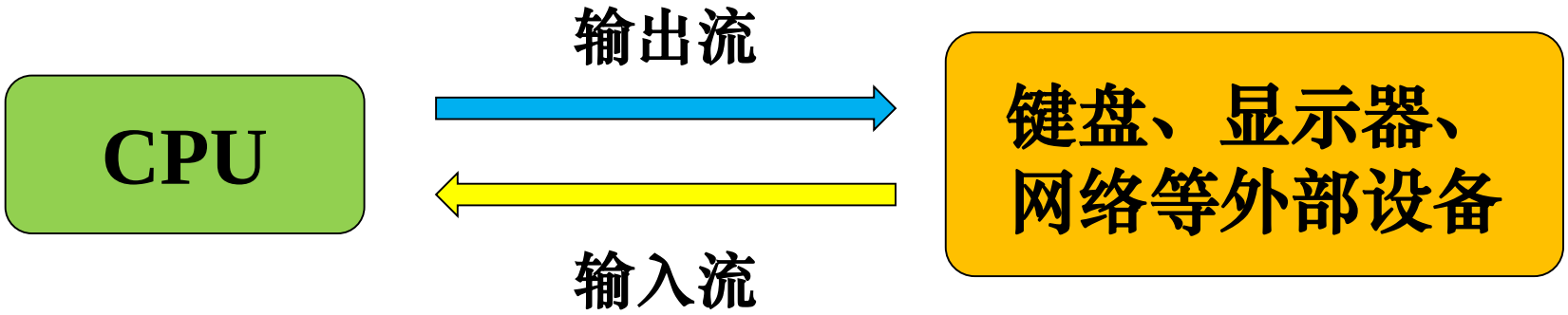
第12章 输入输出流及文件处理

第12章 输入输出流及文件处理

- I/O流
- 字节流
- 字符流
- 文件输入输出
- 对象序列化
- 正则表达式

12.1 I/O流

- 程序中经常会遇到数据输入输出，实际上是对数据源进行读和写的操作。
- Java中输入输出包括字节流、字符流、对象流以及多线程之间通信的管道流。
- System类中提供有标准输入System.in、标准输出System.out和错误输出流System.err。



