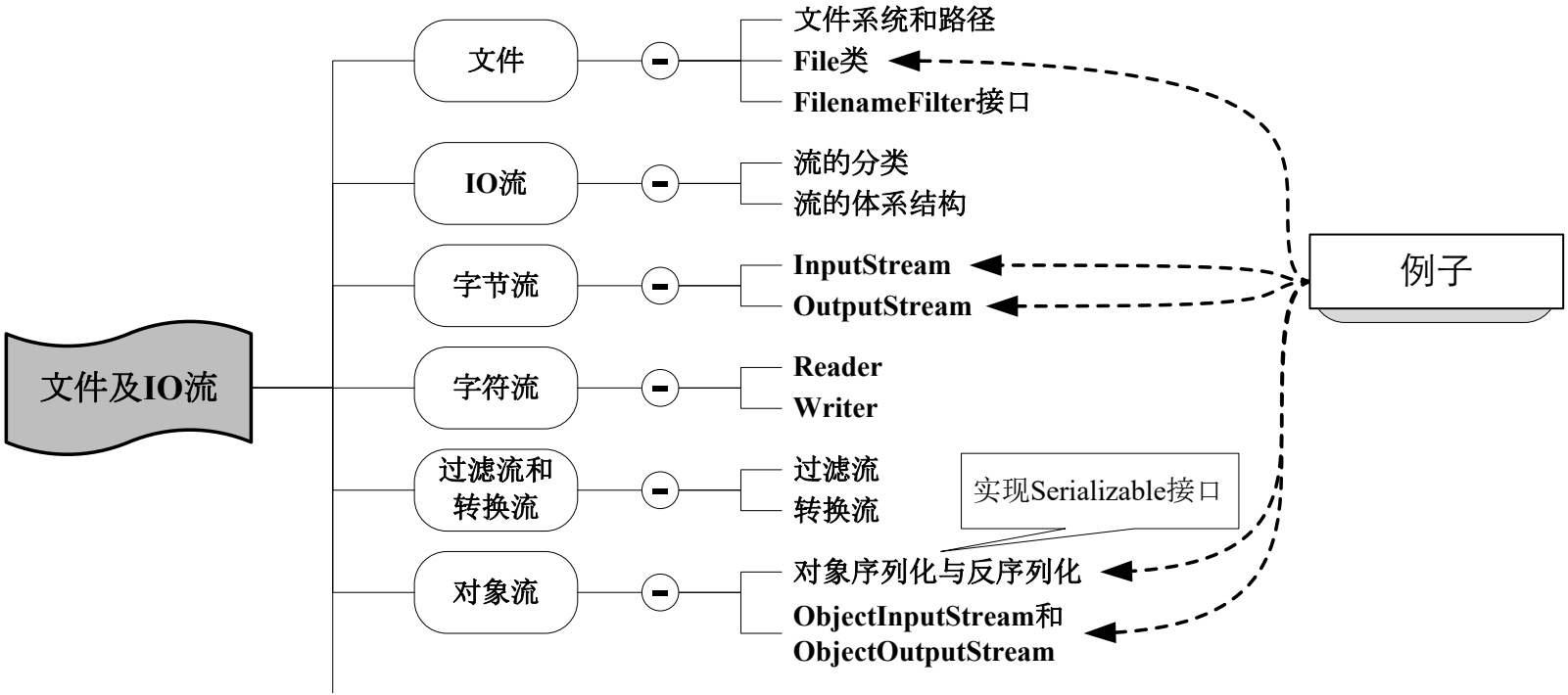
文件及IO流

掌握File类的使用

掌握IO流的分类和体系结构

掌握字符流和字节流的使用


文件及IO流
文件
文件系统和路径
File类
FilenameFilter接口
IO流
流的分类
流的体系结构
字节流
InputStream
OutputStream
字符流
Reader
Writer
过滤流和
转换流
过滤流
转换流
对象流
对象序列化与反序列化
ObjectInputStream和
ObjectOutputStream
例子
实现Serializable接口

1.1.1 文件系统和路径

- 一个文件系统可以包含三类对象：文件、目录和符号链接
- 文件系统中的对象可以使用一条路径作为唯一的识别
- 路径有绝对路径和相对路径两种：
  - 绝对路径：从根路径开始，如“D:\data\test.txt”
  - 相对路径：以当前目录为参照，如 “data\test.txt”

1.1.2 File类

- java.io包中提供了一系列用于文件处理的接口和类
- File类代表与平台无关的文件和目录

分类	方法	功能描述
访问文件名或路径	String getName()	返回File对象所表示的文件名或目录的路径
	String getPath()	返回File对象所对应的路径名
	File getAbsoluteFile()	返回File对象的绝对路径文件
	String getAbsolutePath()	返回File对象所对应的绝对路径名
	String getParent()	返回此File对象所对应目录的父目录
	boolean renameTo(File dest)	重命名File对象对应的文件或目录

