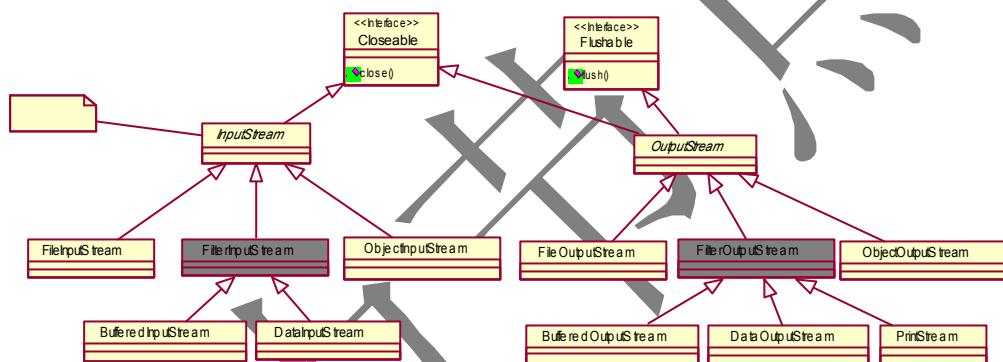


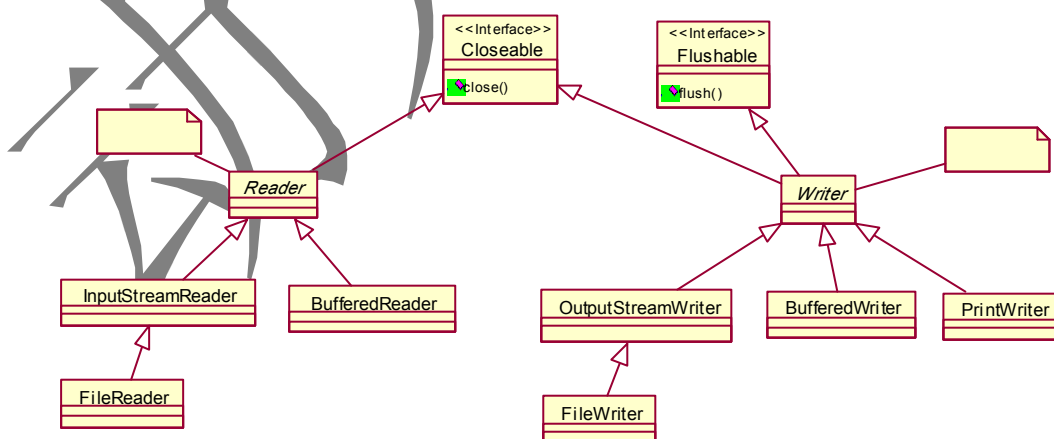
1. 纲要

- a) Java 流概述
- b) 文件流
- c) 缓冲流
- d) 转换流
- e) 打印流
- f) 对象流
- g) File 类
- h) zip 格式

InputStream 和 OutputStream



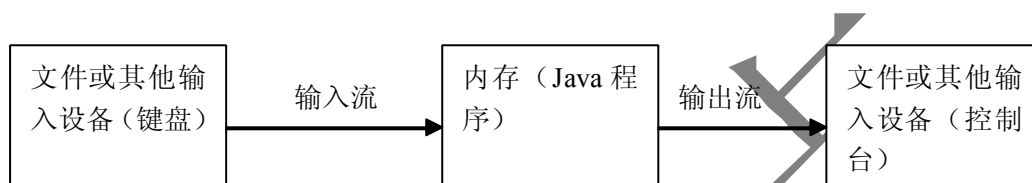
Reader 和 Writer



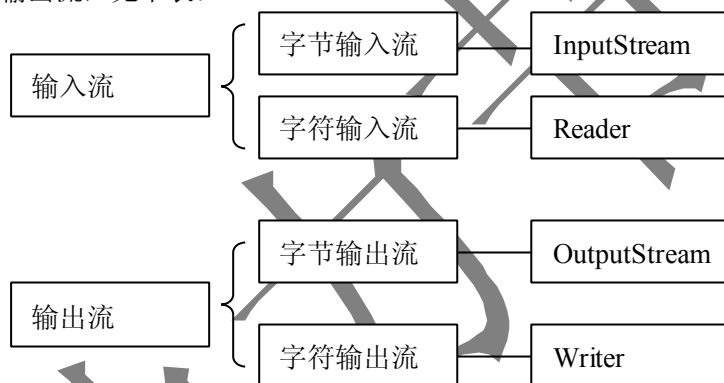
2. 内容

2.1、Java 流概述

文件通常是由一连串的字节或字符构成，组成文件的字节序列称为**字节流**，组成文件的字符序列称为**字符流**。Java 中根据流的方向可以分为输入流和输出流。**输入流**是将文件或其它输入设备的数据加载到内存的过程；**输出流**恰恰相反，是将内存中的数据保存到文件或其他输出设备，详见下图：