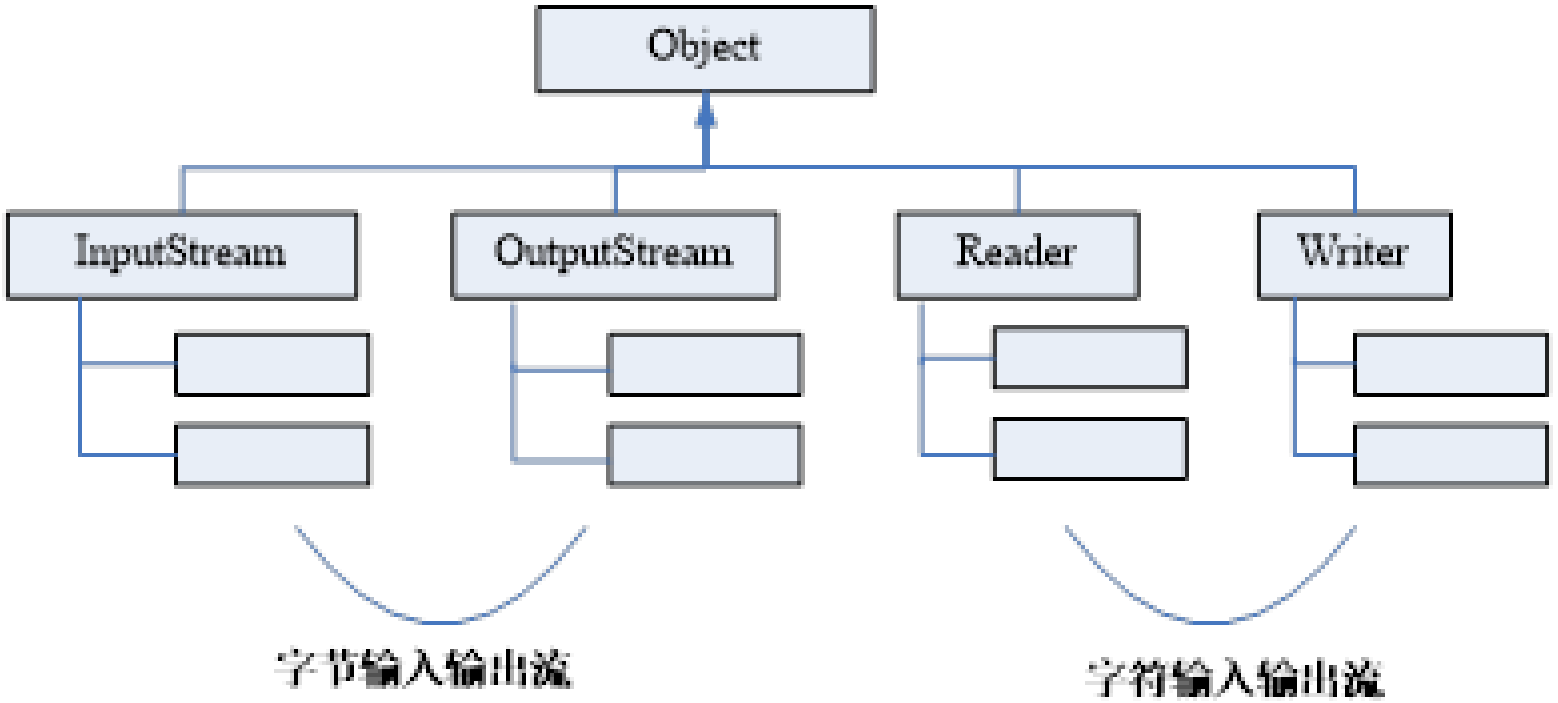
12.1 I/O流

Java中所有的I/O都是通过流来实现的，可以将流理解为连接到数据目标或源的管道，可以通过流读取或写入数据。

根据流的方向分类：输入流和输出流。

根据数据类型分类：
字节（byte）流和字符（Character）流。

Java中I/O流由java.io包封装，其中的类大致可以分为输入和输出两大部分。根据数据类型的不同，流又分为字节（byte）流（InputStream类和 OutputStream类），一次读写8位二进制数； 另一种是字符（Character）流（Reader类和Writer类），一次读写16位二进制数。Java.io包的类的关系如图所示。



