参考：<https://my.oschina.net/xpbug/blog/221547>

What and Why JMX

JMX的全称为Java Management Extensions（java管理扩展）. 顾名思义，是管理Java的一种扩展。这种机制可以方便的管理正在运行中的Java程序。常用于管理线程，内存，日志Level，服务重启，系统环境等。

试想，一个正在运行中的程序，我们如果想改变程序中的一些属性，可以通过什么方法呢？可能有这么几个方法：

（me：基本的就是程序开放几个接口，供外部在程序运行期间改变程序的运行参数，而这些运行的参数可以放在类似配置文件中，也可以存储在内存中以便在程序运行期间更改）

* 对于服务器式的程序，可以制作管理页面，通过HTTP与servlet来更改服务器端程序的属性。
* 对于服务器式的程序，还可以通过SOAP（面向服务架构）方式。但这需要程序开启了SOAP端的服务。
* 可以使用RMI远程调用。但这需要设计开启RMI服务（RMI只适用于java，其他的语言可以使用RPC）。
* 如果是SWT或Swing的程序，则可以通过设计UI管理界面，使用户可以和程序内部交互。
* 还有一种方式，是将可改变的属性放入配置文件XML，properties或数据库，程序轮询配置文件，以求获取最新的配置。

上面几个方法都是常见，但却无法通用的。所谓通用，是指解决方案符合一个标准，使得任何符合此标准的工具都能解析针对此标准的方案实现。这样A公司设计的方案，B公司可以根据标准来解析。JMX就是Java管理标准。

JMX的构成

JMX由三部分组成（MBean,mbean server,remote client）：

1. 程序端的Instrumentation, 我把它翻译成可操作的仪器。这部分就是指的MBean（me：应该是Management Bean的缩写）. MBean类似于JavaBean。最常用的MBean则是Standard MBean和MXBean.
2. 程序端的JMX agent. 这部分指的是MBean Server. MBean Server则是启动与JVM内的基于各种协议的适配器。用于接收客户端的调遣，然后调用相应的MBeans（me：一般一个程序包含多个MBean，即多个操作仪器）.
3. 客户端的Remote Management. 这部分则是面向用户的程序。此程序则是MBeans在用户前投影，用户操作这些投影，可以反映到程序端的MBean中去。这内部的原理则是client通过某种协议调用agent操控MBeans.

JMX agent与Remote Management之间是通过协议链接的，这协议可能包含：

* HTTP
* SNMP
* RMI
* IIOP

JMX agent中有针对上面协议的各种适配器(就是接口)。可以解析通过相应协议传输过来的数据。Remote Management client则可以用现成的工具，如JConsole, 也可以自己书写java code。

接下来，我们一步一步，通过代码示例来熟悉JMX各种特性。

受监管的程序

JMX是用于管理java程序的，为了试验，我们首先需要写一个小程序Echo。然后加入JMX对此程序进行监管。这个程序就是每隔10秒钟，输出一个预先定义好的Message。

首先定义Message类。

public class Message {

    private String title, body, by;

    public Message() {

        title="none";

        body="none";

        by="none";

    }

    public String getTitle() {

        return title;

    }

    public void setTitle(String title) {

        this.title = title;

    }

    public String getBody() {

        return body;

    }

    public void setBody(String body) {

        this.body = body;

    }

    public String getBy() {

        return by;

    }

    public void setBy(String by) {

        this.by = by;

    }

    public void echo() {

        System.out.println("<"+title+">");

        System.out.println(body);

        System.out.println("by " + by);

    }

}

定义Echo类

public class Echo {

    public static Message msg = new Message();

    public static boolean running=true;

    public static boolean pause=false;

    public static void main(String[] args) {

        // 开启JMX Agent。如果不需要JMX,只是单独运行程序，请屏蔽掉下面这行代码。

        new MessageEngineAgent().start();

        while(running) {

            try {

                Thread.sleep(10000);

            } catch (InterruptedException e) {

                e.printStackTrace();

            }

            if (!pause) msg.echo();

        }

    }

}

执行Echo，得到每过10秒钟，则会输出一个消息：

<none>

none

by none

MBean

接下来，开始设计管理程序的MBean. 在设计MBean之前，必须要先了解MBean都包括哪几种。MBean包含：

1. Standard MBean
2. Dynamic MBean
3. Open MBean
4. Model MBean
5. MXBean

最常用最简单的两个就是Standard MBean与MXBean. 首先搞清楚，MBean和MXBean的区别是什么。

Standard MBean与MXBean的区别

这里有一些细节，列出了两种的区别<http://docs.oracle.com/javase/7/docs/api/javax/management/MXBean.html> 。它们最根本的区别是，MXBean在Agent（Manager server）与Client之间会将自定义的Java类型转化为Java Open Type. 这样的好处是Client无需获取MXBean的接口程序，便可访问和操作MXBean的投影。如果使用MBean, client则必须先将MBean的接口程序放到classpath中，否则无法解析MBean中自定义类型。

