



主讲教师 张 智
计算机学院软件工程系
课程群: 421694618

12 Java I/O和网络编程

12.1 Java I/O

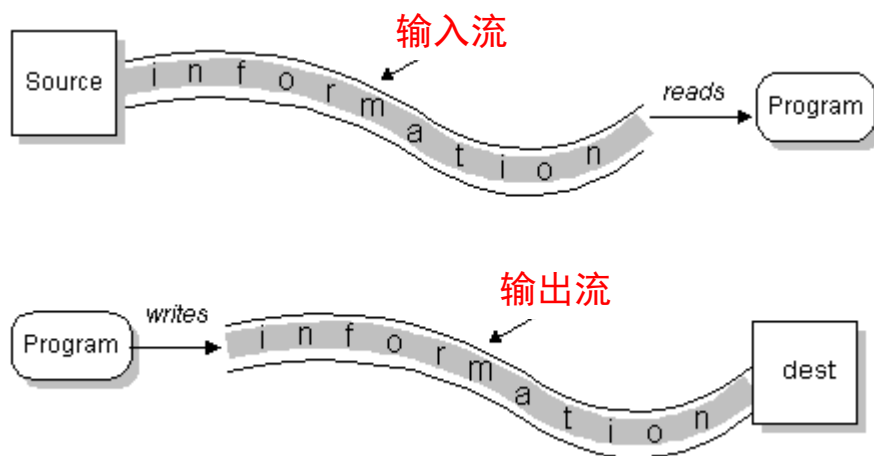
12.2 Java File类

12.3 Java网络编程

12.1 Java I/O

■ 流的概念

- 流是Java内存中的一组有序数据序列
- 输入流：从外部（文件、键盘、网络等）读入数据到内存
- 输出流：把数据从内存输出到外部（文件、控制台、网络等）



以内存为中心

Java的流分类

- 按照输入的方向(参照对象是Java程序): 输入流和输出流
- 按照处理数据的单位: 字节流和字符流
 - 字节流: 一般与机器直接交互的输入输出都是二进制字节流(byte)
 - 字符流: 通常是文本文件, 字符流最小数据单位是char
 - 二者主要区别在于每次读写的字节数不同
- 按照功能的不同分: 节点流和处理流
 - 节点流: 是直接从一个源读写数据的流 (这个流没有经过包装和修饰)
 - 处理流: 是在对节点流封装的基础上的一种流

字节流以byte为最小单位

I/O类结构

I/O流

字符流

Reader

BufferedReader

LineNumberReader

InputStreamReader

FileReader

StringReader

PipedReader

FilterReader

PushbackReader

Writer

BufferedWriter

OutputStreamWriter

FileWriter

PrintWriter

StringWriter

PipedWriter

CharArrayWriter

FilterWriter

字节流

InputStream

FileInputStream

FilterInputStream

BufferedInputStream

DataInputStream

PushbackInputStream

ObjectInputStream

PipedInputStream

SequenceInputStream

StringBufferInputStream

ByteArrayInputStream

OutputStream

FileOutputStream

FilterOutputStream

BufferedOutputStream

DataOutputStream

PrintStream

ObjectOutputStream

PipedOutputStream

ByteArrayOutputStream

如何理解I/O诸多类（常用的类）

- 不管流的分类是多么复杂，其根源来自于四个基本抽象类(基本流)：

	字节流	字符流
输入流	InputStream	Reader
输出流	OutputStream	Writer

- 字节流 ^{转换} → 字符流：

- InputStreamReader、OutputStreamWriter

缓冲流可获得更高的
读取效率(如按行读写)

- 以上类只是流的表示形式，在传输中还会再封装成缓冲流(处理流)：

	字节缓冲流	字符缓冲流
输入流	BufferedInputStream	BufferedReader
输出流	BufferedOutputStream	BufferedWriter

■ 对文件(File)的读/取操作：常用

	字节文件	字符文件
输入流	<code>FileInputStream</code>	<code>FileReader</code>
输出流	<code>FileOutputStream</code>	<code>FileWriter</code>

■ 高层次的对象数据流：对象信息的读写

	对象	说明
输入流	<code>ObjectInputStream</code>	对象反序列化(从文件中读取序列化对象)
输出流	<code>ObjectOutputStream</code>	对象序列化(将指定的对象写入文件)

BufferedReader常用方法

方法	说明
<code>int read()</code>	读取单个字符，如果已到达流末尾，则返回 -1
<code>int read(char cbuf[], int off, int len)</code>	将最多len个字符读入数组中(从off位置开始存放)，返回实际读入的字符个数，如果已到达流末尾，则返回-1
<code>String readLine()</code>	读一行文字并返回该行字符（不包含'\n'换行符和'\r'回车符），若读到文件末尾，则返回null
<code>void close()</code>	关闭流

BufferedWriter常用方法

方法	说明
<code>void write(int c)</code>	写入单个字符，c是指定要写入字符的int
<code>void write(String str)</code>	写入字符串
<code>void newLine()</code>	写入一个行分隔符
<code>void flush()</code>	刷新此输出流并强制写出所有缓冲的输出字节(不关闭流)
<code>void close()</code>	关闭流(在关闭流之前，会先刷新缓存区)

