

JDK5 新版 RMI 编程指南

沈东良(良少) shendl_s@hotmail.com

2007-8-2

<http://blog.csdn.net/shendl/>

前言

我前一段时间需要为我的一个 Java 程序提供远程访问接口,供其他 Java 程序使用。Java 程序可以使用很多种远程访问技术实现这一需求。由于我的远程客户端是 java 程序,因此,我决定使用 RMI 这种远程访问技术。RMI 是 java 平台上最快的远程访问技术。

Spring 框架为包括 RMI 在内的各种远程访问技术提供了很好的工具类,能够使我们方便的公布 RMI 接口和访问 RMI 远程对象。

但是,我的那个 Java 程序并没有使用 Spring 框架。

因此,我研究了怎样在一般的 java 程序中使用 RMI 技术。

我寻找了一些 RMI 资料。根据那些 RMI 资料,构建一个 RMI 服务器需要写大量的代码。这太离谱了!一个小小的需求,竟然需要这么大的精力。

后来,我看了 javadoc,才发现网上的那些 RMI 资料都已经过时了。JDK5 中,RMI 技术已经得到了重大更新。现在使用 JDK 提供的 RMI 类,可以相当简单的发布 RMI 服务!

正文

RMI 简介

RMI (Remote Method Invocation), 远程方法调用, 是一种远程调用 java 方法的技术。是最早开发出来的 java 远程访问技术, 也是 EJB 等远程访问技术的基础。

过去, 使用 RMI 技术共有 6 个步骤要走: (1)定义和实现远端接口中的参数 (2) 定义和实现远端接口 (3) 编写服务端代码 (4) 编写客户端代码 (5) 生成 stub 和 skeltion ,并将 stub 打包到客户端 jar 中, 将 skeltion 打包到服务端 jar 中 (6) 启动 rmiregistry , 并将服务注册到 rmiregistry 中, 然后运行代码。

这还需要执行一些命令行程序, 非常繁杂! 这与 RMI 这样一种简单的技术非常不相称!

好在 JDK5 中, RMI 得到了重大更新。RMI 已经成为一种真正轻量级的简单技术!

JDK5 中, 使用了动态代理技术, 实现了动态生成 stub 和 skeltion 类, 从而省却了相当多的繁琐工作。RMI 的创建和发布已经变得非常简单!

建立 RMI 服务器和注册表

1,Registry 接口

java.rmi.registry

接口 Registry

所有超级接口:

[Remote](#)

```
public interface Registry
extends Remote
```

Registry 是简单远程对象注册表的一个远程接口，它提供存储和获取绑定了任意字符串名称的远程对象引用的方法。bind、unbind 和 rebind 方法用于改变注册表中的名称绑定，lookup 和 list 方法用于查询当前的名称绑定。

Registry 接口，是 RMI 对象的注册表，客户端通过这个注册表查询和得到 RMI 对象。

2, LocateRegistry 类

java.rmi.registry

类 LocateRegistry

[java.lang.Object](#)

```
java.rmi.registry.LocateRegistry
```

```
public final class LocateRegistry
extends Object
```

LocateRegistry 用于获得对特定主机（包括本地主机）上引导远程对象注册表的引用，或用于创建一个接受对特定端口调用的远程对象注册表。

static Registry	createRegistry (int port) 创建并导出接受指定 port 请求的本地主机上的 Registry 实例。
---------------------------------	--

