IO流

天津卓讯科技有限公司

第10章 IO流

- 10.1 File类
- 10.2 输入/输出基本原理
- 10.3 流类概述
- 10.4 文件流
- 10.5 缓冲流
- 10.6 转换流
- 10.7 对象流
- 10.8 其它流
  - 数据流、打印流 、随机访问文件
- 10. 9 try-with-resources语句

10.1 java.io.File类

- File类代表文件（文件和目录）
  - 存储介质中的文件和目录在Java程序中都是用File类的实例来表示。

- 常用构造方法：
  - public File(String pathname)：以pathname为路径创建File对象
    - 绝对路径：文件或目录在存储介质或网络中的真正路径(物理路径)
    - 相对路径：相对于系统属性“user.dir”这个基准目录的路径。(当前字节码运行时所在的目录)
- File类的一个常用属性
  - public static final String separator存储了当前系统的路径分隔符
    - 在 UNIX 系统上，此字段的值为 '/'；在 Windows 系统上为 '\'
    - 为了程序的跨平台特性，文件的路径应该用这个属性值来代表。

10.1.2 File类的常用方法

- 访问File对象的属性：
  - public boolean canRead()
  - public boolean canWrite()
  - public boolean exists()
  - public boolean isDirectory()
  - public boolean isFile()
  - public boolean isHidden()
  - public long lastModified() //毫秒值
  - public long length() //以字节为单位
  - public String getName() //获取文件名
  - public String getPath() //路径名
  - public String getAbsolutePath() //返回此File对象的绝对路径名
  - public File getAbsoluteFile()
  - public String getCanonicalPath() //返回此File对象的规范路径名字符串
  - public File getCanonicalFile() //返回此File对象的规范形式
  - public String getParent() //返回父目录的路径名字符串
  - public URI toURI() //返回此文件的统一资源标识符名

示例

```
import java.io.File;
public  class FileTest{
    public static void main(String[] args){
        File file = new File(args[0]);
        System.out.println("文件或目录是否存在:" +  file.exists());
        System.out.println("是文件吗:" +  file.isFile());
        System.out.println("是目录吗:" +  file.isDirectory());
        System.out.println("名称:" +  file .getName());
        System.out.println("路径: " + file.getPath());
        System.out.println("绝对路径: " + file.getAbsolutePath());
        System.out.println("最后修改时间:" + file.lastModified());
        System.out.println(“文件大小:” + file.length()+ “ 字节”);
        ……
    }
}
```

10.1.2 File类的常用方法

- 浏览目录中的子文件和子目录
  - public String[] list()　//返回此目录下的文件名和目录名的数组
  - public File[] listFiles()//返回此目录下的文件和目录File实例数组
  - public File[] listFiles(FilenameFilter filter) //返回此目录中满足指定过滤器的文件和目录
    - java.io.FilenameFilter接口：实现此接口的类实例可用于过滤文件名
- 对文件的操作：
  - public boolean createNewFile() //不存在时创建此文件对象所代表的空文件
  - public boolean delete()　//删除文件或目录。目录必须是空才能删除
  - public boolean mkdir()　//创建此抽象路径名指定的目录
  - public boolean mkdirs()　//创建此抽象路径名指定的目录，包括所有必需但不存在的父目录
  - public boolean renameTo(File dest)　//重命名

