

个人资料



伯努力不努力

+ 关注

✉ 发私信



恒

访问：253349次

积分：3430

等级：BLOG > 5

排名：第9730名

原创：85篇

转载：0篇

译文：0篇

评论：172条

文章搜索

🔍

博客专栏



Kotlin学习之路

文章：0篇

阅读：0

文章分类

Android (9)

Material Design (5)

架构设计 (3)

自定义控件 (5)

性能优化 (9)

开发笔记 (4)

插件化系列 (10)

混合开发 (1)

开源框架解析 (7)

安卓源码解析 (15)

设计模式 (10)

热修复系列 (3)

原

务实java基础之IO

标签：

java

IO流

2017-08-14 00:06

👁 1818人阅读

💬 评论(3)

★ 收藏

⚠ 举报

☰ 分类：

java (6) ▾

❗ 版权声明：本文为博主原创文章，未经博主允许不得转载。


目录(?)

[+]

“对语言设计人员来说，创建好的输入 / 输出系统是一项特别困难的任务。” 由于存在大量不同的设计方案，所以该任务的困难性是很容易证明的。其中最大的挑战似乎是如何覆盖所有可能的因素。不仅有三种不同的种类的 IO 需要考虑（文件、控制台、网络连接），而且需要通过大量不同的方式与它们通信（顺序、随机访问、二进制、字符、按行、按字等等）。

Java 库的设计者通过创建大量类来攻克这个难题。事实上，Java 的 IO 系统采用了如此多的类，以致刚开始会产生不知从何处入手的感觉，本篇博客我们就来详细学习Java的IO 系统。

先上一张图



文章存档

2017年08月 (2)
2017年07月 (6)
2017年06月 (4)
2017年05月 (14)
2017年04月 (12)

展开

阅读排行

Android Gradle知识梳理 (20338)
一篇博客让你了解RxJava (16389)
全面解析Notification (9552)
一篇博客理解Recyclerview的... (8551)
深入解析OkHttp3 (8371)
设计模式学习之策略模式 (7137)
浅谈安卓中的MVP模式 (6830)
Android性能优化系列之布局... (6107)
Android进程保活全攻略 (上) (5843)
Android热修复学习之旅——... (5688)

评论排行

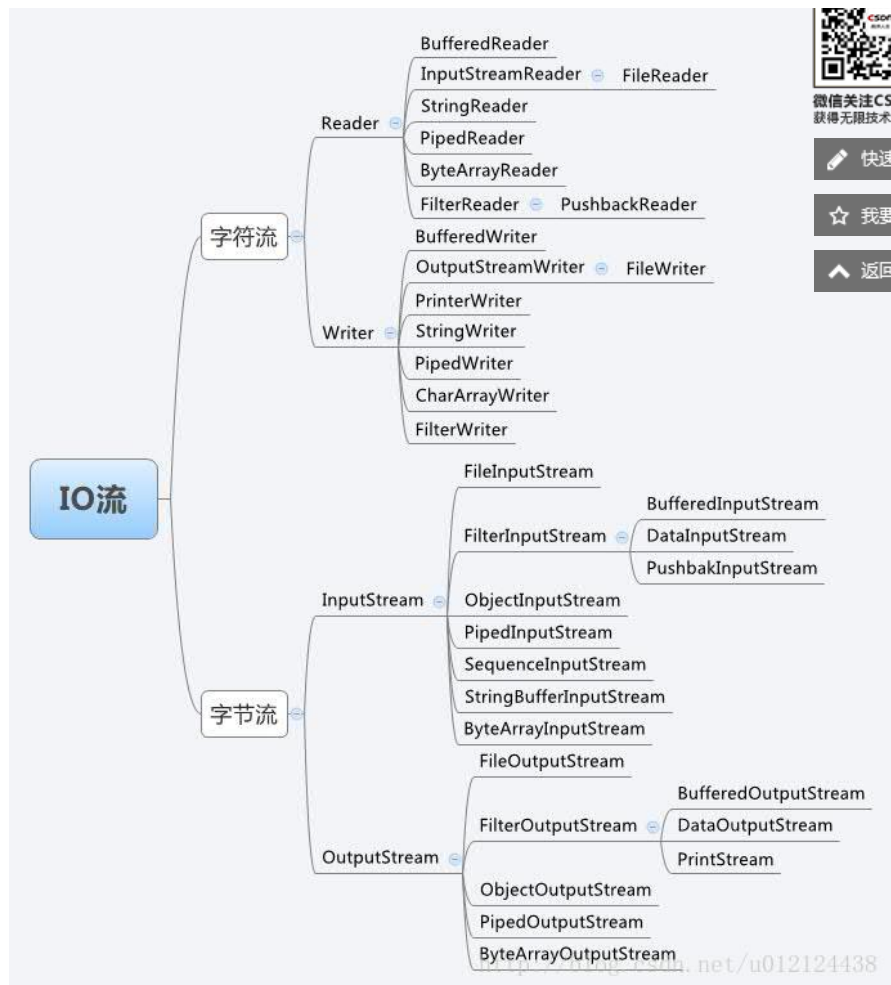
设计模式学习之策略模式 (15)
全面解析Notification (14)
一篇博客让你了解RxJava (9)
Android 贝塞尔曲线解析 (7)
Android性能优化系列之布局... (7)
《深入理解java虚拟机》学习... (7)
Android Studio常用技巧汇总 (7)
Android性能优化系列之内存... (7)
一篇博客理解Recyclerview的... (7)
深入解析Android中Handler消... (6)

推荐文章

* CSDN日报20170725——《新的开始，从研究生到入职亚马逊》
* 深入剖析基于并发AQS的重入锁(Reentrant Lock)及其Condition实现原理
* Android版本的"Wannacry"文件加密病毒样本分析(附带锁机)
* 工作与生活真的可以平衡吗？
* 《Real-Time Rendering 3rd》提炼总结——高级着色：BRDF及相关技术
* 《三体》读后思考-泰勒展开/维度打击/黑暗森林

最新评论

务实java基础之IO
lostsky11 : 夯实，除了代码，自己的母语也要夯实基础
务实java基础之IO
ricardoMye : 很好
务实java基础之IO
sonofbaba : 不错
深入解析OkHttp3
hzc543806053 : @wen_and_zi:一行代码搞定okhttphttps://github.com/microsh...



