网络编程

李玮玮

讲授思路

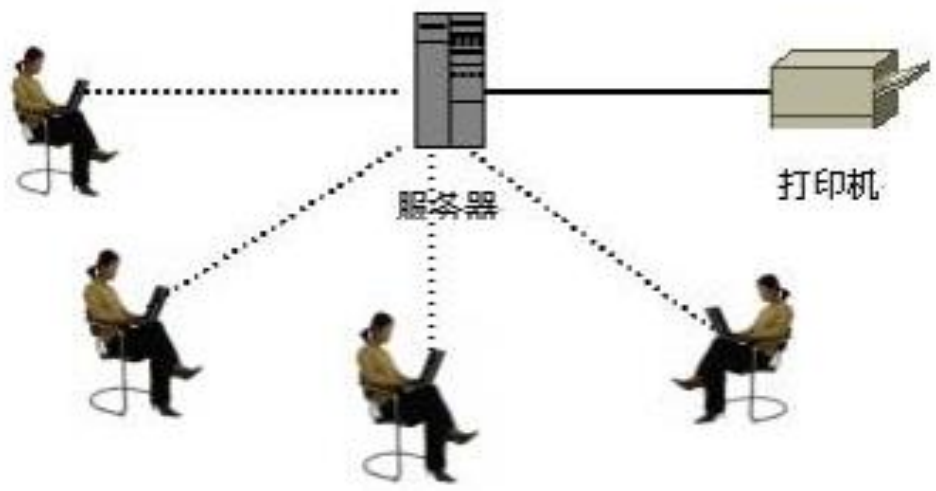
- 网络编程基础
- 基于套接字的Java网络编程

讲授思路-网络编程基础

- TCP/IP基本概念
- URL及应用

网络基础：计算机网络

- 计算机网络：通过一定的物理设备将处于不同位置的计算机连接起来组成的网络。
  - 网络最主要的作用在于共享设备和传输数据。



  - 无论是共享或传输数据，务必需要保证准确地匹配目的主机。

网络基础：IP地址和域名

- 为了准确地定位网络上的目标主机，网络中的每个设备都会有一个唯一的数字标识，即网络设备的IP地址。
  - 通过IP地址，可以精确地匹配目标主机，是网络中资源共享、数据传输的依据。
  - 例如：欲查找当前局域网内打印机，可以通过其IP地址10.7.10.200精确匹配。
- 由于IP地址不易记忆，引入网络域名来确认IP地址。
  - 例如：域名www.baidu.com相对于119.75.218.77来说，更容易记忆。

网络基础：端口

- IP地址可以精确地确定一台主机，但是在这台主机上可能运行着多个应用程序；可以借助主机端口精确地确定客户访问的是这台主机中的哪一个应用程序。
  - 在一台主机上，应用程序可以占用任何一个端口号；一旦应用程序占据这个端口号，其它应用将不能再占用该端口。
  - 在主机中，端口号1~1024是系统保留端口号，用来为常用的网络服务程序所占用。用户自定义应用程序，最好占用其它端口号。
    - 例如：HTTP服务默认占用80端口，FTP服务占用21端口，SMTP服务占用25端口等。

网络基础：TCP/UDP协议

- 确定好目标主机和应用程序之后，就可以进行网络传输。网络传输过程中，数据的传递有两种最常见的形式。
  - TCP传输控制协议，是一种面向连接的、可靠的、基于字节流的传输层通信协议。
    - 需要首先在网络两端建立安全连接，再进行数据传递，确保网络双方完整无误地传输数据。
  - UDP用户数据报协议，是一种无连接的传输层协议，提供面向事务的简单不可靠信息传送服务。
    - 无需建立网络双方连接，直接发送数据包（包含目的地址信息），可能会因为网络问题导致数据传输失败等问题，但是传输速度很快，常用于局域网中传输数据。

网络编程简介

- 网络编程是指通过编程方式实现两个（或多个）设备之间的数据传输。
  - 网络编程是基于“请求-响应”模式的：网络中某一端发出请求，另一端接收到请求后，响应请求方的请求。
  - “请求方”称之为客户端，“响应方”称之为服务器端。
  - 网络编程在客户端和服务器端之间传输数据可以采用TCP方式，也可以采用UDP方式。
- 网络编程开发模式
  - 客户端/服务器端模式（C/S模式）：对于不同的服务器端程序建立不同的客户端程序。
  - 浏览器/服务器端模式（B/S模式）：对于不同的服务器端程序使用统一的“客户端”（即浏览器）即可。

网络编程简介：C/S模式应用程序

- 在网络编程中，C/S模式应用程序的开发，需要同时开发客户端应用程序和服务器端应用程序。
  - 客户端应用程序开发步骤：
    - 客户端建立与服务器端的连接（通过IP地址和端口确定服务器端程序）
    - 客户端封装请求数据，发送给服务器端；客户端获得服务器端响应数据，解析并处理数据
    - 客户端关闭网络连接

