


JMX

Allen Long
Email: allen@huihoo.com
http://www.huihoo.com
2004-04

JMX概述

JMX--Java Management Extensions，即Java管理扩展,是一个为应用程序、设备、系统等植入管理功能的框架。JMX可以跨越一系列异构操作系统平台、系统体系结构和网络传输协议，灵活的开发无缝集成的系统、网络和服务管理应用。

JMX体系结构分为以下四个层次：

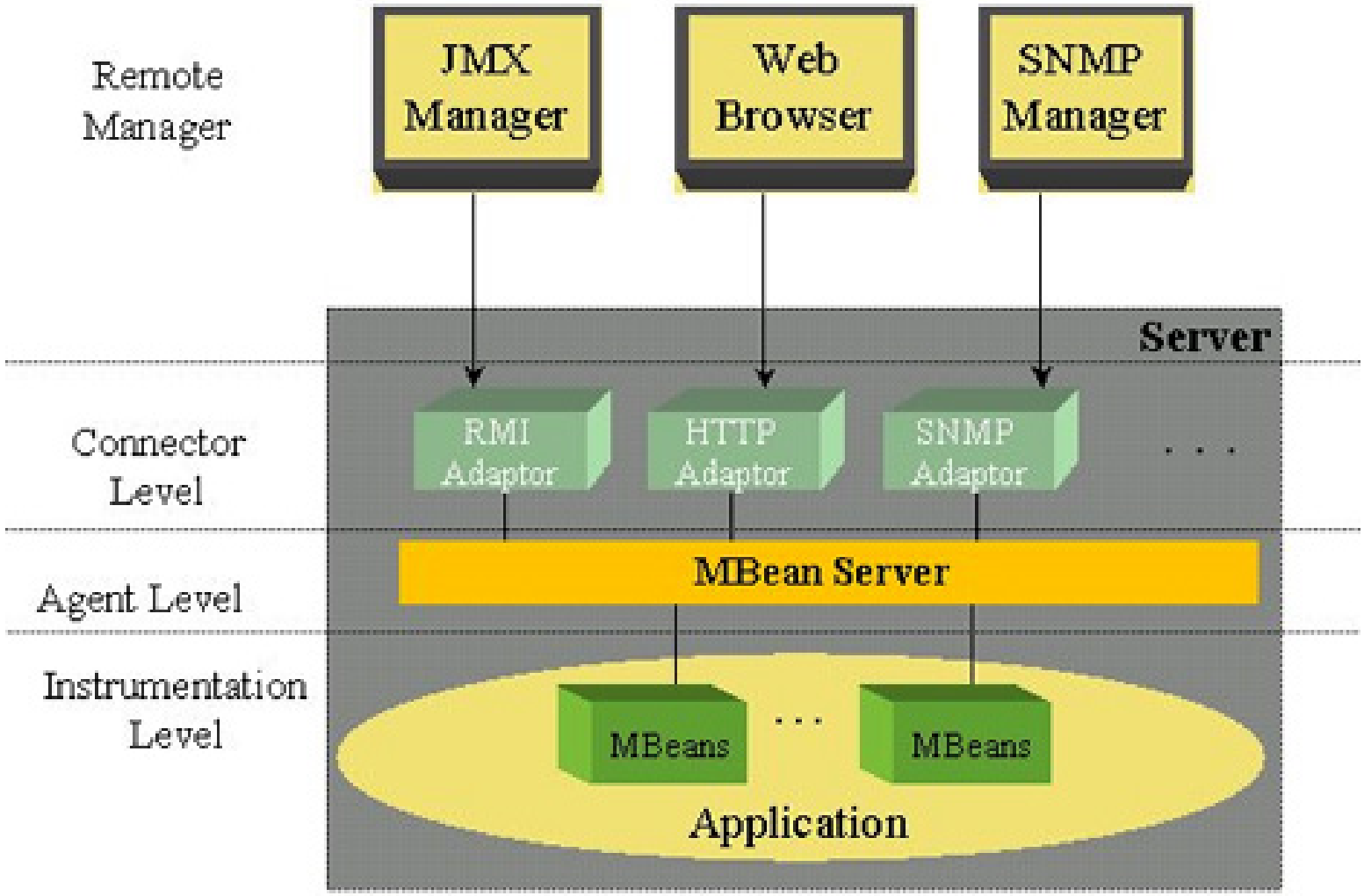
1)设备层（Instrumentation Level）：主要定义了信息模型。在JMX中，各种管理对象以管理构件的形式存在，需要管理时，向MBean服务器进行注册。该层还定义了通知机制以及一些辅助元数据类。

