物联网中间件

第4章 分布式面向对象组件——RMI

参考材料

- https://www.javatpoint.com/RMI
- https://www.tutorialspoint.com/java_rmi/java_rmi_introduction.htm

什么是RMI

- Java RMI (Java Remote Method Invocation) Java远程方法调用，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，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式操作。

- RMI是Enterprise Java Beans的支柱，是建立分布式Java应用程序的方便途径。

什么是RMI

- RMI宗旨就是尽可能简化远程对象的使用。

- RMI允许运行在一个Java虚拟机的对象调用运行在另一个Java虚拟机上的对象的方法。通俗的讲：A机器上面有一个class，通过远程调用，B机器调用这个class 中的方法。

- 这两个虚拟机可以是运行在相同计算机上的不同进程中，也可以是运行在网络上的不同计算机中。

工作原理（原则）

- RMI的基础是接口，RMI构架基于一个重要的原理（原则）：定义接口和接口的具体实现是分开的。

- 基于该原则对象的使用和对象的创建相分离，我们在服务器端定义类，定义开放哪些方法，并创建对象；在客户端调用对象的方法。

- RMI使任务可以在最适合该任务的机器上完成。

RMI开发步骤

- 1、定义一个远程接口（远程接口必须继承接口，每个方法必须抛出远程异常，方法参数和方法返回值都必须是可序列化的）
- 2、实现远程接口
- 3、定义使用远程对象的客户程序
- 4、产生远程访问对象的桩和框
- 5、注册远程对象
- 6、运行服务器和客户程序

RMI包含部分：

- 远程服务的接口定义
- 远程服务接口的具体实现
- 桩（Stub）和框架（Skeleton）文件
- 一个运行远程服务的服务器
- 一个RMI命名服务，它允许客户端去发现这个远程服务
- 一个需要这个远程服务的客户端程序

RMI的用途

- RMI的用途是为分布式Java应用程序之间的远程通信提供服务，提供分布式服务。目前主要应用封装在各个J2EE项目框架中，例如Spring，EJB（Spring和EJB均封装了RMI技术）

JRMP

- RMI目前使用Java远程消息协议JRMP（Java Remote Messaging Protocol）进行通信。

- Java远程消息协议是专为Java语言定制的基于流的协议，主要用于查找和引用远程对象。该协议运行在TCP/IP协议之上。

- 由于是Java专有协议，JRMP要求客户端与服务器端都使用Java语言。

RMI的局限

- 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（意思是只支持客户端和服务器端都是Java程序的代码的远程调用，由于客户机和服务器都是使用Java编写的，二者平台兼容性的要求仅仅是双方都运行在版本兼容的Java虚拟机上）。

RMI调用远程方法的参数和返回值

- 当调用远程对象上的方法时，客户机除了可以将原始类型的数据作为参数以外，还可以将对象作为参数来传递

- 与之相对应的是返回值，可以返回原始类型或对象，这些都是通过Java的对象序列化（serialization）技术来实现的。（换而言之：参数或者返回值如果是对象的话必须实现Serializable接口）

远程接口满足下列要求：

- 远程接口必须直接或间接扩展（继承）Java.rmi.Remote接口，且必须声明为public

- 在远程接口中的方法在声明时，除了要抛出与应用程序有关异常之外，还必须包括RemoteException（或它的超类，IOExcepion或Exception）异常

例子

```
import java.rmi.*;
public interface Adder extends Remote{
    public int add(int x,int y)throws RemoteException;
}
```

服务器端

- 实现远程接口
- 实现远程接口的远程对象需要继承UnicastRemoteObject类，或者使用UnicastRemoteObject类的exportObject() 方法创建stub

- UnicastRemoteObject 是RemoteObject抽象类的一个子类，它定义了一个单播远程对象，该对象引用仅在服务器对应进程运行时可用
- RemoteObject抽象类
  - 实现了Remote接口和序列化Serializable接口，它和它的子类提供RMI服务器函数。

```
import java.rmi.*;
import java.rmi.server.*;
public class AdderRemote extends UnicastRemoteObject implements Adder{
    AdderRemote()throws RemoteException{
        super();
    }
    public int add(int x,int y){
        return x+y;
    }
}
```

