



Files and Streams

Java Advanced Tutorial , [Module 1](#)

Summary

Class File

Streams

Access Files

Object Serialization

New Java IO

Introduction

Part 1

Class File

Streams

Access Files

Object Serialization

New Java IO

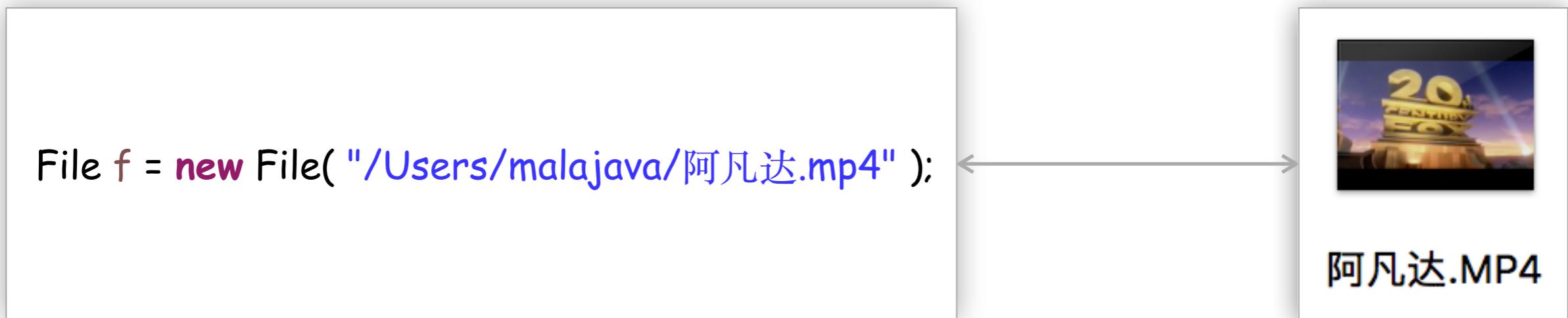
Introduction to File

File Systems

Class File

在JVM中创建的File对象

本地文件系统中的文件



`File`是文件或目录的路径名的抽象表示形式

虚拟机中的一个`File`实例用来表示一个文件或一个目录

`File`类提供可操作文件或目录的方法，但并不支持访问文件的内容

Class File

构造方法	描述
public File (String pathname)	通过将给定路径名字符串转换为抽象路径名来创建一个新 File 实例
public File (String parent, String child)	根据 parent 路径名字符串和 child 路径名字符串创建一个新 File 实例
public File (File parent, String child)	根据 parent 抽象路径名和 child 路径名字符串创建一个新 File 实例
public File (URI uri)	通过将给定的 file: URI 转换为一个抽象路径名来创建一个新的 File 实例

File 类提供了 4 个 public 修饰的构造方法用于创建 File 实例

Class File

常量	描述
public static final char separatorChar	与系统有关的默认名称分隔符(文件系统中的不同层次的路径之间的分隔符)
public static final String separator	与系统有关的默认名称分隔符，为了方便，它被表示为一个字符串
public static final char pathSeparatorChar	与系统有关的路径分隔符(环境变量PATH的取值中不同部分之间的分隔符)
public static final String pathSeparator	与系统有关的路径分隔符，为了方便，它被表示为一个字符串

File 类提供了 4 个表示分隔符的常量

Windows 平台下，pathSeparator 和 pathSeparatorChar 对应的是 :

Windows 平台下，separator 和 separatorChar 对应的是 \

Unix / Linux / OS X 平台下，pathSeparator 和 pathSeparatorChar 对应的是 :

Unix / Linux / OS X 平台下，separator 和 separatorChar 对应的是 /

File Operations

方法	描述
public boolean exists()	测试当前File实例所表示的文件或目录是否存在(存在则返回true)
public boolean isFile()	测试当前File实例所表示的是不是一个标准文件(是则返回true)
public boolean isDirectory()	测试当前File实例所表示的是不是一个目录(是则返回true)
public boolean isHidden()	测试当前File实例所表示的文件或目录是否是隐藏的(是则返回true)
public boolean isAbsolute()	测试当前File实例对应的路径名称是否是绝对路径名(是则返回true)
public long length()	如果当前File实例表示文件(用isFile判断), 则返回该文件的长度
public String[] list()	返回当前File实例所表示的目录内所有目录名称及文件名称对应的数组
public File[] listFiles()	返回当前File实例所表示的目录内所有目录和文件对应的File数组

对任意File实例都应该测试是否存在、测试是否是文件、测试是否是目录

File Operations

方法	描述
<code>public long lastModified()</code>	返回当前File实例所表示的文件或目录最后一次被修改的时间(以毫秒计)
<code>public boolean setLastModified(long time)</code>	设置当前File实例所表示的文件或目录的最后一次修改时间
<code>public boolean canRead()</code>	判断应用程序是否可以 读取 当前File实例所表示的文件或目录
<code>public boolean setReadable(boolean readable)</code>	设置当前File实例所表示的文件或目录的拥有者的读权限(参数 true 表示可读)
<code>public boolean setReadable(boolean readable, boolean ownerOnly)</code>	设置当前File实例所表示的文件或目录的拥有者或所有用户的读权限
<code>public boolean setReadOnly()</code>	标记当前File实例所表示的文件或目录为只读(只读, 不允许修改和执行)
<code>public boolean canRead()</code>	判断应用程序是否可以 修改 当前File实例所表示的文件或目录
<code>public boolean setWritable(boolean writable)</code>	设置当前File实例所表示的文件或目录的拥有者的写权限(参数 true 表示可写)
<code>public boolean setWritable(boolean writable, boolean ownerOnly)</code>	设置当前File实例所表示的文件或目录的拥有者或所有用户的写权限
<code>public boolean canExecute()</code>	判断应用程序是否可以 执行 当前File实例所表示的文件(只有文件才可以执行)
<code>public boolean setExecutable(boolean executable)</code>	设置当前File实例所表示的文件或目录的拥有者的执行权限(true 表示可执行)
<code>public boolean setExecutable(boolean executable, boolean ownerOnly)</code>	设置当前File实例所表示的文件或目录的拥有者或所有用户的执行权限

File Operations

方法	描述
<code>public String getName()</code>	返回当前File实例所表示的文件或目录对应的名称(文件名称或目录名称)
<code>public String getPath()</code>	返回当前File实例所表示的文件或目录的路径(以字符串形式表示)
<code>public File getParentFile()</code>	返回当前File实例的父目录(上级路径)对应的File实例
<code>public String getParent()</code>	返回当前File实例的父目录(以字符串形式表示)
<code>public File getAbsoluteFile()</code>	返回当前File实例的绝对路径对应的File实例
<code>public String getAbsolutePath()</code>	返回当前File实例的绝对路径(以字符串形式表示)
<code>public File getCanonicalFile()</code>	返回当前File实例的规范形式(返回的是一个File实例)
<code>public String getCanonicalPath()</code>	返回当前File实例的规范路径名(以字符串形式表示)
<code>public URI toURI()</code>	返回与当前File实例对应的URI实例 (其形式是 file: /User/malajava/hello.txt)

File Operations

方法	描述
public boolean createNewFile()	创建当前File实例所表示的文件(仅当该File实例所表示的文件不存在时才能够创建成功)
public boolean mkdir()	创建当前File实例所表示的目录(仅创建当前目录，不能创建父目录)
public boolean mkdirs()	创建当前File实例所表示的目录,包括所有必须但不存在的父目录
public boolean renameTo(File dest)	将当前File实例所表示的文件或目录重命名为指定File实例对应的路径名称
public boolean delete()	删除当前File实例所表示的文件或目录
public void deleteOnExit()	在虚拟机终止时，删除当前File实例所表示的文件或目录

注意：只有当应用程序对指定File实例有写权限时才可以执行以上操作

File Operations

静态方法	描述
<code>public static File[] listRoots()</code>	获取所有可用的文件系统的根路径对应的File实例 Windows 系统下如果有多个分区则对应多个File实例 而对于 Unix / Linux / OS X 系统，则所返回的File数组中只有一个元素
<code>public static File createTempFile(String prefix, String suffix, File directory)</code>	在指定的目录中创建一个新的空文件， 并使用给定的前缀和后缀生成该文件的名称。
<code>public static File createTempFile(String prefix, String suffix)</code>	在默认的临时目录中创建一个新的空文件， 并使用给定的前缀和后缀生成该文件的名称。

对于跟路径，Windows 平台下可能是 C:\，而 Unix / Linux / OS X 系统中则是 /

注意：对于创建文件操作，只有拥有目标文件的父目录的写权限时，才能成功

File Filtering

```
package java.io;  
  
@FunctionalInterface  
public interface FilenameFilter {  
  
    boolean accept ( File dir, String name );  
  
}
```

```
File dir = new File( "/Users/malajava/" );  
  
FilenameFilter filter = ..... ;  
  
//作为参数传递给 list 方法  
String[] filenames = dir.list( filter );  
  
//作为参数传递给 listFiles 方法  
File[] files = dir.listFiles( filter );
```

接口 `java.io.FilenameFilter` 可以用作文件名过滤器
`FilenameFilter` 提供了测试指定文件是否应该包含在某一文件列表中
其实例一般作为参数传递给 `File` 对象的 `list` 方法或 `listFiles` 方法

File Filtering

```
FilenameFilter filenameFilter = new FilenameFilter() {  
    @Override  
    public boolean accept ( File dir, String name ) {  
        if (name.endsWith(".java")) {  
            return true; // 当 文件名 是以 .java 结尾时返回 true  
        }  
        return false;  
    }  
};  
  
File dir = new File( "/Users/malajava/" );  
File[] files = dir.listFiles( filenameFilter ); // 作为参数传递给 listFiles 方法
```

FilenameFilter 使用举例

File Filtering

```
package java.io;  
  
@FunctionalInterface  
public interface FileFilter {  
  
    boolean accept ( File pathname );  
  
}
```

```
File dir = new File( "/Users/malajava/" );  
  
FileFilter filter = ..... ;  
  
//作为参数传递给 listFiles 方法  
File[] files = dir.listFiles( filter );
```

接口 `java.io.FileFilter` 可以用作文件过滤器

`FileFilter` 提供了测试指定文件是否应该包含在某一文件列表中
其实例一般作为参数传递给 `File` 对象的 `listFiles` 方法

File Filtering

```
final Calendar c = Calendar.getInstance(); // 获得 Calendar 实例
c.set( 2017, 0, 1, 0, 0, 0 ); // 设置时间为 2017年1月1日00:00:00
c.set( Calendar.MILLISECOND, 0 );
FileFilter fileFilter = new FileFilter() {
    @Override
    public boolean accept( File f ) {
        // 判断当前文件或目录的最后修改时间是否在 2017年1月1日00:00:00之后
        if ( f.lastModified() >= c.getTimeInMillis() ) {
            return true; // 如果是就返回 true 则该 File 对象将包含在被选择的列表中
        }
        return false;
    }
};

File dir = new File( "/Users/malajava/" );
File[] files = dir.listFiles(fileFilter); // 作为参数传递给 listFiles 方法
```

FileFilter 使用举例

Part 2

Class File

Streams

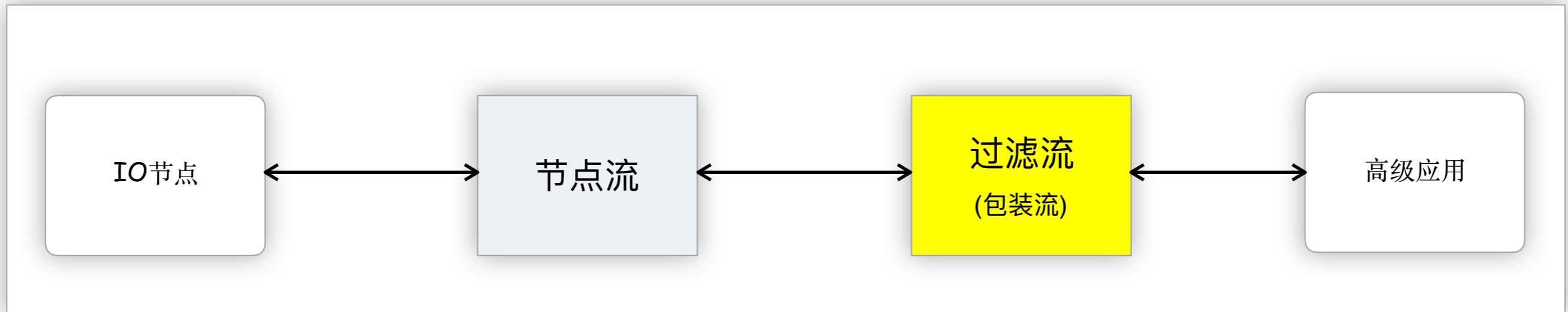
Access Files

Object Serialization

New Java IO

Introduction to Streams

	字节流	字符流
输入流	<code>InputStream</code>	<code>Reader</code>
输出流	<code>OutputStream</code>	<code>Writer</code>