2)代理层（Agent Level）：主要定义了各种服务以及通信模型。该层的核心是一个MBean服务器，所有的管理构件都需要向它注册，才能被管理。注册在MBean服务器上管理构件并不直接和远程应用程序进行通信，它们通过协议适配器和连接器进行通信。而协议适配器和连接器也以管理构件的形式向MBean服务器注册才能提供相应的服务。

3)分布服务层（Distributed Service Level）：主要定义了能对代理层进行操作的管理接口和构件，这样管理者就可以操作代理。然而，当前的JMX规范并没有给出这一层的具体规范。

4)附加管理协议API：定义的API主要用来支持当前已经存在的网络管理协议，如SNMP、TMN、CIM/WBEM等。

JMX架构图


Remote Manager
JMX Manager
Web Browser
SNMP Manager
Server
Connector Level
RMI Adaptor
HTTP Adaptor
SNMP Adaptor
. . .
Agent Level
MBean Server
Instrumentation Level
MBeans
. . .
MBeans
Application

设备层(Instrumentation Level)

该层定义了如何实现JMX管理资源的规范。一个JMX管理资源可以是一个Java应用、一个服务或一个设备，它们可以用Java开发，或者至少能用Java进行包装，并且能被置入JMX框架中，从而成为JMX的一个管理构件(Managed Bean)，简称MBean。管理构件可以是标准的，也可以是动态的，标准的管理构件遵从JavaBeans构件的设计模式；动态的管理构件遵从特定的接口，提供了更大的灵活性。

该层还定义了通知机制以及实现管理构件的辅助元数据类。

管理构件(MBean)

在JMX规范中，管理构件定义如下：它是一个能代表管理资源的Java对象，遵从一定的设计模式，还需实现该规范定义的特定的接口。该定义了保证了所有的管理构件以一种标准的方式来表示被管理资源。

管理接口就是被管理资源暴露出的一些信息，通过对这些信息的修改就能控制被管理资源。一个管理构件的管理接口包括：
1)能被接触的属性值；
2)能够执行的操作；
3)能发出的通知事件；
4)管理构件的构建器。

管理构件通过公共的方法以及遵从特定的设计模式封装了属性和操作，以便暴露给管理应用程序。例如，一个只读属性在管理构件中只有Get方法，既有Get又有Set方法表示是一个可读写的属性。

其余的JMX的构件，例如JMX代理提供的各种服务，也是作为一个管理构件注册到代理中才能提供相应的服务。

JMX对管理构件的存储位置没有任何限制，管理构件可以存储在运行JMX代理的Java虚拟机的类路径的任何位置，也可以从网络上的任何位置导入。

JMX定义了四种管理构件：标准、动态、开放和模型管理构件。每一种管理构件可以根据不同的环境需要进行制定。

标准管理构件(Standard MBean)




标准管理构件的设计和实现是最简单的，它们的管理接口通过方法名来描述。标准管理构件的实现依靠一组命名规则，称之为设计模式。这些命名规则定义了属性和操作。检查标准管理构件接口和应用设计模式的过程被称为内省（Introspection）[22]。JMX代理通过内省来查看每一个注册在MBean 服务器上的管理构件的方法和超类，看它是否遵从一定设计模式，决定它是否代表了一个管理构件，并辨认出它的属性和操作。

动态管理构件(Dynamic MBean)

