****

分布式与云计算

实验报告

|  |  |
| --- | --- |
| 姓 名： | 谭哲文 |
| 学 号： | 8202191123 |
| 学 院： | 计算机学院 |
| 专业班级： | 计科2105 |
| 指导教师： | 余腊生 |

本科生院制

2024年6月

**实验三 RMI API**

1. **实验目的**

1.了解Java RMI体系结构，

2.学会用RMIAPI开发C/S模式的应用程序

**2. 预习与实验要求**

1.预习实验指导书及教材的有关内容，了解I技术原理；

2.尽可能独立思考并完成实验。

**3. 实验原理**

RMI是Java的一组拥护开发分布式应用程序的API。RMI使用Java语言接口定义了远程对象，它集合了Java序列化和Java远程方法协议。RMI用于不同Java虚拟机之间的通信。一个虚拟机中的对象调用另一个虚拟上中的对象的方法，只不过是允许被远程调用的对象要通过一些标志加以标识。RMI中的核心是远程对象（remote object），除了对象本身所在的虚拟机，其他虚拟机也可以调用此对象的方法。每个远程对象都要实现一个或者多个远程接口来标识自己，声明了可以被外部系统或者应用调用的方法。RMI对于远程对象主要是Stub，其中包含远程对象的定位信息。RMI架构如下：

1.客户端体系结构

(1) sub层：负责解释客户程序发出的远程方法调用；然后将其转发到下一层<

(2) 远程引用层：解释和管理客户发出的到远程服务对象的引用，并向下一层即

传输层发起P℃操作，从而将方法调用传送给远程主机

(3) 传输层：基于TCP协议，负责实际的数据传输

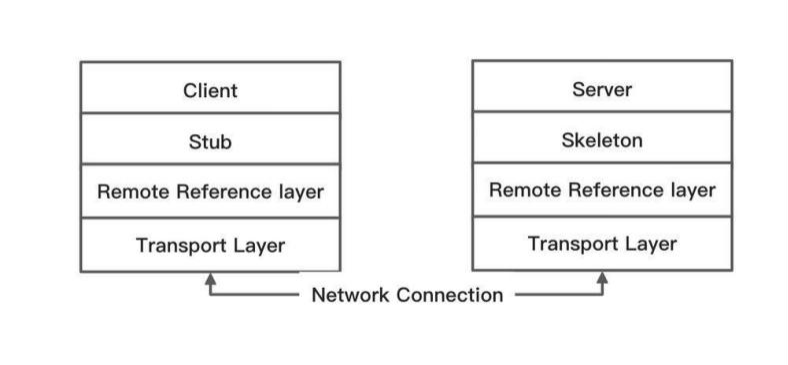
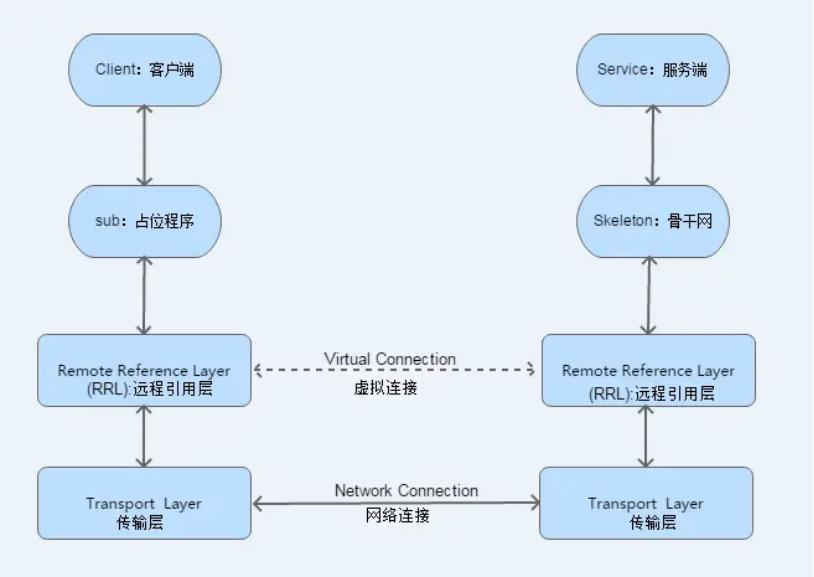
2.服务器端体系结构

