第五章 远程过程调用

李会格 博士/讲师

E-mail: 1034434100@qq.com

前言

- 远程过程调用(Remote Procedure Call, RPC)是一个计算机通信协议，该协议允许运行于一台计算机的程序调用存储于另一台计算机的子程序。程序员可以像调用本地程序一样，无需额外地为这个交互作用编程。

- 由于存在各式各样的变换和细节差异，RPC派生出了各式远程过程通信协议。其中，RMI(Remote Method Invocation)可以被看作SUN公司对RPC的Java版本的实现，它为编程人员提供了应用Java 远程过程调用技术的程序接口。

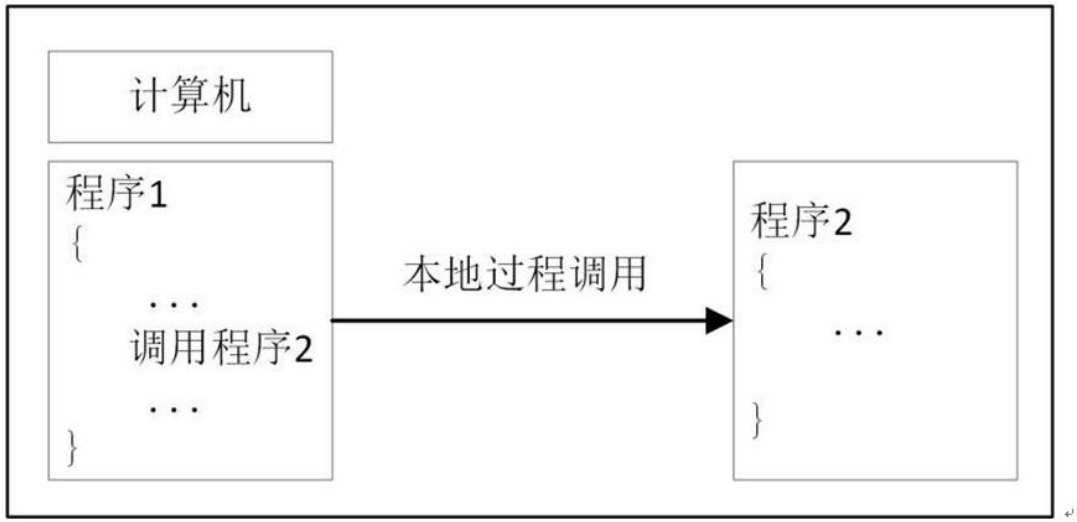
- 本章将详细介绍RPC和RMI的原理及基本流程。

大纲

- 远程过程调用RPC
  - 远程过程调用的概念和历史
  - 远程过程调用原理
  - 远程过程调用应用
- Java远程过程调用RMI
  - RMI简介
  - RMI的原理
  - RMI编程案例
- RPC和RMI的比较
  - RPC与RMI的区别
  - RMI的优点

本地过程调用的概念

- 在传统编程概念中，过程或函数是由程序员在本地编译完成，各个过程互相调用，并都局限在本地运行的一段代码。主程序和过程之间的运行关系是本地调用的关系，称为本地过程调用（Local Procedure Call, LPC）。



