JAVA语言

- 1991年，Sun公司的James Gosling。Bill Joe等人，为电视、控制考面包机等家用电器的交互操作开发了一个Oak（一种橡树的名字）软件，他是Java的前身。当时，Oak并没有引起人们的注意，直到1994年，随着互联网和3W的飞速发展，他们用Java编制了HotJava浏览器，得到了Sun公司首席执行官的支持，得以研发和发展。为了促销和法律的原因，1995年Oak更名为Java。
- 很快Java被工业界认可，许多大公司如IBM，Microsoft，DEC等购买了Java的使用权，并被美国杂志PC Magazine评为1995年十大优秀科技产品。从此，开始了Java应用的新篇章。

JAVA语言

- Java平台由Java虚拟机（Java Virtual Machine）和Java 应用编程接口（Application Programming Interface、简称API）构成。Java 应用编程接口为Java应用提供了一个独立于操作系统的标准接口，可分为基本部分和扩展部分。在硬件或操作系统平台上安装一个Java平台之后，Java应用程序就可运行。现在Java平台已经嵌入了几乎所有的操作系统。这样Java程序可以只编译一次，就可以在各种系统中运行。Java应用编程接口已经从1.1x版发展到1.2版。目前常用的Java平台基于Java1.4，最近版本

2009年04月20日，oracle（甲骨文)宣布收购sun。

- Java分为三个体系JavaSE(Java2 Platform Standard Edition，java平台标准版)，JavaEE(Java 2 Platform,Enterprise Edition，java平台企业版)，JavaME(Java 2 Platform Micro Edition，java平台微型版)。

JAVA的影响

- Java的诞生时对传统计算机模式的挑战，对计算机软件开发和软件产业都产生了深远的影响：
- （1）软件4A目标要求软件能达到任何人在任何地方在任何时间对任何电子设备都能应用。这样能满足软件平台上互相操作，具有可伸缩性和重要性并可即插即用等分布式计算模式的需求。
- （2）基于构建开发方法的崛起，引出了CORBA国际标准软件体系结构和多层应用体系框架。在此基础上形成了Java.2平台和.NET平台两大派系，推动了整个IT业的发展。
- （3）对软件产业和工业企业都产生了深远的影响，软件从以开发为中心转到了以服务为中心。中间提供商，构件提供商，服务器软件以及咨询服务商出现。企业必须重塑自我，B2B的电子商务将带动整个新经济市场，使企业获得新的价值，新的增长，新的商机，新的管理。
- （4）对软件开发带来了新的革命，重视使用第三方构件集成，利用平台的基础设施服务，实现开发各个阶段的重要技术，重视开发团队的组织和文化理念，协作，创作，责任，诚信是人才的基本素质。

Java远程方法调用

- **Java**远程方法调用，即**Java RMI**（Java Remote Method Invocation）是Java编程语言里，一种用于实现远程过程调用的应用程序编程接口

RMI概述

- Java 1.1 中即引入了这种技术，在java2版本后得到显著的增强和扩充
- 大大增强了Java开发分布式应用的能力。Java作为一种风靡一时的网络开发语言，其巨大的威力就体现在它强大的开发分布式网络应用的能力上，而RMI就是开发百分之百纯Java的网络分布式应用系统的核心解决方案之一。
- 其实它可以被看作是RPC的Java版本。但是传统RPC并不能很好地应用于分布式对象系统，而Java RMI则支持存储于不同地址空间的程序级对象之间彼此进行通信，实现远程对象之间的无缝远程调用。