Me:两种方法都类似于远程调用，都需要在服务端和客户端有相应/同的操作接口，只是MXBean会将自定义类型转换为一个公有的类型。

Client🡪agent🡪MBean

基于上述原因，我将使用MXBean做为例子。实际上，JVM自带的几乎全是MXBean。

实现

定义MXBean的接口，注意命名规则，必须以MXBean结尾。

public interface MessageEngineMXBean {

    //结束程序。

    public void stop();

    //查看程序是否暂停。

    public boolean isPaused();

    //暂停程序或者继续程序。

    public void pause(boolean pause);

    public Message getMessage();

    //修改message

    public void changeMessage(Message m);

}

实现部分。

public class MessageEngine implements MessageEngineMXBean {

    private final Message message = Echo.msg;

    @Override

    public void stop() {

        Echo.running = false;

    }

    @Override

    public boolean isPaused() {

        return Echo.pause;

    }

    @Override

    public void pause(boolean pause) {

        Echo.pause = pause;

    }

    @Override

    public Message getMessage() {

        return this.message;

    }

    @Override

    public void changeMessage(Message m) {

        this.message.setBody(m.getBody());

        this.message.setTitle(m.getTitle());

        this.message.setBy(m.getBy());

    }

}

Notification

在JMX中，还有一个重要的概念是Notification。构成Notification的几个接口是：

1. NotificationEmitter, 只要实现此接口，就可以发出Notification和订阅Notification. 类NotificationBroadcasterSupport则实现了NotificationEmitter.
2. NotificationListener, 实现此接口的可以订阅JMX的Notification。
3. Notification, 消息本身。

修改MessageEngine, 使它在pause的时候发送通知给订阅者。我把修改的部分贴上。

public class MessageEngine extends NotificationBroadcasterSupport implements MessageEngineMXBean {

    private long sequenceNumber = 1;

    ... ...

    ... ...

    public MessageEngine() {

        addNotificationListener(new NotificationListener() {

            @Override

            public void handleNotification(Notification notification, Object handback) {

                System.out.println("\*\*\* Handling new notification \*\*\*");

                System.out.println("Message: " + notification.getMessage());

                System.out.println("Seq: " + notification.getSequenceNumber());

                System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

            }

        }, null, null);

    }

    ... ...

    ... ...

    @Override

    public void pause(boolean pause) {

        Notification n = new AttributeChangeNotification(this,

                sequenceNumber++, System.currentTimeMillis(),

                "Pause changed", "Paused", "boolean",

                Echo.pause, pause);

        Echo.pause = pause;

        this.sendNotification(n);

    }

    ... ...

    ... ...

    // 规定可以发送的Notification Type，不在Type list中的Notification不会被发送。

    @Override

    public MBeanNotificationInfo[] getNotificationInfo() {

        String[] types = new String[]{

            AttributeChangeNotification.ATTRIBUTE\_CHANGE

        };

        String name = AttributeChangeNotification.class.getName();

        String description = "An attribute of this MBean has changed";

        MBeanNotificationInfo info =

                new MBeanNotificationInfo(types, name, description);

        return new MBeanNotificationInfo[]{info};

    }

}

Client端如何使用Notification，可以查看后面的Client一节。

JMX Agent

如果说Agent只是被Local使用，比如本地的JConsole，只需要开启MBeanServer,并注册MBean即可。不需要配置协议适配器。但如果需要远程管理，比如远程的JConsole或者自定义的管理器，则还需要配置两者相互打交道的协议适配器。

public class MessageEngineAgent {

    public void start() {

        MBeanServer mbs = ManagementFactory.getPlatformMBeanServer();

        try {

            ObjectName mxbeanName = new ObjectName("com.example:type=MessageEngine");

            MessageEngineMXBean mxbean = new MessageEngine();

            mbs.registerMBean(mxbean, mxbeanName);

        } catch (Exception e) {

            e.printStackTrace();

        }

    }

}

因为java默认自带的了JMX RMI的连接器。所以，只需要在启动java程序的时候带上运行参数，就可以开启Agent的RMI协议的连接器。

java -Dcom.sun.management.jmxremote.port = 9999  \

     -Dcom.sun.management.jmxremote.authenticate = false \

     -Dcom.sun.management.jmxremote.ssl = false \

     jmx.Echo

认证与授权

JMX的认证与授权是非常必要的，我们不可能允许任何client都能连接我们的Server。JMX的认证和授权可以复杂的使用LDAP, SSL。也可以使用最简单的文件存储用户信息方式。本文作为启蒙，只给出最简单的认证方式。