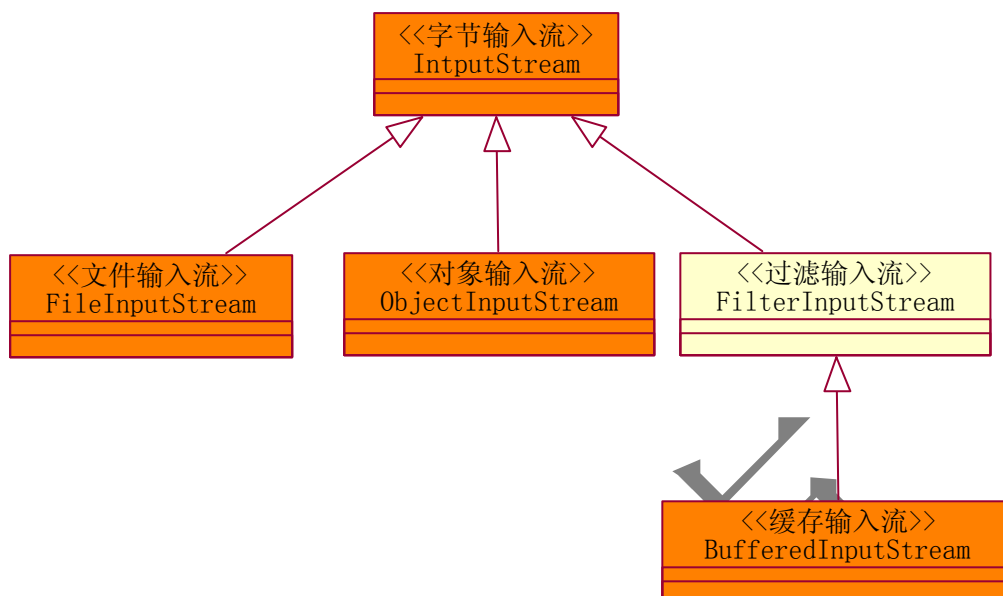


文件是由字符或字节构成，那么将文件加载到内存或再将文件输出到文件，需要有输入和输出流的支持，那么在 Java 语言中又把输入和输出流分为了两个，字节输入和输出流，字符输入和输出流，见下表：



2.1.1、InputStream(字节输入流)

InputStream 是字节输入流，InputStream 是一个抽象类，所有继承了 InputStream 的类都是字节输入流，主要了解以下子类即可：

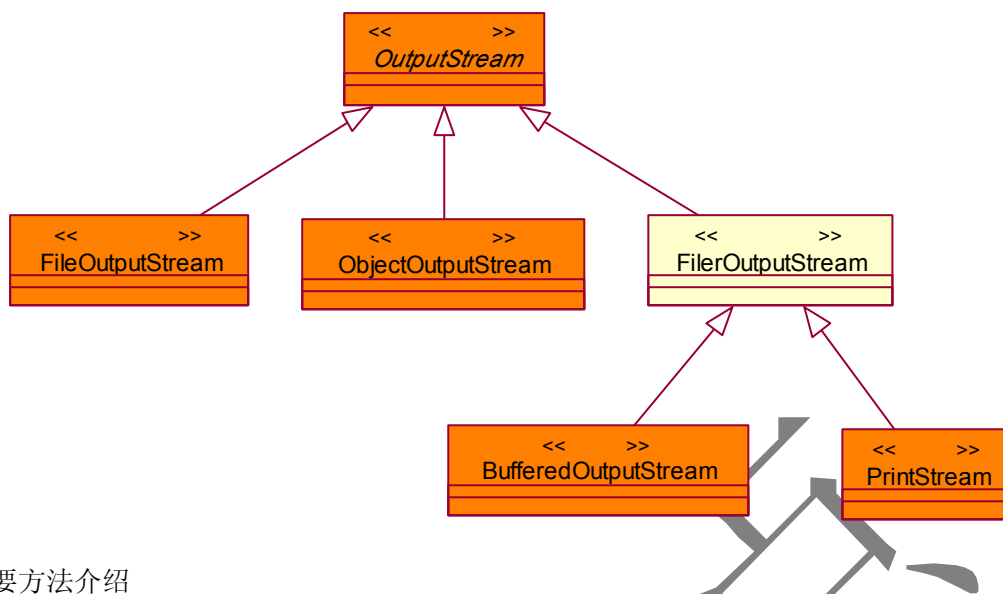


主要方法介绍:

void	close()	关闭此输入流并释放与该流关联的所有系统资源。
abstract int	read()	从输入流读取下一个数据字节。
int	read(byte[] b)	从输入流中读取一定数量的字节并将其存储在缓冲区数组 b 中。
int	read(byte[] b, int off, int len)	将输入流中最多 len 个数据字节读入字节数组。

2.1.2、OutputStream(字节输出流)

所有继承了 OutputStream 都是字节输出流

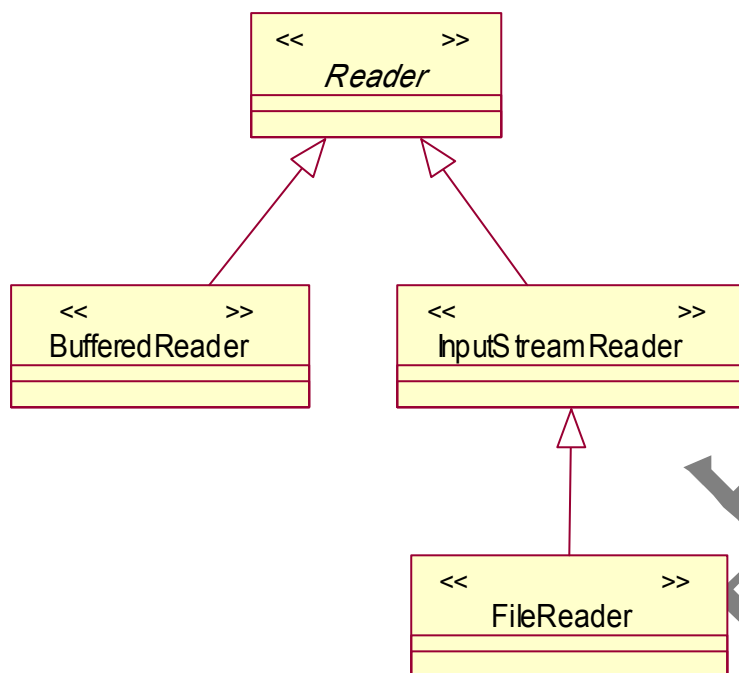


主要方法介绍

void	close()	关闭此输出流并释放与此流有关的所有系统资源。
void	flush()	刷新此输出流并 强制 写出所有缓冲的输出字节。
void	write (byte[] b)	将 b.length 个字节从指定的字节数组写入此输出流。
void	write (byte[] b, int off, int len)	将指定字节数组中从偏移量 off 开始的 len 个字节写入此输出流。
abstract void	write (int b)	将指定的字节写入此输出流。