Java 远程方法调用

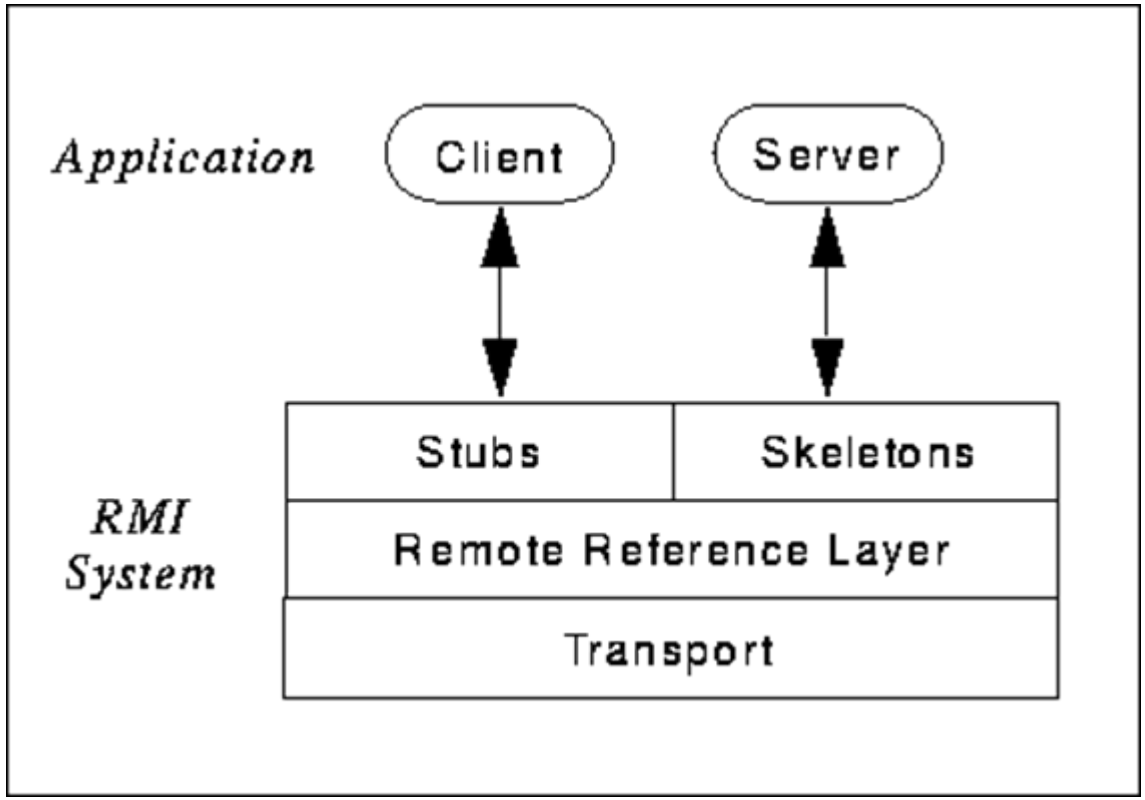
- RMI使客户机上运行的程序可以调用远程服务器上的对象。远程方法调用特性使Java编程人员能够在网络环境中分布操作。

- RMI全部的宗旨就是尽可能简化远程接口对象的使用；使分布在不同虚拟机中的对象的外表和行为都象本地对象一样

RMI的目标

- 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语言里
  - 使编写可靠的分布式运用程序尽可能简单
  - 保留java运行时环境提供的安全性

RMI 体系结构



- 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）*

Java RMI

- 调用远程对象的虚拟机有时称为客户机；包含远程对象的虚拟机称为服务器

```
MyRemoteObject o=...;
  o.myMethod();
```

- 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。
- 客户端得到的远程对象句柄正好与本地的存根代码连接，由后者负责透过网络通信。程序员只需关心如何通过自己的接口句柄发送消息。

存根 stub

- *存根*是代表远程对象的客户机端对象

  存根具有和远程对象相同的接口或方法列表，但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 skeleton

- 在服务器端，*框架*对象处理"远方"的所有细节。
  程序员完全可以象编码本地对象一样来编码远程对象。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，因此它可以发挥自己的作用。
- 关于这种设置的最大的好处是：程序员不必亲自为存根和框架编写代码。JDK 包含工具 rmic，它会为您创建存根和框架的类文件。

存根/框架的关键技术

- 对象串行化技术(object serialization)，该技术将对象的类型和值信息转化为平坦的字节流形式，并利用这种串行化表示和重建与原对像相同的同类型对象，从而实现对象状态的持久性或网络传输。存根/框架利用这一技术对远程过程调用的参数和返回值进行打包与解包
- 动态类装载(dynamic class loading)，用于在程序动态运行时装载客户程序所需的存根，并支持java语言内建的类型检查与类型转换机制。

RMI的位置透明性

- 获取远程对象的引用和获取本地对象的引用有点不同，但一旦获得了引用，就可以象调用本地对象一样调用远程对象
- RMI 基础结构将自动截取请求，找到远程对象，并远程地分派请求。
- 这种位置透明性甚至包括垃圾收集

