# Java常用工具_IO流

## 课程概要



## 学习目标

理解异常的概念和分类

能够根据需求正确处理异常

理解IO流的概念和作用

能够使用File类的常用功能

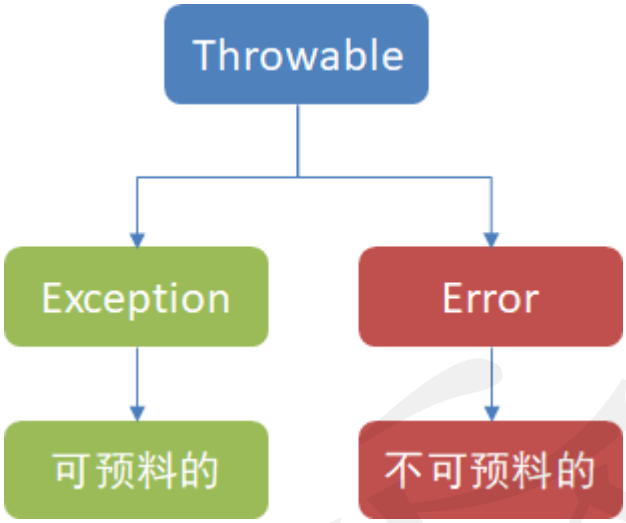
能够使用字符流读写文件

能够使用字节流读写文件

能够使用字符缓冲流读写文件

能够使用字节缓冲流读写文件

## 异常概述




## 什么是异常?

即非正常情况，通俗地说，异常就是程序出现的错误

## 异常的分类（Throwable）

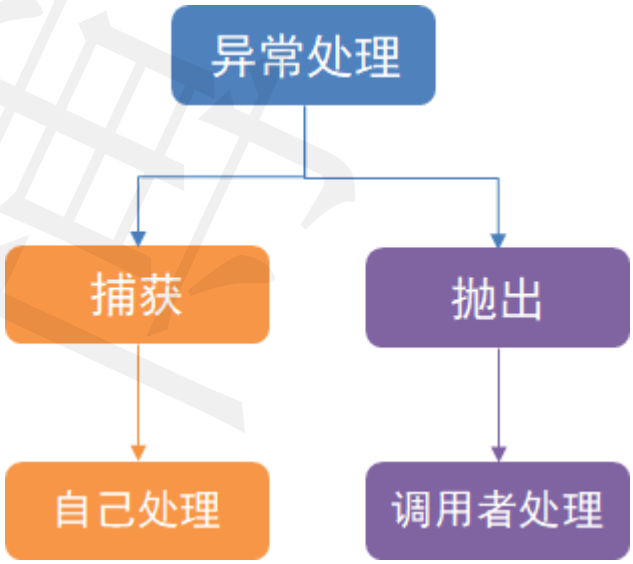
### 异常（Exception）

合理的应用程序可能需要捕获的问题 举例：NullPointerException

### 错误（Error）

合理的应用程序不应该试图捕获的问题 举例：StackOverFlowError

# 异常的处理方式




## JVM默认的异常处理方式

在控制台打印错误信息，并终止程序

## 开发中异常的处理方式

try...catch（finally）：捕获，自己处理

```
try {
  // 尝试执行的代码
} catch(Exception e) {
 // 出现可能的异常之后的处理代码
} finally {
  // 一定会执行的代码，如关闭资源
}
```

throws：抛出，交给调用者处理

```
public void 方法名() throws Exceptoin {

}
```

## 异常处理的注意事项

### 多个异常分别处理：

```
try {
    // 尝试执行的代码
} catch(异常A e) {
    // 出现可能的异常之后的处理代码
} catch(异常B e) {
    // 出现可能的异常之后的处理代码
} finally {
    // 一定会执行的代码，如关闭资源
}
```

