# 第12讲 RMI程序设计（一）

**教学与实践目的**：学会RMI软件架构的程序设计技术。

## 一、相关概念

### 1. RPC（Remote Procedure Call，远程过程调用）

RPC是一种程序设计模式（框架、协议），RPC的主要目标是让构建分布式计算（应用）更容易、透明，在提供强大的远程调用能力时不损失本地调用的语义简洁性。为实现该目标，RPC需提供一种透明调用机制，让使用者不必显式的区分本地调用和远程调用。它通过网络从远程计算机程序上请求服务，而不需要了解底层网络技术的协议。

说白了，就是在服务器端定义远程对象并实现这些对象中的远程方法，然后对外宣称具有这些对象。客户端根据自己的需要可以远程调用这些方法，无需自己书写详细的实现代码。如服务端定义好购房、购车等费用的计算方法，股票的查询方法。例如：两台服务器A和B，有一个应用程序部署在A上，它想调用B上的应用提供的函数／方法，由于它们不在同一个内存空间，不能直接调用，需要通过网络来表达调用的语义和需要传输的数据。 RPC框架负责屏蔽底层的传输方式（TCP或者UDP）、序列化方式以及网络通信细节，具体来说，RPC需要做以下工作：

（1）通信：RPC 假定某些传输协议的存在，如TCP或UDP，为通信程序之间携带信息数据。RPC采用C/S架构，在客户机和服务器中建立一个TCP/UDP连接，则远程调用所交换的数据都在这个连接里传输；

（2）因为RPC的作用是让使用者调用远程函数/方法时，就像调用本地函数/方法，所以不管是本地客户还是远程服务端，需要使用一套统一的接口，然后两边分别实现自己的逻辑；

（3）序列化和反序列化：当A服务器的应用程序发起远程调用时，方法的参数要通过网络协议（如TCP）传输到B服务器，而网络协议是基于二进制的，所以A服务器内存中参数的值需要序列化为二进制后再发送给B服务器；B服务器接收到A服务器发送过来的二进制参数时，需要对其进行反序列化，将其恢复为内存能识别的表达方式，然后通过寻址找到对应的方法进行本地调用，得到其返回值，将返回值再序列化后发回A服务器上的应用，A服务器接收二进制返回值后再进行反序列化，最后将内存能识别的表达方式传给A中的应用程序。

实现RPC思想的协议/框架有很多，隐藏了很多实现细节，便于使用者使用，比如最早的CORBA、Java RMI、Dubbo、Web Service的RPC风格，Hessian，Thrift等等。

### 2. RMI（Remote Method Invocation，远程方法调用）

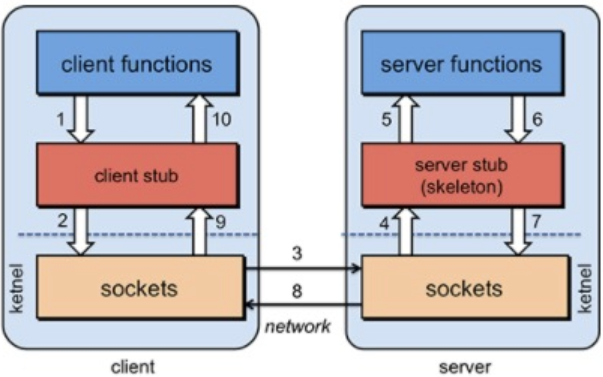
RMI是 RPC的一种面向对象编程方式的Java实现，底层使用TCP协议，它的目的在于对开发人员屏蔽横跨不同JVM和网络连接等细节，使得分布在不同JVM上的对象像是存在于一个统一的JVM中一样，客户端调用远程服务方法，使用起来就好像调用本地方法。有一些用Java实现的RPC框架底层也使用了RMI技术。