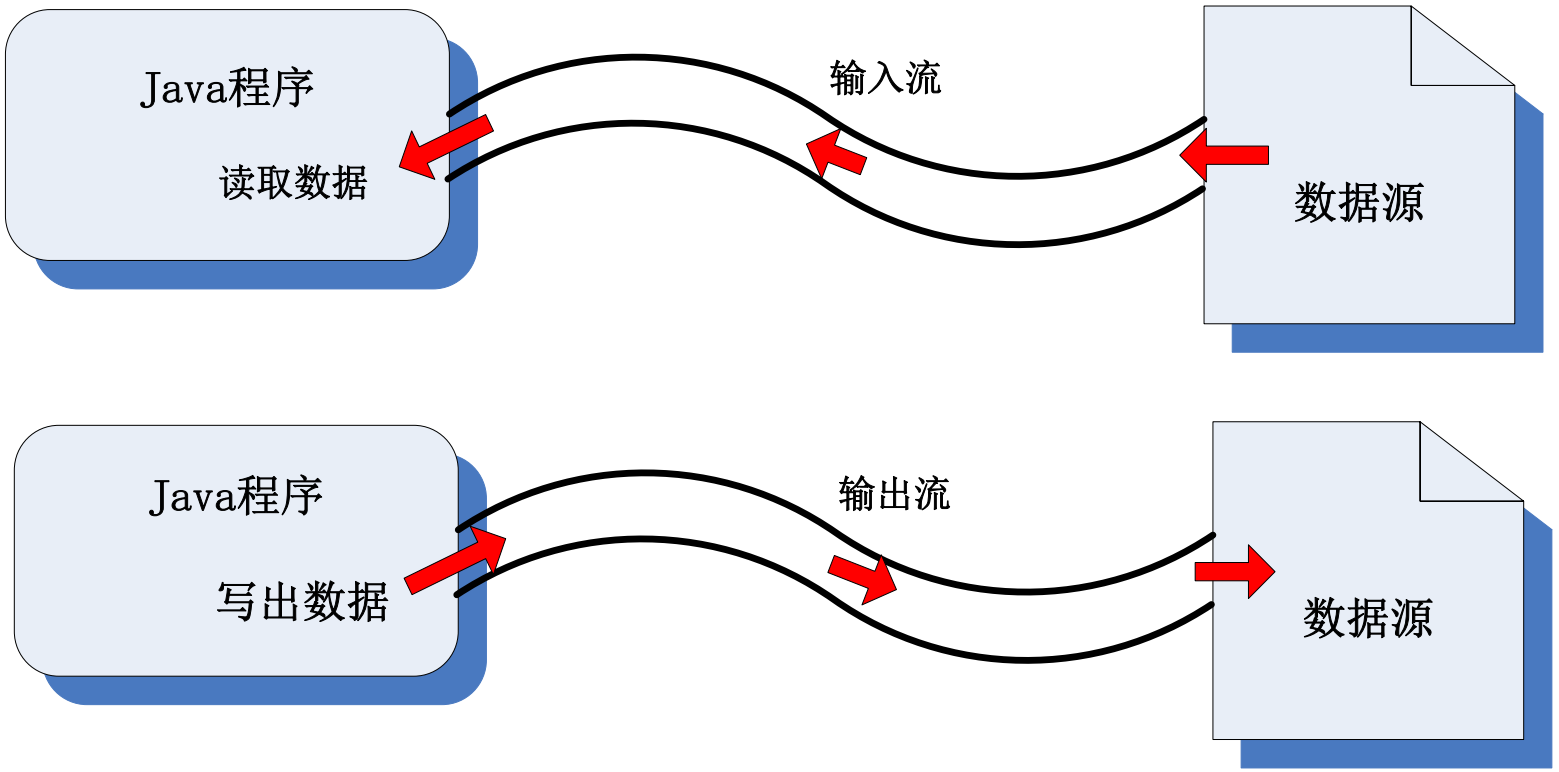
思考

- 用递归算法列出指定目录下的所有子孙文件和目录
- 列出指定目录下的jpg类型图片。

- 删除一个目录的过程是如何进行的？

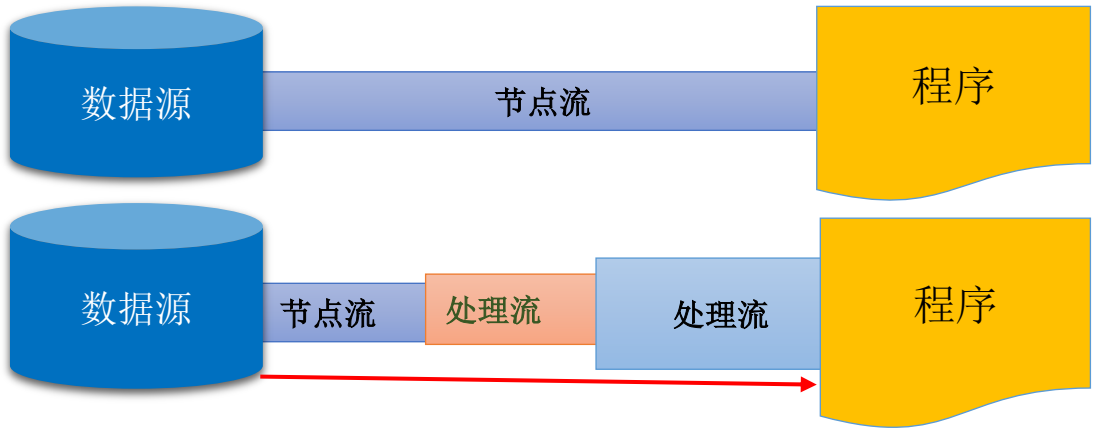
10.2 输入/输出基本原理

- 数据流(Stream)是指数据通信的通道。
- java程序中对数据的输入、输出操作是以“流”方式进行的。JDK中提供了各式的“流”类来获取不同种类的数据。



10.3.1 流类概述

- 按流向分：
  - 输入流：程序可以从中读取数据的流
  - 输出流：程序能向其中写出数据的流
- 按数据传输单位分：
  - 字节流：以字节为单位
  - 字符流：以字符为单位
- 按功能分：
  - 节点流：用于直接操作目标设备的流
  - 过滤流(处理流)：是对一个已存在的流的连接和封装，通过对数据的处理为程序提供更为强大、灵活的读写功能。