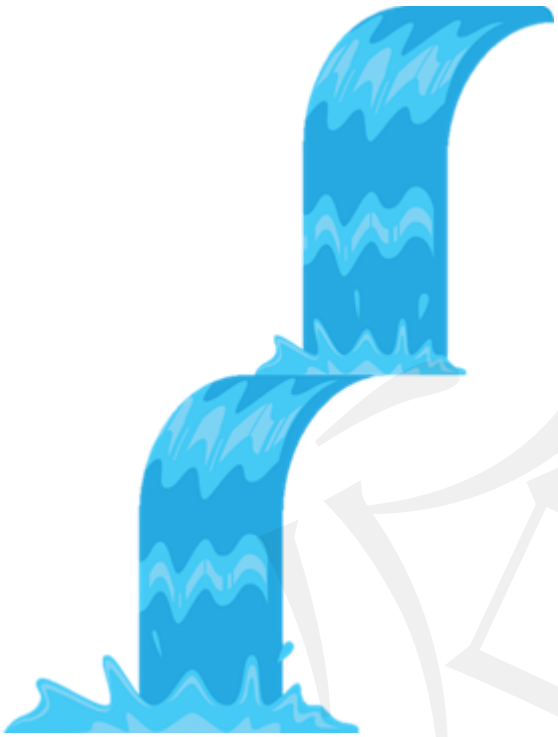
### finally代码块：

可以省略（不能和catch部分同时省略） finally代码之前若有return语句，先执行return语句，再执行finally代码块，最后返回return的结果

### 方法重写：

子类方法不能比父类方法抛出更大的异常

# IO流概述



## 什么是IO流?

I/O，即输入（Input）输出（Output），IO流指的是数据像连绵的流体一样进行传输。

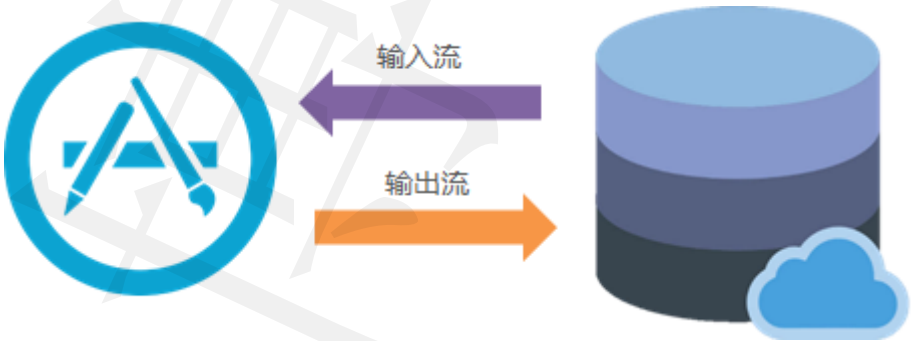
## IO流能干什么?

在本地磁盘或网络上传输（读/写）数据

## IO流的分类

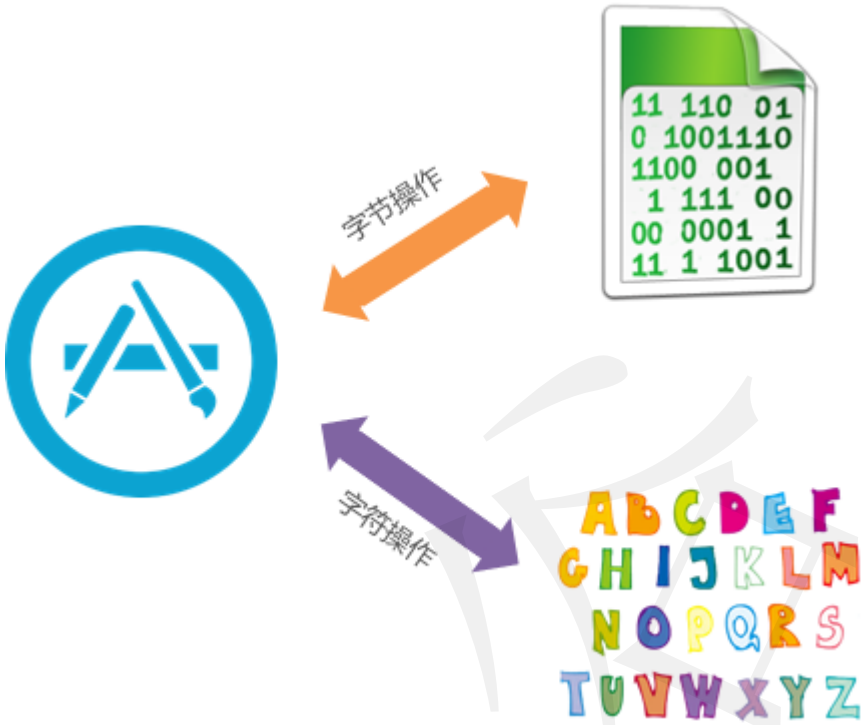
### 按数据流向分:

输入流 输出流




### 按操作方式分:

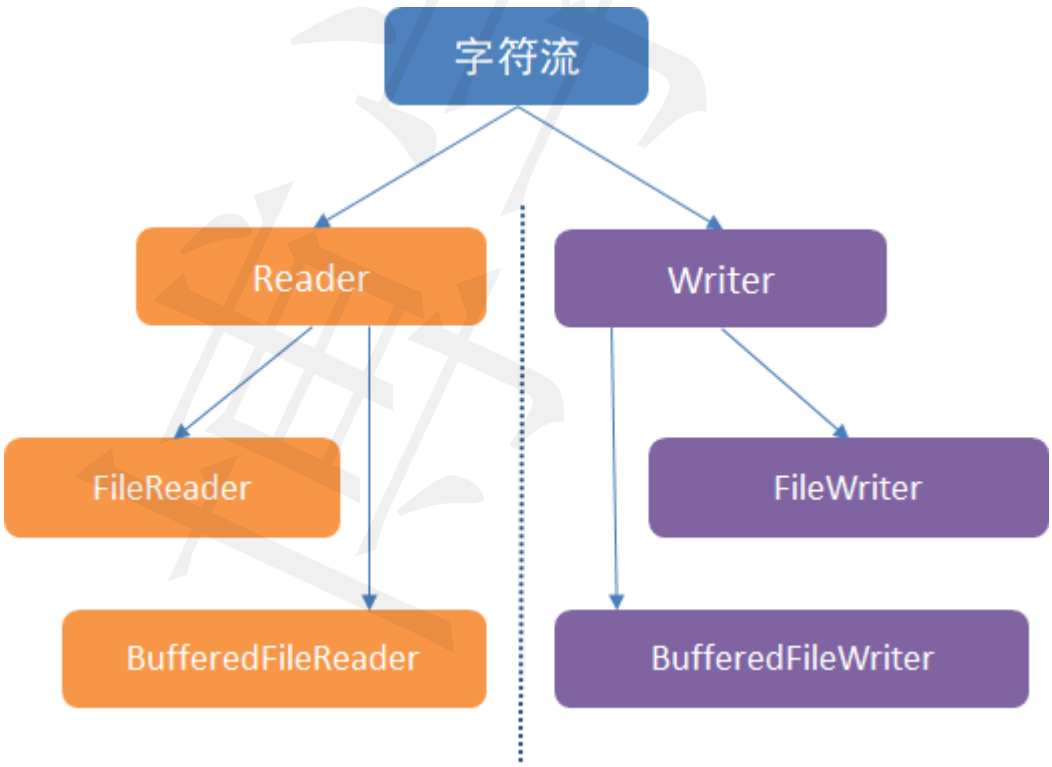
字节流： InputStream OutputStream 字符流： Reader Writer