（1）实现原理：

RMI为客户端B的对象和远程服务端A的对象分别提供了客户辅助对象（客户端stub，桩）和远程服务辅助对象（服务端stub，也称为skeleton，骨架），来帮助本地客户端的对象真正和远程服务对象进行沟通；

RMI在客户端B为stub创建和远程服务对象相同的方法，客户调用stub上的方法，仿佛stub就是真正的服务，stub再负责为我们转发这些请求。 stub会联系服务器A，传送方法调用信息（例如方法名称、变量等），然后等待服务器A对应远程方法的返回；

在服务器端A，skeleton通过Socket连接从stub中接收请求，将调用的信息解包，然后调用真正服务对象上的真正方法。所以，对于服务对象来说，调用是本地的，来自skeleton，而不是远程客户。 skeleton从服务中得到返回值，将它打包，然后运回到客户端辅助对象stub（通过网络Socket的输出流），stub对信息解包，最后将返回值交给客户对象。 RMI的好处在于我们不必亲自写任何网络或I/O代码（Java 1.5后，也不需要用户来手动生成stub和skeleton，完全被隐藏了细节），客户程序感觉不到调用远程方法（即真正的服务所在），其调用过程就好像是在调用自己的本地对象的方法。整个过程示意图如图12.1所示。



A

B

图12.1 RMI示意图

（2）注意：上面的示意图是A为B提供远程服务。其实远程对象不仅可以位于服务器端，也可以位于客户端。客户端也可以为服务端提供远程方法调用，这种调用过程被称为回调。在这一刻， B和A的角色就好像是临时互换了。

## 二、RMI程序的实现

我们按以下步骤来创建RMI规范的程序：

### 1. 创建远程接口

哪些方法是客户远程可以调用的？在远程接口中定义的方法就是可以被客户端远程调用的方法。一个远程接口中可以定义多个远程方法，远程接口本身也可以定义多个。任何远程接口都要满足以下四个要求：

（1）直接或间接继承java.rmi.Remote接口，这是RMI规范的要求；

（2）客户进行远程调用，底层用到了网络和I/O，而网络通信是不可靠的，比如一旦服务器或者客户端有一方突然断开连接，或者网络出现故障，通信就会失败。RMI规范要求远程接口中定义的方法都要声明抛出RemoteException异常，在客户端进行远程方法调用时，RMI框架会把遇到的网络通信失败转换为RemoteException，客户端可以捕获这种异常，并进行相应的处理；

（3）由于远程方法的变量必须被打包并通过网络运送，必须靠序列化完成，所以远程方法的变量和返回值必须属于基本类型或实现了Serializable接口的引用类型；

（4）服务端和客户端都必须各自拥有一份完全相同的的远程接口，不只是接口名字、内部定义的远程方法完全一样，所在的包名称也要一致。例如服务端的若干远程接口定义在rmi包中，那么远程的客户端也需要有定义在rmi包中完全相关的若干远程接口。（大家想一想，客户端调用远程方法要像在调用本地方法一样，而这些方法是定义在接口中的，所以客户端中肯定有一份完全远程接口，客户端就好像是在调用其本地接口中的方法）。

按照上面的要求，**我们来创建一个远程接口：**

我们在自己的项目工程中，新建一个包，命名为rmi，然后在rmi包中创建一个远程接口HelloService：

package rmi;

......

public interface HelloService extends Remote {  
 public String echo(String msg) throws RemoteException;  
 public Date getTime() throws RemoteException;  
}

由于我们现在是在自己的机器上模拟创建服务端和客户端，所以rmi中的接口就可以看作是通信双方共同拥有的一份远程接口，如果有更多的接口，也创建在这个包中。

### 2. 创建服务端的远程接口实现类

远程接口实现类（以下都简称远程服务类）就是远程服务对象所属的类，其实现远程接口中定义的远程方法。其对象实例就是客户真正想要调用方法的对象。这个类除了必须实现远程接口，且为了成为远程服务对象，我们的对象需要某些“远程的”功能，最简单的方式是扩展java.rmi.server.UnicastRemoteObject，让超类帮我们完成这些工作（UnicastRemoteObject可以把实现了远程接口的类“导出”为远程对象，使它具有相应的服务端stub，并使它能够监听远程客户的方法调用请求，这些细节都被RMI框架很好地隐藏了），并且远程接口实现类的构造方法必须声明抛出RemoteException。

