基于 RMI 的 MVC 分层多机系统编程的实现

According to the RMI MVC layering system plait distance realizes

曹大有 徐艳玲 （郧阳师范高等专科学校 计算机科学系 湖北丹江口 442700）

摘要：MVC 架构是将应用程序对象的模型与显示它的 GUI 元素相分离，在 Java 2 Swing 组件体系中应用很广泛，它也对系统的多机分层布局提供了支持。文中讨论了如何基于 RMI 来实现 MVC 多机分层布局的编程过程，并对实现过程进行了详细的讨论。

关键词：MVC 业务层 业务对象 RMI

1 问题提出

模型视图控制器 MVC 的概念来源于 Smalltalk，主要用于设计用户界面。在这样的界面中，应用程序由三个类组成，即：模型、视图和控制器。模型表示数据或应用对象，用于操作和向用户展示内容，它是对数据源和所有基于对这些数据操作的封装。视图是模型的屏幕表示，它是表示模型当前状态的对象，它将用户的请求传给控制器。控制器定义了用户界面与用户输入进行交互的方法，它是操作模型的对象。

使用 MVC 设计模式的主要优点是模型和视图分离，清晰地分解了表示层和业务层。这样一来，我们就可以把业务逻辑的表示部分分离出来，进而我们能够建立或改变视图而不必改变模型或操作模型的控制器逻辑部分，而 MVC 还允许使用多个视图表示同一个模型。MVC 的真正价值在于：推动应用程序将其模型分离出来，形成模型自己的域。

当表示层和业务层分离之后，我们便可以对代码进行分层。一个代码层是指一组责任类似的类，通常它们被组织在一个单独的 Java 包中。上层类仅依赖同层或下层的类。

分层通常需要对层次之间进行明确的定义，不同的层可能被安排在不同的计算机上运行，从而形成一个 n 层系统，这样可以最大可能地减少需运行在用户桌面系统上的程序量。这种分层设计的优点是：层间接口清晰，上下层相互独立。对代码分层本质上就是对责任分层，这样便于代码的维护。总之 MVC 为 n 层系统提供了支持，n 层系统又为软件的开发与部署带来了很多实践上的便利。

例如在用户图形界面中，我们可能有多种方式来修改一个整型数据，并且这个整型数据的变化有可能会引起若干个不同的显示区域的更新。假设我们需要创建一个改变整型数据的滑动条，而这一整型数据又以某种标签的形式显示。当该整型数据改变后，我们需要标签也能立即得到更新，但我们又不希望标签知道关于滑动条的任何信息。如果我们不知道为什么标签不应该了解滑动条，我们可以设想将滑动条换成使用文本域来输入所需要的数据，这时将会怎样？我们不必在每次改变输入源时去修改标签，这就是 MVC 的分层设计模式。而上述问题的分层结构如图 1 所示。

下面我们就来探讨如何通过 RMI 方式进行 MVC 的分层多机编程。

2 基于 RMI 的 MVC 分层多机系统编程的实现

首先是 MVC 模型或者是 RMI 服务器接口的定义：

```
public interface RMIServer extends Remote
{ //远程访问接口定义
    public static final String REGISTRY_NAME
= "RMI Server";
    public abstract void register ( RMIClient
client) throws RemoteException;
    public abstract void deregister( RMIClient
client) throws RemoteException;
```


©《计算机系统应用》编辑部 http://www.c-s-a.org.cn

public abstract void say (int message) throws RemoteException; }

其中整形变量(MVC 模型)我们用 message 表示,由于当 message 改变时,所有依赖于该整形变量的 MVC 视图对象都要得到立即更新。所以我们用 register()和 deregister()方法允许各个 MVC 视图对象(RMI 的客户机)注册及撤销注册,而方法 say (int message)则向各个注册了的 MVC 视图对象发送最新的整形变量的值。以上方法都要向各个客户机或者 MVC 视图对象暴露,由它们进行远程调用。

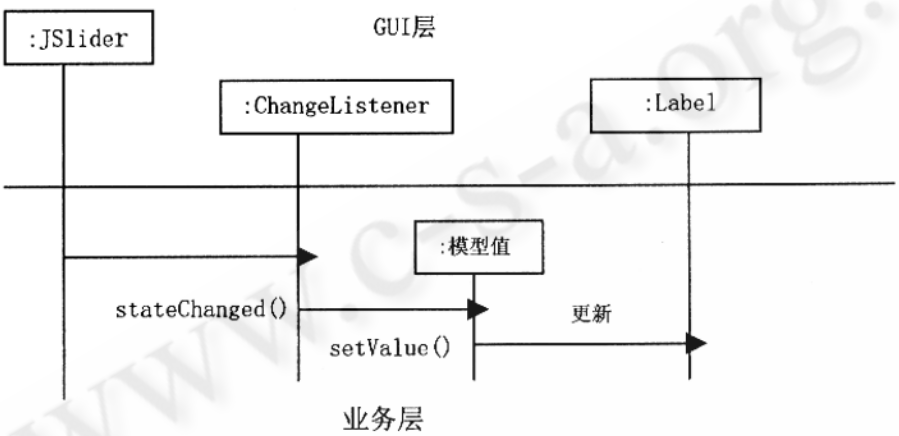

图 1

由于这种方式采用了带回叫机制的 RMI,所以各个 MVC 视图对象(RMI 客户机)也要对服务器或者 MVC 模型暴露,成为对外开放的远程对象。那么 MVC 视图对象或者客户机接口的定义为:

public interface RMIClient extends Remote { //客户端定义

public abstract void said (int message) throws RemoteException; }

该接口中的方法在 MVC 控制器中也要实现。在服务器上调用 say()方法导致 said()方法在所有注册客户机上被调用,表明模型的数值已被改变。

(1) MVC 模型或者是 RMI 服务器的实现

提供了 RMIServer 接口的实现及实例化服务器的代码并在命名注册器中进行注册。

public class RMIServerImpl extends UnicastRemoteObject implements RMIServer{

Vector clients; //保存客户队列

public RMIServerImpl ( ) throws RemoteException{ clients=new Vector( ); }

public void register (RMIClient client) throws RemoteException{

clients.addElement(client); } //增加用户

public void deregister (RMIClient client) throws RemoteException{

clients.removeElement(client); } //删除用户

public void say (int message) throws RemoteException{ //向所有用户发送消息

Vector clients = (Vector) this.clients.clone( );

for(int i=0;i<clients.size ( );i++){

RMIClient client = (RMIClient)clients.elementAt(i);

try { client. said (message); //调用用户端方法

} catch (RemoteException ex) { this. clients. removeElement(client); }}}

public static void main (String args[]) throws RemoteException{

RMIServerImpl callbackServer = new RMIServerImpl( ); //获得服务器

Registry registry = LocateRegistry. getRegistry( );

registry. rebind (REGISTRY _ NAME, callbackServer); }} //在命名服务器上注册服务

RMIServerImpl 类扩展 UnicastRemoteObject 类并实现 RMIServer 接口,用一个 Vector 对象来储存注册的 MVC 视图对象,main( )方法创建一个服务器的实例,同时在本地的命名注册器中以名称 RMI Server 进行注册。

(2) MVC 视图类的实现

该类同样扩展 UnicastRemoteObject 类并实现 RMIClient 接口并用 Label 对象显示 MVC 模型的值。该类在 start( )方法中寻找服务器对象并注册,然后在 said( )方法更新 MVC 模型值。stop( )方法用

©《计算机系统应用》编辑部 http://www.c-s-a.org.cn

来关闭客户机,并向服务器撤销注册。在 main()方法中启动 MVC 视图类对象,它连接到特定的命名注册表、查询命名服务器、注册和处理更新事务。

```
public class RMIClientLabelImpl extends UnicastRemoteObject implements RMIClient{ //标签视图类定义
String host; Frame frame; Label label; RMIServer server;
public RMIClientLabelImpl ( String host ) throws RemoteException{
this.host = host; frame = new Frame( "RMIClientLabelImpl [" +host+ "]");
frame.add( label=new Label(),"South"); //加入标签定义
frame. addWindowListener ( new WindowAdapter(){
public void windowClosing ( WindowEvent ev){
try{ stop( );} catch ( RemoteException ex) {}}});
frame.pack( );} //加入窗口事件监听器
public void start( ) throws RemoteException,NotBoundException{
if(server= =null){ Registry registry= LocateRegistry.getRegistry(host);
server = ( RMIServer ) registry. lookup (RMIServer.REGISTRY_NAME); //获取服务器
server. register ( this ); frame. setVisible (true);}} //向服务器注册
public void stop( ) throws RemoteException {
frame. setVisible(false); RMIServer server = this.server; this.server=null;
if ( server! = null ) server. deregister (this);} //向服务器注销
public void said( int message) throws RemoteException{
label.setText(""+message);} //设置标签内容
public static void main ( String args [ ]) throws RemoteException,NotBoundException{
if(args.length! =1) throw new IllegalArgumentException("Syntax: RMLClientLabelImpl < host>");
RMIClientLabelImpl callbackClient = new RMIClientLabelImpl(args[0]);
callbackClient.start( );}}
```

MVC 控制器主要是操作或改变 MVC 模型的值,它设计方法和 MVC 视图类相似,只不过是在视图类的基础上增加了一个事件处理过程,通过该事件处理过程 MVC 控制器来改变 MVC 模型的值。