IO中的输入字节流类型

InputStream 的作用是标志那些从不同起源地产生输入的类。这些起源地包括（每个都有一个相关的InputStream 子类）：

- (1) 字节数组
- (2) String 对象
- (3) 文件
- (4) “管道”，它的工作原理与现实生活中的管道类似：将一些东西置入一端，它们在另一端出来。
- (5) 一系列其他流，以便我们将其统一收集到单独一个流内。
- (6) 其他起源地，如 Internet 连接等

除此以外，FilterInputStream 也属于 InputStream 的一种类型，用它可将属性或者有用的接口同输入流连接到一起。

1. InputStream是所有的输入字节流的父类，它是一个抽象类。
2. ByteArrayInputStream、StringBufferInputStream、FileInputStream是三种基本的介质流，它们分别将Byte数组、StringBuffer、和本地文件中读取数据。PipedInputStream是从与其它线程共用的管道中读取数据，与Piped相关的知识会用专门的一小节讲解。
3. ObjectInputStream和所有FilterInputStream的子类都是装饰流（装饰器模式的主角）。

基本输入字节流：

深入解析OkHttp3
吴迪123：@wen_and_zi:在拦截器的intercept方法中调用了Chain.proceed

务实java基础之集合总结
hjl3692008：很多基础内容，但是太长了不想看，建议分组归类成多篇文章，作为一个码农整理数据是很重要的。

滴滴插件化VirtualAPK框架原理解析（二....
奋斗小小鸟cy：受益匪浅

深入解析OkHttp3
_thc：大佬，上面说到，Chain与Interceptor会互相递归调用，直到链的尽头。没有看到递归调用呢，...

Android 贝塞尔曲线解析
simon_morning：好东西，学习了 ,正好用的上..

Android 贝塞尔曲线解析
伯努力不努力：@SEU_Calvin:AD:AB = BE:BC按照这个比例，你确定了D点就可以推算出E点的位置

类	功能	如何构造	怎样使用
ByteArrayInputStream	将内存中的Byte数组适配为一个InputStream。	从内存中的Byte数组创建该对象（2种方法）	一般作为数据源，会使用其它装饰流提供额外的功能，一般都建议加个缓冲功能。
StringBufferInputStream	将内存中的字符串适配为一个InputStream。	从一个String对象创建该对象。底层的实现使用StringBuffer。该类被Deprecated。主要原因是StringBuffer不应该属于字节流，所以推荐使用StringReader。	一般作为数据源，同样会使用其它装饰器提供额外的功能。
FileInputStream	最基本的文件输入流。主要用于从文件中读取信息。	通过一个代表文件路径的String、File对象或者FileDescriptor对象创建。	一般作为数据源，同样会使用其它装饰器提供额外的功能。
PipedInputStream	读取从对应PipedOutputStream写入的数据。在流中实现了管道的概念。	利用对应的PipedOutputStream创建。	在多线程程序中作为数据源，同样会使用其它装饰器提供额外的功能。
SequenceInputStream	将2个或者多个InputStream对象转变为一个InputStream。	使用两个InputStream或者内部对象为InputStream的Enumeration对象创建该对象。	一般作为数据源，同样会使用其它装饰器提供额外的功能。
FilterInputStream	给其它被装饰对象提供额外功能的抽象类	主要子类见下表	

装饰、输入字节流：

类	功能	如何构造	怎样使用
DataInputStream	一般和DataOutputStream配对使用,完成基本数据类型的读写。	利用一个InputStream构造。	提供了大量的读取基本数据类新的读取方法。
BufferedInputStream	使用该对象阻止每次读取一个字节都会频繁操作IO。将字节读取一个缓存区，从缓存区读取。	利用一个InputStream、或者带上一个自定义的缓存区的大小构造。	使用InputStream的方法读取，只是背后多一个缓存的功能。设计模式中透明装饰器的应用。
LineNumberInputStream	跟踪输入流中的行号。可以调用getLineNumber()和setLineNumber(int)方法得到和设置行号。	利用一个InputStream构造。	紧紧增加一个行号。可以象使用其它InputStream一样使用。
PushbackInputStream	可以在读取最后一个byte后将其放回到缓存中。	利用一个InputStream构造。	一般仅仅会在设计compiler的scanner时会用到这个类。在我们的java语言的编译器中使用它。很多程序员可能

IO中的输出字节流类型

- 1.OutputStream是所有的输出字节流的父类，它是一个抽象类。
- 2. ByteArrayOutputStream、FileOutputStream是两种基本的介质流，它们分别向Byte数组、和本地文件中写入数据。PipedOutputStream是向与其它线程共用的管道中写入数据
- 3. ObjectOutputStream和所有FilterOutputStream的子类都是装饰流。下表列出了输出字节流的功能及如何使用它们。