动态管理构件提供了更大的灵活性，它可以在运行期暴露自己的管理接口。它的实现是通过实现一个特定的接口DynamicMBean

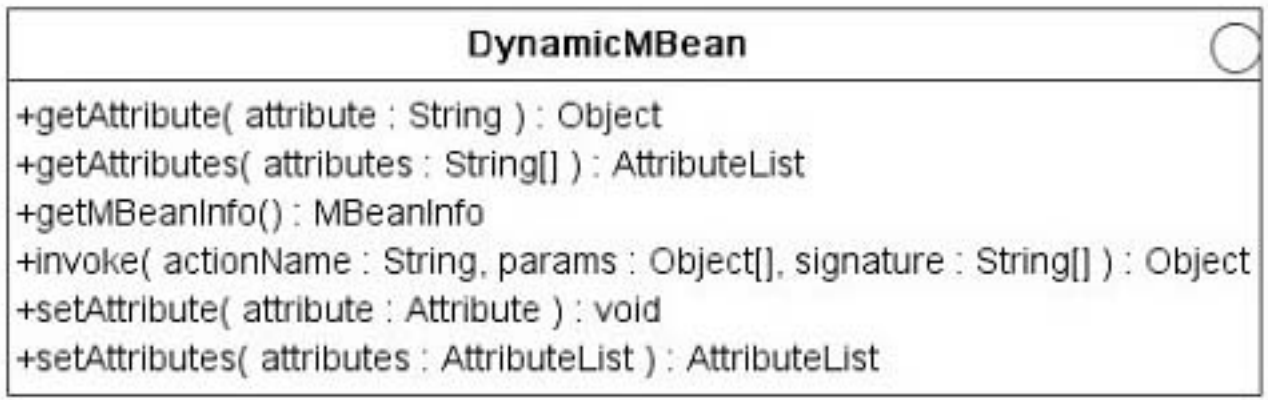


图 1－1 动态 MBean 的接口

JMX代理通过getMBeanInfo方法来获取该动态管理构件暴露的管理接口，该方法返回的对象是MbeanInfo类的实例，包含了属性和操作的签名。由于该方法的调用是发生在动态管理构件向MBean服务器注册以后，因此管理接口是在运行期获取的。不同于标准管理构件，JMX代理不需要通过内省机制来确定动态管理构件的管理接口。由于DynamicMBean的接口是不变的，因此可以屏蔽实现细节。由于这种在运行期获取管理接口的特性，动态管理构件提供了更大的灵活性。

开放管理构件(Open MBean)

开放管理构件是一种专门化的动态管理构件，其中所有的与该管理构件相关的参数、返回类型和属性都围绕一组预定义的数据类型（String、Integer、Float 等）来建立，并且通过一组特定的接口来进行自我描述。JMX代理通过获得一个OpenMBeanInfo对象来获取开放管理构件的管理接口，OpenMBeanInfo是MbeanInfo的子类。

模型管理构件(Model MBean)

模型管理构件也是一种专门化的动态管理构件。它是预制的、通用的和动态的 MBean 类，已经包含了所有必要缺省行为的实现，并允许在运行时添加或覆盖需要定制的那些实现。JMX规范规定该类必须实现为javax.management.modelmbean.RequiredModelMBean，管理者要做的就是实例化该类，并配置该构件的默认行为并注册到JMX代理中，即可实现对资源的管理。JMX代理通过获得一个ModelMBeanInfo对象来获取管理接口。

模型管理构件具有以下新的特点：

1)持久性

定义了持久机制，可以利用Java的序列化或JDBC来存储模型MBean的状态。

2)通知和日志功能

能记录每一个发出的通知，并能自动发出属性变化通知。

3)属性值缓存

具有缓存属性值的能力。

通知模型




一个管理构件提供的管理接口允许代理对其管理资源进行控制和配置。然而，对管理复杂的分布式系统来说，这些接口只是提供了一部分功能。通常，管理应用程序需要对状态变化或者当特别情况发生变化时作出反映。