在java启动的时候，添加运行参数：

java -Dcom.sun.management.jmxremote.port = 9999  \

     -Dcom.sun.management.jmxremote.authenticate = true \

     -Dcom.sun.management.jmxremote.password.file = pathTo/my.password \

     -Dcom.sun.management.jmxremote.access.file = pathTo/my.access \

     -Dcom.sun.management.jmxremote.ssl = false \

     jmx.Echo

my.password里面定义了用户名和密码：

user1 password1

user2 password2

my.access里面定义了用户授权信息：

user1 readOnly

user2 readWrite \

      create jmx.\*,javax.management.timer.\* \

      unregister

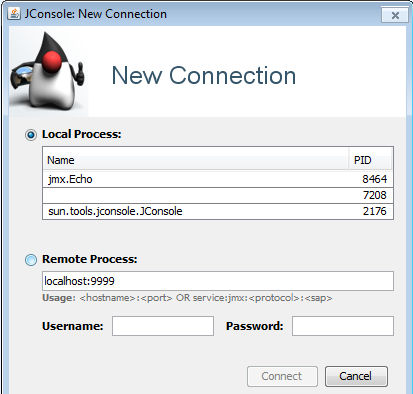
更详细的内容可以从这里找到： <http://docs.oracle.com/javase/7/docs/technotes/guides/management/agent.html> 。

现在可以启动程序了。启动以后，我们使用下面的Client来连接我们写的JMX Agent.

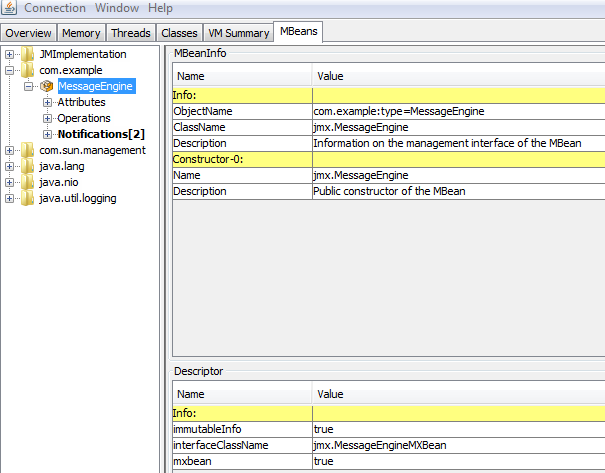
JMX Client

JConsole

JDK提供了一个工具在jdk/bin目录下面，这就是JConsole。使用JConsole可以远程或本地连接JMX agent。如下图所以：



无论是远程还是本地，连接进去所看到的都一样。进去MBeans面板以后，找到MessageEngine。MessageEngine下面有Attributes, Operations和Notification。可以浏览MessageEngine中的Attributes并更改那些可写的属性。也可以执行Operations下面的stop, pause方法。此外，必须订阅Notifications才能收到消息。



JConsole有缺点，它只能对MXBean中的主要基本类型做修改，但不能修改复杂类型。

Custom Client

我们也可以用java写client调用Agent。

public class Client implements NotificationListener {

    public static void main(String[] args) {

        try {

            new Client().start();

        } catch (Exception e) {

            e.printStackTrace();

        }

    }

    public void start() throws Exception {

        // 如果agent不做配置的话，默认jndi path为jmxrmi

        JMXServiceURL url = new JMXServiceURL("service:jmx:rmi://localhost/jndi/rmi://localhost:9999/jmxrmi");

        JMXConnector jmxc = JMXConnectorFactory.connect(url, null);

        MBeanServerConnection server = jmxc.getMBeanServerConnection();

        ObjectName mbeanName = new ObjectName("com.example:type=MessageEngine");

        // 订阅Notification

        server.addNotificationListener(mbeanName, this, null, null);

        // 访问paused属性。

        boolean paused = (Boolean)server.getAttribute(mbeanName, "Paused");

        System.out.println(paused);

        if (!paused) {

            server.invoke(mbeanName, "pause", new Object[]{true}, new String[]{"boolean"});

        }

        // 构建一个jmx.Message类型实例。

        CompositeType msgType = new CompositeType("jmx.Message", "Message Class Name",

                  new String[]{"title","body", "by"},

                  new String[]{"title","body", "by"},

                  new OpenType[]{SimpleType.STRING,SimpleType.STRING,SimpleType.STRING});

        CompositeData msgData = new CompositeDataSupport(msgType,

                new String[]{"title","body","by"},

                new Object[]{"Hello", "This is a new message.", "xpbug"});

        // 调用changeMessage方法

        server.invoke(mbeanName, "changeMessage", new Object[]{msgData}, new String[]{CompositeData.class.getName()});

        server.invoke(mbeanName, "pause", new Object[]{false}, new String[]{"boolean"});

        // 访问修改后的Message属性。

        msgData = (CompositeData)server.getAttribute(mbeanName, "Message");

        System.out.println("The message changes to:");

        System.out.println(msgData.get("title"));

        System.out.println(msgData.get("body"));

        System.out.println(msgData.get("by"));

        Thread.sleep(1000\*10);

    }

    @Override

    public void handleNotification(Notification notification, Object handback) {

        System.out.println("\*\*\* Handling new notification \*\*\*");

        System.out.println("Message: " + notification.getMessage());

        System.out.println("Seq: " + notification.getSequenceNumber());

        System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

    }

}

运行一下client,看看都发生了什么。

源码下载

<http://pan.baidu.com/s/1sjLKewX>