远程过程调用 (RPC)
Remote Procedure Call

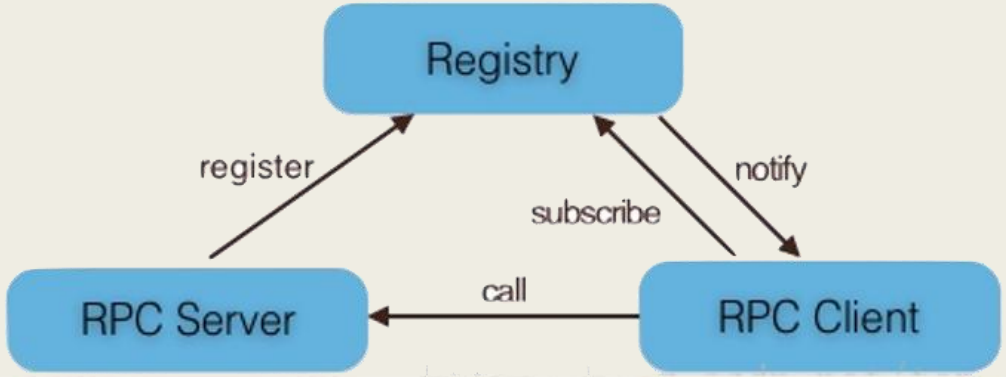
- 一种广义的远程调用技术框架，程序可使用RPC向网络中的另一台计算机上的程序请求服务。
- RPC 的主要目的是为组件提供一种相互通信的方式，使这些组件之间能够相互发出请求并传递这些请求的结果。
- 在 RPC 中，发出请求的程序是客户程序，而提供服务的程序是服务器
- 广泛用于支持分布式应用程序的技术。

RPC的实现方案

- 从0开始：基于SOCKET网络通信和序列化的编程实现
- RMI（java）
- 基于第三方的框架（Netty通信框架， gRPC、 Dubbo，Thrift等）
- SOA架构（WebService等）
- 新发展：RESTful等

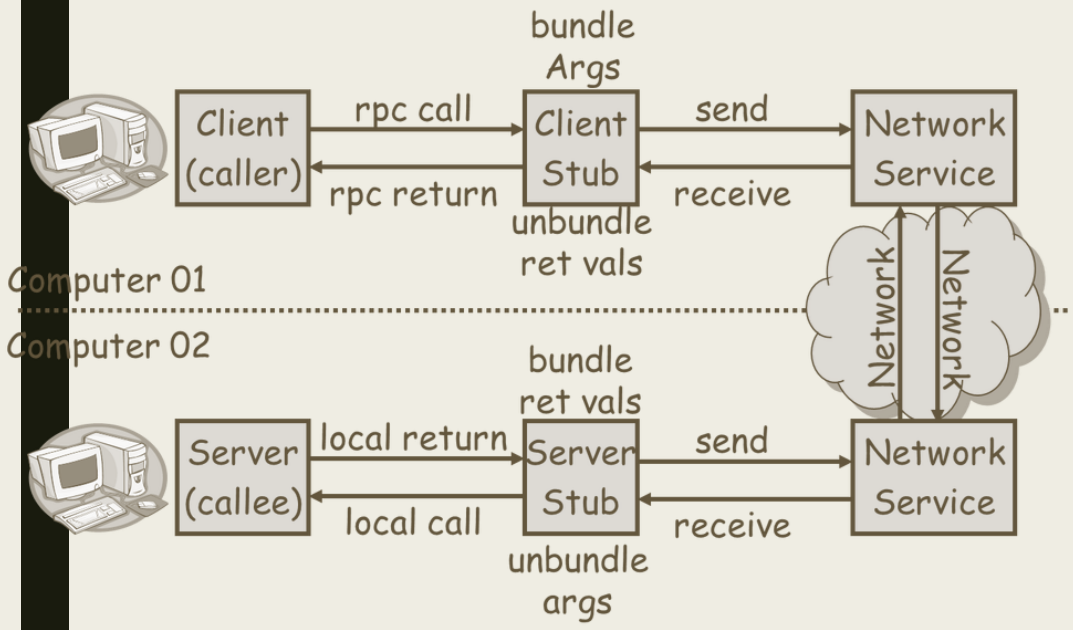
RPC框架原理

- Server: 暴露服务的服务提供方。
- Client: 调用远程服务的服务消费方。
- Registry: 服务注册与发现的注册中心。



服务提供者启动后主动向注册中心注册机器ip、port以及提供的服务列表;
服务消费者启动时向注册中心获取服务提供方地址列表,可实现软负载均衡和Failover;