为此，JMX定义了通知模型。通知模型仅仅涉及了在同一个JMX代理中的管理构件之间的事件传播。JMX通知模型依靠以下几个部分：

1)Notification，一个通用的事件类型，该类标识事件的类型，可以被直接使用，也可以根据传递的事件的需要而被扩展。

2)NotificationListener接口，接受通知的对象需实现此接口。

3)NotificationFilter接口，作为通知过滤器的对象需实现此接口，为通知监听者提供了一个过滤通知的过滤器。

4)NotificationBroadcaster接口，通知发送者需实现此接口，该接口允许希望得到通知的监听者注册。

发送一个通用类型的通知，任何一个监听者都会得到该通知。因此，监听者需提供过滤器来选择所需要接受的通知。

任何类型的管理构件，标准的或动态的，都可以作为一个通知发送者，也可以作为一个通知监听者，或两者都是。

辅助元数据类




辅助元数据类用来描述管理构件。辅助元数据类不仅被用来内省标准管理构件，也被动态管理构件用来进行自我描述。这些类根据属性、操作、构建器和通告描述了管理接口。JMX代理通过这些元数据类管理所有管理构件，而不管这些管理构件的类型。

部分辅助元类如下：

1)MBeanInfo--包含了属性、操作、构建器和通知的信息。
2)MBeanFeatureInfo--为下面类的超类。
3)MBeanAttributeInfo--用来描述管理构件中的属性。
4)MBeanConstructorInfo--用来描述管理构件中的构建器。
5)MBeanOperationInfo--用来描述管理构件中的操作。
6)MBeanParameterInfo--用来描述管理构件操作或构建器的参数。
7)MBeanNotificationInfo--用来描述管理构件发出的通知。

代理层

代理层是一个运行在Java虚拟机上的管理实体，它活跃在管理资源和管理者之间，用来直接管理资源，并使这些资源可以被远程的管理程序所控制。代理层由一个MBean服务器和一系列处理被管理资源的服务所组成。下图表示了代理层的组成：

MBean服务器

Mbean服务器为代理层的核心，设备层的所有管理构件都在其注册，管理者只用通过它才能访问管理构件。

管理构件可以通过以下三种方法实例化和注册：
1)通过另一个管理构件
2)管理代理本身
3)远程应用程序

注册一个管理构件时，必须提供一个唯一的对象名。管理应用程序用这个对象名进行标识管理构件并对其操作。这些操作包括：
1)发现管理构件的管理接口
2)读写属性值
3)执行管理构件中定义的操作
4)获得管理构件发出的通告
5)基于对象名和属性值来查询管理构件

协议适配器和连接器

MBean服务器依赖于协议适配器和连接器来和运行该代理的Java虚拟机之外的管理应用程序进行通信。协议适配器通过特定的协议提供了一张注册在MBean服务器的管理构件的视图。例如，一个HTML适配器可以将所有注册过的管理构件显示在Web 页面上。不同的协议，提供不同的视图。

连接器还必须提供管理应用一方的接口以使代理和管理应用程序进行通信，即针对不同的协议，连接器必须提供同样的远程接口来封装通信过程。当远程应用程序使用这个接口时，就可以通过网络透明的和代理进行交互，而忽略协议本身。

适配器和连接器使MBean服务器与管理应用程序能进行通信。因此，一个代理要被管理，它必须提供至少一个协议适配器或者连接器。面临多种管理应用时，代理可以包含各种不同的协议适配器和连接器。

当前已经实现和将要实现的协议适配器和连接器包括：

1)RMI连接器
2)SNMP协议适配器
3)IIOP协议适配器
4)HTML协议适配器
5)HTTP连接器

代理服务

代理服务可以对注册的管理构件执行管理功能。通过引入智能管理，JMX可以帮助我们建立强有力的管理解决方案。代理服务本身也是作为管理构件而存在，也可以被MBean服务器控制。