示例

- 读文件：读取文本文件并显示
- 写文件：将键盘输入的内容写入文本文件中
- 文件拷贝：文本文件和非文本拷贝
- 对象序列化：对象数据的读写

【返回】

示例1：读文件

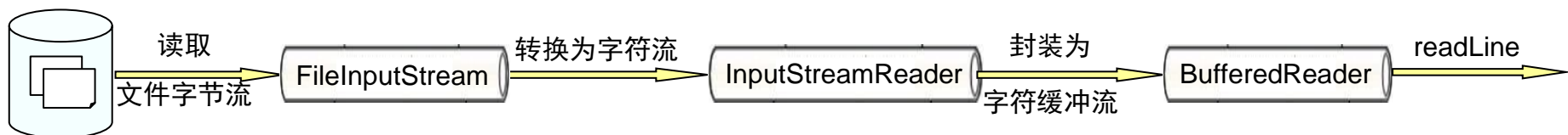
- 准备工作：在D盘新建test文件夹，并新建test.txt文件
- test.txt内容如下：

姓名：小明；性别：男；年龄：19；成绩：630

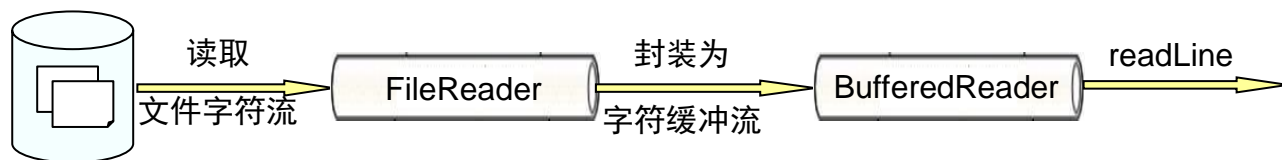
姓名：小翠；性别：女；年龄：18；成绩：650

示例1：读文件

■ 方法1：使用文件字节流FileInputStream读取



■ 方法2：使用文件字符流FileReader读取

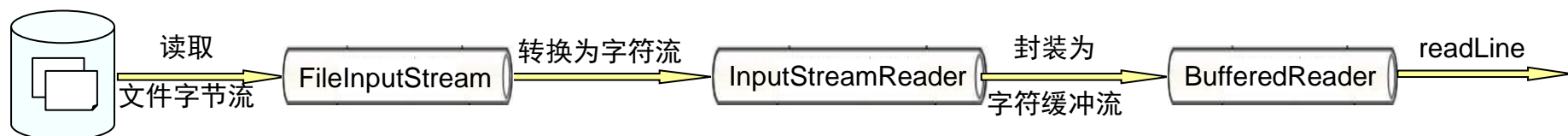


方法1：使用文件字节流FileInputStream读取

```
try {  
    BufferedReader br = new BufferedReader( new InputStreamReader(  
        new FileInputStream("d:\\test\\test.txt") ) );  
  
    String line = "";  
    String tmp = "";  
    while ( ( tmp = br.readLine() ) != null ) {    // 一次读取一行  
        line += tmp + "\n";  
    }  
    br.close();  
    System.out.print(line);  
} catch (IOException e) { e.printStackTrace(); }
```

运行结果

姓名：小明；性别：男；年龄：19；成绩：630
姓名：小翠；性别：女；年龄：18；成绩：650

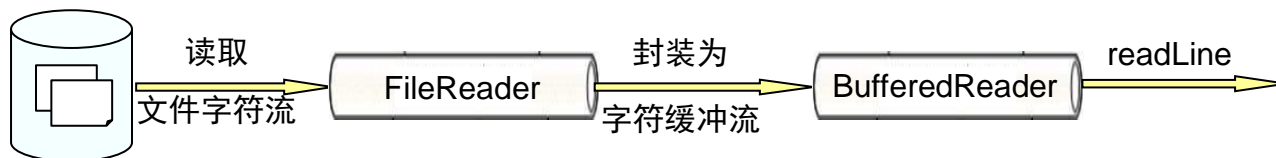


方法2：使用文件字符流FileReader读取

```
try{  
    BufferedReader br = new BufferedReader(new FileReader("d:\\test\\test.txt"));  
    String line = "";  
    String tmp = "";  
    while ( ( tmp = br.readLine() ) != null ) {    // 一次读取一行  
        line += tmp + "\n";  
    }  
    br.close();  
    System.out.print(line);  
} catch(IOException e) { e.printStackTrace(); }
```