分类	方法	功能描述
文件检测	boolean exists()	判断File对象所对应的文件或目录是否存在
	boolean canWrite()	判断File对象所对应的文件或目录是否可写
	boolean canRead()	判断File对象所对应的文件或目录是否可读
	boolean isDirectory()	判断File对象是否为一个目录
	boolean isFile()	判断File对象是否为一个文件
	boolean isAbsolute()	判断File对象是否采用绝对路径
文件信息	long length()	返回File对象所对应文件的长度（以字节为单位）
	long lastModified()	返回File对象的最后一次被修改的时间
文件操作	boolean createNewFile()	检查文件是否存在，当File对象所对应的文件不存在时新建一个文件
	boolean delete()	删除File对象所对应的文件或目录
目录操作	boolean mkdir()	创建一个File对象所对应的路径
	String[] list()	列出File对象所有的子文件名和路径名
	File[] listFile()	列出File对象所有的子文件和路径
	static File[] listRoots()	列出系统所有的根路径

```java
import java.io.File;
import java.io.IOException;

public class FileDemo {

    public static void main(String[] args) {
        // 以当前路径来创建一个File对象,"."代表当前路径
        File file = new File(".");
        // 直接获取文件名，输出"."
        System.out.println(file.getName());
        // 获取相对路径的父路径可能出错，下面代码输出null
        System.out.println(file.getParent());
        // 获取绝对路径
        System.out.println(file.getAbsoluteFile());
        // 获取上一级路径
        System.out.println(file.getAbsoluteFile().getParent());
        // 以指定的文件名创建File对象
        File newFile = new File("test.txt");
        System.out.println("newFile对象是否存在：" + newFile.exists());
        try {
            // 以指定newFile对象来创建一个文件
            newFile.createNewFile();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        // 以newFile对象来创建一个目录，因为newFile已经存在
        // 所以下面方法返回false，即无法创建该目录
        System.out.println("创建目录："+newFile.mkdir());
        // 使用list()方法来列出当前路径下的所有文件和路径
        String[] fileList = file.list();
        System.out.println("====当前路径下所有文件和路径如下====");
        for (String fileName : fileList) {
            System.out.println(fileName);
        }
        // listRoots()静态方法列出所有的磁盘根路径。
        File[] roots = File.listRoots();
        System.out.println("====系统所有根路径如下====");
        for (File root : roots) {
            System.out.println(root);
        }

    }

}
```

1.1.3 FilenameFilter接口

- FilenameFilter是一个文件过滤器接口，可以对文件进行过滤，将符合条件的文件筛选出来
- File类的list()方法可以接受FileNameFilter类型的参数

方法	功能描述
String[] list(FilenameFilter filter)	返回File对象所对应目录中满足指定过滤条件的文件名和子目录名
File[] listFiles(FilenameFilter filter)	返回File对象所对应目录中满足指定过滤条件的文件和子目录