(3) 用滑动条来实现 MVC 控制器

实现过程如下:

```
public class RMIClientSliderImpl extends UnicastRemoteObject implements RMIClient, ChangeListener{ //滑动条控件器构造方法
String host; Frame frame; JSlider slider; RMIServer server;
public RMIClientSliderImpl ( String host) throws RemoteException{
this. host = host; frame = new Frame ( "RMIClientSliderImpl [" +host+ "]");
frame. add ( slider = new JSlider ( )," South"); slider. addChangeListener(this);
frame. addWindowListener ( new WindowAdapter( ){ //窗口事件监听器
public void windowClosing ( WindowEvent ev){
try{ stop( );}catch(RemoteException ex){}}});
frame.pack( );}
public void start( ) throws RemoteException,NotBoundException{
if ( server = = null ) { Registry registry = LocateRegistry.getRegistry(host);
server = ( RMIServer ) registry. lookup (RMIServer.REGISTRY_NAME); //获取服务器
server. register ( this ); frame. setVisible (true);}} //向服务器注册
public void stop( ) throws RemoteException{
frame. setVisible ( false ); RMIServer server
```


©《计算机系统应用》编辑部 http://www.c-s-a.org.cn

```
=this.server;
    this.server=null;if(server! =null)server.
deregister(this);}//向服务器注销
    public void said(int message) throws Re-
moteException{
      slider.setValue(message);}//设置滑动条
的内容
    public void stateChanged(ChangeEvent e-
vent){//滑动条属性改变事件
      try{RMIServer server=this.server;
    if(server! =null)server.say(slider.
getValue());}catch(RemoteException ex){}}
    public static void main(String args[])
throws RemoteException,NotBoundException{
    if(args.length! =1)throw new IllegalArgu-
mentException("Syntax:RMLClientLabelImpl <
host>");
    RMIClientSliderImpl callbackClient = new
RMIClientSliderImpl(args[0]);
    callbackClient.start();}}
```

在目录 LabelClient(包含:RMIClient.java、RMIServer.java、RMIClientLabelImpl.java、RMIServerImpl_Stub.class)、server(包含:RMIClient.java、RMIServer.java、RMIServerImpl.java、RMIServerImpl_Stub.class)、SliderClient(包含:RMIClient.java、RMIServer.java、RMIClientSliderImpl.java、RMIServerImpl_Stub.class)中分别存放了 MVC 视图类、MVC 模型类、MVC 控制器类的代码。按照 RMI 的运行方式就可以将它们分布在三个不同的机器上运行,当在控制器中通过滑动条改变模型值时,视图类中的标签对象的显示内容会立即得到更新。当然按照 MVC 的观点,我们可以随时添加视图类或控制器类,比如我们增加一个通过文本域来修改模型值的控制器类 RMIClientTextFieldImpl(在目录 TextFieldClient(包含:RMIClient.java、RMIServer.java、RMIServerImpl_Stub.class、RMIClientTextFieldImpl.java)中),它的实现方式和 RMIClientSliderImpl 控制器类一样,这样当我们将运行在第四台计算机上时,就会发现无论我们是通过滑动条还是通过文本域来修改模型的值,模型的最新值都会在这三个对象中立即显示出来。

(4) 在单机上模拟的过程

在 DOS 提示符下,进入 server 目录,用命令:

start rmiregistry 启动命名注册器;用命令:

start java RMIServerImpl 启动服务器程序。

在 DOS 提示符下,进入 LabelClient 目录,用命令:

start java RMIClientLabelImpl localhost 启动视图类程序。

在 DOS 提示符下,进入 SliderClient 目录,用命令:

start java RMIClientSliderImpl localhost 启动滑动条控制器类程序。

在 DOS 提示符下,进入 TextFieldClient 目录,用命令:

start java RMIClientTextFieldImpl localhost 启动文本域控制器类程序。

这样当改变滑动条的值或文本域中输入新的值时,模型的最新值都会在其它对象中立即显示出来。

3 总结

文中探讨了基于 RMI 的 MVC 分层系统的编程,并在多机系统上进行了实现。从实现的过程中我们更加体会到:MVC 为 n 层系统提供了支持,n 层系统又为软件的开发与部署带来了很多实践上的便利。当然在实现时,采用了带回叫的 RMI 会话机制,它的优点是模型值一改变,服务器对象将会通知各个视图类对象,但缺点是服务器类对象必须保留视图类对象的列表,且视图类和控制器类必须对远程连接自身开放。Jdk 的版本为 1.5,模拟机上装有 Apache Tomcat 5.5 Web 服务器。

参考文献

1 Steven John Metsker 著,龚 波、冯 军、程群梅等译,设计模式 Java 手册[M],北京:机械工业出版社,2006.

2 John Zukowski(美)著,邱仲潘等译,Java 2 从入门到精通[M],北京:电子工业出版社,2000.

3 Cay S.Horstmann、Gary Cornell(美)著,程峰、黄若波等译,Java 2 核心技术 卷 II:基础知识(第六版)[M],北京:机械工业出版社,2004.

©《计算机系统应用》编辑部 http://www.c-s-a.org.cn