类	功能	如何构造	怎样使用
ByteArrayOutputStream	在内存中创建一个buffer。所有写入此流中的数据都被放入到此buffer中。	无参或者使用一个可选的初始化buffer的大小的参数构造。	一般将其和FilterOutputStream套接得到额外的功能。建议首先和BufferedOutputStream套接实现缓冲功能。通过toByteArray方法可以得到流中的数据。（不透明装饰器的用法）
FileOutputStream	将信息写入文件中。	使用代表文件路径的String、File对象或者FileDescriptor对象创建。还可以加一个代表写入的方式是否为append的标记。	一般将其和FilterOutputStream套接得到额外的功能。
PipedOutputStream	任何写入此对象的信息都被放入对应PipedInputStream对象的缓存中，从而完成线程的通信，实现了“管道”的概念。具体在后面详细讲解。	利用PipedInputStream构造	在多线程程序中数据的目的地的。一般将其和FilterOutputStream套接得到额外的功能。
FilterOutputStream	实现装饰器功能的抽象类。为其它OutputStream对象增加额外的功能。	见下表	见下表

装饰输出字节流：

类	功能	如何构造	怎样使用
DataOutputStream	通常和DataInputStream配合使用，使用它可以写入基本数据类型新。	使用OutputStream构造	包含大量的写入基本数据类型的方法。
PrintStream	产生具有格式的输出信息。（一般地在java程序中DataOutputStream用于数据的存储，即J2EE中持久层完成的功能，PrintStream完成显示的功能，类似于J2EE中表现层的功能）	使用OutputStream和一个可选的表示缓存是否在每次换行时是否flush的标记构造。还提供很多和文件相关的构造方法。	一般是一个终极（“final”）的包装器，很多时候我们都使用它！
BufferedOutputStream	使用它可以避免频繁地向IO写入数据，数据一般都写入	从一个OutputStream或	提供和其它OutputStream

一个缓存区，在调用flush方法后会清空缓存、一次完成数据的写入。

者和一个代表缓存区大小的可选参数构造。

一致的接口，只是内部提供一个缓存的功能。

IO中的字节流使用

1.FileInputStream使用

```
1 public static void main(String[] args) {
2
3     int count=0; //统计文件字节长度
4     InputStreamstreamReader = null; //文件输入流
5     try{
6         streamReader=newFileInputStream(new File("D:/files/test.jpg"));
7
8         while(streamReader.read() != -1) { //读取文件字节，并递增指针到下一个字节
9             count++;
10        }
11        System.out.println("---长度是: "+count+" 字节");
12    }catch (final IOException e) {
13
14        e.printStackTrace();
15    }finally{
16        try{
17            streamReader.close();
18        }catch (IOException e) {
19
20            e.printStackTrace();
21        }
22    }
23 }
24 }
```

上面的程序存在问题是，每读取一个自己我都要去用到FileInputStream，我输出的结果是“—长度是：64982 字节”，那么进行了64982次操作！可能想象如果文件十分庞大，这样的操作肯定会出大问题，所以引出了缓冲区的概念。可以将streamReader.read()改成streamReader.read(byte[]b)此方法读取的字节数目等于字节数组的长度，读取的数据被存储在字节数组中，返回读取的字节数，InputStream还有其它方法mark,reset,markSupported方法，例如：

markSupported 判断该输入流能支持mark 和 reset 方法。

mark用于标记当前位置；在读取一定数量的数据(小于readlimit的数据)后使用reset可以回到mark标记的位置。

FileInputStream不支持mark/reset操作；BufferedInputStream支持此操作；

mark(readlimit)的含义是在当前位置作一个标记，制定可以重新读取的最大字节数，也就是说你如果标记后读取的字节数大于readlimit，你就再也回不到回来的位置了。

通常InputStream的read()返回-1后，说明到达文件尾，不能再读取。除非使用了mark/reset。

2.FileOutputStream 使用

```
1 public class FileCopy {
2
3     public static void main(String[] args) {
4
5         byte[] buffer=new byte[512]; //一次取出的字节数大小,缓冲区大小
6         int numberRead=0;
7         FileOutputStream input=null;
```

```

8      FileOutputStream out =null;
9      try {
10         input=new FileInputStream("D:/files/test.jpg");
11         out=new FileOutputStream("D:/files/test2.jpg"); //如果文件不存在会自动创建
12
13         while ((numberRead=input.read(buffer))!=-1) { //numberRead的目的在于防止最后
14             out.write(buffer, 0, numberRead); //否则会自动被填充0
15         }
16     } catch (final IOException e) {
17
18         e.printStackTrace();
19     }finally{
20         try {
21             input.close();
22             out.close();
23         } catch (IOException e) {
24
25             e.printStackTrace();
26         }
27     }
28 }
29 }
30
31 }

```

3.ObjectInputStream 和ObjectOutputStream使用

该流允许读取或写入用户自定义的类，但是要实现这种功能，被读取和写入的类必须实现

Serializable接口，其实该接口并没有什么方法，可能相当于一个标记而已，但是确实不能缺少的。

```

1  public class ObjectStream {
2
3      /**
4       * @param args
5       */
6      public static void main(String[] args) {
7
8          ObjectOutputStream objectwriter=null;
9          ObjectInputStream objectreader=null;
10
11         try {
12             objectwriter=new ObjectOutputStream(new FileOutputStream("D:/files/student.txt"));
13             objectwriter.writeObject(new Student("gg", 22));
14             objectwriter.writeObject(new Student("tt", 18));
15             objectwriter.writeObject(new Student("rr", 17));
16             objectreader=new ObjectInputStream(new FileInputStream("D:/files/student.txt"));
17             for (int i = 0; i < 3; i++) {
18                 System.out.println(objectreader.readObject());
19             }
20         } catch (IOException | ClassNotFoundException e) {
21
22             e.printStackTrace();
23         }finally{
24             try {
25                 objectreader.close();
26                 objectwriter.close();
27             } catch (IOException e) {
28
29                 e.printStackTrace();
30             }
31         }
32     }
33 }
34 }
35
36 }
37 class Student implements Serializable{
38     private String name;
39     private int age;