**接下来我们就创建一个远程服务类**：

在项目工程中新建chapter12包，然后在chapter12包中再创建一个server子包，在chapter12.server包中创建类HelloServerImpl：

package chapter12.server;  
......  
public class HelloServiceImpl extends UnicastRemoteObject  
 implements HelloService {  
 private String name;  
 public HelloServiceImpl() throws RemoteException {  
 }  
 public HelloServiceImpl(String name) throws RemoteException {  
 this.name = name;  
 }  
 @Override  
 public String echo(String msg) throws RemoteException {   
 System.*out*.println("服务端完成一些echo方法相关任务......");  
 return "echo: " + msg + " from " + name;  
 }  
  
 @Override  
 public Date getTime() throws RemoteException {   
 System.*out*.println("服务端完成一些getTime方法相关任务......");  
 return new Date();  
 }  
}

### 3. 创建远程服务发布程序

远程服务类创建后，如何为客户端服务呢？不是简单地写个服务发布程序，仅仅实例化远程服务类就完事了，还要解决客户端如何寻找服务的问题。

RMI采用一种命名服务机制来使得客户程序可以找到服务器上的一个远程对象。RMI注册器会提供这种命名服务。不妨把RMI注册器比作日常生活中的114电话查询系统，那些希望对外公开联系方式的单位先到114查询系统登记，使得114查询系统记录该单位的名字和联系方式信息。当客户想知道某个单位的联系方式时，只需向114查询系统提供单位的名字，114查询系统就会返回该单位的联系方式。

远程服务发布程序的一大任务就是向RMI注册器注册（绑定）远程对象。远程对象注册到注册器后，就可以被客户端检索到。

注意：对于有多个网卡（特别是多个虚拟网卡的机器），发布的远程服务是默认绑定在0.0.0.0地址，即任意可用的本地地址，为了避免访问端出现访问ip的错误，建议用下面的方法绑定固定ip：

System.*setProperty*("java.rmi.server.hostname",本机器的ip地址);

**我们来实现一个远程服务发布程序：**

在chapter12.server包中新建类HelloServer：

package chapter12.server;  
......  
public class HelloServer {  
 public static void main(String[] args) {  
 try {  
 *//(1)启动RMI注册器，并监听在1099端口（这是RMI的默认端口，正如HTTP的默认端口是80）*

*//对于有多个网卡的机器，建议用下面的命令指绑定固定的ip*

System.*setProperty*("java.rmi.server.hostname",本机器的ip地址);

Registry registry = LocateRegistry.*createRegistry*(1099);  
  
 *//(2)实例化远程服务对象，如果有多个远程接口，只实例化自己实现的接口（为什么可能有没有实例化的接口？）* HelloService helloService = new HelloServiceImpl("李四的远程服务");  
  
 *//(3)用助记符来注册发布远程服务对象,助记符建议和远程服务接口命名相同，这样更好起到”助记"效果* registry.rebind("HelloService",helloService);  
 *//也可以用另外一种方式进行注册发布，建议用上面的方式  
 //Naming.rebind("HelloService",helloService);* System.*out*.println("发布了一个HelloService RMI远程服务");  
  
 } catch (RemoteException e) {  
 e.printStackTrace();  
 }  
 }  
}

远程服务发布程序HelloServer启动后，即使main()方法执行完毕，程序仍然不会结束运行，因为它向注册器注册了远程对象，注册器一直引用这个远程对象，使得这个远程对象不会结束生命周期，因而HelloServer程序也不会结束运行，远程对象一直为客户端提供服务。

