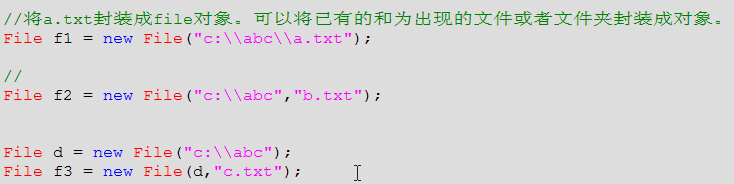


创建File文件的3种方式：





为了目录分隔符可以跨平台应该使用File文件的属性，如下：



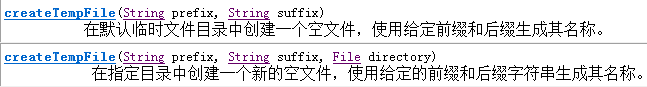
# File类常见方法

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，输出流对象一建立则会创建文件，而且如果文件已经存在，会覆盖。

创建文件还有创建临时文件的相关方法：



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。即使出现异常这个方法也会执行把文件删除。执行这个方法的时候文件不会被删除，等到程序退出的时候文件才会被删除。

3，判断。

boolean canExcute(): 文件存在就会返回true，不论是否是exe文件，如txt文件，只要是已经存在的就会返回true，不存在返回false

boolean exists() :文件是否存在.

isFile():

isDirectory();

isHidden(); // 判断是否是隐藏的，Java的IO流无法读写隐藏的文件，所以最好判断一下是否是隐藏的。

isAbsolute();// 判断是否是绝对路径，如：new File(“abc.txt”)这是相对路径，new File(“d:\\abc.txt”)这是绝对路径。这个方法即使文件或目录不存在也可以判断，因为通过字符串中的路径分隔符等信息就可以判断。

4，获取信息。

getName(): // 获取文件名

getParent(): // 获取父目录， getParent()+getName() == getAbsolutePath();注意：new File(“abc.txt”).getParent()方法将返回null，因为在不同的位置运行abc.txt所在的相对目录就不一样，所以无法确定其父目录就返回null。new File(“aa\\abc.txt”).getParent()方法返回”abc”，像File(“abc.txt”).getParent()这样的会返回null。

getPath(): new File(path); 这个path是什么就返回什么，不论是否是相对路径还是绝对路径

getAbsolutePath() new File(path);这个path无论是否是绝对路径都会返回绝对路径。

long lastModified() 最后修改时间

long length() 文件大小，它跟输入流的[**available**](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/io/InputStream.html#available())()方法类似，但是available方法返回的是int，这是因为available方法返回的长度有限，也就是说输入流指定了一次最多只能读取多少，不论文件多大，返回的available都不会超过这个数，所以如果文件很大的话用available()来判断文件大小就不准确了。

renameTo（）类似剪切，并且重命名

class FileDemo

{

public static void main(String[] args) throws IOException

{

method\_5();

}

public static void method\_5()

{

File f1 = new File("c:\\Test.java");

File f2 = new File("d:\\hahah.java");

sop("rename:"+f2.renameTo(f1)); // 这个方法类似于剪切，它会把源文件移动到目标文件所在的目录并且名字改为和目标文件一样。

}

public static void method\_4()

{

File f = new File("file.txt");

sop("path:"+f.getPath());

sop("abspath:"+f.getAbsolutePath());

sop("parent:"+f.getParent());//该方法返回的是绝对路径中的父目录。如果获取的是相对路径，返回null。

//如果相对路径中有上一层目录那么该目录就是返回结果。

}

public static void method\_3()throws IOException

{

File f = new File("d:\\java1223\\day20\\file2.txt");

//f.createNewFile();

//f.mkdir();

//记住在判断文件对象是否是文件或者目的时，必须要先判断该文件对象封装的内容是否存在。如果不存在则都是返回false

//通过exists判断。

sop("dir:"+f.isDirectory()); // 如果目录不存在，则虽然它是一个目录也返回false

sop("file:"+f.isFile()); // 如果文件不存在，则虽然它是一个文件也返回false

// 为什么要先判断是否存在呢？如：File(“abc.txt“)，调用createNewFile()方法创建的则是文件，如果调用mkdir()方法，则创建的是一个名为abc.txt的文件夹，所以说从名字看是看不出是文件还是目录的，只有它存在已后才能去判断它是文件还是目录。

sop(f.isAbsolute());