(1) skeleton层：负责与客户端stub进行交互，接收客户端的远程方法调用请求

(2) 远程引用层：管理源于客户端的远程引用，并将其转换成能被skeleton层理

解的本地引用

(3) 传输层：与客户端体系结构一样，基于TCP协议，负责实际的数据传输。



Client 端的 Stub 的是 RMI Client 的代理对象，Stub 的主要功能是请求远程方

法时构造一个信息块，RMI 协议会把这个信息块发送给 Server 端。

这个信息块由几个部分组成：

 远程对象标识符。

 调用的方法描述。

 编组后的参数值（RMI协议中使用的是对象序列化）。

Client 端有一个 Stub 可以构造信息块发送给 Server 端，那么 Server 端必定

会有一个接收这个信息快的对象，称为 Skeleton 。

它主要的工作是：

 解析信息快中的调用对象标识符和方法描述，在 Server 端调用具体的对象方法。

 取得调用的返回值或者异常值。 把返回值进行编组，返回给客户端 Stub.

这样一次从 Client 端对 Server 端的调用结果就可以获取到了。

分布式系统中，由于存在多个远程对象，服务端的远程对象发布通常不是静态

地长时间保留在一个端口上。为了解决这个问题，我们引入了一个注册中心，用于记录远程对象、服务端地址以及端口的映射关系。注册中心的地址和端口是固定的，客户端可以通过预定的 IP 地址和端口与注册中心进行通信。注册中心中存储了远程对象的名字以及对应的服务端发布服务的动态地址。

通信流程主要包括以下步骤：

1. 服务端发布远程对象到注册中心： 服务端将自己的远程对象发布到注册中

心，告知注册中心该远程对象的名字、服务端地址和端口。

2. 客户端请求远程对象： 客户端向注册中心发起请求，询问某个远程对象的

地址。客户端通过注册中心查阅相应的远程对象名，获取服务端发布服务的地址。

3. 客户端本地请求转交 Stub 类： 客户端的本地请求将会被转交给 Stub 类。

Stub 类负责将请求的接口、方法、参数等信息进行序列化，以便进行网络传输。

4. 网络传输至服务端的 Skeleton 类： 序列化后的信息通过基于 TCP/IP 协议

的网络传输到服务端的 Skeleton 类。

5. Skeleton 类处理请求： Skeleton 类接收到序列化后的流后，进行反序列化，

并调用相应的远程对象进行处理。处理完请求后，将结果返回给 Skeleton 类。

6. 结果传输至客户端的 Stub 类： 处理结果被序列化后，通过 TCP/IP 协议的

网络传输返回给客户端的 Stub 类。

7. Stub 类返回结果给客户端： Stub 类接收到序列化后的结果，进行反序列

化，然后将反序列化后的对象返回给客户端，完成整个远程调用过程。

**4. 代码设计**

主要步骤如下：

1. 创建远程接口，并且继承java.rmi.Remote接口。

2. 实现远程接口，并且继承：UnicastRemoteObject

3. 创建服务器程序： createRegistry方法注册远程对象，暴露一个监听

4. 创建客户端程序，通过ip和端口连接到指定的服务器，并且将数据序列化

5. 服务器端收到请求，先反序列化。再进行业务逻辑处理。把返回结果序列化

返回首先编写服务端和客户端共用的接口和类。

实现RMI需要有继承远程对象的公共接口，服务端需要编写方法将其继承，重写方法，客户端需要在此基础上调用其方法。

public interface IRemoteObj extends Remote {

public String sayHello(Task task) throws RemoteException;

