第17天 IO流

今日内容介绍

Properties集合

序列化流与反序列化流

打印流

commons-IO

# Properties类

## Properties类介绍

Properties 类表示了一个持久的属性集。Properties 可保存在流中或从流中加载。属性列表中每个键及其对应值都是一个字符串。

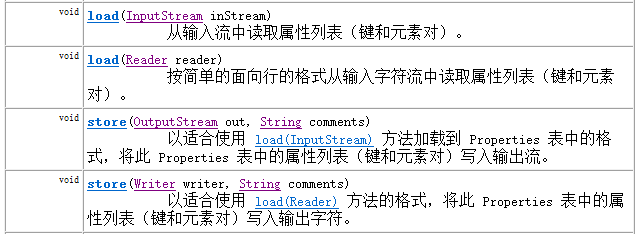
特点：

1、Hashtable的子类，map集合中的方法都可以用。

2、该集合没有泛型。键值都是字符串。

3、它是一个可以持久化的属性集。键值可以存储到集合中，也可以存储到持久化的设备(硬盘、U盘、光盘)上。键值的来源也可以是持久化的设备。

4、有和流技术相结合的方法。



load(InputStream) 把指定流所对应的文件中的数据，读取出来，保存到Propertie集合中

load(Reader)

store(OutputStream,commonts)把集合中的数据，保存到指定的流所对应的文件中，参数commonts代表对描述信息

stroe(Writer,comments);

代码演示：

/\*

\*

\* Properties集合，它是唯一一个能与IO流交互的集合

\*

\* 需求：向Properties集合中添加元素，并遍历

\*

\* 方法：

\* public Object setProperty(String key, String value)调用 Hashtable 的方法 put。

\* public Set<String> stringPropertyNames()返回此属性列表中的键集，

\* public String getProperty(String key)用指定的键在此属性列表中搜索属性

\*/

**public** **class** PropertiesDemo01 {

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

//创建集合对象

Properties prop = **new** Properties();

//添加元素到集合

//prop.put(key, value);

prop.setProperty("周迅", "张学友");

prop.setProperty("李小璐", "贾乃亮");

prop.setProperty("杨幂", "刘恺威");

//System.out.println(prop);//测试的使用

//遍历集合

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

**for** (String key : keys) {

//通过键 找值

//prop.get(key)

String value = prop.getProperty(key);

System.*out*.println(key+"==" +value);

}

}

}

## 将集合中内容存储到文件

需求：使用Properties集合，完成把集合内容存储到IO流所对应文件中的操作

分析：

1，创建Properties集合

2，添加元素到集合

3，创建流

4，把集合中的数据存储到流所对应的文件中

stroe(Writer,comments)

store(OutputStream,commonts)

把集合中的数据，保存到指定的流所对应的文件中，参数commonts代表对描述信息

5，关闭流

代码演示：

**public** **class** PropertiesDemo02 {

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

//1，创建Properties集合

Properties prop = **new** Properties();

//2，添加元素到集合

prop.setProperty("周迅", "张学友");

prop.setProperty("李小璐", "贾乃亮");

prop.setProperty("杨幂", "刘恺威");

//3，创建流

FileWriter out = **new** FileWriter("prop.properties");

//4，把集合中的数据存储到流所对应的文件中

prop.store(out, "save data");

//5，关闭流

out.close();

}

}

## 读取文件中的数据，并保存到集合

需求：从属性集文件prop.properties 中取出数据，保存到集合中

分析：

1，创建集合

2，创建流对象

3,把流所对应文件中的数据 读取到集合中

load(InputStream) 把指定流所对应的文件中的数据，读取出来，保存到Propertie集合中

load(Reader)

4,关闭流

5,显示集合中的数据

代码演示：

**public** **class** PropertiesDemo03 {

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

//1，创建集合

Properties prop = **new** Properties();

//2，创建流对象

FileInputStream in = **new** FileInputStream("prop.properties");

//FileReader in = new FileReader("prop.properties");

//3,把流所对应文件中的数据 读取到集合中

prop.load(in);

//4,关闭流

in.close();

//5,显示集合中的数据

System.*out*.println(prop);

}

}

注意：使用字符流FileReader就可以完成文件中的中文读取操作了

# 序列化流与反序列化流

用于从流中读取对象的

操作流 ObjectInputStream 称为 反序列化流

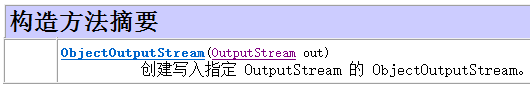
用于向流中写入对象的操作流 ObjectOutputStream 称为 序列化流