2.1.3、Reader(字符输入流)

所有继承了 Reader 都是字符输入流

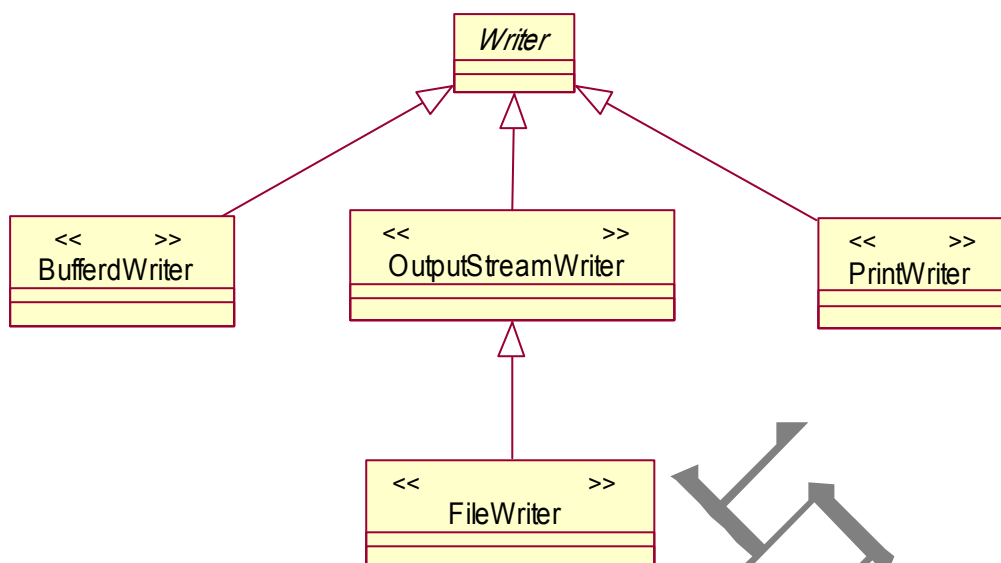


主要方法介绍

abstract void	close() 关闭该流。
int	read() 读取单个字符。
int	read(char[] cbuf) 将字符读入数组。
abstract int	read(char[] cbuf, int off, int len) 将字符读入数组的某一部分。

2.1.4、Writer(字符输出流)

所有继承了 Writer 都是字符输出流



主要方法介绍

Writer	append (char c) 将指定字符追加到此 writer。
abstract void	close () 关闭此流，但要先刷新它。
abstract void	flush () 刷新此流。
void	write (char[] cbuf) 写入字符数组。
abstract void	write (char[] cbuf, int off, int len) 写入字符数组的某一部分。
void	write (int c) 写入单个字符。
void	write (String str) 写入字符串。
void	write (String str, int off, int len) 写入字符串的某一部分。

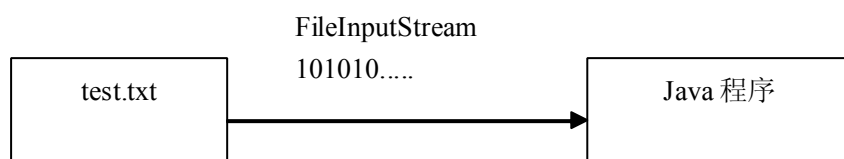
2.2、文件流

文件流主要分为：文件字节输入流、文件字节输出流、文件字符输入流、文件字符输出流

2.2.1、FileInputStream(文件字节输入流)

FileInputStream 主要按照字节方式读取文件，例如我们准备读取一个文件，该文件的名称为

test.txt



【示例代码】

```
import java.io.*;

public class FileInputStreamTest01 {

    public static void main(String[] args) {
        InputStream is = null;
        try {
            is = new FileInputStream("c:\\test.txt");

            int b = 0;
            while ((b = is.read()) != -1) {
                //直接打印
                //System.out.print(b);

                //输出字符
                System.out.print((char)b);
            }

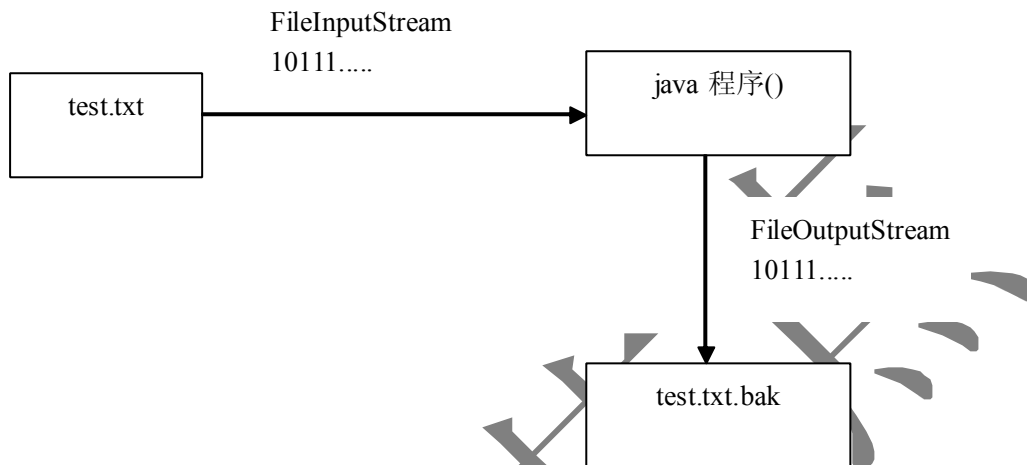
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (is != null) {
                    is.close();
                }
            } catch (IOException e) {}
        }
    }
}
```