本地过程调用示意图

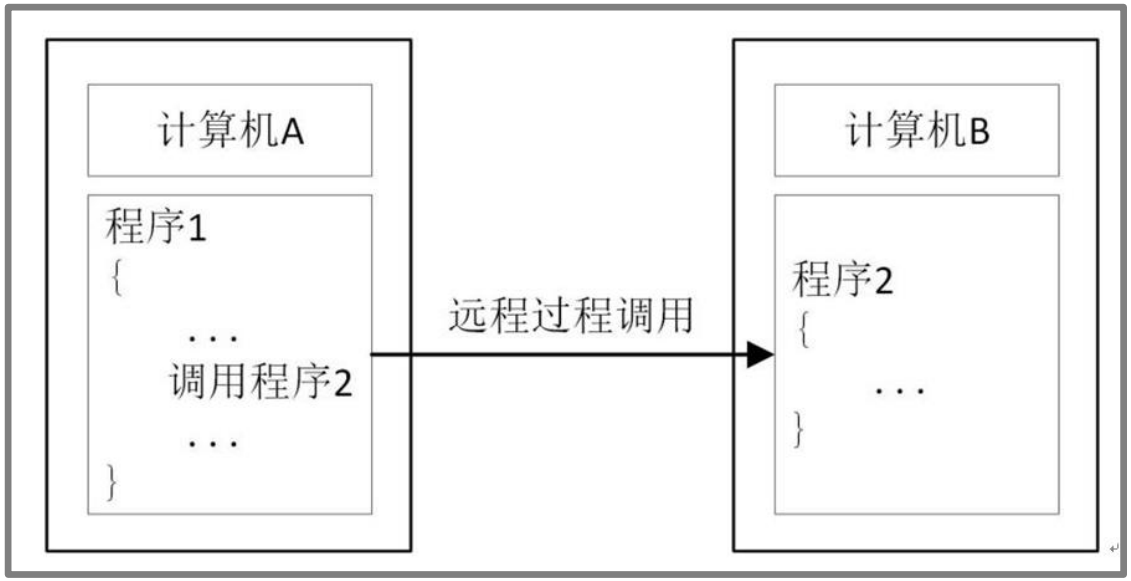
本地过程调用的优缺点

- 因为各个过程代码都在本地，可以很容易的通过加载类库、内存共享等机制直接调用。

- 这种结构在分布式计算和网络日益发展的今天已无法适应实际需求。其调用模式无法充分利用网络上其他主机的资源（如计算资源、存储资源、数据资源、显示资源等），也无法提高代码在实体间的共享程度，使得主机资源大量浪费。

RPC的概念

- 远程过程调用协议RPC（Remote Procedure Call）提供一种透明调用机制去调用另一台计算机提供的函数或者方法，使开发者不必显式的区分本地调用和远程调用，不必关注和了解底层网络技术的协议。



远程过程调用示意图

RPC的历史

- RPC思想最早的原型可追溯至1974年所发布的RFC 674草案——“过程调用协议文档第2版”。在该草案中引入了过程调用范围第2版（PCP）的概念。
- 1981年，Nelson提出RPC的概念和技术。
- 1984年，Birrell和Nelson把RPC用于支持异构型分布式系统间的通讯。
  - Birrell的RPC 模型引入存根进程（stub）作为远程的本地代理，调用RPC运行时库（RPC runtime）来传输网络中的调用。Stub和RPC runtime屏蔽了网络调用所涉及的许多细节，参数的编码/译码及网络通讯是由stub和RPC runtime完成的，因此这一模式被各类RPC所采用。

RPC的历史

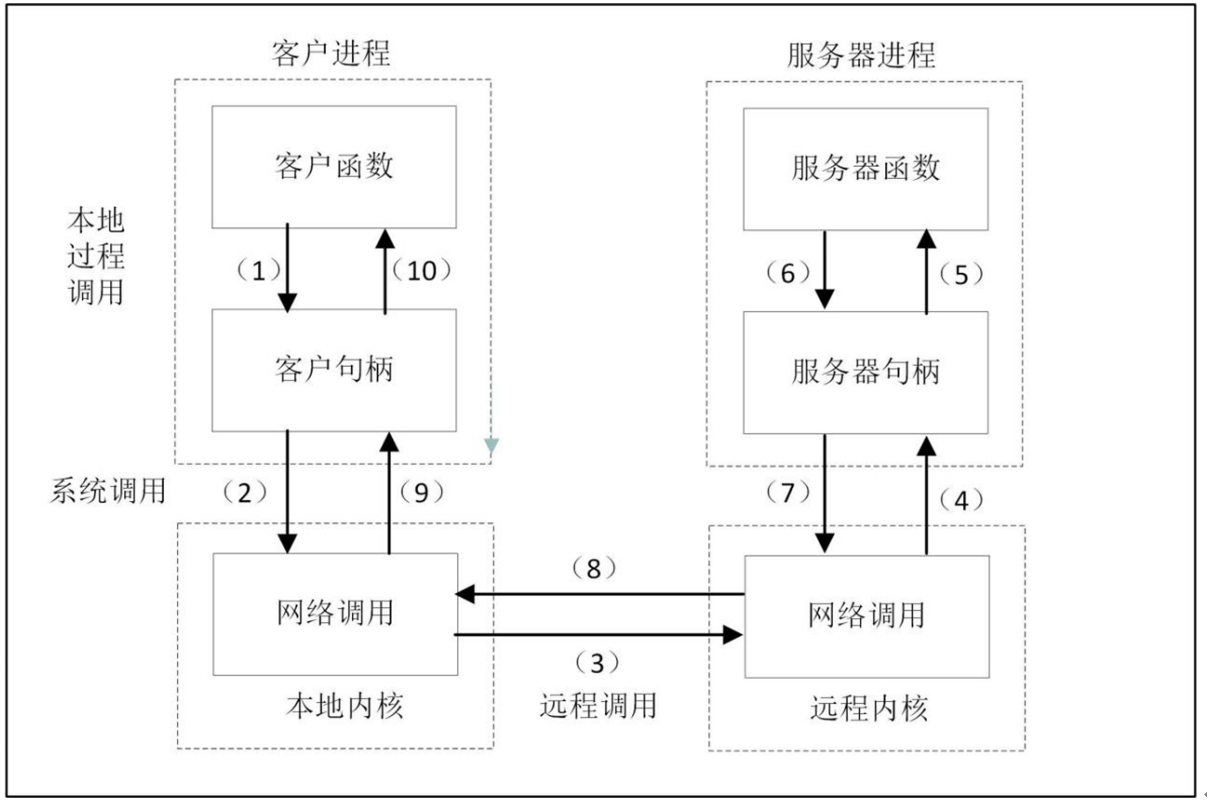
- 由于分布式系统的异构性及分布式计算模式与计算任务的多样性，RPC在方法、协议、语义、实现上不断发展，种类繁多。其中SUN公司建立的RPC较为典型，即Sun RPC。后来IETF ONC宪章重新修订了Sun 版本，使得 ONC RPC 协议成为IETF标准协议。