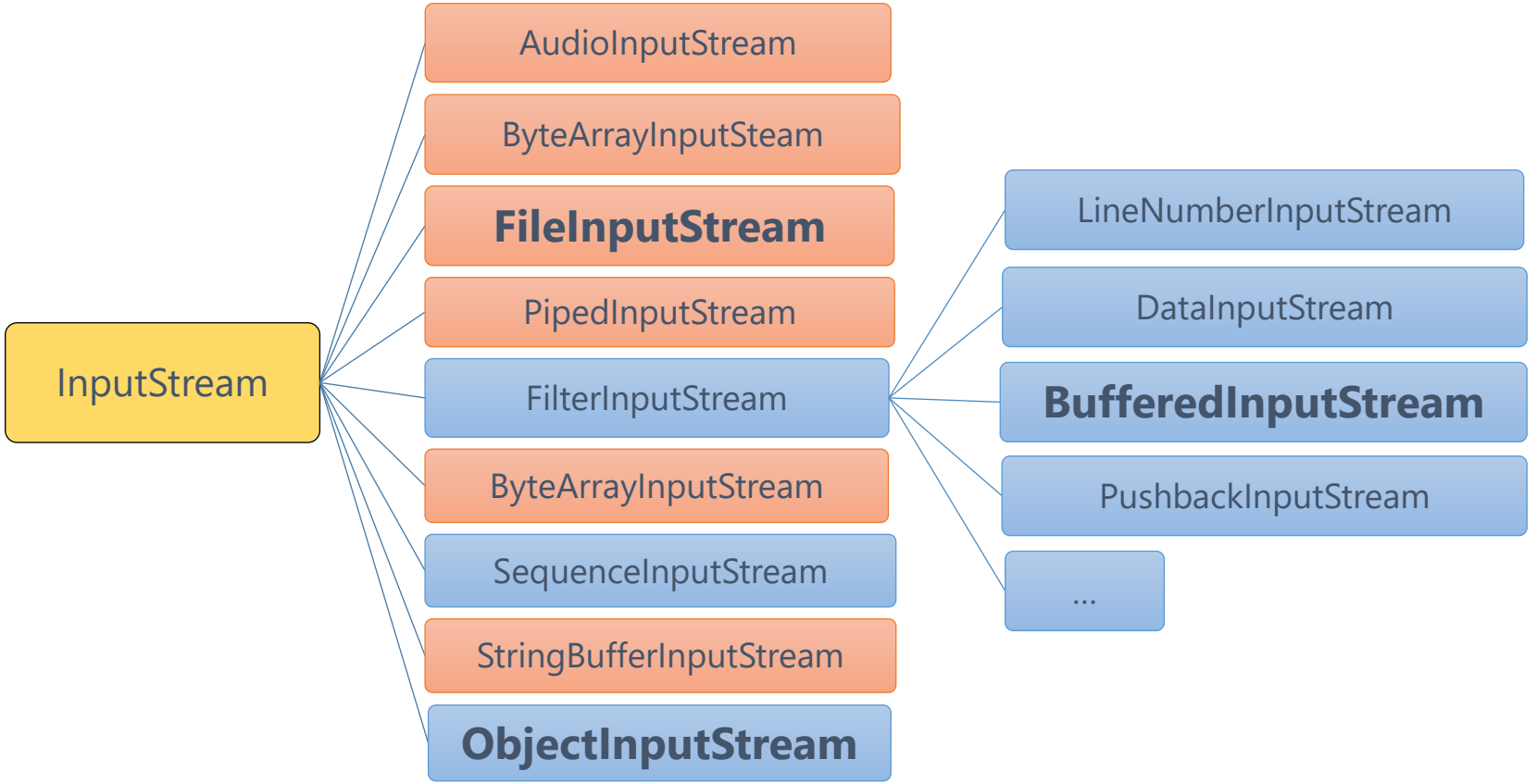
10.3.2 抽象流类

- JDK所提供的所有流类位于java.io包中，都分别继承自以下四种抽象流类。

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

InputStream抽象类

- 继承自InputStream的流都是用于向程序中输入数据的，且数据的单位为字节(8位)。（粉色为节点流，蓝色为过滤流）

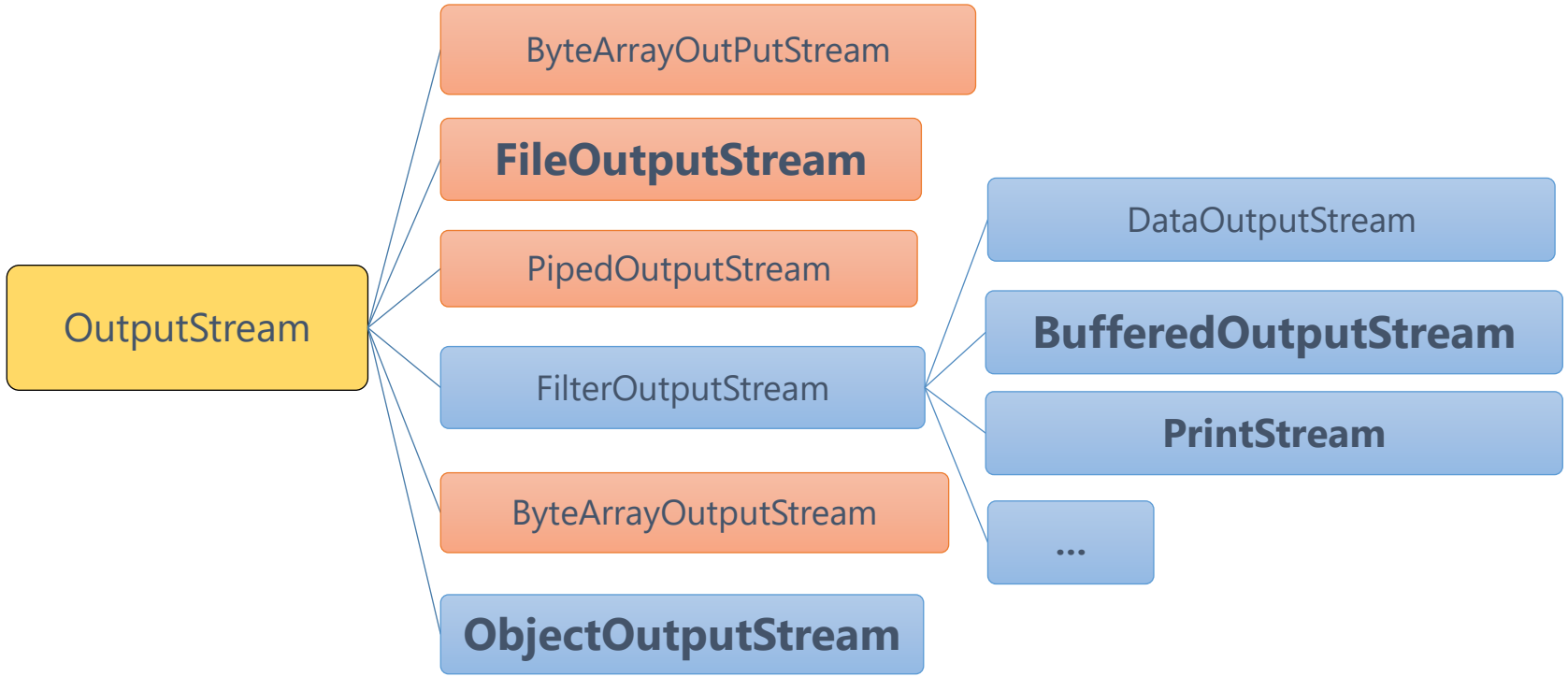


InputStream的基本方法

- public abstract int read() throws IOException
  - 从输入流中读取数据的下一个字节, 返回读到的字节值。若遇到流的末尾,返回-1
- public int read(byte[] b) throws IOException
  - 从输入流中读取b.length个字节的数据并存储到缓冲区数组b中。返回的是实际读到的字节总数。若遇到流的末尾,返回-1
- public int read(byte[] b, int off, int len) throws IOException
  - 读取 len 个字节的数据,并从数组b的off位置开始写入到这个数组中
- public void close() throws IOException
  - 关闭此输入流并释放与此流关联的所有系统资源
- public int available() throws IOException
  - 返回此输入流下一个方法调用可以不受阻塞地从此输入流读取（或跳过）的估计字节数
- public skip(long n) throws IOException
  - 跳过和丢弃此输入流中数据的 n 个字节，返回实现路过的字节数。

