityPhysical` 组描述物理实体——代理程序管理的可标识物理资源（例如机箱、电源、传感器等）。这些实体通过 *entPhysicalTable* 中的行表示。

entityLogical 组

`entityLogical` 组描述代理程序管理的逻辑实体。它们表示提供提取服务的*高值*逻辑实体，这些实体需要较高级别的管理。这些主要与平台硬件管理有关，并包含诸如重新引导 OS、硬件复位和电源控制之类的功能。它们通常对应管理域，例如 Solaris 域或服务控制器。

entityMapping 组

`entityMapping` 组标识 `entityPhysical` 组与 `entityLogical` 组之间的关系。此功能通过 SNMP 中介器在内部处理。

entityGeneral 组

`entityGeneral` 组提供上次物理实体表或物理映射表中的实体更改时间的时间戳。

entityMIBTraps 组

`entityMIBTraps` 组定义了 *entPhysicalChange* 通知，它发信号通知 ENTITY-MIB 中任何对象的更改。

第二章概述了 SNMP 中的通用元素如何表示 Sun 平台 SNMP 模型。

物理模型

sunPlat 物理模型使用 ENTITY-MIB 提供硬件实体的树形分层结构。每一实体的模型均为 ENTITY-MIB 的 entPhysicalTable 中单独一行。

图 5-1 是物理树形分层结构的一个示例。右下角的数字即 *entPhysicalTable* 中相应行的索引 （请参见表 5-1）。

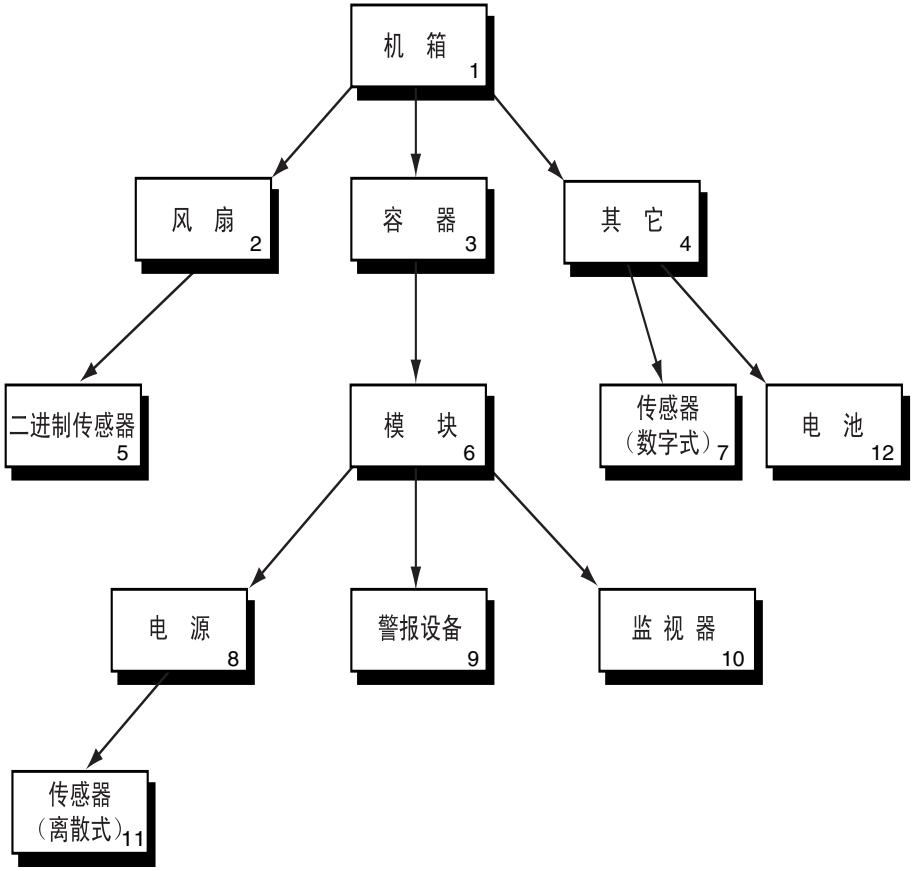

图 5-1　　硬件资源分层结构示例

此信息通过 SNMP 表来表示：

- 物理实体表 (*entPhysicalTable*)

  该表中每个硬件实体占一行。这些行称为*数据项*，并且特定一行称为*实例*。每个数据项包含以下内容：

  - 物理类 (*entPhysicalClass*)
  - 硬件实体的一般特征
  - 一个唯一的索引 (*entPhysicalIndex*)
  - 指向硬件实体所在行的引用 (*entPhysicalContainedIn*)，充当此资源*容器*。对于物理上不包含于其它容器内的组件 （例如机箱），该值为零。

- 物理映射表 (*entPhysicalContainsTable*)

  此表为物理实体表中所表示的硬件资源分层结构提供了一个虚拟的副本。该表是二维的，首先按包含数据项的 *entPhysicalIndex* 进行索引，然后再按每个包含的数据项的 *entPhysicalChildIndex* 进行索引。

表 5-1 所示为上图所基于的 *entPhysicalTable*，而表 5-2 所示为物理映射。

表 5-1 物理实体表

entPhysicalIndex	entPhysicalClass	entPhysicalContainedIn
1	机箱	0
2	风扇	1
3	容器 （例如，包含 FRU 的插槽）	1
4	其它	1
5	传感器 （二进制）	2
6	模块 （例如，可插拔的 FRU）	3
7	传感器 （数字式）	4
8	电源	6
9	警报设备	6
10	监视器	6
11	传感器 （离散式）	8
12	电源 （电池）	4

表 5-2　物理映射表

entPhysicalIndex	entPhysicalChildIndex
1	2
1	3
1	4
2	5
3	6
4	7
4	12
6	8
6	9
6	10
8	11

类

`entPhysicalClass` 是枚举值，表示特定物理实体的常规硬件类型，每种类型以 *entPhysicalTable* 中的一行表示。

以下枚举适用于 Sun Fire B1600 平台 （另见图 5-1 和表 5-1）：

- `other(1)`

  若该枚举无法归类为以下类别之一，则使用枚举 `other`。

- `chassis(3)`

  `chassis` 类表示设备的总容器。任何物理实体类均可包含于机箱内。

- `container(5)`

  `container` 类适用于可包含一个或多个可拆卸物理实体 （同一类型或不同类型）的物理实体。例如，机箱内每一空的或满的插槽均可模拟为容器。而现场可更换部件 (FRU) （例如电源或风扇）则可模拟为容器实体内的模块。

- `powerSupply(6)`

  `power supply` 类适用于可供电的组件。

- `fan(7)`

  `fan` 类适用于本身为风扇或其它冷却设备的物理实体。

- `sensor(8)`

  `sensor` 类适用于可测量一些物理属性的物理实体。

- `module(9)`

  `module` 类适用于自包含的子系统。该子系统同时包含于另一物理实体 （例如 `chassis`）或另一 `module` 内。该实体始终模拟于 `container` 内。

逻辑模型

sunPlat 逻辑模型使用 ENTITY-MIB 提供高值逻辑实体的列表。 ENTITY-MIB 的 *entLogicalTable* 中，每一单独的行表示一个实体。请注意，与物理模型不同的是，逻辑模型是平行结构而不是分层结构。

ENTITY-MIB 不区分逻辑对象的不同类别，这一点与物理对象不同。 SUN-PLATFORM-MIB 为逻辑对象提供了类分层结构，这部分内容将在第七章说明。

entLogicalTable 中的信息可使用不同的命名环境支持多重范围。不过，此功能在本产品中尚未使用。特定值的信息是 *entLogicalDescription* 和 *entLogicalTAddress*，后者给出访问逻辑实体的 IP 地址。

逻辑和物理分层结构映射

ENTITY-MIB 提供了逻辑对象与其所组成物理对象之间的映射。这可通过 *entLPMappingTable* 实现。它是二维表 （类似 *entPhysicalContainsTable*），可以标识指定逻辑实体的物理实体。这些物理实体通过其 *entLPPhysicalIndex*（相当于 *entPhysicalIndex*）标识。

虽然此表格可表示与指定的逻辑实体相关联的所有物理实体，但根据惯例，只引用封闭的物理实体。例如 , 对于物理模块实现的逻辑实体，该映射只引用模块，而并非引用包含在模块内的所有物理实体。

事件和警报模型

ENTITY-MIB 只提供一个 SNMP 通知 (*entConfigChange*)，它发送信号以通知 MIB 中任何表的更改。其最大值设置为每 5 秒钟发送一个陷阱。

SUN-PLATFORM-MIB 定义了更多特殊通知，这些将在第八章中进行说明。

SUN-PLATFORM-MIB

SUN-PLATFORM-MIB：

- 扩展物理实体表以表示组件的新类
- 扩展逻辑实体表以表示高值平台和服务器对象

注意 – SUN-PLATFORM-MIB 中所有对象均带有前缀 *sunPlat*，以使其全局唯一。

物理模型扩展表

SUN-PLATFORM-MIB 为物理实体表中未表示的类提供附加属性。它通过添加以下很少使用的扩展表来扩展物理实体表。

- 设备表扩展

  此表扩充物理实体表，提供设备类受控对象的进一步信息。此类适用于所有 Sun Fire B1600 硬件资源。设备类的子类通过下一级表扩展表示。

  - 设备支架表扩展

    它扩展设备表扩展。它提供与 `container(5)` entPhysicalClass 的受控对象有关的附加信息。

  - 电路包表扩展

    它扩展设备表扩展。它提供与 `module(9)` entPhysical class 的受控对象有关的附加信息。

- 物理表扩展

  它扩展物理实体表。它用于补充物理实体表中的 entPhysicalClass 列。若资源有 `other(1)` 的 entPhysicalClass 子类，但同时也是 sunPlat 模拟的类，即监视器或警报设备类，则此表标识该其 sunPlatPhysicalClass。

- 传感器表扩展

  它扩展设备表扩展。它提供与 `sensor(8)` entPhysicalClass 受控对象有关的附加信息。传感器类的子类通过下一级的扩展表表示，并通过该表的 sunPlatSensorClass 进行标识。

  - 二进制式传感器表扩展

    它扩展传感器表扩展。它提供与 `sensor(8)` entPhysicalClass 和 `binary(1)` sunPlatSensorClass 受控对象有关的附加信息。

  - 数字式传感器表扩展

    它扩展传感器表扩展。它提供与 `sensor(8)` entPhysicalClass 和 `numeric(2)` sunPlatSensorClass 受控对象有关的附加信息。

  - 离散式传感器表扩展

    它扩展传感器表扩展。它提供与 `sensor(8)`entPhysicalClass 和 `discrete(3)` sunPlatSensorClass 受控对象有关的附加信息。

- 风扇表扩展

  它扩展设备表扩展。它提供与 `fan(7)` entPhysicalClass 受控对象有关的附加信息。

- 警报表扩展

  它扩展设备表扩展。它提供与 `other(1)` entPhysicalClass 和 `alarm(8)` sunPlatPhysicalClass 受控对象有关的附加信息。

- 监视器表扩展

  它扩展设备表扩展。它提供与 `other(1)` entPhysicalClass 和 `watchdog(3)` sunPlatPhysicalClass 受控对象有关的附加信息。

- 电源扩展表

  它扩展设备表扩展。它提供与 `powerSupply(6)` entPhysicalClass 受控对象有关的附加信息。

表 5-3 所示为物理实体表的扩展示例。 entPhysicalIndex（此表的第一列）基于图 5-1 中所示的硬件资源分层结构示例。

ENTITY-MIB			SUN-PLATFORM-MIB											
entPhysicalIndex	entPhysicalClass				sunPlatPhysicalClass		sunPlatFanClass		sunPlatSensorClass					sunPlatPowerSupplyClass
1	机箱													
3	容器													
8	电源													G
12	电源													H
2	风扇						A							
	风扇						B							
	风扇						C							
5	传感器								D					
7	传感器								E					
11	传感器								F					
6	模块													
9	其它				警报									
10	其它				监视器									
4	其它				其它									
entPhysicalTable		sunPlatEquipmentTable	sunPlatEquipmentHolderTable	sunPlatCircuitPackTable	sunPlatPhysicalTable	sunPlatWatchdogTable	sunPlatFanTable	sunPlatAlarmTable	sunPlatSensorTable	sunPlatBinarySensorTable	sunPlatNumericSensorTable	sunPlatDiscreteSensorTable	sunPlatDiscreteSensorStatusTable	sunPlatPowerSupplyTable

表 5-3　物理实体扩展表

表 5-4　物理实体表扩展按键 （表 5-3）

引用	说明
A	风扇
B	冷却设备
C	吸热设备
D	二进制式
E	数字式
F	离散式
G	电源
H	电池

逻辑模型表扩展

SUN-PLATFORM-MIB 为逻辑实体表不支持的类提供了附加属性。它通过添加以下很少使用的扩展表来扩展逻辑实体表。

- 逻辑类表扩展

  它扩展 entLogicalTable 以定义逻辑实体类 *SunPlatLogicalClass* 及其状态 *sunPlatLogicalStatus*。 sunPlatLogicalTable 对于 entLogicalTable 中所有项均有效。逻辑类的计算机系统子类由下一级表扩展来表示。

  - 计算机系统表扩展

    此表扩展 entLogicalTable 以提供计算机系统实例通用的属性。 sunPlatUnitaryComputerSystem 表对于 entLogicalTable 中带有 `computerSystem(2)` sunPlat-LogicalClass 的行均有效。

    加载信息表 (*sunPlatInitialLoadInfoTable*) 中的一组数据项与每一计算机系统逻辑实体相关联。此组包含用于控制计算机系统引导设置的参数。

事件和警报日志表

SNMP 陷阱不一定能发送。有鉴与此，同时为了支持管理应用程序准确的追踪平台警报的当前状态， MIB 为每个受控对象维护一个当前问题列表。该表包含每个对象未解决的警报。警报条件清除后，这些警报将自动从表中清除。

SUN-PLATFORM-MIB 定义的日志可用于记录按照事件类型、警报类型或所影响实体分组的事件或警报。由于前面段落提到的原因， Sun Fire B1600 的实施使用这些日志维护每个所监视实体未解决的警报列表。

MIB 中每个可以生成警报的实体 （即，所有物理和逻辑实体）在日志表 (sunPlatLogTable) 中均有一对应项。该表提供当前问题列表的管理状态和控制。日志文件永远处于启用状态，并且没有大小限制。

日志表中的每一项可对应日志记录表中的零项或多项。这些项在第八章中详述。

事件记录

这些记录包含在 sunPlat 陷阱通知中。对模型所做的更改通过两种 SNMP 陷阱与管理应用程序进行通信——事件和警报。

事件

- 创建对象记录

  此记录表明资源添加到对象模型。
- 删除对象记录

  此记录表明资源从对象模型删除。
- 状态更改记录

  此记录表明资源的状态已更改。
- 整数属性值更改记录

  此记录表明由 `INTEGER` 类型的属性所模拟的资源特征的更改。该整数的正负取决于所影响的对象。
- 字符串属性值更改记录

  此记录表明由 `OCTET STRING` 类型的属性所模拟的资源特征的更改。
- OID 属性值更改记录

  此记录表明 `OBJECT IDENTIFIER` 类型的对象标识符属性的更改。

警报

- 通信警报记录

  此记录表明资源支持的通信服务发生故障。
- 环境警报记录

  此记录表明与资源有关的环境条件。
- 设备警报记录

  此记录表明资源发生故障。

- 处理错误警报记录
  此记录表明资源有相关的软件或处理故障。
- 服务性质警报记录
  此记录表明发生服务性质警报。
- 模糊警报记录
  此记录表明发生未知类型的警报。