  客户机不必特地释放远程对象，RMI 基础结构和远程虚拟机为您处理垃圾收集。

RMI程序

- 一个基于RMI的多层结构分布式应用程序通常包括以下几个部分：

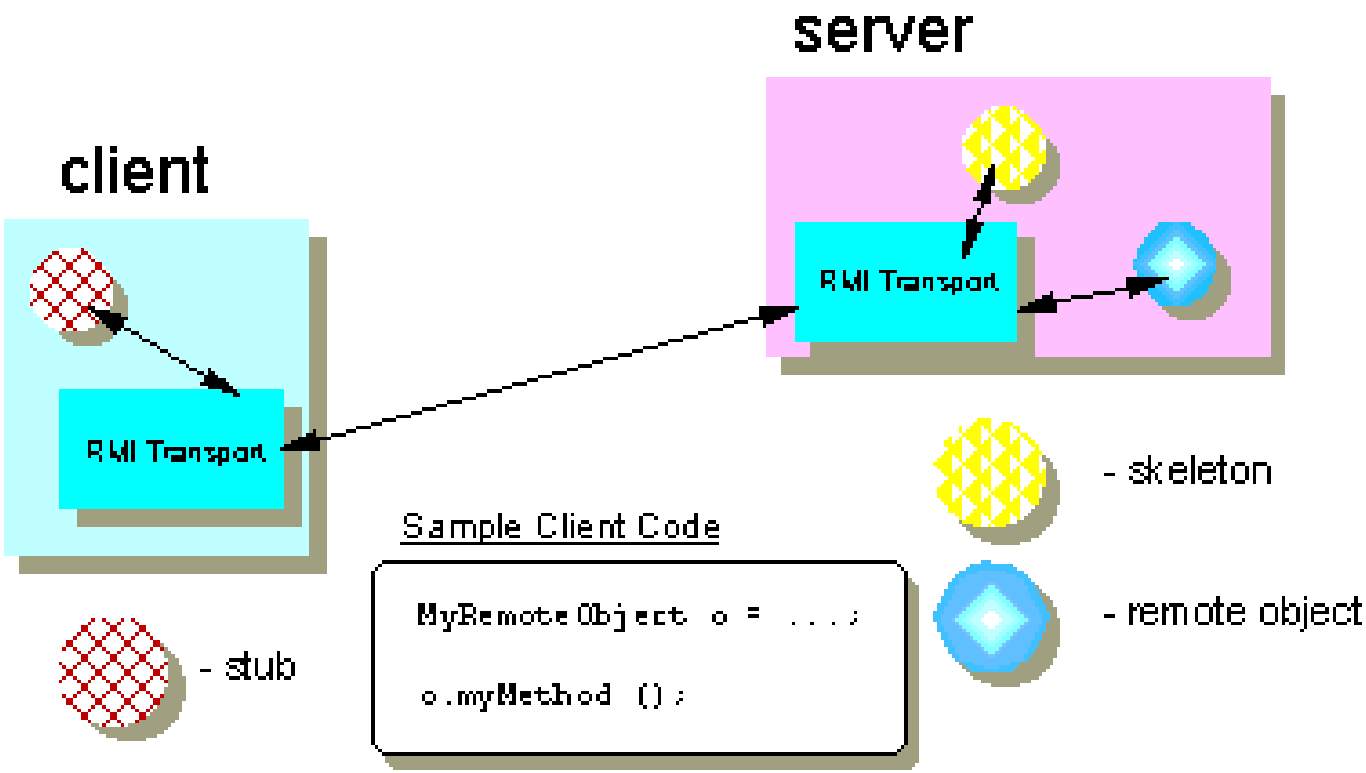
  远程对象接口：规定了客户程序与服务程序进行交互的界面，是客户方与服务方双方必须共同遵守的合约

  远程对象实现：为远程对象接口规定每一个方法提供的具体实现

  服务程序：远程对象实现并不是服务程序本身，它需要由服务器创建并注册，服务程序中这些真正提供服务的对象实例又称为伺服对象(servant)

  客户程序：与终端用户进行交互，并利用远程对象提供的服务完成某一功能

RMI Architecture



Hello World

一个例子：Hello World

RMI的开发步骤概述

- 1，定义用于远程对象的接口 Hello.java
  这个接口定义了客户机能够远程地调用的方法。远程接口和本地接口的主要差异在于，远程方法必须能抛出 RemoteException。
- 2，编写一个实现该接口的类 HelloImpl.java
- 3，编写在服务器上运行的主程序 Server.java
  这个程序必须实例化一个或多个服务器对象
- 4，将远程对象注册到 RMI 名称注册表，以便客户机能够找到对象
- 5，编写客户端程序 Client.java