### 4. 创建客户端程序

客户端程序首先要获得RMI注册器，然后在注册器中用助记符（别名）来查找远程服务，并获得远程服务在本地的stub，之后就可以像使用本地方法一样调用远程服务方法。

在chapter12包下新建client包，在chapter12.client中创建JavaFX程序HelloClientFX，其中rmi操作相关的核心代码封装在rmiInit()方法中，程序运行参考图12.2：

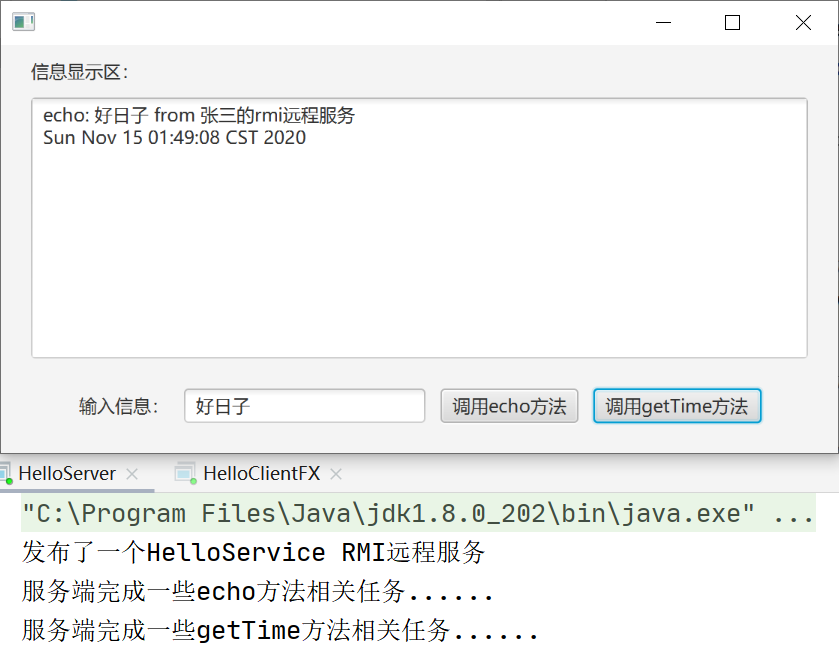


图12.2 HelloClientFX客户端界面

HelloClientFX的参考代码如下：

package chapter12.client;  
......  
public class HelloClientFX extends Application {  
 private TextArea taDisplay = new TextArea();  
 private TextField tfMessage = new TextField();  
 Button btnEcho = new Button("调用echo方法");  
 Button btnGetTime = new Button("调用getTime方法");  
 *//客户端也有一份和服务端相同的远程接口* private HelloService helloService;  
 public static void main(String[] args) {  
 *launch*(args);  
 }  
 @Override  
 public void start(Stage primaryStage) {  
 *//main区域* VBox vBoxMain = new VBox();  
 vBoxMain.setSpacing(10);*//各控件之间的间隔  
 //VBoxMain面板中的内容距离四周的留空区域* vBoxMain.setPadding(new Insets(10, 20, 10, 20));  
 HBox hBox = new HBox();  
 hBox.setSpacing(10);*//各控件之间的间隔  
 //HBox面板中的内容距离四周的留空区域* hBox.setPadding(new Insets(10, 20, 10, 20));  
 hBox.setAlignment(Pos.*CENTER*);  
 hBox.getChildren().addAll(new Label("输入信息："),tfMessage,  
 btnEcho,btnGetTime);  
  
 vBoxMain.getChildren().addAll(new Label("信息显示区："),  
 taDisplay,hBox);  
 Scene scene = new Scene(vBoxMain);  
 primaryStage.setScene(scene);  
 primaryStage.show();