OutputStream抽象类

- 继承自OutputStream的流是程序用于向外输出数据的，且数据的单位为字节(8位)。

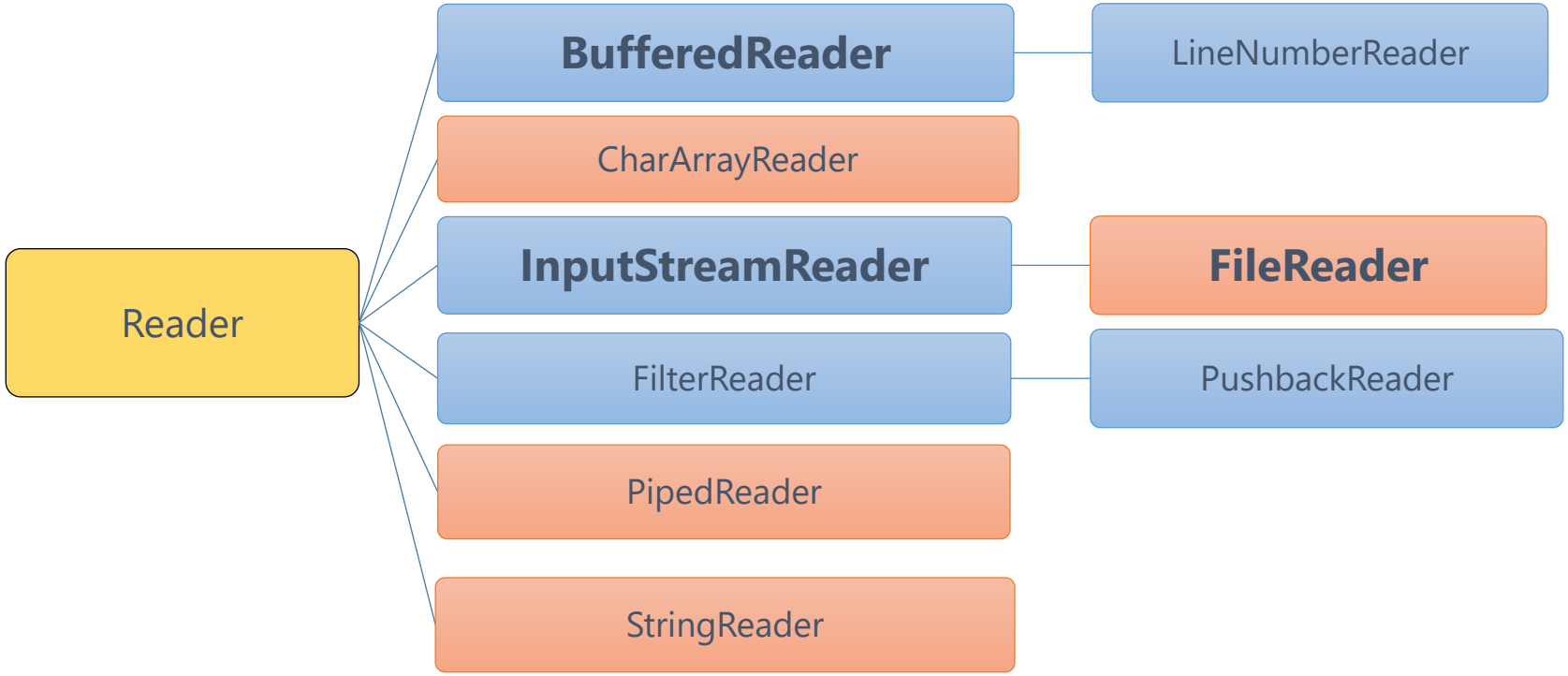


OutputStream的基本方法

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

Reader抽象类

- 继承自Reader的流都是用于向程序中输入数据的，且数据的单位为字符(16位)。

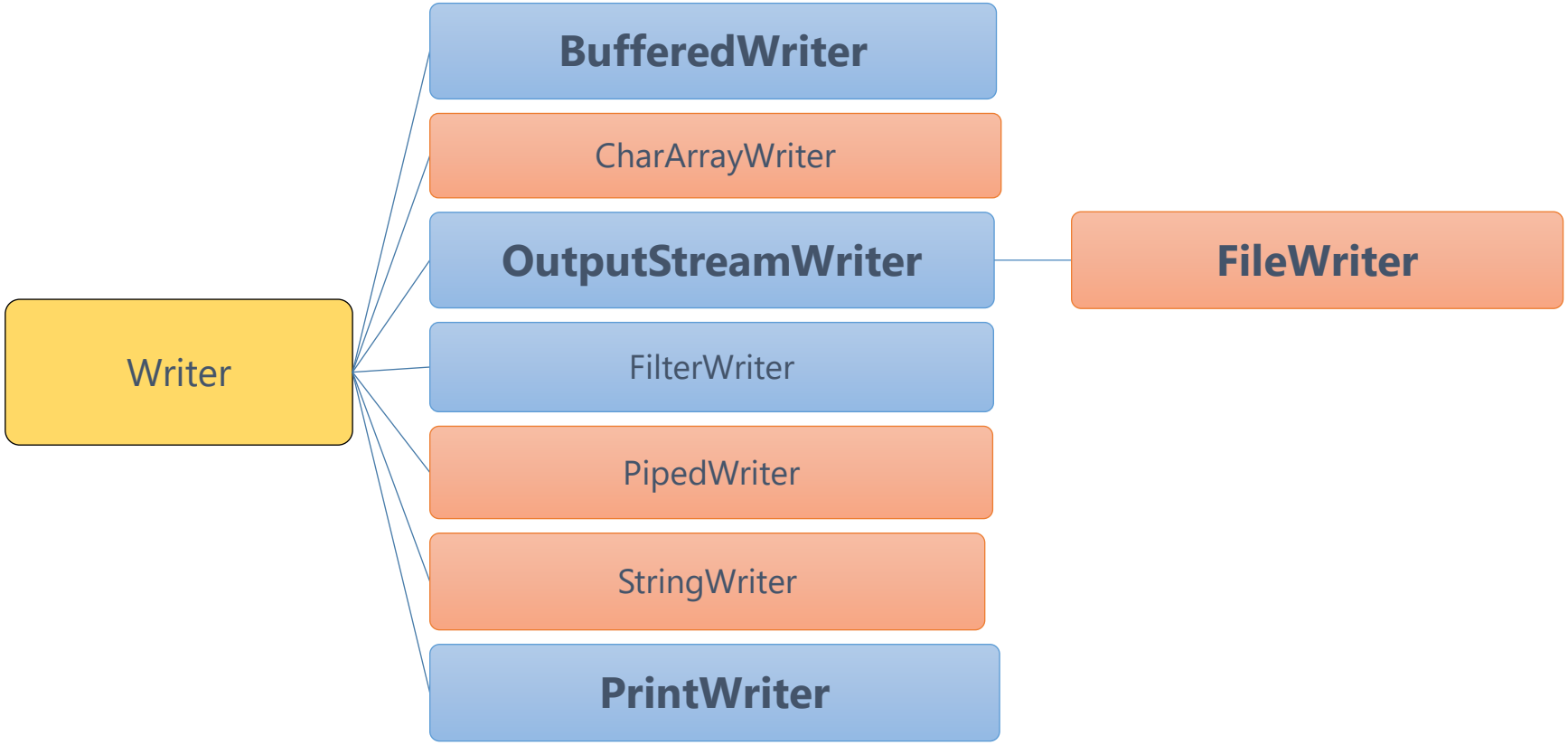


Reader的基本方法

- public int read() throws IOException
  - 读取单个字符，返回作为整数读取的字符，如果已到达流的末尾返回-1
- public int read(char[] cbuf) throws IOException
  - 将字符读入数组，返回读取的字符数；否则返回-1
- public abstract int read(char[] cbuf, int off, int len) throws IOException
  - 读取len个字符，并从数组cbuf的off位置开始写入到这个数组中
- public abstract void close() throws IOException
  - 关闭该流并释放与之关联的所有资源
- public long skip(long n) throws IOException
  - 跳过n个字符。

Writer抽象类

- 继承自Writer的流是程序用于向外输出数据的，且数据的单位为字符(16位)。



Writer的基本方法

- public void write(int c) throws IOException
  - 写入单个字符
- public void write(char[] cbuf) throws IOException
  - 写入字符数组
- public abstract void write(char[] cbuf, int off, int len) throws IOException
  - 写入字符数组的某一部分
- public void write(String str) throws IOException
  - 写入字符串
- public void write(String str, int off, int len) throws IOException
  - 写字符串的某一部分
- public abstract void flush() throws IOException
  - 刷新该流的缓冲，将缓冲的数据全写到目的地
- public abstract void close() throws IOException
  - 关闭此流，但要先刷新它

104 文件流

- 专门用于操作文件的流。Java SE API中提供了4种：
  - FileInputStream继承自InputStream
  - FileOutputStream继承自OutputStream
  - FileReader继承自Reader
  - FileWriter继承自Writer