}

编写方法参数的任务类，由于在客户端实例化，调用远程对象方法需要进行通信，

因为需要序列化，指定序列化的参数。

public class Task implements Serializable{

public static final long serialVersionUID=1L;

String message;

public Task(String message)

{

this.message=message;

}

public String getMessage()

{

.

return message;

}

}

服务端

首先继承远程对象的接口，重写方法。

public class RemoteObjImpl extends UnicastRemoteObject implements IRemoteObj{

private static final long serialVersionUID = -271947229644133464L;

public RemoteObjImpl() throws RemoteException

{

super();

}

@Override

public String sayHello(Task task) {

return "Hello(from server),"+task.getMessage();

}

}服务端需要发布远程对象

public class Server {

public static void main(String[]

args)

throws RemoteException, MalformedURLException, AlreadyBoundException {

// 创建注册中心，监听端口

Registry registry= LocateRegistry.createRegistry(8000);

// 绑定远程对象

registry.bind("Hello", new RemoteObjImpl());

System.out.println("Start...");

}

}

客户端需要连接服务端的注册中心，并通过服务端注册的服务名获取远程对象，

最后调用其方法。

public static void main(String[]args)

throws MalformedURLException, RemoteException, NotBoundException {

Registry registry=LocateRegistry.getRegistry("127.0.0.1",8000);

IRemoteObj remoteObj=(IRemoteObj)registry.lookup("Hello");

System.out.println(remoteObj.sayHello(new Task("client (message from client

task)")));

}

**5. 实验结果**

实验结果展示的是服务端成功调用远程对象的方法，其中，“Hello”来源于服务

端的远程对象方法，“client”则是客户端本地实例化的对象。



**6. 思考题**

1. 如何使用RMI传递参数？

按值传递： 对于基本数据类型和实现了Serializable接口的对象，是按值传递

的。在这种情况下，RMI会将参数的拷贝传递给远程方法。这意味着在远程方法中对参数的修改不会影响到原始对象。

远程对象引用传递： 如果参数是远程对象的引用，则传递的是远程对象的引用。这意味着远程方法可以在服务器端调用传递的远程对象上的方法。这也是RMI的一项关键功能，允许客户端调用服务器端的远程对象。

2. 如何使得被动提供远程服务的服务器能够主动发起数据请求？

在传统的RMI中，通常是客户端发起对服务器端远程对象的请求，而服务器端被

动地等待客户端的调用。然而，有时候服务器需要主动发起数据请求，以便及时地向客户端提供更新的信息。服务端需要与调用服务的客户端建立一种长连接，这样保证调用远程服务的客户端能够接收数据请求。远程服务本身也必须包含监听端口和处理端口对应请求的逻辑，这样服务端在向该端口发送数据请求时，能够确保客户端能够及时进行正确的响应。在这种情况下，可以采用一些方法来实现服务器主动发起数据请求：

轮询： 服务器可以定期轮询客户端，查询是否有新的数据需要传输。这种方式

简单直观，但可能会导致无谓的网络流量，尤其是在没有数据更新的情况下。

回调： 服务器向客户端提供一个回调机制，客户端在启动时向服务器注册回调

函数。当服务器有新数据时，可以调用客户端注册的回调函数，将数据传递给客户端。这种方式需要客户端实现回调接口，并注册到服务器。

3. 如何避免在多个客户端同时发起远程服务调用时产生不一致的情况？

可以通过基于多线程的并发处理，每个线程负责处理不同的请求，能够有效避

免产生不一致的情况。由于不同线程独立执行，它们之间共享进程资源但互不干扰，这有助于确保各个请求的处理是并发安全的。通过设计线程同步机制、事务管理、错误处理和重试策略，以及合理使用分布式锁等手段，可以有效地保障系统在多客户端并发调用时的一致性和可靠性，并确保在复杂的分布式环境中各个远程服务调用的协同运作。