}

public static void method\_2()

{

File f = new File("file.txt");

//sop("exists:"+f.exists());

//sop("execute:"+f.canExecute());

//创建文件夹

File dir = new File("abc\\kkk\\a\\a\\dd\\ee\\qq\\aaa");

sop("mkdir:"+dir.mkdirs());

}

public static void method\_1()throws IOException

{

File f = new File("file.txt");

// sop("create:"+f.createNewFile());

//sop("delete:"+f.delete());

}

//创建File对象

public static void consMethod() {

//将a.txt封装成file对象。可以将已有的和为出现的文件或者文件夹封装成对象。

File f1 = new File("a.txt");

//

File f2 = new File("c:\\abc","b.txt");

File d = new File("c:\\abc");

File f3 = new File(d,"c.txt");

sop("f1:"+f1);

sop("f2:"+f2);

sop("f3:"+f3);

File f4 = new File("c:"+File.separator+"abc"+File.separator+"zzz"+File.separator+"a.txt");

}

public static void sop(Object obj) {

System.out.println(obj);

}

}

# listRoots()

返回可用的盘符列表，如C:\\、D:\\、E：\\，应用场景，如软件要装到最后一个盘符。

例子：

public static void listRootsDemo(){

File[] files = File.listRoots();

for(File f : files){

System.out.println(f);

}

}

# list()、[listFiles](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/io/File.html#listFiles())()

返回该目录下的所有目录和文件的名字

public static void listDemo() {

File f = new File("c:\\abc.txt");

String[] names = f.list();//调用list方法的file对象必须是封装了一个目录。该目录还必须存在。

for(String name : names) {

System.out.println(name);

}

}

练习：

通过下面的方法来过滤要获取的文件，如要获取D:\\java\\目录下的所有\*.java的文件

|  |  |
| --- | --- |
| [String](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/lang/String.html)[] | [**list**](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/io/File.html#list(java.io.FilenameFilter))([FilenameFilter](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/io/FilenameFilter.html) filter)  返回一个字符串数组，这些字符串指定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](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/io/File.html)[] | [**listFiles**](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/io/File.html#listFiles(java.io.FileFilter))([FileFilter](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/io/FileFilter.html) filter)  返回抽象路径名数组，这些路径名表示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](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/io/File.html)[] | [**listFiles**](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/io/File.html#listFiles(java.io.FilenameFilter))([FilenameFilter](mk:@MSITStore:G:\1JavaWeb资料\jdk6.ZH_cn.chm::/j2se6/api/java/io/FilenameFilter.html) filter)  返回抽象路径名数组，这些路径名表示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。 |

其实我们可以自己想象一下如何过滤File，首先是获得所有的File，那list()方法已实现，那过滤的方法无法就是遍历一个一个的File，查看这个File是否是一个文件，且这个文件的名的后缀是否是.java。这个就是过滤的原理。其实FilenameFilter用的也就是这个原理，它有一个accept方法，里面的参数File dir, String name，其实就是把一个File的目录和名字拆分出来传给了我们，那们可以判断new File(dir, name)是否是一个文件且name是否包含.java，如果包含则是符合我们的文件，这里就让accept方法返回true即可。

查看FilenameFilter类：



发现这个类是一个接口，且没有子类，那只能我们去写子类了。看accept方法的说明来知道accept方法的功能，我们要按这个功能来实现这个方法，它的功能就是过滤文件，调用list(FilenameFilter)时，符合FilenameFilter中的规则的就是需要的文件，也就是如果accept方法写死返回true，则任何的文件都是符合的，如果写死返回false则所有的文件都是不符合的，所以我们实现这个方法的要求就是根据文件名来判断是否符合。

那方法里的dir和name是什么呢？可以写代码验证一下：

File file = **new** File("G:\\develop\\workroom\\eclipse\_most\_new\\ICBCFund\\src\\com\\gongyinruixin");

file.list(**new** FilenameFilter() {

@Override

**public** **boolean** accept(File dir, String name) {

System.*out*.println("dir = " + dir.getAbsolutePath() + ", name = " + name);