- 在SUN公司的网络文件系统NFS及开放网络计算环境ONC中，RPC是其基本实现技术。开放软件基金会（OSF）发展的另一个重要的分布式计算软件环境（Distributed Computing Environment, DCE）也是基于RPC的。由于对分布式计算的广泛需求，ONC和DCE成为服务器/客户端（Client/Server）模式分布式计算环境的主流产品，而RPC也成为实现分布式计算的事实标准之一。很多语言都内置了RPC技术，比如Java平台的RMI和.NET平台的Remoting。

大纲

- 远程过程调用RPC
  - 远程过程调用的概念和历史
  - 远程过程调用原理
  - 远程过程调用应用
- Java远程过程调用RMI
  - RMI简介
  - RMI的原理
  - RMI编程案例
- RPC和RMI的比较
  - RPC与RMI的区别
  - RMI的优点

RPC模式



RPC调用流程

- RPC采用服务器-客户端（Client/Sever）模式。请求程序就是一个客户机，而服务提供程序就是一个服务器。

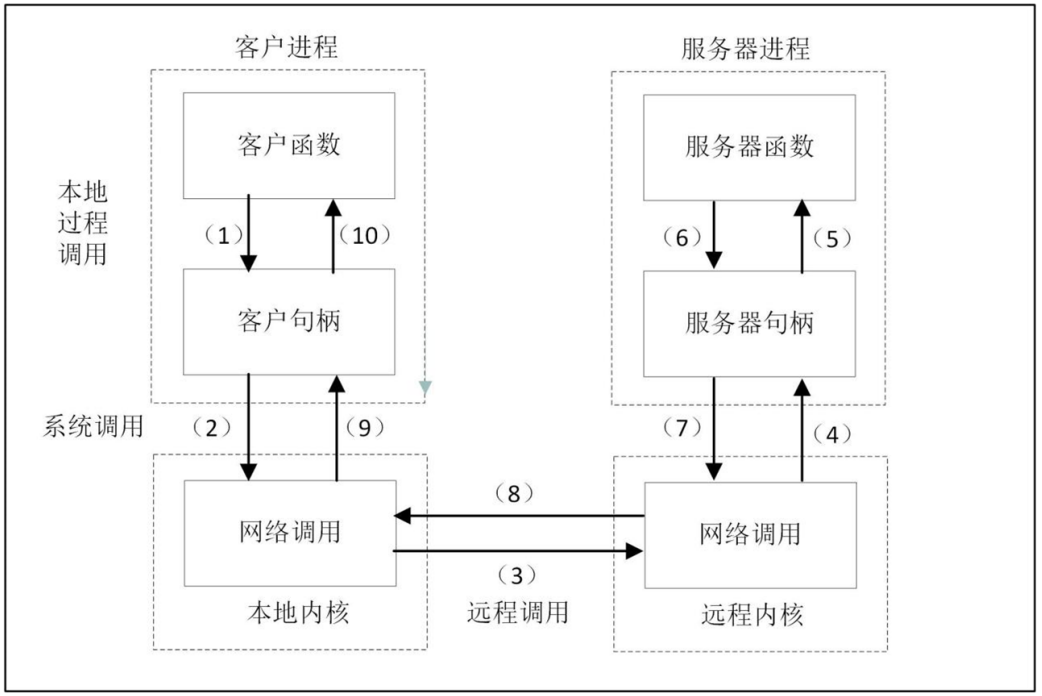
RPC框架结构

- 为了实现客户机与服务器端的透明调用，RPC引入以下结构
  - ✓ 客户句柄
    - 即客户端存根client stub，表现得就像本地程序一样，在底层调用请求和参数序列化并通过通信模块发送给服务器；客户端存根序等待服务器的响应信息，将响应信息反序列化并返回给请求程序。
  - ✓ 网络通信模块
    - 即sockets，用于传输RPC请求和响应，可以基于TCP或UDP协议实现。
  - ✓ 服务器句柄
    - 即服务端存根server stub，负责接收客户端发送的请求和参数并反序列化，根据调用信息触发对应的服务程序，然后将服务程序的响应信息，并序列化并发回给客户端。