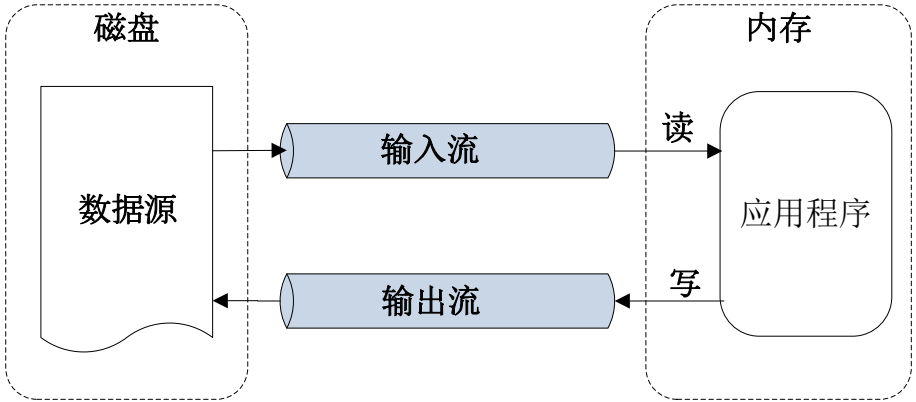
1.2 IO流

- IO流是实现数据输入（Input）和输出（Output）的基础
- 流（Stream）的优势在于使用统一的方式对数据进行操作或传递，简化了代码操作

1.2.1 流的分类

按照流的流向来分：

- 输入流：只能从输入流中读取数据，而不能向输入流中写入数据
- 输出流：只能向输出流中写入数据，而不能从输出流中读取数据



字节流和字符流

按照流所操作的基本数据单元来分：

- 字节流：所操作的基本数据单元是8位的字节（byte)
- 字符流：所操作的基本数据单元是16位的字符（Unicode）

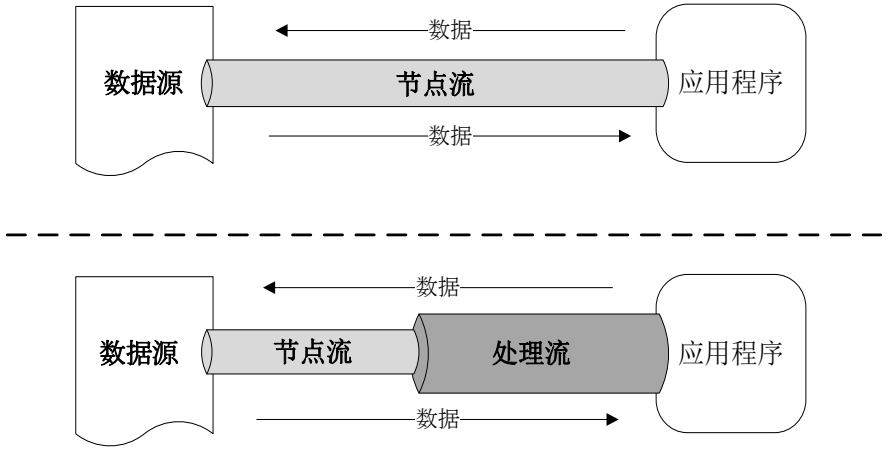
节点流和处理流

按照流的角色来分：

- 节点流：用于从/向一个特定的IO设备（如磁盘、网络）读/写数据的流
- 处理流：对一个已经存在的流进行连接或封装，通过封装后的流来实现数据的读/写功能

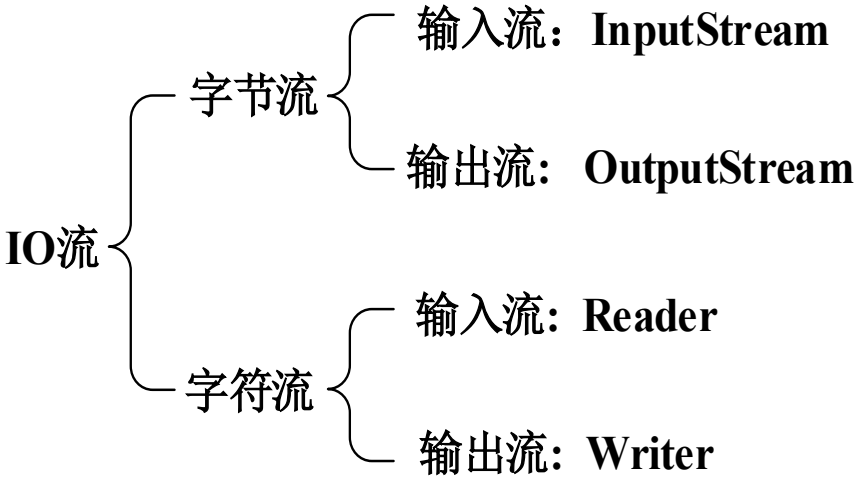
节点流和处理流

- 使用处理流进行输入/输出时，程序不会直接连接到实际的数据源，而是对节点流进行包装
- 使用处理流来包装不同的节点流，消除了不同节点流实现的差异，提供了更便利的方法来完成输入/输出功能

