1. **JMX的Hello World**

一、JMX简介

**什么是JMX？在一篇网文中是这样说的："JMX(Java Management Extensions)是一个为应用程序植入管理功能的框架。JMX是一套标准的代理和服务，实际上，用户可以在任何Java应用程序中使用这些代理和服务实现管理"，这句话我现在看着还是不知所云，云里雾里。**

**我们还是从JMX能给我们提供什么好处入手来理解吧。举一个应用实例：在一个系统中常常会有一些配置信息，比如服务的IP地址，端口号什么的，那么如何来写这些代码呢？**

1. **程序初哥一般是写死在程序里，到要改变时就去改程序，然后再编译发布；**
2. **程序熟手则一般把这些信息写在一个配置文件里（JAVA一般都是\*.properties文件），到要改变时只要改配置文件，但还是重新启动系统，以便读取配置文件里的新值；**
3. **程序好手则会写一个段代码，把配置值缓存起来，系统在读值的时候，先看看配置文件有没有更动。如有更改则重读一遍，否则从缓存里读取值**
4. **程序高手则懂得取物为我所用，用JMX！把配置属性集中在一个类，然后写一个叫MBean的东东，再配置一下就轻松搞定了。而且JMX自动提供了一个WEB页面来给你来改变这些配置信息。**

**二、准备工作**

**JMX是一份规范，SUN依据这个规范在JDK（1.3、1.4、5.0）提供了JMX接口。而根据这个接口的实现则有很多种，比如Weblogic 的JMX实现、MX4J、JBoss的JMX实现。在SUN自己也实现了一份，不过在JDK1.4之前，这件JMX实现（一些JAR包）是可选的，你得去 它的网站上下载。JDK5.0则内嵌了进来，安装JDK5.0就可以开发基于JMX的代码了。**

**但JDK5.0并非包含所有SUN的关于JMX的代码，有一些工具类是排除在JDK5.0之外的。下面根据所使用的JDK版本情况，谈一谈开发环境的准备。**

**1、JDK1.3、1.4**

**去SUN网站下载SUN的JMX实现，共两个ZIP文件，下载网址：**[**http://java.sun.com/products/JavaManagement/download.html**](http://java.sun.com/products/JavaManagement/download.html)**。**

**（1）jmx-1\_2\_1-ri.zip**

**解压后的lib目录包含：jmxri.jar、jmxtools.jar**

**（2）jmx\_remote-1\_0\_1\_03-ri.zip**

**解压后的lib目录包含：jmxremote.jar、jmxremote\_optional.jar、rmissl.jar**

**如果在DOS下用命令行开发，则把这五个JAR包加入到classpath系统变量中。如果你用Eclipse开发，则把JAR包加入到项目属性的Libratries（库）引用中。**

**2、JDK5.0**

**JDK5.0的jre\lib\rt.jar已经包含了jmxri.jar、 jmxremote.jar、rmissl.jar三个包的代码。如果你用到jmxtools.jar、jmxremote\_optional.jar的 类，则需要将这两个类加入到classpath或Eclipse的项目库引用中。**

**3、我使用的开发环境：JDK5.0 ＋ Eclipse3.2。**

**注：因为用到jmxtools.jar中的HtmlAdaptorServer类，所以将此包加入到项目库引用中。jmxremote\_optional.jar暂时不用到，不管它。**

**三、HelloWorld实例**

**1、Hello是一个需要被管理的类（普通类）**

**java 代码**

1. **package com.sunhill.jmx;**
3. **public class Hello implements HelloMBean {**
4. **private String name;**
6. **public String getName() {**
7. **return name;**
8. **}**
10. **public void setName(String name) {**
11. **this.name = name;**
12. **}**
14. **public void printHello() {**
15. **System.out.println("Hello World, " + name);**
16. **}**
18. **public void printHello(String whoName) {**
19. **System.out.println("Hello , " + whoName);**
20. **}**
21. **}**

**2、要管理Hello则必须创建一个相应MBean，如下：**

**java 代码**

1. **package com.sunhill.jmx;**
3. **public interface HelloMBean {**
4. **public String getName();**
6. **public void setName(String name);**
8. **public void printHello();**
10. **public void printHello(String whoName);**
11. **}**

**说明：包含在MBean中方法都将是可以被管理的。MBean起名是有规范的，就是原类名后加上MBean字样。**

**3、创建一个Agent类**

**java 代码**

1. **package com.sunhill.jmx;**
3. **import java.lang.management.ManagementFactory;**
5. **import javax.management.MBeanServer;**
6. **import javax.management.ObjectName;**
8. **import com.sun.jdmk.comm.HtmlAdaptorServer;**
10. **public class HelloAgent {**
11. **public static void main(String[] args) throws Exception {**
12. **//      MBeanServer server = MBeanServerFactory.createMBeanServer();**
13. **MBeanServer server = ManagementFactory.getPlatformMBeanServer();**
14. **ObjectName helloName = new ObjectName("**chengang**:name=HelloWorld");**
15. **server.registerMBean(new Hello(), helloName);**
16. **ObjectName adapterName = new ObjectName(**
17. **"HelloAgent:name=htmladapter,port=8082");**
18. **HtmlAdaptorServer adapter = new HtmlAdaptorServer();**
19. **server.registerMBean(adapter, adapterName);**
20. **adapter.start();**
21. **System.out.println("start.....");**
22. **}**
23. **}**

**说明：**

* **先创建了一个MBeanServer，用来做MBean的容器**
* **将Hello这个类注入到MBeanServer中，注入需要创建一个ObjectName类**
* **创建一个AdaptorServer，这个类将决定MBean的管理界面，这里用最普通的Html型界面。AdaptorServer其实也是一个MBean。**
* **ghl:name=HelloWorld的名字是有一定规则的，格式为：“域名:name=MBean名称”，域名和MBean名称都可以任意取。**

**4、运行HelloAgent，然后打开网页：**[**http://localhost:8082/**](http://localhost:8082/)**,看效果!**

**五、总结**

**在实际系统中我们可以把name变成决定数库链接池的变量，这样我就可以对系统的运行参数进行实现的监控和配置（管理）。而且也可以对一些方法（如printHello）进行远程调用了。**

## JMX简介

一、JMX简介

　　JMX是一种JAVA的正式规范，它主要目的是让程序且有被管理的功能，那么怎么理解所谓的“被管理”呢？试想你开发了一个软件（如ＷＥＢ网 站），它是在２４小时不简断运行的，那么你可能会想要“监控”这个软件的运行情况，比如收到了多少数据，有多少人登录等等。或者你又想“配置”这个软件， 比如现在访问人数比较多，你想把数据连接池设置得大一些。