第六章

物理模型

本章介绍 sunPlat 物理类的分层结构以及 SUN-PLATFORM-MIB 如何表示 sunPlat 模型中定义的受控物理对象类。

本章包括以下各节：

sunPlat 物理类分层结构

图 6-1 显示用于模拟 Sun Fire B1600 内硬件资源的 sunPlat 类的继承性分层结构。

物理实体父类提供了定义受控对象之间关系的属性。它同时还提供了与设备类属性对应的标准 SNMP 属性。

sunPlat 设备类从物理实体父类派生而来，它提供了用于故障监视的对应类中定义的附加属性。

sunPlat 设备支架和 sunPlat 电路包这两个类由 sunPlat 设备父类派生而来，分别表示插入设备的插件和组件。

sunPlat 设备类进一步专门化为提供由 DMTF 派生的类。

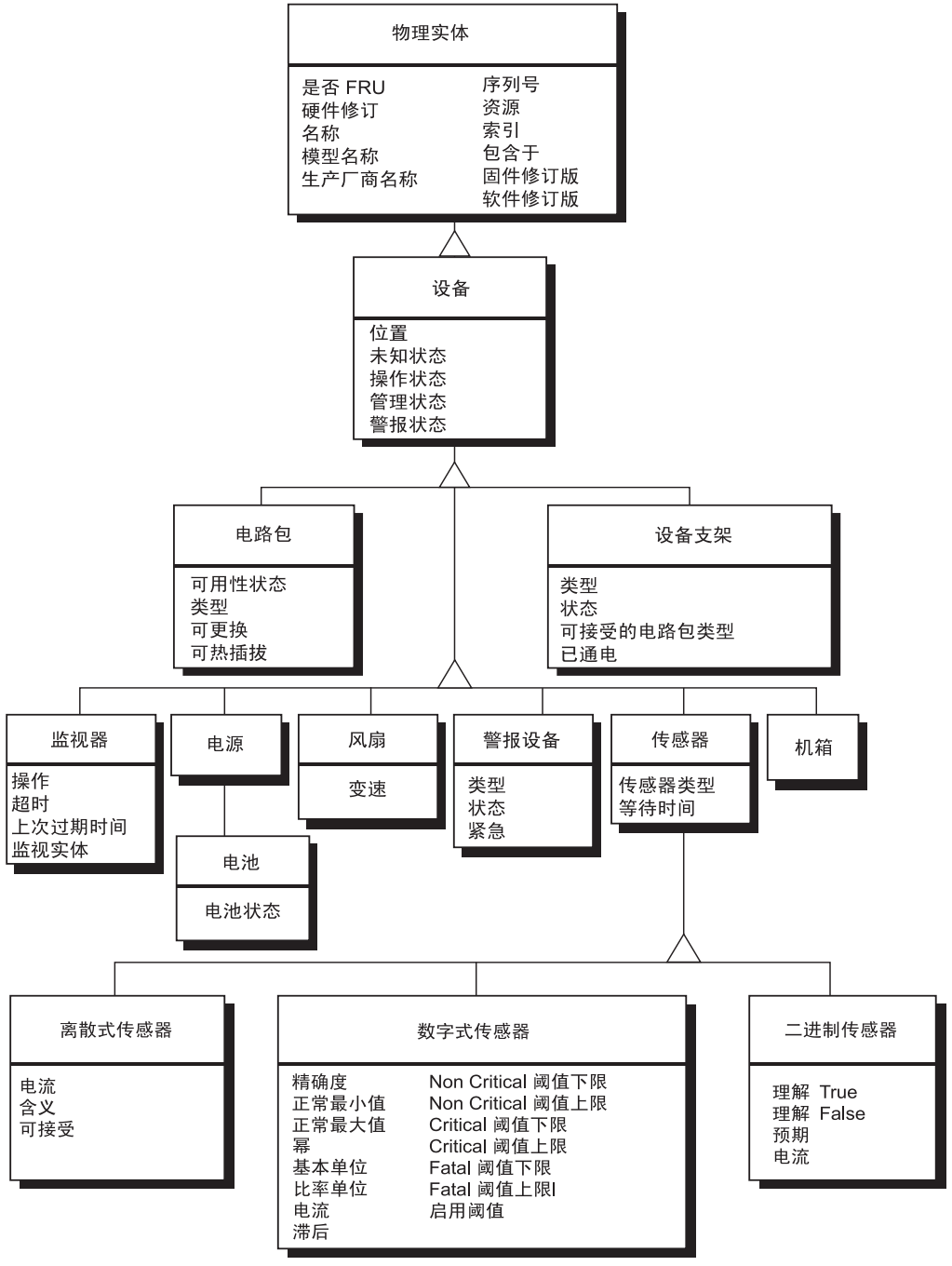

图 6-1　　sunPlat 物理资源继承性类图表

sunPlat 类定义

sunPlat 类的属性用于表示硬件资源的特征。管理器对资源的可用性和可操作性通过受控对象的*状态*表示。不同的 sunPlat 类有不同的属性，表示受控对象状态的各个方面。

物理实体

物理实体父类用于表示所有资源通用的特征。

注意 – 为清楚起见，以下属性名称中省略了 *entPhysical* 前缀。

- *资源*
  这是一个包含资源已知名称的文本字符串。通常这是在产品文档资料内或产品图例中描述资源的名称，或存储于固件中的名称。
- *是否 FRU*
  表示资源是否为现场可更换部件的布尔值。只有具有 *sunPlatCircuitPack* 类的硬件资源视为 FRU。
- *硬件修订*
  这是一个包含资源生产厂商的硬件修订信息的文本字符串。并非所有硬件资源均有相关的硬件修订信息。
- *名称*
  包含资源逻辑名称的文本字符串，操作系统以及相关的实用程序通过该名称识别资源。此名称可以是设备节点，也可以是应用它的系统实用程序使用的已定义名称。并非所有资源均有设备名称。
- *模型名称*
  包含生产厂商的客户可见部件号或部件定义的文本字符串。并非所有硬件资源均有相关联的部件号或定义。
- *序列号*
  包含该资源的生产厂商序列号的文本字符串。并非所有硬件资源均有相关联的序列号。
- *生产厂商名称*
  包含该资源生产厂商名称的文本字符串。并非所有硬件资源均有相关联的生产厂商名称。

物理实体父类同时还包含用于描述硬件资源分层结构的属性。

- *类*
  此枚举类型指明特定物理资源的通用硬件类型。此类支持的值由 ENTITY-MIB 定义。此属性可指示受控对象的相关表扩展。 ENTITY-MIB 类和 sunPlat 类之间的映射如表 6-1 所示：

表 6-1　逻辑实体父类的 “类”属性映射

entPhysical 类	sunPlat 类
机箱 (3)	*sunPlat 机箱*
底板 (4)	尚未实现
容器 (5)	*sunPlat 设备支架*
电源 (6)	*sunPlat 电源*
风扇 (7)	*sunPlat 风扇*
传感器 (8)	*sunPlat 传感器*，附加子类
模块 (9)	*sunPlat 电路包*
端口 (10)	未实施
堆栈 (11)	未实施
其它 (1)	*sunPlat 设备*，附加子类
未知 (2)	未实施

- *索引*
  此整数唯一地标识物理实体表中的数据项，即标识受控对象的项。这些值并未预先分配，并且每次调用代理程序时可能会不同。
- *包含于*
  此整数表示包含此受控对象的受控对象的*索引*属性。因此，该属性可模拟受控对象之间的关系。

注意 – 物理树形分层结构根部的对象 （通常为机箱）在物理上不包含于该表的其它实体中。为表明这一点，其 *entPhysicalContainedIn* 值设为 `0`。

- *固件修订版*
  此文本字符串包含资源生产厂商的固件修订版信息。并非所有硬件资源均有相关的固件修订版信息。
- *软件修订版*
  此文本字符串包含资源生产厂商的软件修订版信息。并非所有硬件资源均有相关的软件修订版信息。

sunPlat 设备类

sunPlat 设备类用于表示所有硬件资源通用的特征。此类包含表示配置和一般运作状态信息的属性。此类进一步细分，以提供特定类型资源的更详细配置信息和监视数据。

entPhysicalClass 类取决于所代表的子类。

sunPlat 设备类具有以下属性：

注意 – 为清楚起见，以下属性名称中省略了 *sunPlatEquipment* 前缀。

- *管理状态*
  此读写属性选取以下枚举值之一，表示资源当前的管理状态：
  - `locked(1)`
  - `unlocked(2)`
  - `shuttingDown(3)`
- *操作状态*
  此只读属性是枚举类型，它表示资源物理上是否已安装以及是否可提供服务。本属性与受控对象的*状态*有关，可选取如表 6-2 所示的值：

表 6-2 操作状态属性值

属性值	说明
`disabled(1)`	资源完全不可操作，并且无法为用户提供服务。
`enabled(2)`	资源完全或部分可操作并且可使用。

- *警报状态*
  此只读属性取值为枚举型，它表示资源的当前警报状态。它表明受控对象的所有未解决的警报的最高严重性。该属性可以取值如下：
  - `critical(1),`
  - `major(2),`
  - `minor(3),`
  - `indeterminate(4),`
  - `warning(5),`
  - `pending(6),`
  - `cleared(7)`

- *未知状态*
  此只读属性表示其它状态属性可能无法反映资源的真实状态。该属性取值为布尔型，它表示受控对象是否可以准确报告资源的故障。若资源无法准确反映其*状态*，则此属性设置为 `true`。
- *位置名称*
  此只读属性包含资源的定位器。对于直接包含在机箱内的资源，此属性值与插槽的图例以及产品的文档资料有关；或者提供机箱内资源位置的空间位置信息。其它硬件资源的*位置*通常对应资源所包含的受控对象的*名称*。

sunPlat 电路包类

sunPlat 电路包类用于表示可更换资源或 FRU 的一般特征。可更换资源定义为硬件模块，其用途是将内部硬件资源打包为可识别的板型。通常， FRU 会有一个已定义的板型和物理外观。它可以是可插拔、可拆卸部件，可插到连接器上；也可以更长久的驻留于支架内；还可以固定到托架、机架或搁板。

此类具有 entPhysicalClass `module(9)`。

sunPlat 电路包类具有以下属性：

注意 – 为清楚起见，以下属性名称中省略了 *sunPlatCircuitPack* 前缀。

- *类型*
  此只读属性是文本字符串，用于评估资源与其容器的兼容性。此属性可标识资源的功能和板型特征。
- *可用性状态*
  此只读属性进一步限定受控对象的*操作状态*。它是使用 BITS 语法的对象，可取值表 6-3 中所示的零或多组值。并非所有值均适用于受控对象的每一类。此值影响受控对象的*状态*属性。

表 6-3 可用性状态属性值

属性值	位编号	十六进制	说明
`inTest(0)`	0	80	资源正处于测试过程中。
`failed(1)`	1	40	资源发生内部故障，无法进行操作。*操作状态*为 `disabled(1)`。
`powerOff(2)`	2	20	资源所需电源尚未通电。
`offLine(3)`	3	10	资源要求执行常规操作使其联机且可用。*操作状态*为 `disabled(1)`。

表 6-3　可用性状态属性值 *（接上页）*

属性值	位编号	十六进制	说明
offDuty(4)	4	08	资源由内部控制进程设为非活动状态。
dependency (5)	5	04	由于所依赖的某些其它资源不可用，因此资源无法进行操作。*操作状态* 为 disabled(1)。
degraded(6)	6	02	资源的可用服务在某些方面降级，如速度或操作性能。不过，资源的服务仍可用。*操作状态* 为 enabled(2)。
notInstalled (7)	7	01	受控对象代表的资源不存在或不完整。*操作状态* 为 disabled(1)。

- *可更换*
  此只读属性取值为布尔型，它表示该资源是否为可更换部件。
- *可热插拔*
  此只读属性取值为布尔型，它表示该可更换资源是否可热插拔。

sunPlat 设备支架

sunPlat 设备支架类用于表示支撑可拆卸硬件资源的硬件资源的特征。

此类具有 entPhysicalClass container(5)。

sunPlat 设备支架类具有以下属性：

注意 – 为清楚起见，以下属性名称中省略了 *sunPlatEquipmentHolder* 前缀。

- *类型*
  此只读属性为枚举型，它表示资源的支架类型，如表 6-4 所示：

表 6-4　设备支架类型属性值

属性值	说明
bay(1)	支架通常是机架内的一个垂直空间部件，它包括用于支撑通讯设备的搁板或托架。 SunPlat 将其在机箱内的作用理解为需要电缆进行信号连接的物理插口。
shelf(2)	机架内用于固定通讯设备的水平支撑物或子机架。

表 6-4　设备支架类型属性值　*（接上页）*

drawer(3)	机架内用于支撑通讯设备的水平附件。
slot(4)	用于可拆卸设备信号连接的带完整连接器的物理插口。
rack(5)	自包含附件内用于支撑通讯设备、支架以及电缆管理系统的支撑结构。

- *可接受类型*
  此只读属性为文本字符串列表，它表示支架支撑的可拆卸资源 （电路包）类型。对这些类型，将进行与可拆卸资源的*类型* 属性的兼容性检测。
- *状态*
  此只读属性为枚举型，它表示支架及其所包含的可更换硬件资源 （电路包）的状态，如表 6-5 所示：

表 6-5　设备支架状态属性值

属性值	说明
holderEmpty(1)	支架中没有可拆卸资源
inTheAcceptableList(2)	支架包含一个可拆卸资源，它是*可接受的电路包类型* 列表中的类型之一
notInTheAcceptableList (3)	支架包含一个网络元素可识别的可拆卸资源，但该资源并非*可接受的电路包类型* 列表中的类型之一
unknownType(4)	支架包含一个无法识别的可拆卸资源

- *已通电*
  此只读属性为枚举型，它表示资源的电源状态。可能的取值有：
  - other(1)
  - unknown(2)
  - powerOff(3)
  - powerOn(4)

sunPlat 电源

sunPlat 电源类用于表示电源。它并未扩展 sunPlat 设备类的特征。电源通常包含表示所监视的特性 （例如电压、电流和温度）的传感器。它还可以包含其它硬件资源，如风扇。通过受控对象之间的相互关系进行模拟。

如果电源是可拆卸资源，则它通过 sunPlat 电路包类的受控对象进行模拟。

此类具有 entPhysicalClass `powerSupply(6)`。

sunPlat 电源类具有以下属性：

注意 – 为清楚起见，以下属性名称中省略了 *sunPlatPowerSupply* 前缀。

- *类*
  此只读属性为枚举型，它表示电源的类，可取以下值：
  - `other(1)`
  - `powerSupply(2)`
  - `battery(3)`

sunPlat 电池

sunPlat 电池类用于表示通过电池供电的电源。

此类具有 entPhysicalClass `powerSupply(6)` 和 SunPlatPowerSupplyClass `battery(3)`。

sunPlat 电池类具有以下属性：

注意 – 为清楚起见，以下属性名称中省略了 *sunPlatBattery* 前缀。

- *状态*
  此只读属性为枚举型，它表示电池的状态，可采用以下值：
  - `other(1)`
  - `unknown(2)`
  - `fullyCharged(3)`
  - `low(4)`
  - `critical(5)`
  - `charging(6)`
  - `chargingAndHigh(7)`
  - `chargingAndLow(8)`
  - `chargingAndCritical(9)`
  - `undefined(10)`
  - `partiallyCharged(11)`

sunPlat 监视器

sunPlat 监视器类用于表示计时器硬件资源的特征，它允许硬件监视操作系统或应用程序的状态。

此类具有 entPhysicalClass `other(1)` 和 SunPlatPowerSupplyClass `watchdog(3)`。

sunPlat 监视器类具有以下属性：

注意 – 为清楚起见，以下属性名称中省略了 *sunPlatWatchdog* 前缀。

- *超时*
  此只读属性是一个整数，它表示以毫秒为单位的时间间隔，监视器经过该时间后若不复位则将超时。
- *操作*
  此只读属性为枚举型，它表示若监视器超出*超时*指定的时间间隔尚未复位时应采取的操作。可能的取值如表 6-6 所示。

表 6-6　监视器操作属性值

操作	说明
`statusOnly(1)`	监视器可通过软件读取，但不执行任何操作
`systemInterrupt(2)`	监视器对所监视的系统生成硬件中断
`systemReset(3)`	监视器复位所监视的系统
`systemPowerOff(4)`	监视器将所监视的系统断电
`systemPowerCycle(5)`	监视器将所监视的系统断电，然后再重新通电

- *上次过期时间*
  此只读属性表明监视器上次过期的日期和时间。
- *监视实体*
  此只读属性为枚举型，它表示可通过监视器进行监视的实体。可能的取值有：
  - `unknown(1)`
  - `other(2)`
  - `operatingSystem(3)`
  - `operatingSystemBootProcess(4)`
  - `operatingSystemShutdownProcess(5)`
  - `firmwareBootProcess(6)`
  - `biosBootProcess(7)`

- application(8)
- serviceProcessors(9)

sunPlat 警报

sunPlat 警报类用于表示发出有关问题的指示的硬件资源的特征，例如蜂鸣、 LED、中继、振动和软件警报。

此类具有 entPhysicalClass other(1) 和 SunPlatPowerSupplyClass alarm(2)。

sunPlat 警报类具有以下属性：

注意 – 为清楚起见，以下属性名称中省略了 *sunPlatAlarm* 前缀。

- *类型*
  此只读属性为枚举型，它表示警报条件下的通信方式。可能的取值如表 6-7 所示。

表 6-7 警报类型属性值

属性值	说明
other(1)	警报设备并非以下类型之一
audible(2)	警报对设备的更改发出声音
visible(3)	警报引起设备上的可见变化
motion(4)	警报引起设备活动
switch(5)	警报引起电子信号变化