文件可以正确的读取，但是我们的汉字乱码了，原因在于使用了字节输入流，它是一个字节一个字节读取的，而汉字是两个字节，所以读出一个字节就打印，那么汉字是不完整的，所以就乱码了

2.2.2、FileOutputStream(文件字节输出流)

FileOutputStream 主要按照字节方式写文件，例如：我们做文件的复制，首先读取文件，读取后在将该文件另写一份保存到磁盘上，这就完成了备份



【示例代码】

```
import java.io.*;

public class FileOutputStreamTest01 {

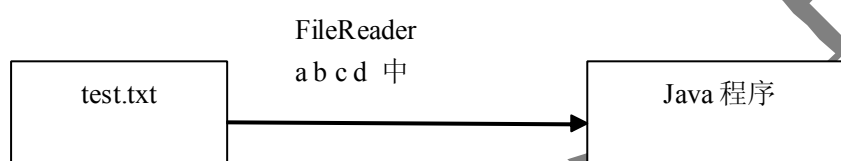
    public static void main(String[] args) {
        InputStream is = null;
        OutputStream os = null;
        try {
            is = new FileInputSteam("c:\\test.txt");
            os = new FileOutputStream("d:\\test.txt.bak");
            int b = 0;
            while ((b = is.read()) != -1) {
                os.write(b);
            }
            System.out.println("文件复制完毕！");
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (is != null) {
                    is.close();
                }
            }
        }
    }
}
```



```
        if (os != null) {  
            os.close();  
        }  
    } catch(IOException e) {}  
}  
}
```

2.2.3、FileReader(文件字符输入流)

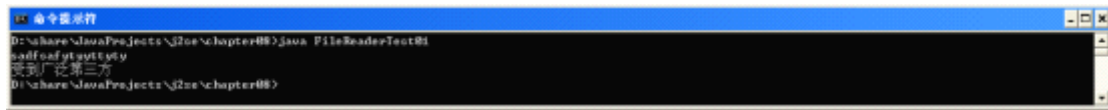
FileReader 是一字符为单位读取文件，也就是一次读取两个字节，如：



【示例代码】

```
import java.io.*;  
  
public class FileReaderTest01 {  
  
    public static void main(String[] args) {  
        Reader r = null;  
        try {  
            r = new FileReader("c:\\test.txt");  
  
            int b = 0;  
            while ((b = r.read()) != -1) {  
                //输出字符  
                System.out.print((char)b);  
            }  
  
        } catch(FileNotFoundException e) {  
            e.printStackTrace();  
        } catch(IOException e) {  
            e.printStackTrace();  
        } finally {  
            try {  
                if (r != null) {  
                    r.close();  
                }  
            } catch(IOException e) {}  
        }  
    }  
}
```

```
}  
}
```



因为采用了字符输入流读取文本文件，所以汉字就不乱码了，因为一次读取两个字节（即一个字符）

2.2.4、FileWriter(文件字符输出流)

【代码示例】

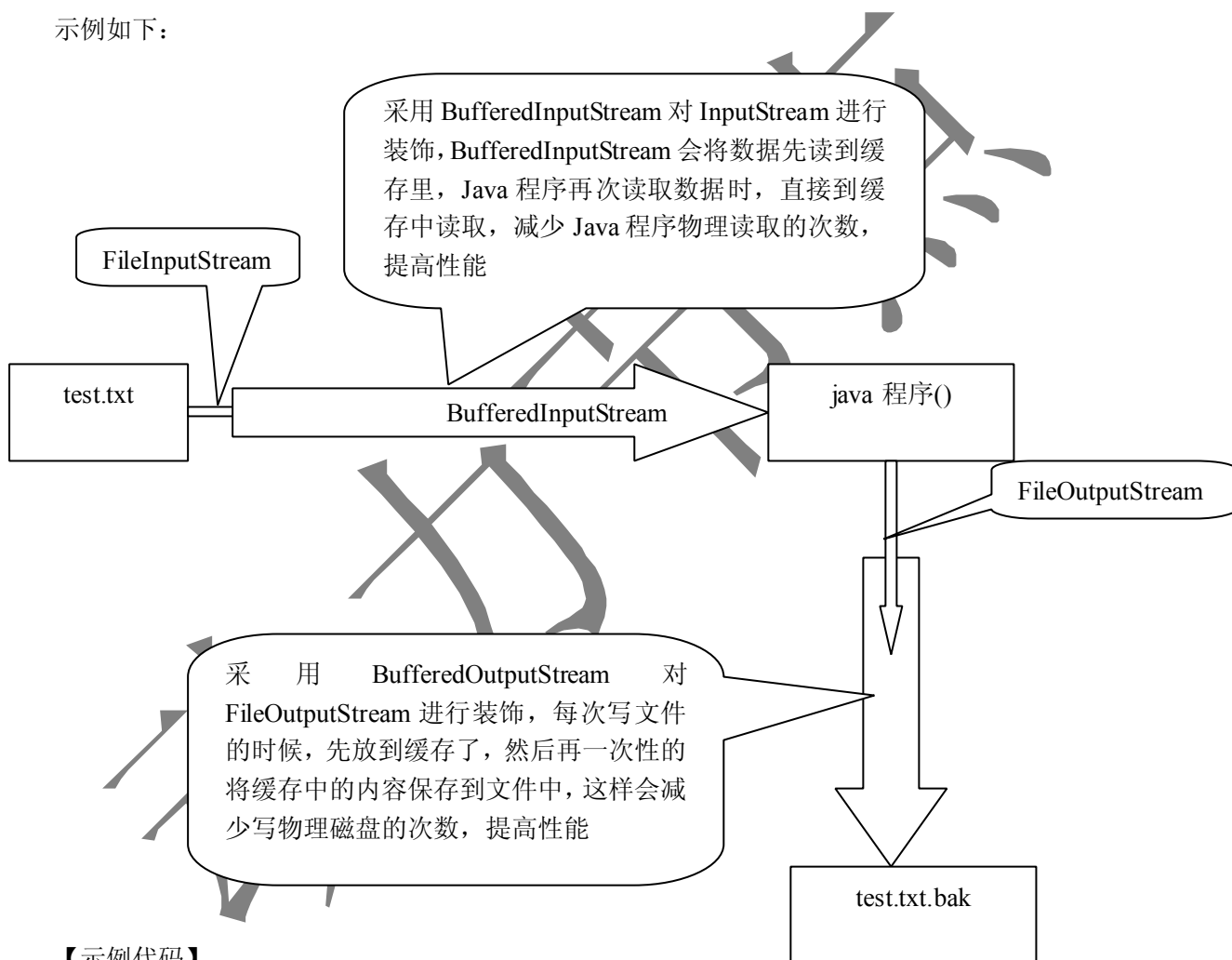
```
import java.io.*;  
  
public class FileWriterTest01 {  
  
    public static void main(String[] args) {  
        Writer w = null;  
        try {  
            //以下方式会将文件的内容进行覆盖  
            //w = new FileWriter("c:\\test.txt");  
            //w = new FileWriter("c:\\test.txt", false);  
  
            //以下为 true 表示，在文件后面追加  
            w = new FileWriter("c:\\test.txt", true);  
            w.write("你好你好!!! ");  
            //换行  
            w.write("\n");  
        } catch (FileNotFoundException e) {  
            e.printStackTrace();  
        } catch (IOException e) {  
            e.printStackTrace();  
        } finally {  
            try {  
                if (w != null) {  
                    w.close();  
                }  
            } catch (IOException e) {}  
        }  
    }  
}
```