远程接口定义

```
import java.rmi.*;
public interface Hello extends java.rmi.Remote {
    String sayHello() throws java.rmi.RemoteException;
}
```

- 这些方法必须能抛出 RemoteException，如果客户机和服务器之间的通信出错，则客户机将捕获此异常。
- 注：该接口本身继承了 java.rmi 包中定义的 Remote 接口。Remote 接口本身没有定义方法，但通过继承它，我们说明该接口可以被远程地调用。

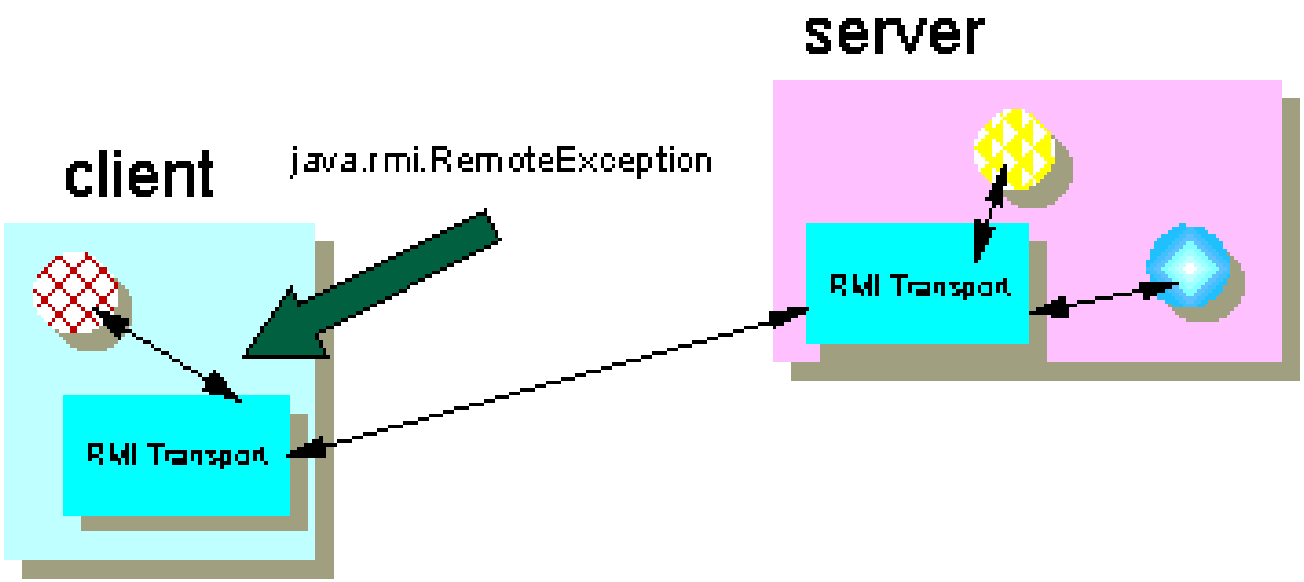
远程异常

- RMI 使用 TCP/IP 套接字来传达远程方法请求。尽管套接字是相当可靠的传输，但还是有许多事情可能出错。

  服务器计算机在方法请求期间崩溃了；客户机和服务器通过因特网连接时，客户机掉线了

- 使用远程对象时比使用本地对象时有更多可能出错的机会。因此客户机程序能够完美地从错误中恢复就很重要了。

Remote Exceptions



- 每个将要被远程调用的方法都必须抛出 RemoteException

实现远程接口

```
import java.rmi.*;
import java.rmi.server.*;

public class HelloImpl implements Hello {
    public HelloImpl() {}
    public String sayHello(String name) {
        return "Hello, "+ name +" !";
    }
}
```