## IO流体系

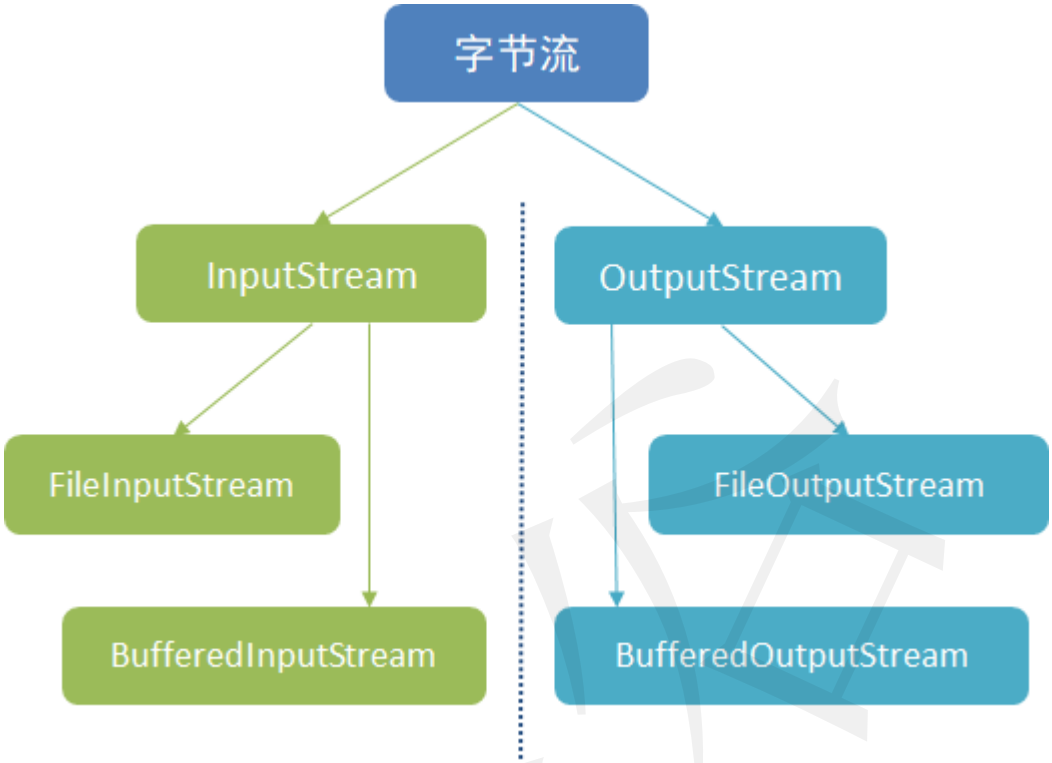
### 字符流：按字符读写数据的IO流

Reader FileReader BufferedFileReader Writer FileWriter BufferedFileReader




### 字节流：按字节读写数据的IO流

InputStream FileInputStream BufferedInputStream OutputStream FileOutputStream BufferedOutputStream