　　当然，你也许会专门为这些管理来开发软件，但如果你借助JMX，则会发现创建这样的管理程序是如此简单。因为你无需为管理程序来开发界面，已经 有通用的JMX管理软件，如MC4J，或者是用一般都附带提供的HTML网页来管理，你要做的仅仅是将自己要被管理和监控类的按照JMX规范修改一下即 可。

　　中间件软件WebLogic的管理页面就是基于JMX开发的，而JBoss则整个系统都基于JMX构架。下面将JMX的一些概念，从JMX规范转帖如下：

二、JMX构架中的各层及相关的组件

1. 工具层（Instrumentation Level）
2. (a) MBeans（标准的，动态的，开放的和模型MBeans）
3. (b) 通知模型：Notification、NotificationListener等类
4. (c) MBean元数据类：Attribute、Opreator等类
5. 代理层（Agent Level）
6. (a) MBean Server
7. (b) 代理服务。如前一篇的HtmlAdaptorServer等。

　　MBean中有getter和setter的就是属性，如前一篇的Hello类中Name。如果只有getter则表示该属性只读。一共有四种MBean，如下：

1. 标准MBeans（Standard MBeans）设计和实现是最简单的，这类MBean使用自己的方法名作为管理接口；
2. 动态MBeans（Dynamic MBeans）必须实现一个指定的接口，由于动态MBeans在运行期间暴露它们的管理接口，因此更为灵活；
3. 开放MBeans（Open MBeans）属于动态MBeans，这类MBean依靠基础数据类型来实现通用管理，并为友情用户进行自我声明；
4. 模型MBeans（Model MBeans）同样也是动态MBeans，这类MBeans是完全可配置的，在运行期间进行自我声明；它们为资源动态工具提供一个一般性的，有默认行为的MBeans类。

　　在前一篇中的Hello、HelloMBean就是一个标准MBeans（Standard MBeans）。后面接下来的几篇，我们会继续介绍其他几种MBean。

## Notification的使用

**一、简介**

　　Mbean之间的通信是必不可少的，Notification就起到了在Mbean之间沟通桥梁的作用。JMX notification 由四部分组成：

* Notification 这个相当于一个信息包，封装了需要传递的信息
* Notification broadcaster　这相当于一个广播器，把消息广播出去
* Notification listerner　这是一个监听器，用于监听广播出来的Notification消息
* Notification filter　这是一个过滤器，过滤掉不需要的Notification消息

　 Notification broadcaster不需要我们实现，JMX的内部已经有了。Notification filter一般也很少用。下面的例子主要用到了Notification、Notification listerner。

**二、实例**

　　在第一篇的Hello中有一个printHello(String whoName)方法，意思根据碰到的是谁来打招呼，比如：

　　Jack从对面走过来，说：“hi”

　　我们回之以礼，说：“Hello, jack”

　　首先这需要Jack先说一个hi（相应一个操作方法），然后他说的话封装成声波（相当Notification消息包）传递出去。然后我们 还要给Jakc装上一个监听器（Hello的耳朵？？^\_^），这个监听器将捕捉到Jack的声波语音包，并进行相应处理，即说“Hello, jack”。

　　好，我们看看如何实现的：

1、Jack类及其相应的MBean

   我们把Jack写成一个MBean，如下：

java 代码

1. import javax.management.Notification;
2. import javax.management.NotificationBroadcasterSupport;
3. public class Jack extends NotificationBroadcasterSupport implements JackMBean {
4. private int seq = 0;
5. public void hi() {
6. Notification n = new Notification(//创建一个信息包
7. "jack.hi",//给这个Notification起个名称
8. this, //由谁发出的Notification
9. ++seq,//一系列通知中的序列号,可以设任意数值
10. System.currentTimeMillis(),//发出时间
11. "Jack");//发出的消息文本
12. //发出去
13. sendNotification(n);
14. }}
15. }
16. }

说明：

* 必需继承NotificationBroadcasterSupport
* 此类只有一个hi方法，方法只有两句：创建一个Notification消息包，然后将包发出去
* 如果你还要在消息包上附加其他数据，Notification还有一个setUserData方法可供使用

2、接下来是他的MBean

java 代码

1. public interface JackMBean {
2. public void hi();
3. }

3、创建一个Listener，监听到的Notification消息包将由此类负责处理。

java 代码

1. import javax.management.Notification;
2. import javax.management.NotificationListener;
3. public class HelloListener implements NotificationListener {
4. public void handleNotification(Notification n, Object handback) {
5. System.out.println("type=" + n.getType());
6. System.out.println("source=" + n.getSource());
7. System.out.println("seq=" + n.getSequenceNumber());
8. System.out.println("send time=" + n.getTimeStamp());
9. System.out.println("message=" + n.getMessage());
10. if (handback != null) {
11. if (handback instanceof Hello) {
12. Hello hello = (Hello) handback;
13. hello.printHello(n.getMessage());
14. }
15. }
16. }}

4、修改HelloAgent如下：

java 代码

1. import javax.management.MBeanServer;
2. import javax.management.MBeanServerFactory;
3. import javax.management.ObjectName;
4. import com.sun.jdmk.comm.HtmlAdaptorServer;
5. public class HelloAgent {
6. public static void main(String args[]) throws Exception{
7. MBeanServer server = MBeanServerFactory.createMBeanServer();
8. ObjectName helloName = new ObjectName("chengang:name=HelloWorld");
9. Hello hello=new Hello();
10. server.registerMBean(hello, helloName);
11. ObjectName adapterName = new ObjectName("HelloAgent:name=htmladapter,port=8082");
12. HtmlAdaptorServer adapter = new HtmlAdaptorServer();
13. server.registerMBean(adapter, adapterName);
14. Jack jack = new Jack();    //重点
15. server.registerMBean(jack, new ObjectName("HelloAgent:name=jack"));    //重点
16. jack.addNotificationListener(new HelloListener(), null, hello);    //重点
17. adapter.start();
18. System.out.println("start.....");
19. }}

**三、运行**

1、先运行HelloAgent启动服务，再打开浏览器输入网址：<http://localhost:8082/>

2、　进入“name=jack”项，然后单击“hi”按钮来执行它。

**四、总结**

Notification和Java的事件模型是一样的，另外如果你买了《Eclipse从入门到精通》， 你会发现第22.4节也使用了和Notification和Java的事件模型相同的设计方式。Notification在我们的实际项目中也用到了，像我们现在的给移动做的项目中（基于JMX实现)，分散在各地方的工作站的日志，就是通过Notification方式，把每条产生的日志封装在 Notification中实时发回主控服务器的。有机会我会发这一系统的关于日志的设计方案写一下，它实现了对各地工作站的集中的、实时的监控，非常实用。