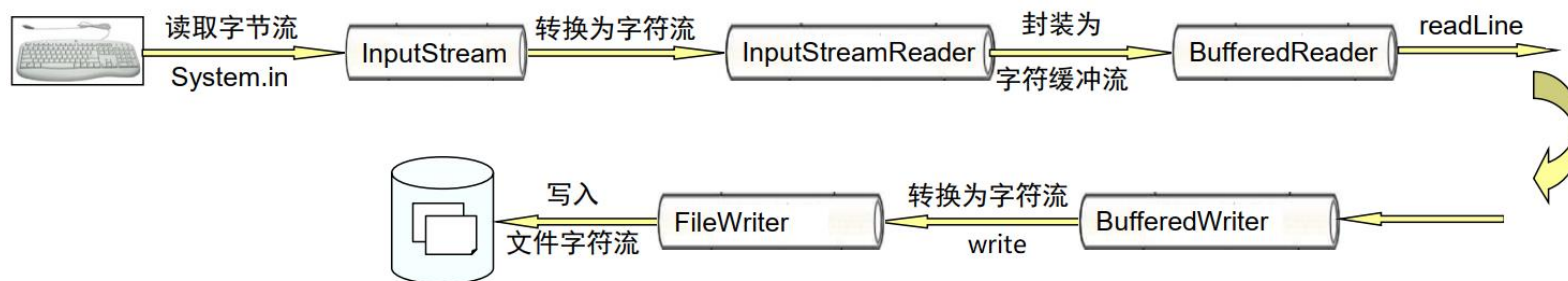
运行结果

姓名：小明；性别：男；年龄：19；成绩：630
姓名：小翠；性别：女；年龄：18；成绩：650

[【返回】](#)

示例2：写文件

- 将键盘输入的内容写入文本文件中（使用FileWriter）



关键代码：

```
System.out.println("请输入文件内容，以Ctrl+D结束:");  
try {  
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  
    BufferedWriter bw = new BufferedWriter(new FileWriter("d:\\test\\test2.txt"));  
    String s = "";  
    while ( ( s = br.readLine() ) != null ) {  
        bw.write(s);  
        bw.newLine();  
    }  
    br.close(); bw.close();  
} catch (IOException e) { e.printStackTrace(); }
```

test2.txt会自动创建

运行情况

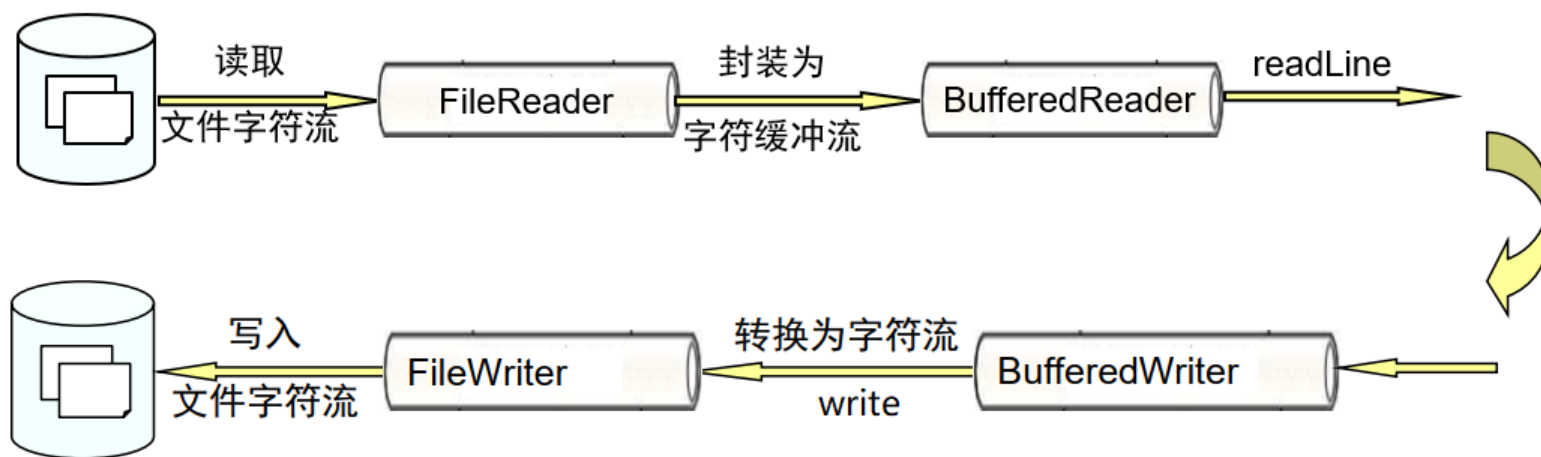
请输入文件内容，以Ctrl+D结束：
hi, 我是小明!
来自武汉
bye
^D