LocateRegistry 类的 createRegistry()方法，创建并得到对远程 RMI 对象注册表的引用。

现在，我们就成功创建了一个 RMI 服务器和对象注册表。

然后，通过调用 Registry 接口的方法：

void	rebind (String name, Remote obj) 用提供的远程引用替换此注册表中指定的 name 绑定。
------	--

我们能够把 RMI 对象发布到 RMI 服务器上，并存放到注册表中，能够被 RMI 客户端发现并访问。

至此，RMI 服务器就建立好了！

编写 RMI 对象

Registry 接口的方法：

void	rebind (String name, Remote obj) 用提供的远程引用替换此注册表中指定的 name 绑定。
------	--

这表明，RMI 对象必须是实现了 Remote 接口的对象。实际上，任何 java 上的可远程访问对象都需要实现 Remote 接口！

但是，仅仅实现了 Remote 接口的对象还不能作为 RMI 对象被发布。

UnicastRemoteObject 类

java.rmi.server

类 UnicastRemoteObject

[java.lang.Object](#)

[java.rmi.server.RemoteObject](#)

[java.rmi.server.RemoteServer](#)

java.rmi.server.UnicastRemoteObject

所有已实现的接口：

[Serializable](#), [Remote](#)

直接已知子类:

[ActivationGroup](#)

```
public class UnicastRemoteObject
extends RemoteServer
```

用于导出带 JRMP 的远程对象和获得与该远程对象通信的 stub。

对于下面的构造方法和静态 `exportObject` 方法，正在导出的远程对象的 stub 按以下方式获得：

- 如果使用 [UnicastRemoteObject.exportObject\(Remote\)](#) 方法导出该远程对象，则加载 stub 类（通常使用 `rmic` 工具从远程对象的类预生成）并按以下方式构造 stub 类的实例。
 - “根类”按以下情形确定：如果远程对象的类直接实现扩展 [Remote](#) 的接口，则远程对象的类为根类；否则，根类为直接实现扩展 `Remote` 接口的远程对象类的最具派生能力的超类。
 - 要加载的 stub 类的名称通过连接带有后缀 “_Stub” 的根类的二进制名称确定。
 - 按使用根类的类加载器的名称加载 stub 类。该 stub 类必须扩展 [RemoteStub](#) 并且必须有公共构造方法，该构造方法有一个属于类型 [RemoteRef](#) 的参数。
- 最后，用 [RemoteRef](#) 构造 stub 类的实例。

使用 `UnicastRemoteObject` 类的方法：

<code>static Remote</code>	<code>exportObject(Remote obj, int port)</code> 使用提供的特定端口导出远程对象，以便能够接收传入的调用。
--	---

可以把实现了 `Remote` 接口的类生成为 RMI 对象。可以发布到 RMI 注册表上！

生成 RMI 类的存根类

使用 `UnicastRemoteObject` 类的方法：

<code>static Remote</code>	<code>exportObject(Remote obj, int port)</code> 使用提供的特定端口导出远程对象，以便能够接收传入的调用。
--	---

使用这个方法返回的 `Remote` 对象，可以被发布到 RMI 的注册表上。使用动态代理，它能够自动生成 stub 类。这样，只要使用 `UnicastRemoteObject.exportObject(Remote obj,`

int port)方法创建RMI注册表上的RMI对象，就不需要我们再关心stub和skelton类的生成问题。

注意，

static RemoteStub	exportObject (Remote obj) 使用匿名端口导出远程对象，以便能够接收传入的调用。
-----------------------------------	---

这个方法创建的 RMI 对象，客户端不能自动生成 stub 类！请不要使用这个方法。

请注意，

static Remote	exportObject (Remote obj, int port) 使用提供的特定端口导出远程对象，以便能够接收传入的调用。
-------------------------------	--

我们使用这个方法时，应该使用 0 作为第二个参数的值。否则可能会出错！

RMI 最佳实践

看了上面的文字，是不是觉得还是有些复杂呢？接下来，我们看看实际怎样编写 RMI 程序，你将发现 RMI 编程确实是非常非常简单！

实现 Remote 接口

RMI 发布的类，必须实现 Remote 接口。我们可以让我们的 RMI 类实现的接口 extend Remote 接口。

这就是 Remote 接口。

```
public interface Remote {}
```

可以看到，它实际上是空的，什么都没有。它仅仅是一个标记，表示 Remote 接口用于标识其方法可以从非本地虚拟机上调用的接口。任何远程对象都必须直接或间接实现此接口。只有在“远程接口”（扩展 java.rmi.Remote 的接口）中指定的这些方法才可远程使用。