RPC调用流程

- 一次RPC调用大致有如下操作：

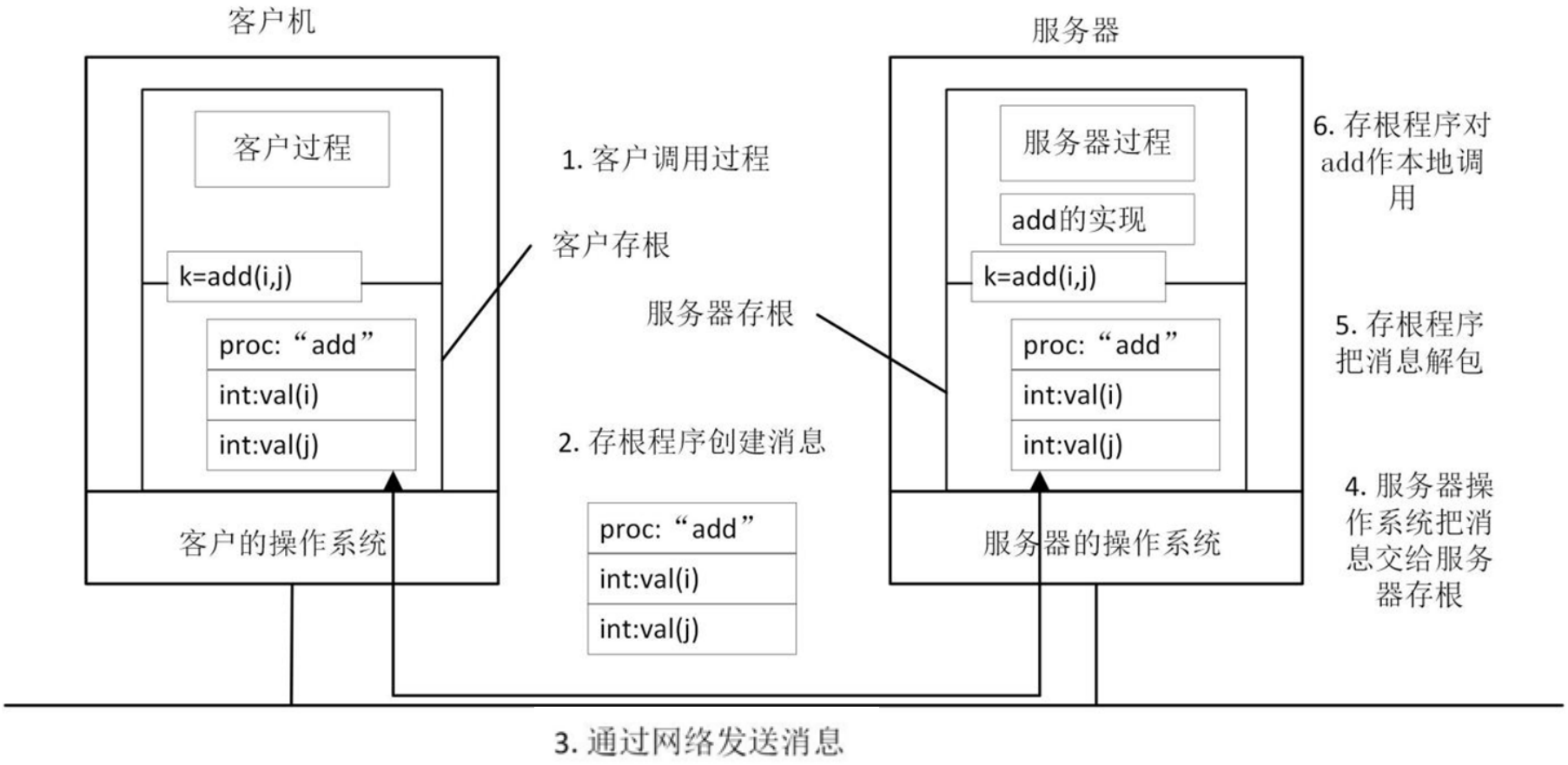
1. 调用客户端句柄，执行传送参数
2. 调用本地系统内核发送网络消息
3. 将消息传送到远程主机；
4. 服务器存根得到消息并取得参数
5. 执行远程过程
6. 执行的过程将结果返回服务器句柄；
7. 服务器句柄返回结果，调用远程系统内核；
8. 消息传回本地主机；
9. 客户端存根由内核接收消息
10. 客户段接收句柄返回的数据；