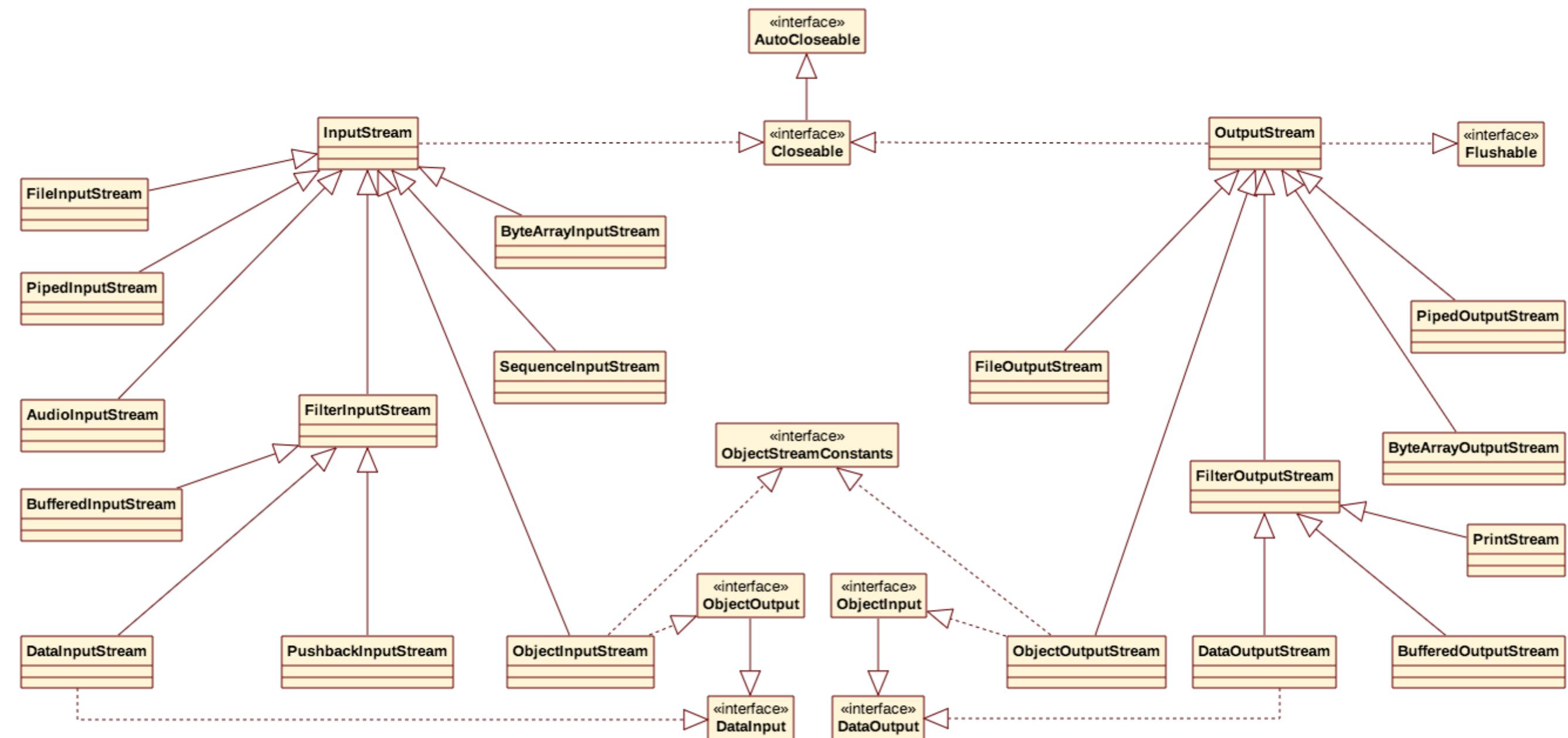


流的分类

Introduction to Streams

在文件和流操作中，绝大多数方法都会抛出 IOException

Interfaces and Classes for Byte-Based Input and Output



Class InputStream



InputStream 类是所有表示字节输入流的类的父类

Class InputStream

方法	描述
<code>public int available()</code>	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
<code>public abstract int read()</code>	从输入流中读取数据的下一个字节 (所有非抽象的子类必须实现该方法)
<code>public int read(byte[] bytes)</code>	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 bytes 中
<code>public int read(byte[] bytes, int offset, int length)</code>	将输入流中最多 length 个数据字节读入 bytes 数组(从 offset 位置开始放入)
<code>public long skip(long n)</code>	跳过和丢弃此输入流中数据的 n 个字节
<code>public boolean markSupported()</code>	测试此输入流是否支持 mark 和 reset 方法 (当某个流支持 mark 和 reset 操作时返回 true)
<code>public void mark(int readlimit)</code>	在此输入流中标记当前位置(对于大部分流而言参数readlimit毫无意义)
<code>public void reset()</code>	将此流重新定位到最后一次对此输入流调用 mark 方法时的位置
<code>public void close()</code>	关闭此输入流并释放与该流关联的所有系统资源 (该方法来自 Closeable 接口)

InputStream 类中定义的方法被所有子类所继承

Class FileInputStream

构造方法	描述
<code>public FileInputStream(String name)</code>	通过打开一个到实际文件的连接来创建一个 FileInputStream, 该文件通过文件系统中的路径名 name 指定
<code>public FileInputStream(File file)</code>	通过打开一个到实际文件的连接来创建一个 FileInputStream, 该文件通过文件系统中的 File 对象 file 指定
<code>public FileInputStream(FileDescriptor descriptor)</code>	通过使用文件描述符 fdObj 创建一个 FileInputStream, 该文件描述符表示到文件系统中某个实际文件的现有连接

FileInputStream 从文件系统中的某个文件中获得输入字节

哪些文件可用取决于主机环境

FileInputStream 用于读取诸如图像数据之类的原始字节流

InputStream and FileInputStream

```
public static void main(String[] args) throws IOException {  
  
    // 创建一个用来读取当前用户主目录下的 Hello.java 文件的输入流  
    InputStream in = new FileInputStream( "/Users/malajava/Hello.java" );  
    // 声明一个变量用来记录读取到的单个字节  
    int b;  
    // 每循环一次读取一个字节，并将读取到的字节赋值给 变量 n，当到达流末尾时返回-1  
    while( ( b = in.read() ) != -1 ) {  
        // 将读取到的字节转换成字符类型，以便于在控制台中输出后查看  
        char c = (char) b;  
        // 将每个字节对应的字符输出到控制台上(文件中的换行符号会当作换行处理，因此输出时不需要带换行)  
        System.out.print( c );  
    }  
  
    in.close();  
}
```

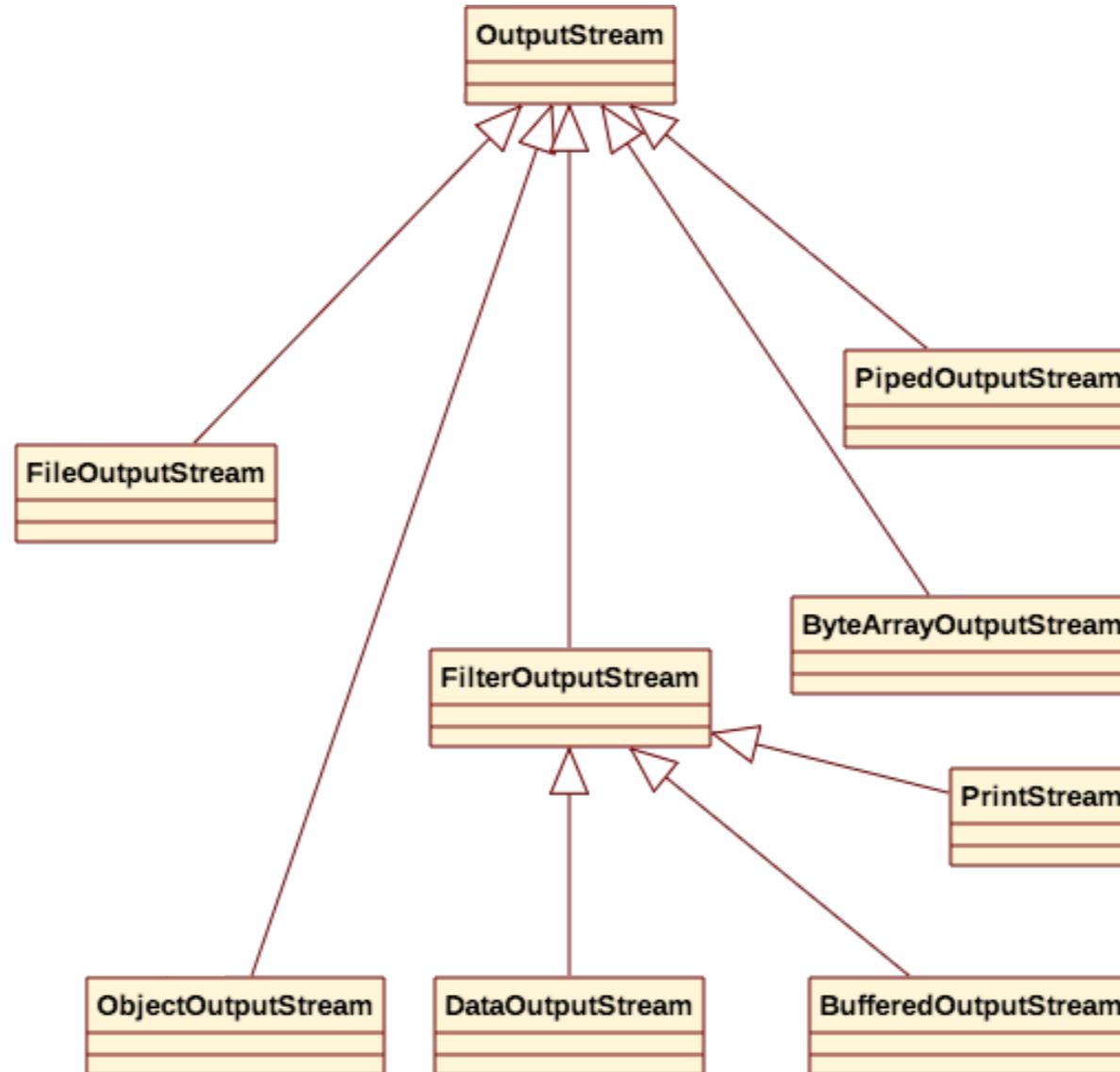
示例：读取指定文件的内容，每次只读取一个字节

InputStream and FileInputStream

```
public static void main(String[] args) throws IOException {
    // 创建一个用来读取当前用户主目录下的 Hello.java 文件的输入流
    InputStream in = new FileInputStream("/Users/malajava/Hello.java");
    // 声明一个变量用来统计读取到的字节数
    int n;
    // 声明并创建一个数组用来保存读取到的字节
    byte[] bytes = new byte[ 32 ];
    // 从输入流中读取字节到数组中，当到达流末尾时返回-1
    while ( ( n = in.read( bytes ) ) != -1 ) {
        // 将读取到的字节内容构造成字符串
        String s = new String( bytes , 0 , n );
        // 将本次读取到的字符串输出到控制台(文件中的换行符号会当作换行处理，因此输出时不需要带换行)
        System.out.print( s );
    }
    in.close();
}
```

示例：读取指定文件的内容，每次最多读取 bytes.length 个字节(实际读取到的字节数记录在变量n中)