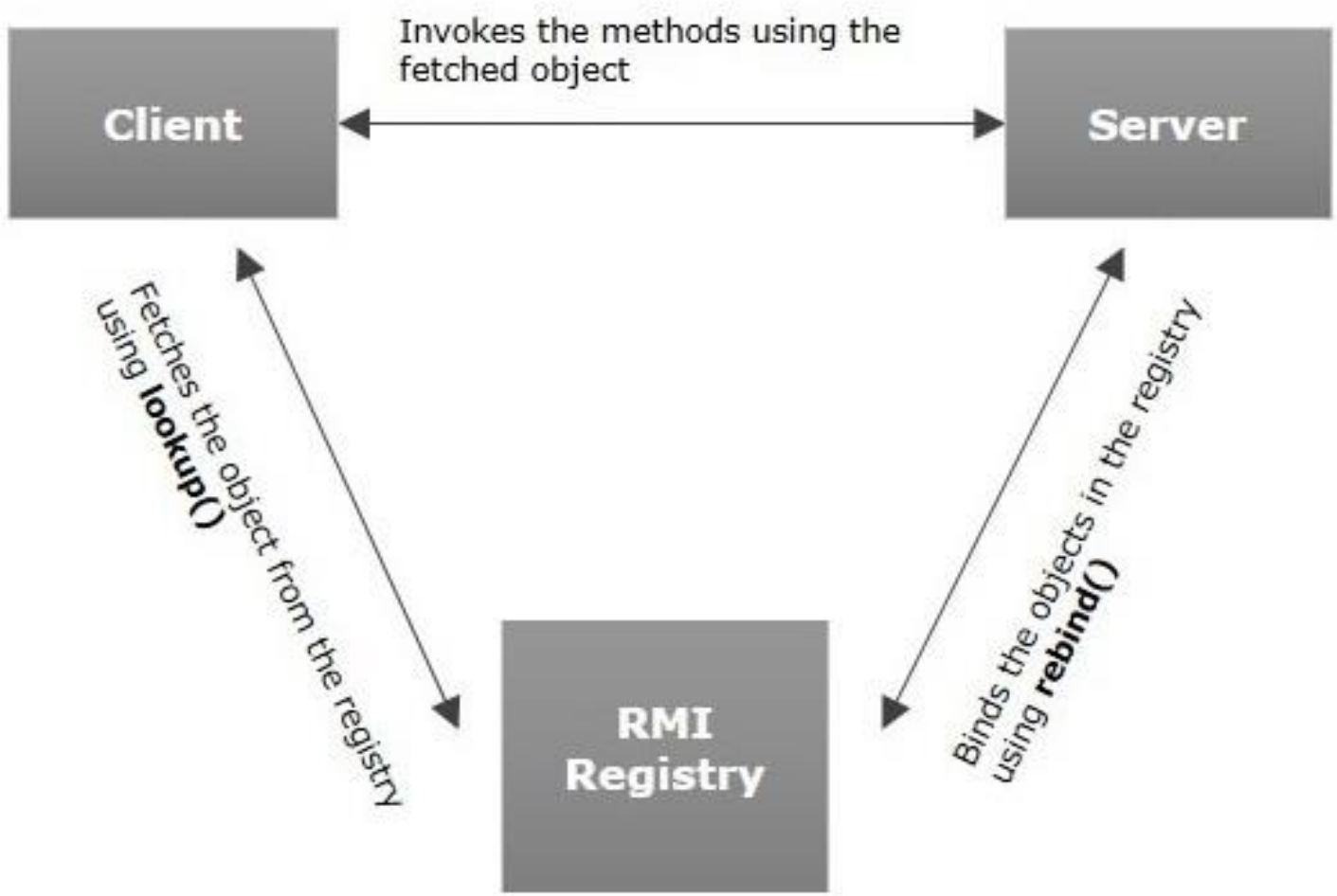
Registry(注册表)

- Registry(注册表、注册中心)是放置所有服务器对象的命名空间。
- 每次服务端创建一个对象时，它都会使用bind()或rebind()方法注册该对象。

- bind(String name, Remote obj) 绑定对此注册表中指定 name 的远程引用。
- name： 与该远程引用相关的名称, 称为绑定名称的唯一名称
- obj： 对远程对象（通常是一个 stub）的引用

Registry(注册表)

- 要调用远程对象，客户端需要该对象的引用
- 即通过服务端绑定的名称从注册表中获取对象(lookup()方法)