RPC调用流程

- RPC将2~8步骤封装，使得远程方法调用看起来像调用本地方法一样。

RPC调用例子



RPC远程计算实例

大纲

- 远程过程调用RPC
  - 远程过程调用的概念和历史
  - 远程过程调用原理
  - 远程过程调用应用
- Java远程过程调用RMI
  - RMI简介
  - RMI的原理
  - RMI编程案例
- RPC和RMI的比较
  - RPC与RMI的区别
  - RMI的优点

RPC应用

- RPC在分布式系统中的系统环境建设和应用程序设计中有着广泛的应用，应用包括如下方面：

✓ 分布式操作系统的进程间通讯

- 进程间通讯是操作系统必须提供的基本设施之一。分布式操作系统必须提供分布于异构的结点机上进程间的通讯机制，RPC是实现消息传送模式的分布式进程间通讯的手段之一。

✓ 构造分布式计算的软件环境

- 由于分布式软件环境本身地理上的分布性，它的各个组成成份之间存在大量的交互和通讯，RPC是其基本的实现方法之一。ONC+和DCE两个流行的分布式计算软件环境都是使用RPC构造的，其它一些分布式软件环境也采用了RPC方式。

RPC应用

✓ 远程数据库服务

- 在分布式数据库系统中，数据库一般驻存在服务器上，客户机通过远程数据库服务功能访问数据库服务器，现有的远程数据库服务是使用RPC模式的。例如，Sybase和Oracle都提供了存储过程机制，系统与用户定义的存储过程存储在数据库服务器上，用户在客户端使用RPC模式调用存储过程。

✓ 分布式应用程序设计

- RPC机制与RPC工具为分布式应用程序设计提供了手段和方便，用户可以无需知道网络结构和协议细节而直接使用RPC工具设计分布式应用程序。

RPC应用

✓ 分布式程序的调试

- RPC可用于分布式程序的调试。使用反向RPC使服务器成为客户并向它的客户进程发出RPC，可以调试分布式程序。例如，在服务器上运行一个远端调试程序，它不断接收客户端的RPC，当遇到一个调试程序断点时，它向客户机发回一个RPC，通知断点已经到达，这也是RPC用于进程通信的例子。

大纲

- 远程过程调用RPC
  - 远程过程调用的概念和历史
  - 远程过程调用原理
  - 远程过程调用应用
- Java远程过程调用RMI
  - RMI简介
  - RMI的原理
  - RMI编程案例
- RPC和RMI的比较
  - RPC与RMI的区别
  - RMI的优点

RMI简介

- 在Java领域中，远程调用方法有
  - **RMI（Remote Method Invocation）**
  - XML-RPC
  - Binary-RPC
  - SOAP（Simple Object Access Protocol ）
  - CORBA（Common Object Request Broker Architecture）
  - JMS（Java Message Service）等

- Java RMI（Remote Method Invocation）是一种用于远程过程调用的应用程序编程接口，是纯Java的网络分布式应用系统的核心解决方案之一。它可以被看作是RPC的Java版本。

RMI简介

- RPC提供了过程的分布能力，但RPC 并不能很好地应用于分布式对象系统。

- RMI 则在 RPC 的基础上向前又迈进了一步，提供分布式对象间的通讯，支持远程对象之间的无缝远程调用。

- 在RMI规范中列出了RMI系统的目标：
  - 支持对存在于不同 Java 虚拟机上对象的无缝的远程调用；
  - 支持服务器对客户的回调；
  - 把分布式对象模型自然的集成到 Java 语言；
  - 使编写可靠的分布式应用程序尽可能简单；
  - 保留 Java 运行时环境提供的安全性。

大纲

- 远程过程调用RPC
  - 远程过程调用的概念和历史
  - 远程过程调用原理
  - 远程过程调用应用
- Java远程过程调用RMI
  - RMI简介
  - RMI的原理
  - RMI编程案例
- RPC和RMI的比较
  - RPC与RMI的区别
  - RMI的优点