```

```

40
41     public Student(String name, int age) {
42         super();
43         this.name = name;
44         this.age = age;
45     }
46
47     @Override
48     public String toString() {
49         return "Student [name=" + name + ", age=" + age + "]";
50     }
51
52
53 }

```

运行后系统输出：

Student [name=gg, age=22]

Student [name=tt, age=18]

Student [name=rr, age=17]

4.DataInputStream与DataOutputStream使用

有时没有必要存储整个对象的信息，而只是要存储一个对象的成员数据，成员数据的类型假设都是Java的基本数据类型，这样的需求不必使用到与Object输入、输出相关的流对象，可以使用DataInputStream、DataOutputStream来写入或读出数据。

```

1  public class Member {
2      private String name;
3      private int age;
4      public Member() {
5      }
6      public Member(String name, int age) {
7          this.name = name;
8          this.age = age;
9      }
10     public void setName(String name) {
11         this.name = name;
12     }
13     public void setAge(int age) {
14         this.age = age;
15     }
16     public String getName() {
17         return name;
18     }
19     public int getAge() {
20         return age;
21     }
22 }

```

打算将Member类实例的成员数据写入文件中，并打算在读入文件数据后，将这些数据还原为Member对象。

```

1  public class DataStreamDemo
2  {
3      public static void main(String[] args)
4      {
5          Member[] members = {newMember("Justin", 90),
6                               newMember("momor", 95),
7                               newMember("Bush", 88)};
8
9          try
10         {
11             DataOutputStreamdataOutputStream = new DataOutputStream(new FileOutputStream("a

```



```

12     for (Member member:members)
13     {
14         //写入UTF字符串
15         dataOutputStream.writeUTF(member.getName());
16         //写入int数据
17         dataOutputStream.writeInt(member.getAge());
18     }
19
20     //所有数据至目的地
21     dataOutputStream.flush();
22     //关闭流
23     dataOutputStream.close();
24
25     DataInputStream dataInputStream = new DataInputStream(new FileInputStream(e
26
27     //读出数据并还原为对象
28     for (int i=0; i<members.length; i++)
29     {
30         //读出UTF字符串
31         String name = dataInputStream.readUTF();
32         //读出int数据
33         int score = dataInputStream.readInt();
34         members[i] = new Member(name, score);
35     }
36
37     //关闭流
38     dataInputStream.close();
39
40     //显示还原后的数据
41     for (Member member : members)
42     {
43         System.out.printf("%s\t%d\n", member.getName(), member.getAge());
44     }
45 }
46 catch (IOException e)
47 {
48     e.printStackTrace();
49 }
50 }
51 }

```

5.PushbackInputStream

PushbackInputStream类继承了FilterInputStream类是InputStream类的修饰者。提供可以将数据插入到输入流前端的能力(当然也可以做其他操作)。简而言之PushbackInputStream类的作用就是能够在读取缓冲区的时候提前知道下一个字节是什么，其实质是读取到下一个字符后回退的做法，这之间可以进行很多操作，这有点向你把读取缓冲区的过程当成一个数组的遍历，遍历到某个字符的时候可以进行的操作，当然，如果要插入，能够插入的最大字节数是与推回缓冲区的大小相关的，插入字符肯定不能大于缓冲区吧！

```

1  /**
2   * 回退流操作
3   */
4  public class PushBackInputStreamDemo {
5      public static void main(String[] args) throws IOException {
6          String str = "hello,rollenholt";
7          PushbackInputStream push = null; // 声明回退流对象
8          ByteArrayInputStream bat = null; // 声明字节数组流对象
9          bat = new ByteArrayInputStream(str.getBytes());
10         push = new PushbackInputStream(bat); // 创建回退流对象，将拆解的字节数组流传入
11         int temp = 0;
12         while ((temp = push.read()) != -1) { // push.read() 逐字节读取存放在temp中，如果读到
13             if (temp == ',') { // 判断读取的是否是逗号
14                 push.unread(temp); // 回到temp的位置
15                 temp = push.read(); // 接着读取字节
16                 System.out.print("(回退" + (char) temp + ")"); // 输出回退的字符

```



```

17     } else {
18         System.out.print((char) temp); // 否则输出字符
19     }
20 }
21 }
22 }
23