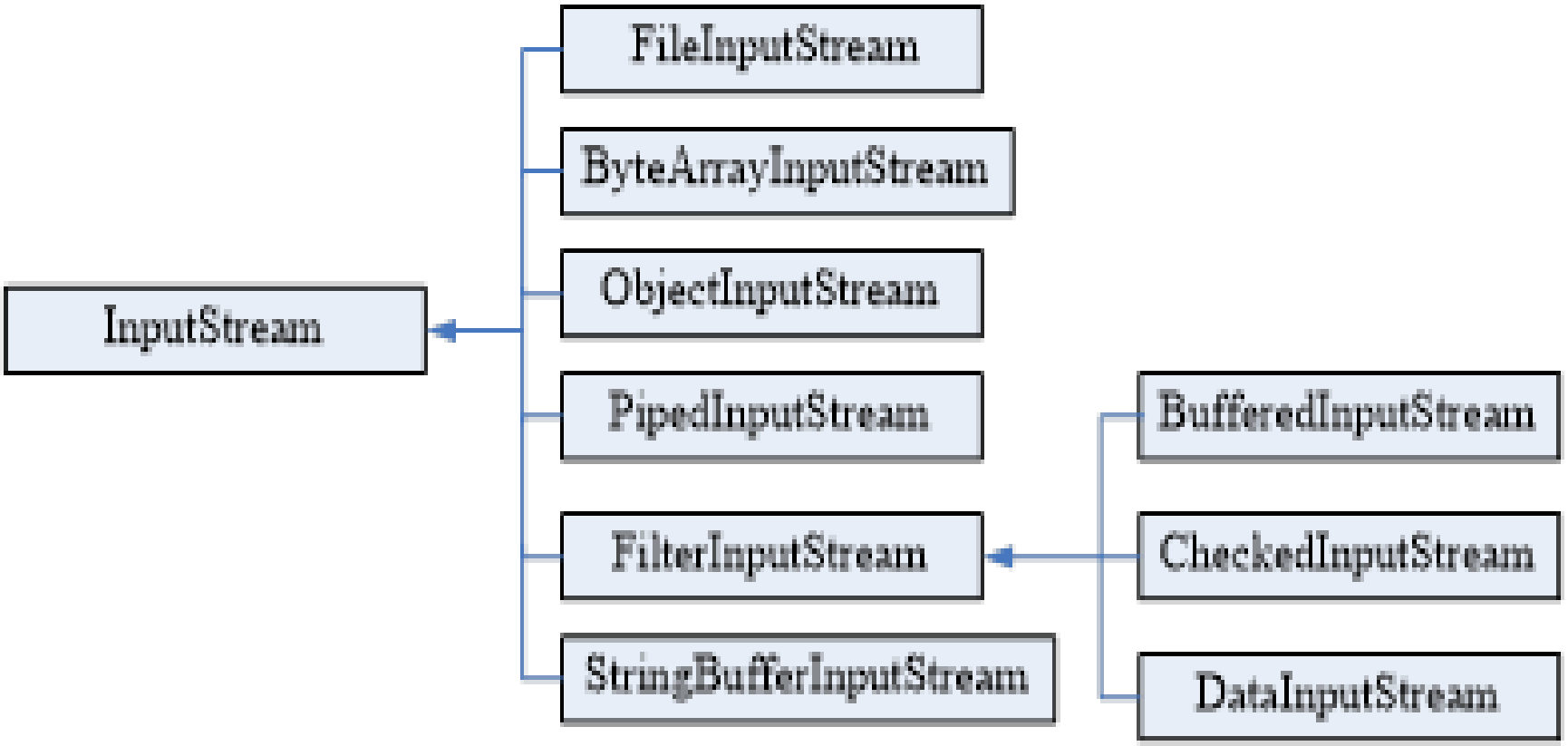
12.2 字节流

字节流是以 8 比特为单位的读和写：

- 字节输入流继承InputStream抽象类。
- 字节输出流继承 OutputStream抽象类。
- 两个抽象类定义了最基本的输入和输出功能。

12.2 字节流

字节输入流类关系图



12.2.2 InputStream类常用方法

- void mark()
  在输入流中标记当前的位置。
- boolean markSupported()
  测试输入流是否支持mark和reset方法。
- int read(byte[] b)
  从输入流中读取数据的下一个字节。
- int reset()
  将流重新定位到最后一次对此输入流调用mark方法时的位置。
- long skip(long n)
  跳过和丢弃输入流中的n个字节。

12.2.2 InputStream类常用方法

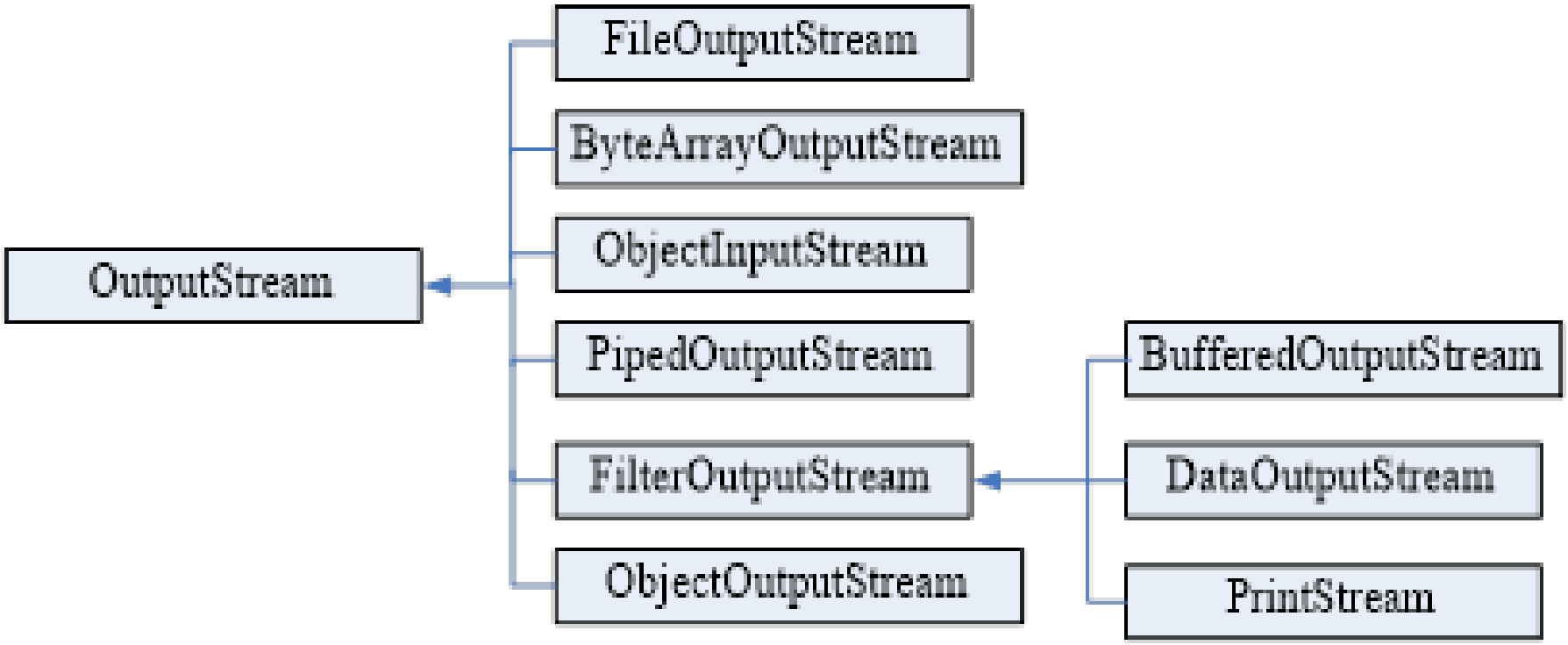
- int available()