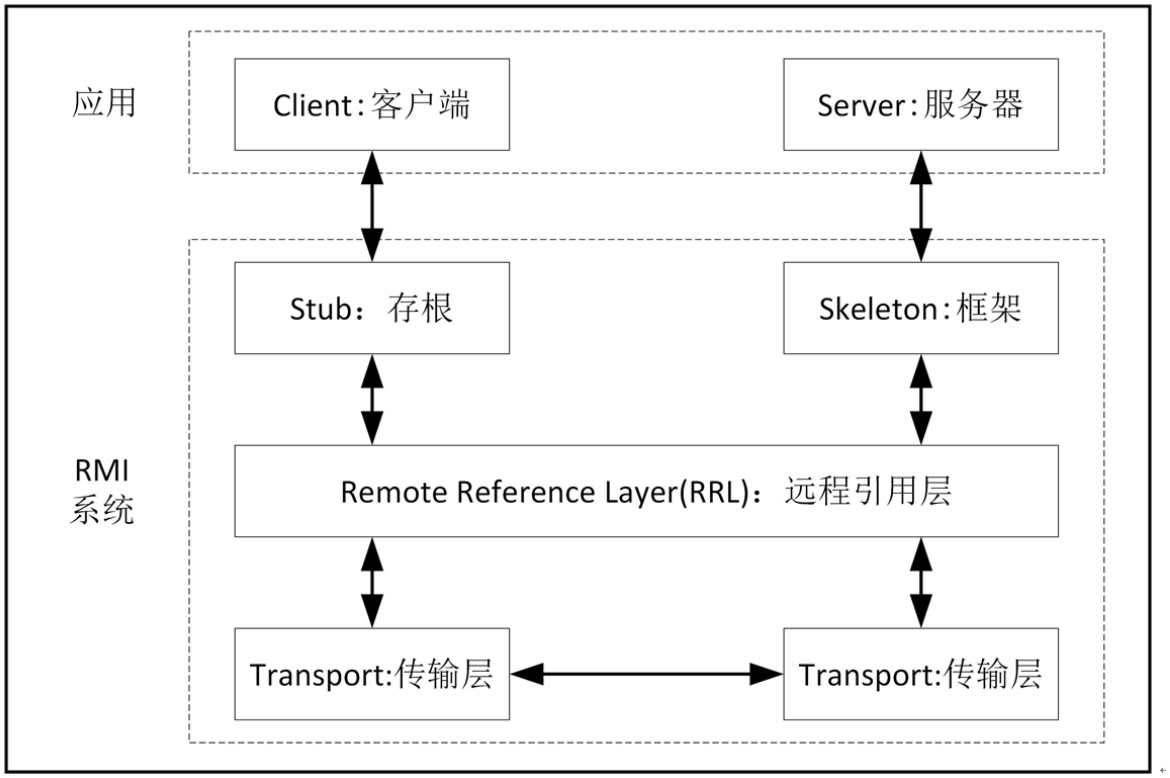
RMI的原理

- Java RMI极大地依赖于接口。在需要创建一个远程对象的时候，程序员通过传递一个接口来隐藏底层的实现细节。程序员只需关心如何通过自己的接口句柄发送消息。

- RMI支持两个类实现一个相同的远程服务接口：一个类实现行为并运行在服务器上，另一个类作为一个远程服务的代理运行在客户机上。

- 客户程序发出关于代理对象的调用方法，RMI将调用请求发送到远程Java虚拟机上的实现方法。实现方法执行后，将结果发送给代理，再通过代理将结果返回给调用者。

RMI体系



RMI系统结构图

- 为了实现位置透明性，RMI 引入了两种特殊类型的对象：存根(stub)和框架(skeleton)

RMI体系

- 存根
  - 是代表远程对象的客户机端对象，具有和远程对象相同的接口或方法列表。当客户机调用存根方法时，存根通过 RMI 基础结构将请求转发到远程对象，实际上由远程对象执行请求；
- 框架
  - 处理“远方”的所有细节，框架将远程对象从 RMI 基础结构分离开来。在远程方法请求期间，RMI基础结构自动调用框架对象
- 远程引用层
  - 处理语义、管理单一或多重对象的通信，决定调用是应发往一个服务器还是多个。
- 传输层
  - 管理实际的连接，并且追踪可以接受方法调用的远程对象。

大纲

- 远程过程调用RPC
  - 远程过程调用的概念和历史
  - 远程过程调用原理
  - 远程过程调用应用
- Java远程过程调用RMI
  - RMI简介
  - RMI的原理
  - RMI编程案例
- RPC和RMI的比较
  - RPC与RMI的区别
  - RMI的优点

RMI编程流程

- Java RMI扩展了Java的对象模型，为分布式对象提供了支持。在RMI中远程调用的对象知道它的目标对象是远程的，因此它必须处理RemoteException；远程对象的实现者也知道它是远程的，因此必须实现Remote接口。

- RMI编程的步骤大致如下：
  1. 生成一个远程接口；
  2. 实现远程对象（服务器端程序）；
  3. 生成存根和框架程序（服务器端程序）；
  4. 编写服务器程序；
  5. 编写客户程序；
  6. 注册并启动远程对象。