```

6.SequenceInputStream使用

当我们需要从多个输入流中向程序读入数据。此时，可以使用合并流，将多个输入流合并成一个 SequenceInputStream 流对象。SequenceInputStream 会将与之相连接的流集组合成一个输入流，并从第一个输入流开始读取，直到到达文件末尾，接着从第二个输入流读取，依次类推，直到到达包含的最后一个输入流的文件末尾为止。合并流的作用是将多个源合并成一个源。其可接收枚举类所封闭的多个字节流对象。

```

1 public class SequenceInputStreamTest {
2     /**
3      * @param args
4      *      SequenceInputStream 合并流，将与之相连接的流集组合成一个输入流并从第一
5      *      直到到达文件末尾，接着从第二个输入流读取，依次类推，直到到达包含的最后
6      *      合并流的作用是将多个源合并成一个源。可接收枚举类所封闭的多个字节流对象
7      */
8     public static void main(String[] args) {
9         doSequence();
10    }
11
12    private static void doSequence() {
13        // 创建一个合并流的对象
14        SequenceInputStream sis = null;
15        // 创建输出流。
16        BufferedOutputStream bos = null;
17        try {
18            // 构建流集合。
19            Vector<InputStream> vector = new Vector<InputStream>();
20            vector.addElement(new FileInputStream("D:\text1.txt"));
21            vector.addElement(new FileInputStream("D:\text2.txt"));
22            vector.addElement(new FileInputStream("D:\text3.txt"));
23            Enumeration<InputStream> e = vector.elements();
24
25            sis = new SequenceInputStream(e);
26
27            bos = new BufferedOutputStream(new FileOutputStream("D:\text4.txt"));
28            // 读写数据
29            byte[] buf = new byte[1024];
30            int len = 0;
31            while ((len = sis.read(buf)) != -1) {
32                bos.write(buf, 0, len);
33                bos.flush();
34            }
35        } catch (FileNotFoundException e1) {
36            e1.printStackTrace();
37        } catch (IOException e1) {
38            e1.printStackTrace();
39        } finally {
40            try {
41                if (sis != null)
42                    sis.close();
43            } catch (IOException e) {
44                e.printStackTrace();
45            }
46            try {
47                if (bos != null)
48                    bos.close();
49            } catch (IOException e) {
50                e.printStackTrace();

```

```
51     }
52     }
53 }
54 }
55
```

7.PrintStream使用

System.out这个对象就是PrintStream

File类

File 类有一个欺骗性的名字——通常会认为它对付的是一个文件，但实情并非如此。它既代表一个特定文件的名字，也代表目录内一系列文件的名字。若代表一个文件集，便可用list()方法查询这个集，返回的是一个字符串数组。之所以要返回一个数组，而非某个灵活的集合类，是因为元素的数量是固定的。而且若想得到一个不同的目录列表，只需创建一个不同的 File 对象即可。事实上，“FilePath”（文件路径）似乎是一个更好的名字。

目录列表器

可用两种方式列出 File 对象。若在不含自变量（参数）的情况下调用list()，会获得 File 对象包含的一个完整列表。然而，若想对这个列表进行某些限制，就需要使用一个“目录过滤器”，该类的作用是指出应如何选择 File 对象来完成显示。

```
1  //: DirList.java
2  // Displays directory listing
3  package c10;
4  import java.io.*;
5  public class DirList {
6  public static void main(String[] args) {
7  try {
8  File path = new File(".");
9  String[] list;
10 if(args.length == 0)
11 list = path.list();
12 else
13 list = path.list(new DirFilter(args[0]));
14
15 for(int i = 0; i < list.length; i++)
16 System.out.println(list[i]);
17 } catch(Exception e) {
18 e.printStackTrace();
19 }
20 }
21 }
22 class DirFilter implements FilenameFilter {
23 String afn;
24 DirFilter(String afn) { this.afn = afn; }
25 public boolean accept(File dir, String name) {
26 // Strip path information:
27 String f = new File(name).getName();
28 return f.indexOf(afn) != -1;
29 }
30 } ///
```

DirFilter 类“实现”了 interface FilenameFilter（关于接口的问题，已在第 7 章进行了详述）。下面让我们看看 FilenameFilter 接口有多么简单：

```
public interface FilenameFilter {

boolean accept(文件目录, 字符串名);
```

```
}
```

之所以要创建这样的一个类，背后的全部原因就是要把 `accept()` 方法提供给 `list()` 方法，使 `list()` 能够“回调” `accept()`，从而判断应将哪些文件名包括到列表中。因此，通常将这种技术称为“回调”，有时也称为“算子”（也就是说，`DirFilter` 是一个算子，因为它唯一的作用就是容纳一个方法）。

通过 `DirFilter`，我们看出尽管一个“接口”只包含了一系列方法，但并不局限于只能写那些方法（但是，至少必须提供一个接口内所有方法的定义。在这种情况下，`DirFilter` 构建器也会创建）。

`accept()` 方法必须接纳一个 `File` 对象，用它指示用于寻找一个特定文件的目录；并接纳一个 `String`，其中包含了要寻找之文件的名字。可决定使用或忽略这两个参数之一，但有时至少要使用文件名。记住 `list()` 方法准备为目录对象中的每个文件名调用 `accept()`，核实哪个应包含在内——具体由 `accept()` 返回的“布尔”结果决定。

1. 匿名内部类

下例用一个匿名内部类来重写显得非常理想。首先创建了一个 `filter()` 方法，它返回指向 `FilenameFilter` 的一个句柄：

```
1  //: DirList2.java
2  // Uses Java 1.1 anonymous inner classes
3  import java.io.*;
4  public class DirList2 {
5      public static FilenameFilter
6      filter(final String afn) {
7          // Creation of anonymous inner class:
8          return new FilenameFilter() {
9              String fn = afn;
10             public boolean accept(File dir, String n) {
11                 // Strip path information:
12                 String f = new File(n).getName();
13                 return f.indexOf(fn) != -1;
14             }
15         }; // End of anonymous inner class
16     }
17     public static void main(String[] args) {
18         try {
19             File path = new File(".");
20             String[] list;
21             if(args.length == 0)
22                 list = path.list();
23             else
24                 list = path.list(filter(args[0]));
25             for(int i = 0; i < list.length; i++)
26                 System.out.println(list[i]);
27         } catch(Exception e) {
28             e.printStackTrace();
29         }
30     }
31 } ///:
```