网络编程简介：C/S模式应用程序

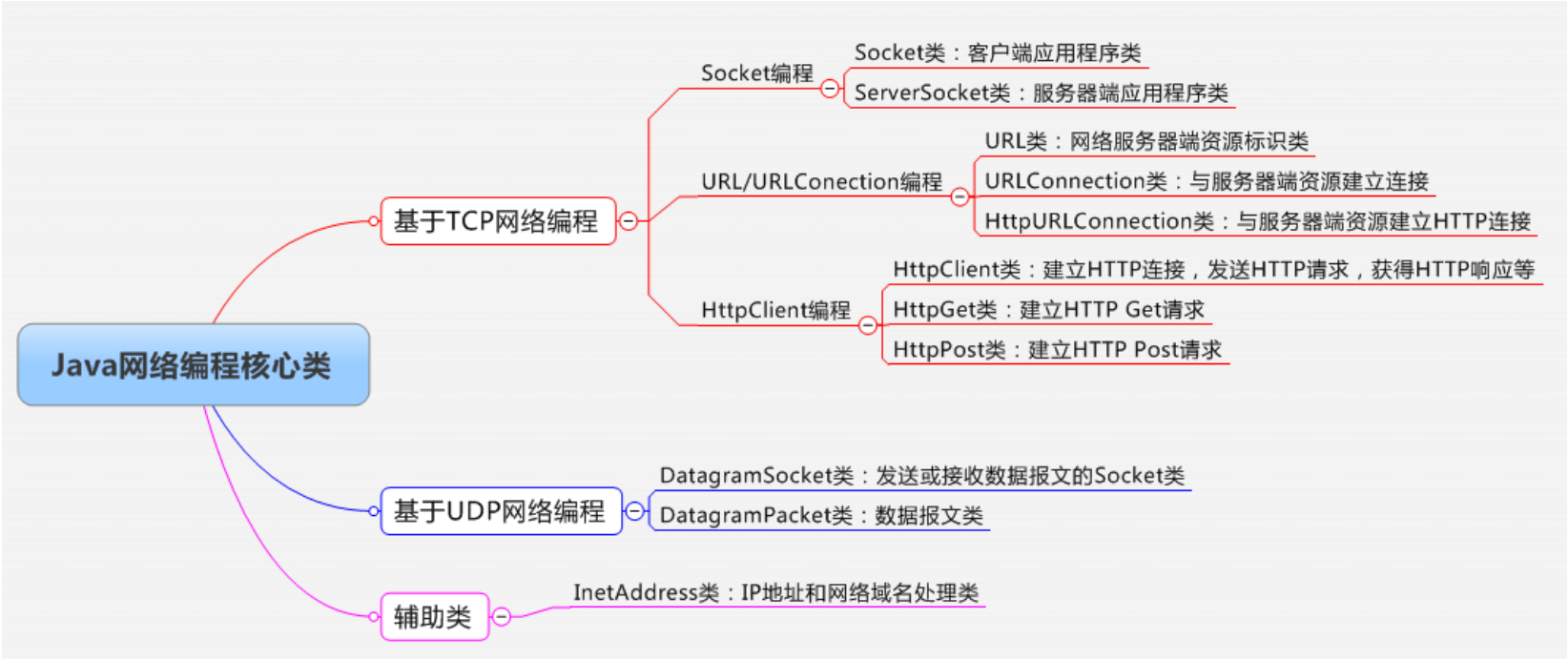
- 在网络编程中，C/S模式应用程序的开发，需要同时开发客户端应用程序和服务器端应用程序。
  - 服务器端应用程序开发步骤：
    - 服务器端监听特定端口。
    - 服务器端接收客户端连接。
    - 服务器端接收客户端请求数据，解析并处理请求数据；服务器端封装响应数据，发送给客户端端。
    - 服务器端关闭网络连接。

网络编程简介：B/S模式应用程序

- B/S模式应用程序的开发，由于客户端统一使用浏览器访问，只需要开发服务器端应用程序即可。
  - 由于客户端使用浏览器访问，服务器端应用程序本质上属于Web应用程序；浏览器和服务器通信协议采用HTTP协议。
  - Web应用程序的开发过程在后续课程中详细介绍，在此不再赘述。

Java网络编程核心类

- Java语言中，实现网络编程需要使用两个核心类包。
  - java.net.*：网络类包，涵盖常用网络操作类。
  - java.io.*：数据消息传输包，在网络双方进行数据传递需要使用该包中的类。