  返回输入流被调用时可以读取（或跳过）的字节数。
- void close()

  关闭此输入流并释放与流关联的所有系统资源

12.2 字节流

字节输出流类关系图



12.2.2 OutputStream类常用方法

- void close()
  关闭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 byte 数组写入输出流。
- void write(byte[] b, int off, int len)
  将指定 byte 数组中从偏移量 off 开始的 len 个字节写入输出流。
- abstract void write(int b)
  将指定的字节写入输出流。

12.2.3 文件数据流

输入：FileInputStream类提供了多个构造方法

- **FileInputStream(File file)**

通过打开一个到实际文件的连接来创建FileInputStream，该文件通过文件系统中的 File 对象指定。

- **FileInputStream(FileDescriptor fdObj)**

通过使用文件描述符 fdObj 创建FileInputStream。

- **FileInputStream(String name)**

通过文件系统中的路径名name打开文件，并创建文件输入流。

输出

FileOutputStram类是OutputStram的直接子类，该类主要负责对本地磁盘文件的顺序输出工作。该类继承了OutputStram的所有方法，并且实现了其中的write（）方法，提供了多个构造方法。

- FileOutputStream(File file)

  创建一个向指定File对象，向文件写入数据的文件输出流。

- FileOutputStream(File file, boolean append)

  创建一个向指定 File 对象，向文件写入数据的文件输出流。

输出

- **FileOutputStream(FileDescriptor fdObj)**

  创建一个向指定文件描述符处写入数据的输出文件流。

- **FileOutputStream(String name)**

  创建一个向具有指定名称的文件中写入数据的输出文件流。

- **FileOutputStream(String name, boolean append)**

  创建一个向具有指定 name 的文件中写入数据的输出文件流。

12.2.4 实例：输入信息保存到文件

```
// MyFileOutput.java
import java.io.*;
class MyFileOutput {
        public static void main(String args[]){
                FileInputStream fin;
                FileOutputStream fout;
                int ch;
                //声明一个整数变量用来读入用户输入 字符
                try{
                        //以标准输入设备为输入文件
                        fin=new FileInputStream(FileDescriptor.in);
                        //以D:\temp\aa.txt为输出文件
                        fout=new FileOutputStream("D:\\temp\\aa.txt");
                        System.out.println("Please input a line of
characters:");
```

```
                while((ch=fin.read()) !='\r')
                //反复读输入流，直到输入回车符为止
                    fout.write(ch);
                fin.close();          //关闭输入和输出流
                fout.close();
                System.out.println("Success!");
            }
            catch(FileNotFoundException e){
                System.out.println("can not create a file.");
            }
            catch(IOException e){
                System.out.println("error in input stream");
            }
        }
}
```

12.2.5 读取并显示文件

```
// TypeFile.java
import java.io.*;
class TypeFile{
        public static void main(String[] args){
                FileInputStream fin;
                FileOutputStream fout;
                int ch;     //声明一个整数变量用来读入用户输入字符
                try{
                        fin=new FileInputStream("D:\\temp\\Test.txt");
                        fout=new FileOutputStream(FileDescriptor.out);
                        while((ch=fin.read())!=-1)
                                fout.write(ch);
                        fin.close();
                        fout.close();
```

读取并显示文件代码示例

```
                }
                 catch(FileNotFoundException e){
                        System.out.println("can not create a file.");
                }
                catch(IOException e){
                        System.out.println("error in input stream");
                }
        }
}
```

12.2.6 文件复制

```
// CopyFile.java
import java.io.*;
class CopyFile{
        public static void main(String[] args){
                FileInputStream fin;
                FileOutputStream fout;
                int ch;
                if(args.length !=2){
                        System.out.println("参数格式不对，请输入源文件名，目标文件名： ");
                        return;
                }
                try{
                        fin=new FileInputStream(args[0]);
                        fout=new FileOutputStream(args[1]);
```

文件复制代码示例

```
                while((ch=fin.read()) !=-1    )
                 fout.write(ch);
                 fin.close();
                 fout.close();
                 System.out.println("File copy scuessed.");
            }
            catch(FileNotFoundException e){
                 System.out.println("can not create a file.");
            }
            catch(IOException e){
                 System.out.println("error in input stream");
            }
        }
}
```