RPC调用流程



- 1）服务消费方（client）调用以本地调用方式调用服务；
- 2）client stub接收到调用后负责将方法、参数等组装成能够进行网络传输的消息体；
- 3）client stub找到服务地址，并将消息发送到服务端；
- 4）server stub收到消息后进行解码；
- 5）server stub根据解码结果调用本地的服务；
- 6）本地服务执行并将结果返回给server stub；
- 7）server stub将返回结果打包成消息并发送至消费方；
- 8）client stub接收到消息，并进行解码；
- 9）服务消费方得到最终结果。

RPC框架的目标：采用相应的服务注册、发现与调用机制，把2~8这些步骤都封装起来，让用户对这些细节透明。

实现的关键技术问题

- 服务发现
- 通讯
- 数据封装与序列化

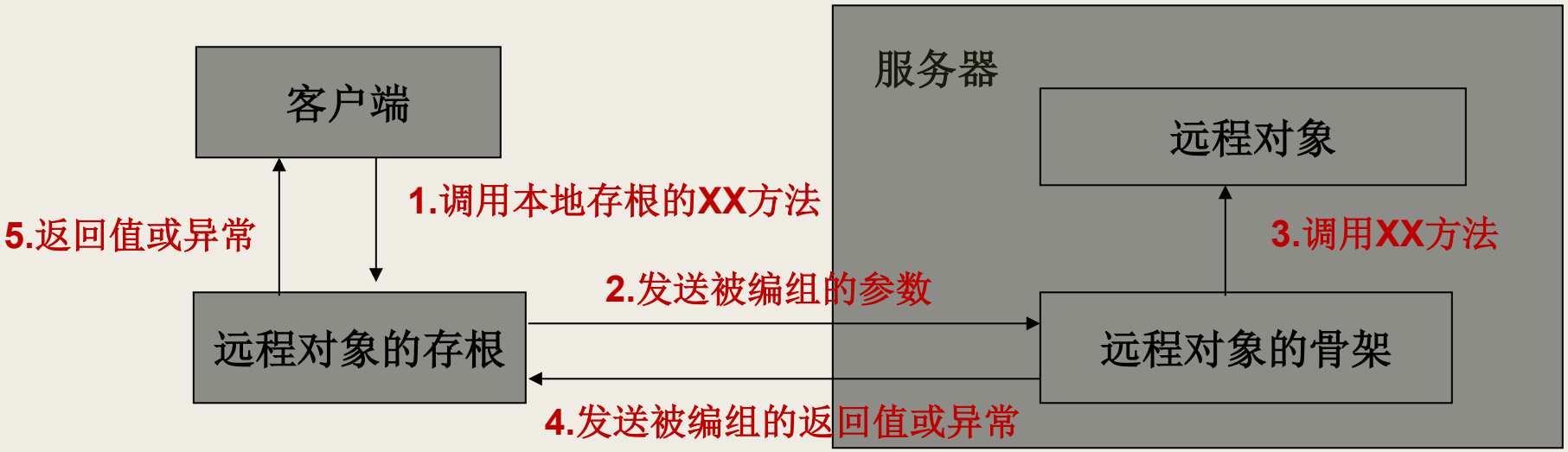
一种特殊RPC-JAVA RMI