URL及应用

- 使用Socket进行网络编程，从网络传输层角度进行分析，适用于绝大多数网络连接方式；但是需要开发人员熟练掌握网络传输、网络协议等基础知识，开发难度较大。
- 在Java语言中，提供了一组URL处理类，封装了Socket编程技术的实现细节，用来方便开发人员进行网络连接、网络数据传输等常用的网络任务。
  - 直接使用URL类，可以方便地处理各种常见协议的网络连接。
  - 使用URL类进行网络连接，相当于直接进行远程输入/输出流操作，只需开发人员熟练掌握常用IO操作即可，大大降低开发难度。

URL网络编程核心操作类

- URL类：统一资源定位符，指向互联网“资源”的指针。
  - 常用构造方法：
    - URL(String url); // 通过给定字符串建立URL对象
  - 常用方法：
    - InputStream openStream();// 打开当前URL连接的输入流
    - URLConnection openConnection(); // 建立URL网络连接
  - 详细请查看：
    http://docs.oracle.com/javase/7/docs/api/java/net/URL.html

URL网络编程核心操作类

- URL类：

```
URL myURL = new URL("http://java.sun.com");

String protocal = myURL.getProtocol();
String host = myURL.getHost();
String file = myURL.getFile();
int port = myURL.getPort();
String ref = myURL.getRef();

System.out.println(protocal + ", " + host + ", " + file + ", " + port + ", " + ref);
```

URL网络编程核心操作类

- URLConnection类：应用程序和 URL 之间的通信链接，用于读取和写入此 URL 引用的资源。
  - 对象建立方法：
    - 通过URL对象的openConnection()方法创建
    - 使用构造方法：URLConnection(URL url);
  - 常用方法：
    - 获得响应消息头类方法：getContentType()、getContentLength()、getContentEncoding()、……
    - 获得响应消息主体：getContent()
    - 获得当前连接输入/输出流对象：getInputStream()、getOutputStream()
  - 具体查看：http://docs.oracle.com/javase/7/docs/api/java/net/URLConnection.html

URL网络编程核心操作类

- URLConnection类：

```
url = new URL("http://");
// 打开连接
URLConnection conn = url.openConnection();
// 得到输入流
InputStream is = conn.getInputStream();
// 关于IO流的用法和写法一定要熟悉
OutputStream os = new FileOutputStream("d:\\baidu.png");

byte[] buffer = new byte[2048];
int length = 0;

while (-1 != (length = is.read(buffer, 0, buffer.length))) {
os.write(buffer, 0, length);
}
is.close();
os.close();
```

URL网络编程核心操作类

- HttpURLConnection类：特定支持HTTP协议的URLConnection。
  - 对象建立方法：
    - 通过URL对象的openConnection()方法创建，强制转换为目标对象
    - 使用构造方法：HttpURLConnection(URL url);
  - 常用方法：
    - 从URLConnection类继承的方法
    - 针对HTTP请求响应消息的特定方法：getRequestMethod()、setRequestMethod()、getResponseCode()、getResponseMessage()、……
  - 具体查看：http://docs.oracle.com/javase/7/docs/api/java/net/HttpURLConnection.html

URL网络编程实例：文件下载

- 下载服务器端文件，基本思路：
  - 创建URL对象：URL url = new URL(文件地址);
  - 获取服务器端输入流：InputStream is = url.openStream();
  - 文件读写：从输入流中读取字节写入到输出流（文件）中。

```
String sUrl = "http://pic42.nipic.com/20140608/
                                12504116_194242259000_2.jpg";
URL url = new URL(sUrl);          // 创建URL对象
InputStream in = url.openStream();        // 获得网络输入流
// 创建文件输出流
FileOutputStream out = new FileOutputStream(“cat.jpg");
int b;
while ((b = in.read()) != -1) {
        out.write(b);                  // 写入文件
}
// 关闭输入输出流
out.close();    in.close();
```

URL网络编程实例：获取响应信息

- 获取服务器HTTP响应消息（消息头和消息主体）。
  - 访问网址：http://software.hebtu.edu.cn/
  - 获取该网页的服务器字符编码、文档类型、服务器响应状态码、网页主体等。

```
URL url = new URL(rootUrl);            //  创建Url对象
// 得到URLConnection连接对象
URLConnection conn = url.openConnection();
HttpURLConnection hc = (HttpURLConnection) conn;
// 获得响应消息头
conn.getContentType();
conn.getContentLength();
conn.getContentEncoding();
// 获得HTTP消息状态码
hc.getResponseCode();
hc.getResponseMessage();
// 获得HTTP响应消息主体
hc.getContent();
```