## 动态MBean：DynamicMBean

**一、前言**

　　动态MBean是在运行期才定义它的属性和方法，也就是说它有什么属性和方法是可以动态改变的。动态MBean主要利用一些辅助类（构造函数类MBeanConstructorInfo、属性类MBeanAttributeInfo、方法类MBeanOperationInfo）来完成这个 功能，所有的动态MBean必须实现DynamicMBean接口。DynamicMBean写好后，使用方法和第一篇文章中普通的MBean一样。

　　给出一个动态MBean的实例，这个实例最初动态构了一个Name属性及一个print方法，当我们执行它的print方法之后，又给此MBean新增了一个print1方法。实例的代码如下：

**二、实例**

1、HelloDynamic类

java 代码

1. import java.lang.reflect.Constructor;
2. import java.util.Iterator;
3. import javax.management.Attribute;
4. import javax.management.AttributeList;
5. import javax.management.DynamicMBean;
6. import javax.management.MBeanAttributeInfo;
7. import javax.management.MBeanConstructorInfo;
8. import javax.management.MBeanException;
9. import javax.management.MBeanInfo;
10. import javax.management.MBeanNotificationInfo;
11. import javax.management.MBeanOperationInfo;
12. import javax.management.MBeanParameterInfo;
13. import javax.management.ReflectionException;
15. /\*\*
16. \* @author Sunny Peng
17. \* @add a feature for dynamic add operation
18. \* @version 1.0
19. \*/
20. public class HelloDynamic implements DynamicMBean {
21. //这是我们的属性名称
22. private String name;
23. private MBeanInfo mBeanInfo = null;
24. private String className;
25. private String description;
26. private MBeanAttributeInfo[] attributes;
27. private MBeanConstructorInfo[] constructors;
28. private MBeanOperationInfo[] operations;
29. MBeanNotificationInfo[] mBeanNotificationInfoArray;
31. public HelloDynamic() {
32. init();
33. buildDynamicMBean();
34. }
36. private void init() {
37. className = this.getClass().getName();
38. description = "Simple implementation of a dynamic MBean.";
39. attributes = new MBeanAttributeInfo[1];
40. constructors = new MBeanConstructorInfo[1];
41. operations = new MBeanOperationInfo[1];
42. mBeanNotificationInfoArray = new MBeanNotificationInfo[0];
43. }
45. private void buildDynamicMBean() {
46. //设定构造函数
47. Constructor[] thisconstructors = this.getClass().getConstructors();
48. constructors[0] = new MBeanConstructorInfo("HelloDynamic(): Constructs a HelloDynamic object", thisconstructors[0]);
49. //设定一个属性
50. attributes[0] = new MBeanAttributeInfo("Name", "java.lang.String", "Name: name string.", true, true, false);
51. //operate method 我们的操作方法是print
52. MBeanParameterInfo[] params = null;//无参数
53. operations[0] = new MBeanOperationInfo("print", "print(): print the name", params, "void", MBeanOperationInfo.INFO);
54. mBeanInfo = new MBeanInfo(className, description, attributes, constructors, operations, mBeanNotificationInfoArray);
55. }
57. //动态增加一个print1方法
58. private void dynamicAddOperation() {
59. init();
60. operations = new MBeanOperationInfo[2];//设定数组为两个
61. buildDynamicMBean();
62. operations[1] = new MBeanOperationInfo("print1", "print1(): print the name", null, "void", MBeanOperationInfo.INFO);
63. mBeanInfo = new MBeanInfo(className, description, attributes, constructors, operations, mBeanNotificationInfoArray);
64. }
66. public Object getAttribute(String attribute\_name) {
67. if (attribute\_name != null)
68. return null;
69. if (attribute\_name.equals("Name"))
70. return name;
71. return null;
72. }
74. public void setAttribute(Attribute attribute) {
75. if (attribute == null)
76. return;
77. String Name = attribute.getName();
78. Object value = attribute.getValue();
79. try {
80. if (Name.equals("Name")) {
81. // if null value, try and see if the setter returns any exception
82. if (value == null) {
83. name = null;
84. // if non null value, make sure it is assignable to the attribute
85. } else if ((Class.forName("java.lang.String")).isAssignableFrom(value.getClass())) {
86. name = (String) value;
87. }
88. }
89. } catch (Exception e) {
90. e.printStackTrace();
91. }
92. }
94. public AttributeList getAttributes(String[] attributeNames) {
95. if (attributeNames == null)
96. return null;
97. AttributeList resultList = new AttributeList();
98. // if attributeNames is empty, return an empty result list
99. if (attributeNames.length == 0)
100. return resultList;
101. for (int i = 0; i < attributeNames.length; i++) {
102. try {
103. Object value = getAttribute(attributeNames[i]);
104. resultList.add(new Attribute(attributeNames[i], value));
105. } catch (Exception e) {
106. e.printStackTrace();
107. }
108. }
109. return resultList;
110. }
112. public AttributeList setAttributes(AttributeList attributes) {
113. if (attributes == null)
114. return null;
115. AttributeList resultList = new AttributeList();
116. // if attributeNames is empty, nothing more to do
117. if (attributes.isEmpty())
118. return resultList;
119. // for each attribute, try to set it and add to the result list if successfull
120. for (Iterator i = attributes.iterator(); i.hasNext();) {
121. Attribute attr = (Attribute) i.next();
122. try {
123. setAttribute(attr);
124. String name = attr.getName();
125. Object value = getAttribute(name);
126. resultList.add(new Attribute(name, value));
127. } catch (Exception e) {
128. e.printStackTrace();
129. }
130. }
131. return resultList;
132. }
134. public Object invoke(String operationName, Object params[], String signature[]) throws MBeanException, ReflectionException {
135. // Check for a recognized operation name and call the corresponding operation
136. if (operationName.equals("print")) {
137. //具体实现我们的操作方法print
138. System.out.println("Hello, " + name + ", this is HellDynamic!");
139. dynamicAddOperation();
140. return null;
141. } else if (operationName.equals("print1")) {
142. System.out.println("这是动态增加的一方法print1");
143. return null;
144. } else {
145. // unrecognized operation name:
146. throw new ReflectionException(new NoSuchMethodException(operationName), "Cannot find the operation " + operationName + " in " + className);
147. }
149. }
151. public MBeanInfo getMBeanInfo() {
152. return mBeanInfo;
153. }
154. }