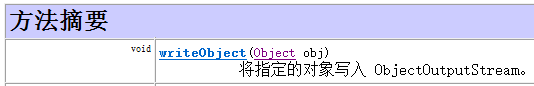
特点：用于操作对象。可以将对象写入到文件中，也可以从文件中读取对象。

## 对象序列化流ObjectOutputStream

ObjectOutputStream 将 Java 对象的基本数据类型和图形写入 OutputStream。可以使用 ObjectInputStream 读取（重构）对象。通过在流中使用文件可以实现对象的持久存储。

注意：只能将支持 java.io.Serializable 接口的对象写入流中





代码演示：

**public** **class** ObjectStreamDemo {

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

/\*

\* 将一个对象存储到持久化(硬盘)的设备上。

\*/

*writeObj*();//对象的序列化。

}

**public** **static** **void** writeObj() **throws** IOException {

//1,明确存储对象的文件。

FileOutputStream fos = **new** FileOutputStream("tempfile\\obj.object");

//2，给操作文件对象加入写入对象功能。

ObjectOutputStream oos = **new** ObjectOutputStream(fos);

//3，调用了写入对象的方法。

oos.writeObject(**new** Person("wangcai",20));

//关闭资源。

oos.close();

}

}

Person类

**public** **class** Person **implements** Serializable {

**private** String *name*;

**private** **int** age;

**public** Person() {

**super**();

}

**public** Person(String name, **int** age) {

**super**();

**this**.*name* = name;

**this**.age = age;

}

**public** String getName() {

**return** *name*;

}

**public** **void** setName(String name) {

**this**.*name* = name;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

@Override

**public** String toString() {

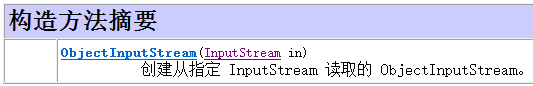
**return** "Person [name=" + *name* + ", age=" + age + "]";

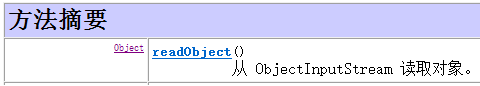
}

}

## 对象反序列化流ObjectInputStream

ObjectInputStream 对以前使用 ObjectOutputStream 写入的基本数据和对象进行反序列化。支持 java.io.Serializable接口的对象才能从流读取。





代码演示

**public** **class** ObjectStreamDemo {

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

*readObj*();//对象的反序列化。

}

**public** **static** **void** readObj() **throws** IOException, ClassNotFoundException {

//1,定义流对象关联存储了对象文件。

FileInputStream fis = **new** FileInputStream("tempfile\\obj.object");

//2,建立用于读取对象的功能对象。

ObjectInputStream ois = **new** ObjectInputStream(fis);

Person obj = (Person)ois.readObject();

System.*out*.println(obj.toString());

}

}

## 序列化接口

当一个对象要能被序列化，这个对象所属的类必须实现Serializable接口。否则会发生异常NotSerializableException异常。

同时当反序列化对象时，如果对象所属的class文件在序列化之后进行的修改，那么进行反序列化也会发生异常InvalidClassException。发生这个异常的原因如下：

该类的序列版本号与从流中读取的类描述符的版本号不匹配

该类包含未知数据类型

该类没有可访问的无参数构造方法

Serializable标记接口。该接口给需要序列化的类，提供了一个序列版本号。serialVersionUID. 该版本号的目的在于验证序列化的对象和对应类是否版本匹配。

代码修改如下，修改后再次写入对象，读取对象测试

**public** **class** Person **implements** Serializable {

//给类显示声明一个序列版本号。

**private** **static** **final** **long** *serialVersionUID* = 1L;

**private** String *name*;

**private** **int** age;

**public** Person() {

**super**();

}

**public** Person(String name, **int** age) {

**super**();

**this**.*name* = name;

**this**.age = age;

}

**public** String getName() {

**return** *name*;

}

**public** **void** setName(String name) {

**this**.*name* = name;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

@Override

**public** String toString() {

**return** "Person [name=" + *name* + ", age=" + age + "]";

}

}

## 瞬态关键字transient

当一个类的对象需要被序列化时，某些属性不需要被序列化，这时不需要序列化的属性可以使用关键字transient修饰。只要被transient修饰了，序列化时这个属性就不会琲序列化了。