Class OutputStream



OutputStream 类是所有表示字节输出流的类的父类

Class OutputStream

方法	描述
<code>public abstract void write(int b)</code>	将指定的字节写入当前输出流
<code>public void write(byte[] bytes)</code>	将 bytes.length 个字节从指定的 byte 数组写入当前输出流
<code>public void write(byte[] bytes, int offset, int length)</code>	将指定 byte 数组中从 offset 开始的 length 个字节写入此输出流
<code>public void flush()</code>	刷新当前输出流并强制写出所有缓冲的输出字节
<code>public void close()</code>	关闭当前输出流并释放与此流有关的所有系统资源

OutputStream 类中定义的方法被所有子类所继承

Class FileOutputStream

构造方法	描述
<code>public FileOutputStream(String name)</code>	创建一个向指定文件中写入数据的文件输出流
<code>public FileOutputStream(String name, boolean append)</code>	创建一个向指定文件中写入数据的文件输出流，并指定是否以追加方式输出到文件
<code>public FileOutputStream(File file)</code>	创建一个向指定 File 对象表示的文件中写入数据的文件输出流
<code>public FileOutputStream(File file, boolean append)</code>	创建一个向指定 File 对象表示的文件中写入数据的文件输出流，并指定是否以追加方式输出
<code>public FileOutputStream(FileDescriptor descriptor)</code>	创建一个向指定文件描述符处写入数据的文件输出流，该文件描述符表示一个到文件系统中的某个实际文件的现有连接

FileOutputStream 是用于向文件中写入字节的输出流

哪些文件可以被写入取决于基础平台

FileOutputStream 用于写入诸如图像数据之类的原始字节流

OutputStream and FileOutputStream

```
public static void main(String[] args) throws IOException {
    // 声明一个字符串，这个字符串中的内容将被输出到指定文件中
    String s = "hello,malajava !";
    // 根据平台默认字符集将给定的字符串编码为 byte 序列(一个byte数组)
    final byte[] bytes = s.getBytes();
    // 创建文件输出流
    OutputStream out = new FileOutputStream( "fileout.txt");
    // 通过循环遍历 bytes 数组
    for( int i = 0 , n = bytes.length ; i < n ; i++ ){
        // 获取 bytes 数组中的 单个字节
        byte b = bytes[ i ];
        // 将单个字节写入到文件输出流中(文件输出流会将这个字节写入文件中)
        out.write( b );
    }
    out.close(); // 关闭文件输出流
}
```

示例 :将给定的字节数组中的内容输出到文件中，每次只向输出流中写入一个字节

OutputStream and FileOutputStream

```
public static void main(String[] args) throws IOException {  
  
    // 声明一个字符串，这个字符串中的内容将被输出到指定文件中  
    String s = "hello,malajava !";  
    // 根据平台默认字符集将给定的字符串编码为 byte 序列(一个byte数组)  
    final byte[] bytes = s.getBytes();  
  
    // 创建文件输出流  
    OutputStream out = new FileOutputStream( "fileout.txt");  
  
    // 将 bytes 数组中的全部字节一次写入到文件输出流中(文件输出流会将这些字节写入文件中)  
    out.write( bytes );  
  
    out.close(); // 关闭文件输出流  
  
}
```

示例 :将给定的字节数组中的内容一次输出到文件中

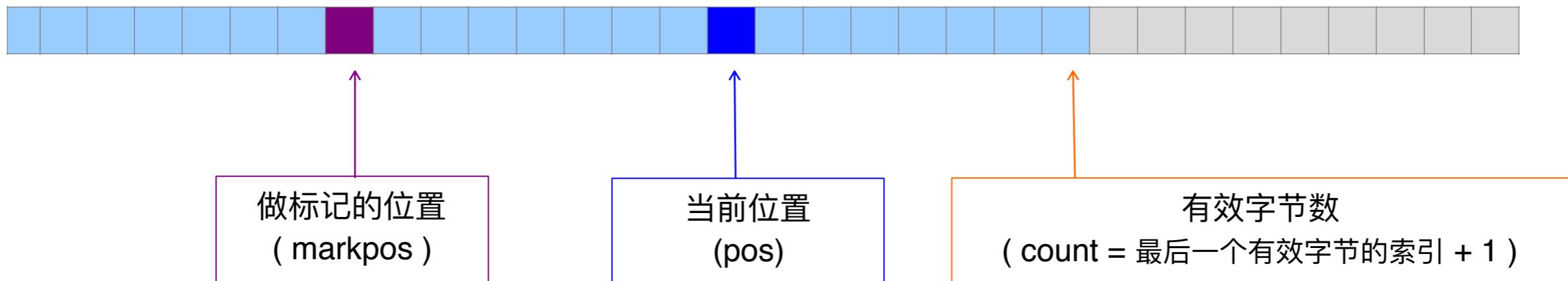
OutputStream and FileOutputStream

```
public static void main(String[] args) throws IOException {
    // 创建一个文件输入流(将从这个文件中复制内容)
    InputStream in = new FileInputStream( "/Users/mala/java/java.jpg" );
    // 创建一个文件输出流(被复制的内容将输出到这个文件中)
    OutputStream out = new FileOutputStream( "copy.jpg" );
    // 声明一个变量用来统计读取到的字节数
    int n;
    // 声明并创建一个数组用来保存读取到的字节
    byte[] bytes = new byte[ 32 ];
    // 从输入流中读取字节到数组中并记录读取到的有效字节数, 当到达流末尾时返回-1
    while ( ( n = in.read( bytes ) ) != -1 ) {
        // 将本次读取到的有效字节写入到文件输出流
        out.write( bytes, 0 , n );
    }
    out.close(); // 先关闭文件输出流
    in.close(); // 再关闭文件输入流
}
```

示例: 用 FileInputStream 和 FileOutputStream 实现文件的复制操作

Class BufferedInputStream

BufferedInputStream 对象内部的 buf 数组(默认8192字节)



BufferedInputStream 是具有缓冲功能的字节输入流

每个 `BufferedInputStream` 对象内部都有一个字节数组用来缓存读取到的字节

`BufferedInputStream` 是一个过滤流(包装流)，它继承自 `FilterInputStream` 类

Class BufferedInputStream

常量	描述
<code>protected byte[] buf</code>	存储数据的内部缓冲区数组
<code>protected int count</code>	字符缓冲区中的有效字节数 (比缓冲区中最后一个有效字节的索引大 1)
<code>protected int markpos = -1;</code>	最后一次调用 mark 方法时 pos 字段的值 (第一次标记之前 markpos 的值是 -1)
<code>protected int pos</code>	缓冲区中的当前位置
<code>protected int marklimit</code>	调用 mark 方法后, 在后续调用 reset 方法失败之前所允许的最大提前读取量

BufferedInputStream 从父类 FilterInputStream 中继承的属性:

`protected volatile InputStream in ;`

FilterInputStream 类中的这个属性用来保存被包装的那个字节输入流

Class BufferedInputStream

构造方法	描述
<code>public BufferedInputStream(InputStream in)</code>	根据给定的字节输入流创建一个具有默认缓冲大小的 BufferedInputStream 实例
<code>public BufferedInputStream(InputStream in , int size)</code>	根据给定的字节输入流创建一个具有指定缓冲大小的 BufferedInputStream 实例

Class BufferedInputStream

```
public static void main(String[] args) throws IOException {  
  
    // 创建一个文件输入流  
    InputStream in = new FileInputStream( "/Users/malajava/Hello.java" );  
    // 创建字节缓冲输入流  
    BufferedInputStream bis = new BufferedInputStream( in );  
    // 声明一个变量用来记录读取到的字节数  
    int n;  
    // 声明并创建一个数组用来存储被读取到的字节  
    byte[] bytes = new byte[32];  
    // 从字节缓冲输入流中读取数据到 字节数组中，并记录读取到的实际字节数，当到达流末尾时返回 -1  
    while( ( n = bis.read( bytes ) ) != -1 ){  
        // 将读取到的有效字节构造成字符串  
        String s = new String( bytes , 0 , n );  
        System.out.print( s );  
    }  
  
    bis.close(); // 关闭字节缓冲输入流  
    in.close(); // 关闭字节输入流  
}
```

示例：从字节缓冲输入流中读取字节，每次最多读取 bytes.length 个（有效字节数被记录在变量 n 中）

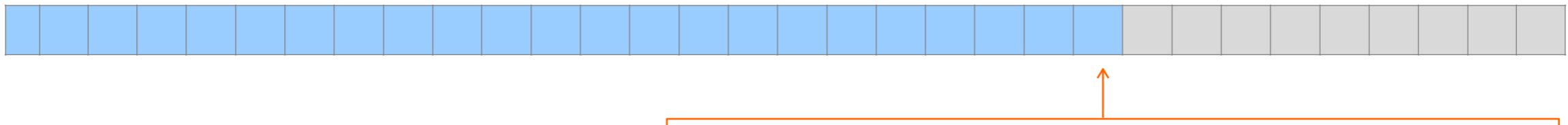
mark and reset

```
public static void main(String[] args) throws IOException {
    InputStream in = new FileInputStream( "/Users/malajava/Hello.java" );
    BufferedInputStream bis = new BufferedInputStream( in );
    boolean marked = false; // 声明一个变量用来记录是否对流做过标记
    int n; // 声明一个变量用来记录读取到的字节 (注意是字节, 不是字节数)
    while( ( n = bis.read() ) != -1 ){// 从缓冲流中读取单个字节, 当读取到流末尾时返回 -1
        char ch = (char) n; // 将读取到的那个字节转换成字符
        System.out.print( ch ); // 将字符输出到控制台
        if( ch == 'a' ){ // 如果当前读取到的字符是 'a' , 则准备做标记
            // 如果当前的流支持 mark 操作
            if( bis.markSupported() ){
                // 那就在当前读取到的字节处做标记
                bis.mark( 250 );
                marked = true;
            }
        }
    }
    System.out.println( "~~~~~" ); // 为了便于区分, 输出一条华丽的分隔线
    if( marked ){ // 如果曾经做过标记
        bis.reset(); // 将缓冲流的 当前位置 重置 到 最后一次做标记的位置
        while( ( n = bis.read() ) != -1 ){
            char ch = (char) n;
            System.out.print( ch ); // 将字符输出到控制台
        }
    }
    bis.close(); // 关闭字节缓冲输入流
    in.close(); // 关闭字节输入流
}
```

因为文件中可能有多个 'a' 字符,
因此可能会做多次标记。
多次标记以最后一次标记为有效

Class BufferedOutputStream

BufferedOutputStream 对象内部的 buf 数组(默认8192字节)



BufferedOutputStream 是具有缓冲功能的字节输出流

每个 `BufferedOutputStream` 对象内部都有一个字节数组用来缓存将要输出的字节

`BufferedOutputStream` 是一个过滤流(包装流), 它继承自 `FilterOutputStream` 类

Class BufferedOutputStream

常量	描述
protected byte[] buf	用来存储数据的内部缓冲区(所有将要输出的字节，都会先输出到这个字节数组中)
protected int count	已经向缓冲区中写入的有效字节数

BufferedOutputStream 从父类 FilterOutputStream 中继承的属性:

protected OutputStream out ;

FilterOutputStream 类中的这个属性用来保存被包装的那个字节输出流

Class BufferedOutputStream

构造方法	描述
<code>public BufferedOutputStream(OutputStream out)</code>	创建一个带有默认缓冲大小的缓冲输出流，并将数据写入到指定的底层输出流
<code>public BufferedOutputStream(OutputStream out , int size)</code>	创建一个带有指定缓冲大小的缓冲输出流，并将数据写入到指定的底层输出流

Class BufferedOutputStream

方法	描述
<code>public synchronized void write(int b)</code>	将指定的单个字节写入到当前缓冲流中
<code>public synchronized void write(byte[] b, int off, int len)</code>	将指定的 byte 数组中从 off 开始的 len 个字节写入到当前缓冲流中
<code>public synchronized void flush()</code>	将当前缓冲区中的字节刷出到底层字节流

BufferedOutputStream 类中重写的方法

Class BufferedOutputStream

```
public static void main(String[] args) throws IOException {  
  
    // 创建一个可以向指定文件输出字节的输出流  
    OutputStream out = new FileOutputStream( "buffered-out.txt" );  
    // 创建一个具有默认缓冲大小的 缓冲输出流  
    BufferedOutputStream bos = new BufferedOutputStream( out );  
  
    String s = "hello,maJava !";  
    // 根据当前平台的默认编码将给定字符串编码为 byte 序列 (编码后是一个byte数组)  
    final byte[] bytes = s.getBytes();  
  
    // 将 bytes 数组中的 字节 写入到 缓冲输出流中  
    bos.write( bytes );  
  
    bos.close(); // 关闭缓冲流 (同时调用 flush 方法将缓冲区中的 字节 写入底层输出流 )  
    out.close(); // 关闭底层的字节输出流  
  
}
```