## File类

### 概念

文件，文件夹，一个File对象代表磁盘上的某个文件或文件夹

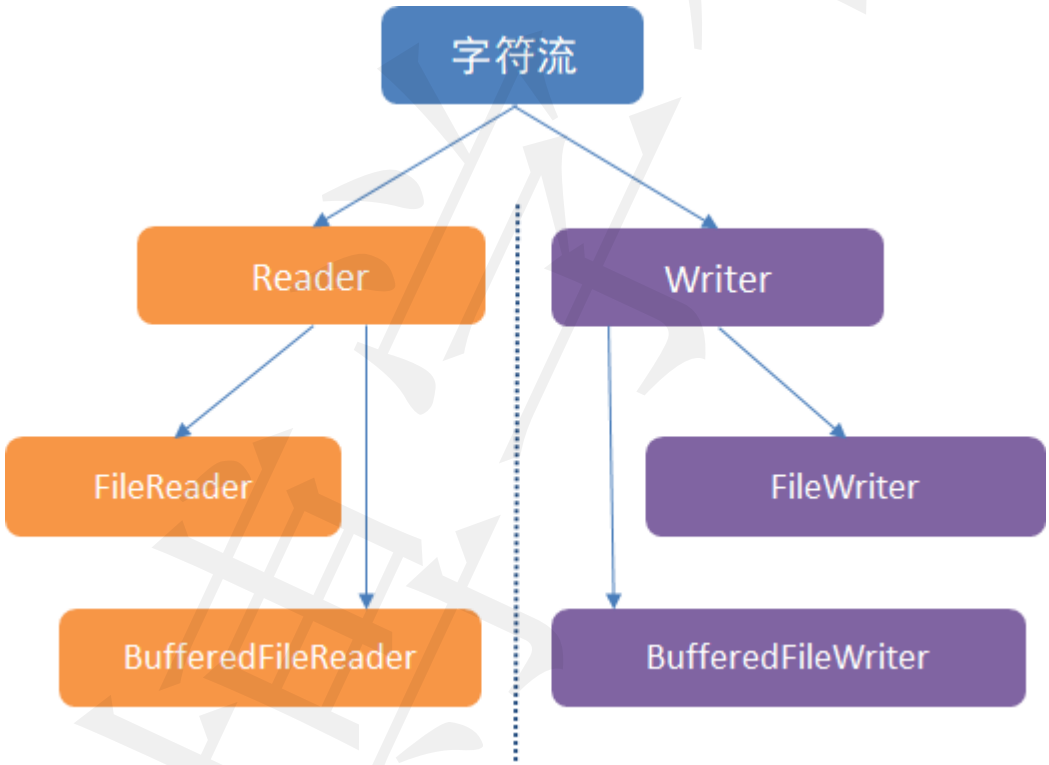
### 构造方法：

```
File(String pathname)
File(String parent, String child)
File(File parent, String child)
```

### 成员方法：

createNewFile(): 创建文件
mkdir()和mkdirs(): 创建目录
isDirectory(): 判断File对象是否为目录
isFile(): 判断File对象是否为文件
exists(): 判断File对象是否存在
getAbsolutePath(): 获取绝对路径
    从本地磁盘开始的路径
    举例: C:\Users\itcast\Desktop
getPath(): 获取文件的相对路径
    相对某一位置的路径
    Java项目中，相对路径从项目名开始
getName(): 获取文件名
list(): 获取指定目录下所有文件（夹）名称数组
listFiles(): 获取指定目录下所有文件（夹）File数组

# 字符流读写文件




## 字符流读数据 - 按单个字符读取

### 创建字符流读文件对象:

```
Reader reader = new FileReader("readme.txt");
```

### 调用方法读取数据:

```
int data = reader.read(); // 读取一个字符，返回该字符代表的整数，若到达流的末尾，返回-1
```

### 异常处理：

```
try {
    // 尝试执行的代码
} catch(Exception e) {
    // 出现异常后的处理代码
}
// 关闭资源：
finally {
     reader.close();
}
```

## 字符流读数据 - 按字符数组读取

### 创建字符流读文件对象：

```
Reader reader = new FileReader("readme.txt");
```

### 调用方法读取数据：

```
// 读取字符到数组中，返回读取的字符数，若到达流的末尾，返回-1
char[] chs = new char[2048];
int len = r.read(chs);
```

### 异常处理：

```
try {
    // 尝试执行的代码
} catch(Exception e) {
    // 出现异常后的处理代码
}
// 关闭资源：
finally {
     reader.close();
}
```

## 字符流写数据 - 按单个字符写入

### 创建字符流写文件对象：

```
Writer writer = new FileWriter("dest.txt");
```

### 调用方法写入数据：

```
int x = '中';
writer.write(x); // 写一个字符
```

**异常处理：**

```
try {
    // 尝试执行的代码
} catch(Exception e) {
    // 出现异常后的处理代码
}
// 关闭资源:
finally {
     writer.close();
}
```