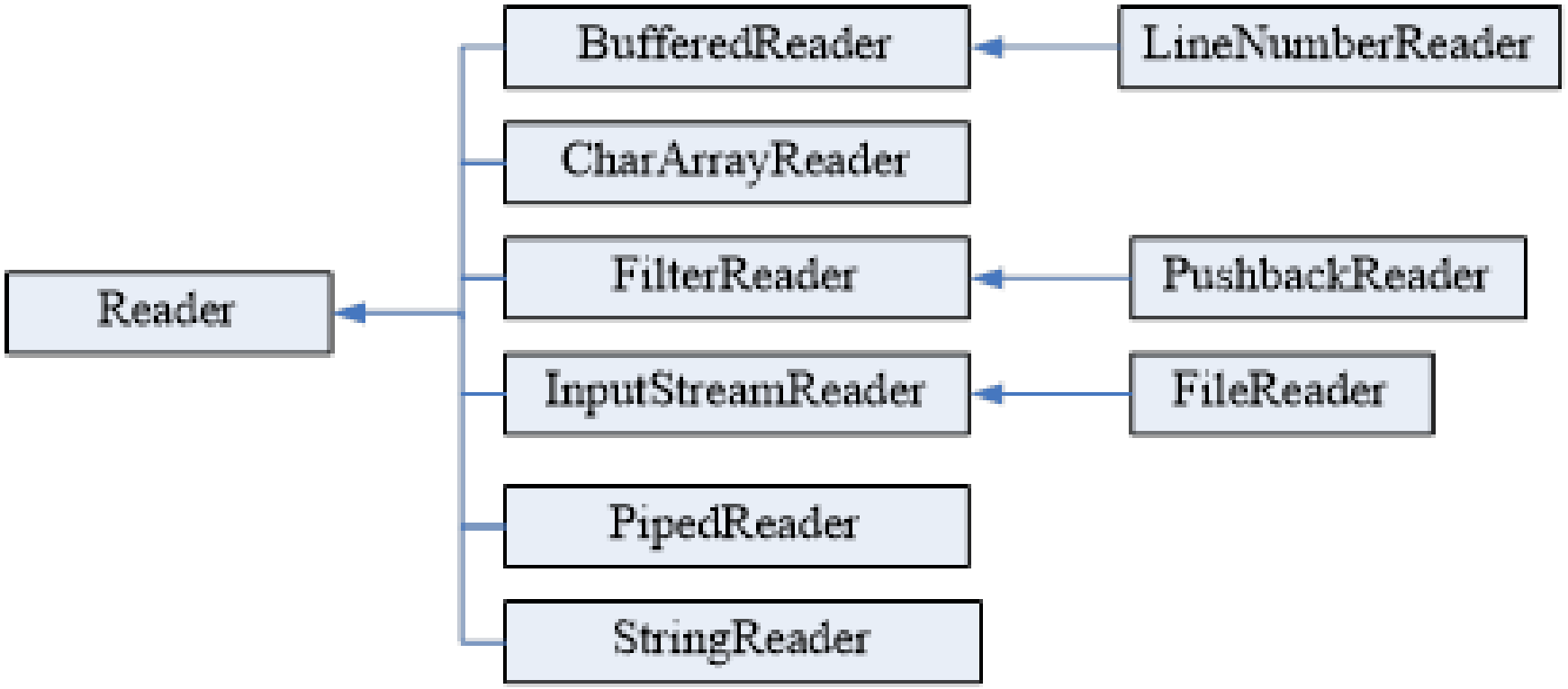
12.3 字符流

为了实现与其他程序语言及不同平台之间交互，Java.io包中加入了字符流处理类，它们是以Reader和Writer为基础派生的一系列类，提供了不同平台之间数据转换功能。

InputStreamReader 和OutputStreamWriter是从字节流到字符流转换的桥梁。前者从输入字节流中读字节，按照指定或者默认的字符集转换为字符。后者将字符数据转换成字节数据写到输出流。