同时静态修饰也不会被序列化，因为序列化是把对象数据进行持久化存储，而静态的属于类加载时的数据，不会被序列化。

代码修改如下，修改后再次写入对象，读取对象测试

**public** **class** Person **implements** Serializable {

/\*

\* 给类显示声明一个序列版本号。

\*/

**private** **static** **final** **long** *serialVersionUID* = 1L;

**private** **static** String *name*;

**private** **transient**/\*瞬态\*/ **int** age;

**public** Person() {

**super**();

}

**public** Person(String name, **int** age) {

**super**();

**this**.*name* = name;

**this**.age = age;

}

**public** String getName() {

**return** *name*;

}

**public** **void** setName(String name) {

**this**.*name* = name;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

@Override

**public** String toString() {

**return** "Person [name=" + *name* + ", age=" + age + "]";

}

}

# 打印流

## 打印流的概述

打印流添加输出数据的功能，使它们能够方便地打印各种数据值表示形式.

打印流根据流的分类：

字节打印流 PrintStream

字符打印流 PrintWriter

方法：

void print(String str): 输出任意类型的数据，

void println(String str): 输出任意类型的数据，自动写入换行操作

代码演示：

/\*

\* 需求：把指定的数据，写入到printFile.txt文件中

\*

\* 分析：

\* 1，创建流

\* 2，写数据

\* 3，关闭流

\*/

**public** **class** PrintWriterDemo {

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

//创建流

//PrintWriter out = new PrintWriter(new FileWriter("printFile.txt"));

PrintWriter out = **new** PrintWriter("printFile.txt");

//2，写数据

**for** (**int** i=0; i<5; i++) {

out.println("helloWorld");

}

//3,关闭流

out.close();

}

}

## 打印流完成数据自动刷新

可以通过构造方法，完成文件数据的自动刷新功能

构造方法：

开启文件自动刷新写入功能

public PrintWriter(OutputStream out, boolean autoFlush)

public PrintWriter(Writer out, boolean autoFlush)

代码演示：

/\*

\* 分析：

\* 1，创建流

\* 2，写数据

\*/

**public** **class** PrintWriterDemo2 {

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

//创建流

PrintWriter out = **new** PrintWriter(**new** FileWriter("printFile.txt"), **true**);

//2，写数据

**for** (**int** i=0; i<5; i++) {

out.println("helloWorld");

}

//3,关闭流

out.close();

}

}

# commons-IO

## 导入classpath

加入classpath的第三方jar包内的class文件才能在项目中使用

创建lib文件夹

将commons-io.jar拷贝到lib文件夹

右键点击commons-io.jar，Build Path→Add to Build Path

## FilenameUtils

这个工具类是用来处理文件名（译者注：包含文件路径）的，他可以轻松解决不同操作系统文件名称规范不同的问题

常用方法：

getExtension(String path)：获取文件的扩展名；

getName()：获取文件名；

isExtension(String fileName,String ext)：判断fileName是否是ext后缀名；

## FileUtils

提供文件操作（移动文件，读取文件，检查文件是否存在等等）的方法。

常用方法：

readFileToString(File file)：读取文件内容，并返回一个String；

writeStringToFile(File file，String content)：将内容content写入到file中；

copyDirectoryToDirectory(File srcDir,File destDir);文件夹复制

copyFile(File srcFile,File destFile);文件夹复制

代码演示：

/\*

\* 完成文件的复制

\*/

**public** **class** CommonsIODemo01 {

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

//method1("D:\\test.avi", "D:\\copy.avi");

//通过Commons-IO完成了文件复制的功能

FileUtils.*copyFile*(**new** File("D:\\test.avi"), **new** File("D:\\copy.avi"));

}

//文件的复制

**private** **static** **void** method1(String src, String dest) **throws** IOException {

//1,指定数据源

BufferedInputStream in = **new** BufferedInputStream(**new** FileInputStream(src));

//2,指定目的地

BufferedOutputStream out = **new** BufferedOutputStream(**new** FileOutputStream(dest));

//3，读

**byte**[] buffer = **new** **byte**[1024];

**int** len = -1;

**while** ( (len = in.read(buffer)) != -1) {

//4,写

out.write(buffer, 0, len);

}

//5，关闭流

in.close();

out.close();