- *状态*
  此只读属性为枚举型，它表示警报的状态。可能的取值如表 6-8 所示。

表 6-8 警报状态属性值

属性值	说明
unknown(1)	警报状态未定义或无法觉察
off(2)	警报处于非活动状态
steady(3)	警报处于活动状态
alternating(4)	警报在活动与非活动状态之间循环

- *紧急*
  此只读属性为枚举型，它表示警报闪烁、振动和 / 或发出声音的相对频率。可能的取值有：
  - `other(1)`
  - `unknown(2)`
  - `notSupported(3)`
  - `informational(4)`
  - `nonCritical(5)`
  - `critical(6)`
  - `unrecoverable(7)`

sunPlat 风扇

sunPlat 风扇类用于表示活动的冷却设备的特征。风扇通常包含一个表示其转速的传感器。这通过 sunPlat 风扇受控对象和 sunPlat 传感器类的转速器受控对象之间的物理树形关系来模拟。

此类具有 entPhysicalClass `fan(7)`。

sunPlat 风扇类具有以下属性：

注意 – 为清楚起见，以下属性名称中省略了 *sunPlatFan* 前缀。

- *类*
  此只读属性为枚举型，它表示冷却设备的类，可取以下值：
  - `other(1)`
  - `fan(2)`
  - `refrigeration(3)`
  - `heatPipe(4)`

sunPlat 传感器

sunPlat 传感器父类用于表示测量其它硬件资源特征的硬件资源的一般特征。

此类具有 entPhysicalClass `sensor(8)`。

sunPlat 传感器类具有以下属性：

注意 – 为清楚起见，以下属性名称中省略了 *sunPlatSensor* 前缀。

- *类*
  此只读属性为枚举型，它表示传感器的类，可采用以下值：
  - `binary(1)`
  - `numeric(2)`
  - `discrete(3)`
- *类型*
  此只读属性为枚举型，它表示传感器测量的属性。*类型* 的一些可能取值如表 6-9 所示。

表 6-9 传感器类型属性值

类型	说明
`temperature(3)`	测量环境温度的传感器
`voltage(4)`	测量电压的传感器
`current(5)`	测量电流的传感器
`tachometer(6)`	测量设备速度 / 转速的传感器
`counter(7)`	计数已定义事件的通用传感器

- *潜伏期*
  此只读属性表明以下内容：
  - 传感器巡回检测的位置，此整数表示以毫秒为单位的更新时间间隔。
  - 传感器事件驱动的位置，此值表示处理该事件的最大预期潜伏期。

sunPlat 二进制传感器

sunPlat 二进制传感器类用于表示返回二进制输出的传感器特征。它扩展 sunPlatSensor 表以提供专用于二进制传感器的属性。

此类具有 entPhysicalClass `sensor(8)` 和 SunPlatPowerSupplyClass `binary(1)`。

sunPlat 二进制传感器类具有以下属性：

注意 – 为清楚起见，以下属性名称中省略了 *sunPlatBinarySensor* 前缀。

- *当前值*
  此只读属性为布尔型，它表示传感器时间最近的取值。
- *预期*
  此只读属性为布尔型，它表示传感器的预期值。
- *理解 True*
  此只读属性为文本字符串，它表示如何理解来自传感器的 `true` 值。
- *理解 False*
  此只读属性为文本字符串，它表示如何理解来自传感器的 `false` 值。

sunPlat 数字传感器

sunPlat 数字传感器类用于表示返回数字的传感器特征。数字传感器的取值通过下面定义的测量单位限定：

测量单位 = *基本单位* *10 *幂*

这种限制允许诸如毫安培和微伏之类的测量单位。如果定义了*比率单位*，则测量单位可以进一步定义如下：

测量单位 = *基本单位* *10 *幂* 每 *比率单位*

这种限定允许诸如 rpm 和 km/hr 之类的测量单位。

此类具有 entPhysicalClass `sensor(8)` 和 SunPlatPowerSupplyClass `numeric(2)`。

sunPlat 数字传感器类具有以下属性：

注意 – 为清楚起见，以下属性名称中省略了 *sunPlatNumericSensor* 前缀。

- *基本单位*
  此只读属性为枚举型，它表示测量单位，在限制之前，如上述定义。此类型值的示例如下：
  - `degC(3)`
  - `volts(6)`
  - `amps(7)`
- *幂*
  此只读属性为整数，通过 10 的次幂缩放*基本单位*。例如，如果 *sunPlatNumericSensorBaseUnits* 设为 `volts` 而 *sunPlatNumericSensorExponent* 设为 `-6`，则返回值的单位为 `microvolts`。

- *比率单位*
  此只读属性为枚举型，它表示传感器测量的是绝对值（当该值为 none 时）还是比率。后一种情况下，*sunPlatNumericSensorBaseUnits* 指定的单位表达为“每单位时间”。例如，如果 *sunPlatNumericSensorBaseUnits* 设为 degC 而 *sunPlatNumericSensorRateUnits* 设为 perSecond，则得到的值以 degC/second 为单位表示。

  此类型值的示例如下：
  - perMicrosecond(2)
  - perMillisecond(3)
  - perSecond(4)
  - perMinute(5)
  - perHour(6)
  - none(1)
- *当前值*
  此只读属性为整数，它表示时间最近的传感器取值。
- *正常最小值*
  此只读属性为整数，它表示定义的阈值，低于该值时，传感器无法表示。此值以上面定义的测量单位表示。该属性对于某些传感器不可用。
- *正常最大值*
  此只读属性为整数，它表示定义的阈值，超出该值时，传感器无法表示。此值以上面定义的测量单位表示。该属性对于某些传感器不可用。
- *精确度*
  此只读属性为整数，以两位的百分数表示传感器测量属性的错误级别。该值根据传感器的读数在其动态变化范围内是否为线性而变化。
- *Non Critical 阈值下限*
  此只读属性为整数，它表示发生 nonCritical 情况的阈值下限。
- *Non Critical 阈值上限*
  此只读属性为整数，它表示发生 nonCritical 情况的阈值上限。
- *Critical 阈值下限*
  此只读属性为整数，它表示发生 critical 情况的阈值下限。
- *Critical 阈值上限*
  此只读属性为整数，它表示发生 critical 情况的阈值上限。
- *Fatal 阈值下限*
  此只读属性为整数，它表示发生 fatal 情况的阈值下限。
- *Fatal 阈值上限*
  此只读属性为整数，它表示发生 fatal 情况的阈值上限。
- *滞后*
  此只读属性为整数，它描述阈值的滞后。

- *启用阈值*
  只读属性，写入时将传感器复位为其缺省值。

sunPlat 离散传感器

sunPlat 离散式传感器类用于表示无法通过 sunPlat 数字传感器或 sunPlat 二进制传感器类表示的传感器。

此类具有 entPhysicalClass `sensor(8)` 和 SunPlatPowerSupplyClass `discrete(3)`。

该类包含两个表。sunPlatDiscreteSensor 表包含一个属性 *sunPlatDiscreteSensorCurrent*，该属性表示传感器的当前状态， 以 sunPlatDiscreteSensorStates 表中索引的形式表示。

sunPlat 离散式传感器类具有以下属性：

注意 – 为清楚起见，以下属性名称中省略了 *sunPlatDiscreteSensorState* 前缀。

- *索引*
  此只读属性为数字，它表示 sunPlatDiscreteSensorStates 表中行的索引，该索引标识传感器的状态。
- *含义*
  此只读属性为字符串，它表示 sunPlatDiscreteSensorStatesTable 表中对应行所代表的状态。
- *可接受*
  此只读属性为布尔型，它表示表中的行代表的状态是否可接受。

sunPlat 机箱

sunPlat 机箱类用于表示主要附件。它并未扩展 sunPlat 设备类的特征。机箱包含所有可模拟的硬件资源，其它资源无法包含它。

此类具有 entPhysicalClass `chassis(3)`。

第七章

逻辑模型

本章介绍 sunPlat 逻辑类分层结构以及 SUN-PLATFORM-MIB 如何表示 sunPlat 模型中定义的受控对象类。

本章包括以下各节：

sunPlat 逻辑类分层结构

图 7-1 显示 sunPlat 逻辑类的继承性分层结构。

逻辑实体类提供所有逻辑对象的一般信息。

单机系统类添加与模拟计算机系统 （例如， Sun Fire B1600 机箱内的 Sun Fire B100s blade）的电源状态报告有关的属性，它也可用来实现强制复位。

管理域类不添加附加属性，但它可用于表示逻辑对象，该逻辑对象表示与模拟系统的管理性联系。对于 Sun Fire B1600 平台，它用于表示系统控制器。

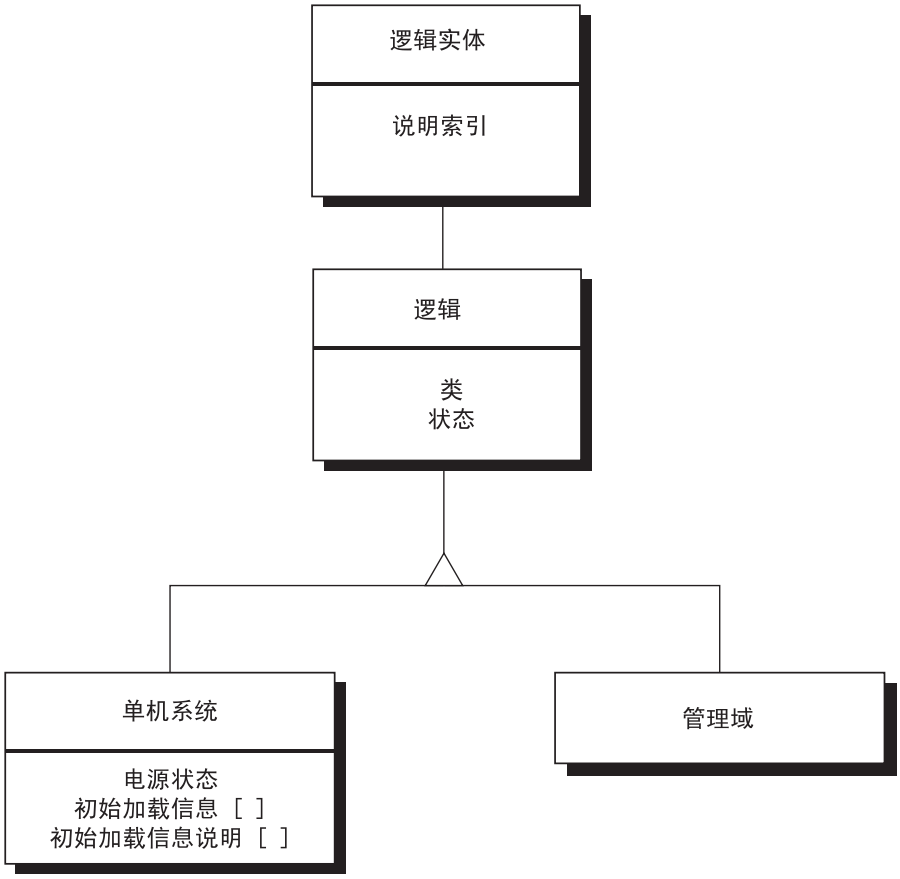

图 7-1　　sunPlat 逻辑资源继承性类图表

sunPlat 逻辑类定义

sunPlat 逻辑类的属性用于表示逻辑资源的特征。这种资源表示*高值*对象，例如多域系统中的域。资源对管理器的可用性和可操作性可通过受控对象的状态表示。不同的 sunPlat 类有不同的属性，表示受控对象状态的各个方面。

逻辑实体

此类表示提供身份信息的逻辑实体。重要的对象包括：

注意 – 为清楚起见，以下对象名称中省略了 *entLogical* 前缀。

- *说明*

  此对象标识受控对象的类型。

- *TAddress*

  此对象提供 IP 地址和 UDP 端口号。可通过它直接管理实体。对于 Sun Fire B1600 系统中的 Sun Fire B100s blade，此对象给出 blade 的 IP 地址以及端口 161，通过它联系 blade 的标准 Solaris SNMP 代理程序。

逻辑

此类表示此逻辑实体所代表的资源的状态类型。该类包含以下对象：

注意 – 为清楚起见，以下对象名称中省略了 *sunPlatLogical* 前缀。

- *类*

  此属性为枚举类型，它表明逻辑类的类型，可采用以下值：

  - `other(1)`
  - `computerSystem(2)`
  - `adminDomain(3)`

- *状态*

  此属性为枚举类型，它表明逻辑类的状态。它可采用以下值：

  - `ok(1)`
  - `error(2)`
  - `degraded(3)`
  - `unknown(4)`
  - `predFail(5)`
  - `starting(6)`
  - `stopping(7)`
  - `service(8)`
  - `stressed(9)`
  - `nonRecover(10)`
  - `noContact(11)`
  - `lostComm(12)`
  - `stopped(13)`

sunPlat 单机系统

此类的具体属性可使用 sunPlatUnitaryComputerSystemTable 表示。它具有 `computerSystem(2)` 的 sunPlat 逻辑类。

该类包含以下对象：

注意 – 为清楚起见，以下对象名称中省略了 *sunPlatUnitaryComputerSystem* 前缀。

- *电源状态*

  此属性表明读取时的当前电源状态。它还可启用电源状态的远程控制，例如，打开或关闭 blade，或断开后接通电源以实现强制复位。

  该属性可采用以下值：

  - `unknown(1)`
  - `fullPower(2)`
  - `psLowPower(3)`
  - `psStandby(4)`
  - `psOther(5)`
  - `powerCycle(6)`
  - `powerOff(7)`
  - `psWarning(8)`
  - `hibernate(9)`
  - `softOff(10)`
  - `reset(11)`

- *应用设定*

  对该属性的写入操作将应用一组缺省的或自定义的引导参数。

  与 sunPlatUnitaryComputerSystemTable 中的每个数据项相关联的是 *sunPlatInitialLoadInfoTable* 中的一组数据项，它同时定义当前的引导参数设定以及另一组备用的设定，该设定可通过对 *sunPlatUnitaryComputerSystemApplySettings* 对象执行写操作来应用。