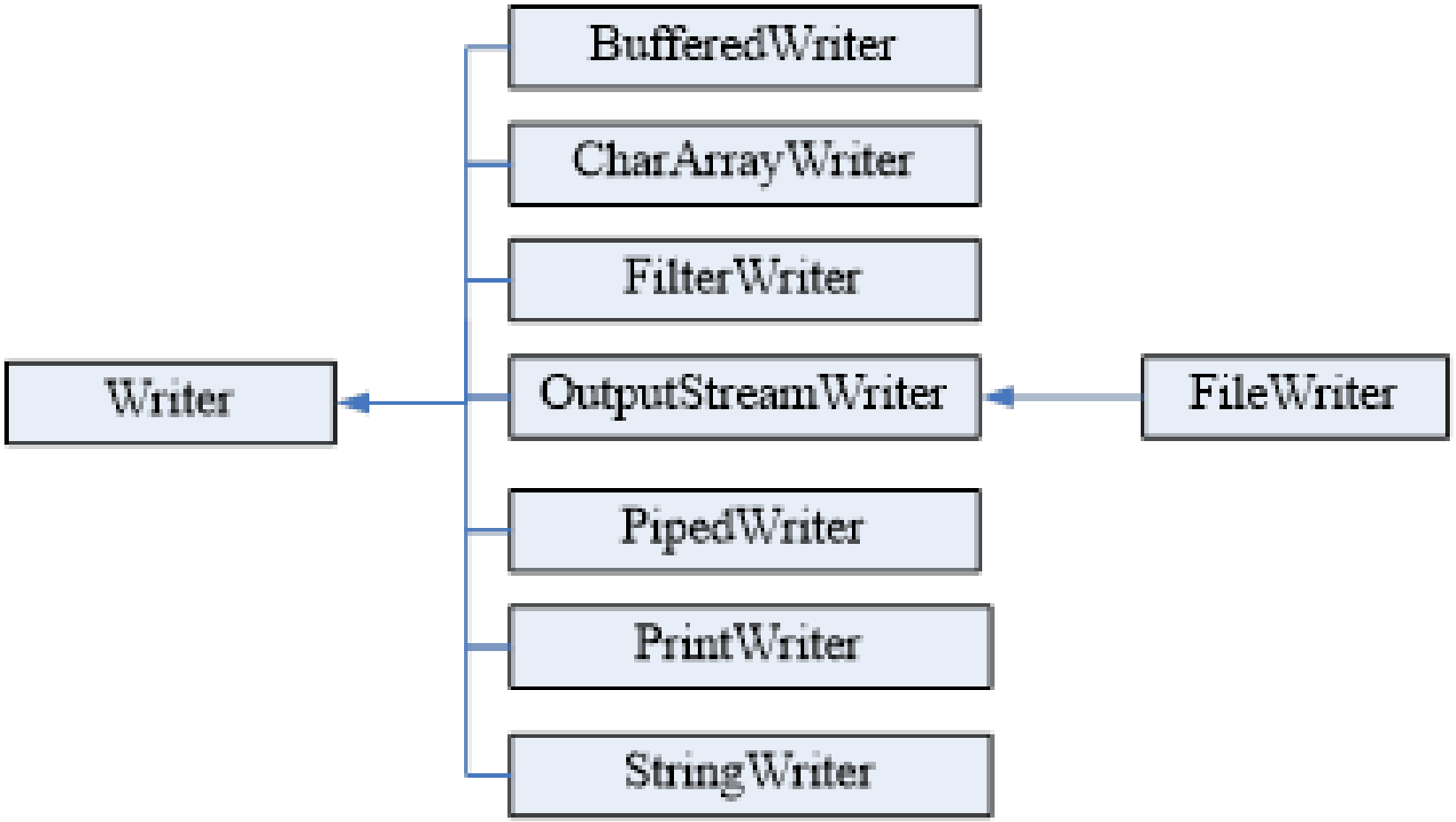
12.3 字符流

字符输入流类关系图



12.3 字符流

字符输出流类关系图



12.3.1 InputStreamReader和OutputStreamWriter

InputStreamReader类的方法主要有:

- void close()
  关闭流并释放与之关联的所有资源。
- String getEncoding()
  返回流使用的字符编码名称。
- int read()
  读取单个字符。
- int read(char[] cbuf, int offset, int length)
  读数组中的某部分。
- boolean ready()
  判断流是否准备就绪。

InputStreamReader和OutputStreamWriter

OutputStreamWriter提供的方法主要有:

- void close()
  关闭流。
- void flush()
  刷新流的缓冲。
- String getEncoding()
  返回流的字符编码名称。
- 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, int off, int len)
  写入字符串的某一部分。

12.3.2 字符流实例

```
import java.io.*;
public class FileToUnicode{
        public static void main(String args[]) {
                String s;
                InputStreamReader ir;
                BufferedReader in;
                try{
                        ir = new InputStreamReader(System.in);
                        in = new BufferedReader(ir);
                        while (!(s = in.readLine()).equals ("exit")){
                                System.out.println("Read: "+s);
                        }
                }
                catch(IOException e){
                         e.printStackTrace();
                }
        }
}
```