}

}

/\*

\* 完成文件、文件夹的复制

\*/

**public** **class** CommonsIODemo02 {

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

//通过Commons-IO完成了文件复制的功能

FileUtils.*copyFile*(**new** File("D:\\test.avi"), **new** File("D:\\copy.avi"));

//通过Commons-IO完成了文件夹复制的功能

//D:\基础班 复制到 C:\\abc文件夹下

FileUtils.*copyDirectoryToDirectory*(**new** File("D:\\基础班"), **new** File("C:\\abc"));

}

}

# 总结

## IO流总结

字节流

* + 字节输入流 InputStream
    - FileInputStream 操作文件的字节输入流
    - BufferedInputStream高效的字节输入流
    - ObjectInputStream 反序列化流
  + 字节输出流 OutputStram
    - FileOutputStream 操作文件的字节输出流
    - BufferedOutputStream 高效的字节输出流
    - ObjectOuputStream 序列化流
    - PrintStream 字节打印流

字符流

* + 字符输入流 Reader
    - FileReader 操作文件的字符输入流
    - BufferedReader 高效的字符输入流
    - InputStreamReader 输入操作的转换流(把字节流封装成字符流)
  + 字符输出流 Writer
    - FileWriter 操作文件的字符输出流
    - BufferedWriter 高效的字符输出流
    - OutputStreamWriter 输出操作的转换流(把字节流封装成字符流)
    - PrintWriter 字符打印流

方法：

* + 读数据方法：
    - read() 一次读一个字节或字符的方法
    - read(byte[] char[]) 一次读一个数组数据的方法
    - readLine() 一次读一行字符串的方法(BufferedReader类特有方法)
    - readObject() 从流中读取对象(ObjectInputStream特有方法)
  + 写数据方法：
    - write(int) 一次写一个字节或字符到文件中
    - write(byte[] char[]) 一次写一个数组数据到文件中
    - write(String) 一次写一个字符串内容到文件中
    - writeObject(Object ) 写对象到流中(ObjectOutputStream类特有方法)
    - newLine() 写一个换行符号(BufferedWriter类特有方法)

向文件中写入数据的过程

1，创建输出流对象

2，写数据到文件

3，关闭输出流

从文件中读数据的过程

1. 创建输入流对象
2. 从文件中读数据
3. 关闭输入流

文件复制的过程

1. 创建输入流（数据源）
2. 创建输出流（目的地）
3. 从输入流中读数据
4. 通过输出流，把数据写入目的地
5. 关闭流

File类

* + 方法
    - 获取文件名称 getName()
    - 获取文件绝对路径 getAbsolutePath()
    - 获取文件大小 length()
    - 获取当前文件夹中所有File对象 File[] listFiles()
    - 判断是否为文件 isFile()
    - 判断是否为文件夹 isDirectory()
    - 创建文件夹 mkdir() mkdirs()
    - 创建文件 createNewFile()

异常

* + try..catch…finally捕获处理异常
  + throws 声明异常
  + throw 抛出异常对象

异常的分类

* + 编译期异常 Exception

|- 运行期异常 RuntimeException

注意：

编译期异常，必须处理，不然无法编译通过

运行期异常，程序运行过程中，产生的异常信息

Properties：Map集合的一种，它是Hashtable集合的子集合,它键与值都是String类型,它是唯一能与IO流结合使用的集合

* + 方法
    - load( InputStream in ) 从流所对应的文件中，读数据到集合中
    - load( Reader in ) 从流所对应的文件中，读数据到集合中
    - store( OutputStream out , String message ) 把集合中的数据，写入到流所对应的文件中
    - store( Writer out , String message) 把集合中的数据，写入到流所对应的文件中

实现文件内容的自动追加

* + 构造方法
  + FileOutputStream(File file, boolean append)
  + FileOutputStream(String fileName, boolean append)
  + FileWriter(File, boolean append)
  + FileWriter(String fileName, boolean append)

实现文件内容的自动刷新

* + 构造方法
  + PrintStream(OutputStream out, boolean autoFlush)
  + PrintWriter(OutputStream out, boolean autoFlush)
  + PrintWriter(Writer out, boolean autoFlush)

Commons-IO

方法

* + readFileToString(File file)：读取文件内容，并返回一个String；
  + writeStringToFile(File file，String content)：将内容content写入到file中；
  + copyDirectoryToDirectory(File srcDir,File destDir);文件夹复制
  + copyFileToDirectory (File srcFile,File destFile);文件复制