- 二进制文件（字节文件）：图片、音频、视频等。
- 文本文件（字符文件）：*.txt、*.properties、*.xml、*.html
- 文件续写问题

示例: 字节流完成文件复制功能

```
int b = 0;
FileInputStream in = null;
FileOutputStream out = null;
try {
    in = new FileInputStream("d:/IOTest/soruce.jpg");
    out = new FileOutputStream("d:/IOTest/dest.jpg");
    while ((b = in.read()) != -1) {   //有没有效率更高的方式?
        out.write(b);
    }
    System.out.println("文件复制成功");
} catch (FileNotFoundException e) {
    System.out.println("找不到指定文件");
    e.printStackTrace();
} catch (IOException e) {
    System.out.println("文件复制错误");
    e.printStackTrace();
}finally{  ...}
```

练习

- 用FileReader和FilerWriter实现文件内容的复制功能

10.5 缓冲流—过滤流

- 缓冲流是建立在相应的节点流之上，对读写的数据提供了缓冲的功能，提高了读写的效率，还增加了一些新的方法。Java SE API提供四种缓冲流：
  - BufferedInputStream 可以对任何的InputStream流进行包装(套接)
  - BufferedOutputStream 可以对任何的OutputStream流进行包装(套接)
  - BufferedReader 可以对任何的Reader流进行包装
    - 新增了readLine()方法用于一次读取一行字符串('\r'或'\n'作为行结束)
  - BufferedWriter 可以对任何的Writer流进行包装
    - 新增了newLine()方法用于写入一个行分隔符。
- 注意：
  - 对于缓冲输出流，写出的数据会先缓存在内存缓冲区中，关闭此流前要用flush()方法将缓存区的数据立刻写出。
  - 关闭过滤流时，会自动关闭过滤流所包装（套接）的所有底层流。

示例

```
BufferedInputStream bis = null;
BufferedOutputStream bos = null;
byte[]  buf = new byte[1024];
try{
  bis = new BufferedInputStream(new FileInputStream(src));
  bos = new BufferedOutputStream(new FileOutputStream(dest));
  for(int len = 0; (len = bis.read(buf)) != -1;){
    bos.write(buf, 0, len);
  }
  bos.flush();
}catch(IOException e){
  e.printStackTrace();
}finally{
  if(bos != null){
    try {bos.close();} catch (IOException e) {e.printStackTrace();}
  }
  if(bis != null){
    try {bis.close();} catch (IOException e) {e.printStackTrace();}
  }
}
```

练习

- 用缓冲流来改写文本文件复制功能

10.6 转换流——过滤流

- 转换流用于在字节流和字符流之间转换。JaveSE API提供了两种转换流：
  - InputStreamReader需要和InputStream“套接”，它可以将字节流中读入的字节解码成字符
  - OutputStreamWriter需要和OutputStream“套接”，它可以将要写入字节流的字符编码成字节
- 常用于：
  - 解决乱码问题
  - 网络编程中，只提供字节流。

示例

```
System.out.println("请输入信息(退出输入e):");
InputStreamReader isr = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(isr);
String s = null;
try {
    while ((s = br.readLine()) != null) {  //阻塞程序
        if (s.equals("e")) {
            System.out.println("安全退出!!");
            break;
        }
        System.out.println("-->:"+s.toUpperCase());
        System.out.println("继续输入信息");
    }
} catch (IOException e) {
    e.printStackTrace();
} finally {
    try {       if (null != br) {    br.close();  }    } catch (IOException e) {       e.printStackTrace();    }
}
```

字符乱码问题

- 使用字符流读取不同编码类型的字符时，会出现乱码问题
  - 解码：字节→字符　　编码：字符→字节

- JVM默认的字符集：跟你的源文件的编码有关。
  - Charset cs = Charset.defaultCharset();

常用字符集概述

- 字符集(charset)：是一套文字符号及其编码的集合。
- ASCII 美式字符集
  - 单字节7位编码。共可表示127个字符 a→ 97
  - 大小写英文字母、阿拉伯数字和标点符号以及33个控制符号
- ISO-8859-1(Latin1) 西欧字符集
  - 单字节8位编码。共可表示255个字符 a→97
  - 与ASCII编码兼容

中文字符集

- GB2312
  - 双字节编码
  - 包括对简体中文字符的编码，一共收录了7445个字符，包括6763个汉字和682个其他符号，未收录繁体中文汉字和一些生僻字。
  - 与ISO-8859-1字符编码兼容
  - 早期中文标准
- GBK
  - 双字节编码
  - 完全兼容GB2312，收录了21886个字符，包括繁体和简体中文字符
  - GB2312的扩展。使用广泛
- GB18030
  - 2或4字节编码
  - 向下兼容 GBK 和 GB2312 标准。收录了27484个汉字
  - 中文新标准。目前使用还不广泛