说明：

* 实现于接口DynamicMBean
* 借助于各种辅助类完成一个类的构造。构造函数类MBeanConstructorInfo、属性类MBeanAttributeInfo、方法类MBeanOperationInfo
* 这里所有public方法是实现于DynamicMBean的。主要提供：setAttribute设置属性、getAttribute 取得属性、setAttributes设置一组属性、getAttributes取得一组属性、invoke方法调用、getMBeanInfo　 MBeanServer由这个方法得到关键的MBean类的构造信息。

2、HelloAgent类

　　前面说了HelloDynamic和普通MBean的使用方法是一样的，因此HelloAgent和第一篇的HelloAgent基本一样，就是把Hello改成HelloDynamic而已。为了实例完整，也一并帖出来吧。

java 代码

1. import javax.management.MBeanServerFactory;
2. import javax.management.ObjectName;
3. import com.sun.jdmk.comm.HtmlAdaptorServer;
4. public class HelloAgent {
5. public static void main(String[] args) throws Exception {
6. MBeanServer server = MBeanServerFactory.createMBeanServer();
7. ObjectName helloName = new ObjectName("chengang:name=HelloDynamic");
8. HelloDynamic hello = new HelloDynamic();
9. server.registerMBean(hello, helloName);
10. ObjectName adapterName = new ObjectName("HelloAgent:name=htmladapter,port=8082");
11. HtmlAdaptorServer adapter = new HtmlAdaptorServer();
12. server.registerMBean(adapter, adapterName);
13. adapter.start();
14. System.out.println("start.....");
15. }
16. }

**3、运行**

　　先运行HelloAgent。再打开浏览器，输入网址：<http://localhost:8082/>。单击进入“[name=HelloDynamic](http://localhost:8082/ViewObjectRes/chengang:name=HelloDynamic) ”项，执行print方法后再回到上一页面你会发现又多了一个print1方法。

**4、总结**

**动态MBean的代码稍显复杂，但对于一些特殊需求的情况，它将显示出强大威力。而且它还是模型MBeans（Model MBeans）的基础。不过在一般的项目中，动态MBean还是用得比较少，所谓利器深藏之而不用，非常时方现光芒。**

## 用Apache的commons-modeler来辅助开发JMX

一、前言

　　每一个MBean都要有一个接口，比如前面的Hello要有一个HelloMBean接口。要多维护一个接口，的确是件麻烦的事。 Apache的commons-modeler利用JMX中的动态MBean原理很好的解决了这一问题，commons-modeler使用得我们可以只 写Hello，而不用写HelloMBean这个接口。不过这是有代价的，它要求我们写一个mbean的xml描述文件（唉，少了一件事，却又多出另一件 事来）。但commons-modeler还是有优点的，就是它让mbean的装配更加灵活，把多个mbean的装配都集中在一个XML文件里来了。

　　开始实例之前，你需要先去apache网站下载commons-modeler，以及modeler的依赖项目commons-logging。下载 网址为：http://jakarta.apache.org/site/downloads/downloads\_commons.html，下载的文 件是ZIP压缩包，解压后找到commons-logging.jar和commons-modeler.jar。如果在DOS下用命令行开发，则把这两 个JAR包加入到classpath系统变量中。如果你用Eclipse开发，则把JAR包加入到项目属性的Libratries（库）引用中。

二、HelloWorld实例

 　我们以本系统的第一篇“1、JMX的Hello World”为例，来重新实现一次。

1、Hello.java的代码不变（注：为了在Eclipse上和原来的Hello文件放在不同的地方，我把新Hello放到了mbean.modelbean包），如下：

java 代码

1. package mbean.modelbean;
3. import mbean.standard.HelloMBean;
5. public class Hello implements HelloMBean {
6. private String name;
7. public String getName() {
8. return name;
9. }
10. public void setName(String name) {
11. this.name = name;
12. }
13. public void printHello() {
14. System.out.println("Hello World, " + name);
15. }
16. public void printHello(String whoName) {
17. System.out.println("Hello , " + whoName);
18. }
19. }

2、MBean不用写了，但需要写一个XML描述文件。文件名任取，这里取名为：mbeans-descriptors

xml 代码

1. <?xml version="1.0"?>
2. <mbeans-descriptors>
3. <mbean name="Hello" description="the hello bean" domain="chengang" group="helloGroup" type="mbean.modelbean.Hello">
4. <attribute name="name" description="a name attribute" type="java.lang.String" writeable="true"/>
5. <operation name="printHello" description="a operation to print hello" impact="INFO" returnType="String"/>
6. </mbean>
7. </mbeans-descriptors>

这里只对<mbean>标签做一下说明：

    \* name mbean在xml中的唯一标识，不一定要和类同名

    \* description　mbean的注释说明信息

    \* domain mbean所属域

    \* group mbean所属组

    \* type mbean的类全名（包名＋类名）

    \* classname 指定实现代理功能的ModelMbean的全名，如果不指定则默认为BaseModelMBean

3、接下来改写HelloAgent

java 代码

1. package mbean.modelbean;
3. import java.io.InputStream;
4. import javax.management.MBeanServer;
5. import javax.management.ObjectName;
6. import javax.management.modelmbean.ModelMBean;
7. import org.apache.commons.modeler.ManagedBean;
8. import org.apache.commons.modeler.Registry;
9. import com.sun.jdmk.comm.HtmlAdaptorServer;
11. public class HelloAgent {
12. public static void main(String[] args) throws Exception {
13. //基于xml中的信息构建一个Registry
14. Registry registry = Registry.getRegistry(null, null);
15. InputStream stream = HelloAgent.class.getResourceAsStream("Mbeans-descriptors.xml");
16. registry.loadMetadata(stream);
17. stream.close();
18. //由Registry得到一个MBeanServer
19. MBeanServer server = registry.getMBeanServer();
21. //得到Hello在描述文件中的信息类，对应于xml文件<mbean>标签的name属性。
22. ManagedBean managed = registry.findManagedBean("Hello");
23. //创建ObjectName
24. ObjectName helloName = new ObjectName(managed.getDomain() + ":name=HelloWorld");
25. //得到ModelMBean
26. ModelMBean hello = managed.createMBean(new Hello());
27. //注册MBean
28. server.registerMBean(hello, helloName);
30. ObjectName adapterName = new ObjectName("HelloAgent:name=htmladapter,port=8082");
31. HtmlAdaptorServer adapter = new HtmlAdaptorServer();
32. server.registerMBean(adapter, adapterName);
33. adapter.start();
34. System.out.println("start.....");
35. }
36. }

HelloAgent是效复杂的地方，我们来和以前的HelloAgent逐步比较一下，前后有什么不同：

（1）首先，新的HelloAgent需要将xml信息读入到Registry对象中，这是旧HelloAgent所没有的。

Registry registry = Registry.getRegistry(null, null);