示例：将字节数组中的每个字节一次写入到缓冲输出流，并最终通过底层输出流将内容输出到文件中

Class PrintStream

```
public class FilterOutputStream extends OutputStream {  
    protected OutputStream out;  
    .....  
}  
  
public class PrintStream extends FilterOutputStream implements Appendable, Closeable {  
    .....  
    private BufferedWriter textOut;  
    private OutputStreamWriter charOut;  
    .....  
}
```

PrintStream 类继承了 FilterOutputStream 类 并 实现了 Appendable 接口
PrintStream 内部通过 OutputStreamWriter 将字节流转换成字符流
通过 BufferedWriter 将所要打印的内容缓存，从而支持 append 操作
与其它流不同，PrintStream 永远不会抛出 IOException

Class PrintStream

方法	描述
public PrintStream(String fileName)	创建具有指定文件名称且不带自动行刷出的新打印流
public PrintStream(String fileName, String csn)	创建具有指定文件名称和字符集且不带自动行刷出的新打印流
public PrintStream(File file)	创建具有指定文件且不带自动行刷出的新打印流
public PrintStream(File file, String csn)	创建具有指定文件名称和字符集且不带自动行刷出的新打印流
public PrintStream(OutputStream out)	创建新的输出到指定底层输出流的打印流
public PrintStream(OutputStream out, boolean autoFlush)	创建新的输出到指定底层输出流的打印流(并指定是否自动刷出)
public PrintStream(OutputStream out, boolean autoFlush, String csn)	创建新的输出到指定底层输出流的打印流(并指定是否自动刷出、指定字符集)

Class PrintStream

```
public static void main(String[] args) throws IOException {  
  
    PrintStream ps = new PrintStream( "print.txt" );  
  
    ps.append( '张' ); // 向打印流末尾追加单个字符  
  
    ps.append( "三丰" ); // 向打印流末尾追加字符串  
  
    ps.println(); // 打印换行符  
  
    ps.append( "武当掌门张三丰" , 0, 4 ); // 从给定字符串中的第 0 个开始取 4 个字符追加到打印流末尾  
  
    ps.println();  
  
    ps.println( true ); // 通过打印流打印一个 boolean 类型的数据  
  
    ps.close();  
}
```

Standard Streams

常量	描述
<code>public final static InputStream in = null ;</code>	"标准"输入流 (默认是读取用户键盘输入的数据)
<code>public final static PrintStream out = null ;</code>	"标准"输出流 (默认是输出到控制台中)
<code>public final static PrintStream err = null ;</code>	"标准"错误输出流 (默认是输出到控制台中)

```
public final class System {  
    ....  
    private static native void registerNatives();  
    static {  
        registerNatives();  
    }  
    ....  
}
```

System 类静态代码块中
通过调用 registerNatives 方法
实现对 in 、 out 等变量的初始化
并完成其它的操作

Standard Streams

常量	描述
<code>public static void setIn(InputStream in)</code>	重新分配"标准"输入流
<code>public static void setOut(PrintStream out)</code>	重新分配"标准"输出流
<code>public static void setErr(PrintStream err)</code>	重新分配"标准"错误输出流

```
public static void setOut( PrintStream out ){  
    checkIO();  
    setOut0(out);  
}
```

```
private static native void setOut0( PrintStream out );
```

System 类中 setOut 方法的实现：
调用了一个私有的本地方法
这个本地方法完成流的重新分配

Redirecting standard streams

```
public static void main(String[] args) throws IOException {
    System.out.println("hello,standard"); // 这句将输出到控制台中
    // 创建一个可以向指定文件中输出内容的打印流
    PrintStream out = new PrintStream("standard.txt");
    // 声明一个变量记录原来的"标准"输出流
    final PrintStream standard = System.out;
    System.out.print("redirecting"); // 这句将输出到控制台中

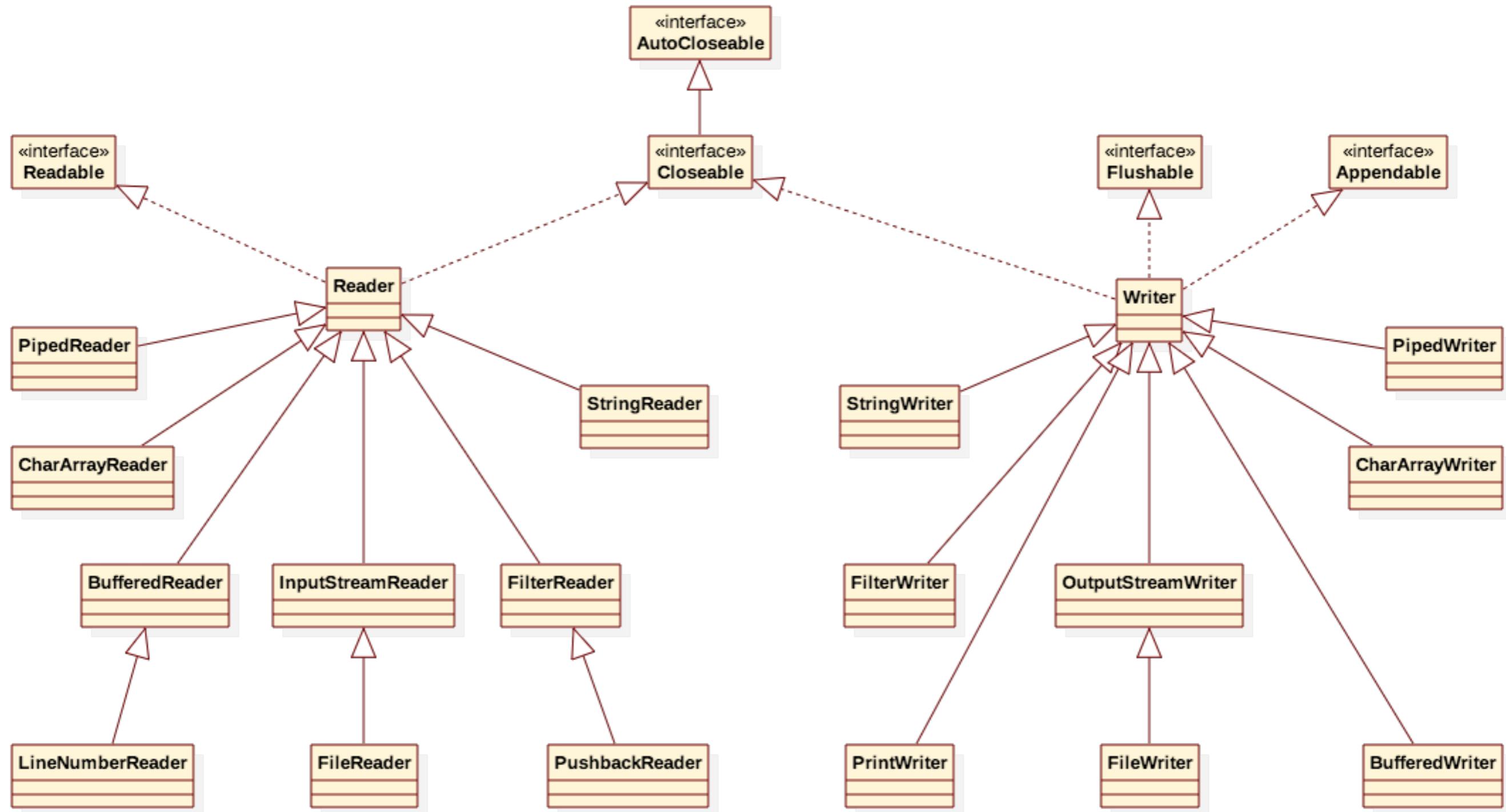
    // 重定向 "标准" 输出流到指定的打印流
    System.setOut(out);

    System.out.print("redirected"); // 这句将输出到指定的打印流中

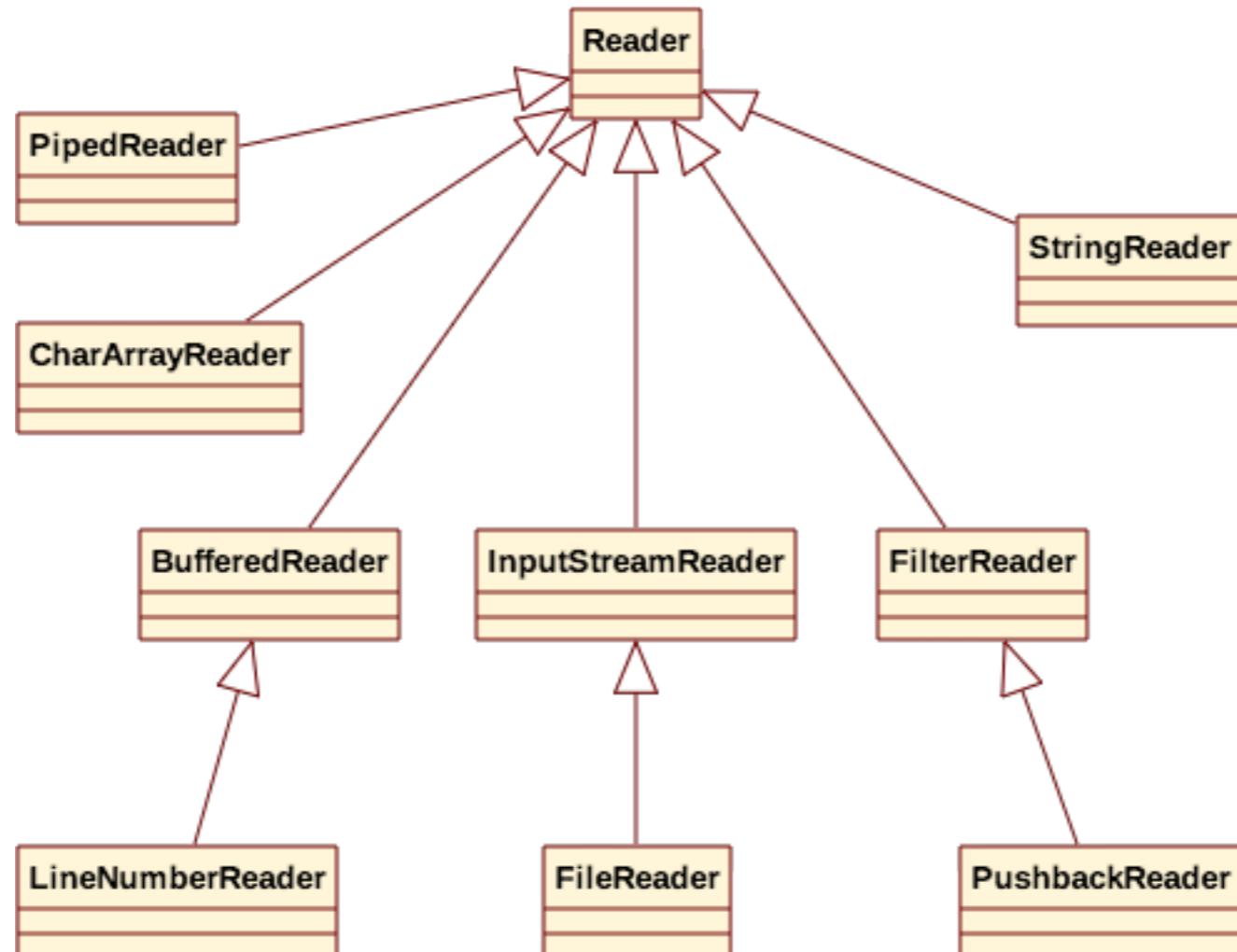
    // 将"标准"输出流重定向到原来的流
    System.setOut(standard);

    System.out.println("i am back"); // 这句将输出到控制台中
}
```

