Java IO

IO流分类

- 按照“流”的数据流向，可以将其划分为：**输入流**和**输出流**
- 按照“流”中处理数据的单位，可将其划分为：**字节流**和**字符流**
    - java中的字节是有符号类型，而字符是无符号类型

字节流的抽象基类：InputStream/OutputStream

字符流的抽象基类：Reader/Writer

这四个类派生出来的子类名称都是以其为父类名作为子类名的后缀，如InputStream的子类FileInputStream,Reader的子类FileReader

---

字符流

1.1 Writer:字符输出流

- BufferedWriter
- ChatArrayWriter
- FileWriter
- OutputStreamWriter
    - FileWriter
- PrintWriter
- StringWriter

主要方法	含义
Writer append(char c)	将指定的字符附加到此作者
Writer append(CharSequence csq)	将指定的字符序列附加到此作者
Writer append(CharSequence csq, int start, int end)	将指定字符序列的子序列附加到此作者
abstract void close()	关闭流，先刷新
abstract void flush()	刷新流
void write(char[] cbuf)	写入一个字符数组
abstract void write(char[] cbuf, int off, int len)	写入字符数组的一部分
void write(int c)	写一个字符
void write(String str, int off, int len)	写一个字符串的一部分

1.2 FileWriter

构造方法	含义
OutputStreamWriter(OutputStream out)	使用默认字符编码
OutputStreamWriter(OutputStream out, String charsetName)	使用命名字符集
OutputStreamWriter(OutputStream out, Charset cs)	使用给定字符集
OutputStreamWriter(OutputStream out, CharsetEncoder enc)	使用给定字符集编码器

方法列表	含义
Writer append(CharSequence csq)	将指定的字符序列附加到此作者
Writer append(CharSequence csq, int start, int end)	将指定字符序列的子序列附加到此作者
void close()	关闭流，先刷新
void flush()	刷新流
String getEncoding()	返回此流使用的字符编码的名称
void write(char[] cbuf, int off, int len)	写入字符数组的一部分
void write(int c)	写一个字符
void write(String str, int off, int len)	写一个字符串的一部分

代码实例

```
import java.io.*;
public class  FileWriterDemo
{
    public static void main(String[] args) throws IOException
    {
        //创建一个FileWriter对象。该对象一被初始化就必须要明确被操作的文件。
        //而且该文件会被创建到指定目录下。如果该目录下已有同名文件，将被覆盖。
        //其实该步就是在明确数据要存放的目的地。
        FileWriter fw = new FileWriter("demo.txt");

        //调用write方法，将字符串写入到流中。
        fw.write("abcde");

        //关闭流资源，但是关闭之前会刷新一次内部的缓冲中的数据。
        //将数据刷到目的地中。
        //和flush区别：flush刷新后，流可以继续使用，close刷新后，会将流关闭。
        fw.close();
    }
}
```

```
public class FileWriterDemo1 {
    public static void main(String[] args) throws IOException
    {

        //传递一个true参数，代表不覆盖已有的文件。并在已有文件的末尾处进行数据续写。
        FileWriter fw = new FileWriter("demo.txt",true);

        fw.write("nihao\r\nxiexie");

        fw.close();
    }
}
```

1.3 BufferedWriter

BufferedWriter 是缓冲字符输出流。它继承于Writer。
BufferedWriter 的作用是为其他字符输出流添加一些缓冲功能，使用BufferedWriter可以提高我们写入文件的效率。

构造方法	含义
BufferedWriter(Writer out)	创建使用默认大小的输出缓冲区的缓冲字符输出流
BufferedWriter(Writer out, int sz)	创建一个新的缓冲字符输出流，使用给定大小的输出缓冲区

方法列表	含义
void close()	关闭流，先刷新
void flush()	刷新流
void newLine()	写一行行分隔符
void write(char[] cbuf, int off,int len)	写入字符数组的一部分
void write(int c)	写一个字符
void write(String s, int off, int len)	写一个字符串的一部分