Client
Invokes the methods using the fetched object
Server
Fetches the object from the registry using lookup()
RMI Registry
Binds the objects in the registry using rebind()

Registry

- Server 是提供服务的地方，真正实现约定接口的地方；
- Client 是你的调用方，需要使用服务的一方。

- 通常情况下 Registry 可以和 Server 在同一个 JVM 里

- public static registry createregistry(int port)
- throws remoteexception
- 创建并导出接受指定 port 请求的本地主机上的 registry 实例。

- 参数:
- port - 注册表在其上接受请求的端口
- 返回:
- 注册表

```
import java.rmi.*;
import java.rmi.registry.*;
public class MyServer{
    public static void main(String args[]){
        try{
            Adder stub=new AdderRemote();
           Registry registry = LocateRegistry.createRegistry(5000);
            String name = "add";
            registry.bind(name, stub);
        }catch(Exception e){System.out.println(e);}
    }
}
```

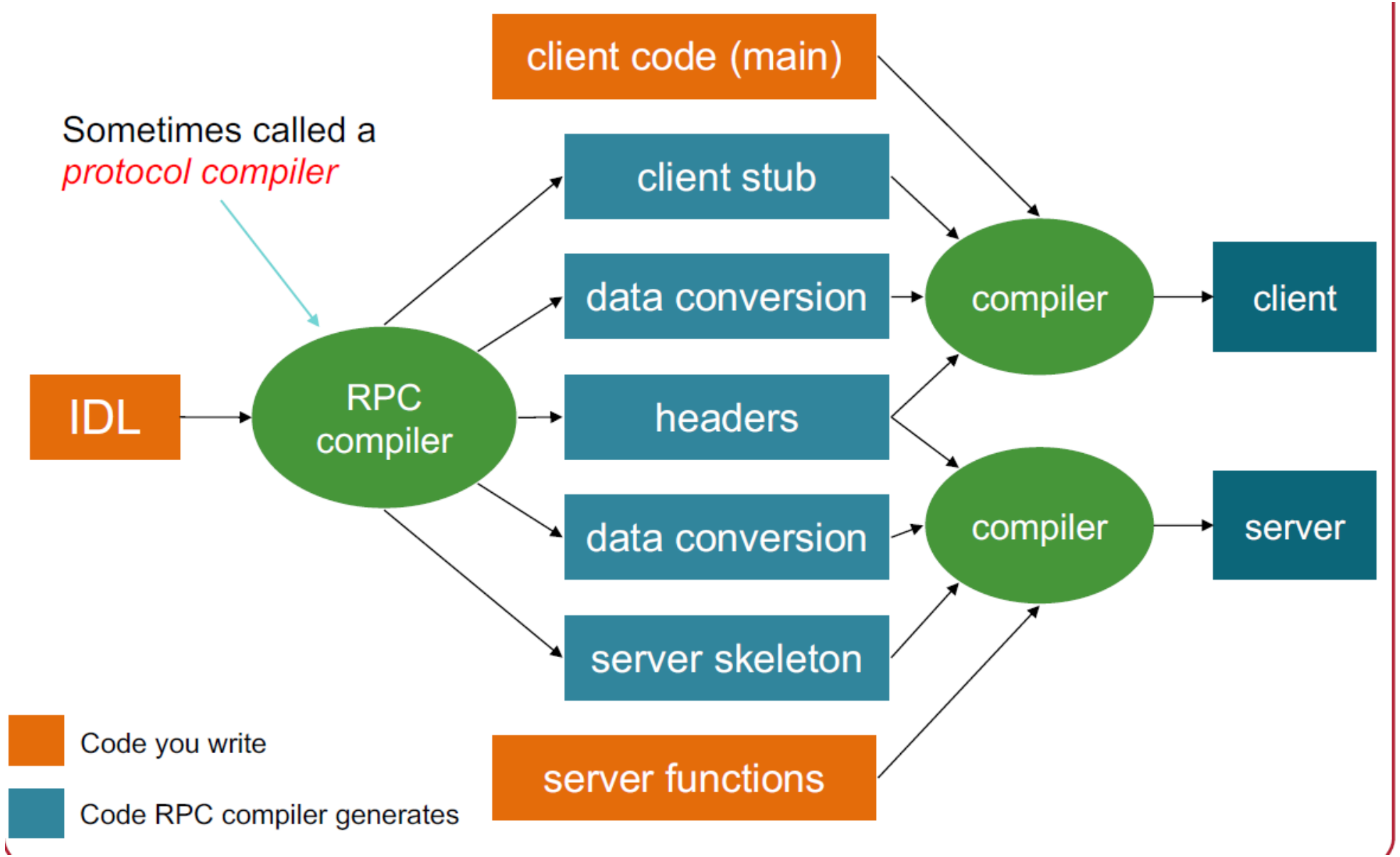
客户端

- Client 要想使用 远程接口中的方法，必须通过 ip 和 port 获得 注册中心
- 然后在注册中心中找到对应的远程对象
- 之后便可以像调用本地方法一样调用远程方法（底层会通过网络，到达远程服务方，服务方执行成功后返回结果给客户端）：