2.3、缓冲流

缓冲流主要是为了提高效率而存在的，减少物理读取次数，缓冲流主要有：BufferedInputStream、BufferedOutputStream、BufferedReader、BufferedWriter，并且BufferedReader提供了实用方法readLine()，可以直接读取一行，BufferWriter提供了newLine()可以写换行符。

2.3.1、采用字节缓冲流改造文件复制代码

示例如下：



【示例代码】

```
import java.io.*;

public class BufferedStreamTest01 {

    public static void main(String[] args) {
        InputStream is = null;
        OutputStream os = null;
        try {
            is = new BufferedInputStream(
```

```
        new FileInputStream("c:\\test.txt"));
    os = new BufferedOutputStream(
        new FileOutputStream("d:\\test.txt.bak"));

    int b = 0;
    while ((b = is.read()) != -1) {
        os.write(b);
    }
    //手动调用 flush，将缓冲区中的内容写入到磁盘
    //也可以不用手动调用，缓存区满了自动回清楚了
    //而当输出流关闭的时候也会先调用 flush
    os.flush();
    System.out.println("文件复制完毕！");
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
} finally {
    try {
        if (is != null) {
            is.close();
        }
        if (os != null) {
            //在 close 前会先调用 flush
            os.close();
        }
    } catch (IOException e) {}
}
}
```

//可以显示的调用 flush，flush 的含义是刷新缓冲区,也就是将缓存区中的数据写到磁盘上，不再放到内存里了,在执行 os.close()时，其实默认执行了 os.flush(),我们在这里可以不用显示的调用

2.3.2、采用字符缓冲流改造文件复制代码

```
import java.io.*;

public class BufferedReaderTest01 {

    public static void main(String[] args) {
        BufferedReader r = null;
        BufferedWriter w = null;
        try {
```

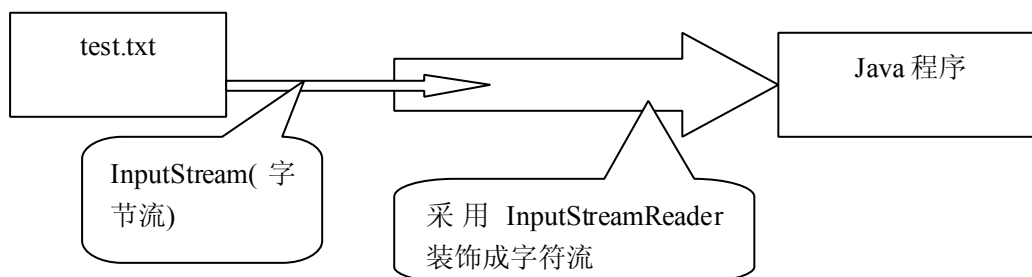
```
r = new BufferedReader(
    new FileReader("c:\\test.txt"));
w = new BufferedWriter(
    new FileWriter("d:\\test.txt.bak"));
String s = null;
while ((s = r.readLine()) != null) {
    w.write(s);
    //w.write("\n");
    //可以采用如下方法换行
    w.newLine();
}
System.out.println("文件复制完毕！");
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
} finally {
    try {
        if (r != null) {
            r.close();
        }
        if (w != null) {
            //在 close 前会先调用 flush
            w.close();
        }
    } catch (IOException e) {}
}
}
```

2.4、转换流

转换流主要有两个 `InputStreamReader` 和 `OutputStreamWriter`

- `InputStreamReader` 主要是将字节流输入流转换成字符输入流
- `OutputStreamWriter` 主要是将字节流输出流转换成字符输出流

2.4.1、InputStreamReader



【示例代码】，对 FileInputStreamTest01.java 进行改造，使用字符流

```
import java.io.*;

public class InputStreamReaderTest01 {

    public static void main(String[] args) {
        BufferedReader br = null;
        try {
            br = new BufferedReader(
                new InputStreamReader(
                    new FileInputStream("c:\\test.txt")));

            String s = null;
            while ((s = br.readLine()) != null) {
                System.out.println(s);
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (br != null) {
                    br.close();
                }
            } catch (IOException e) {}
        }
    }
}
```

