初学者的第一个JMX应用 - HelloJMX - 程序之路 - 博客频道

分类：
JMX *（4）*

JMX Notes ---- PART I (HelloJMX)kert 原创（参与分：558，专家分：520） 发表：2002-9-5 下午6:00 更新：2002-9-5 下午6:11 版本：1.0 阅读：**3617**次
[i]JMX越来越多得出现在各种技术杂志、白皮书和报告中，虽然它已不再是一个buzzword了。不仅仅是SUN?，最近许多厂商都宣布已经或是准备支持这一技术。不仅IBM、BEA、HP、Marcomedia(JRun)这些大的厂商，而且还有许多小的软件公司和开源项目也都加入了这一行列，包括AdventNet，JBoss等（JBoss3.0的Micorkernel结构就是建立在JMX的基础上）。
为什么JMX那么受欢迎，或是JMX到底有那些优势只得人们去关注？在本文与接下来的一系列文章中我们会一步步的学习和了解JMX。[/i]

简介
在学习第一个实例之前，让我们先对JMX进行一些初步的了解。正所谓"知根知底，百战百胜"么J。
先来了解一下JMX中的术语：

- MBean：是Managed Bean的简称。在JMX中MBean代表一个被管理的资源实例，通过MBean中暴露的方法和属性，外界可以获取被管理的资源的状态和操纵MBean的行为。事实上，MBean就是一个Java Object，同JavaBean模型一样，外界使用自醒和反射来获取Object的值和调用Object的方法，只是MBean更为复杂和高级一些。
- MBeanServer：MBean生存在一个MBeanServer中。MBeanServer管理这些MBean，并且代理外界对它们的访问。并且MBeanServer提供了一种注册机制，是的外界可以通过名字来得到相应的MBean实例。
- JMX Agent：Agent只是一个Java进程，它包括这个MBeanServer和一系列附加的MbeanService。当然这些Service也是通过MBean的形式来发布。
- Protocol Adapters and Connectors
  JMX Agent通过各种各样的Adapter和Connector来与外界(JVM之外)进行通信。同样外界（JVM之外）也必须通过某个Adapter和Connector来向JMX Agent发送管理或控制请求。Adapter和Connector的区别在于：Adapter是使用某种Internet协议来与JMX Agent获得联系，Agent端会有一个对象(Adapter)来处理有关协议的细节。比如SNMP Adapter和

HTTP Adapter。而Connector则是使用类似RPC的方式来访问Agent，在Agent端和客户端都必须有这样一个对象来处理相应的请求与应答。比如RMI Connector。
JMX Agent可以带有任意多个Adapter，因此可以使用多种不同的方式访问Agent。

JMX基本构架：

JMX分为三层，分别负责处理不同的事务。它们分别是：

- Instrumentation 层
  Instrumentation层主要包括了一系列的接口定义和描述如何开发MBean的规范。通常JMX所管理的资源有一个或多个MBean组成，因此这个资源可以是任何由Java语言开发的组件，或是一个JavaWrapper包装的其他语言开发的资源。
- Agent 层
  Agent用来管理相应的资源，并且为远端用户提供访问的接口。Agent层构建在Intrumentation层之上，并且使用并管理Instrumentation层内部描述的组件。通常Agent由一个MBeanServer和多个系统服务组成。另外Agent还提供一个或多个Adapter或Connector以供外界的访问。
  JMX Agent并不关心它所管理的资源是什么。
- Distributed 层
  Distributed层关心Agent如何被远端用户访问的细节。它定义了一系列用来访问Agent的接口和组件，包括Adapter和Connector的描述。

Hello JMX! Step by step

通常，在我们的项目中加入JMX管理框架需要接触到所有的以上提到的JMX的三层。接下来我们就实现一个简单的HelloJMX例子来一步步的介绍这三层框架。

Instrumentation层

Instrumentation层定义了一系列的接口和一套实现MBean的规范。我们使用JMX管理框架实现的每一个MBean都需要符合这一套接口和规范。在JMX中有两类MBean：静态的(Standard)MBean和动态的(Dynamic)MBean。Standard MBean实现简单，只需符合一套继承规范即可，特别适用于正在开发的项目。Dynamic MBean需要继承一个DynamicMBean接口，开发较复杂，但是可以在运行时动态修改因此灵活而功能强大。

我们在这里先实现一个StandardMBean。

为了实现StandardMBean，必须遵循一套继承规范。必须每一个MBean定义一个接口，而且这个接口的名字必须是其被管理的资源的对象类的名称后面加上"MBean"。例如：我们的