- 实现Hello接口的类HelloImpl

编写 RMI 服务器概述

- 除了实现接口之外，还需要编写服务器的主程序。
- 要么只在实现类中编码一个 main 方法；要么为主程序编码一个单独的 Java 类。

```
.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.*;

public class Server{
    public Server() {}
    public static void main(String args[]) {
        System.setSecurityManager(new RMISecurityManager());
        final HelloImpl obj = new HelloImpl();

            Hello stub = (Hello)UnicastRemoteObject.exportObject(obj, 0);
            // Bind the remote object's stub in the registry
            Registry registry = LocateRegistry.createRegistry(3333);
            registry.rebind("Hello", stub);
            for(int i = 0; i < registry.list().length; i++)
                System.out.println(registry.list()[i]);
            System.err.println("Server ready....");
            System.err.println("Listinging on port 3333 ....");
        } catch (Exception e) {
            e.printStackTrace();
        }
}
}
```

设置安全性管理器

```
public class Server{
    public Server() {}
    public static void main(String args[]) {
        System.setSecurityManager(new RMISecurityManager());
……
```

- 第一步是安装 RMI 安全性管理器。尽管这不是严格必须的，但它确实允许服务器虚拟机下载类文件。

  例如，假设客户机调用服务器中的方法，该方法接受对应用程序定义的对象类型（例如 BankAccount）的引用。通过设置安全性管理器，我们允许 RMI 运行时动态地将 BankAccount 类文件复制到服务器，从而简化了服务器上的配置。

命名远程对象

```
public class Server{
    public Server() {}
    public static void main(String args[]) {
        System.setSecurityManager(new RMISecurityManager());
        final HelloImpl obj = new HelloImpl();

            Hello stub = (Hello)UnicastRemoteObject.exportObject(obj, 0);
            // Bind the remote object's stub in the registry
            Registry registry = LocateRegistry.createRegistry(3333);
            registry.rebind("Hello", stub);
```

服务器的下一步工作是创建服务器对象的初始实例，然后将对象的名称写到 RMI 命名注册表。

命名远程对象

- RMI 命名注册表允许您将 URL 名称分配给对象以便客户机查找它们。要注册名称，需调用静态 rebind 方法，它是在 Naming类上定义的。这个方法接受对象的 URL 名称以及对象引用。
- 名称字符串是很有趣的部分。它包含 rmi:// 前缀、运行 RMI 对象的服务器的计算机主机名和对象本身的名称。

  注：可以调用由 java.net.InetAddress 类定义的 getLocalHost 方法，而不必象我们在这里所做的一样硬编码主机名。

客户机开发

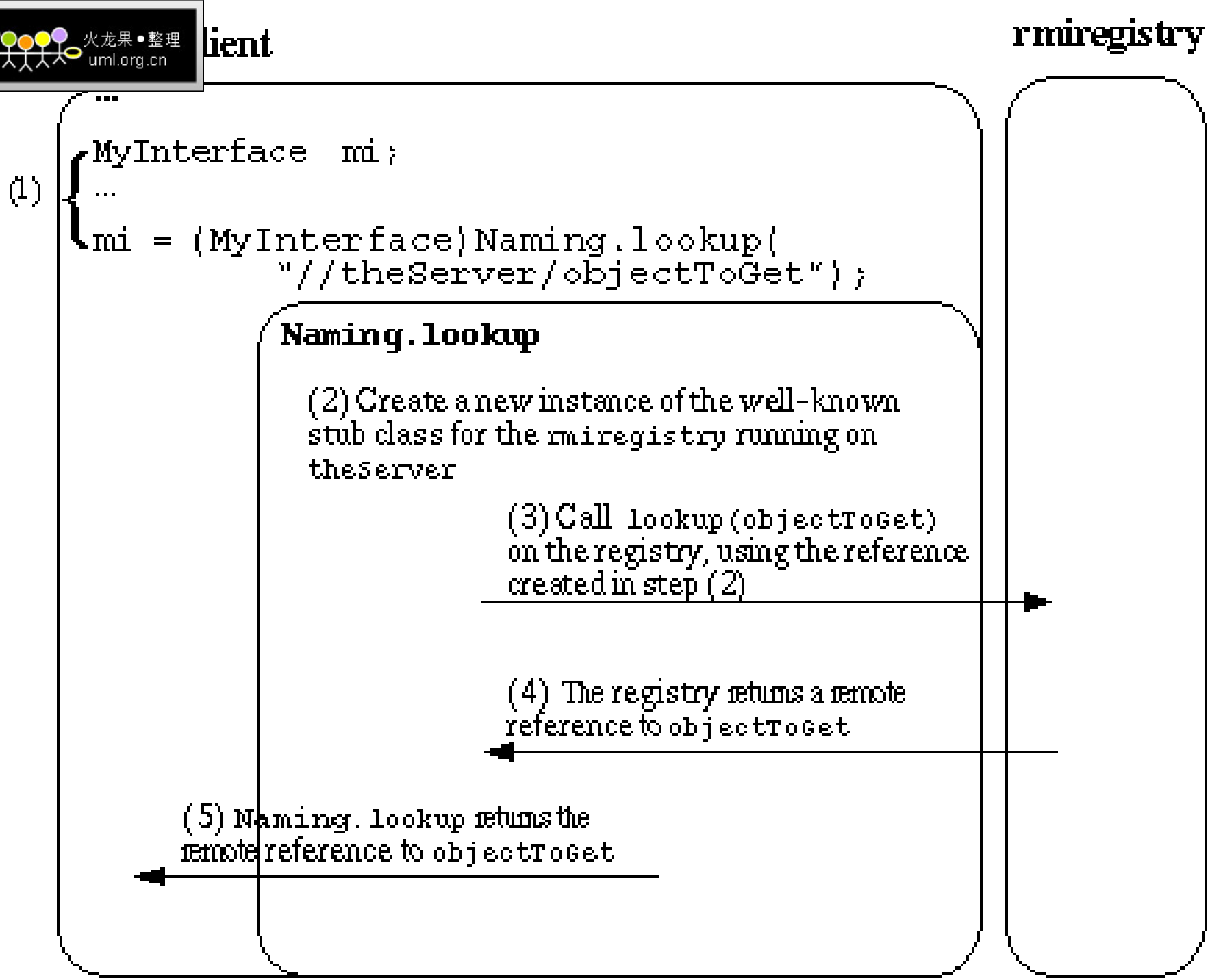
- 首先，确定是想编写客户机独立应用程序还是客户机 applet。应用程序的设置简单些，但 applet 更容易部署，因为 Java RMI 基础结构能够将它们下载到客户机机器。
- 在客户机中，代码需要首先使用 RMI 注册表来查找远程对象。一旦这样做了之后，客户机就可以调用由远程接口定义的方法。