Ctrl+D结束输入

【返回】

示例3：文件拷贝

■ 文本拷贝：使用FileReader + FileWriter



关键代码：

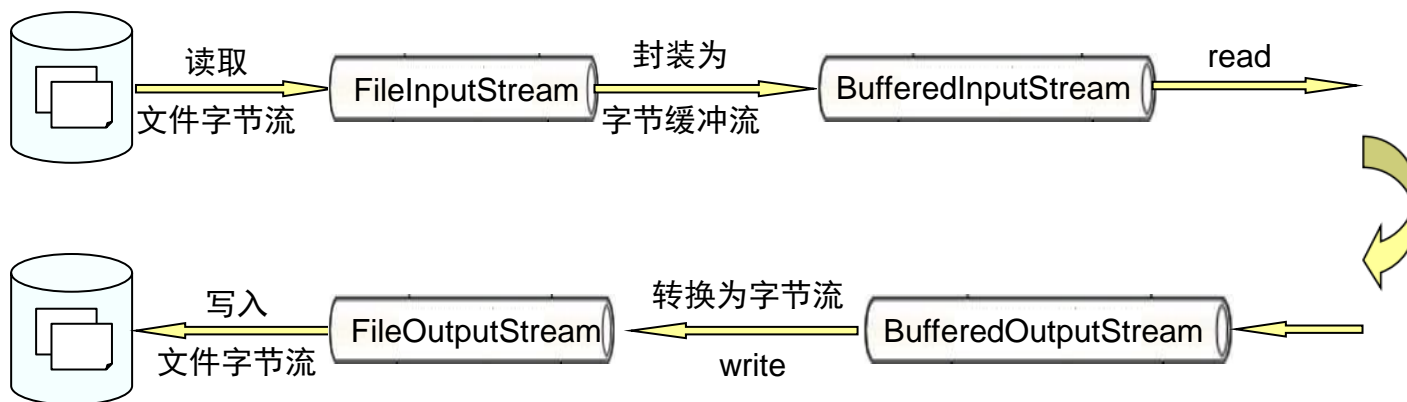
```
System.out.println("开始文件复制...");
long t = System.currentTimeMillis(); //获取当前系统时间
String path = "d:\\test\\test.txt";
String target = "d:\\test\\test2.txt";
try {
    BufferedReader br = new BufferedReader(new FileReader(path));
    BufferedWriter bw = new BufferedWriter(new FileWriter(target));
    String s;
    while ( ( s = br.readLine() ) != null ) {
        bw.write(s);
        bw.newLine();
    }
    br.close(); bw.close();
} catch (IOException e) { e.printStackTrace(); }
t = System.currentTimeMillis() - t; //计算时间差
System.out.println("文件复制结束\n耗时"+ t + "ms");
```

运行情况

开始文件复制...
文件复制结束
耗时1ms

示例3：文件拷贝（续）

■ 非文本拷贝：使用 FileInputStream + FileOutputStream



关键代码：

```
System.out.println("开始文件复制...");
long t = System.currentTimeMillis(); //获取当前系统时间
String path = "d:\\test\\1.gif";
String target = "d:\\test\\1_copy.gif ";
try {
    BufferedInputStream br = new BufferedInputStream( new FileInputStream(path) );
    BufferedOutputStream bw = new BufferedOutputStream( new FileOutputStream(target));
    int i;
    while ( ( i = br.read() ) != -1 ) {
        bw.write(i);
    }
    br.close(); bw.close();
} catch (IOException e) { e.printStackTrace(); }
t = System.currentTimeMillis() - t; //计算时间差
System.out.println("文件复制结束\n耗时"+ t + "ms");
```