**return** **false**;

}

});

运行这段代码，发现dir其实就是File中定义的目录，name就是该目录下的所有目录名和文件名，了解了这个上面的需要就可以解决了：

File file = **new** File("G:\\develop\\workroom\\eclipse\_most\_new\\ICBCFund\\src\\com\\gongyinruixin");

String[] names = file.list(**new** FilenameFilter() {

@Override

**public** **boolean** accept(File dir, String name) {

// 如果是一个文件并且是以.java结尾，则是符合要求的File

**return** (**new** File(dir, name).isFile() && name.endsWith(".java"));

}

});

**for** (String name : names) {

System.*out*.println(name);

}

**例出所有的文件和文件夹，包括子目录的**

**public** **static** **void** main(String[] args) **throws** IOException {

File dir = **new** File("G:\\develop\\workroom\\eclipse\_most\_new\\JavaDemo\\src");

*showAllFile*(dir);

}

**public** **static** **void** showAllFile(File dir) {

System.*out*.println(dir.getName());

File[] files = dir.listFiles();

**for** (File file : files) {

**if** (file.isDirectory()) {

*showAllFile*(file);

} **else** {

System.*out*.println(file.getName());

}

}

}

结果如下：

src

com

udp

demo

Main.java

TextClient.java

UdpSend.java

FileClient.java

Main.java

这样的结果没有目录层级结构，优化代码如下：

**public** **static** **void** main(String[] args) **throws** IOException {

File dir = **new** File("G:\\develop\\workroom\\eclipse\_most\_new\\JavaDemo\\src");

*showAllFile*(dir, 0);

}

**public** **static** **void** showAllFile(File dir, **int** tabCount) {

System.*out*.println(*getTab*(tabCount) + dir.getName());

File[] files = dir.listFiles();

tabCount++; // dir目录下面的目录或文件的缩进增加1

**for** (File file : files) {

**if** (file.isDirectory()) {

*showAllFile*(file, tabCount);

} **else** {

System.*out*.println(*getTab*(tabCount) + file.getName());

}

}

}

**public** **static** String getTab(**int** count) {

StringBuilder sb = **new** StringBuilder();

**for** (**int** i = 0; i < count; i++) {

sb.append('\t');

}

**return** sb.toString();

}

运行结果如下：

src

com

udp

demo

Main.java

TextClient.java

UdpSend.java

FileClient.java

Main.java

**删除所有目录和文件**

**public** **static** **void** main(String[] args) **throws** IOException {

File dir = **new** File("D:\testDir");

*deleteAllFile*(dir);

}

**public** **static** **void** deleteAllFile(File dir) {

**if** (dir == **null** || !dir.exists()) {

**throw** **new** RuntimeException("目录不存在");

}

File[] files = dir.listFiles();

**for** (File file : files) {

**if** (file.isDirectory()) {

*deleteAllFile*(file);

} **else** {

file.delete();

}

}

dir.delete();

}

**把某个目录下面的所有java文件列表保存到文件中**

**public** **static** **void** main(String[] args) **throws** IOException {

File dir = **new** File("G:\\develop\\workroom\\eclipse\_most\_new\\JavaDemo\\src");

ArrayList<File> fileList = **new** ArrayList<File>();

*fileToList*(dir, fileList);

File destFile = **new** File("D:\\javaList.txt");

*saveFileList*(destFile, fileList);

}

**public** **static** **void** fileToList(File dir, ArrayList<File> fileList) {

File[] files = dir.listFiles();

**for** (File file : files) {

**if** (file.isDirectory()) {

*fileToList*(file, fileList);

} **else** {

**if** (file.getName().endsWith(".java")) {

fileList.add(file);

}

}

}

}

**public** **static** **void** saveFileList(File destFile, ArrayList<File> fileList) **throws** IOException {

BufferedWriter bw = **new** BufferedWriter(**new** FileWriter(destFile));

**for** (File file : fileList) {

bw.write(file.getAbsolutePath());

bw.newLine();

bw.flush();

}

bw.close();