客户端代码

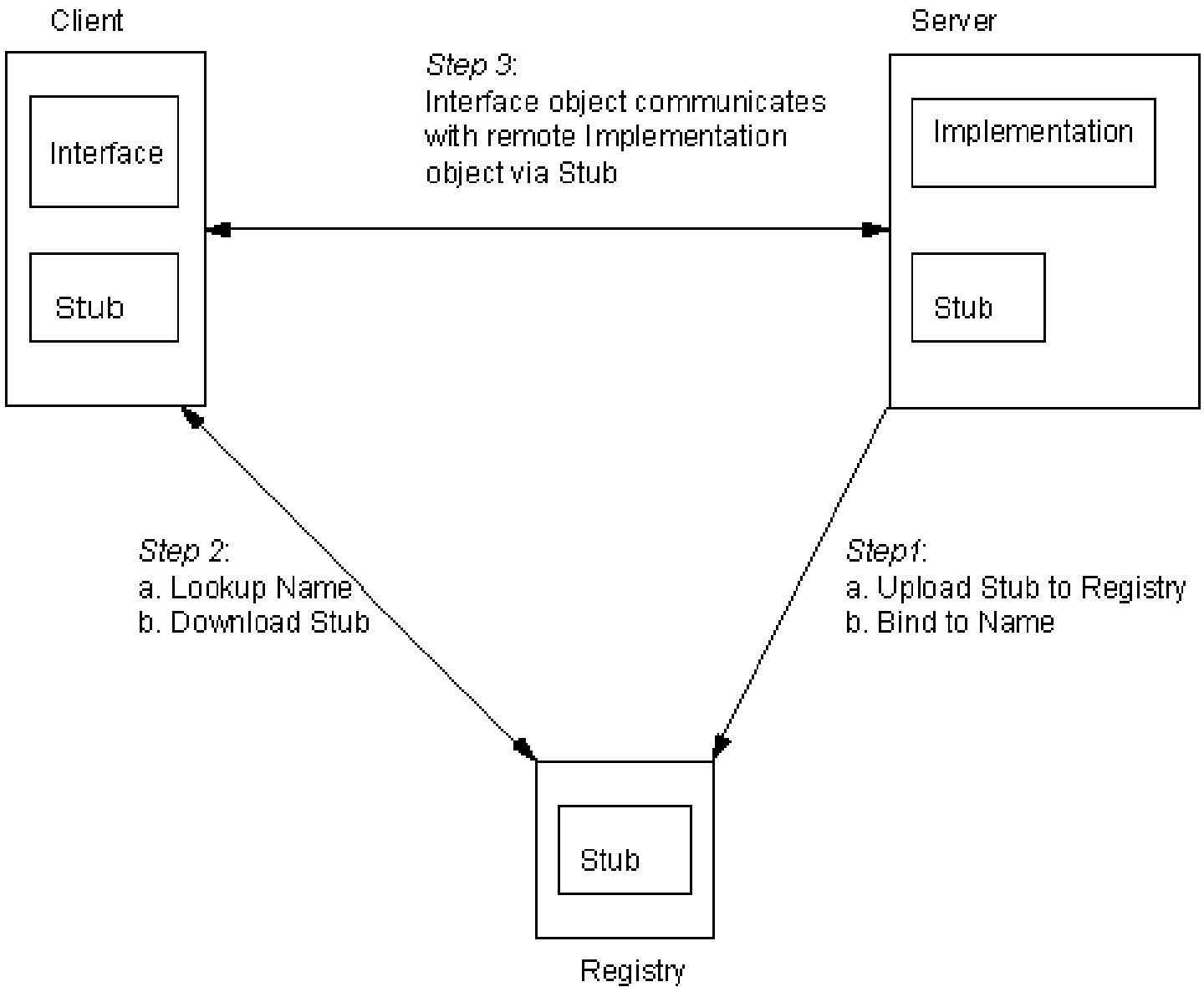
```
import java.rmi.registry.*;
import java.rmi.*;

public class Client {
    private Client() {}
    public static void main(String[] args) {
        String host = (args.length < 1) ? "localhost" : args[0];
        String name = (args.length == 2) ? args[1] : "World";
        try {
            String urlo="rmi://" + host + ":3333/Hello";
            Hello stub = (Hello)Naming.lookup(urlo);

            //Registry registry = LocateRegistry.getRegistry(host);
            //Hello stub = (Hello)registry.lookup("Hello");
            String response = stub.sayHello(name);
            System.out.println("Response: " + response);
        ……————————————
```