代码实例

```
import java.io.*;

public class BufferWriterDemo {
    public static void main(String[] args) throws IOException {
        //创建一个字符写入流对象。
        FileWriter fw = new FileWriter("buf.txt");
        //为了提高字符写入流效率。加入了缓冲技术。
```

```
        //只要将需要被提高效率的流对象作为参数传递给缓冲区的构造函数即可。
        BufferedWriter bw=new BufferedWriter(fw);
        char[] c={'a','b','c','d','e'};
        bw.write(c,0,4);

        //换行
        bw.newLine();

        //再次写入
        bw.write(c,2,2);

        //刷新流
        bw.flush();
        //其实关闭缓冲区，就是在关闭缓冲区中的流对象。
        bw.close();
    }
}
```

1.4 CharArrayReader

CharArrayReader 用于写入字符，它继承于Writer。操作的数据是以字符为单位。

构造函数	含义
CharArrayWriter()	新建CharArrayWriter
CharArrayWriter(int initialSize)	用指定的初始大小创建一个新的CharArrayWriter

方法列表	含义
CharArrayWriter append(char c)	将指定的字符附加到此作者
CharArrayWriter append(CharSequence csq)	将指定的字符序列附加到此作者
CharArrayWriter append(CharSequence csq, int start, int end)	将指定字符序列的子序列附加到此作者
char[] toCharArray()	返回输入数据的副本
void writeTo(Writer out)	将缓冲区的内容写入另一个字符流
void reset()	重置缓冲区以便您可以再次使用它，而不丢弃已经分配的缓冲区

1.5 FilterWriter

FilterWriter是字符类型的过滤输出流。

构造方法`protected FilterWriter(Writer out)`创建一个新的过滤的作者

1.6 PrintWriter

PrintWriter 是字符类型的打印输出流，它继承于Writer。

构造方法	含义
PrintWriter(File file)	使用指定的文件创建一个新的PrintWriter，而不需要自动的线路刷新
PrintWriter(File file, String csn)	使用指定的文件和字符集创建一个新的PrintWriter，而不需要自动进行线条刷新
PrintWriter(OutputStream out)	从现有的OutputStream创建一个新的PrintWriter，而不需要自动线路刷新
PrintWriter(OutputStream out, boolean autoFlush)	从现有的OutputStream创建一个新的PrintWriter
PrintWriter(Writer out)	创建一个新的PrintWriter，没有自动线冲洗
PrintWriter(Writer out, boolean autoFlush)	创建一个新的PrintWriter
PrintWriter(String fileName)	使用指定的文件名创建一个新的PrintWriter，而不需要自动执行行刷新
PrintWriter(String fileName, String csn)	使用指定的文件名和字符集创建一个新的PrintWriter，而不需要自动线路刷新

1.7 Reader:字符输入流

- CharArrayReader
- BufferedReader:LineNumberReader
- FilterReader
- InputStreamReader
- InputSteamReader:FileReader
- String

有输出流那么当然就有输入流，Reader是字符输入流的基类，Reader的方法列表如下：

```
abstract` `void` `close() 关闭流并释放与之相关联的任何系统资源。
void` `mark(``int` `readAheadLimit) 标记流中的当前位置。
boolean` `markSupported() 告诉这个流是否支持mark（）操作。
int` `read() 读一个字符
int` `read(``char``[] cbuf) 将字符读入数组。
abstract` `int` `read(``char``[] cbuf, ``int` `off, ``int` `len) 将字符读入数组的一部分。
int` `read(CharBuffer target) 尝试将字符读入指定的字符缓冲区。
boolean` `ready() 告诉这个流是否准备好被读取。
void` `reset() 重置流。
long` `skip(``long` `n) 跳过字符
```

1.8 FileReader

构造方法：

```
FileReader(File file) 创建一个新的 FileReader ，给出 File读取。
FileReader(FileDescriptor fd) 创建一个新的 FileReader ，给予 FileDescriptor从中读取。
FileReader(String fileName) 创建一个新的 FileReader ，给定要读取的文件的名称。
```

