2015年8月11日09:17:55

JDK是java开发包,JRE是java运行环境，JVM是java虚拟机。安装JDK后会自带JRE，JVM也包含在JRE里面。JVM保证了java程序运行于平台无关。所有的java程序都运行在java虚拟机上。

安装完JDK后，还不能用java来编程，需要配置环境变量。主要是配置JAVA\_HOME为安装路径，还有path和classpath，path为java的exe可执行文件，后者为库文件，就是一些java包。

垃圾回收

在 Java 里面，垃圾回收是一个自动的系统行为，程序员不能控制垃圾回收的功  
能和行为。比如垃圾回收什么时候开始，什么时候结束，还有到底哪些资源需要回收等，都  
是程序员不能控制的。  
（2） ：有一些跟垃圾回收相关的方法，比如：System.gc() , 记住一点，调用这些方法，  
仅仅是在通知垃圾回收程序，至于垃圾回收程序运不运行，什么时候运行，都是无法控制的。  
（3） ：程序员可以通过设置对象为 null（后面会讲到）来标示某个对象不再被需要了，

1. 第一个java程序

public class test

{

public static void main(String[] args)

{

System.out.println("hello world");

}

}

Java的mian方法和C#的Main方法有区别。

Java的是小写m，C#是大写M,参数和C#控制台类似。C#的string可以小写。而java不行，必须大写String.

2015年9月1日10:44:29

1. java的字符串的操作
2. 字符串转换为char数组

用toCharArray方法

String str1= "hello";

**char** c[] = str1.toCharArray();

b. 将字符数组转换为string直接将数组放在构造函数的参数里，也可以指定转换部分数组为字符串

String str2= **new** String(c);

String str3 = **new** String(c,0,3); //部分转换

* 1. 将字符串转换为byte数组

**byte** b[] = str1.getBytes() ; // 将字符串变为byte数组

System.***out***.println(**new** String(b)) ; // 将全部的byte数组变为字符串

System.***out***.println(**new** String(b,1,3)) ; // 将部分的byte数组变为字符串

d. 取出某一个字符charAt

String str1 = "hello" ; // 定义String对象

System.out.println(str1.charAt(3)) ; // 取出字符串中第四个字符

e.求字符串的长度，用length()方法(估计是对属性用了getter和setter封装)

str1.length()

f. 查找

System.out.println(str1.indexOf("c")) ; // 查到返回位置

System.out.println(str1.indexOf("c",3)) ; // 查到返回位置，从第4个位置开始查

//找

System.out.println(str1.indexOf("x")) ; // 没有查到返回-1

g． 去掉空格

String str4 = " hello " ; // 定义字符串

System.***out***.println(str4.trim()) ; // 去掉左右空格后输出

h. 截取字符串

String str1 = "hello world" ; // 定义字符串

System.out.println(str1.substring(6)) ; // 从第7个位置开始截取

System.out.println(str1.substring(0,5)) ; // 截取0~5个位置的内容

1. 分割字符串

String str6 = "hello world" ; // 定义字符串

String s[] = str6.split(" ") ; // 按空格进行字符串的拆分

**for**(**int** i=0;i<s.length;i++){ // 循环输出

System.***out***.println(s[i]) ;

}

j. 转换大小写

System.out.println("将\"hello world\"转成大写：" + "hello world".toUpperCase()) ;

System.out.println("将\"HELLO WORLD\"转成小写：" + "HELLO WORLD".toLowerCase()) ;

k. 判断开头和结尾

String str1 = "\*\*HELLO" ; // 定义字符串

String str2 = "HELLO\*\*" ; // 定义字符串

if(str1.startsWith("\*\*")){ // 判断是否以“\*\*”开头

System.out.println("（\*\*HELLO）以\*\*开头") ;

}

if(str2.endsWith("\*\*")){ // 判断是否以“\*\*”结尾

System.out.println("（HELLO\*\*）以\*\*结尾") ;