- Java在JDK1.1中实现，增强了Java开发分布式应用的能力
- RPC的Java jdk版本

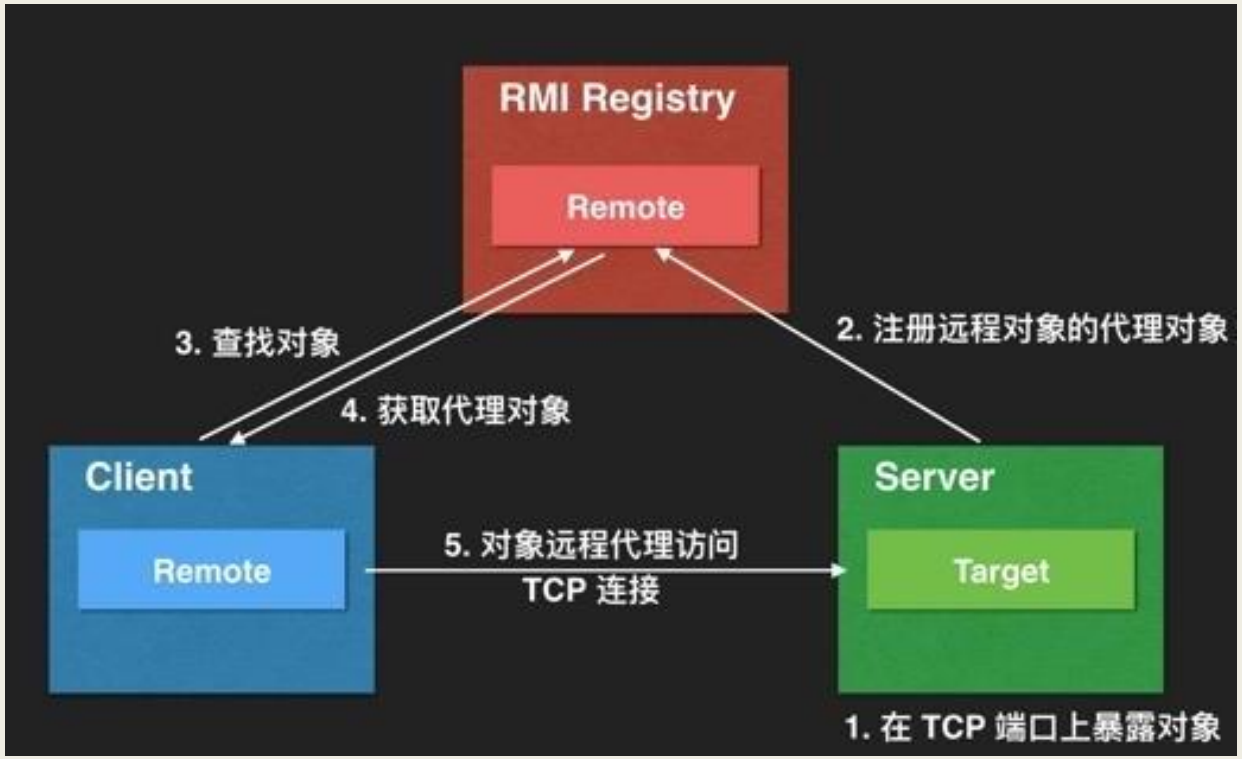
RMI基本原理

RMI采用代理来负责客户与远程对象之间通过Socket进行通信的细节

RMI为远程对象分别生成客户端代理(存根，Stub)和服务器端代理(骨架，Skeleton)。



运行机制


RMI Registry
Remote
3. 查找对象
4. 获取代理对象
2. 注册远程对象的代理对象
Client
Remote
5. 对象远程代理访问
TCP 连接
Server
Target
1. 在 TCP 端口上暴露对象

RMI 注册中心的实现-JNDI

- RMI采用一种命名服务机制来使客户程序可以找到服务器上的一个远程对象。
- RMI的命名服务API被整合到JNDI(Java Naming and Directory Interface)中。
  - *命名服务*
  - *目录服务*