JMX规范定义了代理服务有：

1)动态类装载--通过管理小程序服务可以获得并实例化新的类，还可以使位于网络上的类库本地化。

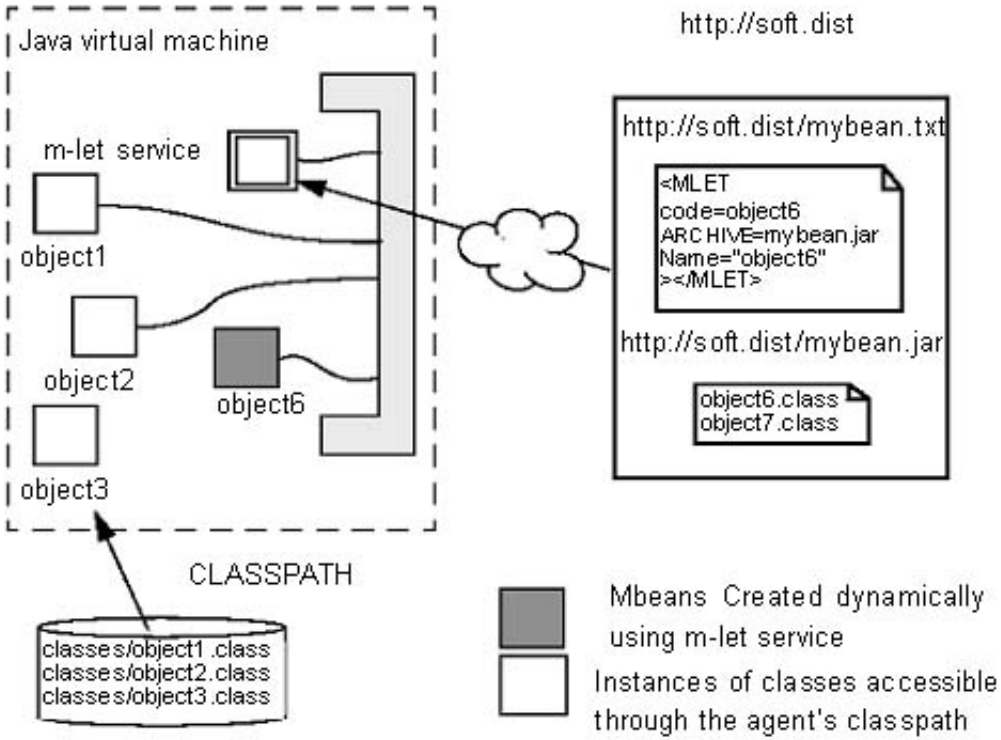
2)监视服务--监视管理构件的属性值变化，并将这些变化通知给所有的监听者。

3)时间服务--定时发送一个消息或作为一个调度器使用。

4)关系服务--定义并维持管理构件之间的相互关系。

动态类装载

动态类装载是通过m-let（management applet）服务来实现的，它可以从网络上的任何URL处下载并实例化管理构件，然后向MBean服务器注册。在一个M-let服务过程中，首先是下载一个m-let文本文件，该文件是XML格式的文件，文件的内容标识了管理构件的所有信息，比如构件名称、在MBean服务器中唯一标识该构件的对象名等。然后根据这个文件的内容，m-let服务完成剩余的任务。



监视服务

通过使用监视服务，管理构件的属性值就会被定期监视，从而保证始终处于一个特定的范围。当监视的属性值的变化超出了预期定义的范围，一个特定的通告就会发出。JMX规范当前规定了三种监视器：

1)计数器监视器，监视计数器类型的属性值，通常为整型，且只能按一定规律递增。
2)度量监视器，监视度量类型的属性值，通常为实数，值能增能减。
3)字符串监视器，监视字符串类型的属性值。