注意 `filter()` 的自变量必须是 `final`。这一点是匿名内部类要求的，使其能使用来自本身作用域以外的一个对象。

之所以认为这样做更好，是由于 `FilenameFilter` 类现在同 `DirList2` 紧密地结合在一起。然而，我们可采取进一步的操作，将匿名内部类定义成 `list()` 的一个参数，使其显得更加精简。如下所示：

```

2  //: DirList3.java
3  // Building the anonymous inner class "in-place"
4  import java.io.*;
5  public class DirList3 {
6  public static void main(final String[] args) {
7  try {
8  File path = new File(".");
9  String[] list;
10 if(args.length == 0)
11 list = path.list();
12 else
13 list = path.list(
14 new FilenameFilter() {
15 public boolean
16 accept(File dir, String n) {
17 String f = new File(n).getName();
18 291
19 return f.indexOf(args[0]) != -1;
20 }
21 });
22 for(int i = 0; i < list.length; i++)
23 System.out.println(list[i]);
24
25 } catch(Exception e) {
26 e.printStackTrace();
27 }
28 }
29 } ///:

```

main()现在的自变量是 final，因为匿名内部类直接使用 args[0]。

这展示了如何利用匿名内部类快速创建精简的类，以便解决一些复杂的问题。由于Java 中的所有东西都与类有关，所以它无疑是一种相当有用的编码技术。它的一个好处是将特定的问题隔离在一个地方统一解决。但在另一方面，这样生成的代码不是十分容易阅读，所以使用时必须慎重。

2.顺序目录列表

经常都需要文件名以排好序的方式提供。由于 **Java 1.0** 和 **java 1.1** 都没有提供对排序的支持（从 **Java 1.2**开始提供），所以必须用 **SortVector** 将这一能力直接加入自己的程序。就象下面这样：

```

1  //: SortedDirList.java
2  // Displays sorted directory listing
3  import java.io.*;
4  import c08.*;
5  public class SortedDirList {
6  private File path;
7  private String[] list;
8  public SortedDirList(final String afn) {
9  path = new File(".");
10 if(afn == null)
11 list = path.list();
12 else
13 list = path.list(
14 new FilenameFilter() {
15 public boolean
16 accept(File dir, String n) {
17 String f = new File(n).getName();
18 return f.indexOf(afn) != -1;
19 }
20 });
21 sort();
22 }
23 void print() {
24 for(int i = 0; i < list.length; i++)
25 System.out.println(list[i]);
26 }
27 private void sort() {
28 StrSortVector sv = new StrSortVector();

```

```

29 for(int i = 0; i < list.length; i++)
30 sv.addElement(list[i]);
31 // The first time an element is pulled from
32 // the StrSortVector the list is sorted:
33 for(int i = 0; i < list.length; i++)
34 list[i] = sv.elementAt(i);
35 }
36 // Test it:
37 public static void main(String[] args) {
38 SortedDirList sd;
39 if(args.length == 0)
40 sd = new SortedDirList(null);
41 else
42 sd = new SortedDirList(args[0]);
43 sd.print();
44 }
45 } ///:~

```

这里进行了另外少许改进。不再是将 path (路径) 和 list (列表) 创建为 main() 的本地变量，它们变成了类的成员，使它们的值能在对象“生存”期间方便地访问。

这种排序不要求区分大小写，所以最终不会得到一组全部单词都以大写字母开头的列表，跟着是全部以小写字母开头的列表。然而，我们注意到在以相同字母开头的一组文件名中，大写字母是排在前面的——这对标准的排序来说仍是一种不合格的行为。Java 1.2 已成功解决了这个问题。

检查与创建目录

File 类并不仅仅是对现有目录路径、文件或者文件组的一个表示。亦可用一个 File 对象新建一个目录，甚至创建一个完整的目录路径——假如它尚不存在的话。亦可用它了解文件的属性（长度、上一次修改日期、读 / 写属性等），检查一个 File 对象到底代表一个文件还是一个目录，以及删除一个文件等等。下列程序完整展示了如何运用 File 类剩下的这些方法：

```

1  ///: MakeDirectories.java
2  // Demonstrates the use of the File class to
3  // create directories and manipulate files.
4  import java.io.*;
5  public class MakeDirectories {
6  private final static String usage =
7  "Usage: MakeDirectories path1 ... \n" +
8  "Creates each path\n" +
9  "Usage: MakeDirectories -d path1 ... \n" +
10 "Deletes each path\n" +
11 "Usage: MakeDirectories -r path1 path2\n" +
12 "Renames from path1 to path2\n";
13 private static void usage() {
14 System.err.println(usage);
15 System.exit(1);
16 }
17 private static void fileData(File f) {
18 System.out.println(
19 "Absolute path: " + f.getAbsolutePath() +
20 "\n Can read: " + f.canRead() +
21 "\n Can write: " + f.canWrite() +
22 "\n getName: " + f.getName() +
23 "\n getParent: " + f.getParent() +
24 "\n getPath: " + f.getPath() +
25 "\n length: " + f.length() +
26 "\n lastModified: " + f.lastModified());
27 if(f.isFile())
28 System.out.println("it's a file");
29 else if(f.isDirectory())
30 System.out.println("it's a directory");
31 }
32 public static void main(String[] args) {
33 if(args.length < 1) usage();