Interfaces and Classes for Character-Based Input and Output



Class Reader



Reader 类是所有表示字符输入流的类的父类

Class Reader

方法	描述
<code>public int read()</code>	从输入流中读取下一个字符
<code>public int read(char[] chars)</code>	从输入流中读取一定数量的字符，并将其存储在缓冲区数组 chars 中
<code>public abstract int read(char[] chars, int offset, int length)</code>	将输入流中最多 length 个字符读入 chars 数组(从 offset 位置开始放入)
<code>public long skip(long n)</code>	跳过和丢弃此输入流中数据的 n 个字符
<code>public boolean markSupported()</code>	测试此输入流是否支持 mark() 和 reset() 操作
<code>public void mark(int readlimit)</code>	在此输入流中标记当前位置(对于大部分流而言参数readlimit毫无意义)
<code>public void reset()</code>	重置此流 (将当前位置重新定位到最后一次对此输入流调用 mark 方法时的位置)
<code>public boolean ready()</code>	判断是否准备读取此流
<code>public int read(java.nio.CharBuffer target)</code>	从输入流中将字符读入指定的字符缓冲区
<code>public abstract void close()</code>	关闭此输入流并释放与该流关联的所有系统资源 (该方法来自 Closeable 接口)

Reader 类中定义的方法被所有子类所继承

Class InputStreamReader

构造方法	描述
<code>public InputStreamReader(InputStream in)</code>	创建一个使用默认字符集的 InputStreamReader
<code>public InputStreamReader(InputStream in, String charsetName)</code>	创建使用指定字符集的 InputStreamReader
<code>public InputStreamReader(InputStream in, Charset cs)</code>	创建使用给定字符集的 InputStreamReader
<code>public InputStreamReader(InputStream in, CharsetDecoder dec)</code>	创建使用给定字符集解码器的 InputStreamReader



使用 InputStreamReader 可以将字节输入流包装成字符输入流，从而实现从字节到字符的转换

Class InputStreamReader

```
public static void main(String[] args) throws IOException {
    // 创建一个可以读取指定文件内容的文件输入流
    InputStream in = new FileInputStream( "/Users/malajava/names.txt" );
    // 创建一个 InputStreamReader ( 将指定的字节数入流按照默认字符集包装成字符输入流 )
    Reader reader = new InputStreamReader( in );
    // 声明一个变量用来保存读取到的字符
    int ch;
    // 从 字符输入流中读取单个字符并保存到变量 n 中, 当到达流末尾时返回-1
    while( ( ch = reader.read() ) != -1 ){
        // 因为 read 读取字符后返回的是 int 类型数据, 因此将其强制类型转换为 char 类型
        char c = (char) ch;
        System.out.print( c );
    }
    reader.close(); // 关闭字符输入流
    in.close(); // 关闭字节输入流
}
```

示例：通过 InputStreamReader 将字节输入流包装成字符数入流，通过循环逐个读取字符

Class FileReader

构造方法	描述
<code>public FileReader(String name)</code>	通过指定文件路径名来创建一个新 FileReader 实例
<code>public FileReader(File file)</code>	通过指定 File 实例 来创建一个新 FileReader 实例
<code>public FileReader(FileDescriptor descriptor)</code>	在给定 FileDescriptor 创建一个新 FileReader 实例

FileReader 类被设计为用来读取字符文件

FileReader 类继承了 InputStreamReader 因此可以具有将字节流转换成字符流的能力

Class FileReader

```
public static void main(String[] args) throws IOException {
    // 创建一个以字符为单位读取文件的输入流
    FileReader r = new FileReader( "/Users/malajava/names.txt" );
    // 声明一个变量用来统计读取到的字符数
    int n;
    // 声明并创建一个字符数组，用来保存读取到的字符
    final char[] chars = new char[ 32 ];
    // 从 reader 中读取字符到 chars 数组中并记录读取到的字符数，当到达流末尾时返回-1
    while( ( n = r.read( chars ) ) != -1 ){
        // 将 chars 数组中的 [ 0 , n ) 之间的字符构造成字符串
        String s = new String( chars , 0 , n );
        System.out.print( s );
    }
    r.close(); // 关闭文件输入流
}
```

示例：通过 FileReader 读取指定文件中的内容，每次最多读取 chars.length 个字符

Class BufferedReader

构造方法	描述
<code>public BufferedReader(Reader in)</code>	创建一个使用默认大小输入缓冲区的缓冲字符输入流
<code>public BufferedReader(Reader in , int size)</code>	创建一个使用指定大小输入缓冲区的缓冲字符输入流

BufferedReader 类内部采用 char 数组(默认大小为8192)来缓存读入的字符。

BufferedReader 从字符输入流中读取文本，缓冲各个字符，从而实现字符、数组和行的高效读取

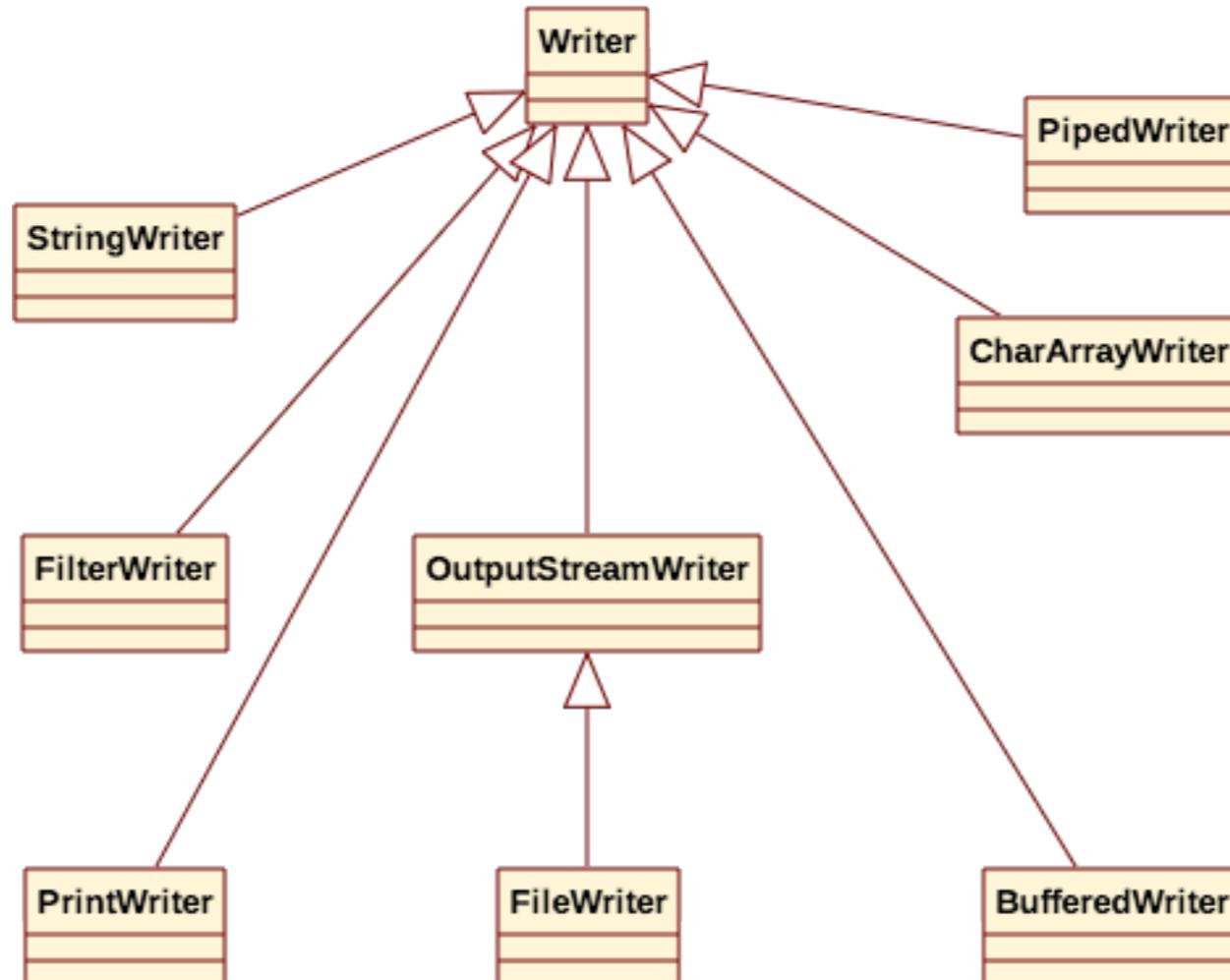
BufferedReader 类几乎重写了 Reader 中的所有方法，另外增加了 readLine 方法

Class BufferedReader

```
public static void main( String[] args ) throws IOException {
    // 创建一个可以读取指定文件内容的字符输入流
    Reader r = new FileReader( "/Users/malajava/names.txt" );
    // 将指定的字符输入流包装成 字符缓冲输入流
    BufferedReader br = new BufferedReader( r );
    // 声明一个变量用来接收从缓冲输入流中读取到的字符串
    String s;
    // 从字符缓冲输入流中读取一行数据(每次都读到换行符为止), 当到达流末尾时返回 null
    while( ( s = br.readLine() ) != null ){
        // 将读取到的字符串输出到控制台
        System.out.println( s );
    }
    br.close();
    r.close();
}
```

示例：将字符输入流包装成字符缓冲输入流，每次读取一行数据

Class Writer



Writer 类是所有表示字符输出流的类的父类

Class Writer

方法	描述
<code>public void write(int c)</code>	将指定的字符写入当前输出流
<code>public void write(char[] chars)</code>	将 bytes.length 个字符从指定的 chars 数组写入当前输出流
<code>public abstract void write(char[] chars, int offset, int length)</code>	将指定 chars 数组中从 offset 开始的 length 个字符写入此输出流
<code>public void write(String string)</code>	将指定的字符串写入当前输出流
<code>public void write(String string, int offset, int length)</code>	将指定的字符串中的一部分(从 offset 开始的 length 个)写入当前输出流
<code>public Writer append(char c)</code>	将指定字符追加到当前输出流末尾
<code>public Writer append(CharSequence csq)</code>	将指定的字符序列追加到当前流末尾
<code>public Writer append(CharSequence csq, int start, int end)</code>	将指定字符序列中的 [start , end) 之间的字符追加到当前输出流末尾
<code>public abstract void flush()</code>	刷新当前输出流并强制写出所有缓冲的输出字符
<code>public abstract void close()</code>	关闭当前输出流并释放与此流有关的所有系统资源

Writer 类中定义的方法被所有子类所继承

Class OutputStreamWriter

构造方法	描述
<code>public OutputStreamWriter(OutputStream out)</code>	创建使用默认字符编码的 OutputStreamWriter
<code>public OutputStreamWriter(OutputStream out, String charsetName)</code>	创建使用指定字符集的 OutputStreamWriter
<code>public OutputStreamWriter(OutputStream out, Charset cs)</code>	创建使用给定字符集的 OutputStreamWriter
<code>public OutputStreamWriter(OutputStream out, CharsetDecoder dec)</code>	创建使用给定字符集编码器的 OutputStreamWriter



使用 `OutputStreamWriter` 可以将字符输出流转换为字节输入流，从而实现从字符到字节的转换

Class OutputStreamWriter

```
public static void main(String[] args) throws IOException {  
  
    OutputStream out = new FileOutputStream( "w.txt" );  
  
    Writer w = new OutputStreamWriter( out );  
  
    w.write( '国' ); // 将单个字符写入缓冲输出流  
    w.write( '军' );  
    w.write( "威武" ); // 将一个字符序列(字符串)写入缓冲输出流  
  
    w.write( "\n" ); // 将单个字符写入缓冲输出流  
  
    w.append( "hello" ); // 将一个字符串序列(字符串)追加到缓冲输出流末尾  
  
    w.flush(); // 将缓冲输出流中的字符刷出  
  
    w.close(); // 关闭缓冲输出流  
    out.close(); // 关闭字节输出流  
}
```

示例：通过 OutputStreamWriter 将 字符或字符串 写入到底层字节输出流中

Class OutputStreamWriter

```
public static void main(String[] args) throws IOException {
    // 创建一个以字节为读取单位的文件输入流
    InputStream in = new FileInputStream( "/Users/malajava/utf8.txt" );
    // 创建一个以字节为输出单位的文件输出流
    OutputStream out = new FileOutputStream( "after.txt" );

    // 将文件输入流 in 按照 指定字符集 ( UTF-8 ) 包装成 字符数入流
    Reader r = new InputStreamReader( in , "UTF-8" );
    // 将文件输出流 out 按照 指定字符集 ( GBK ) 包装成 字符输出流
    Writer w = new OutputStreamWriter( out , "GBK" );
    // 声明一个变量用来保存读取到的字符
    int ch ;
    // 每次读取一个字符，当读到流末尾时返回 -1
    while( ( ch = r.read() ) != -1 ){
        // 将读取到的字符写入到 字符输出流
        w.write( ch );
    }
    w.close(); // 关闭字符输出流
    out.close(); // 关闭字节输出流
    r.close(); // 关闭字符输入流
    in.close(); // 关闭字节输入流
}
```