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

public class MyClient{
    public static void main(String args[]){
        try{
            String name = "add";
            Registry registry = LocateRegistry.getRegistry("localhost", 5000);
            Adder stub = (Adder) registry.lookup(name);
            System.out.println("X+Y: "+stub.add(34,4));
        }catch(Exception e){}
    }
}
```

演示


client code (main)
Sometimes called a
protocol compiler
client stub
data conversion
compiler
client
IDL
RPC
compiler
headers
compiler
server
data conversion
server skeleton
server functions
Code you write
Code RPC compiler generates

答疑

Java Socket 编程

- 套接字使用TCP/IP提供了两台计算机之间的通信机制。 客户端程序创建一个套接字，并尝试连接服务器的套接字。

- 当连接建立时，服务器会创建一个 Socket 对象。客户端和服务器现在可以通过对 Socket 对象的写入和读取来进行通信。

- java.net.Socket 类代表一个套接字，并且 java.net.ServerSocket 类为服务器程序提供了一种来监听客户端，并与他们建立连接的机制。

使用套接字建立TCP连接步骤

- 服务器实例化一个 ServerSocket 对象，表示通过服务器上的端口通信。
- 服务器调用 ServerSocket 类的 accept() 方法，该方法将一直等待，直到客户端连接到服务器上给定的端口。
- 服务器正在等待时，一个客户端实例化一个 Socket 对象，指定服务器名称和端口号来请求连接。

- Socket 类的构造函数试图将客户端连接到指定的服务器和端口号。如果通信被建立，则在客户端创建一个 Socket 对象能够与服务器进行通信。

- 在服务器端， accept() 方法返回服务器上一个新的 socket 引用，该 socket 连接到客户端的 socket。

- 连接建立后，通过使用 I/O 流在进行通信，每一个socket都有一个输出流和一个输入流，客户端的输出流连接到服务器端的输入流，而客户端的输入流连接到服务器端的输出流。

- TCP 是一个双向的通信协议，因此数据可以通过两个数据流在同一时间发送.

- 以下是一些类提供的一套完整的有用的方法来实现 socket。

InetAddress 类的方法

- 这个类表示互联网协议(IP)地址

序号	方法描述
1	**static InetAddress getByAddress(byte[] addr)** 在给定原始 IP 地址的情况下，返回 InetAddress 对象。
2	**static InetAddress getByAddress(String host, byte[] addr)** 根据提供的主机名和 IP 地址创建 InetAddress。
3	**static InetAddress getByName(String host)** 在给定主机名的情况下确定主机的 IP 地址。
4	**String getHostAddress()** 返回 IP 地址字符串（以文本表现形式）。
5	**String getHostName()** 获取此 IP 地址的主机名。
6	**static InetAddress getLocalHost()** 返回本地主机。
7	**String toString()** 将此 IP 地址转换为 String。

ServerSocket 类的方法

- 服务器应用程序通过使用 java.net.ServerSocket 类以获取一个端口,并且侦听客户端请求。

ServerSocket 类有四个构造方法：

序号	方法描述
1	**public ServerSocket(int port) throws IOException** 创建绑定到特定端口的服务器套接字。
2	**public ServerSocket(int port, int backlog) throws IOException** 利用指定的 backlog 创建服务器套接字并将其绑定到指定的本地端口号。
3	**public ServerSocket(int port, int backlog, InetAddress address) throws IOException** 使用指定的端口、侦听 backlog 和要绑定到的本地 IP 地址创建服务器。
4	**public ServerSocket() throws IOException** 创建非绑定服务器套接字。

POSIX套接字：listen() 函数 （服务器端）

- 如果作为一个服务器，在调用socket()、bind()之后就会调用listen()来监听这个socket。