代码实例：

```
import java.io.*;

public class FileReaderDemo {
    public static void main(String[] args) throws IOException
    {
        //创建一个文件读取流对象，和指定名称的文件相关联。
        //要保证该文件是已经存在的，如果不存在，会发生异常FileNotFoundException
        FileReader fr = new FileReader("demo.txt");

        int ch = 0;

        while((ch=fr.read())!=-1)
        {
            System.out.println("ch="+(char)ch);
        }

        //定义一个字符数组。用于存储读到字符。
        //该read(char[])返回的是读到字符个数。
        char[] buf = new char[1024];

        int num = 0;
        while((num=fr.read(buf))!=-1)
        {
            System.out.println(new String(buf,0,num));
```

```
        }

        //关闭流
        fr.close();
    }
}
```

1.9 BufferedReader

构造方法

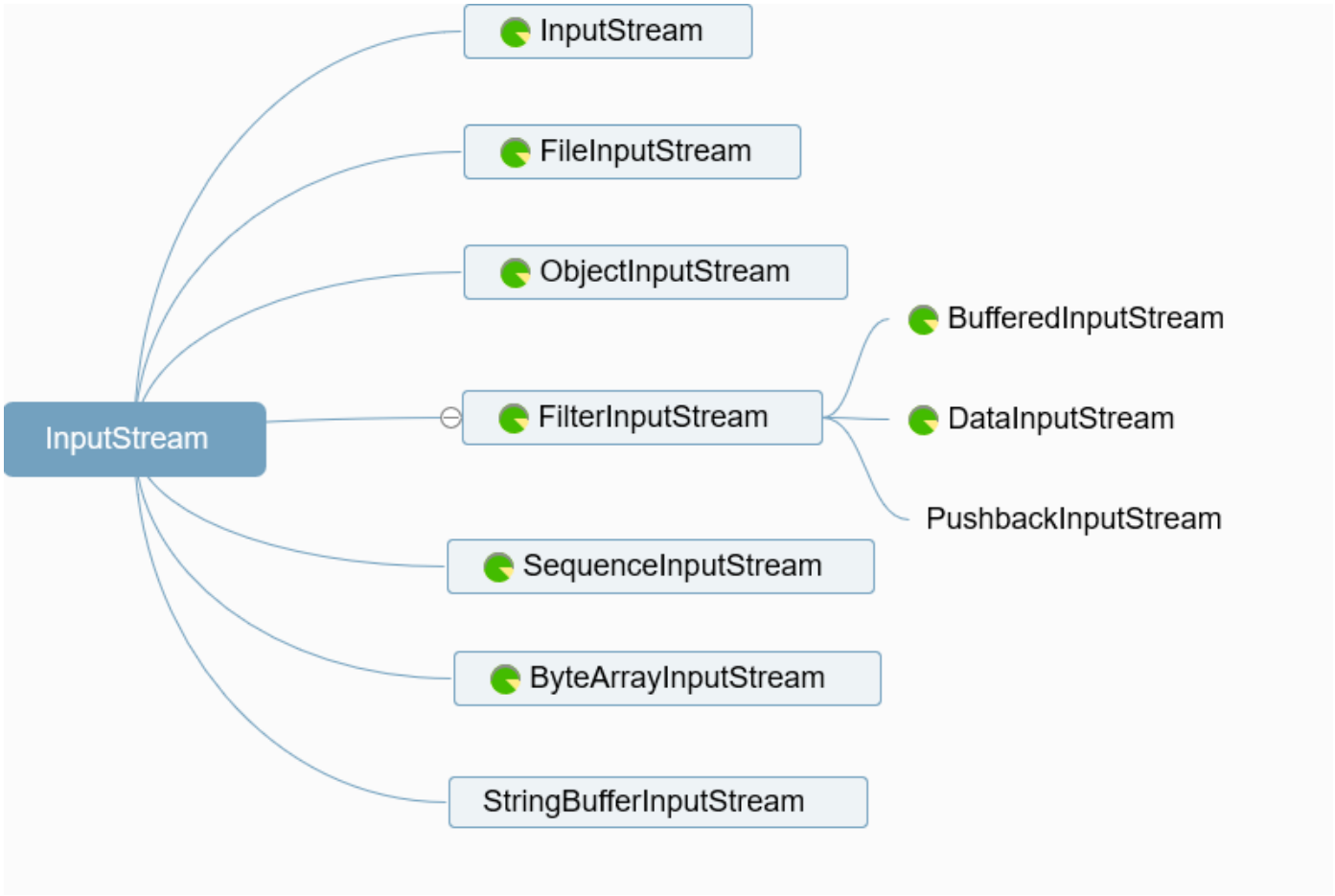
```
void` `close() 关闭流并释放与之相关联的任何系统资源。
Stream<String> lines() 返回一个 Stream ，其元素是从这个 BufferedReader读取的行。
void` `mark(``int` `readAheadLimit) 标记流中的当前位置。
boolean` `markSupported() 告诉这个流是否支持mark（）操作。
int` `read() 读一个字符
int` `read(``char``[] cbuf, ``int` `off, ``int` `len) 将字符读入数组的一部分。
String readLine() 读一行文字。
boolean` `ready() 告诉这个流是否准备好被读取。
void` `reset() 将流重置为最近的标记。
long` `skip(``long` `n) 跳过字符
```