```

```

34  if(args[0].equals("-r")) {
35  if(args.length != 3) usage();
36  File
37  old = new File(args[1]),
38  rname = new File(args[2]);
39  old.renameTo(rname);
40  fileData(old);
41  fileData(rname);
42  return; // Exit main
43  }
44  int count = 0;
45  boolean del = false;
46  if(args[0].equals("-d")) {
47  count++;
48  del = true;
49  }
50  for( ; count < args.length; count++) {
51  File f = new File(args[count]);
52  if(f.exists()) {
53  System.out.println(f + " exists");
54  if(del) {
55  System.out.println("deleting..." + f);
56  f.delete();
57  }
58  }
59  else { // Doesn't exist
60  if(!del) {
61  f.mkdirs();
62  System.out.println("created " + f);
63  }
64  }
65  fileData(f);
66  }
67  }
68  } ///:~

```

在 fileData()中，可看到应用了各种文件调查方法来显示与文件或目录路径有关的信息。

main()应用的第一个方法是 renameTo()，利用它可以重命名（或移动）一个文件至一个全新的路径（该路由参数决定），它属于另一个 File 对象。这也适用于任何长度的目录。

若试验上述程序，就可发现自己能制作任意复杂程度的一个目录路径，因为mkdirs() 会帮我们完成所有工作。在 Java 1.0 中，-d 标志报告目录虽然已被删除，但它依然存在；但在 Java 1.1 中，目录会被实际删除。

IO中的输入字符流

1.Reader是所有的输入字符流的父类，它是一个抽象类。2.CharReader、StringReader是两种基本的介质流，它们分别将Char数组、String中读取数据。PipedReader是从与其它线程共用的管道中读取数据。

3. BufferedReader很明显就是一个装饰器，它和其子类负责装饰其它Reader对象。

4. FilterReader是所有自定义具体装饰流的父类，其子类PushbackReader对Reader对象进行装饰，会增加一个行号。

5. InputStreamReader是一个连接字节流和字符流的桥梁，它将字节流转变为字符流。FileReader可以说是一个达到此功能、常用的工具类，在其源代码中明显使用了将FileInputStream转变为Reader的方法。我们可以从这个类中得到一定的技巧。

IO中的输出字符流

1. `Writer`是所有的输出字符流的父类，它是一个抽象类。
2. `CharArrayWriter`、`StringWriter`是两种基本的介质流，它们分别向`Char`数组、`String`中写入数据。`PipedWriter`是向与其它线程共用的管道中写入数据
3. `BufferedWriter`是一个装饰器为`Writer`提供缓冲功能。
4. `PrintWriter`和`PrintStream`极其类似，功能和使用也非常相似。
5. `OutputStreamWriter`是`OutputStream`到`Writer`转换的桥梁，它的子类`FileWriter`其实就是一个实现此功能的具体类（具体可以研究一下Source Code）。功能和使用和`OutputStream`极其类似，后面会有它们的对应图。

IO字符流的使用

1. `FileReader`与`PrintWriter`

```
1 public class Print {
2
3     /**
4      * @param args
5      */
6     public static void main(String[] args) {
7         // TODO自动生成的方法存根
8         char[] buffer=new char[512]; //一次取出的字节数大小,缓冲区大小
9         int numberRead=0;
10        FileReader reader=null; //读取字符文件的流
11        PrintWriter writer=null; //写字符到控制台的流
12
13        try {
14            reader=new FileReader("D:/files/copy1.txt");
15            writer=new PrintWriter(System.out); //PrintWriter可以输出字符到文件，也可以输出到控制台
16            while ((numberRead=reader.read(buffer))!=-1) {
17                writer.write(buffer, 0, numberRead);
18            }
19        } catch (IOException e) {
20
21            e.printStackTrace();
22        }finally{
23            try {
24                reader.close();
25            } catch (IOException e) {
26
27                e.printStackTrace();
28            }
29            writer.close(); //这个不用抛异常
30        }
31    }
32 }
33
34 }
```

2. `BufferedWriter`与`BufferedReader`

```
1 public class FileConcatenate {
2
3     /**
4      * 包装类进行文件级联操作
5      */
6     public static void main(String[] args) {
7
8         try {
9             concennateFile(args);
10        } catch (IOException e) {
11
12        }
```



```

12         e.printStackTrace();
13     }
14 }
15 public static void concatenateFile(String... fileName) throws IOException {
16     String str;
17     //构建对该文件您的输入流
18     BufferedWriter writer=new BufferedWriter(new FileWriter("D:/files/copy2.txt"));
19     for (String name: fileName) {
20         BufferedReader reader=new BufferedReader(new FileReader(name));
21
22         while ((str=reader.readLine())!=null) {
23             writer.write(str);
24             writer.newLine();
25         }
26     }
27 }
28
29 }