示例：实现简单的文本文件编码转换器

Class FileWriter

构造方法	描述
<code>public FileWriter(String name)</code>	通过指定文件路径名来创建一个新 FileWriter 实例
<code>public FileWriter(String name , boolean append)</code>	通过指定文件路径名来创建一个新 FileWriter 实例，并指定是否以追加方式输出到文件
<code>public FileWriter(File file)</code>	通过指定 File 实例 来创建一个新 FileWriter 实例
<code>public FileWriter(File file , boolean append)</code>	通过指定 File 实例 来创建一个新 FileWriter 实例，并指定是否以追加方式输出到文件
<code>public FileWriter(FileDescriptor descriptor)</code>	在给定 FileDescriptor 创建一个新 FileWriter 实例

FileWriter 类被设计为用来写入字符文件

FileWriter 类继承了 OutputStreamWriter 因此可以具有将字符流转换成字节流的能力

Class FileWriter

```
public static void main(String[] args) throws IOException {  
  
    // 创建一个可以向指定文件中输出字符的文件输出流  
    FileWriter w = new FileWriter( "sinaean.txt" );  
  
    final int begin = 0x4e00; // 指定 UNICODE 中 汉字字符的起始位置  
    final int end = 0x9fa5; // 指定 UNICODE 中 汉字字符的终止位置  
    for( int x = begin , i = 1 ; x <= end ; x++ , i++ ){  
        w.write( x ); // 将字符写入到文件输出流中  
        w.write( "\t" ); // 将 制表符 写入到文件输出流中  
        if( i % 30 == 0 ){  
            w.write( "\n" ); // 将 换行符 写入到文件输出流中  
            w.flush(); // 将文件输出流中的内容刷出  
        }  
    }  
  
    w.close(); // 关闭文件输出流  
}
```

示例：通过 `FileWriter` 将 UNICODE 中的 汉字字符 输出到指定文件中

Class BufferedWriter

构造方法	描述
<code>public BufferedWriter(Writer out)</code>	创建一个使用默认大小输出缓冲区的缓冲字符输出流
<code>public BufferedWriter(Writer out , int size)</code>	创建一个使用给定大小输出缓冲区的新缓冲字符输出流

BufferedWriter 类内部采用 char 数组(默认大小为8192)来缓存将要被输出的字符。

BufferedWriter 将文本写入字符输出流，缓冲各个字符，从而提供单个字符、数组和字符串的高效写入

BufferedWriter 类另外增加了 newLine 方法，用来向输出缓冲区中写入一个行分隔符

Class PrintWriter

方法	描述
<code>public PrintWriter(String fileName)</code>	创建具有指定文件名称且不带自动行刷出的新字符打印流
<code>public PrintWriter(String fileName, String csn)</code>	创建具有指定文件名称和字符集且不带自动行刷出的新字符打印流
<code>public PrintWriter(File file)</code>	创建具有指定文件且不带自动行刷出的新字符打印流
<code>public PrintWriter(File file, String csn)</code>	创建具有指定文件名称和字符集且不带自动行刷出的新字符打印流
<code>public PrintWriter(OutputStream out)</code>	创建新的输出到指定底层输出流的字符打印流
<code>public PrintWriter(OutputStream out, boolean autoFlush)</code>	创建新的输出到指定底层输出流的字符打印流(并指定是否自动刷出)
<code>public PrintWriter(Writer out)</code>	创建新的输出到指定字符输出流的字符打印流
<code>public PrintWriter(Writer out, boolean autoFlush)</code>	创建新的输出到指定字符输出流的字符打印流(并指定是否自动刷出)

Part 3

Class File

Streams

Access Files

Object Serialization

New Java IO

Introduction to Access File

Sequential-Access Files

Interface DataOutput

方法	描述
<code>void write(int b)</code>	将参数 b 的八个低位写入输出流
<code>void write(byte[] b)</code>	将数组 b 中的所有字节写入输出流
<code>void write(byte[] b , int offset , int length)</code>	将数组 b 中从 offset 开始的 length 个字节按顺序写入输出流
<code>void writeBoolean(boolean v)</code>	将一个 boolean 值写入输出流
<code>void writeByte(int v)</code>	将参数 v 的八个低位写入输出流
<code>void writeShort(int v)</code>	将两个字节写入输出流，用它们表示参数值
<code>void writeChar(int v)</code>	将一个 char 值写入输出流，该值由两个字节组成
<code>void writeInt(int v)</code>	将一个 int 值写入输出流，该值由四个字节组成
<code>void writeLong(long v)</code>	将一个 long 值写入输出流，该值由八个字节组成
<code>void writeFloat(float v)</code>	将一个 float 值写入输出流，该值由四个字节组成
<code>void writeDouble(double v)</code>	将一个 double 值写入输出流，该值由八个字节组成
<code>void writeBytes(String s)</code>	将一个字符串写入输出流
<code>void writeChars(String s)</code>	将字符串 s 中的所有字符按顺序写入输出流，每个字符用两个字节表示
<code>void writeUTF(String s)</code>	将表示长度信息的两个字节写入输出流，后跟字符串 s 中每个字符的 UTF-8 修改版表示形式

DataOutput 接口用于将数据从任意 Java 基本类型转换为一系列字节，并将这些字节写入二进制流

Class DataOutputStream

```
public static void main(String[] args) throws IOException {
    // 注意，这里输出的内容，只能用 DataInputStream 按顺序读会，不能用记事本来读取
    FileOutputStream out = new FileOutputStream("binary.data");
    DataOutputStream dos = new DataOutputStream(out);

    dos.writeInt(99);
    dos.writeChar('麻');
    dos.writeChar('辣');
    dos.writeBoolean(true);
    dos.writeDouble(3.14D);
    dos.flush();

    dos.close();
    out.close();
}
```

DataOutputStream类实现了 DataOutput 接口，继承了 FilterOutputStream 类

Interface DataInput

方法	描述
<code>void readFully(byte[] bytes)</code>	从输入流中读取一些字节，并将它们存储在缓冲区数组 bytes 中
<code>void readFully(byte[] b , int offset , int length)</code>	从输入流中读取 length 个字节到 bytes 数组的 offset 处
<code>boolean readBoolean()</code>	读取一个输入字节，如果该字节不是零，则返回 true，如果是零，则返回 false
<code>byte readByte()</code>	读取并返回一个输入字节
<code>short readShort()</code>	读取两个输入字节并返回一个 short 值
<code>char readChar()</code>	读取两个输入字节并返回一个 char 值
<code>int readInt()</code>	读取四个输入字节并返回一个 int 值
<code>long readLong()</code>	读取八个输入字节并返回一个 long 值
<code>float readFloat()</code>	读取四个输入字节并返回一个 float 值
<code>double readDouble()</code>	读取八个输入字节并返回一个 double 值
<code>int skipBytes(int n)</code>	试图在输入流中跳过数据的 n 个字节，并丢弃跳过的字节
<code>void readUTF()</code>	读入一个已使用 UTF-8 修改版格式编码的字符串

DataInput 接口用于从二进制流中读取字节，并根据所有 Java 基本类型数据进行重构

Class DataInputStream

```
public static void main(String[] args) throws IOException {  
    // 这里读取前面例子中写出的那个文件(不要自己用记事本写一个文件然后读取)  
    FileInputStream in = new FileInputStream("binary.data");  
  
    DataInputStream dis = new DataInputStream(in);  
  
    // 注意，读取的顺序要跟写出时完全一致  
  
    System.out.println(dis.readInt());  
  
    System.out.println(dis.readChar());  
  
    System.out.println(dis.readChar());  
  
    System.out.println(dis.readBoolean());  
  
    System.out.println(dis.readDouble());  
  
    dis.close();  
  
    in.close();  
}
```

DataInputStream类实现了 DataInput 接口，继承了 FilterInputStream 类

Parse The wtmpx File

位置范围	字节长度	含义
000-031	32	/* 用户登录名 */
032-035	4	/* inittab id */
036-067	32	/* device name (console, lnxx) */
068-071	4	/* 进程ID*/
072-073	2	/* 登录类型7-登入，8登出 */
074-075	2	/* process termination*/
076-077	2	/* exit status*/
	2	/* 这是C数据类型补齐产生的空位*/
080-083	4	/* 登录时刻*/ /*单位是秒 */
084-087	4	/* 登录时刻中的毫秒部分(单位是毫秒) */
088-091	4	/* session ID, used for windowing */
092-111	20	/* reserved for future use */
112-113	2	/* significant length of ut_host */
114-371	257	/* 登录IP*/

wtmpx 文件是 Unix 系统中记录每个用户登入、登出等数据的日志文件
该文件中 每 372 个字节表格一个用户的记录，每 372 个字节的含义如上表示

Parse The wtmpx File

数据名	数据含义	是否需要采集	备注说明
logname	用户登录名	是	匹配同一次登录会话的必须数据之一
pid	进程ID	是	匹配同一次登录会话的必须数据之二
type	登录类型7-登入，8登出	是	type的值在1-8之间，但只处理7与8两种情况
logtime	登录时刻*/*单位是秒	是	要采集的数据，logtime是登入或登出时刻
logip	登录IP	是	要采集的数据(没有IP的数据也可以剔除)

解析 Unix 系统日志文件 wtmpx 文件，把得到的内容输出到文本文件中

登录记录总数计算公式为：登录记录总数 = 日志文件大小 / 372

对于登入/登出时刻，不要一次读取8个字节，应先读秒，后读毫秒，之后再算总毫秒

Random-Access Files

Class RandomAccessFile

```
public class RandomAccessFile implements DataOutput, DataInput, Closeable {  
  
    /** 此处省略 RandomAccessFile 类中的所有代码 **/  
  
}
```

RandomAccessFile 类的实例支持对随机访问文件的读取和写入
随机访问文件的行为类似存储在文件系统中的一个大型 byte 数组
存在指向该隐含数组的光标或索引，称为[文件指针](#)，可以通过getFilePointer()来获取它或用seek()来设置它
输入操作从文件指针开始读取字节，并随着对字节的读取而前移此文件指针
如果是读写模式，也可以从文件指针开始写入字节，并随着对字节的写入而前移此文件指针

Class RandomAccessFile

构造方法	描述
<code>public RandomAccessFile(File file , String mode)</code>	创建从中读取和向其中写入(可选)的随机访问文件流，该文件由 File 参数指定
<code>public RandomAccessFile(String name , String mode)</code>	创建从中读取和向其中写入(可选)的随机访问文件流，该文件具有指定名称

"r"	以只读方式打开。调用结果对象的任何 read 方法都将导致抛出 IOException
"rw"	打开以便读取和写入。如果该文件尚不存在，则尝试创建该文件
"rws"	打开以便读取和写入，对于 "rw"，还要求对文件的内容或元数据的每个更新都同步写入到底层存储设备
"rwd"	打开以便读取和写入，对于 "rw"，还要求对文件内容的每个更新都同步写入到底层存储设备

Class RandomAccessFile

```
public static void main(String[] args) throws IOException {
    RandomAccessFile raf = new RandomAccessFile("my.data", "rw");
    final byte[] usernameBytes = new byte[32];
    final byte[] passwordBytes = new byte[32];
    String name = "zhangsanfeng";
    String password = "wudang2017";
    byte[] nb = name.getBytes();
    byte[] pb = password.getBytes();
    System.arraycopy(nb, 0, usernameBytes, 0, nb.length);
    System.arraycopy(pb, 0, passwordBytes, 0, pb.length);
    raf.write(usernameBytes); // 向文件中写入 32 个字节
    raf.write(passwordBytes); // 向文件中写入 32 个字节
    char gender = '男';
    raf.writeChar(gender); // 向文件中写入 2 个字节
    int age = 108;
    raf.writeInt(age); // 向文件中写入 4 个字节
    // 获取并输出文件指针当前指向的位置
    System.out.println("file pointer : " + raf.getFilePointer());
    raf.seek(0); // 将文件指针定位到 0
    Arrays.fill(usernameBytes, (byte)0); // 将数组填充为初始状态
    raf.readFully(usernameBytes);
    // 因为字节数组中可能含有无效字节，因此在构建字符串后需要 trim
    System.out.println(new String(usernameBytes).trim());
    Arrays.fill(passwordBytes, (byte)0); // 将数组填充为初始状态
    raf.readFully(passwordBytes);
    System.out.println(new String(passwordBytes).trim());
    gender = raf.readChar();
    System.out.println(gender);
    age = raf.readInt();
    System.out.println(age);
    raf.close();
}
```