2. 字节流

概述：字节流的基本操作和字符流类相同，但它不仅可以操作字符，还可以操作其他媒体文件。

2.1 InputStream字节输入流

概述：InputStream类是字节输入流的抽象类，是所有字节输入流的父类，InputStream类具有层次结构如下图所示：



常用方法：

```
int` `available() 从下一次调用此输入流的方法返回可从该输入流读取（或跳过）的字节数，而不会阻塞。
void` `close() 关闭此输入流并释放与流相关联的任何系统资源。
void` `mark(``int` `readlimit) 标记此输入流中的当前位置。
boolean` `markSupported() 测试此输入流是否支持 mark和 reset方法。
abstract` `int` `read() 从输入流读取数据的下一个字节。
int` `read(``byte``[] b) 从输入流中读取一些字节数，并将它们存储到缓冲器阵列 b 。
int` `read(``byte``[] b, ``int` `off, ``int` `len) 从输入流读取最多 len个字节的数据到字节数组。
byte``[] readAllBytes() 从输入流读取所有剩余字节。
int` `readNBytes(``byte``[] b, ``int` `off, ``int` `len) 将所请求的字节数从输入流读入给定的字节数组。
void` `reset() 将此流重新定位到最后在此输入流上调用 mark方法时的位置。
long` `skip(``long` `n) 跳过并丢弃来自此输入流的 n字节的数据。
long` `transferTo(OutputStream out) 从该输入流中读取所有字节，并按读取的顺序将字节写入给定的输出流。
```

2.2 FileInputStream

构造方法：

```
FileInputStream(File file) 通过打开与实际文件的连接来创建一个 FileInputStream ，该文件由文件系统中的 File对象 file命名。
FileInputStream(FileDescriptor fdObj) 通过使用文件描述符 fdObj创建 FileInputStream ，该文件描述符表示与文件系统中的实际文件的现有连接。
FileInputStream(String name) 通过打开与实际文件的连接来创建一个 FileInputStream ，该文件由文件系统中的路径名 name命名。
```

方法列表：

```
int` `available() 返回从此输入流中可以读取（或跳过）的剩余字节数的估计值，而不会被下一次调用此输入流的方法阻塞。
void` `close() 关闭此文件输入流并释放与流相关联的任何系统资源。
protected` `void` `finalize() 已过时。
finalize方法已被弃用。 为了执行清理而覆盖finalize子类应该修改为使用替代清理机制，并删除覆盖的finalize方法。 当覆盖finalize方法时，其实现必须明确确保按照``super``.finalize()所述调用``super``.finalize() 。 有关迁移选项的更多信息，请参阅Object.finalize()的规范。
FileChannel getChannel() 返回与此文件输入流相关联的唯一的FileChannel对象。
FileDescriptor getFD() 返回表示与此 FileInputStream正在使用的文件系统中的实际文件的连接的FileDescriptor对象。
int` `read() 从该输入流读取一个字节的数据。
int` `read(``byte``[] b) 从该输入流读取最多 b.length个字节的数据到一个字节数组。
int` `read(``byte``[] b, ``int` `off, ``int` `len) 从该输入流读取最多 len个字节的数据到字节数组。
long` `skip(``long` `n) 跳过并从输入流中丢弃 n字节的数据。
```