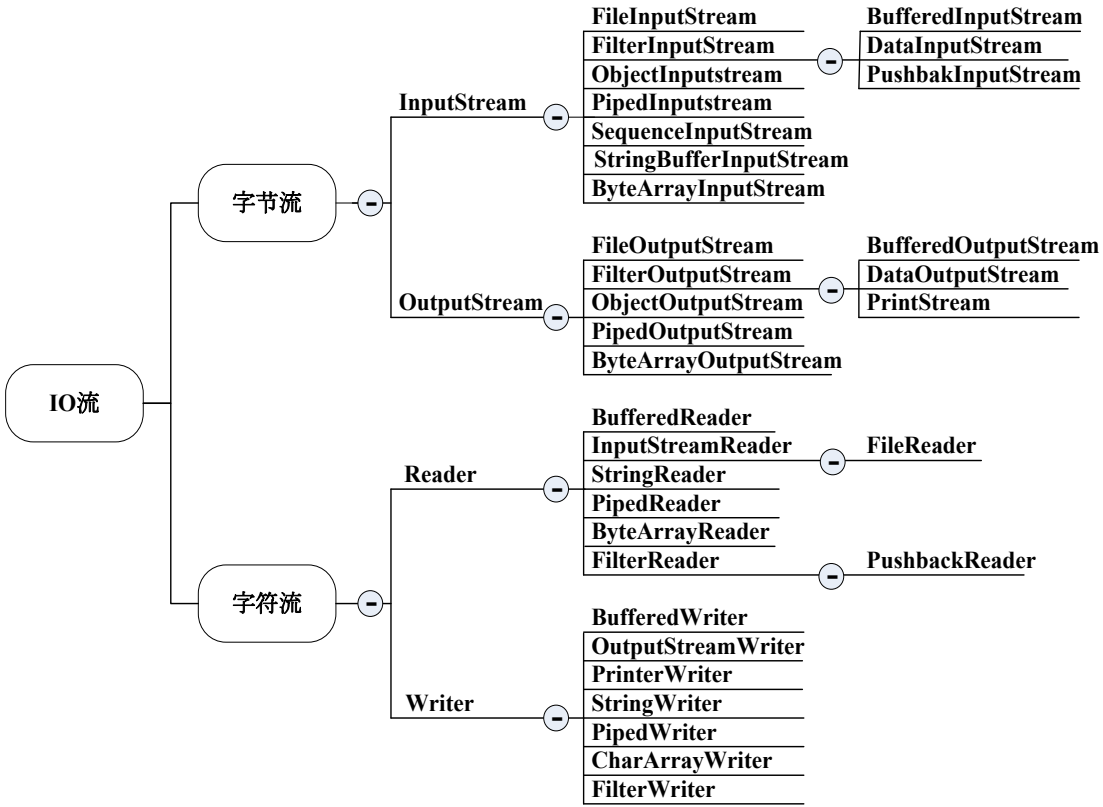


1.2.2 流的体系结构

- Java的IO流都是由4个抽象基类派生



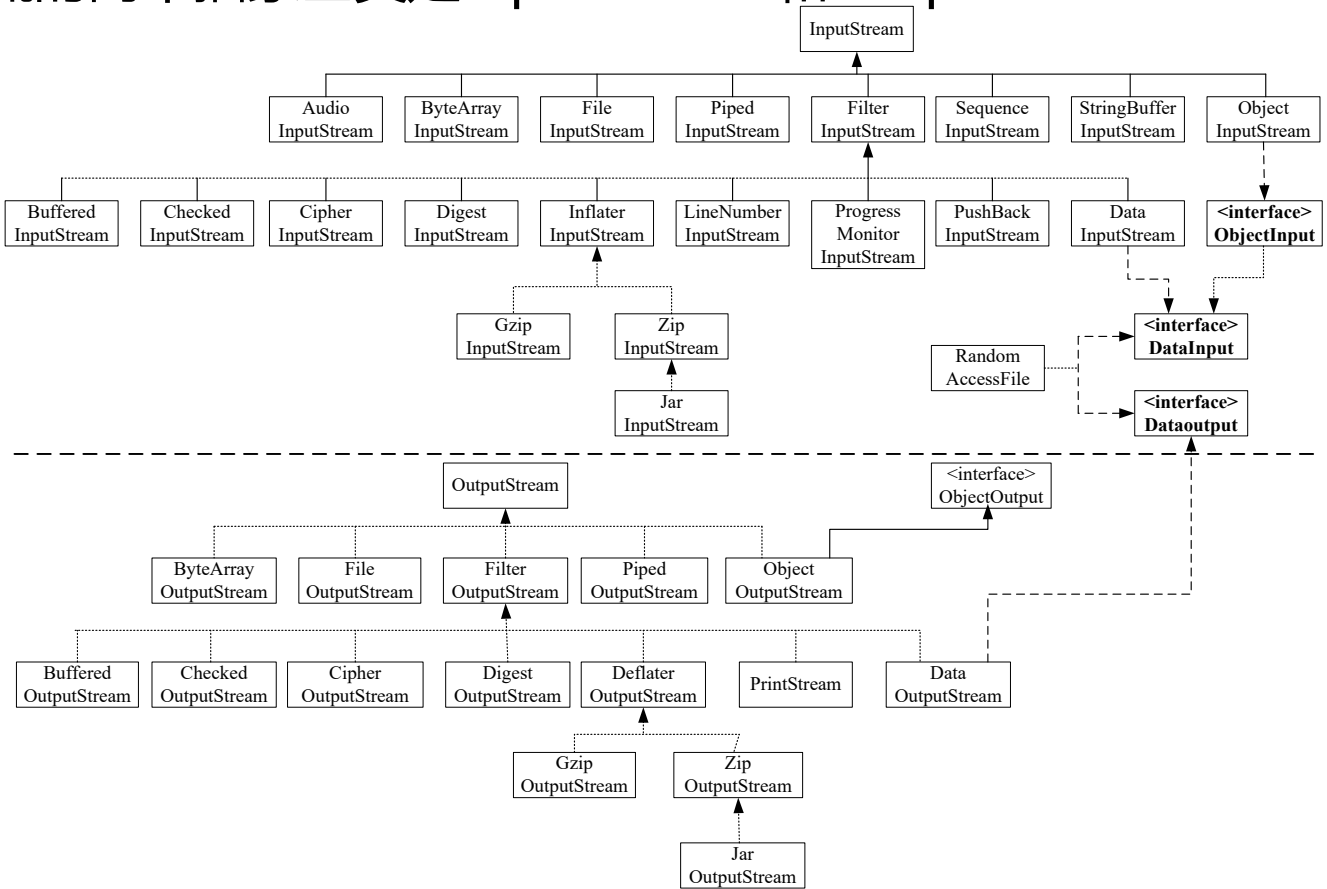
- Java的IO流体系共涉及40多个类：



- 在使用IO流时注意一个规则：
  - 如果进行输入/输出的内容是文本内容，则使用字符流
  - 如果进行输入/输出的内容是二进制内容，则使用字节流

1.3 字节流

- 字节流的两个抽象基类是InputStream和OutputStream

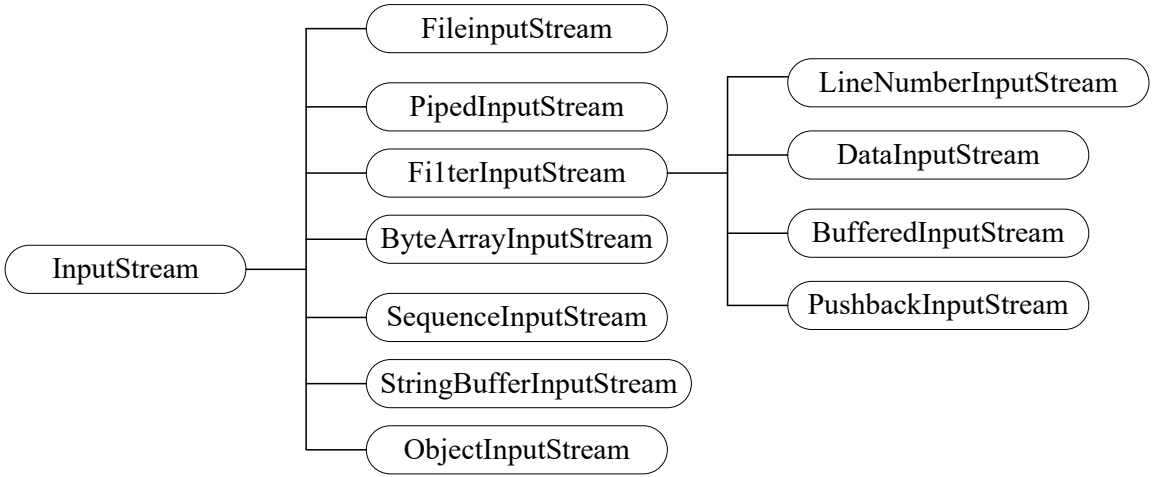


1.3.1 InputStream

- InputStream是字节输入流

方法	功能描述
abstract int read()	读取一个字节并返回，如果遇到源的末尾，则返回-1
int read(byte[] b)	将数据读入到字节数组中，并返回实际读取的字节数；当已经到达流的末尾而没有可用的字节时返回-1
int read(byte[] b, int offset, int len)	将数据读入到字节数组中，offset表示在数组中存放数据的开始位置， len表示所读取的最大字节数；当已经到达流的末尾而没有可用的字节时返回-1
int available()	用于返回在不发生阻塞的情况下，从输入流中可以读取的字节数
void close()	关闭此输入流，并释放与该流关联的所有系统资源