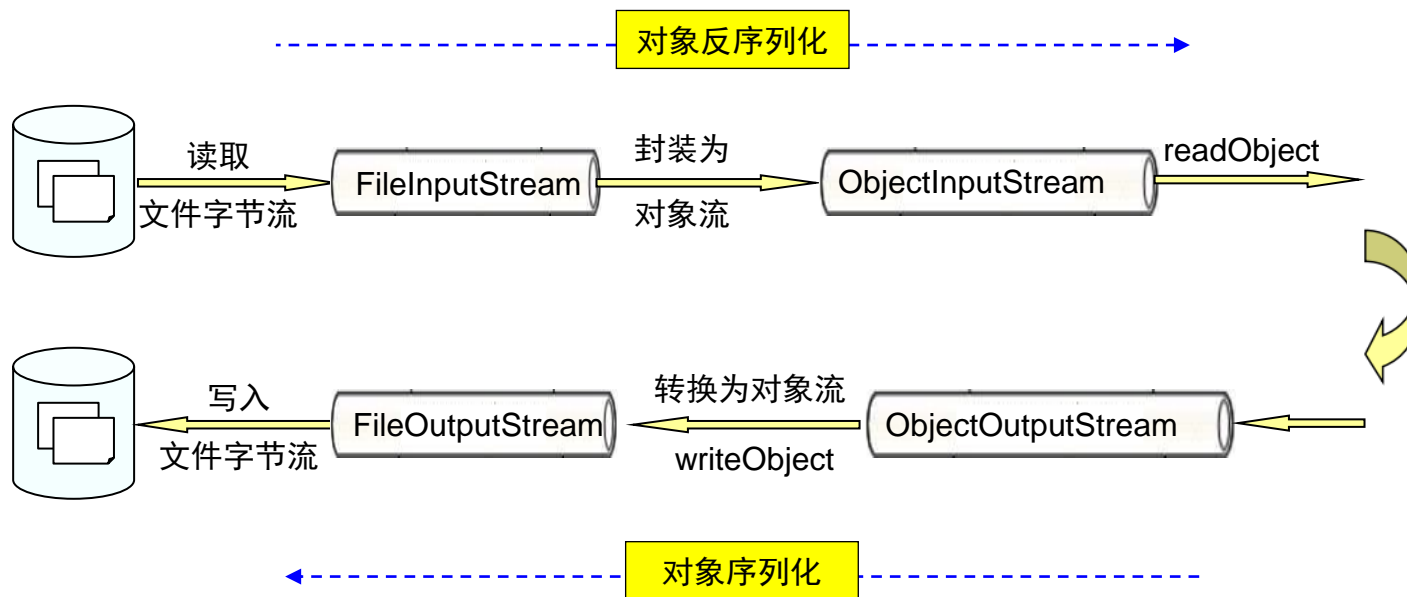
运行情况

开始文件复制...
文件复制结束
耗时2ms

[【返回】](#)

示例4：对象序列化

■ 使用 FileInputStream + FileOutputStream



示例:

必须实现Serializable接口


```
class Person implements Serializable {  
    private static final long serialVersionUID = 1L;    //值任意，默认为1L  
  
    String name;  
    int age;  
  
    @Override  
    public String toString() {  
        return "name=" + name + ", age=" + age;  
    }  
}
```

建议添加 serialVersionUID
来保证序列化版本的一致性
(针对类成员变量的修改)

序列化关键代码：

```
Person person = new Person();  
person.name = "小明";  
person.age = 18;  
  
try {  
    ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("D:\\test\\person.txt"));  
    out.writeObject(person);  
    out.close();  
} catch (Exception e) {  
    e.printStackTrace();  
}  
  
System.out.println("Serialized data is saved");
```

序列化对象写入person.txt文件



反序列化关键代码：

```
Person p2=null;
try {
    ObjectInputStream in = new ObjectInputStream(new FileInputStream("D:\\test\\person.txt"));
    p2 = (Person) in.readObject();
    in.close();
} catch (IOException | ClassNotFoundException e) {
    e.printStackTrace();
}
System.out.println("Reading serialized data");
System.out.println(p2);
```

从person.txt文件读取序列化对象

需要强转

[【返回】](#)

12.2 Java File类

- 创建目录
- 读取目录
- 删除目录或文件

1. 创建目录

- `mkdir()`方法：创建一个文件夹，成功则返回`true`，失败则返回`false`
 - 只能在已经存在的文件夹下建立新的文件夹
- `mkdirs()`方法：创建一个文件夹和它的所有父文件夹
 - 父级目录不存在也可以一并进行创建，可用于创建多级目录

常用

示例：

```
public class Test {  
    public static void main(String[] args) {  
        String dirname = "d:/test/java/example"; ← Windows系统中Java可使用"/"作为文件分隔符  
        File file = new File(dirname);    // 创建File对象  
        boolean flag = file.mkdirs();    // 创建目录  
        if( flag ) {  
            System.out.println(dirname+"目录创建成功");  
        }  
        else {  
            System.out.println(dirname+"目录已存在");  
        }  
    }  
}
```

2. 读取目录

- 一个目录其实就是一个File对象，它包含其他文件和文件夹
- 常用方法：

- `boolean exists()`：当前文件或文件夹是否存在
- `boolean isFile()`：当前File对象是否是文件
- `boolean isDirectory()`：当前File对象是否是目录
- `String[] list()`：返回指定目录下所有的文件和目录名称
- `File[] listFiles()`：返回指定目录下所有的File对象
- `String getName()`：获得当前文件或文件夹的名称

示例:

```
public class Test {  
    public static void main(String[] args) {  
        String dirname = "d:/test";  
        File file = new File(dirname);  
        if ( file.isDirectory() ) {  
            System.out.println("目录 " + dirname);  
            File[] files= file.listFiles();  
            for ( File f : files ) {  
                if ( f.isDirectory() ) {  
                    System.out.println( f.getName() + " 是一个目录" );  
                } else {  
                    System.out.println( f.getName() + " 是一个文件" );  
                }  
            }  
        }  
        else {  
            System.out.println( dirname + " 不是一个目录" );  
        }  
    }  
}
```

运行情况

```
目录 d:/test  
1.gif 是一个文件  
java 是一个目录  
test.txt 是一个文件  
test2.txt 是一个文件
```

2. 删除目录或文件

谨慎删除

- delete()方法：删除指定的文件或文件夹
 - 删除文件夹时，必须保证该文件夹下没有其他文件

示例：

```
// 递归删除文件及目录
public static void deleteFolder(File folder) {
    File[] files = folder.listFiles();
    if ( files != null ) {
        for ( File f : files ) {
            if ( f.isDirectory() ) {
                deleteFolder(f);      //递归
            } else {
                System.out.println(f.getName() + "被删除");
                f.delete();
            }
        }
    }
    System.out.println(folder.getName() + "被删除");
    folder.delete();      //删除自身
}
```