2.3 OutputStream字节输出流

- ByteArrayOutputStream
- FileOutputStream
- FilterOutputStream
  - BufferedOutputStream
  - DataOutputStream
- ObjectOutputStream
- OutputStream

方法列表：

```
void` `close() 关闭此输出流并释放与此流相关联的任何系统资源。
void` `flush() 刷新此输出流并强制任何缓冲的输出字节被写出。
void` `write(``byte``[] b) 将 b.length字节从指定的字节数组写入此输出流。
void` `write(``byte``[] b, ``int` `off, ``int` `len) 从指定的字节数组写入 len字节，从偏移量 off开始输出到此输出流。
abstract` `void` `write(``int` `b) 将指定的字节写入此输出流。
```

2.4 FileOutPutStream

构造方法：

```
FileOutputStream(File file) 创建文件输出流以写入由指定的 File对象表示的文件。
FileOutputStream(FileDescriptor fdObj) 创建文件输出流以写入指定的文件描述符，表示与文件系统中实际文件的现有连接。
FileOutputStream(File file, ``boolean` `append) 创建文件输出流以写入由指定的 File对象表示的文件。
FileOutputStream(String name) 创建文件输出流以指定的名称写入文件。
FileOutputStream(String name, ``boolean` `append) 创建文件输出流以指定的名称写入文件。
```

方法列表：

```
void` `close() 关闭此文件输出流并释放与此流相关联的任何系统资源。
protected` `void` `finalize() 已过时。
finalize方法已被弃用。 为了执行清理，覆盖finalize子类应被修改为使用替代的清理机制，并删除覆盖的finalize方法。 当覆盖finalize方法时，其实现必须明确确保按照``super``.finalize()中所述调用``super``.finalize() 。 有关迁移选项的更多信息，请参阅Object.finalize()的规范。
FileChannel getChannel() 返回与此文件输出流相关联的唯一的FileChannel对象。
FileDescriptor getFD() 返回与此流相关联的文件描述符。
void` `write(``byte``[] b) 将 b.length字节从指定的字节数组写入此文件输出流。
void` `write(``byte``[] b, ``int` `off, ``int` `len) 将 len字节从指定的字节数组开始，从偏移量off开始写入此文件输出流。
void` `write(``int` `b) 将指定的字节写入此文件输出流。
```

具体使用：

```
package com.soft.ioex;

import java.io.*;

public class FileOutputStreamDemo1 {
    public static void main(String[] args) throws IOException {
        // 使用文件名称创建流对象
        FileOutputStream fos = new FileOutputStream("fos.txt");
        // 写出数据
        // 写出第1个字节
        fos.write(97);
        // 写出第2个字节
        fos.write(98);
        // 写出第3个字节
        fos.write(99);
        // 关闭资源
        fos.close();
    }
}
```

```
public class FileOutputStreamDemo2 {
    public static void main(String[] args) throws IOException {
// 使用文件名称创建流对象
        FileOutputStream fos = new FileOutputStream("fos.txt");
// 字符串转换为字节数组
        byte[] b = "abide".getBytes();
// 写出字节数组数据
        fos.write(b);
// 关闭资源
        fos.close();
    }
}

public class FileOutputStreamDemo3 {
    public static void main(String[] args) throws IOException {
// 使用文件名称创建流对象
        FileOutputStream fos = new FileOutputStream("fos.txt");
// 字符串转换为字节数组
        byte[] b = "abide".getBytes();
// 写出从索引2开始，2个字节。索引2是c，两个字节，也就是cd。
        fos.write(b, 2, 2);
// 关闭资源
        fos.close();
    }
}
```