12.4 文件输入/输出

通过调用File类的各种方法，能够实现创建、删除、重命名文件等操作。

- 可以用list方法列出目录中的文件名。
- RandomAccessFile类支持对随机读写文件。

12.4.1 文件基本操作

```
import java.util.*;
import java.io.*;
public class FileTest {
        public static void main(String args[]) {
                try {
                        // 创建一个表示不存在子目录的File对象
                        File fp = new File("MyFile");
                        // 创建该目录
                        fp.mkdir();
                        // 创建描述MyFile目录下文件的File对象
                        File fc = new File(fp, "ChildFile.txt");
                        //测试文件是否存在，如果存在就删除
                        if(fc.exists())
                                fc.delete();
```

12.4.1 文件基本操作

```
else
        // 不存在时创建该文件
        fc.createNewFile();
// 创建输出流
FileWriter fo = new FileWriter(fc);
BufferedWriter bw = new BufferedWriter(fo);
PrintWriter pw = new PrintWriter(bw);
// 向文件中写入5行文本
for (int i = 0; i < 5; i++) {
        pw.println("[" + i + "]Hello World!!! 你好，本文件
由程序创建！ ！ ！ ");
}
// 关闭输出流
pw.close();
```

12.4.1 文件基本操作

```
// 打印提示信息
System.out.println("恭喜你，目录以及文件成功建立，数据成功写入！！！ ");
//获取文件名
System.out.println("File name: " + fc.getName());
//获取路径
System.out.println("File path: " + fc.getPath());
//获取绝对路径
System.out.println("Absolute path: " + fc.getAbsolutePath());
//获取上级路径
System.out.println("Parent: " + fc.getParent());
//文件是否可写
System.out.println("Can Write?  " + fc.canWrite());
```

12.4.1 文件基本操作

```
        //文件是否可读
        System.out.println("is readable?: " + fc.canRead());
        //文件长度
        System.out.println("File Size: " + fc.length());
        } catch (Exception e) {
                e.printStackTrace();
        }
}
}
```

12.4.2 文件随机读写

```
public class RandomAccessFileExample {
    public static void main(String args[])  {
        try{
            RandomAccessFile raf = new
RandomAccessFile("random.txt", "rw");
            raf.writeBoolean(true);    //将文件设置为可写
            raf.writeInt(168168);      //写入整数
            raf.writeChar('i');        //写入字符
            raf.writeDouble(168.168);  //写入小数
            raf.seek(1);
            System.out.println(raf.readInt());
            System.out.println(raf.readChar());
            System.out.println(raf.readDouble());
```

12.4.2 文件随机读写

```
                raf.seek(0);
                System.out.println(raf.readBoolean());
                raf.close();
        }
        catch (Exception e) {
                e.printStackTrace();
        }
    }
}
```

12.5 对象序列化

- 序列化将对象的状态信息转换为可以存储或传输的形式的过程。
- 在序列化期间，对象将其当前状态写入到临时或持久性存储区。
- 以后，可通过从存储区中读取或反序列化对象的状态，重新创建该对象。
- 很多情况下，对象内部状态是需要被持久化的，将运行中的对象状态保存下来，即使是在Java虚拟机退出的情况下，在需要的时候也可以还原。

12.5 对象序列化

- 对象序列化机制是Java内建的一种对象持久化方式，可以很容易实现在JVM中的活动对象与字节数组（流）之间进行转换，使得Java对象可以被存储，被网络传输，在网络的一端将对象序列化成字节流，经过网络传输到网络的另一端，可以从字节流重新还原为Java虚拟机中的运行状态对象。

12.5.1 存储对象

对于任何需要被序列化的对象，都必须要实现接口Serializable，它只是一个标识接口，本身没有任何成员，只是用来标识说明当前的实现类的对象可以被序列化。

```
public class Car{
        String brand;
        String color;
        int  no;
        … …
}
```

12.5.2 Car对象序列化实例

```
import java.io.Serializable;
public class Car implements Serializable{
      String brand;
      String color;
      int  no;
     Car(String brandIn,String colorIn,int noIn){
          brand=brandIn;
          color=colorIn;
          no=noIn;
     }
     public  String toString (){
            return new String("brand="+brand + ", color=" +
color + ", no=" +no);
     }
}
```

12.5.3 存储和读取序列化对象信息

```
import java.io.*;
import java.util.*;
public class SeriSample{
    public static void main(String args[]){
        Car car1=new Car("Audo","red", 352);
        try {
            FileOutputStream fos =
                new FileOutputStream("d://test.txt");
            ObjectOutputStream os =
                new ObjectOutputStream(fos);
            os.writeObject(car1);
            os.close();
         } catch (Exception e) {
            System.out.println(e);
         }
```