  - int listen(int sockfd, int backlog);
- sockfd：即为要监听的socket描述字
- backlog为相应socket可以排队的最大连接个数

ServerSocket 类的常用方法

序号	方法描述
1	**public int getLocalPort()** 返回此套接字在其上侦听的端口。
2	**public Socket accept() throws IOException** 侦听并接受到此套接字的连接。
3	**public void setSoTimeout(int timeout)** 通过指定超时值启用/禁用 SO_TIMEOUT，以毫秒为单位。
4	**public void bind(SocketAddress host, int backlog)** 将 ServerSocket 绑定到特定地址（IP 地址和端口号）。

POSIX套接字：accept()函数 （服务器端）

- TCP客户端依次调用socket()、connect()之后就想TCP服务器发送了一个连接请求。TCP服务器监听到这个请求之后，就会调用accept()函数取接收请求，这样连接就建立好了。之后就可以开始网络I/O操作了。
  - int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
- sockfd即为要监听的socket描述字
- addr为客户端的socket地址
- addrlen为客户端socket地址的长度
- **返回值是一个新的套接字描述符，可以把它理解成是一个客户端的socket,这个socket包含的是客户端的ip和port信息 。**

Socket 类的方法

- java.net.Socket 类代表客户端和服务器都用来互相沟通的套接字
- 客户端要获取一个 Socket 对象通过实例化
- 服务器获得一个 Socket 对象则通过 accept() 方法的返回值。

Socket 类有五个构造方法

序号	方法描述
1	**public Socket(String host, int port) throws UnknownHostException, IOException.** 创建一个流套接字并将其连接到指定主机上的指定端口号。
2	**public Socket(InetAddress host, int port) throws IOException** 创建一个流套接字并将其连接到指定 IP 地址的指定端口号。
3	**public Socket(String host, int port, InetAddress localAddress, int localPort) throws IOException.** 创建一个套接字并将其连接到指定远程主机上的指定远程端口。
4	**public Socket(InetAddress host, int port, InetAddress localAddress, int localPort) throws IOException.** 创建一个套接字并将其连接到指定远程地址上的指定远程端口。

序号	方法描述
1	**public void connect(SocketAddress host, int timeout) throws IOException** 将此套接字连接到服务器，并指定一个超时值。
2	**public InetAddress getInetAddress()** 返回套接字连接的地址。
3	**public int getPort()** 返回此套接字连接到的远程端口。
4	**public int getLocalPort()** 返回此套接字绑定到的本地端口。
5	**public SocketAddress getRemoteSocketAddress()** 返回此套接字连接的端点的地址，如果未连接则返回 null。
6	**public InputStream getInputStream() throws IOException** 返回此套接字的输入流。
7	**public OutputStream getOutputStream() throws IOException** 返回此套接字的输出流。
8	**public void close() throws IOException** 关闭此套接字。

演示

套接字、远程过程调用、分布式面向对象组件中间件总结

- POSIX套接字（重要）
  - PPT：ch2.pdf
  - Linux系统提供的TCP/IP通信接口
  - 网络编程的基础，后续很多更高级的中间件都是对套接字进行的封装
  - 要求熟悉POSIX套接字当中各个函数的功能，以及整个套接字的通信流程，以及TCP建立连接和断开连接在套接字当中的体现

套接字、分布式面向对象组件中间件总结

- JAVA套接字（了解）
  - PPT：ch4_RMI.pdf
  - JAVA对操作系统套接字接口的封装
  - 了解JAVA套接字与POSIX接口的异同

套接字、分布式面向对象组件中间件总结

- 远程过程调用（重要）
  - Ppt：ch3.pdf
  - 熟悉远程过程调用的流程
  - 熟悉远程过程调用程序实现的流程

套接字、分布式面向对象组件中间件总结

- CORBR（了解）
  - Ppt：ch4_CORBA.pdf
  - 了解CORBA中间件的设计理念以及特性

套接字、分布式面向对象组件中间件总结

- JAVA RMI
  - Ppt：ch5_RMI.pdf

答疑