3.5 File

Javaio流中还有一个非常常用的类：File。

File 是“**文件**”和“**目录路径名**”的抽象表示形式。
File 直接继承于Object，实现了Serializable接口和Comparable接口。实现Serializable接口，意味着File对象支持序列化操作。而实现Comparable接口，意味着File对象之间可以比较大小；File能直接被存储在有序集合(如TreeSet、TreeMap中)。

构造方法：

```
File(File parent, String child) 从父抽象路径名和子路径名字符串创建新的 File实例。
File(String pathname) 通过将给定的路径名字符串转换为抽象路径名来创建新的 File实例。
File(String parent, String child) 从父路径名字符串和子路径名字符串创建新的 File实例。
File(URI uri) 通过将给定的 file: URI转换为抽象路径名来创建新的 File实例。
```

方法列表：

```
boolean` `canExecute() 测试应用程序是否可以执行此抽象路径名表示的文件。
boolean` `canRead() 测试应用程序是否可以读取由此抽象路径名表示的文件。
boolean` `canWrite() 测试应用程序是否可以修改由此抽象路径名表示的文件。
int` `compareTo(File pathname) 比较两个抽象的路径名字典。
```

boolean` `createNewFile() 当且仅当具有该名称的文件尚不存在时，原子地创建一个由该抽象路径名命名的新的空文件。
static` `File createTempFile(String prefix, String suffix) 在默认临时文件目录中创建一个空文件，使用给定的前缀和后缀生成其名称。
static` `File createTempFile(String prefix, String suffix, File directory) 在指定的目录中创建一个新的空文件，使用给定的前缀和后缀字符串生成其名称。
boolean` `delete() 删除由此抽象路径名表示的文件或目录。
void` `deleteOnExit() 请求在虚拟机终止时删除由此抽象路径名表示的文件或目录。
boolean` `equals(Object obj) 测试此抽象路径名与给定对象的相等性。
boolean` `exists() 测试此抽象路径名表示的文件或目录是否存在。
File getAbsoluteFile() 返回此抽象路径名的绝对形式。
String getAbsolutePath() 返回此抽象路径名的绝对路径名字符串。
File getCanonicalFile() 返回此抽象路径名的规范形式。
String getCanonicalPath() 返回此抽象路径名的规范路径名字符串。
long` `getFreeSpace() 通过此抽象路径名返回分区 named中未分配字节的数量。
String getName() 返回由此抽象路径名表示的文件或目录的名称。
String getParent() 返回此抽象路径名的父目录的路径名字符串，如果此路径名未命名为父目录，则返回 ``null` `。
File getParentFile() 返回此抽象路径名的父目录的抽象路径名，如果此路径名不指定父目录，则返回 ``null` `。
String getPath() 将此抽象路径名转换为路径名字符串。
long` `getTotalSpace() 通过此抽象路径名返回分区 named的大小。
long` `getUsableSpace() 通过此抽象路径名返回分区 named上此虚拟机可用的字节数。
int` `hashCode() 计算此抽象路径名的哈希码。
boolean` `isAbsolute() 测试这个抽象路径名是否是绝对的。
boolean` `isDirectory() 测试此抽象路径名表示的文件是否为目录。
boolean` `isFile() 测试此抽象路径名表示的文件是否为普通文件。
boolean` `isHidden() 测试此抽象路径名命名的文件是否为隐藏文件。
long` `lastModified() 返回此抽象路径名表示的文件上次修改的时间。
long` `length() 返回由此抽象路径名表示的文件的长度。
String[] list() 返回一个字符串数组，命名由此抽象路径名表示的目录中的文件和目录。
String[] list(FilenameFilter filter) 返回一个字符串数组，命名由此抽象路径名表示的目录中满足指定过滤器的文件和目录。
File[] listFiles() 返回一个抽象路径名数组，表示由该抽象路径名表示的目录中的文件。
File[] listFiles(FileFilter filter) 返回一个抽象路径名数组，表示由此抽象路径名表示的满足指定过滤器的目录中的文件和目录。
File[] listFiles(FilenameFilter filter) 返回一个抽象路径名数组，表示由此抽象路径名表示的满足指定过滤器的目录中的文件和目录。
static` `File[] listRoots() 列出可用的文件系统根。
boolean` `mkdir() 创建由此抽象路径名命名的目录。
boolean` `mkdirs() 创建由此抽象路径名命名的目录，包括任何必需但不存在的父目录。
boolean` `renameTo(File dest) 重命名由此抽象路径名表示的文件。
boolean` `setExecutable(``boolean` `executable) 为此抽象路径名设置所有者的执行权限的便利方法。
boolean` `setExecutable(``boolean` `executable, ``boolean` `ownerOnly) 设置该抽象路径名的所有者或每个人的执行权限。
boolean` `setLastModified(``long` `time) 设置由此抽象路径名命名的文件或目录的最后修改时间。
boolean` `setReadable(``boolean` `readable) 一种方便的方法来设置所有者对此抽象路径名的读取权限。
boolean` `setReadable(``boolean` `readable, ``boolean` `ownerOnly) 设置此抽象路径名的所有者或每个人的读取权限。