每一个监视器都是作为一个标准管理构件存在的，需要提供服务时，可以由相应的管理构件或远程管理应用程序动态创建并配置注册使用。

监视服务

计数器监视器的使用情况：

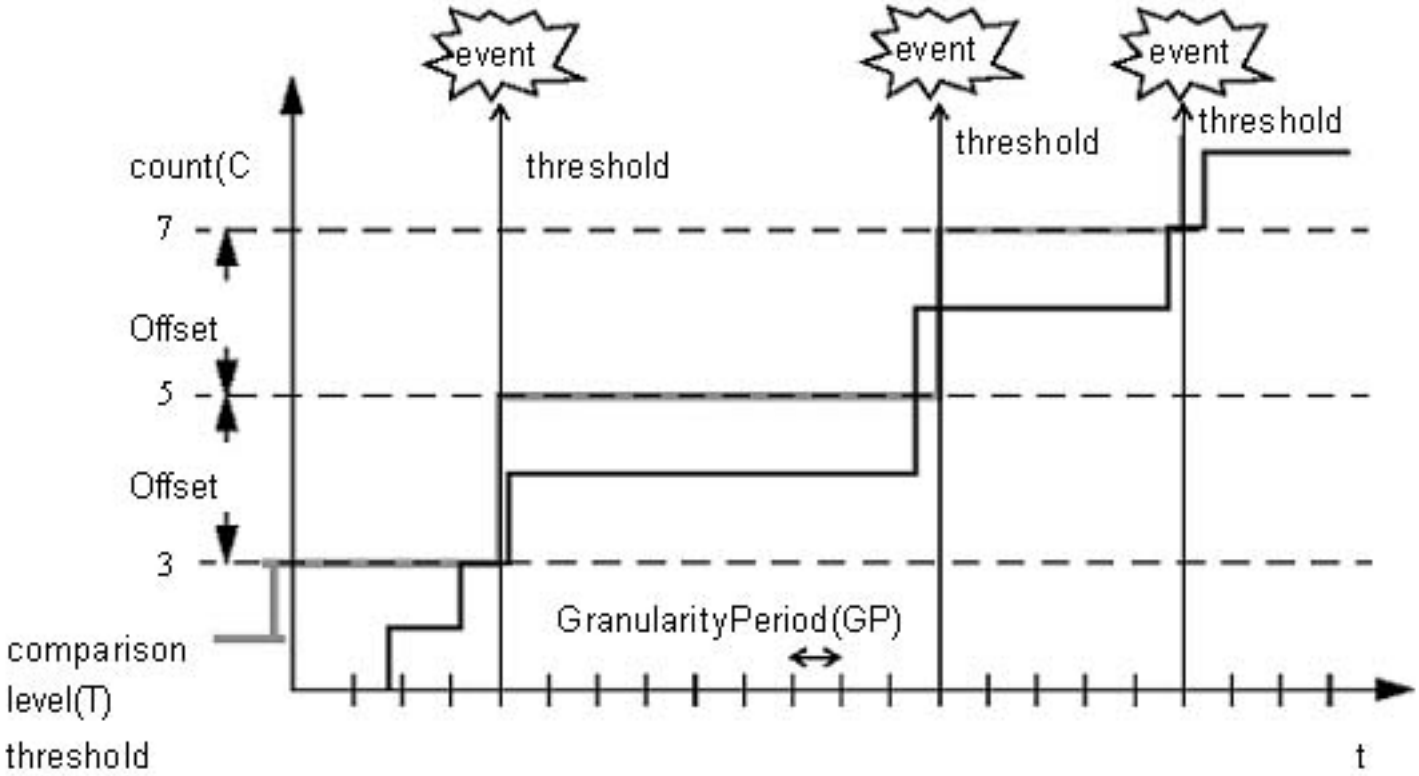


图 1－4 计数监视的操作

时间服务

时间服务可以在制定的时间和日期发出通告，也可以定期的周期性的发出通告，依赖于管理应用程序的配置。时间服务也是一个管理构件，它能帮助管理应用程序建立一个可配置的备忘录，从而实现智能管理服务。

关系服务

JMX规范定义了管理构件之间的关系模型。一个关系是用户定义的管理构件之间的N维联系。

关系模型定义如下一些术语：

1)角色：就是一个关系中的一类成员身份，它含有一个角色值。
2)角色信息：描述一个关系中的一个角色。
3)关系类型：由角色信息组成，作为创建和维持关系的模板。
4)关系：管理构件之间的当前联系，且必须满足一个关系类型的要求。
5)角色值：在一个关系中当前能满足给定角色的管理构件的列表。
6)关系服务：是一个管理构件，能接触和维持所有关系类型和关系实例之间的一致性。

在关系服务中，管理构件之间的关系由通过关系类型确定的关系实例来维护。仅仅只有注册到MBean服务器上并且能被对象名标识的管理构件才能成为一个关系的成员。关系服务从来就不直接操作它的成员--管理构件，为了方便查找它仅仅提供了对象名。

关系服务能锁定不合理关系类型的创建，同样，不合理的关系的创建也会被锁定。角色值的修正也要遵守一致性检查。

由于关系是定义在注册的管理构件之间的联系，所以当其中的管理构件卸载时，就会更改关系。关系服务会自动更改角色值。所有对关系实例的操作比如创建、更新、删除等都会使关系服务发出通告，通告会提供有关这次操作的信息。