讲授思路-基于套接字的Java网络编程

- Socket通信
- Socket通信的过程
- Socket基于TCP协议的网络编程
- Socket基于UDP协议的网络编程

Socket网络编程简介

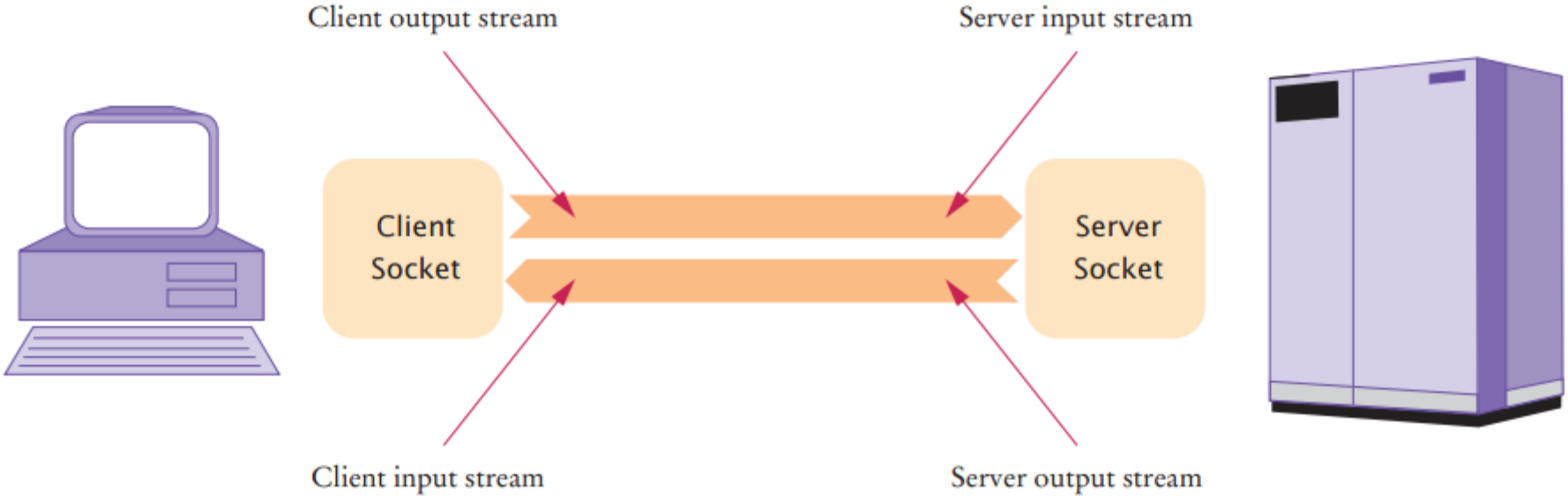
- 客户端和服务器端建立连接后，连接两端将会建立一个虚拟“线缆”，在网络编程中称之为Socket（套接字）；其后在网络两端传输数据都是通过Socket进行的。
  - Socket借助IP地址和端口号，对应某一台主机中的某一个应用程序。
  - Socket的主要作用是维护网络连接、在网络双方传输数据。

Socket网络编程核心操作类

- Socket类：客户端套接字类。实现客户端向服务器发送数据、接收服务器数据等功能；实现服务器端向客户端发送数据、接收客户端数据等功能。
  - 构造方法：
    - Socket(InetAddress address, int port);
    - Socket(String host, int port);
  - 常用方法：
    - getInputStream();// 获得网络输入流
    - getOutputStream();// 获得网络输出流
    - close();// 关闭Socket连接
  - 其它方法查看：http://docs.oracle.com/javase/7/docs/api/java/net/Socket.html

Socket网络编程核心操作类

- 客户端与服务器端通信时，借助网络输入/输出流进行传输；但是对于客户端和服务器而言，输入流或输出流是相对的。



客户端Socket应用程序

- 客户端Socket应用程序所做的工作主要有：
  - 与服务器端建立连接（通过IP和端口号确定主机上的程序）。
    - Socket client = new Socket(“localhost”，80);
  - 向服务器端发送数据，接收服务器端数据。
    - 向服务器端发送数据：os = client.getOutputStream();
    - 接收服务器端数据：is = client.getInputStream();
  - 关闭Socket连接。
    - client.close();

客户端Socket应用程序

- 示例：客户端Socket请求服务器内容。

```
Socket   client = new Socket("127.0.0.1", 8888);
// 获得网络输入/输出流
InputStream is = client.getInputStream(); // 获得网络输入流
OutputStream os = client.getOutputStream(); // 获得网络输出流
// 发送数据到server
String request = "this is a client request!";
os.write(request.getBytes());
os.flush();// 刷新请求
// 接收响应
byte[] b = new byte[1024];
StringBuffer strb = new StringBuffer();
while (is.read(b) != -1) {
strb.append(new String(b));
}
System.out.println(strb.toString());
// 关闭连接
is.close();
os.close();
client.close();
```