InputStream stream = HelloAgent.class.getResourceAsStream("Mbeans-descriptors.xml");

registry.loadMetadata(stream);

stream.close();

（2）接着创建MBeanServer的方式也不同了

现在：MBeanServer server = registry.getMBeanServer();

以前: MBeanServer server = MBeanServerFactory.createMBeanServer();

（3）Hello相应的ObjectName创建也略不相同。主要是域名在XML描述文件里设置好了。

现在：ManagedBean managed = registry.findManagedBean("Hello");

     ObjectName helloName = new ObjectName(managed.getDomain() + ":name=HelloWorld");

以前：ObjectName helloName = new ObjectName("chengang:name=HelloWorld")

（4）得到MBean的方式也不同，这里就是关键的不同点

现在：ModelMBean hello = managed.createMBean(new Hello());

以前：Hello hello = new Hello();

注意：为什么现在要比以前多一个createMbean步骤呢，就是因为现在的写法并没有写Mbean，所以需要动态才需要生成一个。而以前就直接把 new Hello()注册到mbean server就可以了，server会自动找到它的HelloMBean接口文件。

　也就上面四点区别，其他代码完全一样。测试和查看效果的方法和以前一样，在此不累述了。

　commons Modeler为Hello动态生成了一个MBean接口：BaseModelBean

**参考资料**

|  |
| --- |
| 用Commons Modeler 开发JMX应用  Modeler组件是Jakarta Commons 项目针对Model MBeans提供的一个便利的开发组件。  首先介绍一下基本的概念：Managed bean简称Mbean，是对可被管理的资源的抽象定义，ModelBean是JMX定义的Mbean中动态和灵活的一种。但是要实现它开发人员必须设置大量的Metadata信息。Modeler组件针对ModelBean的开发，通过采用xml来定义metadata的方式来简化了ModelBean的开发强度，同时它和提供了注册的支持和缺省的ModelMbean的支持。这些我们都将在下面的例子中看到。另外关于JMX和ModelMBean的介绍有很多可以参考。这里都不提了。    下面主要通过例子来写一下如何使用它进行JMX开发。  在实际开发中，我们都是首先定义需要被管理的资源对象，然后把它注册到MBeanServer中进行发布。最后再通过客户端访问。  所以我的例子也通过这个来完成。  1． 资源对象和MBean的实现  首先编写一个需要管理的对象。该对象很简单有一个属性和一个操作。  public class TestBean {      private String oneAttr;      public String getOneAttr(){          return "one attribute be testing";      }      public void setOneAttr(String attr){          this.oneAttr = attr;      }     public String toString(){          return "toString be testing";      }  }  根据需要被管理的要求编写Mbeans-descriptors.xml  <?xml version="1.0"?>  <mbeans-descriptors>    <mbean         name="TestBean"            description="the test bean"                 domain="mydomain"                  group="testGroup"                   type="mbeans.TestBean">      <attribute   name="oneAttr"            description="one Attr to be tested"                   type="java.lang.String"              writeable="true"/>      <operation   name="toString"            description="one Oper to be tested"                 impact="INFO"             returnType="String">      </operation>    </mbean>  </mbeans-descriptors>    通过这个xml文件的定义就描述了ModelBean所需要的metadata信息和一个基本的ModelMBean的实现。  关于这个xml文件有几个需要说明的地方：  <mbean>的属性classname,name和type，  name属性是每个Mbean被Registry对象注册的对象名。  type属性是是真正被管理资源的全名（包括包名）。  classname属性是用户扩展的用于实现代理功能的ModelMbean的全名，如果不提供Modeler会使用BaseModelMBean；如果提供了代理的ModelMBean对象，在使用时可以使用如下的代码样本访问他所代理的资源对象。   Function foo(){                TestObject object = this.resource;  //得到所代理的对象。                Object.foo();                                     //具体实现的调用。  }  Mbeans-descriptors.xml的其他的部分都比较好理解，具体参考DTD就可以了。  下面我们就要把完成MBeanServer的实现和Mbean的注册。    2． MBeanServer的实现和MBean的注册  MBeanServer的实现代码：          registry = Registry.getRegistry();          InputStream stream = MyMBeanServer.class                  .getResourceAsStream("Mbeans-descriptors.xml");         registry.loadRegistry(stream);         stream.close();         mserver = registry.getMBeanServer();    通过Modeler组件提供的Registry对象，可以很方便的完成MBeanServer的创建。  MBean的注册代码：                /\*创建被管理的资源实例\*/              TestBean bean = new TestBean();                /\*Modeler组件提供的对managedbean配置信息的封装对象，通过它建立资源和ModelMBean的联系\*/          ManagedBean managed = registry.findManagedBean("TestBean");          String domain = managed.getDomain();                /\*得到ModelMbean\*/          ModelMBean mbean = managed.createMBean(bean);                /\*创建ObjectName\*/          ObjectName oname = new ObjectName(domain+":test=zcx");                /\*注册MBean\*/          mserver.registerMBean(mbean,oname);    3． MBeanServerConnector的实现。  在JMXRemoteAPI 发布之后，Agent需要提供ConnectorServer，我们可以这样注册一个JMXMP的ConnectorServer.            /\*定义JMXServiceURL这个对象是客户端和服务器联系的关键\*/        JMXServiceURL address = new JMXServiceURL("jmxmp", null, 7777);        connectorSrv=JMXConnectorServerFactory.newJMXConnectorServer(address, null, null);        ObjectName csName = new ObjectName(":type=cserver,name=mycserver");        mserver.registerMBean(connectorSrv, csName);         connectorSrv.start();    在这里需要说明的是通过JMXServiceURL的可以定义ConnectorServer的类型是JMXMP,RMI或者其他什么类型。按照规范规定这个JMXServiceURL应该在服务器端通过JNDI或者SLP等进行发布。然后客户端通过相应的访问得到这个对象，进行通信。如何通过JNDI访问这个对象，我就不在这里写了。  4． MBeanServerAdaptor的实现。        madaptor = new HtmlAdaptorServer();          mserver.registerMBean(madaptor,new,ObjectName("adaptor:protocol=HTTP"));          madaptor.setPort(8888);             madaptor.start();    5． MBeanClient的实现。    使用http浏览器，我们可以通过HtmlAdaptorServer访问MBean。    另外我们通过JMXRemoteAPI也可以在客户端访问MBean。  客户端代码如下：  JMXServiceURL address = new  JMXServiceURL("jmxmp","zcx" , 7777);  connector = JMXConnectorFactory.connect(address);  mbsConn = connector.getMBeanServerConnection();    通过MBeanServerConnection对象我们可以对MBean进行各种操作。    到这里一个简单的JMX的应用流程就完成了。 |