*//初始化rmi相关操作*  
 new Thread(()->{rmiInit();}).start();  
 btnEcho.setOnAction(event -> {  
 try {  
 String msg = tfMessage.getText();   
 taDisplay.appendText(helloService.echo(msg) + "\n");  
 } catch (RemoteException e) {  
 e.printStackTrace();  
 }  
 });  
  
 btnGetTime.setOnAction(event -> {  
 try {  
 String msg = helloService.getTime().toString();  
 taDisplay.appendText(msg + "\n");  
 } catch (RemoteException e) {  
 e.printStackTrace();  
 }  
 });  
 }  
  
 */\*\*  
 \* 初始化rmi相关操作  
 \*/***public void rmiInit() {** try {  
 *//(1)获取RMI注册器* Registry registry = LocateRegistry.*getRegistry*("127.0.0.1",1099);  
 System.*out*.println("RMI远程服务别名列表：");  
 for (String name : registry.list()) {  
 System.*out*.println(name);  
 }  
  
 *//(2)客户端(调用端)到注册器中使用助记符寻找并创建远程服务对象的客户端(调用端)stub，之后本地调用helloService的方法，实质就是调用了远程服务器上同名的远程接口下的同名方法* helloService = (HelloService)registry.lookup("HelloService");  
 *//另外一种创建stub的方式  
 //helloService = (HelloService)Naming.lookup("rmi://127.0.0.1:1099/" + "HelloService");* } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

调用本地接口中的方法，实质是调用远程服务器上同名的远程接口中的同名远程方法

调用本地接口中的方法，实质是调用远程服务器上同名的远程接口中的同名远程方法



图12.3 RMI程序示意图

### 5. 程序运行

首先运行服务端的HelloServer，在1099端口监听；然后运行客户端HelloClientFX，进行远程方法调用。

说明：在实际应用中，服务端和客户端一般是分别部署在不同主机，如图12.3所示，特别要注意的就是两端的远程接口要完全一致，所属的包也要同名（这里是在本地模拟RMI的服务端和客户端之间的交互，所以是共享了一份rmi包中的远程接口）。本地测试完成后，可以和同学两人一组，两人的机器都临时关闭防火墙，然后修改程序相应代码中的ip地址，分别充当客户端和服务端角色，进行交互测试。

## 三、课堂计分程序

本次课堂计分，既需要学生端编写客户端程序，远程调用教师端的远程服务方法；也需要学生端编写远程服务，由教师端进行远程回调。故涉及到两个远程接口：其中一个接口是教师端负责实现，另外一个接口则需要学生端实现来提供远程服务。

### **1. 创建两个远程接口**

**在rmi包下**，创建两个远程接口RmiMsgService和RmiKitService。RmiMsgService接口代码如下：

package rmi;  
import java.rmi.Remote;  
import java.rmi.RemoteException;

*/\*\*  
 \* 教师端和学生端各自一份的远程服务接口，定义了2个远程方法  
 \* 以下远程方法全部由教师端实现，学生端调用  
 \*/*public interface RmiMsgService extends Remote {  
  
 *//声明远程方法一，用于学生发送信息给教师端，该方法由教师端实现，学生端调用* public String send(String msg) throws RemoteException;  
  
 *//声明远程方法二 用于学生发送学号和姓名给教师端，该方法由教师端实现，学生端调用* public String send(String yourNo, String yourName) throws RemoteException;  
}

RmiKitService接口代码如下：

package rmi;  
import java.rmi.Remote;  
import java.rmi.RemoteException;

*/\*\*  
 \* 教师端和学生端各自一份的远程服务接口，定义若干实用的远程方法  
 \* 这些远程方法全部由学生端实现，教师端进行调用  
 \*/*public interface RmiKitService extends Remote {  
 *//以下远程方法全部由学生端实现，教师端进行回调  
  
 //远程方法一 将ipv4格式字符串转为长整型* public long ipToLong(String ip) throws RemoteException;  
  