2.4.2、OutputStreamWriter

```
import java.io.*;

public class OutputStreamWriterTest01 {

    public static void main(String[] args) {
        BufferedWriter bw = null;
        try {
            bw = new BufferedWriter(
                new OutputStreamWriter(
                    new FileOutputStream("c:\\603.txt")));
            bw.write("asdfsdfdsf");
        }
    }
}
```

```
        bw.newLine();
        bw.write("风光风光风光好");
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            if (bw != null) {
                bw.close();
            }
        } catch (IOException e) {}
    }
}
```

2.5、打印流

打印流主要包含两个：PrintStream 和 PrintWriter，分别对应字节流和字符流

2.5.1、完成屏幕打印的重定向

System.out 其实对应的就是 PrintStream，默认输出到控制台，我们可以重定向它的输出，可以定向到文件，也就是执行 System.out.println("hello") 不输出到屏幕，而输出到文件

【示例代码】

```
import java.io.*;

public class PrintStreamTest01 {

    public static void main(String[] args) {
        OutputStream os = null;
        try {
            os = new FileOutputStream("c:/console.txt");
            System.setOut(new PrintStream(os));
            System.out.println("asdfkjfd;lldffdfdrerere");
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (os != null) {
                    os.close();
                }
            }
        }
    }
}
```

```
    }  
    } catch(IOException e) {}  
  }  
}  
}
```

2.5.2、接受屏幕输入

【示例代码】

System.in 可以接收屏幕输入

```
import java.io.*;  
  
public class PrintStreamTest02 {  
  
    public static void main(String[] args) {  
        BufferedReader br = null;  
        try {  
  
            br = new BufferedReader(  
                new InputStreamReader(System.in));  
            String s = null;  
            while ((s=br.readLine()) != null) {  
                System.out.println(s);  
                //为 q 退出循环  
                if ("q".equals(s)) {  
                    break;  
                }  
            }  
        } catch(FileNotFoundException e) {  
            e.printStackTrace();  
        } catch(IOException e) {  
            e.printStackTrace();  
        } finally {  
            try {  
                if (br != null) {  
                    br.close();  
                }  
            } catch(IOException e) {}  
        }  
    }  
}
```


2.6、对象流

对象流可以将 Java 对象转换成二进制写入磁盘，这个过程通常叫做**序列化**，并且还可以从磁盘读出完整的 Java 对象，而这个过程叫做**反序列化**。

对象流主要包括：ObjectInputStream 和 ObjectOutputStream

2.6.1、如何实现序列化和反序列化

如果实现序列化该类必须实现序列化接口 `java.io.Serializable`，该接口没有任何方法，该接口只是一种**标记接口**，标记这个类是可以序列化的

- 序列化

```
import java.io.*;

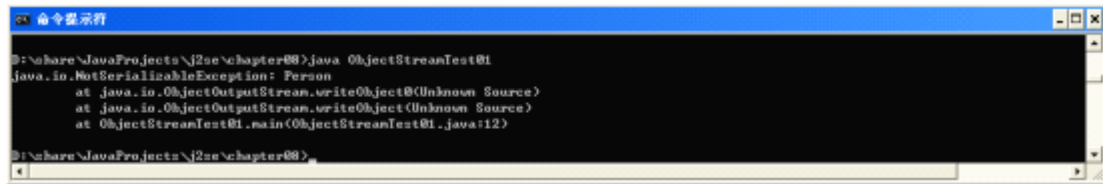
public class ObjectOutputStreamTest01 {

    public static void main(String[] args) {
        ObjectOutputStream oos = null;
        try {
            oos = new ObjectOutputStream(
                new FileOutputStream("c:/Person.dat"));
            Person person = new Person();
            person.name = "张三";
            oos.writeObject(person);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (oos != null) {
                    oos.close();
                }
            } catch (IOException e) {}
        }
    }
}

class Person {

    String name;

}
```



不能序列化, 对序列化的类是有要求的, 这个序列化的类必须实现一个接口 `Serializable`, 这个接口没有任何方法声明, 它是一个标识接口, 如: `java` 中的克隆接口 `Cloneable`, 也是起到了一种标识性的作用

- 序列化

```
import java.io.*;
```

```
public class ObjectStreamTest02 {
```

```
    public static void main(String[] args) {
```

```
        ObjectOutputStream oos = null;
```

```
        try {
```

```
            oos = new ObjectOutputStream(
                new FileOutputStream("c:/Person.dat"));
```

```
            Person person = new Person();
```

```
            person.name = "张三";
```

```
            oos.writeObject(person);
```

```
        } catch (FileNotFoundException e) {
```

```
            e.printStackTrace();
```

```
        } catch (IOException e) {
```

```
            e.printStackTrace();
```

```
        } finally {
```

```
            try {
```

```
                if (oos != null) {
```

```
                    oos.close();
```

```
                }
```

```
            } catch (IOException e) {}
```

```
        }
```

```
    }
```

```
//实现序列化接口
```

```
class Person implements Serializable {
```

```
    String name;
```

```
}
```

以上可以完成序列化

- 反序列化

```
import java.io.*;
```

```
public class ObjectStreamTest03 {

    public static void main(String[] args) {
        ObjectInputStream ois = null;
        try {
            ois = new ObjectInputStream(
                new FileInputStream("c:/Person.dat"));
            //反序列化
            Person person = (Person)ois.readObject();
            System.out.println(person.name);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (ois != null) {
                    ois.close();
                }
            } catch (IOException e) {}
        }
    }
}

//实现序列化接口
class Person implements Serializable{

    String name;
}
```