## 模型Bean：Model Bean

在上一节是用apache的commons-modeler来开发的一个model，只不过commons-modeler帮助我们实现了很多的代 码，而我们只需要写描述XML文件就行了。这一节，来一个实打实的Model Bean，不借助任何第三方工具包。例子还是沿用Hello这个类，以便于和以前的实现相比较。

一、Model MBean实例

1、Hello.java还是和以前的一样。这里它没有再加上一个MBean接口了，只是一个很普通的类。

java 代码

1. public class Hello{
2. private String name;
3. public String getName() {
4. return name;
5. }
6. public void setName(String name) {
7. this.name = name;
8. }
9. public void printHello() {
10. System.out.println("Hello World, " + name);
11. }
12. public void printHello(String whoName) {
13. System.out.println("Hello , " + whoName);
14. }
15. }

2、接下来是HelloAgent的写法，和以前就差一句：把“new Hello()”这一句删除了，加上了ModelMbeanUtils.createModlerMbean();

java 代码

1. import javax.management.MBeanServer;
2. import javax.management.MBeanServerFactory;
3. import javax.management.ObjectName;
4. import javax.management.modelmbean.RequiredModelMBean;
6. import com.sun.jdmk.comm.HtmlAdaptorServer;
8. public class HelloAgent {
9. public static void main(String[] args) throws Exception {
10. MBeanServer server = MBeanServerFactory.createMBeanServer();
11. ObjectName helloName = new ObjectName("chengang:name=HelloWorld");
12. //Hello hello = new Hello();
13. RequiredModelMBean hello = ModelMBeanUtils.createModlerMBean();
14. server.registerMBean(hello, helloName);
15. ObjectName adapterName = new ObjectName("HelloAgent:name=htmladapter,port=8082");
16. HtmlAdaptorServer adapter = new HtmlAdaptorServer();
17. server.registerMBean(adapter, adapterName);
18. adapter.start();
19. System.out.println("start.....");
20. }
21. }

3、ModelMbeanUtils这个类是要我们自己来实现的，也是写model Bean最麻烦的地方，它主要是返回一个RequiredModelMBean类，此类主要包括了一个ModelMBeanInfo类的信息。在 ModelMBeanInfo中定义了所有对需要管理的属性和方法的描述。具体代码如下：

java 代码

1. import javax.management.MBeanParameterInfo;
2. import javax.management.modelmbean.ModelMBeanAttributeInfo;
3. import javax.management.modelmbean.ModelMBeanInfo;
4. import javax.management.modelmbean.ModelMBeanInfoSupport;
5. import javax.management.modelmbean.ModelMBeanOperationInfo;
6. import javax.management.modelmbean.RequiredModelMBean;
7. public class ModelMBeanUtils {
8. private static final boolean READABLE = true;
9. private static final boolean WRITABLE = true;
10. private static final boolean BOOLEAN = true;
11. private static final String STRING\_CLASS = "java.lang.String";
12. public static RequiredModelMBean createModlerMBean() {
13. RequiredModelMBean model = null;
14. try {
15. model = new RequiredModelMBean();
16. model.setManagedResource(new Hello(), "ObjectReference");
17. ModelMBeanInfo info = createModelMBeanInfo();
18. model.setModelMBeanInfo(info);
19. } catch (Exception e) {
20. e.printStackTrace();
21. }
22. return model;
23. }
24. private static ModelMBeanInfo createModelMBeanInfo() {
25. //////////////////////////////////////////////////////////////////
26. //                        属性                                        //
27. //////////////////////////////////////////////////////////////////
28. // 构造name属性信息
29. ModelMBeanAttributeInfo nameAttrInfo = new ModelMBeanAttributeInfo(//
30. "Name", // 属性名
31. STRING\_CLASS, //属性类型
32. "people name", // 描述文字
33. READABLE, WRITABLE, !BOOLEAN, // 读写
34. null // 属性描述子
35. );
36. //////////////////////////////////////////////////////////////////
37. //                        方法                                        //
38. //////////////////////////////////////////////////////////////////
39. //构造 printHello()操作的信息
40. ModelMBeanOperationInfo print1Info = new ModelMBeanOperationInfo(//
41. "printHello", //
42. null, //
43. null, //
44. "void", //
45. MBeanOperationInfo.INFO, //
46. null //
47. );
48. // 构造printHello(String whoName)操作信息
49. ModelMBeanOperationInfo print2Info;
50. MBeanParameterInfo[] param2 = new MBeanParameterInfo[1];
51. param2[0] = new MBeanParameterInfo("whoName", STRING\_CLASS, "say hello to who");
52. print2Info = new ModelMBeanOperationInfo(//
53. "printHello", //
54. null,//
55. param2,//
56. "void", //
57. MBeanOperationInfo.INFO, //
58. null//
59. );
60. //////////////////////////////////////////////////////////////////
61. //                        最后总合                                    //
62. //////////////////////////////////////////////////////////////////
63. // create ModelMBeanInfo
64. ModelMBeanInfo mbeanInfo = new ModelMBeanInfoSupport(//
65. RequiredModelMBean.class.getName(), // MBean类
66. null, // 描述文字
67. new ModelMBeanAttributeInfo[] { // 所有的属性信息（数组）
68. nameAttrInfo },//只有一个属性
69. null, // 所有的构造函数信息
70. new ModelMBeanOperationInfo[] { // 所有的操作信息（数组）
71. print1Info,
72. print2Info },//
73. null, // 所有的通知信息(本例无)
74. null//MBean描述子
75. );
76. return mbeanInfo;
77. }
78. }

4、看效果的方法和以前一样，运行HelloAgent，用浏览器打开：http://localhost:8082 。效果图和standard mbean一样，就不再帖出来了，去第一篇去看吧

二、总结

   我们发现模型Mbean(Model MBean)要比标准MBean(standard mbean)复杂多了，那有什么理由让我们选择使用模型MBean吗？我认为，最大的理由就是模型MBean可以动态配置。试想一下这个应用场景：由于安 全或其他原因，系统要把某个MBean公开的可管理方法隐藏起来。这时，如果你是用标准MBean，这需要修改接口类，然后重新编译发布；如果用 Apache commons-modeler来写的模型MBean，则只需要修改XML文件就行了，不需要重新编译发布（可能要重启一下系统）。这就是模型Mbean 优势之所在了。

　细心的人会发现动态MBean和这一节的模型Mbean非常相似，但它们还是有很大不同的：动态MBean没有Hello类，它要自己实现Hello类中的方法逻辑。

## 7、用JDK5.0的JConsole来连接MBean

前面所有看效果都是通过Html网页来看的。JDK5.0自带了一个jmx客户端，叫jconsole，位于c:\jdk\bin\jconsole.exe。我们来用用这个客户端来连接Mbean Server。