 *//远程方法二 将长整型转为ipv4字符串格式* public String longToIp(long ipNum) throws RemoteException;  
  
 *//远程方法三 将“-”格式连接的MAC地址转为Jpcap可用的字节数组* public byte[] macStringToBytes(String macStr) throws RemoteException;  
  
 *//远程方法四 将Jpcap的byte[]格式的MAC地址转为"-"连接MAC字符串* public String bytesToMACString(byte[] macBytes) throws RemoteException;  
}

### 2. 创建RmiKitService接口的实现类

**在chapter12包下**，新建RmiKitService接口的远程服务类RmiKitServiceImpl，在该类中，请override实现全部四个方法。

RmiKitServiceImpl程序框架如下：

package chapter12;  
......

*/\*\*  
 \* 远程服务接口的实现类  
 \*/*public class RmiKitServiceImpl extends UnicastRemoteObject implements RmiKitService {

......  
 }

### 3. 创建学生端的远程服务发布程序

**在chapter12包中**，新建类RmiStudentServer，该类负责在RMI注册器中注册和启动学生端的远程服务对象（绑定助记符名称使用RmiKitService，供教师端检索），供教师端进行远程回调。

该程序框架如下：

package chapter12;  
......   
*/\*\*  
 \* 该服务器程序负责在RMI注册器中注册和启动远程服务对象  
 \*/*public class RmiStudentServer {  
 ......

*//**助记符（别名）使用RmiKitService，供教师端检索*

registry.rebind("RmiKitService",rmiKitService);

......  
}

### 4. 创建学生端客户程序

**在chapter12包中**，新建JavaFX应用程序RmiStudentClientFX.java（可参考HelloClientFX结构，按需添加控件），参考界面如图12.4所示。

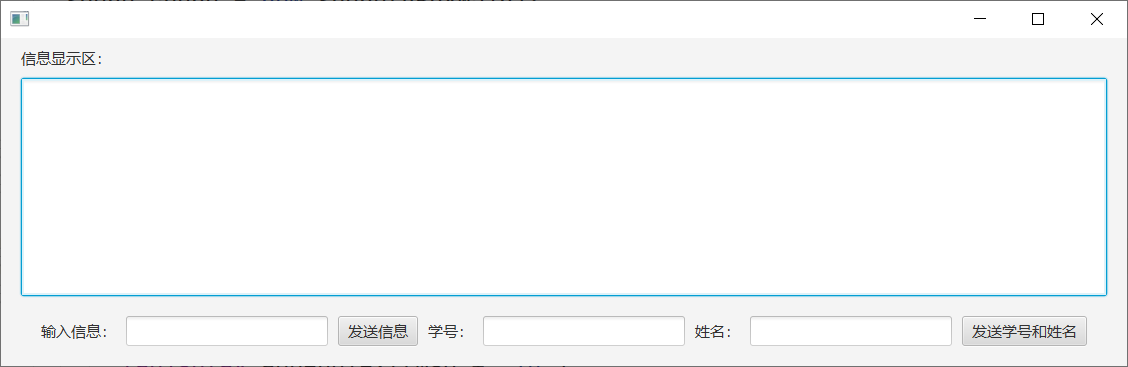


图12.4 学生客户端程序参考界面

“发送信息”按钮和“发送学号和姓名”按钮分别用于调用教师端的两个远程服务方法（即定义在远程接口中的两个不同的send方法），这个两个远程方法的返回值就是教师端返回给学生的相关信息。

注意：教师端提供的远程服务的助记符（别名）为RmiMsgService，程序框架如下：

package chapter12;  
......  
public class RmiStudentClientFX extends Application {  
 private TextArea taDisplay = new TextArea();  
 private TextField tfMessage = new TextField();  
 private TextField tfNO = new TextField();  
 private TextField tfName = new TextField();  
 Button btnSendMsg = new Button("发送信息");  
 Button btnSendNoAndName = new Button("发送学号和姓名");  
 *//客户端也有一份相同的远程接口* private RmiMsgService rmiMsgService;  
  
 public static void main(String[] args) {  
 *launch*(args);  
 }  
  
 */\*\*  
 \* 初始化rmi相关操作  
 \*/* public void rmiInit() {  
 ......