InputStream类是抽象类，不能直接实例化，因此需使用其子类来完成具体功能



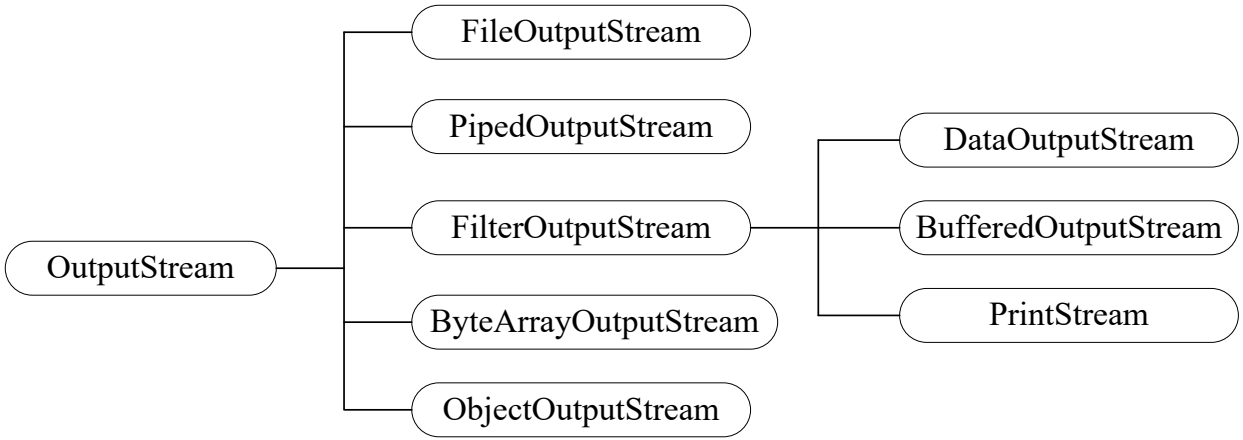
```java
import java.io.FileInputStream;
import java.io.IOException;

public class FileInputStreamDemo {

    public static void main(String[] args) {

        // 声明文件字节输入流
        FileInputStream fis = null;
        try {
            // 实例化文件字节输入流
            fis = new FileInputStream(
                    "src\\com\\qst\\chapter01\\FileInputStreamDemo.java");
            // 创建一个长度为1024的字节数组作为缓冲区
            byte[] bbuf = new byte[1024];
            // 用于保存实际读取的字节数
            int hasRead = 0;
            // 使用循环重复读文件中的数据
            while ((hasRead = fis.read(bbuf)) > 0) {
                // 将缓冲区中的数据转换成字符串输出
                System.out.print(new String(bbuf, 0, hasRead));
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                // 关闭文件输入流
                fis.close();
            } catch (IOException e) {

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

1.3.2 OutputStream

OutputStream是字节输出流，使用OutputStream可以往数据源以字节为单位写入数据。

void write(int c)	将一个字节写入到文件输出流中
void write(byte[] b)	将字节数组中的数据写入到文件输出流中
void write(byte[] b, int offset, int len)	将字节数组中的offset开始的len个字节写到文件输出流中
void close()	关闭此输入流，并释放与该流关联的所有系统资源
void flush()	将缓冲区中的字节立即发送到流中，同时清空缓冲

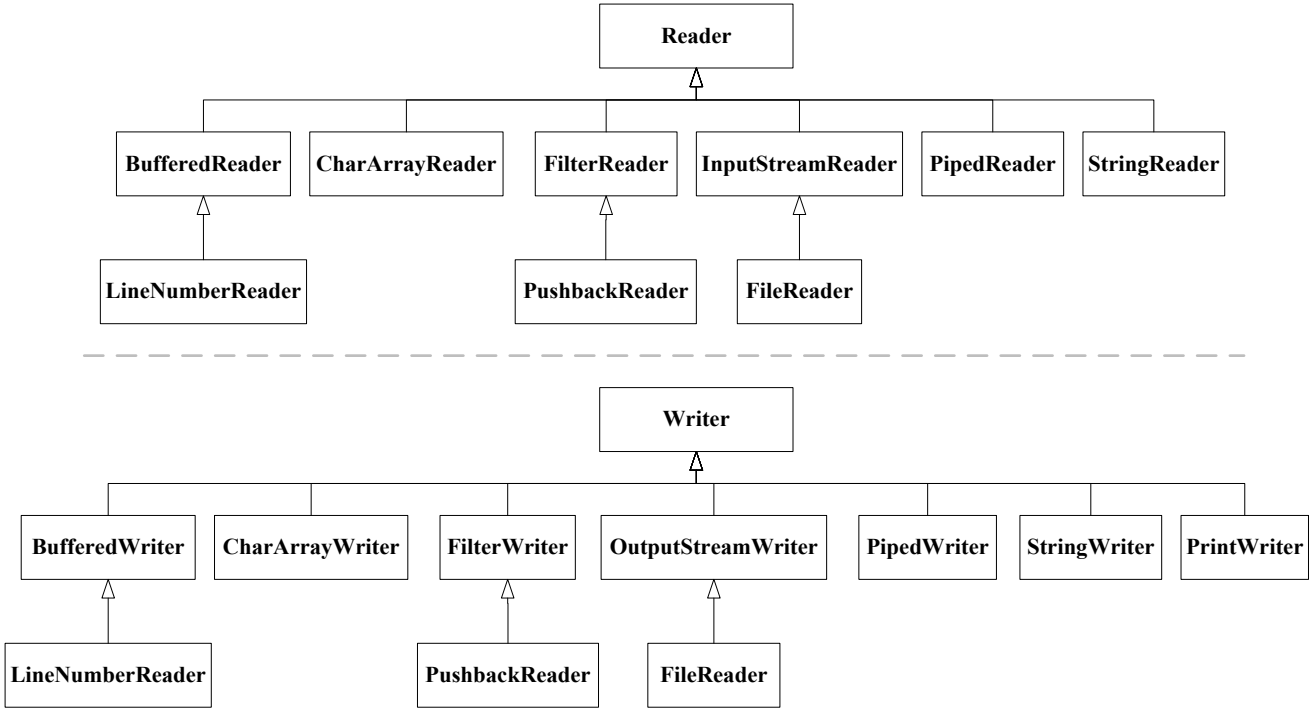
OutputStream与InputStream一样都是抽象类，不能直接实例化，因此都是使用其子类来完成具体功能。