Unicode字符集

- Unicode字符集包括全世界所有语言的文字和符号。
  - 它为每种语言中的每个字符设定了统一并且唯一的二进制编码

- Unicode的常用编码实现方案：
  - UTF-16：用固定4字节存储一个Unicode字符。
    - Java和windows xp内使用
  - UTF-8：用1~6个字节存储一个Unicode字符。
    - 互联网和UNIX/Linux广泛支持, MySQL Server内部也使用
    - 英文字母、数字和符号用1个字节存储→兼容ISO-8859-1
    - 中文用3个字节存储→不兼容任何GBXXX

10.7 对象流

- 用于存储和读取基本类型数据或对象的过滤流
  - ObjectOutputStream保存基本类型数据或对象：序列化
  - ObjectInputStream读取基本类型数据或对象：反序列化
- 能被序列化的对象所对应的类必须实现java.io.Serializable这个标识性接口

示例

```
Student stu = new Student(101, "张三", 22);
FileOutputStream fos = null;
try {
    fos = new FileOutputStream("d:/IOTest/os.dat");
} catch (FileNotFoundException e) {
    e.printStackTrace();
    System.exit(-1);
}
ObjectOutputStream oos = null;
try {
    oos = new ObjectOutputStream(fos);
    oos.writeObject(stu);
    oos.writeDouble(123.456);
    System.out.println("序列化成功!!!");
} catch (IOException e) {
    e.printStackTrace();
}finally{
}
```

```
FileInputStream fis = null;
try {
    fis = new FileInputStream("d:/IOTest/os.dat");
} catch (FileNotFoundException e) {
    e.printStackTrace();
    System.exit(-1);
}
ObjectInputStream ois = null;
try {
    ois = new ObjectInputStream(fis);
    Student stu = (Student)ois.readObject();
    System.out.println(stu);
    System.out.println(ois.readDouble());
} catch (IOException e) {e.printStackTrace();
} catch (ClassNotFoundException e) {e.printStackTrace();
}finally{
}
```

序列化的相关问题

- transient关键字修饰成员变量时，表示这个成员变量是不需要序列化的。
- static修饰的成员变量也不会被序列化。
- 实现了Serializable接口的类都应该生成一个private static final long serialVersionUID 序列化版本ID作为标识。

10.8 其它流类

- 数据流
  - DataInputStream和DataOutputStream
- 打印流
  - PrintStream和PrintWriter都属于打印流
  - 它们的输出操作永远也不会抛出IOException
- 随机访问文件
  - RandomAccessFile类，自身具备读写的方法
    - void seek(long pos); //文件指针移动到指定的指针偏移量
    - long getFilePointer(); //获取当前文件指针偏移量
    - int read(byte[] b); //读数据
    - void write(byte[] b, int off, int len); //写数据

10.9 try-with-resources语句

- JDK7中提供的，它可以自动关闭相关的资源的语法：

```
try (资源声明1; 资源声明2; ...) {
    ...
}catch(...){
}finally{
...
}
```

- 资源是指实现了java.lang.AutoCloseable或java.io.Closeable的类的对象。如：InputStream、OutputStream、Reader、Writer、Connection、ResultSet等。
- 它会确保在本语句结束时关闭try语句中声明的所有资源。资源被关闭的顺序与它们被创建的顺序相反。

示例

```
public static void copy(File src, File dest){
  byte[]  buf = new byte[1024];
  try(BufferedInputStream bis = new BufferedInputStream(new FileInputStream(src));
  BufferedOutputStream bos=new BufferedOutputStream(new FileOutputStream(dest))){
     for(int len = 0; (len = bis.read(buf)) != -1;){
        bos.write(buf, 0, len);
     }
     bos.flush();
  }catch (Exception e) {
     e.printStackTrace();
  }
}
```

总结

- File类
- IO流分类
- 文件流
  - FileInputStream/FileOutputStream
  - FileReader/FileWriter
- 缓冲流
  - BufferedInputStream/BufferedOutputStream
  - BufferedReader/BufferedWriter
- 转换流：InputStreamReader/OutputStreamWriter
- Object流：ObjectInputStream/ObjectOutputStream

自学

- 操作zip压缩文件
  - ZipOutputStream类用于压缩
  - ZipInputStream类用于解压缩

- Apache组织提供的开源jar包：commons-io.jar