## 字符流写数据 - 按字符数组写入

**创建字符流写文件对象：**

```
Writer writer = new FileWriter("dest.txt");
```

**调用方法写入数据（写入字符数组）：**

```
char[] chs = {'橙', '心', '橙', '意'};
writer.write(chs); // 写一个字符数组
```

**调用方法写入数据（写入字符串）：**

```
writer.write("小黑爱学习"); // 写入一个字符串
```

**异常处理：**

```
try {
    // 尝试执行的代码
} catch(Exception e) {
    // 出现异常后的处理代码
}
// 关闭资源:
finally {
     writer.close();
}
```

字符流读、写数据的常见操作到这里就介绍完了。接下来稍微提高一点难度：使用字符流拷贝文件。

## 字符流拷贝文件 - 按单个字符读写

### 创建字符流读文件对象：

```
Reader reader = new FileReader("readme.txt");
```

### 创建字符流写文件对象：

```
Writer writer = new FileWriter("dest.txt");
```

### 调用方法读取数据：

```
int data = reader.read();
```

### 调用方法写入数据：

```
writer.write(data);
```

### 异常处理：

```
throws IOException
```

### 关闭资源：

```
finally {
    reader.close();
    writer.close();
}
```

每次读取一个字符，效率太低，如果能一次性读取多个字符，效率是不是会提高呢？答案是肯定的。

## 字符流拷贝文件 – 按字符数组读写

### 创建字符流读文件对象：

```
Reader reader = new FileReader("readme.txt");
```

### 创建字符流写文件对象：

```
Writer writer = new FileWriter("dest.txt");
```

### 调用方法读取数据：

```
char[] chs = new char[2048];•int len = reader.read(chs);
```

### 调用方法写入数据：

```
writer.write(chs, 0, len);
```

### 异常处理：

```
throws IOException
```

### 关闭资源：

```
finally {
    reader.close();
    writer.close();
}
```

字符流操作数据的基本用法介绍完了。在实际生产环境中，流的操作非常的缓慢、耗时（打开资源、操作资源、关闭资源），所以，实际生产环境中的流操作对效率的要求很高。为此，Java的设计者们提供了高效的缓冲流供开发者使用。