}

# Properties

Properties是hashtable的子类。

也就是说它具备map集合的特点。而且它里面存储的键值对都是字符串。

是集合中和IO技术相结合的集合容器。

该对象的特点：可以用于键值对形式的配置文件。

那么在加载数据时，需要数据有固定格式：键=值。

// 设置和获取元素。

public static void setAndGet() {

Properties prop = new Properties();

prop.setProperty("zhangsan","30");

prop.setProperty("lisi","39");

//System.out.println(prop);

String value = prop.getProperty("lisi");

//System.out.println(value);

prop.setProperty("lisi",89+"");

Set<String> names = prop.stringPropertyNames();

for(String s : names) {

System.out.println(s+":"+prop.getProperty(s));

}

}

演示如何将流中的数据存储到集合中，即Properties中的load方法的原理

//想要将info.txt中键值数据存到集合中进行操作。（注：配置文件的后缀名不一定是.prop，这个后缀可以随意起的）

/\*

1,用一个流和info.txt文件关联。

2，读取一行数据，将该行数据用"="进行切割。

3，等号左边作为键，右边作为值。存入到Properties集合中即可。

\*/

public static void method\_1()throws IOException

{

BufferedReader bufr = new BufferedReader(new FileReader("info.txt"));

String line = null;

Properties prop = new Properties();

while((line=bufr.readLine())!=null)

{

String[] arr = line.split("=");

///System.out.println(arr[0]+"...."+arr[1]);

prop.setProperty(arr[0],arr[1]);

}

bufr.close();

System.out.println(prop);

}

load方法演示

public static void loadDemo()throws IOException

{

Properties prop = new Properties();

FileInputStream fis = new FileInputStream("info.txt");

//将流中的数据加载进集合。

prop.load(fis);

prop.setProperty("wangwu","39"); // 这里设置改变了wangwu的年龄，但是改变的只是内存中的，要想改变配置文件里的值必须调用Properties的store方法进行存储。

FileOutputStream fos = new FileOutputStream("info.txt");

prop.store(fos,"haha");// “haha”是一个注释信息，可有可无。

// System.out.println(prop);

prop.list(System.out);// 将属性列表输出到控制台

fos.close();

fis.close();

}

练习：限制程序运行次数。当运行次数到达5次时，给出，请您注册的提示。并不再让该程序执行。

/\*

用于记录应用程序运行次数。

如果使用次数已到，那么给出注册提示。

很容易想到的是:计数器。

可是该计数器定义在程序中，随着程序的运行而在内存中存在，并进行自增。

可是随着该应用程序的退出，该计数器也在内存中消失了。

下一次在启动该程序，又重新开始从0计数。

这样不是我们想要的。

程序即使结束，该计数器的值也存在。

下次程序启动在会先加载该计数器的值并加1后在重新存储起来。

所以要建立一个配置文件。用于记录该软件的使用次数。

该配置文件使用键值对的形式。

这样便于阅读数据，并操作数据。

键值对数据是map集合。

数据是以文件形式存储，使用io技术。

那么map+io -->properties.

配置文件可以实现应用程序数据的共享。

\*/

import java.io.\*;

import java.util.\*;

class RunCount

{

public static void main(String[] args) throws IOException

{

Properties prop = new Properties();

File file = new File("count.ini");

if(!file.exists())

file.createNewFile();

FileInputStream fis = new FileInputStream(file);

prop.load(fis);

int count = 0;

String value = prop.getProperty("time");

if(value!=null)

{

count = Integer.parseInt(value);

if(count>=5)

{

System.out.println("您好，使用次数已到，拿钱!");

return ;

}

}

count++;

prop.setProperty("time",count+"");

FileOutputStream fos = new FileOutputStream(file);

prop.store(fos,"");

fos.close();

fis.close();

}

}

# Properties配置文件和XML配置文件的区别

<persons>

<person id="001">

<name>zhagnsan</name>

<age>30</age>

<address>北京</address>

</person>

<person id="002">

<name>lisi</name>

<age>25</age>

<address>上海</address>

</person>

</persons>

Xml配置文件可以配置比较复杂的配置，而Properties（键值对）只能配置比较简单的，如上面的xml配置，如果使用键值对的方式来配置就比较难完成了。

# IO包中的其他类

笔记见第21天文档中