JNDI中的核心类和接口

Java.rmi. Naming

Java.rmi. Remote

Java.rmi.registry.LocateRegistry

java.rmi.server.UnicastRemoteObject

远程服务类

- 需实现如下两个类/接口
  - *Java.rmi. Remote*
  - *java.rmi.server.UnicastRemoteObject*

服务注册中心的生成

LocateRegistry.createRegistry（int port）:在本机指定端口打开一个注册中心

Naming/Context/Registry 类
-远程服务对象和名称的绑定、暴露和查找

(1) bind(String name,Object obj)：如果name已与其他对象绑定，抛出NameAlreadyBoundException。

(2) rebind(String name,Object obj)：不会抛出NameAlreadyBoundException,obj指定的对象覆盖原来已绑定的对象。

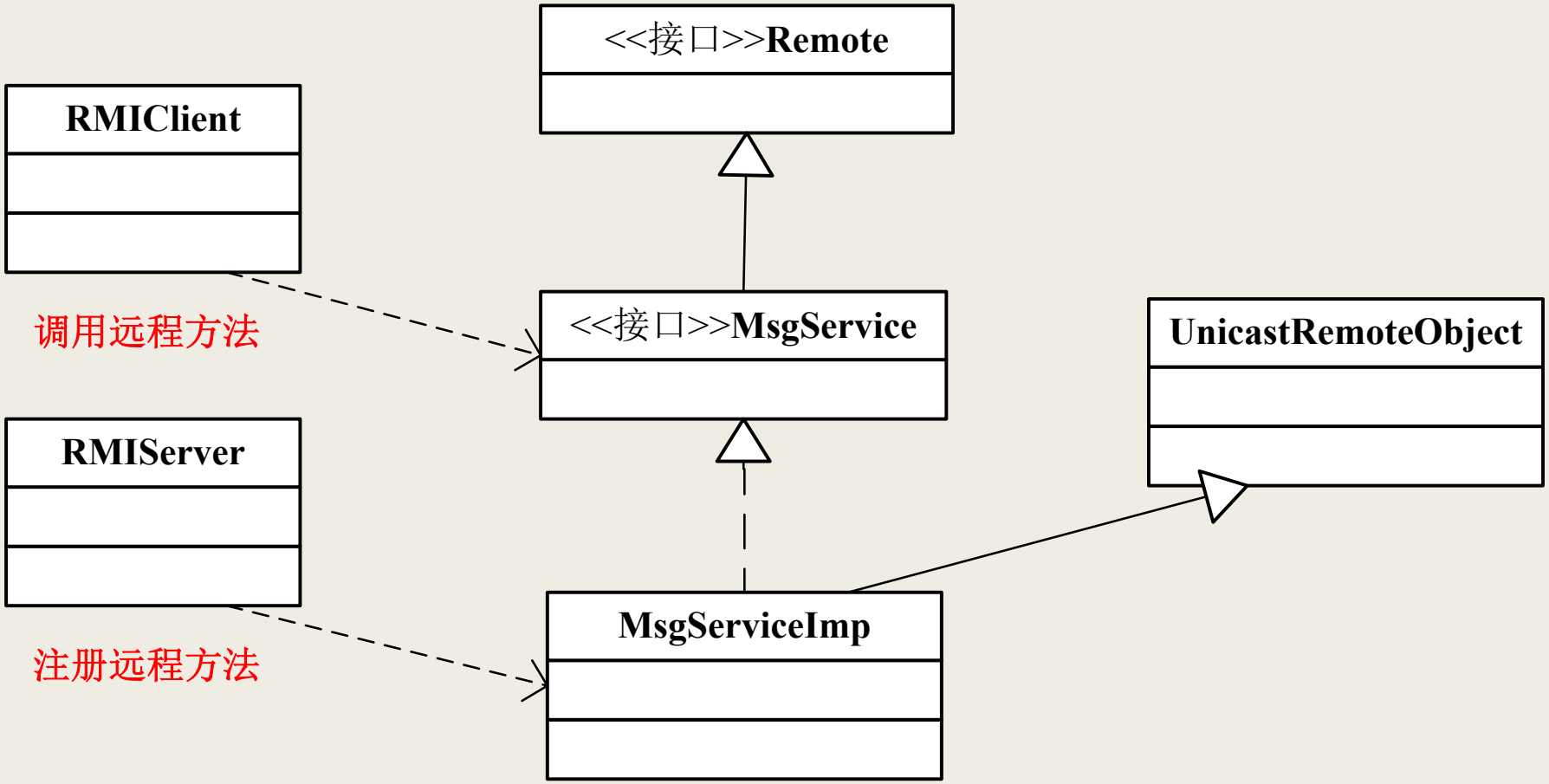
(3) Object lookup(String name)：返回name所绑定的对象。