sunPlat 管理域

此类并未对逻辑实体类添加任何属性，因此也没有相关联的 MIB 对象。该类具有 `adminDomain(3)` 的 sunPlat 逻辑类。

第八章

sunPlat 通知

本章介绍 SUN-PLATFORM-MIB 中定义的 SunPlat 通知类及其属性。

sunPlat 通知类是由代理程序发送到已注册的网络管理器的异步消息。与巡回检测受控对象相比，它们能更有效地传送事件信息。

本章包括以下各节：

sunPlat 通知类分层结构

图 8-1 显示 sunPlat 通知类的继承性分层结构。

通知类组通过抽象和具体类的分层结构来表示。在此分层结构中，展开这些类的通用属性。

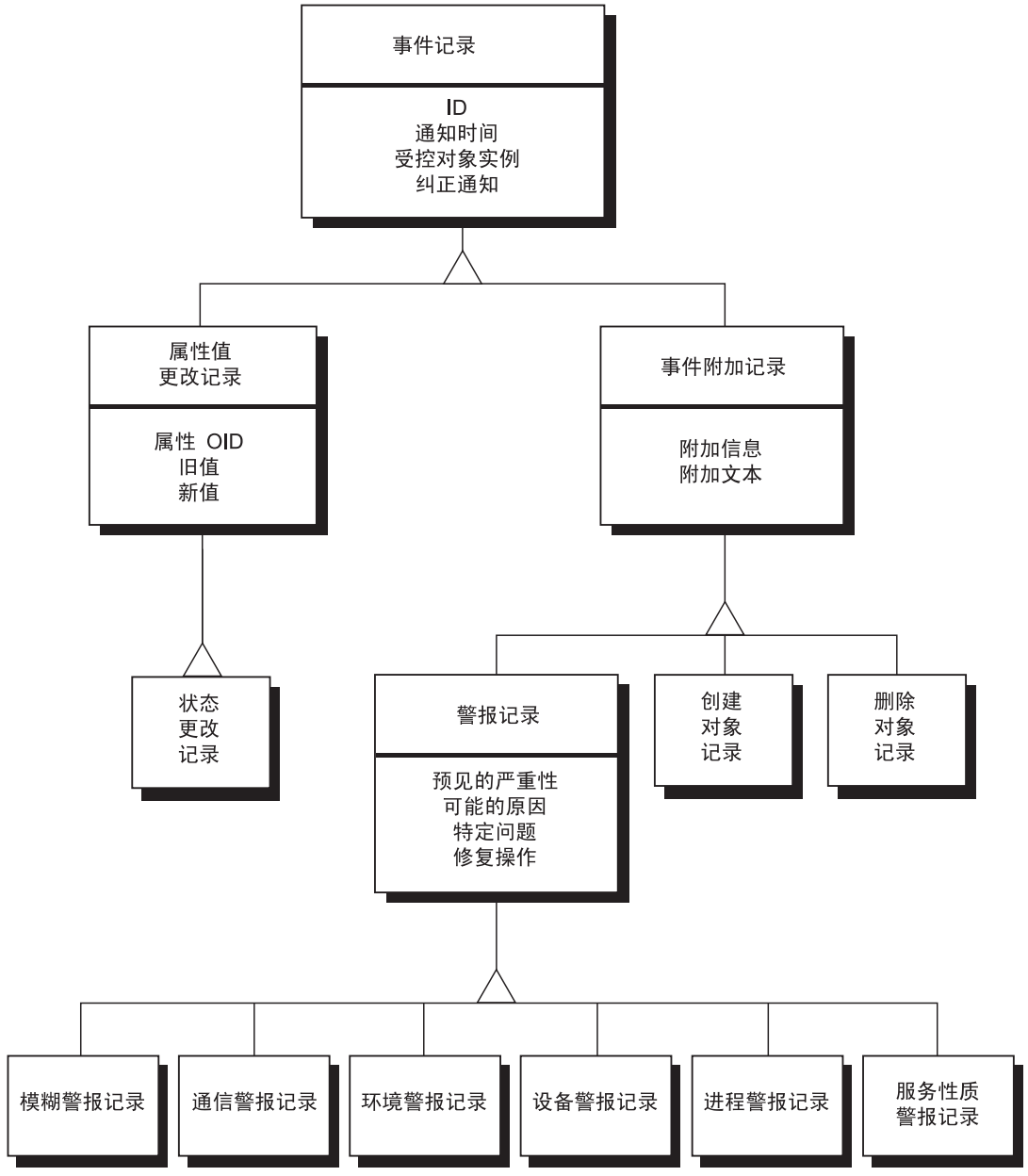

图 8-1　事件记录继承性类图表

sunPlat 类定义

sunPlat 事件记录

sunPlat 事件记录父类表示所有通知通用的属性。此类进一步细分以提供和其记录的特定事件相关的其它信息。

sunPlat 事件记录具有以下属性：

注意 – 为清楚起见，以下属性名称中省略了 *sunPlatLogRecord* 前缀。

- *ID*
  唯一标识通知并表明代理程序生成通知顺序的一个整数。请注意，代理程序不保证其顺序可以反映生成通知的基本事件的顺序。
- *通知时间*
  此只读属性为时间戳，它标识当前生成通知的时间。
- *受控对象实例*
  此只读属性为一个 OID，可为 MIB 中表示事件相关资源的数据项提供直接引用。
- *相关通知*
  此只读属性是以逗号分隔的 ID 值列表，可标识与此事件相关联的其它事件。

sunPlat 事件附加记录

sunPlat 事件附加记录父类表示以下事件发生时生成的通知的通用属性：

- 创建对象
- 删除对象
- 警报

此类可进一步细分以提供适用于其记录的特定事件的附加信息。

sunPlat 事件附加记录具有以下属性：

注意 – 为清楚起见，以下属性名称中省略了 *sunPlatLogRecord* 前缀。

- *附加信息*
  此只读属性是一个可选的对象 OID，可提供与此通知有关的附加信息。
- *附加文本*
  一个只读属性的可选字符串，提供与此通知有关的附加信息，通过其标签和 `entPhysical` 名称标识所影响的组件。

sunPlat 对象创建记录

sunPlat 对象创建记录类用于表明相关资源的分层结构下添加了一个资源；此情况可能是由热插拔事件引起。*附加信息* 属性包含 *物理实体表* 中表示添加资源的数据项的 OID。

在管理器对象实例 `0.0` 下创建逻辑对象。

sunPlat 对象删除记录

sunPlat 对象删除记录用于表明相关资源分层结构下的资源已删除。*附加信息* 属性包含 *物理实体表* 中表示所删除资源的数据项的 OID。

注意 – 此 OID 不再有效，但仍可用于接收管理器。

---

sunPlat 警报记录

sunPlat 警报记录父类表示所有表示警报的通知通用的附加属性。

此类可进一步细分以标识出现的警报类。

sunPlat 警报记录类具有以下属性：

注意 – 为清楚起见，以下属性名称中省略了 *sunPlatAlarmRecord* 前缀。

- *预见的严重性*
  此只读属性为枚举型，它定义了 6 种严重性级别，用于表明问题对资源服务的影响。这些值如表 8-1 中所示。

表 8-1　sunPlat 警报记录预见的严重性值

预见的严重性	说明
indeterminate(1)	无法确定警报的严重性级别
critical(2)	发生影响服务的情况，并需要立即执行纠错操作。
major(3)	发生影响服务的情况，并需要执行紧急纠错操作。
minor(4)	发生影响服务的情况，应按顺序执行纠错操作，以避免引发更严重的故障。
warning(5)	检测到一个可能的或迫近的故障情况会影响服务，应采取操作以避免引发更严重的故障。
cleared(6)	此值清除此资源的同一警报类的所有警报，这些警报具有相同的*可能原因*和*特定问题*（若指定）。

- *可能原因*
  此只读属性是一个可选的枚举型，它进一步限定了关于生成警报的情况类型。此类型值的示例如下：
  - coolingSystemFailure(134)
  - IODeviceError(75)
  - powerProblem(58)
  - softwareProgramError(283)
- *特定问题*
  此只读属性是一个可选的文本字符串，它标识警报*可能原因*的详细信息。
- *修复操作*
  此只读属性是一个字符串，列出推荐的修复操作。

sunPlat 模糊警报记录

sunPlat 模糊警报记录类并不扩展 sunPlat 警报记录类提供的信息。此类用于记录无法归到以下任一类的所有警报：

sunPlat 通信警报记录

sunPlat 通信警报记录类并不扩展 sunPlat 警报记录类提供的信息。此类用于记录相关资源检测到的通信错误。

sunPlat 环境警报记录

sunPlat 环境警报记录类并不扩展 sunPlat 警报记录类提供的信息。此类用于记录相关资源检测到的环境问题。

sunPlat 设备警报记录

sunPlat 设备警报记录类并不扩展 sunPlat 警报记录类提供的信息。此类用于记录相关资源检测到的故障。

sunPlat 进程警报记录

sunPlat 进程警报记录类并不扩展 sunPlat 警报记录类提供的信息。此类用于记录相关资源检测到的软件或进程故障。

sunPlat 服务性质警报记录

sunPlat 服务性质警报记录类并不扩展 sunPlat 警报记录类提供的信息。此类用于记录相关资源检测到的服务性质更改。

sunPlat 属性值更改记录

sunPlat 属性值更改记录父类表示通知 （代表相关资源中属性更改）的附加通用属性。

此类可进一步细分，表示每种可能的属性类型。

sunPlat 属性值更改记录具有以下属性：

注意 – 为清楚起见，以下属性名称中省略了 *sunPlatLogRecordChange* 前缀。

- *OID*
  此只读属性是一个 OID，它提供对物理实体表或逻辑实体表中对象的直接引用，表示受控对象的已更改其值的属性。

根据所影响属性的语法，新值和旧值使用以下对象成对表示：

- *新整数*
  此只读属性标识受控对象已更改属性的新 INTEGER 值。该类型的正负取决于所更改的属性。
- *旧整数*
  此只读属性标识受控对象已更改属性的旧 INTEGER 值。该类型的正负取决于所更改的属性。
- *新字符串*
  此只读属性标识属性更改通知中的新 OCTET STRING 值。
- *旧字符串*
  此只读属性标识属性更改通知中的旧 OCTET STRING 值。
- *新 OID*
  此只读属性标识属性更改通知中的新 OBJECT IDENTIFIER 值。
- *旧 OID*
  此只读属性标识属性更改通知中的旧 OBJECT IDENTIFIER 值。

sunPlat 状态更改记录

sunPlat 状态更改记录类并不扩展 sunPlat 属性值更改记录类提供的信息。此类用于表明受控对象属性的更改，该属性反映资源*状态*的某一方面。

第二部分　安装和配置

第九章

管理软件组件

本章介绍组成 Sun Fire B1600 管理软件的组件，并列出了安装 SNMP 软件的要求。

本章包括以下各节：

系统管理选项

以下系统管理选项可用于 Sun Fire B1600：

- 使用 SNMP 监视和控制系统
- 支持安全管理的 SNMPv3 功能
- 使用 Sun Management Center 3.0[1] 监视系统

检测

根据不同的平台类型，您可以使用以下代理程序：

- 运行在 Sun Fire B100s blade 上的域代理程序 （域硬件监视）

1. 有关详细说明 （包括安装和配置），请参考 *《用于 Sun Fire B1600 的 Sun Management Center 3.0 补充资料》*（部件号：817-2540-10）。

软件安装在所监视服务器的本地，只监视该服务器。对于 Sun Fire B1600，每台 blade 均单独监视，并且每个代理程序实例只能查看一台 blade。

- 通过系统控制器代理的平台代理程序 （平台硬件监视）

  软件安装在远程 （平台代理）服务器上，通过系统控制器访问平台检测程序。这种配置允许您监视系统控制器管理的所有硬件。对于 Sun Fire B1600，可以监视 blade 的整个搁板，包括各种类型的所有 blade、电源和系统控制器。

图 9-1 中，Sun Fire B1600 搁板 A 和 B 使用平台硬件监视，而 Sun Fire B1600 搁板 C 则使用域硬件监视。

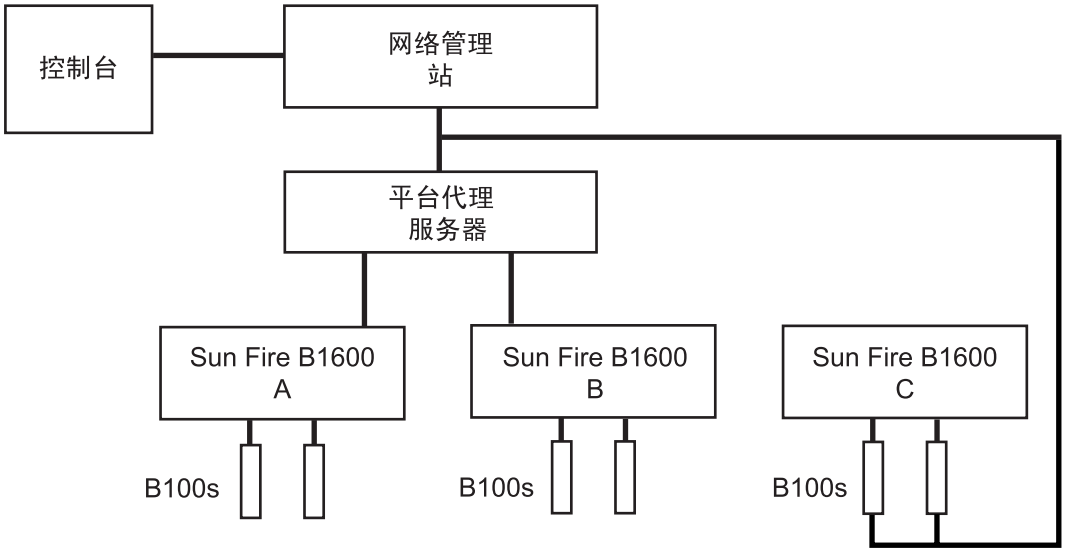

图 9-1　域和平台硬件监视示例

系统要求

在安装 SNMP Management Agent 之前，请确保您的系统满足本节所述的前提条件和从属条件。

操作环境

SNMP Management Agent 软件需要 Solaris 8 Update 3 或其后续版本。

磁盘空间要求

平台代理服务器上至少要有 512 MB 的可用磁盘空间 （推荐 1.0GB）。

增补程序

除了标准 Solaris 操作系统软件以外，还必须安装以下增补程序：

Solaris 8

Sun Fire B100s blade 不需要增补程序。

安装 SNMP Management Agent 软件前，必须在平台代理服务器和 blade 上安装 Java 1.4 （平台和域硬件监视均需要安装）（请参见第 61 页的 “Java 环境”）。

Solaris 9

不需要增补程序。

Java 环境

为了完全监视 Sun Fire B100s blade，必须在监视的每台 Sun Fire B100s blade 以及平台代理服务器上预装 Java J2SE 1.4 组件。

注意 – 此安装可升级任何现有的 J2SE 软件。若您不希望升级该软件 （例如，因为有应用程序限制于缺省的 J2SE 1.3.1 版本），则可以安装 J2RE，使其与缺省的系统 J2SE 共存。这需要 Sun SNMP Management Agent 软件的附加配置，将在附录 A 中介绍。

若您只监视 Sun Fire B1600 搁板，而不使用目标检测，则只需在平台代理服务器上预装 Java J2SE 1.4 组件即可。这种情况下，无法提供硬盘驱动器、 CPU 信息和以太网 MAC 地址的检测。

为确保 Java 1.4 文件安装在正确的位置 （`/usr/j2se`），请使用 `j2sdk-1_4_0_03-solaris-sparc.tar.Z` 软件包进行安装。

该文件可在以下网址获得：

`http://java.sun.com/j2se/1.4/download.html`

选择 Solaris SPARC 32-bit tar.Z 的 SDK 下载

请遵循上述下载位置提供的指导进行操作。

注意 – 在撰写本文档时此文件名如上。请确保获得此文件的最新版本。文件名的格式为 j2sdk-1_4_0_< 版本号 >-solaris-sparc.tar.Z，其中 < 版本号 > 是软件的修订版号。

由于此安装会替换系统 J2SE，为确保原有的 Java 应用程序可继续正确运行，您还必须安装 64 位 J2SE 1.4 软件包，它包含在文件 j2sdk-1_4_0_< 版本号 >-solaris-sparcv9.tar.Z 中。

警告 – Solaris 8 上， J2SE 1.4 会替换 J2SE 1.3.1。因此在安装 J2SE 1.4 之前，必须先卸载 J2SE 1.3.1。若您安装 Solaris 8 的后续按季度更新，则 J2SE 1.3.1 软件包会覆盖某些 J2SE 1.4 软件包。为确保将 J2SE 1.4 安装在正确的位置，请使用 pkgadd 命令安装它。

确认安装

为确保您已正确安装，请使用以下命令：

```
# /usr/j2se/bin/java -version
java version "1.4.1_03"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.1_03-
b04)
Java HotSpot(TM) Client VM (build 1.4.1_03-b04, mixed mode)
```

此命令将报告系统上安装的版本。

Java SNMP API

安装软件包包含一个较新版本的 Java SNMP API，即 SUNWjsnmp。安装 Management Agent 软件前，请使用 pkgrm 命令删除此软件包的现有版本。

安装软件包

组成管理软件的软件包可分为以下几组：

- 域硬件监视所需的软件包
- 平台代理服务器上的平台硬件监视所需的软件包
- 在目标计算机上的平台硬件监视所需的软件包

这些软件包如表 9-1 所示。

表 9-1 SNMP Management Agent 软件包说明

软件包	软件包名称	功能
SUNWbgpc	SPA 个性化模块结构	支持个性模块的结构
SUNWbgptk	SPA 个性化模块工具包	可重复使用的组件模型和数据访问库
SUNWbgpr	SPA 个性化模块 （root 用户）	RDP 启动脚本
SUNWbgcm	SPA 硬件平台对象管理器	平台对象管理器
SUNWbgcmr	SPA 硬件平台对象管理器 （root 用户）	平台对象管理器启动脚本
SUNWbgpm	SPA SNMP 协议中介器 / 主代理程序	SNMP 协议支持和主代理程序
SUNWbgpmr	SPA SNMP 协议中介器 / 主代理程序 （root 用户）	SNMP 组件启动脚本
SUNWbgidr	SPA 域搜索 （root 用户）	域代理程序搜索启动脚本
SUNWbgod	SPA 平台搜索	平台代理程序搜索守护程序
SUNWbgodr	SPA 平台搜索 （root 用户）	平台代理程序搜索启动脚本
SUNWbgpji	SPA Sun Fire B100s 域个性化模块	B100 域检测程序
SUNWbgpjo	SPA Sun Fire B1600 平台个性化模块	B1600 平台检测程序

注意 – 这些软件包之间存在内部从属性，并且它们必须按指定顺序安装 （请参见第 68 页的 “安装 SNMP 软件”）。

升级软件

要升级软件，您必须在重新安装新版本之前删除现有软件版本 （请参见第十三章）。

软件包发行

软件包以 tar 存档包的形式提供。表 9-2 所示为 `SUNWspa.1.0.tar.Z` 存档包的内容，它包含为 Sun Fire B1600 的平台管理以及 Sun Fire B100 的域管理提供 SNMP 支持的软件包。