2.6.2、关于 transient 关键字

```
import java.io.*;

public class ObjectStreamTest04 {

    public static void main(String[] args) {
        writeObject();
        readObject();
    }
}
```

```
private static void readObject() {
    ObjectInputStream ois = null;
    try {
        ois = new ObjectInputStream(
            new FileInputStream("c:/Person.dat"));
        //反序列化
        Person person = (Person)ois.readObject();
        System.out.println(person.name);
        System.out.println(person.age);
    } catch(ClassNotFoundException e) {
        e.printStackTrace();
    } catch(FileNotFoundException e) {
        e.printStackTrace();
    } catch(IOException e) {
        e.printStackTrace();
    } finally {
        try {
            if (ois != null) {
                ois.close();
            }
        } catch(IOException e) {}
    }
}

private static void writeObject() {
    ObjectOutputStream oos = null;
    try {
        oos = new ObjectOutputStream(
            new FileOutputStream("c:/Person.dat"));
        Person person = new Person();
        person.name = "张三";
        person.age = 20;

        oos.writeObject(person);
    } catch(FileNotFoundException e) {
        e.printStackTrace();
    } catch(IOException e) {
        e.printStackTrace();
    } finally {
        try {
            if (oos != null) {
                oos.close();
            }
        }
    } catch(IOException e) {}
}
```

```
    }  
    }  
}  
  
//实现序列化接口  
class Person implements Serializable {  
  
    String name;  
  
    //采用 transient 关键字修饰此属性，序列化时会忽略  
    transient int age;  
}
```

2.6.3、关于 serialVersionUID 属性

【示例代码】，在 person 中加入一个成员属性 sex，然后在读取 person.dat 文件

```
import java.io.*;  
  
public class ObjectStreamTest05 {  
  
    public static void main(String[] args) {  
        //writeObject();  
        readObject();  
    }  
  
    private static void readObject() {  
        ObjectInputStream ois = null;  
        try {  
            ois = new ObjectInputStream(  
                new FileInputStream("c:/Person.dat"));  
            //反序列化  
            Person person = (Person)ois.readObject();  
            System.out.println(person.name);  
            System.out.println(person.age);  
        } catch (ClassNotFoundException e) {  
            e.printStackTrace();  
        } catch (FileNotFoundException e) {  
            e.printStackTrace();  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```
        } finally {
            try {
                if (ois != null) {
                    ois.close();
                }
            } catch (IOException e) {}
        }
    }

    /*
    private static void writeObject() {
        ObjectOutputStream oos = null;
        try {
            oos = new ObjectOutputStream(
                new FileOutputStream("c:/Person.dat"));
            Person person = new Person();
            person.name = "张三";
            person.age = 20;

            oos.writeObject(person);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (oos != null) {
                    oos.close();
                }
            } catch (IOException e) {}
        }
    }
    */
}
```

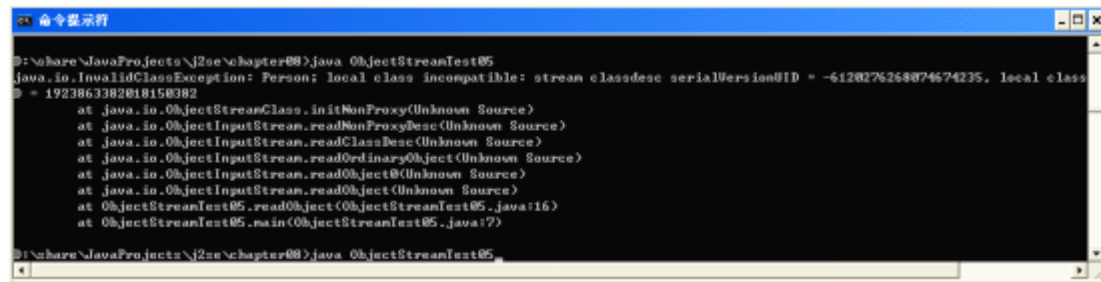
//实现序列化接口

```
class Person implements Serializable{
```

```
    String name;
```

```
    int age;
```

```
boolean sex;
}
```



错误的原因：在序列化存储 Person 时，他会为该生成一个 serialVersionUID=-6120276268074674235，而我们在该类中加入了一个 sex 属性后，那么在使用的时候他就会为该生成一个新的 serialVersionUID= 1923863382018150382，这个两个 UID（-6120276268074674235 和 1923863382018150382）不同，所以 Java 认为是不兼容的两个类。如果解决呢？

通常在实现序列化的类中增加如下定义：

```
static final long serialVersionUID = -1111111111111111L;
```

如果在序列化类中定义了成员域 serialVersionUID，系统会把当前 serialVersionUID 成员域的值作为类的序列号（类的版本号），这样不管你的类如何升级，那么他的序列号（版本号）都是一样的，就不会产生类的兼容问题。

【代码示例】，解决序列化版本冲突的问题

```
import java.io.*;

public class ObjectStreamTest06 {

    public static void main(String[] args) {
        //writeObject();
        readObject();
    }

    private static void readObject() {
        ObjectInputStream ois = null;
        try {
            ois = new ObjectInputStream(
                new FileInputStream("c:/Person.dat"));
            //反序列化
            Person person = (Person)ois.readObject();
            System.out.println(person.name);
            System.out.println(person.age);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
```

```
        e.printStackTrace();
    } finally {
        try {
            if (ois != null) {
                ois.close();
            }
        } catch (IOException e) {}
    }
}

private static void writeObject() {
    ObjectOutputStream oos = null;
    try {
        oos = new ObjectOutputStream(
            new FileOutputStream("c:/Person.dat"));
        Person person = new Person();
        person.name = "张三";
        person.age = 20;

        oos.writeObject(person);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            if (oos != null) {
                oos.close();
            }
        } catch (IOException e) {}
    }
}
```