测试代码

```
String dirname = "d:/test/java";
File folder = new File(dirname);

if (!folder.exists()) {
    System.out.println("目录不存在");
    return;
}

Scanner sc = new Scanner(System.in);
System.out.print("确认删除" + dirname + "文件夹(y/n)?");

String ans = null;
while ( ans == null ) {
    String unvalidatedString = sc.next();
    if (unvalidatedString.equalsIgnoreCase("y")) {
        ans = unvalidatedString;
        break;
    }
    if (unvalidatedString.equalsIgnoreCase("n")) {
        ans = unvalidatedString;
        break;
    }
    System.out.println("请输入 'y' 或 'n'");
}

if ( ans.equalsIgnoreCase("y") ) {
    deleteFolder(folder);
    System.out.println(dirname + "目录成功删除");
} else {
    System.out.println(dirname + "目录没有删除");
}
```

[【返回】](#)

12.3 Java网络编程

- [Socket编程](#)
- [URL编程](#)

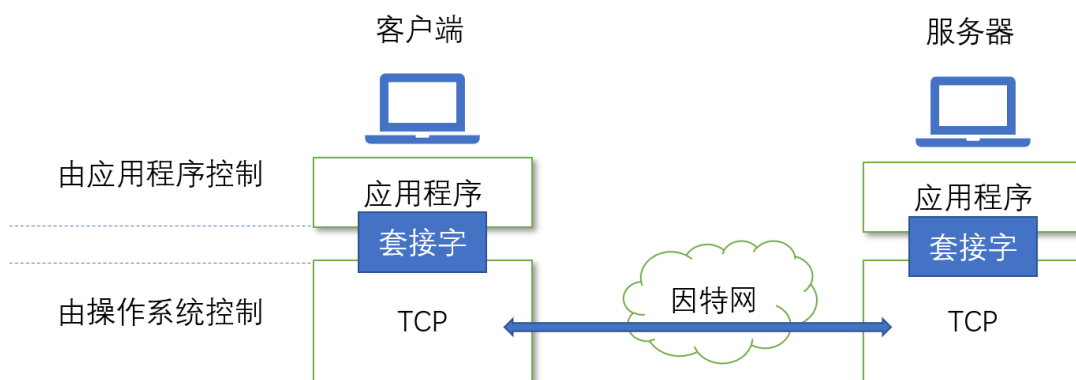
【[返回](#)】

1. Socket编程

■ Socket概念

网络上的两个程序通过一个双向的通信连接实现数据的交换，这个连接的一端称为一个socket（套接字）

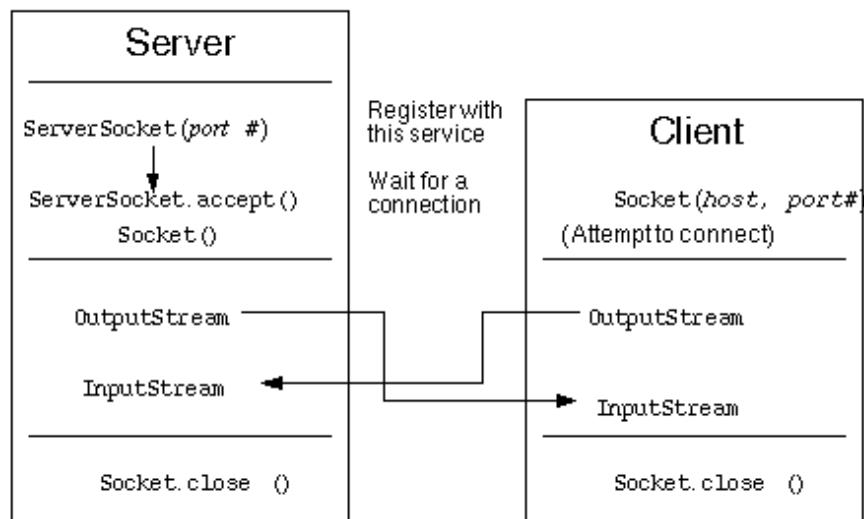
socket是Web应用程序进行网络通信的接口



端口号

- 如果要发起网络连接，不仅需要知道远程机器的IP地址或名字，而且还需要一个端口号
 - 客户端和服务端必须事先约定所使用的端口
 - 如果系统两部分所使用的端口不一致，那就不能进行通信
 - "IP地址+端口号" 来区分不同的服务
- TCP/IP系统中的端口号是一个16位的数字，范围是0~65535
 - 一般来说，小于1024的端口号保留给预定义的服务
 - 例如：HTTP服务端口号80，FTP服务端口号21

Socket编程基本框架



ServerSocket、Socket类在java.net包

Socket连接主要流程:

- 服务器分配一个端口号port: `new ServerSocket(int port)`
- 如果客户端请求连接, 服务器则用`accept()`方法从请求队列中取出一个socket连接打开
- 客户端在自己主机上的port端口建立连接: `new Socket(主机地址, 端口号port)`
- 服务器和客户使用Socket的 `InputStream` 和 `OutputStream` 进行通信