Socket网络编程核心操作类

- ServerSocket类：服务器端套接字类。监听服务器指定端口，接收客户端连接请求。
  - 构造方法：
    - ServerSocket(int port);
  - 常用方法：
    - accept();// 用于产生“阻塞”，直到接收一个连接，返回客户端Socket对象
    - close();// 关闭服务器端Socket监听
  - 其它方法参考：http://docs.oracle.com/javase/7/docs/api/java/net/ServerSocket.html

服务器端Socket应用程序

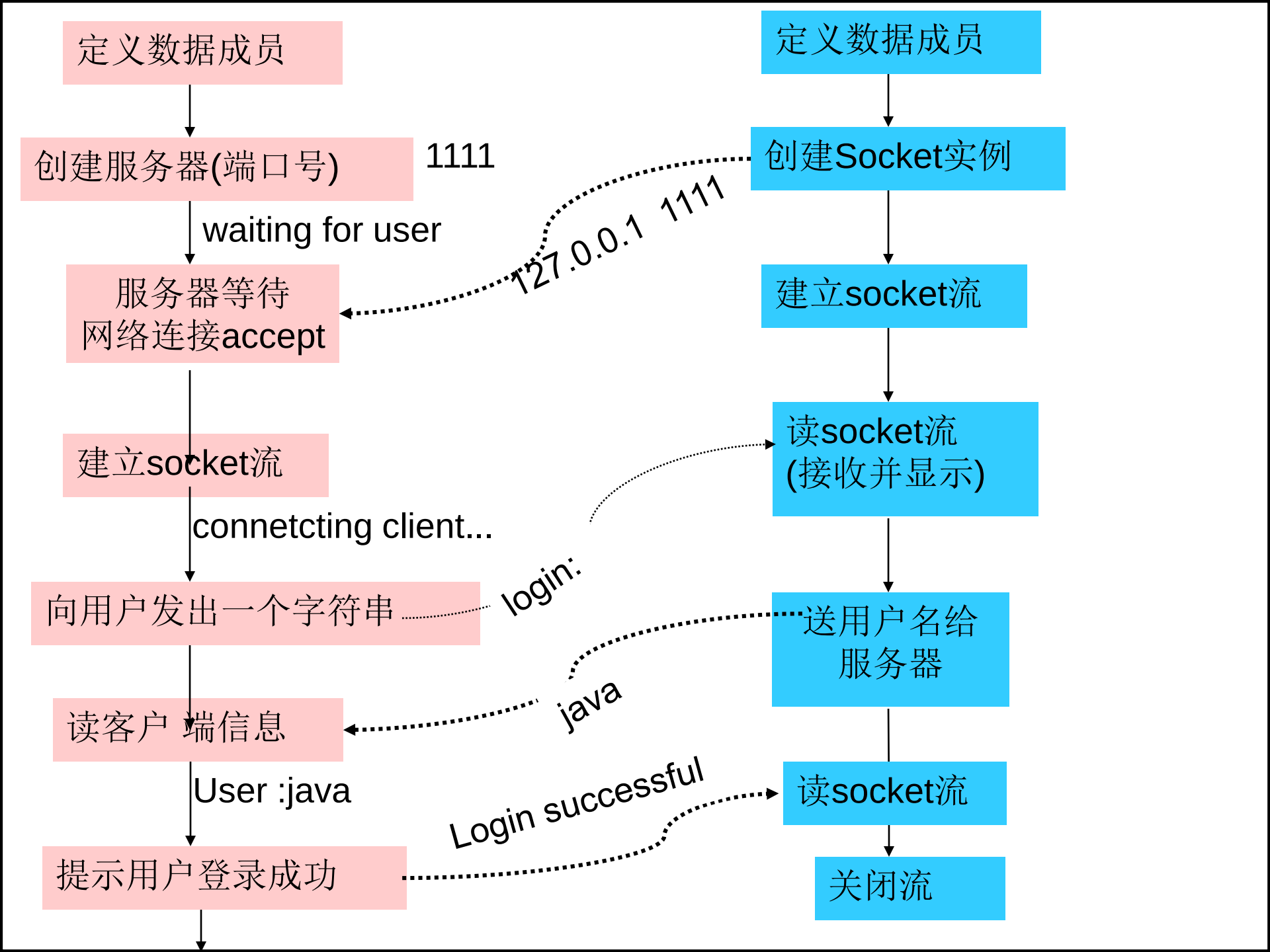
- 服务器端所做的主要工作有：
  - 监听特定端口。
    - ServerSocket server = new ServerSocket(8888);
  - 接收客户端连接。
    - Socket client = server.accept();
  - 接收客户端请求，向客户端发送响应。
    - 接收客户端请求数据：is = client.getInputStream();
    - 向客户端响应数据：os = client.getOutputStream();
  - 关闭连接。
    - 关闭客户端：client.close();
    - 关闭服务器端：server.close();