对象为kert.jmxnotes.HelloJMX，为了构造一个StandardMBean，我们必须定义的接口的名称为kert.jmxnotes.HelloJMXMBean。Agent依赖StandardMBean接口来访问被管理的资源，因此需要在HelloJMXMBean中定义相应的方法。

```
6   public interface HelloJMXMBean {
7       void sayHello();
8
9       void hello(String msg);
10
11      String getMessage();
12  }
```

在这个MBean中定义了三个方法，分别是sayHello()，hello(String)和getMessage()。

接下来是真正的资源对象，因为命名规范的限制，因此对象名称必须为HelloJMX。

```
9   public class HelloJMX
10          implements HelloJMXMBean {
11
12      private String msg;
13
14      public void sayHello() {
15          System.out.println("Hello JMX " + (msg == null?"":msg));
16      }
17
18      public void hello(String msg) {
19          this.msg = msg;
23      }
24
25      public String getMessage() {
26          return msg;
27      }
28  }
```

这样一个可以被JMX管理的资源就创建好了。

Agent层

通常JMX Agent是内置在我们的程序中的，也就是说它不是一个外置的服务。我们必须在我们的应用程序中显示的构造一个JMX Agent，来管理我们的资源。

```
25          final MBeanServer mBeanServer =
26                  MBeanServerFactory.createMBeanServer(DOMAIN);
27          final HelloJMX helloMBean = new HelloJMX();
28          final ObjectName helloON = new ObjectName(DOMAIN + ":name=HelloJMX");
29          mBeanServer.registerMBean(helloMBean, helloON);
```

(line25~26)通常我们首先需要建立一个MBeanServer，MBeanServer用来管理我们的MBean。我们通常是通过MBeanServer来获取我们MBean的信息，间接的调用MBean的方法。

(line27)然后生成我们的资源的一个对象。JMX Agent管理的资源和普通的Java对象并没有区别，因此使用通常的建立对象的方式即可。

(line28~29)然后，我们要显示的将这个对象注册到MBeanServer中去。JMX 使用SNMP规范中的ObjectName作为标识和查找MBean的方式。

之后，我们的Hello MBean就已经成功注册在MBeanServer中了，同其他Component/Container系统的模式一样，Container会代理它所管理的所有Component的行为。

Distributed 层

我们需要在远端管理我们的MBean，在JMX结构介绍中我们提过Agent可以有一个或多个Adapter或Connector以供远端用户访问只用。同MBeanServer一样我们也需要显示的说明我们的Agent可以支持那些Adaptor和Connector。

```
30          final HtmlAdaptorServer htmlAdaptor = new HtmlAdaptorServer();
31          final ObjectName htmlAdaptorON = new ObjectName(DOMAIN + ":name=HtmlA
daptor");
32          mBeanServer.registerMBean(htmlAdaptor, htmlAdaptorON);
33          htmlAdaptor.setPort(9999);
34
37          LOG.info("Starting the HtmlAdaptor....");
38          htmlAdaptor.start();
```

在Sun的JMX参考实现中，提供了一个HtmlAdaptor。支持Http访问协议，并且有一个不错的

Html界面。我们的HelloJMX就是用这个作为远端管理的界面。注意我们需要显示的设置HtmlAdaptor的端口号，和调用它的start()方法。因为事实上HtmlAdaptor是一个简单的HttpServer，它将Http请求转换为JMX Agent的请求。

运行这个程序，打开浏览器输入http://localhost:9999你就可以使用Browser作为你的管理界面，管理你的应用程序。

Notification
JMX也提供了一种通知机制。这种通知是由用户决定的，当应用出现某种状况时，可以利用通知来提醒管理人员。
为我们的HelloJMX加入通知也很简单。
首先要声明我们的应用(HelloJMX)支持JMX通知。只需修改一下我们的HelloJMX即可。

```
9    public class HelloJMX extends NotificationBroadcasterSupport
10           implements HelloJMXMBean {
```

同样HelloJMX也可以通过实现NotificationBroadcaster接口来支持通知机制。
然后修改hello(String):void方法。

```
18       public void hello(String msg) {
19           this.msg = msg;
20           final Notification notification = new Notification("kert.jmx.hello", this, -1,
21                   System.currentTimeMillis(), "message is changed");
22           sendNotification(notification);
23       }
```

我们想在hello方法被调用的时候发送一个通知。

同Java2的事件模型一样，通知必须有一个接受者。在JMX中，这个接受者的角色由实现了NotificationListener接口的对象完成。
在我们的例子中，我们让HelloAgent充当这个角色。

```
12   public class HelloAgent implements NotificationListener {
        .......
43       public void handleNotification(Notification notification, Object o) {
44           System.out.println(this.getClass().getName() +
```

```
45                  " Notification Listener --" + notification.getMessage());
46      }
```

小结

很简单不是么，这样就可以和JMX问声好了，还不赶快去？：）

顶

0