```java
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Scanner;

public class FileOutputStreamDemo {

    public static void main(String[] args) {

        // 建立一个从键盘接收数据的扫描器
        Scanner scanner = new Scanner(System.in);

        // 声明文件字节输出流
        FileOutputStream fos = null;
        try {
            // 实例化文件字节输出流
            fos = new FileOutputStream("D:\\mytest.txt");
            System.out.println("请输入内容: ");
            String str = scanner.nextLine();
            // 将数据写入文件中
            fos.write(str.getBytes());
            System.out.println("已保存! ");
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                // 关闭文件输出流
                fos.close();
                scanner.close();
            } catch (IOException e) {

                e.printStackTrace();
            }
        }

    }

}
```

1.4 字符流

字符流所处理的数据基本单元是字符，其输入/输出操作都是在字符的基础上进行。

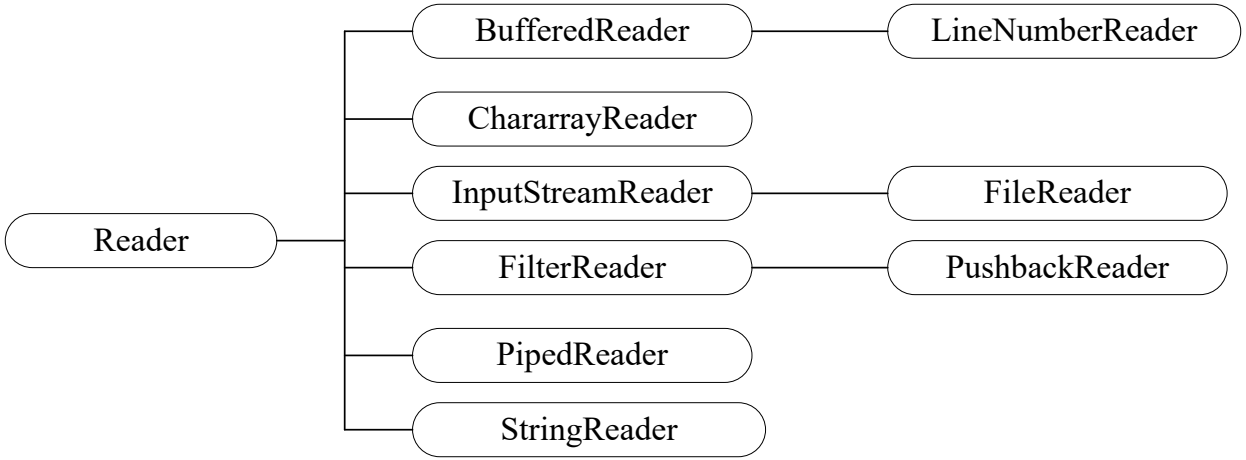


1.4.1 Reader

Reader是字符输入流，用于从数据源以字符为单位进行读取数据。

方法	功能说明
int read()	用于从流中读出一个字符并返回
int read(char[] buffer)	将数据读入到字符数组中，并返回实际读取的字符数；如果已到达流的末尾而没有可用的字节时，则返回-1
int read(char[] buffer, int offset, int len)	将数据读入到一个字符数组，放到数组offset指定的位置开始，并用len来指定读取的最大字符数；当到达流的末尾时，则返回-1
void close()	关闭Reader流，并释放与该流关联的所有系统资源

Reader是抽象类，不能直接实例化，因此使用其子类来完成具体功能。



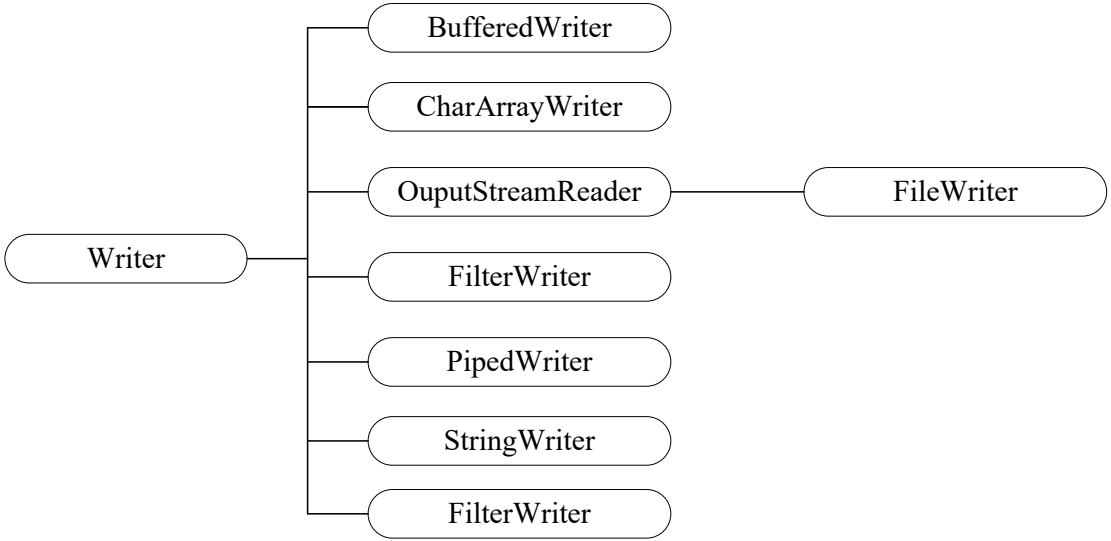
```java
import java.io.BufferedReader;
import java.io.FileReader;