Class RandomAccessFile

数据名	数据含义	是否需要采集	备注说明
logname	用户登录名	是	匹配同一次登录会话的必须数据之一
pid	进程ID	是	匹配同一次登录会话的必须数据之二
type	登录类型7-登入，8登出	是	type的值在1-8之间，但只处理7与8两种情况
logtime	登录时刻*/*单位是秒	是	要采集的数据，logtime是登入或登出时刻
logip	登录IP	是	要采集的数据(没有IP的数据也可以剔除)

尝试用 RandomAccessFile 的方式解析 wtmpx 文件并获取上方表格中的数据

将解析后得到的内容，以字符串形式，打印 (Printstream) 到一个单独的文件中

Part 4

Class File

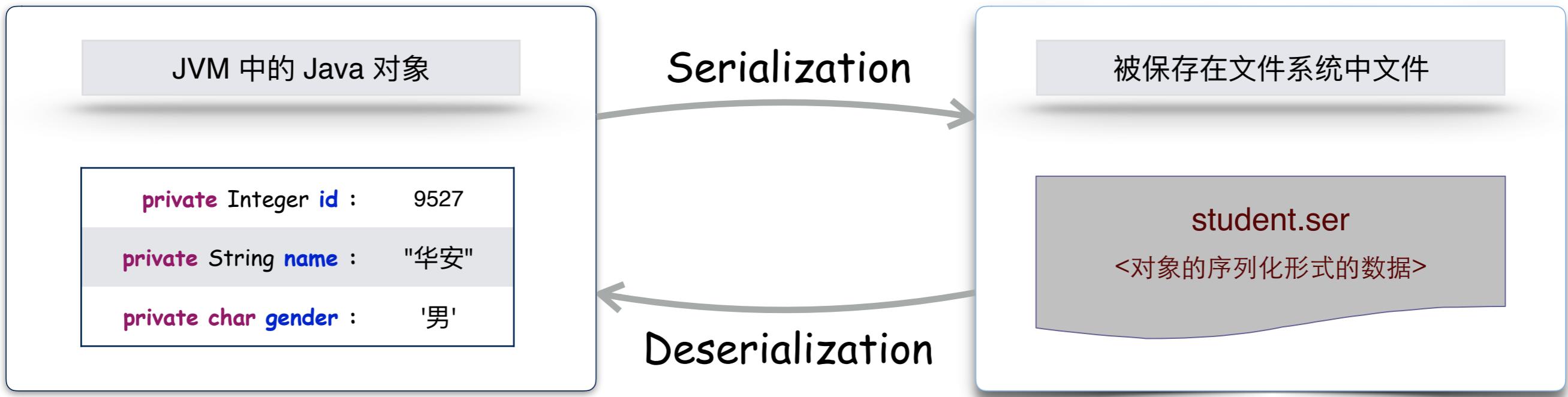
Streams

Access Files

Object Serialization

New Java IO

Introduction to Serialization and Deserialization



将 JVM 中的 Java 对象转换为字节序列的过程称为对象的序列化(Serialization)

把字节序列恢复为 JVM 中的 Java 对象的过程称为对象的反序列化(Deserialization)

在 Java 中可以通过实现 Serializable 或 Externalizable 接口来支持序列化操作，也可以自定义序列化

Interface Serializable

```
package java.io;

public interface Serializable {
    /** 序列化接口没有方法或字段，仅用于标识可序列化的语义 */
}
```

```
public class Student implements Serializable {

    private static final long serialVersionUID = 2010482733477708163L;
}
```

类通过实现 `java.io.Serializable` 接口以启用其序列化功能，未实现此接口的类将无法使其任何状态序列化或反序列化。实现 `Serializable` 接口通常需要添加一个 `serialVersionUID` 属性，每个类的该属性取值应尽量与其它类的该属性值不同。

Interface ObjectOutputStream

方法	描述
<code>public void write(int b)</code>	写入单个字节
<code>public void write(byte[] bytes)</code>	写入 bytes 数组
<code>public void write(byte[] bytes, int offset, int length)</code>	写入 bytes 数组的 length 个字节 (从数组的 offset 位置开始)
<code>public void writeObject(Object o)</code>	将对象写入底层存储或流
<code>public void flush()</code>	刷出该流的缓冲
<code>public void close()</code>	关闭该流

ObjectOutput 扩展 DataOutput 接口以包含对象的写入操作

DataOutput 包括基本类型的输出方法，ObjectOutput 扩展了该接口，以包含对象、数组和 String 的输出方法

Class ObjectOutputStream

构造方法	描述
<code>protected ObjectOutputStream()</code>	为完全重新实现 ObjectOutputStream 的子类提供一种方法,让它不必分配仅由 ObjectOutputStream 的实现使用的私有数据
<code>public ObjectOutputStream(OutputStream out)</code>	创建写入指定 OutputStream 的 ObjectOutputStream 实例

Class ObjectOutputStream

```
public static void main(String[] args) throws IOException {  
  
    OutputStream out = new FileOutputStream( "student.ser" );  
    ObjectOutputStream oos = new ObjectOutputStream( out );  
  
    Student s = new Student();  
    s.setId( 9527 );  
    s.setName( "华安" );  
    s.setGender( '男' );  
    s.setBirthdate( new Date() );  
  
    oos.writeObject( s );  
  
    oos.close();  
    out.close();  
}
```

```
public class Student implements Serializable {  
  
    private static final long serialVersionUID = 6708635483501840254L;  
  
    private Integer id;  
    private String name;  
    private char gender;  
    private transient Date birthdate;  
  
    /** 此处省略了 getter 和 setter */  
}
```

若 Student 类没有实现 Serializable 接口，则对 Student 对象执行序列化操作将抛出 NotSerializableException 被 transient 修饰的属性将不会被序列化，对于不需要序列化和反序列化的属性，可以使用 transient 关键字修饰

Interface ObjectInput

方法	描述
<code>public int available()</code>	返回可以无阻塞地读取的字节数
<code>public int read()</code>	读取数据字节
<code>public int read(byte[] bytes)</code>	读入 byte 数组
<code>public int read(byte[] bytes, int offset, int length)</code>	读入 byte 数组 (从 offset 位置开始放入)
<code>public Object readObject()</code>	读取并返回对象
<code>public long skip(long n)</code>	跳过输入的 n 个字节
<code>public void close()</code>	关闭输入流

ObjectInput 扩展 DataInput 接口以包含对象的读操作

DataInput 包括基本类型的输入方法，ObjectInput 扩展了该接口，以包含对象、数组和 String 的输出方法

Class ObjectInputStream

构造方法	描述
<code>protected ObjectInputStream()</code>	为完全重新实现 ObjectInputStream 的子类提供一种方式，让它不必分配仅由 ObjectInputStream 的实现使用的私有数据
<code>public ObjectInputStream(InputStream in)</code>	创建从指定 InputStream 读取的 ObjectInputStream 实例

Class ObjectOutputStream

```
public static void main(String[] args) throws IOException, ClassNotFoundException {  
    InputStream in = new FileInputStream("student.ser");  
  
    ObjectInputStream ois = new ObjectInputStream(in);  
  
    Object o = ois.readObject();  
  
    if( o instanceof Student ){  
        Student s = (Student) o;  
        System.out.println( s.getId() + " , " + s.getName() + " , " + s.getGender() );  
    }  
  
    ois.close();  
    in.close();  
}
```

反序列化时，Student 类的serialVersionUID属性值应该跟序列化时保持一致，否则亦将抛出 NotSerializableException

Matters needing attention

父类实现序列化，子类自动实现，无需再显示实现 Serializable

对于不可序列化的类，必须有一个无参数构造方法，以便允许初始化其属性

不可序列化的类的子类是可序列化时，由子类负责保存和恢复不可序列化父类的状态

同时该父类的字段需要是可访问的(被public或protected修饰 或者存在相应的get和set方法)

某类对象的实例变量引用其他对象，序列化该对象时，被引用的对象也将被序列化

但是，序列化操作不写出没有实现 java.io.Serializable 接口的任何对象的字段

并不是所有属性都要被序列化：

安全方面的原因：被 private 修饰的 属性 在序列化后，可能不再受保护

资源分配的原因：对于 Socket 和 Thread，如果可以序列化，那将无法为其分配资源

不需要序列化的属性可以通过 transient 关键字来修饰

Part 5

Class File

Streams

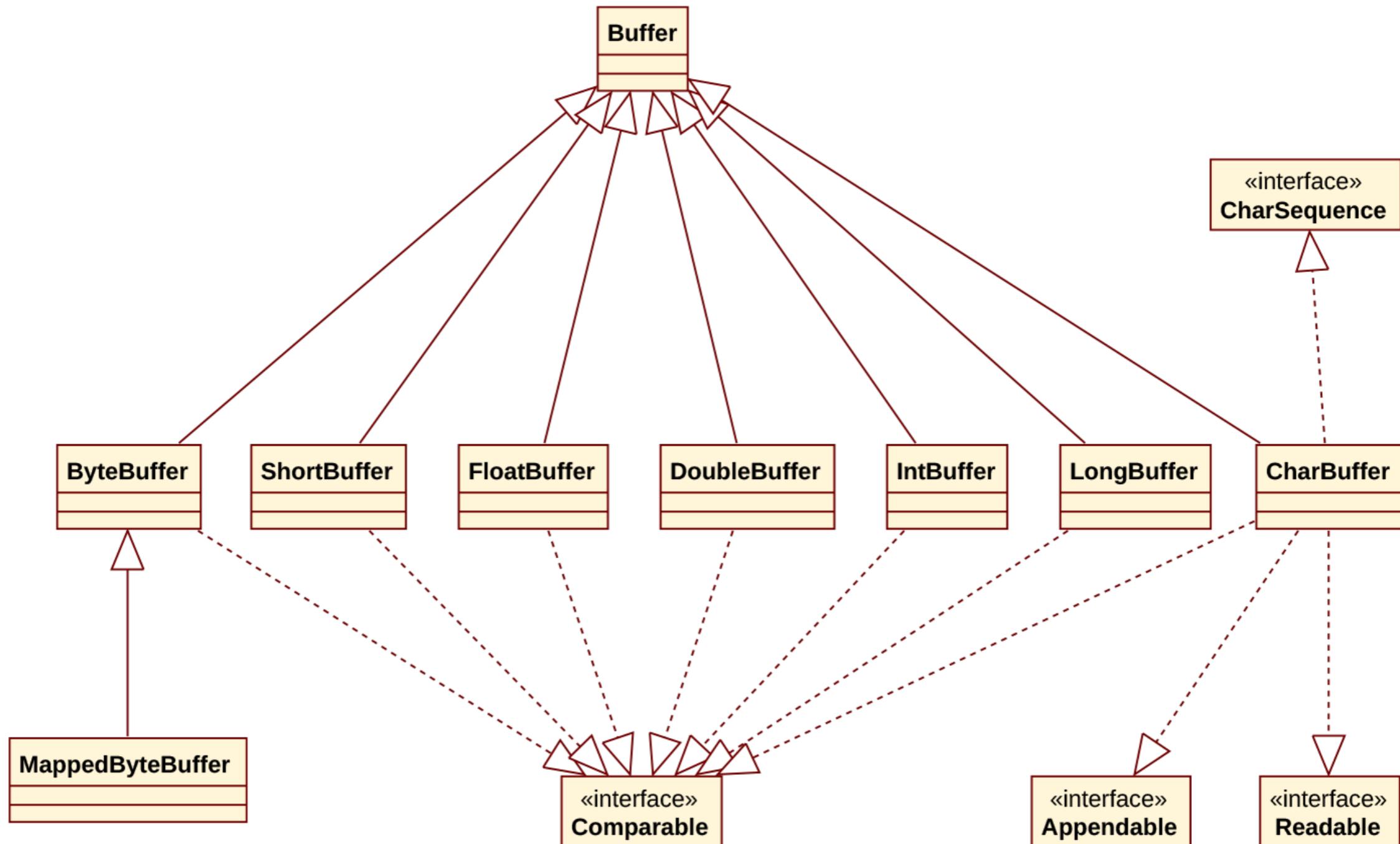
Access Files

Object Serialization

New Java IO

Introduction to New IO

Introduction to Buffer



Class Buffer

```
package java.nio;

public abstract class Buffer {
    .....
    private int mark = -1 ;
    private int position = 0 ;
    private int limit ;
    private int capacity ;
    .....
}
```