```

4.Console

该类提供了用于读取密码的方法，可以禁止控制台回显并返回char数组，对两个特性对保证安全有作用，平时用的不多，了解就行。

5.StreamTokenizer

它可以把输入流解析为标记 (token) , StreamTokenizer 并非派生自InputStream或者OutputStream，而是归类于io库中，因为StreamTokenizer只处理InputStream对象。

```

1  /**
2   * 使用StreamTokenizer来统计文件中的字符数
3   * StreamTokenizer 类获取输入流并将其分析为“标记”，允许一次读取一个标记。
4   * 分析过程由一个表和许多可以设置为各种状态的标志控制。
5   * 该流的标记生成器可以识别标识符、数字、引用的字符串和各种注释样式。
6   *
7   * 默认情况下，StreamTokenizer认为下列内容是Token：字母、数字、除C和C++注释符号以外的
8   * 如符号"/"不是Token，注释后的内容也不是，而"\\"是Token。单引号和双引号以及其中的内容
9   * 统计文章字符数的程序，不是简单的统计Token数就万事大吉，因为字符数不等于Token。按照
10  * 引号中的内容就算是10页也算一个Token。如果希望引号和引号中的内容都算作Token，应该调
11  * st.ordinaryChar('\'');
12  * st.ordinaryChar('\"');
13  */
14 public class StreamTokenizerExample {
15
16     /**
17     * 统计字符数
18     * @param fileName 文件名
19     * @return 字符数
20     */
21 public static void main(String[] args) {
22     String fileName = "D:/files/copy1.txt";
23     StreamTokenizerExample.statis(fileName);
24 }
25 public static long statis(String fileName) {
26
27     FileReader fileReader = null;
28     try {
29         fileReader = new FileReader(fileName);
30         //创建分析给定字符流的标记生成器
31         StreamTokenizer st = new StreamTokenizer(new BufferedReader(
32             fileReader));
33
34         //ordinaryChar方法指定字符参数在此标记生成器中是“普通”字符。
35         //下面指定单引号、双引号和注释符号是普通字符
36         st.ordinaryChar('\'');
37         st.ordinaryChar('\"');
38         st.ordinaryChar('/');
39

```

```

40     String s;
41     int numberSum = 0;
42     int wordSum = 0;
43     int symbolSum = 0;
44     int total = 0;
45     //nextToken方法读取下一个Token.
46     //TT_EOF指示已读到流末尾的常量。
47     while (st.nextToken() !=StreamTokenizer.TT_EOF) {
48         //在调用 nextToken 方法之后，ttype字段将包含刚读取的标记的类型
49         switch (st.ttype) {
50             //TT_EOL指示已读到行末尾的常量。
51             case StreamTokenizer.TT_EOL:
52                 break;
53             //TT_NUMBER指示已读到一个数字标记的常量
54             case StreamTokenizer.TT_NUMBER:
55                 //如果当前标记是一个数字，nval字段将包含该数字的值
56                 s = String.valueOf((st.nval));
57                 System.out.println("数字有: "+s);
58                 numberSum ++;
59                 break;
60             //TT_WORD指示已读到一个文字标记的常量
61             case StreamTokenizer.TT_WORD:
62                 //如果当前标记是一个文字标记，sval字段包含一个给出该文字标记的字符
63                 s = st.sval;
64                 System.out.println("单词有: "+s);
65                 wordSum ++;
66                 break;
67             default:
68                 //如果以上3中类型都不是，则为英文的标点符号
69                 s = String.valueOf((char) st.ttype);
70                 System.out.println("标点有: "+s);
71                 symbolSum ++;
72             }
73         }
74         System.out.println("数字有 " + numberSum+"个");
75         System.out.println("单词有 " + wordSum+"个");
76         System.out.println("标点符号有: " + symbolSum+"个");
77         total = symbolSum + numberSum +wordSum;
78         System.out.println("Total = " + total);
79         return total;
80     } catch (Exception e) {
81         e.printStackTrace();
82         return -1;
83     } finally {
84         if (fileReader != null) {
85             try {
86                 fileReader.close();
87             } catch (IOException e1) {
88             }
89         }
90     }
91 }
92
93
94 }

```

顶

4

踩

0

• [▲ 上一篇](#) 务实java基础之集合总结

相关文章推荐

- Java基础(15) : IO流—掌握对象序列化和反序列化...
- 黑马程序员—Java基础学习笔记之IO流
- Java基础(14) : IO流—理解I/O概念和掌握相关类的..
- 黑马程序员——Java基础之IO流 (一)
- 【java基础】IO流的操作分析
- java 从零开始, 学习笔记之基础入门<IO流> (十...
- JAVA基础-IO流
- 学习JAVA基础之IO
- 务实java基础之集合总结
- java io (字符、字节流) 基础

猜你在找

- 【直播】机器学习&数据挖掘7周实训--韦玮
- 【直播】3小时掌握Docker最佳实战-徐西宁
- 【直播】计算机视觉原理及实战--屈教授
- 【直播】机器学习之矩阵--黄博士
- 【直播】机器学习之凸优化--马博士
- 【套餐】系统集成项目管理工程师顺利通关--徐朋
- 【套餐】机器学习系列套餐 (算法+实战) --唐宇迪
- 【套餐】微信订阅号+服务号Java版 v2.0--翟东平
- 【套餐】微信订阅号+服务号Java版 v2.0--翟东平
- 【套餐】Javascript 设计模式实战--曾亮

查看评论



lostsky11

3楼 2小时前发表 评论

夯实, 除了代码, 自己的母语也要夯实基础



ricardoMye

2楼 5小时前发表 评论

很好



sonofbaba

1楼 16小时前发表 评论

不错

发表评论

用户名: qq_36596145

评论内容:



提交

* 以上用户言论只代表其个人观点, 不代表CSDN网站的观点或立场

[公司简介](#) | [招贤纳士](#) | [广告服务](#) | [联系方式](#) | [版权声明](#) | [法律顾问](#) | [问题报告](#) | [合作伙伴](#) | [论坛反馈](#)

网站客服 杂志客服 微博客服 webmaster@csdn.net 400-660-0108 | 北京创新乐知信息技术有限公司 版权所有 | 江苏知之为计算机有限公司 |

江苏乐知网络技术有限公司

京 ICP 证 09002463 号 | Copyright © 1999-2017, CSDN.NET, All Rights Reserved