这种标记接口，现在实际上可以用“标注”来达到同样的目的。当初标注没有引入 Java 中，因此就有了这样的空接口！

另外，所有能够远程访问的方法，都必须抛出 RemoteException。这是因为，远程调用可能会碰到一些如网络中断这样的错误。

可发布为 RMI 的类

可发布为 RMI 的类，它必须有参数为空的构造器，构造器必须声明为可抛出 `RemoteException`。

建议构造其中这样写：

```
UnicastRemoteObject.exportObject(this, 0);
```

也就是说，构造器构造的是一个可以动态生成存根的 RMI 对象。

当然，你也可以不这么写，但是那样的话就要在注册 RMI 对象时多写一些代码了！

另外，所有远程调用方法的参数，必须实现 `Serializable` 接口。

发布 RMI 对象

下面是我的发布 RMI 对象的代码：

//这是 RMI 服务器使用的端口，我使用了配置文件

```
String rMIPort = Config.getInstance().get("RMIPort");
```

//这样就在指定的端口创建并启动了 RMI 服务器。否则，你需要在命令行中输入：

```
//start rmiregistry
```

//手工启动 RMI 服务器

```
Registry registry = LocateRegistry.createRegistry(new Integer(rMIPort)
    .intValue());
```

//我在配置文件中配置了一些 RMI 类，每一个包括注册表的名字和 RMI 类的全名

```
Map<String, String> rmis = Config.getInstance().getRmis();
```

```
Set<Map.Entry<String, String>> entrys = rmis.entrySet();
```

//用叠代器全部拿到，然后一个个放到 RMI 注册表中，公布出来！

```
Iterator it = entrys.iterator();
```

```
while (it.hasNext()) {
```

```
    Map.Entry<String, String> tmp = (Map.Entry<String, String>) it
        .next();
```

```
    try {
```

//创建 RMI 对象，可以动态生成存根

```
        Remote stub = (Remote) Class.forName(tmp.getValue())
            .newInstance();
```

//以指定的名字发布 RMI 对象到注册表

```
        registry.rebind(tmp.getKey(), stub);
```

```
    } catch (InstantiationException e) {
```

```
        /*
```

```
        */
```

```
        e.printStackTrace();
```

```
    } catch (IllegalAccessException e) {
```

```

        /*
        */

        e.printStackTrace();
    } catch (ClassNotFoundException e) {
        /*
        */

        e.printStackTrace();
    } catch (RemoteException e) {
        /*
        */

        e.printStackTrace();
    }
}

```

访问 RMI 对象的客户端

访问 RMI 对象的客户端也是非常非常简单的。

如：

//RMI 服务器的 IP 地址

```
String rMIHost=Config.getInstance().get("RMIHost");
```

//RMI 服务器的端口

```
String rMIPort=Config.getInstance().get("RMIPort");
```

//得到RMI服务器注册表的引用

```
Registry registry = LocateRegistry.getRegistry(rMIHost, new
Integer(rMIPort).intValue());
```

//查找并得到RMI对象的实例。

```
IServerREQTerminalConfigService stub = (IServerREQTerminalConfigService)
registry.lookup("ServerREQTerminalConfigService");
```

//调用远程RMI服务器上的对象的方法！

```
stub.rmiRemote();
```

就这么简单！

结语

在 JDK5 发布之前，使用 RMI 是一件非常繁琐的事情。甚至很多程序员把 EJB 当作对 RMI 的封装，当作 RMI 的简化版！

而今，RMI 已经得到了巨大的改进。作为 Java 平台上性能最好远程访问技术，如今也是最简单的远程访问技术，RMI 理应得到更广泛的应用。