RMI编程案例

- 在例子中，允许用户调用查找函数来获得满足条件的学生信息。服务器为用户提供一个操作，按条件查找满足条件的学生并返回给客户端。
- 1.定义一个远程接口StudentService

```
// StudentService.java
package rmi.service;
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.List;
import rmi.model.*;

//继承自Remote类，远程对象调用的接口
public interface StudentService extends Remote {
    public List<StudentEntity> search(String name) throws RemoteException;
}
```

RMI编程案例

- 2.实现远程的接口StudentService的PersonServiceImpl类，服务端就在此远程接口的实现类中。该类将实现学生的查询方法search(String name)。

```
// StudentServiceImpl.java
package rmi.serviceimpl;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.LinkedList;
import java.util.List;
import rmi.model.StudentEntity;
import rmi.service.*;

//继承自UnicastRemoteObject，为远程对象的实现类
public class StudentServiceImpl extends UnicastRemoteObject implements
StudentService {
    public StudentServiceImpl() throws RemoteException {
        super();
    }
```

RMI编程案例

- 2.实现远程接口StudentService的PersonServiceImpl类

```
    @Override
    public List<StudentEntity> search(String name) throws RemoteException {
        System.out.println("开始查询!");
        List<StudentEntity> personList=new LinkedList<StudentEntity>();
        //假设查询有两个同学符合查询结果
        StudentEntity person1=new StudentEntity();
        person1.setAge(25);
        person1.setId(0);
        person1.setName(name);
        personList.add(person1);
        StudentEntity person2=new StudentEntity();
        person2.setAge(26);
        person2.setId(1);
        person2.setName(name);
        personList.add(person2);
        return personList;
    }
}
```

RMI编程案例

- 其中， PersonServiceImpl类使用到的StudentEntity是一个在服务器上定义的方便查询的类，其必须实现序列化Serializable接口。

```
// StudentEntity.java
package rmi.model;
import java.io.Serializable;

//注意对象必须继承Serializable
public class StudentEntity implements Serializable {
    private int id;
    private String name;
    private int age;
    public void setId(int id) {
        this.id = id;
    }
    public int getId() {
        return id;
    }
```

RMI编程案例

- 其中， PersonServiceImpl类使用到的StudentEntity是一个在服务器上定义的方便查询的类，其必须实现序列化Serializable接口。

```
// StudentEntity.java(续)
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void setAge(int age){
        this.age = age;
    }
    public int getAge(){
        return age;
    }
}
```

RMI编程案例

- 3.编写服务器程序，发布远程调用服务。

```
// ServerProgram.java
package rmi.remotingservice;
import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;
import rmi.service.*;
import rmi.serviceImpl.*;

public class ServerProgram{
    public static void main(String[] args) {
          try {
            StudentService studentService=new StudentServiceImpl();
            LocateRegistry.createRegistry(6600);//注册服务的端口
            //绑定本地地址和服务的路径
            Naming.rebind("rmi://127.0.0.1:6600/SearchService",
                                                studentService);
            System.out.println("开始服务!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

RMI编程案例

- 4.创建一个客户程序执行RMI调用。任何客户程序都需要从使用绑定程序远程对象引用开始。在客户程序中，通过lookup操作为远程对象查找一个远程对象引用。在获取了一个初始的远程对象引用后，调用远程方法。

```
// ClientProgram.java
package rmi.remotingclient;
import java.rmi.Naming;
import java.util.List;
import rmi.model.StudentEntity;
import rmi.service.*;
```

RMI编程案例

- 4.创建一个客户程序执行RMI调用。