public class ReaderDemo {

    public static void main(String[] args) {
        // 声明一个BufferedReader流的对象
        BufferedReader br = null;
        try {
            // 实例化BufferedReader流，连接FileReader流用于读文件
            br = new BufferedReader(new FileReader(
                    "src\\com\\qst\\chapter01\\ReaderDemo.java"));
            String result = null;
            //循环读文件，一次读一行
            while ((result = br.readLine()) != null) {
                //输出
                System.out.println(result);
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                // 关闭缓冲流
                br.close();
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }

    }

}
```

1.4.2 Writer

Writer是字符输出流，用于往数据源以字符为单位进行写数据。

方法	功能说明
void write(int c)	写入单个字符
void write(char[] buffer)	写入字符数组
void write(char[] buffer, int offset, int len)	写入字符数组的某一部分，从offset开始的len个字符
void write(String str)	写入字符串

Writer是抽象类，不能直接实例化，需要使用其子类来完成具体功能。



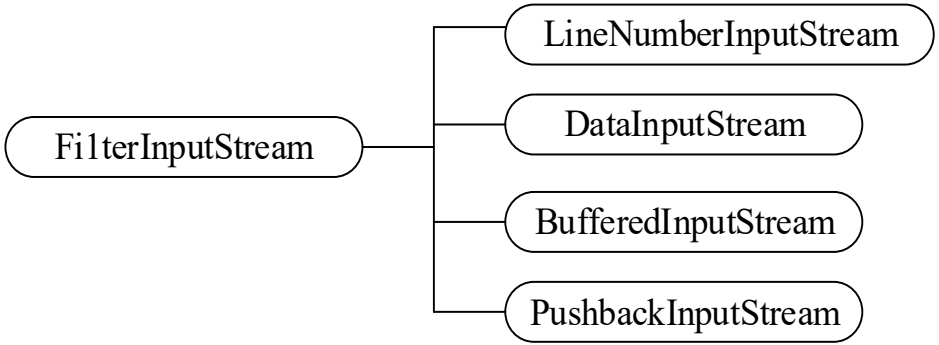
```java
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

public class WriterDemo {

    public static void main(String[] args) {

        // 建立一个从键盘接收数据的扫描器
        Scanner scanner = new Scanner(System.in);

        // 声明文件字符输出流
        FileWriter fw = null;
        try {
            // 实例化文件字符输出流
            fw = new FileWriter("D:\\mytest2.txt");
            System.out.println("请输入内容：");
            String str = scanner.nextLine();
            // 将数据写入文件中
            fw.write(str);
            System.out.println("已保存！");
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                // 关闭文件字符输出流
                fw.close();
                scanner.close();
            } catch (IOException e) {

                e.printStackTrace();
            }
        }

    }

}
```

1.5.1 过滤流

- 过滤流用于对一个已有的流进行连接和封装处理。
- 过滤流又分为:
  - 过滤输入流
  - 过滤输出流

FilterInputStream

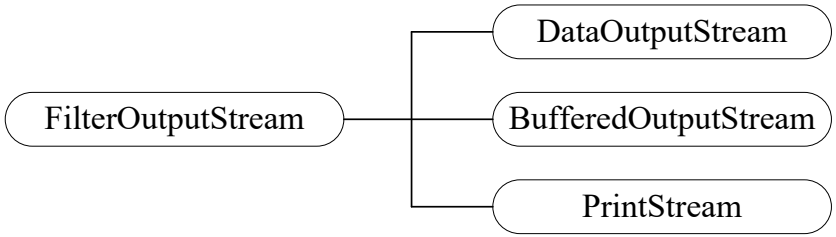
FilterInputStream为过滤输入流



```java
import java.io.BufferedInputStream;
import java.io.FileInputStream;

public class BufferedInputStreamDemo {

    public static void main(String[] args) {
        // 定义一个BufferedInputStream类型的变量
        BufferedInputStream bi = null;
        try {
            // 利用FileInputStream对象创建一个输入缓冲流
            bi = new BufferedInputStream(new FileInputStream("src\\com\\qst\\chapter01\\BufferedInputStreamDemo.java"));
            int result = 0;
            //循环读数据
            while ((result = bi.read()) != -1) {
                //输出
                System.out.print((char) result);
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                // 关闭缓冲流
                bi.close();
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }

    }

}
```

FilterOutputStream

FilterOutputStream为过滤输出流。




```
public class PrintStreamDemo {
  public static void main(String[] args) {
    try (PrintStream ps = new PrintStream(new
      FileOutputStream("D:\\test.txt"))) {
      // 使用PrintStream打印一个字符串
      ps.println("这是PrintStream打印流往文件中写数据！");
    } catch (IOException ioe) {
      ioe.printStackTrace();
    }
  }
}
```

1.5.2 转换流

Java的IO流体系中提供了两个转换流：

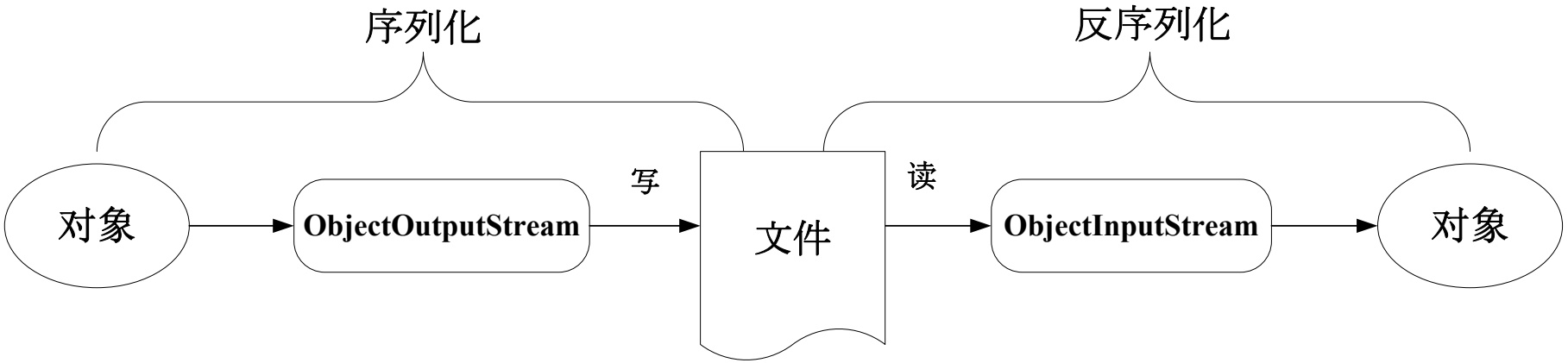
InputStreamReader：将字节输入流转换成字符输入流

OutputStreamWriter：将字符输出流转换成字节输出流

1.6.1 对象序列化与反序列化

对象的序列化（Serialize）：将对象数据写到一个输出流中的过程；

对象的反序列化：从一个输入流中读取一个对象。



- 一个类的对象是可序列化的，则该类必须实现java.lang包下的Serializable接口或Externalizable接口
- Java中的Serializable接口只是一个标志接口，接口中没有任何的方法，只是用于表明该类的实例对象是可以序列化的

1.6.2 ObjectOutputStream

- ObjectOutputStream是OutputStream的子类，该类也实现了ObjectOutput接口，其中ObjectOutput接口支持对象序列化。
- 该类的一个构造方法如下：

```
ObjectOutputStream(OutputStream outStream) throws IOException
```

ObjectOutputStream类的常用方法及功能

方法	功能描述
final void writeObject(Object obj)	写入一个obj对象到调用的流中
void writeInt(int i)	写入一个32位int值到调用的流中
void writeBytes(String str)	以字节序列形式将字符串str写入到调用的流中
void writeChar(int c)	写入一个16位的char值到调用的流中

1.6.3 ObjectInputStream

ObjectInputStream是InputStream的子类，该类也实现了ObjectInput接口，其中ObjectInput接口支持对象序列化。

ObjectInputStream类的一个构造方法如下：

```
ObjectInputStream(InputStream inputStream) throws IOException
```

ObjectInputStream类的常用方法及功能

方法	功能描述
final Object readObject()	从流中读取对象
int readInt()	从流中读取一个整型值
String readUTF()	从流中读取UTF-8格式的字符串
char readChar()	读取一个16位的字符

```java
import java.io.FileInputStream;
import java.io.ObjectInputStream;

//反序列化
public class ObjectInputStreamDemo {

    public static void main(String[] args) {

        // 创建一个ObjectInputStream对象输入流
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream("d:\\PersonObject.txt"))) {
            // 从ObjectInputStream对象输入流中读取一个对象，并强制转换成Person对象
            Person person =(Person)ois.readObject();
            System.out.println("序列化完毕！读出的对象信息如下：");
            System.out.println(person);
        } catch (Exception ex) {
            ex.printStackTrace();
        }

    }

}
```

总结

- 按照流的流向来分，可以将流分为输入流和输出流
- 按照流所操作的基本数据单元来分，可以将流分为字节流和字符流
- 按照流的角色来分，可以将流分为节点流和处理流
- Java的IO流都是由InputStream、OutputStream、Reader和Writer这4个抽象基类派生的
- 过滤流用于对一个已有的流进行连接和封装处理
- 转换流InputStreamReader能够将字节输入流转换成字符输入流
- 转换流OutputStreamWriter能够将字符输出流转换成字节输出流
- 对象的序列化（Serialize）是将对象数据写到一个输出流中的过程；而对象的反序列化是从一个输入流中读取一个对象