解压缩后，这些软件包位于下表所示的目录中。有关安装该软件的详细说明，请参考第十章。

注意 – 请确保获得此文件的最新版本。

表 9-2　SNMP Management Agent 软件包

软件包	说明	目录
SUNspa.1.0.22.tar.Z	安装到平台代理服务器上的 SNMP 平台代理软件包	platform/proxy/SUNWbgpc platform/proxy/SUNWbgpk platform/proxy/SUNWbgcm platform/proxy/SUNWbgcmr platform/proxy/SUNWbgod platform/proxy/SUNWbgodr platform/proxy/SUNWbgpjo platform/proxy/SUNWbgpm platform/proxy/SUNWbgpmr platform/proxy/SUNWjdrt platform/proxy/SUNWjsnmp
	安装到 Sun Fire B100 blade 上的 SNMP 检测程序软件包	platform/target/SUNWbgpc platform/target/SUNWbgptk platform/target/SUNWbgpr platform/target/SUNWbgcm
	安装到 Sun Fire B100 blade 上的 SNMP 域代理程序软件包	domain/SUNWbgpc domain/SUNWbgptk domain/SUNWbgcm domain/SUNWbgcmr domain/SUNWbgidr domain/SUNWbgpji domain/SUNWbgpm domain/SUNWbgpmr domain/SUNWjdrt domain/SUNWjsnmp

要解压缩 tar 文件，请键入以下命令：

```
$ zcat SUNWspa.1.0.tar.Z | tar xf -
```

在 Sun Fire B100s 上安装域或目标软件包

域软件包 （域硬件监视）或目标软件包 （平台硬件监视）必须安装到 Sun Fire B100 blade 上。可通过几种方式实现，包括以下方法：

- 在每台 Sun Fire B100 blade 上解压缩域或目标软件包
- 在一个共享的目录下解压缩域或目标软件包，该目录对于每台要安装该软件的 Sun Fire B100 blade 的 root 用户均可见
- 将域或目标软件包设置为网络安装

对系统文件的影响

安装该软件将在 /etc/init.d 目录下生成几个新的启动脚本 （如表 9-3 所示），以及到 /etc/rc<n>.d 的链接。

表 9-3　启动脚本

组件	启动脚本	软件包名称	软件包说明
平台对象管理器	spapom	SUNWbgcmr	平台对象管理器 （root 用户）
域硬件搜索模块	spaibdm*	SUNWbgidr	域硬件搜索模块 （root 用户）
远程数据插件	spardp†	SUNWbgpr	个性化模块 （root 用户）
SNMP 协议中介器 / 主代理程序	spama	SUNWbgpmr	SNMP 协议中介器 SNMP 主代理程序 （root 用户）

* Sun Fire B100s blade 上只有域软件包。

† Sun Fire B100s blade 上只有平台 / 目标软件包。

搜索模块由 inetd 自动启动，并由此在 /etc/inetd.conf 文件中创建一个新项。

平台对象管理器 (POM) 监视 IP 端口上来自其客户机的请求的活动性。这些端口在 /etc/services 文件中注册。

第十章

安装

本章说明如何在 Sun Fire B1600 上安装管理软件。

本章包括以下各节：

选择安装

当决定安装何种软件配置时，需要考虑两个主要因素：

1. 检测程序配置
2. 管理接口配置

检测程序配置

根据不同的平台类型，您可以使用以下代理程序：

- 运行在所监视系统上的域代理程序

  软件安装在所监视的 Sun Fire B100s blade （域）的本地。每台 blade 分别监视，一次只能查看一台 blade。这称作*域硬件监视*。

- 通过系统控制器代理的平台代理程序

  软件安装在远程 （平台代理）服务器上，该服务器通过 Sun Fire B1600 系统控制器访问平台检测程序，以及要监视 Sun Fire B100s 的 （目标） blade 上。这种配置允许您监视系统控制器管理的所有硬件，包括电源、系统控制器以及所有 blade。这称作 *平台硬件监视*。

  若您使用的是平台硬件监视，为获取有关硬盘驱动器、 CPU 和以太网 Mac 地址的信息，您需要在每台所监视的 Sun Fire B100s blade 上安装目标软件包。

注意 – Sun Fire B100s blade 在域硬件监视中称为 *域*，而在平台硬件监视中则称为 *目标*。

另见第 59 页的 “检测”。

管理接口配置

管理软件设计为可使用多种方式进行管理，目前支持如下方式：

- 使用 `snmpdx` 的 SNMP，不带主代理程序 （缺省安装）
- 使用主代理程序和 `snmpdx` 的 SNMP

  这种情况下，您必须手动配置安装，如第十一章和第十二章所述。

安装 SNMP 软件

本节简要介绍安装监视软件的过程。每种类型安装的详细过程将在本章后面的各节继续说明。

在安装该软件之前，请确保：

- 在域或目标 (Sun Fire B100) 以及平台代理服务器上已安装所需级别的 Solaris （请参见第 60 页的 “操作环境”）。
- 已安装所有必需的增补程序 （请参见第 61 页的 “增补程序”）以及未包含在 SNMP 软件中的所有附加基本软件包 （请参见第 61 页的 “Java 环境”）。
- 通过升级现有的 J2SE 或安装单独的 J2RE 安装了 Java 1.4，如第 61 页的 “Java 环境”所述。

确定系统满足上述所有要求后，方可继续安装 SNMP 软件。

现在，您需要决定使用域硬件监视还是平台硬件监视。

- 若选择域硬件监视，则请遵循第 69 页的 “安装用于域硬件监视的软件”中所述的步骤进行安装。
- 若选择平台硬件监视，则请遵循第 70 页的 “安装用于平台硬件监视的软件”中所述的步骤进行安装。

安装用于域硬件监视的软件

将这些软件包安装到要监视的每台 blade 上 （请参见第 65 页的 “在 Sun Fire B100s 上安装域或目标软件包”）。

▼ 安装软件

1. 请确保已安装 Java 1.4。

   请参见第 61 页的 “Java 环境”。

2. 请确保已删除所有现有的 `SUNWjsnmp` 版本。

   请参见第 62 页的 “Java SNMP API”。

3. 为避免报告从属性问题，请按照所示的顺序在 Sun Fire B100s 上安装域代理软件包。

```
# pkgadd -d . SUNWbgptk SUNWbgpc SUNWbgcm SUNWbgcmr SUNWbgidr \
SUNWbgpji SUNWjsnmp SUNWjdrt SUNWbgpm SUNWbgpmr
```

4. 配置 Java 环境。

   a. 若您已如第 61 页的 “Java 环境”所述安装了 J2SE 1.4，则可忽略这一步。

   b. 若您已如第 115 页的 “安装 J2RE 1.4”所述安装了 J2RE 1.4，则请如第 117 页的 “域硬件监视”所述编辑域硬件监视启动脚本。

5. 配置软件。

   请参见第十一章。

6. 重新引导 Sun Fire B100s blade。

   现在，您已完成域硬件监视软件的安装。请转至第 75 页的 “接口选项”继续。

7. 通过键入以下命令，确保进程已正确启动：

```
# ps -ef | grep spa.snmp
    root 15789     1  1 13:44:01 pts/2    0:00 /usr/j2se/bin/java
-Dcom.sun.spa.snmp.LOG_LEVEL=INFO -Djdmk.security.file=//etc
#
# ps -ef | grep spa.wbem

    root   278     1  0   Feb 24 ?44:19 /usr/j2se/bin/java
    -Dcom.sun.spa.wbem.pomi.port=3333 -Xms64m -Xmx768m -Dcom.sun
#
```

若输出与上述类似，则表明进程正在运行。

安装用于平台硬件监视的软件

- 决定是否要在 Sun Fire B100s blade 上安装 Java 1.4 （请参见第 61 页的 “Java 环境”中的论述）。 Java 1.4 的安装对于支持目标检测程序至关重要。
  - 要安装支持目标检测程序的软件，请按照第 70 页的 “安装支持目标检测程序的软件”中的过程开始安装。
  - 要安装不支持目标检测程序的软件，请按照第 73 页的 “安装不支持目标检测程序的软件”中的过程开始安装。
- 在平台代理服务器上安装平台代理软件包。
- 根据需要，在所监视的每台 blade 上安装目标平台代理软件包。
- 设置系统控制器 SMS IP 地址。

▼ 安装支持目标检测程序的软件

1. 请确保已在充当平台代理的服务器上安装了 Java 1.4。

   请参见第 61 页的 “Java 环境”和下面的步骤 4。

2. 请确保已从平台代理服务器删除所有现有的 SUNWjsnmp 版本。

   请参见第 62 页的 “Java SNMP API”。

3. 为避免从属性问题，请按所示顺序在平台代理服务器上安装平台代理软件包。

```
# pkgadd -d . SUNWbgptk SUNWbgpc SUNWbgcm SUNWbgcmr SUNWbgod \
SUNWbgodr SUNWbgpjo SUNWjsnmp SUNWjdrt SUNWbgpm SUNWbgpmr
```

4. 配置 Java 环境。

   a. 若您已如第 61 页的 “Java 环境”所述安装了 J2SE 1.4，则忽略这一步。

   b. 若您已如第 115 页的 “安装 J2RE 1.4”所述安装了 J2RE 1.4，则请如第 117 页的 “平台硬件监视”所述编辑平台硬件监视启动脚本。

5. 配置软件。

   请参见第十一章。

6. 键入以下命令，手动启动平台代理程序：

```
# /etc/init.d/spapom start
# /etc/init.d/init.snmpdx stop
# /etc/init.d/spama stop
# /etc/init.d/snmpdx start
# pkill -1 inetd
```

   或重新引导平台代理服务器。

7. 通过键入以下命令，确保进程已正确启动：

```
# ps -ef | grep spa.snmp
    root 15789     1  1 13:44:01 pts/2    0:00 /usr/j2se/bin/java
-Dcom.sun.spa.snmp.LOG_LEVEL=INFO -Djdmk.security.file=//etc
#
# ps -ef | grep spa.wbem

    root   278     1  0   Feb 24 ?44:19 /usr/j2se/bin/java
    -Dcom.sun.spa.wbem.pomi.port=3333 -Xms64m -Xmx768m -Dcom.sun
#
# netstat -a | grep mismi
     *.mismi            *.*              0     0 24576     0 LISTEN
       *.mismi                            *.*  0
0 24576      0 LISTEN
#
```

   若输出与上述类似，则表明进程正在运行。

8. 请确保已在所监视的目标 Sun Fire B100s blade 上安装了 Java 1.4。

   请参见第 61 页的 “Java 环境”和下面的步骤 11。

9. 请确保已从目标 blade 上删除所有现有的 `SUNWjsnmp` 版本。

   请参见第 62 页的 “Java SNMP API”。

10. 在目标 blade 上安装目标平台代理软件包。

    这些软件包使用所监视计算机的 Solaris 接口访问检测程序数据。

    请按顺序安装这些软件包，以避免从属性问题。

```
# pkgadd -d . SUNWbgptk SUNWbgpc SUNWbgcm SUNWbgpr
```

11. 配置 Java 环境。

    a. 若您已如第 61 页的 “Java 环境”所述安装了 J2SE 1.4，则忽略这一步。

    b. 若您已如第 115 页的 “安装 J2RE 1.4”所述安装了 J2RE 1.4，则请如第 117 页的 “平台硬件监视”所述编辑目标硬件监视启动脚本。

12. 键入以下命令，手动启动目标检测程序：

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

    或重新引导系统。

13. 通过键入以下命令，确保进程已正确启动：

```
# netstat -an | grep 1099
      *.1099         *.*          0       0 24576       0 LISTEN
```

    若输出与上述类似，则表明进程正在运行。

14. 使用 `setupsc` 设置 SMS IP 地址。

    从第 74 页的 “配置系统控制器”继续。

▼ 安装不支持目标检测程序的软件

1. 请确保已在充当平台代理的服务器上安装 Java 1.4。

   请参见第 61 页的 “Java 环境”。

2. 请确保已从平台代理服务器上删除所有现有的 `SUNWjsnmp` 版本。

   请参见第 62 页的 “Java SNMP API”。

3. 为避免从属性问题，请按所示顺序在平台代理服务器上安装平台代理软件包：

```
# pkgadd -d . SUNWbgptk SUNWbgpc SUNWbgcm SUNWbgcmr SUNWbgod \
SUNWbgodr SUNWbgpjo SUNWjsnmp SUNWjdrt SUNWbgpm SUNWbgpmr
```

4. 配置 Java 环境。

   a. 若您已如第 61 页的 “Java 环境”所述安装了 J2SE 1.4，可忽略这一步。

   b. 若您已如第 115 页的 “安装 J2RE 1.4”所述安装了 J2RE 1.4，则请如第 117 页的 “平台硬件监视”所述编辑平台硬件监视启动脚本。

5. 配置软件。

   请参见第十一章。

6. 键入以下命令，手动启动平台代理：

```
# /etc/init.d/spapom start
# /etc/init.d/init.snmpdx stop
# /etc/init.d/spama stop
# /etc/init.snmpdx start
# pkill -1 inetd
```

   或重新引导平台代理服务器。

7. 通过键入以下命令，确保进程已正确启动：

```
# ps -ef | grep spa.snmp
    root 15789     1  1 13:44:01 pts/2    0:00 /usr/j2se/bin/java
-Dcom.sun.spa.snmp.LOG_LEVEL=INFO -Djdmk.security.file=//etc
#
# ps -ef | grep spa.wbem

    root   278     1  0   Feb 24 ?44:19 /usr/j2se/bin/java
    -Dcom.sun.spa.wbem.pomi.port=3333 -Xms64m -Xmx768m -Dcom.sun
#
# netstat -a | grep mismi
     *.mismi            *.*              0     0 24576     0 LISTEN
      *.mismi                              *.*   0
0 24576       0 LISTEN
#
```

若输出与上述类似，则表明进程正在运行。

8. 使用 `setupsc` 设置 SMS IP 地址。

继续执行下面的操作。

配置系统控制器

安装 SNMP 软件后，您必须在系统控制器上将 SMS IP 地址设置为平台代理服务器的地址。要实现这一点，请登录到系统控制器的控制台，运行 `setupsc` 并添加平台代理服务器的 IP 地址。

下例中， IP 地址设置为 `10.5.1.1`。

对每个问题均按 [ENTER] 键以接受当前值，直到显示下面一行：

```
Enter the SMS IP address
```

输入 IP 地址，然后按 [ENTER] 键，接下来继续按 [ENTER] 键以响应其余的问题。

注意 – `setupsc` 命令在 *《Sun Fire B1600 Blade System Chassis Software Setup Guide》*中讨论。

代码示例 10-1 设置 SMS IP 地址

```
hornet-sc>setupsc
Entering Interactive setup mode.
Use Ctrl-z to exit & save. Use Ctrl-c to abort.
Do you want to configure the enabled interfaces [y]?
Should the SC network interface be enabled [y]?
Should the SC telnet interface be enabled for new connections [y]?
Do you want to configure the network interface [y]?
Should the SC use DHCP to obtain its network configuration [n]?
Enter the SC IP address [129.156.174.140]:
Enter the SC IP netmask [255.255.255.0]:
Enter the SC IP gateway [129.156.174.1]:
Do you want to configure the SC private addresses [y]?
Enter the SSC0/SC IP private address [129.156.174.118]:
Enter the SSC1/SC IP private address [129.156.174.128]:
Do you want to enable a VLAN for the SC [n]?
Enter the SMS IP address [0.0.0.0]: 10.5.1.1
<truncated>

hornet-sc>
```

接口选项

缺省安装通过 SNMP 充当 `snmpdx` 的子代理程序进行管理。安装过程中无需用户输入，不过您可在配置完毕后自定义软件的使用。

通过编辑配置文件，可为 SNMP 和 `snmpdx` 添加主代理程序功能。

注意 – 在所有情况下，SNMP 访问控制列表 (ACL) 的缺省配置均为拒绝访问。您必须对其进行配置方可启用访问 （请参见第十一章）。

使用 snmpdx 的 SNMP （缺省值）

此选项将 SNMP 中介器注册为 snmpdx 的子代理程序，它将对中介器的请求定向到一个自动分配的 UDP 端口号。这些请求可通过 snmpdx 或转发，如图 10-1 中的点线所示 （若这已在中介器的 ACL 文件中启用）（请参见第十二章）。

表 10-1 图 10-1 端口摘要

键	功能	spama.conf 中的参数	缺省值
1	snmpdx 转发请求到 SNMP 中介器的端口	SPAPM_REQ_PORT	
2	中介器将陷阱发送到 SNMP 管理器的端口	SPAPM_TRAP_PORT	162