Socket常用方法

方法	说明
Socket(String host, int port)	创建套接字并将其连接到指定主机上的指定端口号
getInetAddress()	返回套接字连接的地址
getLocalPort()	返回套接字绑定到的本地端口
getPort()	返回套接字连接到的远程端口
getInputStream()	返回套接字的输入流
getOutputStream()	返回套接字的输出流
void close()	关闭套接字

ServerSocket常用方法

方法	说明
ServerSocket(int port)	创建绑定到特定端口的服务器套接字
accept()	侦听并接受套接字的连接(如果没有连接则一直等待)

服务器端代码

Server.java

```
public class Server {  
    public static void main(String[] args) {  
        try {  
            ServerSocket serverSocket = new ServerSocket(5432);  
            System.out.println("启动服务器,等待连接....");  
            Socket socket = serverSocket.accept();  
            System.out.println("客户端:" + socket.getInetAddress() + ":" + socket.getLocalPort() + "已连接到服务器");  
  
            //读取客户端消息  
            BufferedReader br = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
            String msg = br.readLine();  
            System.out.println("接收客户端的消息: " + msg);  
  
            //向客户端发送消息  
            BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));  
            bw.write("hi,我是服务器\n"); // '\n'必须  
            bw.flush();  
            // '\n'必须  
  
            br.close();        bw.close();        socket.close();        serverSocket.close(); // 关闭所有流  
        } catch (Exception e) { e.printStackTrace(); }  
    }  
}
```

客户端代码

Client.java

```
public class Client {  
    public static void main(String[] args) {  
        try {  
            Socket socket = new Socket("127.0.0.1", 5432);  
            System.out.println("启动客户端....");  
  
            //向服务器端发送消息  
            BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));  
            bw.write("hi,我是客户端\n"); // '\n'必须  
            bw.flush();  
  
            //读取服务器返回的消息  
            BufferedReader br = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
            String msg = br.readLine();  
            System.out.println("接收服务器信息: " + msg);  
  
            br.close();      bw.close();      socket.close();    // 关闭所有流  
        } catch (Exception e) { e.printStackTrace(); }  
    }  
}
```

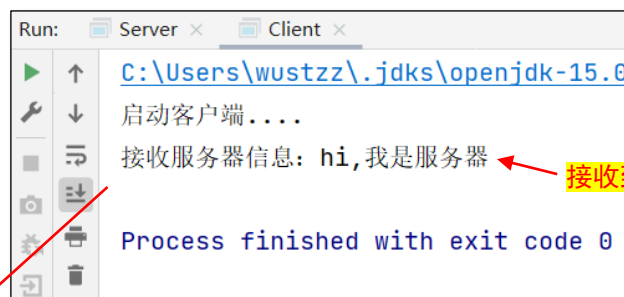
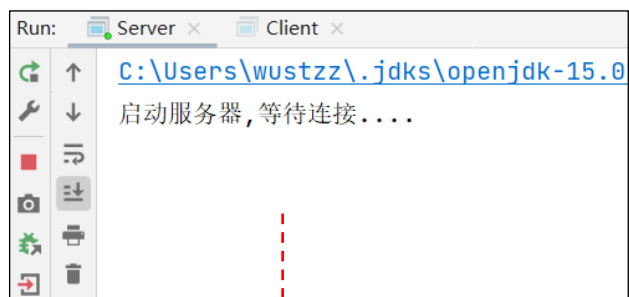
在指定端口创建socket连接

127.0.0.1是本地地址

\n'必须

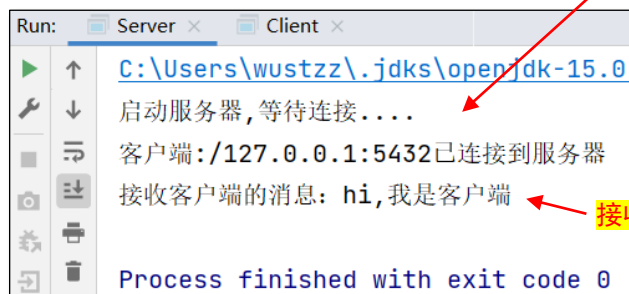
运行情况

■ 先运行服务器端，再运行客户端



接收到服务端发送的消息

客户端启动之后



接收到客户端发送的消息

Socket编程练习

- 在网络应用中，客户端通常要定期检测服务器工作状态，如：客户端每隔30秒发送询问包，服务器收到后将报告工作状态。
- 请用Java语言的TCP技术模拟。

```
try {  
    ServerSocket serverSocket = new ServerSocket(5120);  
    Socket socket = serverSocket.accept();  
    InputStream is = socket.getInputStream();  
    BufferedReader br = new BufferedReader(new InputStreamReader(is));  
    OutputStream os = socket.getOutputStream();  
    BufferedWriter bw = new BufferedWriter(os);  
    while (true){  
        if( br.readLine().equals("isLive?") ) {  
            System.out.println("Client ask");  
            bw.write("Working\n\r");  
            bw.flush();  
        }  
    }  
} catch (Exception e) {  
    e.printStackTrace();  
}
```