## 字符缓冲流拷贝文件的标准代码

### 创建字符流读文件对象：

```
BufferedReader br = new BufferedReader(
    new FileReader("readme.txt"));
```

### 创建字符流写文件对象：

```
BufferedWriter bw = new BufferedWriter(
    new FileWriter("dest.txt"));
```

### 异常处理：

```
throws IOException
```

### 使用while循环读写数据：

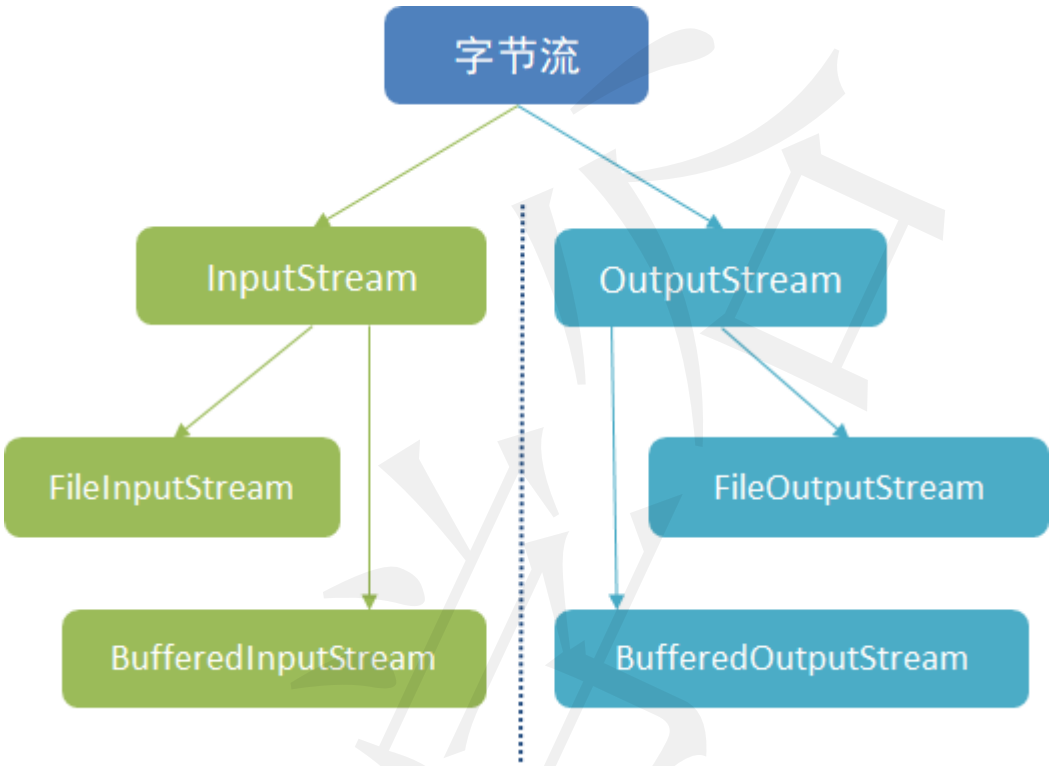
```
char[] chs = new char[2048];
int len;
while((len = br.read(chs)) != -1) {
    bw.write(chs, 0, len);
}
```

### 关闭资源：

```
br.close();

bw.close();
```

## 字节流读写文件




### 字节流拷贝文件 – 按单个字节读写

**创建字节流读文件对象：**

```
InputStream is = new FileInputStream("Desktop.jpg");
```

**创建字节流写文件对象：**

```
OutputStream os = new FileOutputStream("D:\\博学谷桌面.jpg");
```

**异常处理：**

```
throws IOException
```

**使用while循环读写数据：**

```
int b;
while((b = is.read()) != -1) {
    os.write(b);
}
```

**关闭资源：**

```
is.close();

os.close();
```

## 字节流拷贝文件 – 按字节数组读写

**创建字节流读文件对象：**

```
InputStream is = new FileInputStream("Desktop.jpg");
```

**创建字节流写文件对象：**

```
OutputStream os = new FileOutputStream("D:\\博学谷桌面.jpg");
```

**异常处理：**

```
throws IOException
```

**定义字节数组，每次读取2048个字节：**

```
byte[] b = new byte[2048];
```

**使用while循环读写数据：**

```
int len;

while((len = is.read(b)) != -1) {
    os.write(b, 0, len);
}
```

**关闭资源：**

```
is.close();

os.close();
```

## 字节缓冲流拷贝文件的标准代码

### 创建字节缓冲流读文件对象：

```
BufferedInputStream bis = new BufferedInputStream(
    new FileInputStream("jdk-11.0.1_doc-all.zip"));
```

### 创建字节缓冲流写文件对象：

```
BufferedOutputStream bos = new
    BufferedOutputStream(new FileOutputStream("D:\\jdk-11.0.1_doc-all.zip"));
```

### 异常处理：

```
throws IOException
```

### 定义字节数组，每次读取2048个字节：

```
byte[] bs = new byte[2048];
```

### 使用while循环读写数据：

```
int len;

while((len = bis.read(bs)) != -1) {

    bos.write(bs, 0, len);

}
```

### 关闭资源：

```
bis.close();

bos.close();
```

## 案例：模拟文件上传功能

### 需求：使用控制台模拟实际开发中上传用户头像的功能

### 分析：

1. 在控制台录入用户头像的路径
2. 解析路径字符串中文件名是否合法：
3. 后缀名为：.jpg、.png、.bmp
4. 判断该路径表示的File对象是否存在，是否为文件：

   file.exist()

   file.isFile()

5. 读取该文件并写入到指定目录
6. 提示头像上传成功 或 失败