服务器端Socket应用程序

- 示例：建立服务器端程序，监听8888端口号，返回客户端请求数据。

```
server = new ServerSocket(8888);
// 接收客户端连接
System.out.println("server listener");
Socket client = server.accept();
// 获得客户端请求
InputStream is = client.getInputStream();
byte[] b = new byte[1024];
is.read(b);
System.out.println("Server received:" + new String(b));

// 向客户端发送响应
OutputStream os = client.getOutputStream();
os.write(("this is server return string !").getBytes());
// 关闭网络连接
is.close();
os.close();
client.close();
server.close();
```


定义数据成员
创建服务器(端口号)
1111
waiting for user
服务器等待
网络连接accept
建立socket流
connetcting client...
向用户发出一个字符串
读客户 端信息
User :java
提示用户登录成功
定义数据成员
创建Socket实例
127.0.0.1 1111
建立socket流
读socket流
(接收并显示)
login:
送用户名给
服务器
java
读socket流
Login successful
关闭流

单Socket客户端和单服务器端一次通讯

- ServerSocket建立后，通过accept来等待Client连接
- Client连接Server端
- Server端建立inputstream和outputstream
- Client端建立inputstream和outputstream
- 双方一次通讯
- 各自关闭自己的inputstream和outputstream

单服务器端接收多次通讯

- ServerSocket建立后，通过accept来等待Client连接
- Client连接Server端
- Server端建立inputstream和outputstream
- Client端建立inputstream和outputstream
- 双方一次通讯
- Client关闭
- Server端等待下次连接

单服务器端接收多次通讯

- 示例：建立服务器端程序，监听8888端口号，返回客户端多次请求数据。等待结束标记。

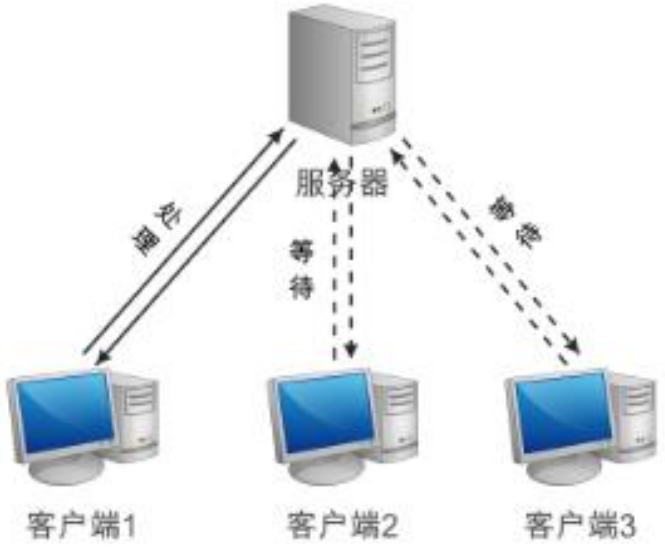
单服务器端接收多次通讯

- 示例 客户端多

```
server = new ServerSocket(8888);
while(true){
// 接收客户端连接
System.out.println("server listener");
Socket client = server.accept();
// 获得客户端请求
InputStream is = client.getInputStream();
byte[] b = new byte[1024];
is.read(b);
String result = new String(b);
System.out.println("Server received:" + result);
// 向客户端发送响应
OutputStream os = client.getOutputStream();
os.write(("this is server return string !").getBytes());
// 关闭网络连接
is.close();
os.close();
client.close();
if (result.startsWith("exit")) {
break;
}
}
server.close();
```

编程实例：聊天程序

- 要开发一款聊天程序，需要实现客户端与服务器端通信功能。
  - 服务器接收到一个客户端请求后要进行处理，若同时第2个客户端、第3个客户端向服务器发送请求，则其它客户端只能等待第1个客户端请求处理完毕后，才能有机会获得服务器处理。如何实现服务器并发处理多个客户端请求？



多线程网络编程简介

- 在客户端，可以把处理服务器端响应消息的任务放到一个单独线程中，在主线程中接收用户输入和发送请求消息。
- 服务器的作用就是为了服务多个客户端，对于多个客户端并发请求的处理，在网络编程中，采用多线程处理方式解决。每当服务器端接收到客户端请求后，产生一个新的线程处理客户端请求。
  - 开启线程比较消耗系统资源，所以应用程序中可以开辟的线程个数总是有限的。
  - 可以结合使用线程池技术，实现服务器端的并发处理请求。

单服务器端多线程接收多次通讯

- ServerSocket建立后，通过accept来等待Client连接
- Client连接Server端
- Server开启一个新的线程去处理
- Server端建立inputstream和outputstream
- Client端建立inputstream和outputstream
- Client关闭
- Server端等待下次连接