注意 – 尽管缺省的配置为自动配置，但您仍然需要配置 snmpdx 和 / 或中介器的 ACL 文件，以支持您的管理器配置。

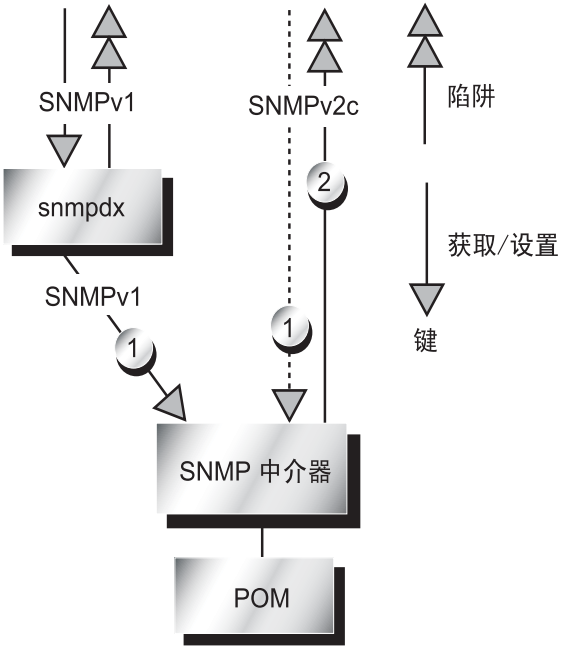

图 10-1 SNMP 是 snmpdx 的子代理程序时的数据流

SNMP 加主代理程序和 snmpdx

此选项为 SNMP 和 snmpdx 添加了主代理程序 SNMPv3 的安全性功能。 snmpdx 自动启动功能被禁用，且主代理程序注册于端口 161。自动为 snmpdx 指定一个新的端口号。

来自中介器的陷阱可选择通过主代理程序转发到 SNMPv3 或直接发送。

来自 snmpdx 的陷阱只能直接转发到 SNMP 管理器，无法通过主代理程序转发。

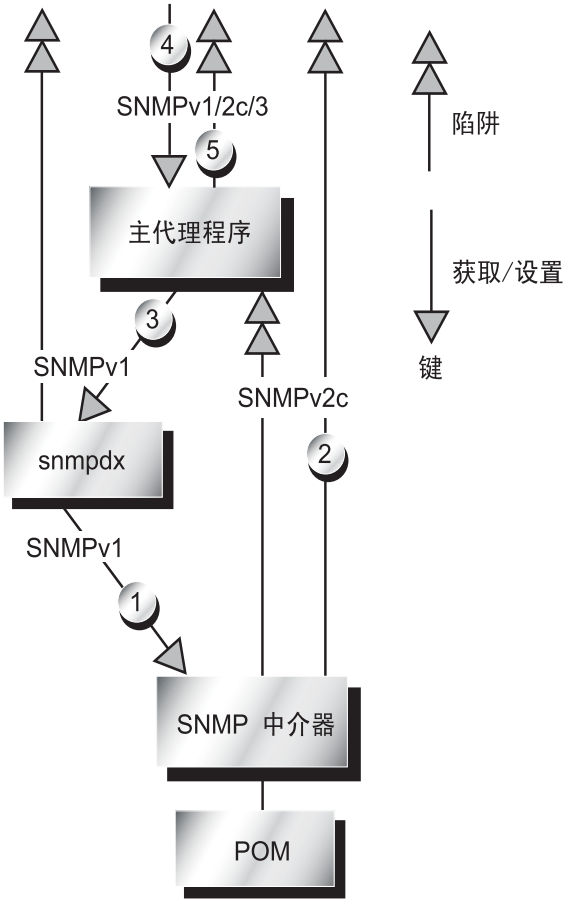

图 10-2　使用主代理程序时的数据流

表 10-2　图 10-2 的端口摘要

键	功能	spama.conf 中的参数	缺省值
1	snmpdx 转发请求到 SNMP 中介器子代理程序的端口	SPAPM_REQ_PORT	
2	中介器将陷阱定向发送到 SNMP 管理器的端口	SPAPM_TRAP_PORT	162
3	主代理程序转发请求到 snmpdx 的端口	SNMPDX_REQ_FORWARD_PORT	
4	主代理程序监视请求的端口	MASTER_AGENT_REQ_PORT	161
5	主代理程序将陷阱发送到 SNMP 管理器的端口	SPAPM_TRAP_PORT	162

第三方主代理程序加 SNMP

此选项通过使用一个手动分配的端口号或第三方主代理程序，将 SNMP 中介器注册为子代理程序。要启用直接访问，您必须手动配置中介器的 ACL 文件，如第十二章所述。

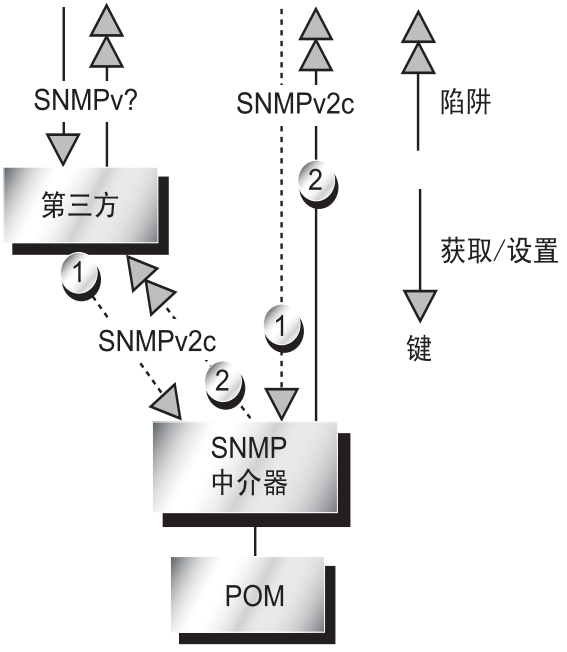

图 10-3　使用第三方主代理程序时的数据流

表 10-3　图 10-3 的端口摘要

键	功能	spama.conf 中的参数	缺省值
1	SNMP 中介器用于启用直接访问的端口，或第三方主代理程序转发请求到 SNMP 中介器的端口	SPAPM_REQ_PORT	
2	SNMP 中介器用于将 SNMPv2c 陷阱发送到第三方主代理程序或直接发送到 SNMP 管理器的端口	SPAPM_TRAP_PORT	

第十一章

配置文件

本章概述了可进行编辑以配置软件的文件。它列出了可配置的参数并介绍了访问控制的概念。

请先阅读本章再参考第十二章，因为第十二章说明的是如何使用本章所述的文件来配置基本的 SNMP 选项。

本章包括以下各节：

注意 – 有关 SNMP 软件包以及如何安装它们的信息，请参考第九章和第十章。

配置文件

以下文件位于 `/etc/opt/SUNWspa/` 目录下，它们决定 SNMP 的配置：

- **通用配置文件**
  - `spama.conf` －定义如何配置主代理程序和中介器
- **中介器配置文件**
  - `spapm.acl` －定义中介器的访问控制
  - `spapm_snmpdx.acl` －定义作为 `snmpdx` 子代理程序的中介器的访问控制

- **主代理程序配置文件**

  若您并未使用主代理程序，则无需配置这些文件。

  - `spapm.acl` －定义主代理程序的访问控制
  - `spapm.acl` －定义主代理程序的 SNMPv3 用户和环境访问控制
  - `spama.security` －定义 `spama.uacl` 中引用的 SNMPv3 用户

在后续各节中将详细描述这些文件。

通用配置文件

spama.conf

`spama.conf` 文件包含许多可配置的参数，如以下各节及表 11-1 中所述。代码示例 11-1 是 `spama.conf` 文件的一个示例。

通用选项

START_MEDIATOR

将此参数设置为 `yes` 以运行中介器，否则设置为 `no` （另见图 10-1）。

缺省值为：

```
START_MEDIATOR=yes
```

START_MASTER_AGENT

若要使用 SNMPv3 安全性，并且未使用第三方主代理程序来提供此功能，则可将此参数设置为 `yes`，以启动主代理程序 （请参见图 10-2 及其附表）。

若不需要 SNMPv3 安全性，并且使用了其它主代理程序 （包括 snmpdx），或不希望使用任何主代理程序，则请将该参数设置为 `no` （另见图 10-1 和图 10-3 及其附表）。

缺省值为：

```
START_MASTER_AGENT=no
```

AGENT_INTERFACE_NAME

若已启用主代理程序（请参见上述内容），则此处设置的值指定应绑定主代理程序的网络接口，协议中介器绑定至 localhost。

若未启用主代理程序，则此处设置的值定义中介器绑定的网络接口的主机名。若并未指定值，则缺省设置为通过缺省接口进行访问。

若将中介器作为子代理程序（例如， snmpdx 的子代理程序）使用，则请将该值设置为 localhost。

缺省值为：

AGENT_INTERFACE_NAME=localhost

主代理程序选项

主代理程序可与 snmpdx 一起工作。主代理程序启动脚本可停止 snmpdx，接管其 SNMP 端口，并在另一端口将 snmpdx 作为子代理程序重新启动。

若 SNMPDX_REQ_FORWARD_PORT 参数值为 null，则主代理程序启动脚本将在暂时的匿名范围 32768 到 65535 之间搜索空闲端口，并在该端口上重新启动 snmpdx 作为子代理程序。启动脚本同时搜索 /etc/services，并且不使用那里列出的任何端口。

不过，若您指定 SNMPDX_REQ_FORWARD_PORT 值，则主代理程序使用此端口转发请求到 snmpdx。这种情况下，主代理程序不检查该端口是否已占用。

MASTER_AGENT_REQ_PORT

主代理程序从该端口接收来自管理器的请求。对于大多数配置来说，无需更改该值（另见图 10-2 及其附表）。

若未指定，则缺省值为 161。

ENABLE_SNMPV2C_SETS

此参数控制主代理程序是否允许使用 SNMPv1 或 SNMPv2c 执行设置操作。由于 SNMPv1 和 SNMPv2C 协议本身不安全，将该值设置为 yes 将大大降低安全性。

缺省值为：

ENABLE_SNMPV2C_SETS=no

SNMPDX_REQ_FORWARD_PORT

此参数控制主代理程序转发请求到 snmpdx 的端口。若不指定值，则主代理程序将执行自动配置 （请参见本节介绍、图 10-2 及其附表）。

若指定了值，则还需手动配置 snmpdx 以在此端口监听。

缺省值为：

SNMPDX_REQ_FORWARD_PORT=

SNMPV3_USER

此参数决定发出 SNMPv3 陷阱的用户。

缺省值为：

SNMPV3_USER=defaultUser

注意 – 要发送 SNMPv3 陷阱，必须设置 SPAPM_TRAPS_ARE_V3=yes。

协议中介器选项

SUB_AGENT

此参数决定中介器或主代理是通过主代理程序 （如 snmpdx）启动还是自动启动。

缺省值为：

SUB_AGENT=yes

若指定为 yes，则必须通过传递如下 < *端口* > 参数来启动中介器：

```
# /etc/init.d/spama start <端口>
```

其中， < *端口* > 是 UDP 端口，中介器通过它监听 SNMP 请求。例如，与 snmpdx 一起使用时，中介器的调用通过 /etc/snmp/conf/spapm.rsrc 文件中以下一行控制：

```
command = "etc/init.d/spama start $PORT"
```

注意 – 若 SUB-AGENT=YES，则忽略 START_MASTER_AGENT 的值。若已启用主代理程序功能，则必须将 SUB_AGENT 设置为 no。

若将该值指定为 no，则中介器在系统启动时启动 （通过启动脚本 /etc/rc3.d/S80spama），并且监听 SNMP 请求的 UDP 端口由如下所述的 SPAPM_REQ_PORT 设置定义。

SPAPM_REQ_PORT

此参数决定 SUB_AGENT 设置为 no 时中介器接收请求的端口。请参见图 10-1、图 10-2 和图 10-3 及其附表。

缺省值为：

```
SPAPM_REQ_PORT=
```

对于缺省设置，若 START_MASTER_AGENT=yes，则自动分配一个端口号。若 START_MASTER_AGENT=no，则使用缺省端口号 33000。

这样，若 START_MASTER_AGENT=no 时，可将 SPAPM_REQ_PORT 设置为所需值，这样本地主代理程序可使用固定端口访问中介器，远程 SNMP 管理器也可以直接访问中介器。

SPAPM_TRAPS_ARE_V3

此参数决定中介器陷阱是 SNMPv3 还是 SNMPv2c。

缺省值将陷阱设置为 SNMPv2c，即：

```
SPAPM_TRAPS_ARE_V3=no
```

注意 – 若启用 SNMPv3 陷阱 （SPAPM_TRAPS_ARE_V3=yes），则还必须设置 START_MASTER_AGENT=yes 和 START_MEDIATOR=yes。

SPAPM_TRAP_PORT

此参数决定中介器陷阱发送到的端口号。请参见图 10-1、图 10-2 和图 10-3 及其附表。

缺省值为：

SPAPM_TRAP_PORT=162

SPAPM_TRAP_INTERFACE

此参数决定中介器发送 SNMP 陷阱的接口。若未定义此参数，则陷阱从主机的缺省接口发出。

注意 – SNMPv2c 陷阱直接发送，而非通过 `snmpdx`。

未定义缺省值：

SPAPM_TRAP_INTERFACE=

SPAPM_OPTIONS

此参数允许您通过指定以下一个或多个选项来改变中介器的运作：

- `-a` 发送属性更改通知
- `-a` 发送状态更改通知
- `-c` 发送创建对象通知
- `-C` 启用初始化过程中的创建对象通知
- `-c` 发送删除对象通知
- `-l` 缺省情况下启用当前问题列表日志

此参数的格式及缺省值如下：

SPAPM_OPTIONS="-ascCdl"

表 11-1 `spama.conf` 中的缺省值

参数	缺省安装值
START_MEDIATOR	START_MEDIATOR=yes
START_MASTER_AGENT	START_MASTER_AGENT=no
AGENT_INTERFACE_NAME	AGENT_INTERFACE_NAME=localhost
MASTER_AGENT_REQ_PORT	MASTER_AGENT_REQ_PORT=161 （也是未指定时的缺省值）
ENABLE_SNMPV2C_SETS	ENABLE_SNMPV2C_SETS=no
SNMPDX_REQ_FORWARD_PORT	SNMPDX_REQ_FORWARD_PORT=

表 11-1　spama.conf 中的缺省值 *（接上页）*

参数	缺省安装值
SNMPV3_USER	SNMPV3_USER=defaultUser
SUB_AGENT	SUB_AGENT=yes
SPAPM_REQ_PORT	SPAPM_REQ_PORT=
SPAPM_TRAPS_ARE_V3	SPAPM_TRAPS_ARE_V3=no
SPAPM_TRAP_PORT	SPAPM_TRAP_PORT=162
SPAPM_TRAP_INTERFACE	SPAPM_TRAP_INTERFACE=
SPAPM_OPTIONS	SPAPM_OPTIONS="-ascCdl"

代码示例 11-1　spama.conf 文件示例

```
#!/sbin/sh
#
#ident  "@(#)spama.conf1.17 01/29/03 SMI"
#
# Copyright 2003 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.

#
# This file is used to control the configuration of the Master Agent and
# Protocol Mediator
#

#
# Master Agent / Mediator configuration
#

##############################################################################
# General options
##############################################################################

#
# Set to "yes" if the mediator component should be started
#
START_MEDIATOR=yes

#
# Set to "yes" to enable the master agent
#
START_MASTER_AGENT=no

#
```

代码示例 11-1 spama.conf 文件示例 *（接上页）*

```
# Hostname of the network interface for the agent to bind to. If this
# is not specified the agent will be accessible via the default
# interface.
#
# If the mediator is being used as a sub-agent this should be
# set to localhost.
#
# If the master agent is enabled, this setting applies to its interface,
# the protocol mediator being bound to localhost.
AGENT_INTERFACE_NAME=localhost

##############################################################################
# Master Agent options
##############################################################################

#
# SNMP port for master agent to receive SNMP get/set requests.
#
# This port number will be used to listen for SNMP get/set
# requests.
#
# If this value is blank, default will be 161 if START_MASTER_AGENT=yes
#
MASTER_AGENT_REQ_PORT=

#
# set to "yes" to enable SNMPv1/SNMPv2c SET operations via the master agent.
#
ENABLE_SNMPV2C_SETS=no

#
# SNMP sub-agent port to which non-SNMP Protocol Mediator (snmpdx) requests
# will be sent.
#
# If this port setting is blank (default), automatic configuration will be
# performed. The port number for snmpdx will be dynamically determined
# if snmpdx is already using the UDP port where the Master Agent listens
# for SNMP get/set requests (by default UDP port 161) at Master Agent startup.
#
# If this port setting is blank but snmpdx is not using the same port
# as the Master Agent will listen for SNMP get/set requests, the Master
# Agent will use the port number which is being used by snmpdx to forward
# the SNMP set/get requests to snmpdx.
#
# If this port is set, it is expected that the user should perform
# the "listening" port configuration for the sub-agents and the Master
# Agent will use this port number to forward non-SNMP Protocol Mediator
```