一、vm参数方式

1、还是用第一篇的那个HelloAgent，修改HelloAgent，将第一句：

MBeanServer server = MBeanServerFactory.createMBeanServer();

改为：MBeanServer server = ManagementFactory.getPlatformMBeanServer();

注：ManagementFactory的全路径为：java.lang.management.ManagementFactory

2、修改Eclipse的run选项，把“-Dcom.sun.management.jmxremote=HelloAgent”这一句加入到run选项中

3、运行HelloAgent，然后在Dos窗口输入“jconsole”来启到JConsole

4、单击“连接”

二、RMI方式

　　还是用jconsole，但方式变了。这里不需要象上面那样修改Eclipse run的vm选项。

1、还是用第一篇的HelloAgent，加上一段代码，启动一个JMXConnectorServer服务

java 代码

1. import javax.management.MBeanServer;
2. import javax.management.MBeanServerFactory;
3. import javax.management.ObjectName;
4. import javax.management.remote.JMXConnectorServer;
5. import javax.management.remote.JMXConnectorServerFactory;
6. import javax.management.remote.JMXServiceURL;
8. import com.sun.jdmk.comm.HtmlAdaptorServer;
10. public class HelloAgent {
11. public static void main(String args[]) throws Exception {
12. MBeanServer server = MBeanServerFactory.createMBeanServer();
13. ObjectName helloName = new ObjectName("chengang:name=HelloWorld");
14. Hello hello = new Hello();
15. server.registerMBean(hello, helloName);
16. ObjectName adapterName = new ObjectName("HelloAgent:name=htmladapter,port=8082");
17. HtmlAdaptorServer adapter = new HtmlAdaptorServer();
18. server.registerMBean(adapter, adapterName);
19. adapter.start();
20. System.out.println("start.....");
22. // Create an RMI connector and start it
23. JMXServiceURL url = new JMXServiceURL("service:jmx:rmi:///jndi/rmi://localhost:9999/server");
24. JMXConnectorServer cs = JMXConnectorServerFactory.newJMXConnectorServer(url, null, server);
25. cs.start();
26. System.out.println("rmi　start.....");
27. }
28. }

2、在Dos运行一个命令：rmiregistry 9999

3、运行HelloAgent，然后再在dos下运行命令jconsole

输入service:jmx:rmi:///jndi/rmi://localhost:9999/server

三、总结

　　连接MBeanServer的方式除了Html、JConsole，还有一些第三方的客户端，比较有名的是MC4j，通过这些客户端我们可以很容易去访问MBean。这也就是我们为什么要用JMX的其中一个原因：试想如果我自己搞一套标准，势必要自己开发一个客户端，那会是一个不小的工作量。

## 8、编写程序来连接MBean

前面用Html、jconsole等方法连接上了MBeanServer，并能够通过这些界面来操纵MBean。但有时我们需要不借助这些客户端，而是在自己的程序来操纵这些MBean，这就要求我们知道如何在代码里连接MBean。

　　基于上一篇为jconsole而修改的例子，给出一个示例的客户端程序，基本的操作都有了：

java 代码

1. import java.util.Iterator;
2. import java.util.Set;
3. import javax.management.Attribute;
4. import javax.management.MBeanInfo;
5. import javax.management.MBeanServerConnection;
6. import javax.management.MBeanServerInvocationHandler;
7. import javax.management.ObjectInstance;
8. import javax.management.ObjectName;
9. import javax.management.remote.JMXConnector;
10. import javax.management.remote.JMXConnectorFactory;
11. import javax.management.remote.JMXServiceURL;
12. public class Client {
13. public static void main(String[] args) throws Exception {
14. JMXServiceURL url = new JMXServiceURL("service:jmx:rmi:///jndi/rmi://localhost:9999/server");
15. JMXConnector jmxc = JMXConnectorFactory.connect(url, null);
16. MBeanServerConnection mbsc = jmxc.getMBeanServerConnection();
17. ObjectName mbeanName = new ObjectName("chengang:name=HelloWorld");
18. // 把所有Domain都打印出来
19. System.out.println("Domains:---------------");
20. String domains[] = mbsc.getDomains();
21. for (int i = 0; i < domains.length; i++) {
22. System.out.println("\tDomain[" + i + "] = " + domains[i]);
23. }
24. // MBean的总数
25. System.out.println("MBean count = " + mbsc.getMBeanCount());
26. // 对name属性的操作（属性名的第一个字母要大写）
27. mbsc.setAttribute(mbeanName, new Attribute("Name", "PANDA"));// 设值
28. System.out.println("Name = " + mbsc.getAttribute(mbeanName, "Name"));// 取值
29. // 得到proxy代理后直接调用的方式
30. HelloMBean proxy = (HelloMBean) MBeanServerInvocationHandler.newProxyInstance(mbsc, mbeanName, HelloMBean.class, false);
31. proxy.printHello();
32. proxy.printHello("Raymend");
33. // 远程调用的方式
34. mbsc.invoke(mbeanName, "printHello", null, null);
35. mbsc.invoke(mbeanName, "printHello", new Object[] { "熊猫烧香" }, new String[] { String.class.getName() });
36. // 得mbean的信息
37. MBeanInfo info = mbsc.getMBeanInfo(mbeanName);
38. System.out.println("Hello Class: " + info.getClassName());
39. System.out.println("Hello Attriber：" + info.getAttributes()[0].getName());
40. System.out.println("Hello Operation：" + info.getOperations()[0].getName());
41. // 得到所有的MBean的ObjectName
42. System.out.println("all ObjectName：---------------");
43. Set set = mbsc.queryMBeans(null, null);
44. for (Iterator it = set.iterator(); it.hasNext();) {
45. ObjectInstance oi = (ObjectInstance) it.next();
46. System.out.println("\t" + oi.getObjectName());
47. }
48. // 注销
49. // mbsc.unregisterMBean(mbeanName);
50. // 关闭MBeanServer连接
51. jmxc.close();
52. }
53. }

运行后的效果如下：

Domains:---------------

 Domain[0] = HelloAgent

 Domain[1] = JMImplementation

 Domain[2] = chengang

MBean count = 3

Name = Chen.Gang

Hello Class: mbean.connector.Hello

Hello Attriber：Name

Hello Operation：printHello

all ObjectName：---------------

 chengang:name=HelloWorld

 JMImplementation:type=MBeanServerDelegate

 HelloAgent:name=htmladapter,port=8082

**它有两个Console输出，这里是另一个**

Hello World, PANDA

Hello ,  熊猫

Hello World, PANDA

Hello , 熊猫烧香

## 9、基于JBoss来写MBean

　前面都是用JDK自带的JMX实现来写的MBean，JMX的实现不独SUN一家，JBOSS也有自己的JMX实现。如果你使用JBOSS来做WEB服务 器，那么基于JBOSS的实现来写MBean，是一个不错的选择。象我们公司就是用JBOSS的，因此所有MBean都是基于JBoss来写的。基于 JBoss的MBean和基于SUN的MBean有什么不同吗？有一些不同之外，但绝大部份都一样。