RMI：注册和查找



RMI:远程方法调用

```
import java.rmi.registry.*;
import java.rmi.*;

public class Client {
    private Client() {}
    public static void main(String[] args) {
        String host = (args.length < 1) ? "localhost" : args[0];
        String name = (args.length == 2) ? args[1] : "World";
        try {
            ……————
            String response = stub.sayHello(name);
            System.out.println("Response: " + response);
        ————————————————————
```

部署和运行

1 编译：

> javac *.java

2 启动服务器：

> java -Djava.security.policy=policy.txt Server

3 运行客户端：

>java Client localhost 同学们

RMI：补充

- RMI目前使用Java远程消息交换协议JRMP（Java Remote Messaging Protocol）进行通信。JRMP是专为Java的远程对象制定的协议。因此，Java RMI具有Java的"Write Once，Run Anywhere"的优点，是分布式应用系统的百分之百纯Java解决方案。用Java RMI开发的应用系统可以部署在任何支持JRE（Java Run Environment Java，运行环境）的平台上。
- 但由于JRMP是专为Java对象制定的，因此，RMI对于用非Java语言开发的应用系统的支持不足。不能与用非Java语言书写的对象进行通信。

小结

- 从最基本的角度看，RMI是Java的远程过程调用(RPC)机制
- 通常包括以下几个部分：

  远程对象接口、

  远程对象实现、

  服务程序、

  客户程序

小结：RMI的优点

面向对象：RMI可将完整的对象作为参数和返回值进行传递，而不仅仅是预定义的数据类型。

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。

设计方式：对象传递功能使您可以在分布式计算中充分利用面向对象技术的强大功能

安全：RMI使用Java内置的安全机制保证下载执行程序时用户系统的安全

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。

小结：RMI的优点

可连接现有/原有的系统：RMI可通过Java的本机方法接口JNI与现有系统进行进行交互。

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。

分布式垃圾收集：RMI采用其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。

并行计算：RMI采用多线程处理方法，可使您的服务器利用这些Java线程更好地并行处理客户端的请求。

纯Java的网络分布式应用系统的核心解决方案之一

谢谢