代码示例 11-1 spama.conf 文件示例 *（接上页）*

```
# requests.
#
SNMPDX_REQ_FORWARD_PORT=

#
# SNMPv3 user
#
# The Mediator will use this user to send the V3 traps (if enabled with
# SPAPM_TRAPS_ARE_V3).
#
# If this value is blank, default will be 'defaultUser'.
SNMPV3_USER=

################################################################################
# Protocol Mediator options
################################################################################

#
#
# Sub-agent configuration
#
# If the master agent is not being used (i.e. START_MASTER_AGENT=no), then
# setting SUB_AGENT=yes indicates that the mediator should be started with a
# port number argument by snmpdx or a third party master agent. Otherwise, set
# to no if the mediator is to be started with a manually configured port
# number.
#
# If START_MASTER_AGENT=yes then this setting is ignored.
#
SUB_AGENT=yes

#
# Mediator request port.If the START_MASTER_AGENT="no" and SUB_AGENT="no", the
# default is 33000, otherwise it is dynamically allocated.
#
SPAPM_REQ_PORT=

#
# set to yes to enable v3 mediator traps (requires START_MASTER_AGENT=yes and
# START_MEDIATOR=yes)
#
SPAPM_TRAPS_ARE_V3=no

#
# Default port for traps
#
SPAPM_TRAP_PORT=162
```

代码示例 11-1 spama.conf 文件示例 *（接上页）*

```
# This is also used to define the interface to which SNMP traps will be
# sent by the protocol mediator independently of the setting of
# AGENT_INTERFACE_NAME. If not defined, traps will be issued from the
# host's default interface.
#
SPAPM_TRAP_INTERFACE=
#
# Agent option flags
#
# -a    Send attribute change notifications
# -s    Send state change notifications
# -c    Send object creation notifications
# -C    Enable object creation notifications during initialization
# -d    Send object deletion notifications
# -l    Enable current problem list logs by default
#
SPAPM_OPTIONS="-ascCdl"
```

访问控制

访问控制基于 IP 地址以及管理器主机团体和每个子代理程序指定的团体。团体和主机的访问权限在 ACL 文件中定义。

ACL 文件还定义了代理程序向其发送陷阱的主机。发送陷阱时，代理程序将其发送到 ACL 文件的 < *陷阱发送主机列表* > 中列出的所有主机。

ACL 文件共有 3 个：

- `spapm.acl` 控制对中介器的访问权限，可直接通过管理应用程序进行访问，或更常见的情况是通过 `snmpdx`。它还定义了 SNMP 陷阱所需的收件人。
- `spapm_snmpdx.acl` 控制当中介器配置为 `snmpdx` 的子代理程序时其访问权限。
- `spama.acl` 控制通过 SNMPv3 主代理程序进行的访问。

SNMPv3 的访问权限控制、团体和陷阱转发参数在 `spama.uacl` 和 `spama.security` 中定义。

ACL 文件的格式

ACL 文件包含一个定义团体和管理器访问权限的 `acl` 组，以及一个定义发送陷阱的团体和主机的 `trap` 组。 ACL 文件还包含注释行，该行第一个字符是井号 (#)。

注意 – `spapm_snmpdx.acl` 文件的语法略有不同。请参见该脚本中的注释以了解细节。

`acl` 组

`acl` 组包含一个或多个团体配置列表，它使用以下语法：

```
acl = {
    < 列表 1>
    < 列表 2>
    ...
    < 列表 N>
}
```

此文件中的 `acl` 组指定了特定团体和管理器的访问权限。它包含一个具有以下格式的团体配置列表：

```
{
    communities = < 团体列表 >
    access = < 访问权限 >
    managers = < 主机列表 >
}
```

< *团体列表* > 是此访问权限控制适用的 SNMP 团体名称列表。此列表中的团体名称以逗号分隔。

< *访问权限* > 指定授予所有运行于在管理器项中指定的计算机上的管理器的权限。它可能有两种值：

- `read-write`
- `read-only`

< *主机列表* > 项指定被授予访问权限的管理器主机。< *主机列表* > 是一个以逗号分隔的主机的列表，每台主机均可按以下方式之一表达：

- 主机名 （例如， `hubble`）
- IP 地址 （例如， `123.456.789.12`）
- 子网掩码 （例如， `123!255!255!255`）

注意 – 为了在 ACL 文件中区分 IP 地址和子网掩码，子网掩码中的每一整数间以惊叹号 (!) 分隔，而并非以点号分隔。

代码示例 11-2 `acl` 组示例

```
acl = {
     {
          communities = public, private
          access = read-only
          managers = rag, tag, bobtail
     }
     {
          communities = tigger
          access = read-write
          managers = brittas
     }
}
```

`trap` 组

`trap` 组指定代理程序将陷阱发送到的主机。仅当需要中介器发送 SNMPv2 陷阱才需要进行配置；若中介器通过 SNMP 主代理程序发送 SNMPv3 陷阱则无需配置。

此组包含一个或多个陷阱团体定义，使用以下语法：

```
trap = {
     < 团体 1>
     < 团体 2>
    ...
     < 团体 N>
}
```

每行均定义一组主机与要发送到这些主机的陷阱中的 SNMP 团体字符串之间的关联。每个 trap-community 定义均为以下格式：

```
{
    trap-community = < 陷阱团体字符串 >
    hosts = < 陷阱发送主机列表 >
}
```

< *陷阱团体字符串* > 项指定 SNMP 团体字符串。它包含在要发送到主机的陷阱中。主机在 hosts 项中指定。

< *陷阱发送主机列表* > 项指定一个以逗号分隔的主机列表。如下例所示，每台主机必须通过其名称或完整 IP 地址标识：

代码示例 11-3 trap 组示例

```
trap = {
      {
            trap-community = tigger
            hosts = gandalf, frodo
      }
}
```

中介器配置文件

本节介绍以下文件的格式：

第 93 页的 “spapm.acl 文件”

第 95 页的 “spapm_snmpdx.acl 文件”

spapm.acl 文件

spapm.acl 文件定义协议中介器的访问权限控制。该文件的一般格式如第 91 页的 “ACL 文件的格式”中所述。本节包含的信息主要与 spapm.acl 文件有关，该文件的示例如代码示例 11-4 所示。

缺省情况下，该文件位于 /etc/opt/SUNWspa。

若存在 ACL 文件，则其定义的访问权限适用于所有通过其 SNMP 适配器访问代理程序的管理器和平台代理服务器。若代理程序启动时不存在 ACL 文件，则通过 SNMP 适配器访问代理程序的所有管理器均得到完全访问权限，并且不生成陷阱。

要启用 SNMP 适配器的访问权限控制和陷阱，请确保启动代理程序时已存在 ACL 文件。由于 ACL 文件包含与安全性有关的信息，因此请为它分配限制性的访问权限，使其只对 `root` 用户可读。

`acl` 和 `trap` 组分别遵循如第 91 页的 “`acl` 组”和第 92 页的 “`trap` 组”中所述的格式。

若中介器注册为主代理程序 （如 `snmpdx`）的子代理程序，则必须在 `spapm.acl` 文件中将 `localhost` 指定为管理器，因为这是通过主代理程序转发的 SNMP 包的起点。使用 `snmpdx` 时，它不加更改地转发团体字符串。因此您还必须在 `spapm_snmpdx.acl` 文件中指定本文件所列出的团体 （请参见代码示例 11-4）。

代码示例 11-4 `spapm.acl` 文件示例

```
#
#  @(#)spapm.acl     1.6 03/01/29 SMI
#
# Copyright 2003 Sun Microsystems, Inc. All rights reserved.
#
# Template ACL file for Sun SNMP Management Agent for Sun Fire B1600

acl = {
    {
        communities = public, private
        access = read-only
        managers = rag, tag, bobtail
    }
    {
        communities = tigger
        access = read-write
         managers = brittas
     }
}

trap = {
    {
        trap-community = tigger
        hosts = brittas
    }
}
```

trap 组定义发送 SNMPv2c 通知的目的地。

spapm_snmpdx.acl 文件

缺省配置中，中介器作为 snmpdx 的子代理程序运行。您可以更改此文件以启用基于源主机名的访问权限。团体和访问权限必须与 spama.acl 文件中的相匹配。

此文件中的 acl 组指定了特定团体和管理器的访问权限。它包含一个具有以下格式的团体列表：

```
#    {
#         communities = <团体列表>
#         access = <访问权限>
#         managers = <主机列表>
#    }
```

- *团体列表* 是此访问权限控制所适用的、以逗号分隔的团体名称列表。
- *访问权限* 指定授予*主机列表* 中包括的管理器的权限。
- *主机列表* 是一个被授予指定*访问权限* 的、以逗号分隔的主机名称列表。

代码示例 11-5 的第一个示例中，系统 rag、tag 和 bobtail 配置为团体 public 和 private 上的 read-write 访问权限。对于团体 tigger，系统 brittas 配置为 read-write 访问权限。

第二个示例适用于使用 SNMPv3 主代理程序 （spama.conf 中 START_MASTER_AGENT=yes）的配置，即 snmpdx 接收到的 SNMP 包视为从本地主机发出。public 和 private 团体配置为 Read-only 访问权限。tigger 团体配置为 read-write 访问权限。这些团体通过主代理程序从 SNMPv3 环境映射。因此，需要此访问权限的任何 SNMPv3 环境均必须在此文件有一个对应的团体。

代码示例 11-5 spapm_snmpdx.acl 文件示例

```
#  @(#)spapm_snmpdx.acl1.8 03/01/29 SMI
#
# Copyright 2003 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# Template snmpdx Access Control file for Sun SNMP Management Agent for Sun
# Fire B1600
#
# Example 1:
#
# acl = {
#     {
#          communities = public, private
#          access = read-only
#          managers = rag, tag, bobtail
```

代码示例 11-5 spapm_snmpdx.acl 文件示例 *（接上页）*

```
#      }
#      {
#           communities = tigger
#           access = read-write
#           managers = rag, tag, bobtail
#      }
# }
#
# Example 2:
#
 acl = {
      {
           communities = public, private
           access = read-only
           managers = localhost
      }
      {
           communities = tigger
           access = read-write
           managers = localhost
      }
 }
#
# Trap destinations are defined in spapm.acl and spama.acl.
# This entry does not need to be edited.
trap = {
}
```

主代理程序配置文件

仅当启用主代理程序功能时 （spama.conf 中 START_MASTER_AGENT=yes），才需要对这些文件进行配置。

本节说明以下文件的格式：

- 第 97 页的 “spama.acl 文件”
- 第 97 页的 “spama.uacl 文件”
- 第 99 页的 “spama.security 文件”

spama.acl 文件

spama.acl 文件定义主代理程序的访问权限控制。本文件的一般格式如第 91 页的“ACL 文件的格式”中所述。本节包含主要与 spama.acl 文件有关的信息。

缺省情况下，该文件位于 /etc/opt/SUNWspa。

要启用 SNMP 适配器的访问权限控制和陷阱，请确保启动代理程序时已存在 ACL 文件。由于 ACL 文件包含与安全性有关的信息，因此请为它分配限制性的访问权限，使其只对 root 用户可读。

本文件定义了 SNMPv1 和 SNMPv2c 的访问权限，同时还定义了 SNMP 通知的收件人。若 spama.conf 中 SPAPM_TRAPS_ARE_V3=yes，则陷阱作为 SNMPv3 陷阱发送；否则作为 SNMPv2c 陷阱发送。

acl 组

若您使用的是 SNMPv3，则可能希望禁止 SNMPv1 和 SNMPv2c 的写入访问权限。因此，此 acl 通常只允许只读访问权限。

代码示例 11-6 acl 组示例

```
acl = {
      {
           communities = public, private
           access = read-only
           manager = localhost
      }
}
```

trap 组

此文件的 trap 组遵循如第 92 页的 “trap 组”中所述的格式。

spama.uacl 文件

当主代理程序激活时，此文件和 spama.security 文件结合使用，启用 SNMPv3 安全性。本文件的一般格式如第 91 页的 “ACL 文件的格式”中所述。本节说明其附加配置参数。本文件的示例如代码示例 11-7 所示。为清楚起见，大部分注释均已删除。

缺省情况下，该文件位于 /etc/opt/SUNWspa。

acl 组

acl 组包含以下参数：

- *环境名称*—逗号分隔的环境名称列表。
- *访问权限*—可能的取值为：
  - read-only
  - read-write
- *安全性级别*—可能的取值为：
  - noAuthNoPrivacy
  - authNoPrivacy
  - authPrivacy
- *用户*—逗号分隔的用户名称列表。

下例中，授予 defaultUser 访问权限；并且在 public 和 null 环境中，来自 defaultUser 带有最低安全性权限 authNoPrivacy 的请求均得到验证。拒绝所有其它 SNMP 请求。

此文件中没有 trap 组。

代码示例 11-7 spama.uacl 文件示例

```
#ident "@(#)spama.uacl 1.4 01/29/03 SMI"
#
# Copyright 2003 Sun Microsystems, Inc. All rights reserved.
# This software is the proprietary information of Sun Microsystems, Inc.
# Use is subject to license terms.
#
# Template ACL file
#
 acl = {
      {
          context-names = public,null
          access = read-write
          security-level=authNoPriv
          users = defaultUser
      }
 }
```

spama.security 文件

spama.security 文件指定允许访问主代理程序的用户，以及 SNMPv3 加密和验证密钥。

本文件由一组以下格式的 userEntry 行组成：

```
userEntry=< 引擎 ID>,< 用户名 >,< 安全性名称 >,< 验证算法 >,< 验证密钥 >,< 隐私权算法 >,
< 隐私权密钥 >,< 存储类型 >,< 模板 >
```

警告 – 请勿编辑本文件中除 userEntry 行以外的任何其它参数。

这些字段将在表 11-2 中进行说明。

表 11-2 spama.security 中的用户可配置参数

参数	说明
引擎 ID	所用 SNMP 引擎的 ID。它可取以下值： ■ 十六进制字符串 ■ 以 <地址>:<端口>:<IANA 编号> 格式表示引擎ID 的文本字符串 ■ 字符串 localEngineID，这适用于大多数情况
用户名	此项适用的用户名称
安全性名称	映射到此用户名的安全性名称。通常情况下两者应相同。
验证算法	所用的验证算法。可使用以下算法之一： • usmHMACMD5AuthProtocol • usmHMACSHHAuthProtocol • usmNoAuthProtocol
验证密钥	验证算法所用的密钥。可使用以下密钥之一： • 文本口令 （最少 8 个字符） • 本地化的十六进制密钥，例如，0x0098768905AB67EFAA855A453B665B12
隐私权算法	所用的隐私权算法。可使用以下算法之一： • usmDESPrivProtocol • usmNoPrivProtocol （未指定时的缺省值）

表 11-2　spama.security 中的用户可配置参数　*（接上页）*

参数	说明
隐私权密钥	隐私权算法所用的密钥。可使用以下密钥之一： • 文本口令 （最少 8 个字符） • 本地化的十六进制密钥，例如， 0x0098768905AB67EFAA855A453B665B12
存储类型	唯一可接受的值是 3，即未指定时的缺省值。
模板	缺省值是 false （无需更改此值）

本文件的示例如代码示例 11-8 所示。为清楚起见，大部分注释均已删除。

缺省情况下，该文件位于 /etc/opt/SUNWspa。

缺省的 spama.security 文件包含两个示例用户，您可以更改或取消注释以定义自己的用户。第一个示例用户名为 defaultUser，它具有以下特性：

- 只使用 MD5 算法验证
- 无隐私权
- 验证口令为 "mypassword"

第二个标准用户名为 defaultUser，它具有以下特性：

- 使用 MD5 算法验证，验证口令为 "mypassword"
- 使用 DES 算法的隐私权，隐私权口令为 "mypassword"
- 使用 DES 算法的隐私权

代码示例 11-8　spama-security 文件示例

```
#ident  "@(#)spama.security 1.7 01/29/03 SMI"
#
# Copyright 2002 Sun Microsystems, Inc. All rights reserved.
# This software is the proprietary information of Sun Microsystems, Inc.
# Use is subject to license terms.
#
# Template security file

# localEngineBoots=0

# defaultUser configuration. Authentication only.
# userEntry=localEngineID,defaultUser,,usmHMACMD5AuthProtocol,mypasswd
```