}  
  
 @Override  
 public void start(Stage primaryStage) {  
 ......

new Thread(() -> {  
 rmiInit();  
}).start();

......   
 }  
}

### 5. 完成课堂计分

首先运行学生端的远程服务程序RmiStudentServer，然后运行学生端客户程序RmiStudentClientFX，在客户端程序中使用“发送信息”按钮发送一些测试信息，看教师端是否返回测试通过的信息。如果测试通过，则完成以下两个课堂任务：

（1）文本框中分别输入学号和姓名，使用“发送学号和姓名”按钮进行发送，如果调用成功，教师端的会返回获取2分的信息；

（2）输入信息的文本框中输入“**学号&我的RMI服务器已经启动**”，例如输入 20181111111&我的RMI服务器已经启动，然后再通过“发送信息”按钮发送，教师端将随机调用你实现的四个方法中的某2个方法，如果方法实现正确，教师端将返回获得3分的信息。

参考界面如图12.5所示：

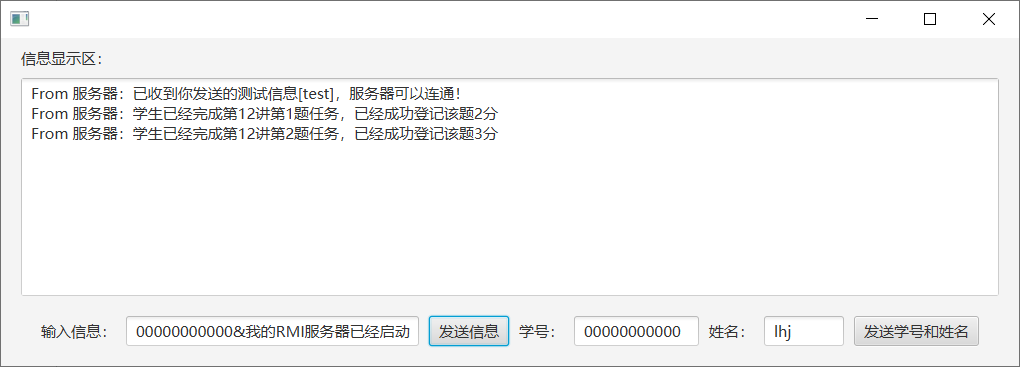


图12.5 计分界面

如果你第2题无法获取分数，可能原因是：

（1）没有运行自己的远程服务程序，或运行有误；

（2）你的4个远程方法的实现存在错误；

（3）你的远程服务别名没有按要求命名；

（4）没有临时关闭防火墙，造成教师端访问被拒；

（5）可能是你机器上有多个网卡（虚拟网卡）造成教师端访问了错误的ip地址，有两个解决方案，任选其一，优选方案1：

1. 在你的远程服务发布程序中，使用System.*setProperty*("java.rmi.server.hostname",本机器的ip地址);

绑定固定IP，这样就不会出现访问到其它IP的问题；

2. 临时禁用无关的网卡（尤其是虚拟网卡）。

## 四、扩展练习

（1）RMI服务发布程序运行后，实际需要两个端口，一个是默认的1099注册端口，还会随机使用一个服务端口用作TCP连接，而这个随机的端口就会使得程序安全部署造成困难，因为防火墙不知道该开放哪个端口。请自行搜索解决方案，使得RMI的服务端口可以固定；

（2）RMI客户端向服务端检索服务时，如果服务端没有开启，会有很长的时间才会超时，请自行搜索解决方案，如何可以控制客户端连接服务端的超时时间。