```
// ClientProgram.java(续)
public class ClientProgram {
    public static void main(String[] args){
        try{
            //调用远程对象，RMI路径与接口必须与服务器配置一致
           StudentService studentService=(StudentService)
                  Naming.lookup("rmi://127.0.0.1:6600/ SearchService");
           List<StudentEntity> personList = studentService.search("王明");
           for(StudentEntity person:personList){
                  System.out.println("ID:"+person.getId()+"Age:"
                     +person.getAge()+"Name:"+person.getName());
           }
        }catch(Exception ex){
           ex.printStackTrace();
        }
    }
}
```

RMI编程案例

- 5. 注册并启动远程服务，在客户端即可进行远程调用。

- 本案例并没有前一小节中的第3步“生成存根和框架程序”，但是存根stub和框架skeleton类实际上还是存在的
  - 早期的Java版本为了RMI运行程序，须使用rmic来编译生成stub和skeleton程序；同时使用rmiregistry或者start rmiregistry 命令来运行RMI注册工具到系统默认的端口上。
  - 从JDK 5.0以后，stub和skeleton的类的产生不需要使用单独的rmic编译器了，它们由JAVA虚拟机自动处理。其中，存根类是通过Java动态类下载机制下载，由服务端产生，然后根据需要动态的加载到客户端

大纲

- 远程过程调用RPC
  - 远程过程调用的概念和历史
  - 远程过程调用原理
  - 远程过程调用应用
- Java远程过程调用RMI
  - RMI简介
  - RMI的原理
  - RMI编程案例
- RPC和RMI的比较
  - RPC与RMI的区别
  - RMI的优点

RPC与RMI的区别

- 1) RPC理论上支持多种语言和平台，它使得程序员不用理会操作系统之间以及语言之间的差异。RMI只用于Java，可看作是Java版本的RPC。

- 2) RMI和RPC之间最主要的区别在于方法是如何被调用的
  - ✓ RPC中通过网络服务协议向远程主机发送请求，请求包含了一个参数集和一个文本值，通常形成“classname.methodname（参数集）”的形式。然后RPC远程主机搜索与之相匹配的类和方法，执行该方法并把结果编码通过网络协议返回。
  - ✓ 而在RMI中，通过客户端的存根对象作为远程接口进行远程方法的调用。远程接口使每个远程方法都具有方法签名。如果一个方法在服务器上执行，但是没有相匹配的签名被添加到这个远程接口上，那么这个新方法就无法被RMI客户方所调用。

RPC与RMI的区别

- 3) 另外，RPC不支持对象的概念。传送到RPC服务的消息由外部数据表示（External Data Representation，XDR）语言表示。这种语言抽象了字节序类和数据类型结构之间的差异。只有由XDR定义的数据类型才能被传递，RPC不允许传递对象。可以说RMI 是面向对象方式的Java RPC，RMI调用远程对象方法，允许方法返回Java对象以及基本数据类型。

大纲

- 远程过程调用RPC
  - 远程过程调用的概念和历史
  - 远程过程调用原理
  - 远程过程调用应用
- Java远程过程调用RMI
  - RMI简介
  - RMI的原理
  - RMI编程案例
- RPC和RMI的比较
  - RPC与RMI的区别
  - RMI的优点

RMI的优点

1. 面向对象
   - RMI可将完整的对象作为参数和返回值进行传递，即可以将类似Java哈希表这样的复杂类型作为一个参数进行传递，而不需额外的客户程序代码（将对象分解成基本数据类型），直接跨网传递对象。
2. 可移动属性
   - RMI可将属性（类实现程序）从客户机移动到服务器，或者从服务器移到客户机。
3. 设计方式
   - 对象传递功能可以在分布式计算中充分利用面向对象技术的强大功能，如二层和三层结构系统。用户能够传递属性，那么就可以在自己的解决方案中使用面向对象的设计方式。

RMI的优点

4. 安全性
   - RMI使用专门设计的安全管理程序，可保护系统和网络免遭潜在的恶意下载程序的破坏。在情况严重时，服务器可拒绝下载任何执行程序。
5. 编写一次，到处运行
   - RMI是Java“编写一次，到处运行”方法的一部分。任何基于RMI的系统均可100%地移植到任何Java虚拟机上。
6. 分布式垃圾收集
   - 与Java 虚拟机内部的垃圾收集类似，RMI采用分布式垃圾收集功能收集不再被网络中任何客户程序所引用的远程服务对象。

RMI的优点

7. 并行计算
   - RMI采用多线程处理方法，可使服务器利用这些Java线程更好地并行处理客户端的请求。Java分布式计算解决方案：RMI从JDK 1.1开始就是Java平台的核心部分，因此，它存在于任何一台1.1 Java虚拟机中。所有RMI系统均采用相同的公开协议，所以，所有Java 系统均可直接相互对话，而不必事先对协议进行转换。
8. 便于编写和使用
   - RMI使得Java远程服务程序和访问这些服务程序的Java客户程序的编写工作变得轻松、简单。近来也出现了基于RMI的一些远程过程调用的框架，如Dubbo。Dubbo是一款高性能、轻量级的开源Java RPC框架。提供了三大核心能力：面向接口的远程方法调用，智能容错和负载均衡，以及服务自动注册和发现等。

本章小结

- 远程过程调用RPC是实现分布式计算最核心底层的协议之一，允许运行于一台计算机的程序调用存储于另一台计算机的子程序。

- 本章先概述远程过程调用RPC的概念，再详细说明远程过程调用的原理和调用流程；然后介绍了基于Java的远程方法调用RMI的概念、原理，并通过实例来理解RMI机制；最后总结了RPC与RMI的区别并介绍了RMI的主要优点。

课堂练习

- 仿照下面两个网址中的搭建模式，完成其中一种远程调用的搭建与测试
  - ✓ https://blog.csdn.net/qfyh_djh/article/details/136594522
  - ✓ https://www.cnblogs.com/nemuzuki/p/17658815.html