Class Buffer

方法	描述
<code>public abstract boolean isReadOnly()</code>	
<code>public abstract boolean isDirect()</code>	
<code>public abstract boolean hasArray()</code>	
<code>public abstract Object array()</code>	
<code>public abstract int arrayOffset()</code>	
<code>public final int capacity()</code>	
<code>public final int position()</code>	
<code>public final Buffer position(int newPosition)</code>	
<code>public final int limit()</code>	
<code>public final Buffer limit(int newLimit)</code>	
<code>public final boolean hasRemaining()</code>	
<code>public final int remaining()</code>	

Class Buffer

方法	描述
<code>public final Buffer mark()</code>	
<code>public final Buffer reset()</code>	
<code>public final Buffer flip()</code>	
<code>public final Buffer rewind()</code>	
<code>public final Buffer clear()</code>	

Class CharBuffer

```
package java.nio;

public abstract class CharBuffer extends Buffer implements Comparable<CharBuffer>, Appendable, CharSequence, Readable {
    .....
    final char[] hb; // Non-null only for heap buffers
    final int offset;
    boolean isReadOnly; // Valid only for heap buffers
    .....
}
```

Class CharBuffer

方法	描述
<code>public static CharBuffer allocate(int capacity)</code>	
<code>public static CharBuffer wrap(char[] array)</code>	
<code>public static CharBuffer wrap(char[] array, int offset, int length)</code>	
<code>public static CharBuffer wrap(CharSequence csq)</code>	
<code>public static CharBuffer wrap(CharSequence csq, int start, int end)</code>	
<code>public final char[] array()</code>	
<code>public CharBuffer append(char c)</code>	
<code>public CharBuffer append(CharSequence csq)</code>	
<code>public CharBuffer append(CharSequence csq, int start, int end)</code>	
<code>public abstract CharBuffer duplicate()</code>	
<code>public abstract CharBuffer slice()</code>	
<code>public final int length()</code>	

Class CharBuffer

方法	描述
<code>public abstract CharBuffer put(char c)</code>	
<code>public final CharBuffer put(char[] src)</code>	
<code>public CharBuffer put(char[] src, int offset, int length)</code>	
<code>public final CharBuffer put(String src)</code>	
<code>public CharBuffer put(String src, int start, int end)</code>	
<code>public CharBuffer put(CharBuffer src)</code>	
<code>public abstract CharBuffer put(int index, char c)</code>	
<code>public abstract char get()</code>	
<code>public CharBuffer get(char[] dst)</code>	
<code>public CharBuffer get(char[] dst, int offset, int length)</code>	
<code>public abstract char get(int index)</code>	
<code>public int read(CharBuffer target)</code>	

Class CharBuffer

Class ByteBuffer

```
package java.nio;

public abstract class ByteBuffer extends Buffer implements Comparable<ByteBuffer> {
    .....
    final byte[] hb; // Non-null only for heap buffers
    final int offset;
    boolean isReadOnly; // Valid only for heap buffers
    .....
}
```

Class ByteBuffer

方法	描述
<code>public static ByteBuffer allocate(int capacity)</code>	
<code>public static ByteBuffer allocateDirect(int capacity)</code>	
<code>public static ByteBuffer wrap(byte[] array)</code>	
<code>public static ByteBuffer wrap(byte[] array, int offset, int length)</code>	
<code>public abstract ByteBuffer put(byte b)</code>	
<code>public final ByteBuffer put(byte[] src)</code>	
<code>public ByteBuffer put(byte[] src, int offset, int length)</code>	
<code>public ByteBuffer put(ByteBuffer src)</code>	
<code>public abstract ByteBuffer put(int index, byte b)</code>	

Class ByteBuffer

方法	描述
<code>public abstract ByteBuffer putShort(short value)</code>	
<code>public abstract ByteBuffer putShort(int index, short value)</code>	
<code>public abstract ByteBuffer putChar(char value)</code>	
<code>public abstract ByteBuffer putChar(int index, char value)</code>	
<code>public abstract ByteBuffer.putInt(int value)</code>	
<code>public abstract ByteBuffer.putInt(int index, int value)</code>	
<code>public abstract ByteBuffer.putLong(long value)</code>	
<code>public abstract ByteBuffer.putLong(int index, long value)</code>	
<code>public abstract ByteBuffer.putFloat(float value)</code>	
<code>public abstract ByteBuffer.putFloat(int index, float value)</code>	
<code>public abstract ByteBuffer.putDouble(double value)</code>	
<code>public abstract ByteBuffer.putDouble(int index, double value)</code>	

Class ByteBuffer

方法	描述
<code>public abstract byte get()</code>	
<code>public ByteBuffer get(byte[] dst)</code>	
<code>public ByteBuffer get(byte[] dst, int offset, int length)</code>	
<code>public abstract byte get(int index)</code>	
<code>public abstract ShortBuffer asShortBuffer()</code>	
<code>public abstract CharBuffer asCharBuffer()</code>	
<code>public abstract IntBuffer asIntBuffer()</code>	
<code>public abstract LongBuffer asLongBuffer()</code>	
<code>public abstract FloatBuffer asFloatBuffer()</code>	
<code>public abstract DoubleBuffer asDoubleBuffer()</code>	
<code>public abstract ByteBuffer asReadOnlyBuffer()</code>	
<code>public final byte[] array()</code>	

Class ByteBuffer

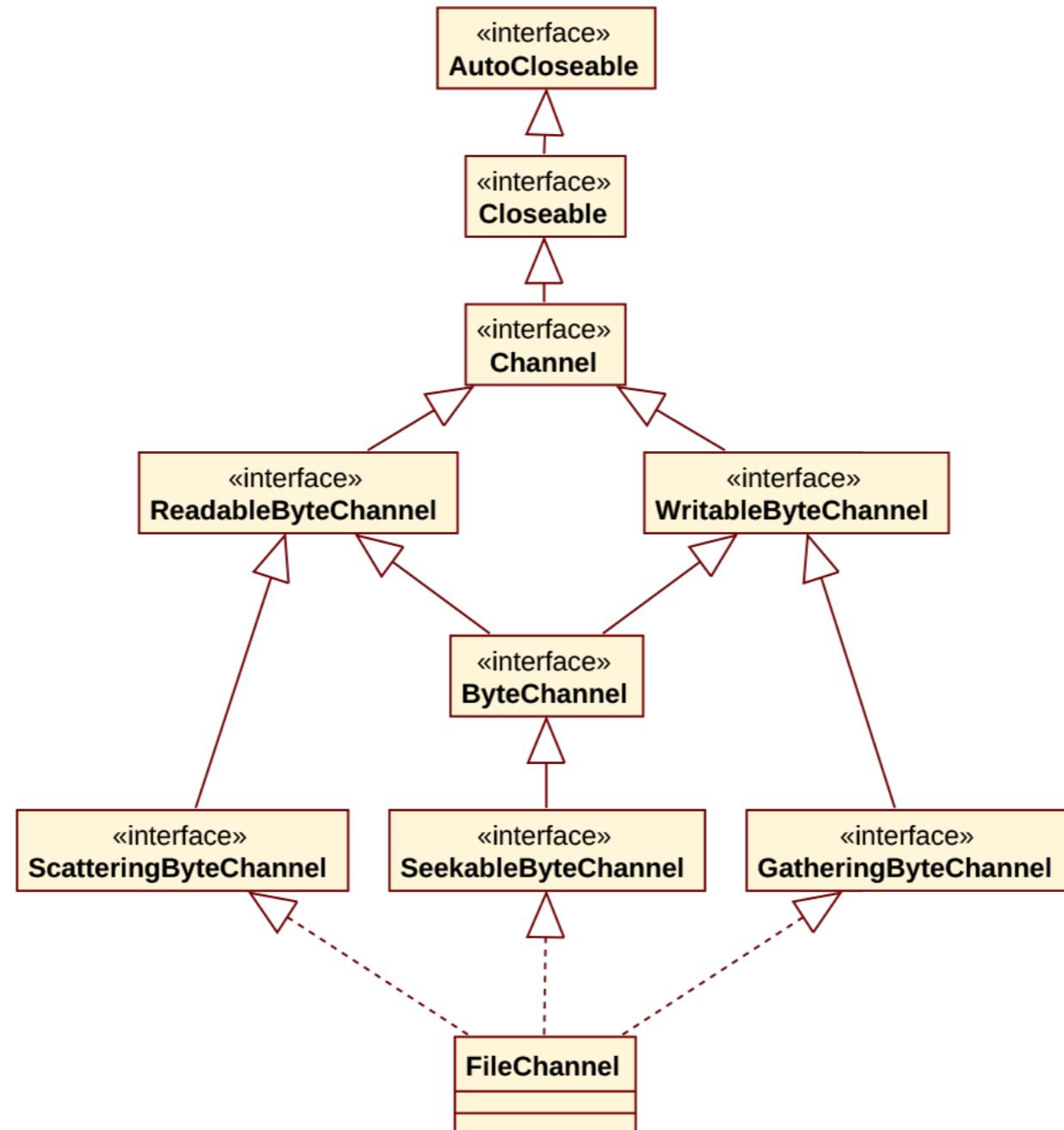
方法	描述
<code>public abstract short getShort()</code>	
<code>public abstract short getShort(int index)</code>	
<code>public abstract char getChar()</code>	
<code>public abstract char getChar(int index)</code>	
<code>public abstract int getInt()</code>	
<code>public abstract int getInt(int index)</code>	
<code>public abstract long getLong()</code>	
<code>public abstract long getLong(int index)</code>	
<code>public abstract float getFloat()</code>	
<code>public abstract float getFloat(int index)</code>	
<code>public abstract double getDouble()</code>	
<code>public abstract double getDouble(int index)</code>	

Class ByteBuffer

Introduction to Character Encoding

Class Charset

Introduction to Channel



Interface Path

Class Paths

Class Files

Interface BasicFileAttributes

方法	描述
<code>public FileTime creationTime()</code>	
<code>public FileTime lastAccessTime()</code>	
<code>public FileTime lastModifiedTime()</code>	
<code>public boolean isDirectory()</code>	
<code>public boolean isRegularFile()</code>	
<code>public boolean isSymbolicLink()</code>	
<code>public boolean isOther()</code>	
<code>public long size()</code>	
<code>public Object fileKey()</code>	

Interface BasicFileAttributes

```
public static void main(String[] args) throws IOException {
    Path path = Paths.get( "/Users/malajava/TheFifthElement.mp4" );
    BasicFileAttributes attrs = Files.readAttributes( path , BasicFileAttributes.class );

    FileTime creation = attrs.creationTime();
    System.out.println( "创建时间: " + creation );

    FileTime lastModified = attrs.lastModifiedTime();
    System.out.println( "最后一次修改文件的时间: " + lastModified );

    FileTime lastAccess = attrs.lastAccessTime();
    System.out.println( "最后一次访问文件的时间: " + lastAccess );

    System.out.println( "文件大小: " + attrs.size() );
    System.out.println( "是否是目录: " + attrs.isDirectory() );
    System.out.println( "是否是文件: " + attrs.isRegularFile() );
    System.out.println( "是否是符号链接: " + attrs.isSymbolicLink() );
    // 对于 fileKey 而言 Unix/Linux/OS X 下输出的可能是 (dev=1000004,ino=18097051) , 而 Windows 下可能是 null
    System.out.println( "文件惟一标识符: " + attrs.fileKey() );
}
```



感谢学习

www.oracleacademy.net

Produced by oracle.han@foxmail.com