Writer

```
try {  
    Socket socket = new Socket("127.0.0.1", 5120);  
    OutputStream os = socket.getOutputStream();  
    BufferedWriter bw = new BufferedWriter(os);  
    InputStream is = socket.getInputStream();  
    BufferedReader br = new BufferedReader(new InputStreamReader(is));  
    while (true){  
        Thread.sleep(30000);  
        bw.write("isLive?\n");  
        bw.flush();  
        if(br.readLine().equals("Working")) {  
            System.out.println("Server is working");  
        }  
    }  
} catch (Exception e) {  
    e.printStackTrace();  
}
```

思考：如何改成多线程编程实现

[【返回】](#)

2. URL编程

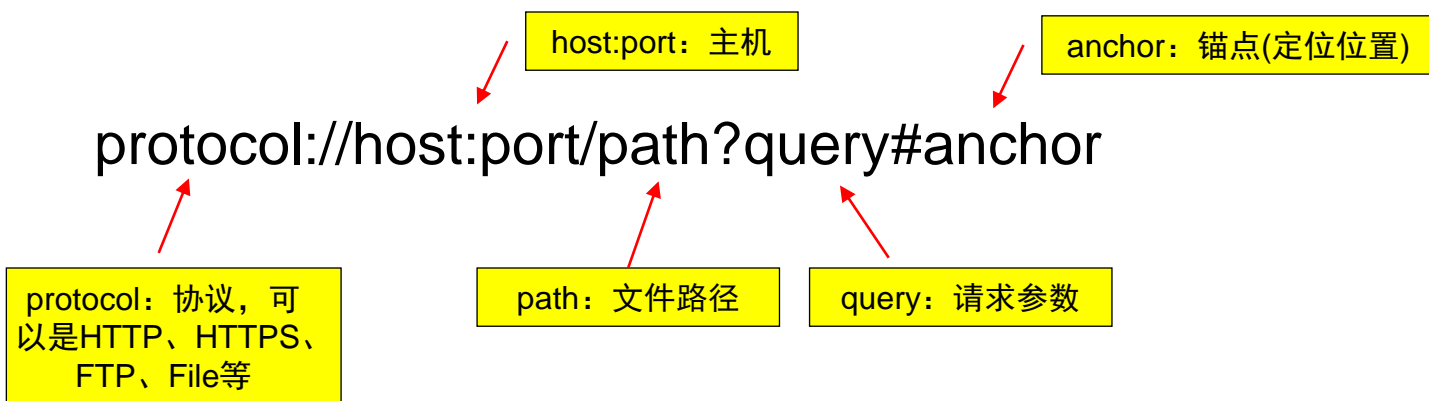
■ URL概念

URL(Uniform Resource Locator)是统一资源定位符，表示为互联网上的资源，俗称为网页地址



URL组成

■ URL基本组成：



URL常用方法

方法	说明
<code>URL(String url)</code>	通过给定的URL字符串创建URL
<code>URLConnection.openConnection()</code>	打开一个URL连接(URLConnection)
<code>String getProtocol()</code>	获得协议名
<code>String getHost()</code>	获得主机名
<code>int getPort()</code>	获得端口号
<code>String getPath()</code>	获得路径
<code>String getQuery()</code>	获得请求参数
<code>String getRef()</code>	获得锚点

URLConnection常用方法

方法	说明
<code>getInputStream()</code>	获得URL的输入流，用于读取资源
<code>getOutputStream()</code>	获得URL的输出流，用于写入资源
<code>URL getURL()</code>	获得URLConnection 对象连接的URL
<code>Object getContent()</code>	获得URL链接内容
<code>int getContentLength()</code>	获得内容长度值

网页抓取示例

主要代码

```
try {  
    URL url = new URL("https://news.wust.edu.cn/2021/0918/c59a245438/page.htm");  
    URLConnection conn = url.openConnection();  
    BufferedReader br = new BufferedReader( new InputStreamReader( conn.getInputStream() ) );  
    BufferedWriter bw = new BufferedWriter( new FileWriter("d:\\test\\data.html") );  
  
    String line;  
    while ( ( line = br.readLine() ) != null ) {  
        System.out.println(line);  
        bw.write(line);  
        bw.newLine();  
    }  
    br.close();  
    bw.close();  
} catch (Exception e) {  
    e.printStackTrace();  
}
```

抓取的页面内容存在data.html文件中

运行结果(控制台输出抓取的网页内容)



```
Run: Server x Client x  
C:\Users\wustzz\jdk\openjdk-15.0.1\bin\java.exe "-javaagent:C:  
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "  
<html xmlns="http://www.w3.org/1999/xhtml">  
<head>  
<meta http-equiv="Content-Type" content="text/html; charset=utf-  
<title>我校举行“九一八”爱国主义教育课暨入校宣誓仪式</title>  
<meta name="description" content=" 武科大网讯（见习记者金红志艺摄影向
```

huan hang

【完】