代码示例 11-8 spama-security 文件示例 *（接上页）*

```
# defaultUser configuration.Authentication and encryption.
#
userEntry=localEngineID,defaultUser,null,usmHMACMD5AuthProtocol,mypasswd,usmD
ESPrivProtocol,mypasswd,3,
```

第十二章

配置软件

本章介绍软件安装后的缺省配置，并说明如何修改第十一章中所述的文件。

本章包括以下各节：

缺省配置

软件安装时的缺省配置如下：

- 禁用主代理程序 (`START_MASTER_AGENT=no`)。
- 启用中介器 (`START_MEDIATOR_AGENT=yes`)。
- 中介器配置为 `snmpdx` 的子代理程序。

注意 – 出于安全性原因，应配置 `snmpdx` ACL 文件，以限制除监视代理程序的系统外其它系统的访问权限。

访问控制

要启用中介器的访问控制，请如第 93 页的 “spapm.acl 文件”中所述配置中介器的 ACL 文件。

若您使用的是 `snmpdx`（缺省配置），请修改 `spapm_snmpdx.acl` 文件以设置访问权限，并修改 `spapm.acl` 文件以设置陷阱收件人。

启动和停止中介器

使用标准的 snmpdx 启动脚本，启动中介器：

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

使用中介器脚本，停止中介器：

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

手动配置直接访问

由于 snmpdx 只支持 SNMPv1，因此若希望使用 SNMPv2c 专用的 get-bulk 操作且不使用主代理程序，则可以配置中介器所用的端口，以启用直接的 SNMPv2c 访问。

要手动配置中介器，请在 spama.conf 文件中做如下更改：

1. 设置 SUB_AGENT=no。
2. 将 SPAPM_REQ_PORT 设置为所需的端口号。
   发送到中介器的 SNMPv2c 请求必须发送到此端口。

中介器作为第三方主代理程序的子代理程序

要将中介器配置为第三方主代理程序 （它支持端口号的规范）的子代理程序，可通过命令行参数实现：

1. 配置中介器的 ACL 文件，允许从本地主机进行访问 （请参见第 93 页的“spapm.acl 文件”）。
2. 使用适当的端口号，配置主代理程序，以使用以下调用启动中介器：

```
/etc/init.d/spama start <端口>
```

3. 配置主代理程序，将请求转发到以下 OID 子树：
   - .iso.org.dod.internet.mgmt.mib-2.entityMIB

- .iso.org.dod.internet.private.enterprises.sun.products.sunFire.sunPlatMIB

或以下数字格式：

- .1.3.6.1.2.1.47
- .1.3.6.1.4.1.42.2.70.101

中介器和 SNMPv3 主代理程序

要启用中介器和主代理程序，则至少必须做以下更改：

1. 在 spama.conf 文件中：

   a. 设置 `START_MASTER_AGENT=yes`。

   b. 设置 `SUB_AGENT=no`。

2. 配置中介器的 ACL 文件，允许从本地主机进行访问，如第 93 页的 “spapm.acl 文件”中所述。

3. 配置 snmpdx，允许从本地主机进行访问，如第 95 页的 “spapm_snmpdx.acl 文件”中所述。

4. 配置主代理程序的 ACL 文件，允许从所需的管理器进行访问，如第 97 页的 “spama.acl 文件”中所述。

5. 配置安全性文件，以定义 SNMPv3 用户、环境以及验证和加密级别，如第 97 页的 “spama.acl 文件”和第 99 页的 “spama.security 文件”中所述。

启动和停止代理程序

使用中介器脚本，启动中介器和主代理程序：

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

使用中介器脚本，停止中介器和主代理程序：

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

转发 SNMPv3 陷阱

要配置主代理程序以转发来自中介器的 SNMPv3 陷阱，请在 spama.conf 文件中设置如下 （请参见第 82 页的 “spama.conf”）：

1. **设置** `SPAPM_TRAPS_ARE_V3=yes`。
2. （**可选**）**设置** `SNMPV3_USER`。

注意 – 必须在 `spama.uacl` 和 `spama.security` 文件中将陷阱用户配置为 SNMPV3 用户。

第十三章

卸载软件

本章说明如何卸载该软件。

通常情况下，只需使用 pkgrm 命令删除所安装的软件包即可卸载 SNMP。此过程可删除所有相关的文件和链接，并重新启用 snmpdx。

SNMP 软件自动更改的配置会恢复到原来的状态。但是，如果您更改了任何外部文件设置，例如 snmpdx ACL 文件，则必须在卸载 SNMP 软件后手动恢复这些设置。

注意 – 下面列出的过程并不卸载 Java SNMP API 软件包 SUNWjsnmp。若要重新安装本软件包的 Solaris 版本，则必须首先删除 Java SNMP API.。

平台代理程序和目标代理程序软件包

要从平台代理服务器删除平台代理程序软件包，请键入以下命令：

```
# pkgrm SUNWbgpmr SUNWbgpm SUNWjdrt SUNWjsnmp SUNWbgpjo \
SUNWbgodr SUNWbgod SUNWbgcmr SUNWbgcm SUNWbgpc SUNWbgptk
```

警告 – 删除 SUNWjdrt 和 SUNWjsnmp 软件包时请谨慎，因为两者均为系统软件包，其它产品可能也在使用它们。

要从 Sun Fire B100 blade 删除目标平台软件包，请键入以下命令：

```
# pkgrm SUNWbgpr SUNWbgcm SUNWbgpc SUNWbgptk
```

域代理程序软件包

要从 Sun Fire B100 blade 删除域代理程序软件包，请键入以下命令：

```
# pkgrm SUNWbgpmr SUNWbgpm SUNWjdrt SUNWjsnmp SUNWbgpji \
SUNWbgidr SUNWbgcmr SUNWngcm SUNWbgpc SUNWbgptk
```

警告 – 删除 SUNWjdrt 和 SUNWjsnmp 软件包时请谨慎，因为两者均为系统软件包，其它产品可能也在使用它们。

第十四章

错误诊断

本章提供的信息可帮助您对系统进行错误诊断。

问题

- 使用缺省配置 (snmpdx) 时， SNMP 代理程序不响应。

1. 键入以下命令，确保中介器正在运行：

```
# ps -ef | grep spa.snmp
    root 15789     1  1 13:44:01 pts/2    0:00 /usr/j2se/bin/java
-Dcom.sun.spa.snmp.LOG_LEVEL=INFO -Djdmk.security.file=//etc
```

如果响应与上述内容类似，则表明中介器进程正在运行。

键入以下命令，停止并重新启动中介器：

```
# /etc/init.d/spama stop
# /etc/init.d/init.snmpdx start
```

2. 键入以下命令，确保已安装正确的 Java 版本：

```
# /usr/j2se/bin/java -version
java version "1.4.1_03"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.1_03-b04)
Java HotSpot(TM) Client VM (build 1.4.1_03-b04, mixed mode)
```

上述命令报告的版本应为 1.4 或其后续版本。若并非正确版本，则请如第 61 页的“Java 环境”中所述安装 Java 1.4 JDK。

3. 键入以下命令，确保已安装正确的 SUNWjsnmp 版本：

```
# pkginfo -l SUNWjsnmp | grep VERSION
   VERSION:  5.0
```

若所示版本为 1.0，则请删除 SUNWjsnmp 软件包，并重新安装 SUNWspa.*.tar.Z 存档中的版本 （请参见第 62 页的 “Java SNMP API”）。

4. 请确保 spama.conf 文件包含以下各项：

```
START_MASTER_AGENT=no
START_MEDIATOR=yes
SUB_AGENT=yes
```

5. 通过键入以下命令，确保中介器在 snmpdx 中已正确注册：

```
# cat /var/snmp/snmpdx.st
spapm spapm 2516 34050
snmpd snmpd 2567 34053
```

上述 spapm 项显示中介器已注册为 snmpdx 的子代理程序。

6. 请确保 /etc/snmp/conf/spapm.reg 和 /etc/snmp/conf/spapm.rsrc 未损坏。

键入以下命，停止并重新启动中介器：

```
# /etc/init.d/spama stop
# /etc/init.d/init.snmpdx start
```

7. 请确保已正确设置 ACL 文件中的权限。
   - spapm_snmpdx.acl 定义了所用 SNMP 管理器的访问权限。
   - spapm.acl 定义了 localhost 的访问权限。

   有关更多信息，请参见第十一章。

问题

- 使用平台代理程序时，并无用于硬盘驱动器 (HDD) 或以太网 MAC 地址的检测程序。

  只有当操作环境运行于目标 Sun Fire B100s blade 上时，才可获得此信息。

1. 请确保 Sun Fire B100s blade 已引导。
2. 通过键入以下命令，确保目标检测程序在 Sun Fire B100s blade 上运行：

```
# netstat -an | grep 1099
      *.1099         *.*          0       0 24576       0 LISTEN
```

若并未监听端口，则目标检测程序尚未运行。

通过键入以下命令，在 Sun Fire B100s blade 上启动检测程序：

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

3. 键入以下命令，确保已安装正确的 Java 版本：

```
# /usr/j2se/bin/java -version
java version "1.4.1_03"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.1_03-b04)
Java HotSpot(TM) Client VM (build 1.4.1_03-b04, mixed mode)
```

上述命令报告的版本应为 1.4 或其更高版本。不正确的版本也可以启动检测程序，但会在短时间内出错。

若并非正确版本，请如第 61 页的 “Java 环境”中所述安装 Java 1.4 JDK。

问题

- 可以访问代理程序，但没有受监视平台的检测程序。

1. 通过键入以下命令，确保搜索守护程序正在运行：

```
# netstat -a | grep mismi
      *.mismi               *.*                  0       0 24576       0 LISTEN
      *.mismi                                 *.*  0
0 24576       0 LISTEN
```

上述输出表明搜索守护程序正在监听来自受控平台的请求。

a. 请确保 /etc/services 包含以下各项：

```
mismi            8265/tcp                  # MISMI Discovery
```

b. 请确保 /etc/inetd.conf 包含以下各项：

```
# MISMIDISCOVERY - mismiDiscovery daemon
mismi   stream  tcp6    nowait  root    /opt/SUNWspa/bin/mismiDiscovery
mismiDiscovery
```

c. 键入以下命令，确保 /etc/inetd.conf 是到 /etc/inet/inetd.conf 的符号链接：

```
# ls -l /etc/inetd.conf
lrwxrwxrwx   1 root     root           17 Jan  7 17:08 /etc/inetd.conf ->
./inet/inetd.conf
```

若该链接不存在，则安装 SUNWbgodr 软件包时更新文件失败。

通过键入以下命令，更正 inetd 配置并重新启动：

```
# pkill -1 inetd
```

2. 通过键入以下命令，确保已搜索平台：

```
# netstat -a | grep mismi
      *.mismi               *.*                0       0 24576       0 LISTEN
blade-174-119.36780  hornet-sc.mismi       8192      0 24820      0 ESTABLISHED
      *.mismi                           *.*   0
0 24576       0 LISTEN
```

上述输出显示搜索守护程序正在监听，并已建立一个到平台系统控制器 <hornet-sc> 的连接。

如果没有显示连接信息，请如第 74 页的 “配置系统控制器”中所述检查系统控制器设置。

问题

- SNMPv3 的 `get` 和 `set` 请求超时。
  - 可能原因

可能已编辑或删除 `spama.security` 文件中的 `localEngineId` 或 `localEngineBoots` 的数目。

  - 检查

无法简单地确定是否已编辑该文件。

  - 修复

如下所示重新启动代理程序，然后重新启动管理应用程序使两者同步。

```
# /etc/init.d/spama stop
# /etc/init.d/spama start
```

问题

- SNMP 的 `get` 和 `set` 请求超时。
  - 可能原因

在负荷很重的系统中， `snmpdx` 主代理程序发送到 SNMP 中介器的请求有可能超时。此超时时间当前设置为 2 秒 (2000000 µs)。

  - 检查

无法简单地确定管理应用程序报告的超时是发生在 `snmpdx` 和 SNMP 中介器之间，还是发生在管理应用程序和 `snmpdx` 之间。

  - 修复

可编辑 `/etc/snmp/conf/spapm.reg` 文件中的超时属性来延长超时时间。编辑该文件后，键入以下命令重新启动中介器：

```
# /etc/init.d/spama stop
# /etc/init.d/init.snmpdx start
```

附录 A

安装与 J2SE 1.3.1 共存的 J2RE 1.4

本附录介绍如何在装有 J2SE 1.3.1 的平台代理服务器和 B100s 域上安装 Java 2 运行时环境 (J2RE) 标准版 1.4，使两者共存，以及如何修改启动脚本以定位安装。

本附录包括以下各节：

安装 J2RE 1.4

如第 61 页的 “Java 环境”中所述，请遵循以下步骤安装与 J2SE 1.3.1 共存的 J2RE 1.4：

可从以下网址获得 J2RE 1.4 的自解压二进制文件：

```
http://java.sun.com/j2se/1.4/download.html
```

请遵循下面的指导安装 J2RE。上述网址还提供了有关下载该文件的详细信息。

注意 – 本产品只需要 32 位支持，因此无须安装 J2RE 的 64 位增补程序。

在下面的步骤中，请将 < 版本号 > 替换为相应的 J2RE 更新版本号。

例如，若下载 `1.4.0_01` 更新，则以下命令：

```
# chmod +x j2re-1_4_<版本号>-solaris-sparc.sh
```

应为：

```
chmod +x j2re-1_4_0_01-solaris-sparc.sh
```

1. 下载并检查文件大小

   所需文件为：

   j2re-1_4_< 版本号 >-solaris-sparc.sh

   下载文件之前，请注意文件的大小 （在下载页面上提供）。一旦下载完毕，请检查下载的软件文件是否完整无损。

   请确保将文件下载到 root 用户可访问的位置 （例如，位于 /tmp 目录下）。

2. 运行 su 命令，成为 root 用户，然后输入超级用户口令。

3. 请确保为该自解压二进制文件设置了执行权限。

```
# chmod +x j2re-1_4_< 版本号 >-solaris-sparc.sh
```

4. 请转至要安装该文件的目录。

```
# cd /usr
```

5. 运行该自解压二进制文件。

   将创建一个名为 /usr/j2re1.4.< 版本号 > 的目录，该目录包含 J2RE。

6. 请确保已正确安装 J2RE。

```
# /usr/j2re1.4.1_01/bin/java -version
java version "1.4.1_01"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.1_01-
b01)
Java HotSpot(TM) Client VM (build 1.4.1_01-b01, mixed mode)
```

   上述命令报告的版本应为 1.4 或其后续版本。上述示例所示的版本为 1.4.1_01。

7. 删除该自解压二进制文件。

8. 退出 root 用户 shell。

编辑启动脚本

本节解释安装 J2RE 1.4 后如何修改启动脚本。

请将本节与第十章结合起来阅读。

域硬件监视

这些步骤与第 69 页的 “安装用于域硬件监视的软件”中的步骤 4 有关。

1. **在监视的每台 B100s blade 上，编辑以下启动脚本：**
   - `/etc/init.d/spaibdm`
   - `/etc/init.d/spapom`

   将这一行

   `JAVA=/usr/j2se/bin/java`

   替换为

   `JAVA=/usr/j2re1.4.<版本号>/bin/java`
2. **在监视的每台 B100s blade 上，编辑以下启动脚本：**
   - `/etc/init.d/spama`

   将这一行

   `JAVA_JAVA=/usr/j2se/bin/java`

   替换为

   `JAVA_JAVA=/usr/j2re1.4.<版本号>/bin.java`

平台硬件监视

步骤 1 和步骤 2 与第 70 页的 “安装支持目标检测程序的软件”中的步骤 4 以及第 73 页的 “安装不支持目标检测程序的软件”中的步骤 4 有关。

步骤 3 只与第 73 页的 “安装不支持目标检测程序的软件”中的步骤 11 有关。

1. **在平台代理服务器上，编辑以下启动脚本：**

- /etc/init.d/spapom

将这一行

JAVA=/usr/j2se/bin/java

替换为

JAVA=/usr/j2re1.4.< 版本号 >/bin/java

2. 在平台代理服务器上，编辑以下启动脚本：
   /etc/init.d/spama

   将这一行

   JAVA_JAVA=/usr/j2se/bin/java

   替换为

   JAVA_JAVA=/usr/j2re1.4.< 版本号 >/bin.java

3. 在监视的每台 B100s blade （目标）上，编辑以下启动脚本：
   - /etc/init.d/spardp

   将这一行

   JAVA=/usr/j2se/bin/java

   替换为

   JAVA=/usr/j2re1.4.< 版本号 >/bin/java

索引

K

L

M

N

O

P

Q

R

S

T

W

X

Y

Z