JMX关系模型只能保证所有的管理构件满足它的设计角色，也就是说，不允许一个管理构件同时出现在许多关系中。

分布服务层

当前，SUN并没有给出这一层的具体规范，下面给出的只是一个简要描述。

该层规定了实现JMX应用管理平台的接口。这一层定义了能对代理层进行操作的管理接口和组件。这些组件能：
1)为管理应用程序提供一个接口，以便它通过一个连接器能透明和代理层或者JMX管理资源进行交互。
2)通过各种协议的映射（如SNMP、HTML等），提供了一个JMX代理和所有可管理组件的视图。
3)分布管理信息，以便构造一个分布式系统，也就是将高层管理平台的管理信息向其下众多的JMX代理发布。
4)收集多个JMX代理端的管理信息并根据管理终端用户的需要筛选用户感兴趣的信息并形成逻辑视图送给相应的终端用户。
5)提供了安全保证。

通过管理应用层和另一管理代理和以及他的设备层的联合，就可以为我们提供一个完整的网络管理的解决方案。这个解决方案为我们带来了独一无二的一些优点：轻便、根据需要部署、动态服务、还有安全性。

附加管理协议API

该层提供了一些API来支持当前已经存在的一些管理协议。

这些附加的协议API并没有定义管理应用的功能，或者管理平台的体系结构，他们仅仅定义了标准的Java API和现存的网络管理技术通信，例如SNMP。

网络管理平台和应用的开发者可以用这些API来和他们的管理环境进行交互，并将这个交互过程封装在一个JMX管理资源中。例如，通过SNMP可以对一个运行有SNMP代理的交换机进行管理，并将这些管理接口封装成为一个管理构件。在动态网络管理中，可以随时更换这些管理构件以适应需求。

这些API可以帮助开发者根据最通常的工业标准来部署他们的管理平台和应用。新的网路管理的解决方案可以和现存的基础结构合为一体，这样，现存的网络管理也能很好的利用基于Java技术的网络管理应用。

这些API目前在JCP（Java Community Process）内作为独立的JSR（Java Specification Request）开发。

他们包括：
1)SNMP Manager API
2)CIM/WBEM manager and protocol API

JMX的当前实现及应用

自从SUN发布了JMX规范，许多大公司纷纷行动起来，实现规范或者实现相应的基于JMX的网络管理系统，下面列出了当前的主要实现及应用情况：

1)SUN为JMX规范了作出了相应的参考实现，并在此基础上开发了一个全新的用于网络管理的产品JDMK（Java动态管理工具集），其中定义了资源的开发过程和方法、动态JMX代理的实现、远程管理应用的实现。同时，JDMK也提供了一个完整的体系结构用来构造分布式的网络管理系统，并提供了多种协议适配器和连接器，如SNMP协议适配器、HTML协议适配器、HTTP连接器、RMI连接器。
2)IBM Tivoli实现了JMX规范的产品为TivoliJMX，它为JAVA管理应用程序和网络提供了架构、设计模式、一些API集和一些服务。
3)Adventnet开发的关于JMX的产品为AdventNet Agent Toolkit，它使得定义新的SNMP MIB、开发JMX和Java SNMP Agent的过程自动化。
4)JBoss实现的J2EE应用服务器以JMX为微内核，各个模块以管理构件的形式提供相应的服务。
5)BEA的Weblogic应用服务器也将JMX技术作为自己的管理基础。
6)JFoxMX:huihoo.org推出的JMX1.2的规范实现,并通过SUN测试、认证。

总结

本课程详细介绍了JMX规范。

JMX体系结构分为四层，即设备层、代理层、分布服务层和附加协议API。

但SUN当前只实现了前两层的具体规范，其余的规范还在制定当中。

JMX代理要和远程应用程序通信，需要提供至少一个连接器和协议适配器。

参考资料

- http://java.sun.com/products/JavaManagement/
  SUN公司的JMX站点
- http://www.huihoo.org/jfox/jfoxmx/

  JFoxMX - SUN JMX1.2参考实现
- http://www.huihoo.com
  国内一个关于中间件的专业站点

结束

谢谢大家！

Allen@huihoo.com
http://www.huihoo.com