　　下面是我最早发的一篇关于JMX的文章，是我对公司所做项目的笔记，它上面的JMX例子就是基于JBOSS的。博客搬了几次家，文章删的删丢的丢，但这篇文章还保留着，简单修改一下，再帖上吧。

### 一、  HelloWorld实例

**1、准备工作**

　　JBOSS实现了JMX规范，这个实例是基于JBOSS来实现的。请先去下载一个JBOSS，我是jboss-3.2.6，下载地址：<http://www.jboss.com/downloads/index#as>。这个实例需要JBOSS的两个JAR包的支持：jboss-system-3.2.6.jar、jboss-jmx-3.2.6.jar，如果你和我一样用Eclipse来开发（推荐），那么把这个两个包加入到项目的库引用中（加入到库引用的方法参考前面两章）。

**2、程序代码**

　　假设我们有一个叫message的属性要需要经常进行改动配置的，那么我们就把它写成一个MBean。

1、HelloWorldServiceMBean接口

　　在写MBean之前，我们先需要写一个MBean接口，接口里的方法都是属性的set/get方法。这个接口必须继承接口ServiceMBean。

java 代码

1. import org.jboss.system.ServiceMBean;
2. public interface HelloWorldServiceMBean extends ServiceMBean {
3. String getMessage();
4. void setMessage(String message);
5. }

2、HelloWorldService实现类

　　然后写出HelloWorldServiceMBean接口的实现类HelloWorldService，这个实现类还必须继承ServiceMBeanSupport类。这种类再简单不过了，就是属性和相应的set/get方法，EJB中叫实体类、Hibernate中叫POJO。

java 代码

1. import org.jboss.system.ServiceMBeanSupport;
2. public class HelloWorldService extends ServiceMBeanSupport implements HelloWorldServiceMBean {
3. private String message;
4. public String getMessage() {
5. System.out.println("getMessage()=" + message);
6. return message;
7. }


11. public void setMessage(String message) {
12. System.out.println("setMessage(" + message + ")");
13. this.message = message;
14. }
15. }

3、配置文件jboss-service.xml

xml 代码

1. <?xml version="1.0" encoding="UTF-8"?>
2. <server>
3. <mbean code="example.mbean.HelloWorldService" name="www.chengang.com.cn:service=HelloWorld">
4. <attribute name="Message">Hello World</attribute>
5. </mbean>
6. </server>

说明：

l code项指向MBean的实现类HelloWorldService

l name项是一个名称，格式一般是：[说明性文字]:service=[类名]

l attribute是为属性设置初始值，这样当JBOSS一加载HelloWorldService类时，message属性就有了一个初始值Hello World。注意Message的第一个字母必须是大写。

### 二、将实例部署到JBOSS

　　在jboss-3.2.6\server\default\deploy目录下创建一个hello.sar目录，然后创建如下目录文件结构：

hello.sar

|----example

|            |----mbean

|                    |----HelloWorldService.class (注意：是\*.class，不是\*.java)

|                    |----HelloWorldServiceMBean.class

|----META-INF

             |----jboss-service.xml

其他说明：

l 也可以将hello.sar目录用zip格式压缩成一个同名的hello.sar文件，放到jboss-3.2.6\server\default\deploy目录下。

l JBOSS支持热部署，也就是说你在布置这个目录时并不需要重启JBOSS。

### 三、MBean的效果 打开网址：<http://127.0.0.1:8080/jmx-console/>

后单击“service=HelloWorld”项打开详细页面

将“HelloWorld”改成“Hello World，ChenGang”，再单击“Apply Changes”应用修改

### 四、其他类如何使用Messag属性

　　现在我们可以通过一个自动提供的WEB页面来设置Message属性了，接下来的问题是：“在其他类中应该如何得到Message的属性值”。MBean在JBoss是只保留一个MBean的实例（单例模式？），也就是说问题转成我们如何去取得这个唯一实例。例程如下：

1、创建一个使用到Message属性的类

java 代码

1. import org.jboss.mx.util.MBeanProxyExt;
2. import org.jboss.mx.util.ObjectNameFactory;
3. public class Client {
4. public void go() {
5. HelloWorldServiceMBean mbean = (HelloWorldServiceMBean) MBeanProxyExt.create(HelloWorldServiceMBean.class, ObjectNameFactory.create(www.chengang.com.cn:service=HelloWorld));
6. String msg = mbean.getMessage();
7. System.out.println("Client.go()=" + msg);
8. }
9. }

注意：go方法里是三句。第一句比较长，它是根据jboss-service.xml文件中设置的MBean名称，来取得此MBean在JBOSS中的实例。

2、在Mbean中加一个相应的调用Client.go的方法

　在HelloWorldServiceMBean接口中加入一句：

java 代码

1. void callGo();

在HelloWorldService类中加入现实方法：

java 代码

1. public void callGo() {
2. new Client().go();
3. }

3、更新布署

   将三个类的class文件：Clien.class、HelloWorldServiceMBean.class、HelloWorldService.class，更新到JBOSS的hello.sar\example\mbean目录下。然后重启JBOSS。

4、查看效果

   打开JBOSS提供的MBean设置页面，发现多了一个callGo。

   单击callGo项后的invoke按钮，得到DOS输出

### 五、其他说明

l 本实例仅演示了一个Message属性，你当然可以在HelloWorldService中加入更多属性，别忘了在HelloWorldServiceMBean接口也加入相应的set/get方法。

l 本实例的message属性是String类型的，但JMX也支持其他的类型，甚至是XML信息。对于jboss-service.xml中的XML信息，这时属性类型要求是org.w3c.dom.Element，JMX将它封装成了一个XML的DOM对象。

l 回顾一下，JMX也是将配置信息写在了一个文件（jboss-service.xml文件）里嘛，相对于将配置文件写到\*.properties文件的方式，它似乎也没什么 新鲜的地方。但通过本章实例我们可以看到JMX的一些好处：我们不用写代码去读配置文件的信息，而且JMX支持的属性类型是多种多样的（如上面说的 org.w3c.dom.Element）。更重要的是JMX还提供了一整套的属性之前互相访问、互相调用的功能，这个HelloWorld实例所反映的 只是冰山一角而已。