(4) void unbind(String name):注销对象。

Name的完整名字URL表示：rmi://主机名:1099/XXX

注意：默认情况下rebind()会把远程对象注册到本地主机上监听1099端口的rmiregistry进程中。

简单的RMI应用实现


<<接口>>Remote
RMIClient
调用远程方法
<<接口>>MsgService
UnicastRemoteObject
RMIServer
注册远程方法
MsgServiceImp

RMI应用的开发步骤标准套路流程

创建一个RMI应用包括以下步骤：

(1) 创建远程接口：直接或间接继承java.rmi.Remote接口

Remote接口中不包含任何抽象方法，仅做标记使用，表明该接口的实现类对象能被远程主机所调用。

(2) 创建远程类：实现远程接口

(3) 创建服务器程序：负责在rmiregistry注册中心内注册远程对象。

(4) 创建客户程序：负责定位远程对象，并且调用远程对象的方法。

RMI应用-创建远程接口

RMI规范要求远程对象所属的类实现一个远程接口，远程接口必须符合以下条件：

(1) 直接或间接继承java.rmi.Remote接口

(2) 接口中的所有方法声明抛出java.rmi.RemoteException或RemoteException的父异常。

直接继承Remote接口的方式：

```
import java.rmi.*;
 public interface  XXXService extends Remote {
  public XXX method1() throws RemoteException;
  public XXX  method2() throws RemoteException;
}
```

RMI应用-创建远程接口

间接继承Remote接口

(2) public interface A {

```
public XXX method1() throws Exception;
public XXX  method2() throws IOException; }
public interface B extends A,Remote { }
```

此方式的优点：

1) A接口无需修改，只要创建一个继承Remote接口的子接口B，就能增加对RMI的支持。

2) A不依赖于RMI，能隐藏系统中的RMI细节，使得系统可以在保持接口A不变的情况下，灵活地更改实现细节。

RMI应用-创建远程类

远程类必须实现一个远程接口，还需要将远程类的对象导出为远程对象。

(1) 远程类继承java.rmi.server.UnicastRemoteObject类，并且构造方法必须声明抛出RemoteException。

UnicastRemoteObject构造方法的内容:

```
protected UnicastRemoteObject() throws RemoteException
   {  this(0); }
protected UnicastRemoteObject(int port) throws RemoteException
   { this.port = port;    exportObject((Remote) this, port);  }

public static Remote exportObject(Remote obj, int port) throws RemoteException
```

RMI应用-创建远程类

(2) 如果一个类已经继承了其他类，无法继承UnicastRemoteObject类，可以在其构造方法中显示调用静态方法exportObject()方法。声明抛出RemoteException。

```
class XXX extends OtherClass implements RemoteXXX
{
   public XXX() throws RemoteException
   {
      ........;
      UnicastRemoteObject.exportObject(this,0);
   }
}
```

(3) 在服务器程序中直接调用UnicastRemoteObject.exportObject(),把一个实现了远程接口的类的对象导出为远程对象。

RMI应用-创建服务端程序

服务端程序的主要任务:

(1) 生成远程对象

(2) 将该对象绑定到注册中心里

RMI应用-创建客户端程序

客户端程序的主要任务:

(1)通过JNDI在注册中心内查找要访问的远程对象的存根

(2)调用存根对象的方法(远程对象的方法)

优势和劣势

简单

主要问题

- 基于java
- 注册中心
- 效率和性能
- 安全性
- 灵活性和扩展性

常用于内部局域网的java平台