单服务器端多线程接收多次通讯

- 示例：服务器端开启新的线程接收客户端请求。

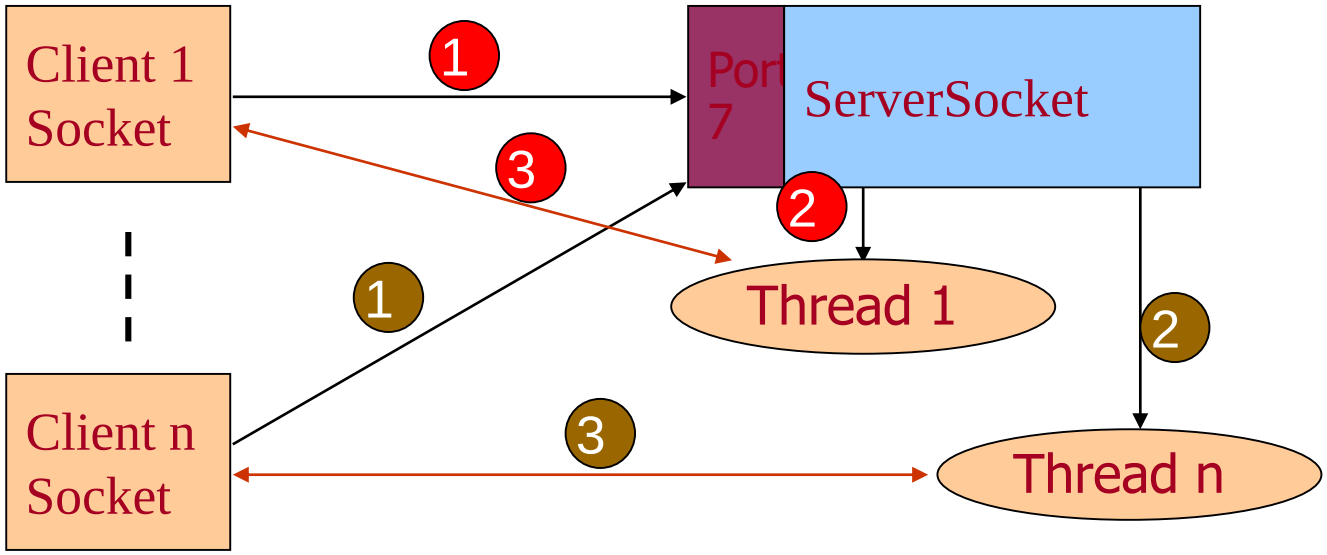
```
public class MyRunnable implements Runnable {
private Socket socket;

public MyRunnable(Socket socket) {
this.socket = socket;
}

public void run() {
        //处理接收请求返回数据….
}
}
```

多线程网络编程实现方法

- 服务器端程序执行过程
  - 主线程监听客户端连接，线程处理客户端请求。



客户端端多线程接收网络响应

- 客户端程序示例：
  - 开辟新线程监听服务器端响应，主线程接收用户输入并向服务器发送请求。

```
// 建立网络连接
Socket client = new Socket("localhost", 8888);
// 启动线程，读取服务器端响应消息
new HandleServerResponseMessage(client).start();
// 接收用户键盘输入，发送到服务器端
BufferedReader in = new BufferedReader(
                        new InputStreamReader(System.in));
PrintWriter out = new PrintWriter(client.getOutputStream());
String msg = null;
while ((msg = in.readLine()) != null) {
            out.println(msg);
            out.flush();
}
```

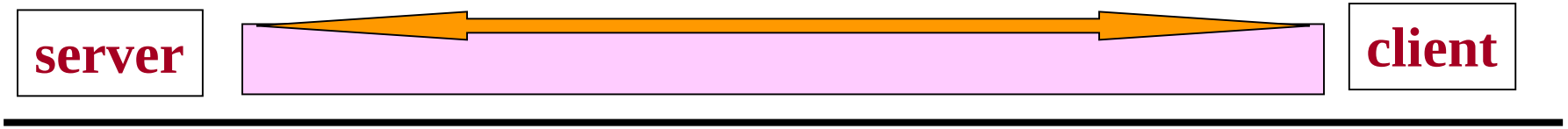
多线程网络编程实现方法

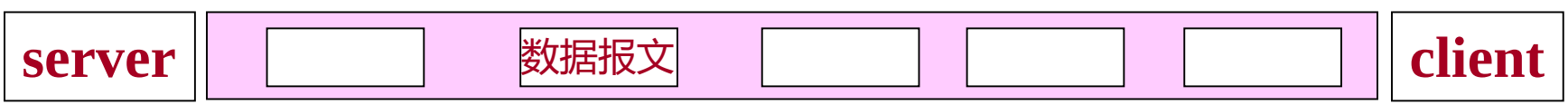
- 服务器端程序示例：

```
// 建立服务器端Socket
ServerSocket server = new ServerSocket(8888);
// 监听并处理客户端请求
while (true) {
// 接收客户端请求
Socket client = server.accept();
// 开启新的线程请求
new HandleClientRequestMessage(client).start();
}
```