boolean` `setReadOnly() 标记由此抽象路径名命名的文件或目录，以便只允许读取操作。
boolean` `setWritable(``boolean` `writable) 一种方便的方法来设置所有者对此抽象路径名的写入权限。
boolean` `setWritable(``boolean` `writable, ``boolean` `ownerOnly) 设置此抽象路径名的所有者或每个人的写入权限。
Path toPath() 返回从此抽象路径构造的一个java.nio.file.Path对象。
String toString() 返回此抽象路径名的路径名字符串。
URI toURI() 构造一个表示此抽象路径名的 file: URI。
URL toURL() 已过时。 此方法不会自动转义URL中非法的字符。 建议在新的代码转换的抽象路径到URL通过先转换成URI，经由toURI方法，然后经由转换URI为URL URI.toURL方法。

代码实例

```
import java.io.*;

/*
File类常见方法:
1，创建。
    boolean createNewFile():在指定位置创建文件，如果该文件已经存在，则不创建，返回false。
                            和输出流不一样，输出流对象一建立创建文件。而且文件已经存在，会覆盖。

    boolean mkdir():创建文件夹。
    boolean mkdirs():创建多级文件夹。
2，删除。
    boolean delete()：删除失败返回false。如果文件正在被使用，则删除不了返回falsel。
    void deleteOnExit();在程序退出时删除指定文件。


3，判断。
    boolean exists() :文件是否存在.
    isFile():
    isDirectory();
    isHidden();
    isAbsolute();

4，获取信息。
    getName():
    getPath():
    getParent():

    getAbsolutePath()
    long lastModified()
    long length()

*/
```

递归列出指定目录下的文件或者文件夹

```
package com.soft.ioex;

public class RecursionSum {
    public static void main(String[] args) {
        //计算1~num的和，使用递归完成
        int num=5;
        // 调用求和的方法
        int sum=getSum(num);
        // 输出结果
        System.out.println(sum);
    }
    /** 递归算法实现
     *  @param num: 入参
     * @return int
     */

    private static int getSum(int num) {
        //num为1时,方法返回1,相当于是方法的出口,num总有是1的情况
        if (num==1) {
            return 1;
        }
        //num不为1时,方法返回 num +(num-1)的累和，递归调用getSum方法
        return num+getSum(num-1);
    }
}

package com.soft.ioex;

public class RecursionFactorial {
    public static void main(String[] args) {
        int res = getFactorial(5);
        System.out.println(res);
    }
    public static int getFactorial(int num) {
        //1的阶乘为1
        if (num == 1) {
            return 1;

        }
        //n不为1时,方法返回 n! = n*(n-1)! 递归调用getValue方法

        return num * getFactorial(num - 1);
    }
}
```