//实现序列化接口

class Person implements Serializable{

//加入版本号，防止序列化兼容问题

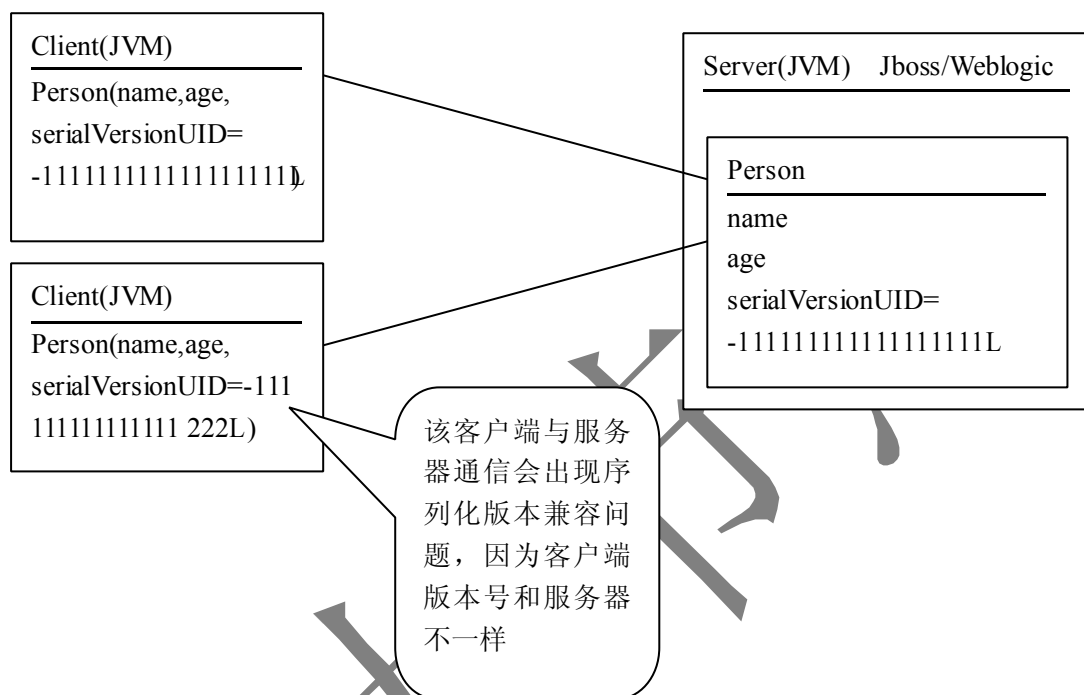
private static final long serialVersionUID = -1111111111111111L;

String name;


```
int age;  
  
boolean sex;  
}
```

以上不再出现序列化的版本问题，因为他们有统一的版本号：-1111111111111111L

进一步理解一下 serialVersionUID



【代码示例】，将 Person 的版本号修改为-11111111111111222L，该客户端与服务器通信会出现序列化版本兼容问题

```
import java.io.*;  
  
public class ObjectStreamTest07 {  
  
    public static void main(String[] args) {  
        //writeObject();  
        readObject();  
    }  
  
    private static void readObject() {  
        ObjectInputStream ois = null;  
        try {  
            ois = new ObjectInputStream(  
                new FileInputStream("c:/Person.dat"));  
            //反序列化  
            Person person = (Person)ois.readObject();  
            System.out.println(person.name);  
        }  
    }  
}
```

```
        System.out.println(person.age);
    } catch(ClassNotFoundException e) {
        e.printStackTrace();
    } catch(FileNotFoundException e) {
        e.printStackTrace();
    } catch(IOException e) {
        e.printStackTrace();
    } finally {
        try {
            if (ois != null) {
                ois.close();
            }
        } catch(IOException e) {}
    }
}
```

```
private static void writeObject() {
    ObjectOutputStream oos = null;
    try {
        oos = new ObjectOutputStream(
            new FileOutputStream("c:/Person.dat"));
        Person person = new Person();
        person.name = "张三";
        person.age = 20;

        oos.writeObject(person);
    } catch(FileNotFoundException e) {
        e.printStackTrace();
    } catch(IOException e) {
        e.printStackTrace();
    } finally {
        try {
            if (oos != null) {
                oos.close();
            }
        } catch(IOException e) {}
    }
}
}
```

//实现序列化接口

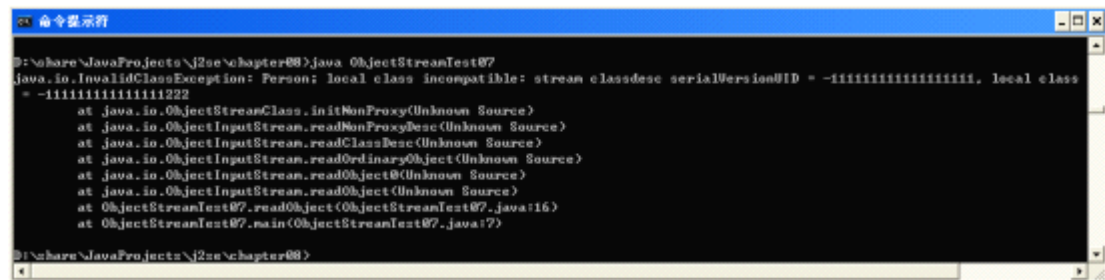
```
class Person implements Serializable {
```

```
//加入版本号，防止序列化兼容问题
private static final long serialVersionUID = -1111111111111222L;

String name;

int age;

}
```



serialVersionUID 就和序列化有关

2.7、File 类

File 提供了大量的文件操作：删除文件，修改文件，得到文件修改日期，建立目录、列表文件等等。

如何递归读取目录及子目录下的文件

```
import java.io.*;

public class FileTest01 {

    public static void main(String[] args) {
        listFile(new File("D:\\share\\03-J2SE\\"), 0);
    }

    //递归读取某个目录及子目录下的所有文件
    private static void listFile(File f, int level) {
        String s = "";
        for (int i=0; i<level; i++) {
            s+="--";
        }
        File[] files = f.listFiles();
        for (int i=0; i<files.length; i++) {
            System.out.println(s + files[i].getName());
            if (files[i].isDirectory()) {
                listFile(files[i], level+1);
            }
        }
    }
}
```

```
}  
}
```

2.8、zip 格式

参见:

java.util.zip.*包下的 api

动力节点