UDP网络编程简介

- 建立网络连接时，有两种传输层协议（TCP传输协议和UDP传输协议）。
  - UDP传输协议：一种无连接的传输层协议，提供面向事务的简单不可靠信息传送服务。





UDP网络编程简介

- 建立网络连接时，有两种传输层协议（TCP传输协议和UDP传输协议）。
  - UDP传输协议：一种无连接的传输层协议，提供面向事务的简单不可靠信息传送服务。

比较项	TCP协议	UDP协议
连接性	网络双方需要建立连接	不需要网络双方建立连接
安全性	完全可靠，确保数据可达	不可靠，数据可能传输失败
传输速度	无限制	数据包大小不越过64K
传输方式	在连接的虚拟线路上传输	在网络中传输

UDP网络编程核心类

- UDP传输协议通过数据包方式向服务器发送数据，那么在数据包中肯定需要包含服务器的IP信息、端口信息等内容。因此，UDP网络编程必须提供以下对象来完成不同的任务：
  - 网络两端接收消息或发送消息的对象（监听本机端口、发送消息、接收消息）。
  - 数据包对象（包含目的地IP和端口信息、数据报文信息）。

UDP网络编程核心类

- DatagramSocket类：客户端/服务器端网络Socket端口对象。
  - 构造方法：
    - DatagramSocket ( ) ; // 创建一个空的Socket对象
    - DatagramSocket (int port) ; // 创建指定监听端口的Socket对象
  - 常用方法：
    - void send (DatagramPacket p) ; // 发送数据报文
    - void receive (DatagramPacket p) ; // 接收数据报文
  - 具体查看：http://docs.oracle.com/javase/7/docs/api/java/net/DatagramSocket.html

UDP网络编程核心类

- DatagramPacket类：数据报文对象。
  - 构造方法：
    - DatagramPacket (byte[] buf, int len)； // 用空数组创建对象，用来接收数据。
    - DatagramPacket (byte[] buf, int offset, int len)； // 接收数据的特定部分。
    - DatagramPacket (byte[] buf, int len, InetAddress addr, int port)； // 包含数据的数组创建对象，用来发送数据，同时指明数据目的地和目标端口号。
    - DatagramPacket (byte[] buf, int offset, int len, InetAddress addr, int port)； // 发送数据的指定部分。
  - http://docs.oracle.com/javase/7/docs/api/java/net/DatagramPacket.html

UDP网络编程实例

- 客户端程序：
  - 基本工作流程：
    - 创建DatagramSocket对象
    - 封装请求数据，创建DatagramPacket对象
    - 发送请求
  - 实例程序：

UDP网络编程实例

- 客户端程序：
  - 实例程序：

```
// 创建DatagramSocket对象
DatagramSocket client = new DatagramSocket();
// 准备请求数据
byte[] buf = “客户端请求数据".getBytes();
InetAddress address = InetAddress.getLocalHost();
// 创建DatagramPacket对象
DatagramPacket request =
            new DatagramPacket(buf, buf.length, address , 8888);
// 发送请求
client.send(request);
```

UDP网络编程实例

- 服务器程序：
  - 基本工作流程：
    - 创建DatagramSocket对象，监听特定端口
    - 创建DatagramPacket对象（空缓冲区）
    - 接收客户端请求
    - 封装服务器响应数据，创建DatagramPacket对象
    - 发送服务器响应给指定客户端
  - 实例程序：

UDP网络编程实例

- 服务器程序：
  - 实例程序：

```
// 创建DatagramSocket对象，监听特定端口
DatagramSocket server = new DatagramSocket(8888);
// 准备空缓冲区
byte[] buf = new byte[1024];
// 循环等待客户端请求
while (true) {
        // 创建DatagramPacket对象
        DatagramPacket request = new DatagramPacket(buf, buf.length);
        // 接收客户端请求
        server.receive(request);
        // 准备服务器端响应数据包
        byte[] resBuf = "from server: ".getBytes();
        DatagramPacket response = new DatagramPacket(
                resBuf, resBuf.length, request.getAddress(), request.getPort());
        // 发送服务器响应
        server.send(response);
}
```

总结

- 网络编程基础
  - TCP/IP基本概念
  - URL及应用
- 基于套接字的Java网络编程
  - Socket通信
  - Socket通信的过程
  - Socket基于TCP协议的网络编程
  - Socket基于UDP协议的网络编程

Thank You