12.5.3 存储和读取序列化对象信息

```
FileInputStream fis = null;
car1 = null;
try {
    try {
        fis = new FileInputStream("d://test.txt");
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
    ObjectInputStream os = new
ObjectInputStream(fis);
    // 强制类型转换
    car1 = (Car) os.readObject();
    os.close();
} catch (ClassNotFoundException e) {
    System.out.println(e);
```

12.5.3 存储和读取序列化对象信息

```
            } catch (IOException e) {

                        e.printStackTrace();
            }
            System.out.println("brand:\t" + car1.brand);
            System.out.println("color:\t" + car1.color);
            System.out.println("no:\t" + car1.no);
            System.out.println("Car.toString(): " + car1);
        }
}
```

12.6 正则表达式

◆ 在程序开发中会遇到需要匹配、查找、替换、判断字符串的情况发生，而这些情况有时又比较复杂，如果用纯编码方式解决，往往会浪费程序员的时间及经历。正则表达式成为解决这一矛盾的主要手段。

◆ 正则表达式是一种可以用于模式匹配和替换的规范，一个正则表达式就是由普通的字符以及特殊字符组成的文字模式，用于描述在查找文字主题时待匹配的一个或多个字符串。

◆ 正则表达式定义了字符串的模式，可以用来搜索、编辑或处理文本。

12.6 正则表达式

java.util.regex 包主要包括三个类：

- Pattern 类
- Matcher 类
- PatternSyntaxException类。

Pattern类

Pattern对象是一个正则表达式的表一表示。

Pattern类没有公共构造方法。要创建一个Pattern对象，必须首先调用其公共静态编译方法，该方法返回一个Pattern对象。

Pattern类的主要方法

Pattern的主要方法如下:

- static Pattern compile(String regex)
  将给定的正则表达式编译并赋予给Pattern类
- static Pattern compile(String regex, int flags)
  同上，但增加flag参数的指定，可选的flag参数。
- int flags()
  返回当前Pattern的匹配flag参数。
- Matcher matcher(CharSequence input)
  生成一个给定命名的Matcher对象。

Pattern类的主要方法

- static boolean matches(String regex, CharSequence input)

编译给定的正则表达式并且对输入的字串以该正则表达式为模开展匹配,该方法适合于该正则表达式只会使用一次的情况，也就是只进行一次匹配工作，因为这种情况下并不需要生成一个Matcher实例。

- String pattern()

返回该Patter对象所编译的正则表达式。

Pattern类的主要方法

- String[] split(CharSequence input)

  将目标字符串按照Pattern里所包含的正则表达式为模进行分割。

- String[] split(CharSequence input, int limit)

  作用同上，增加参数limit目的在于要指定分割的段数，如将limit设为2，那么目标字符串将根据正则表达式分为割为两段。

Matcher类

- Matcher对象是对输入字符串进行解释和匹配操作的引擎。
- Matcher没有公共构造方法。
- 需要调用Pattern对象的matcher方法获得一个Matcher对象。

Matcher类的主要方法

- boolean find()

  尝试在目标字符串里查找下一个匹配子串。

- boolean find(int start)

  重设Matcher对象，并且尝试在目标字符串里从指定的位置开始查找下一个匹配的子串。

- boolean lookingAt()

  检测目标字符串是否以匹配的子串起始。

- boolean matches()

  尝试对整个目标字符展开匹配检测，也就是只有整个目标字符串完全匹配时才返回真值。

Matcher类的主要方法

- Pattern pattern()

返回Matcher对象的现有匹配模式，也就是对应的Pattern对象。

- String replaceAll(String replacement)

将目标字符串里与既有模式相匹配的子串全部替换为指定的字符串。

- PatterSyntaxException

是一个非强制异常类，它指示一个正则表达式模式中的语法错误。

例12.9 使用正则表达式匹配字符串的开头和结束

```
import  java.util.regex.Matcher;
import  java.util.regex.Pattern;
public class TestRegularExpression_01{
    public static void main(String args[]){
        //查找以Java开头，任意结尾的字符串
        Pattern pattern = Pattern.compile("^Java.*");
        Matcher matcher = pattern.matcher("Java不是人");
        boolean b = matcher.matches();
        System.out.println(b);
    }
}
```

例12.10 使用正则表达式进行多条件字符串分割

```
import  java.util.regex.Pattern;
public class TestRegularExpression_02{
    public static void main(String args[]){
        Pattern pattern = Pattern.compile(“[, |]+”);
        String[] strs = pattern.split(“Java Hello World  Java,
Hello,, world | Sun”);
        for(int i=0; i<strs.length; i++){
            System.out.println(strs[i]);
        }
    }
}
```