}

l.判断内容是否相等

String str1 = "HELLO" ; // 定义字符串

String str2 = "hello" ; // 定义字符串

System.out.println("\"HELLO\" equals \"hello\" " + str1.equals(str2)) ;

System.out.println("\"HELLO\" equalsIgnoreCase \"hello\" "

+ str1.equalsIgnoreCase(str2)) ; // 不区分大小写的比较

j. 替换

String str = "hello" ; // 定义字符串

String newStr = str.replaceAll("l","x") ; // 现在将所有的l替换成x

System.out.println("替换之后的结果：" + newStr) ;

2015年9月1日21:58:29

1. 引用传递

public static void main(String [] args)

{

String str1="hello";

System.out.println(str1);

fun(str1);

System.out.println(str1);

}

public static void fun(String str2)

{

Str2="world";

}

输出：

hello

hello

虽然string是引用传递，但在java中string具有内容不可变行，一旦赋值该内存空间的内容就不变，如果给变量重新赋值就会新开辟空间。在调用fun时str1和str2都指向“hello”，然偶执行str2=“world”时，str指向了新的空间，该空间内容为world。而str1指向的内存空间不变还是“hello”

如果传递的不是string，是自定义的类，就是普通的引用传递了，内容就会变。如果是int类型，则不变。这些特性和C#是一样的。

1. 类中调用本类的对象

class Demo{ // 定义Demo类

private int temp = 30 ; // 声明temp属性并封装

public void fun(Demo d2){ // 接收本类的引用

d2.temp = 50 ; // 直接通过对象调用本类的私有属性

}

public int getTemp(){ // getter

return temp ;

}

public void setTemp(int t){ // setter

temp = t ;

}

};

public class RefDemo04{

public static void main(String args[]){

Demo d1 = new Demo() ; // 实例化Demo对象

d1.setTemp(50) ; // 只能通过setter方法修改内容

d1.fun(d1) ; // 此处把Demo的对象传回到自己的类中

System.out.println("temp = " + d1.getTemp()) ;

}

};

2015年9月2日10:47:53

1. Main方法可以输入多个参数

public class StaticDemo08{

public static void main(String args[]){

if(args.length!=3){ // 输入的参数如果不足3个，则出错

System.out.println("输入的参数不足三个，程序退出~") ;

System.exit(1) ; // 直接退出此程序

}

for(int i=0;i<args.length;i++){ // 循环输出输入的参数

System.out.println("第"+(i+1)+"个参数：" + args[i]) ;

}

}

};

代码块（普通块，构造块，静态块，）

构造块，构造块优先于构造方法之前执行，直接在类用用花括号括起就行。

class Demo{

{ // 直接在类中编写代码块，称为构造块

System.out.println("1、构造块。") ;

}

public Demo(){ // 定义构造方法

System.out.println("2、构造方法。") ;

}

};

public class CodeDemo02{

public static void main(String args[]){

new Demo() ; // 实例化对象

new Demo() ; // 实例化对象

new Demo() ; // 实例化对象

}

};

静态块，优先于主方法执行，在类中，优先于构造方法执行。用于为静态属性初始化。

class Demo{

{ // 直接在类中编写代码块，称为构造块

System.out.println("1、构造块。") ;

}

static{ // 使用static，称为静态代码块

System.out.println("0、静态代码块") ;

}

public Demo(){ // 定义构造方法

System.out.println("2、构造方法。") ;

}

};

public class CodeDemo03{

static{ // 在主方法所在的类中定义静态块

System.out.println("在主方法所在类中定义的代码块") ;

}

public static void main(String args[]){

new Demo() ; // 实例化对象

new Demo() ; // 实例化对象

new Demo() ; // 实例化对象

}

};

构造方法私有化

用private修饰构造方法，则不能在外部实例化对象。但是可以在类的内部实例化对象。

常常用于单例模式。

class Test

{

private static Test t = new Test();

private Test()

{}

public static Test getInstance()

{

if(t==null)

{

t= new Test();

return t;

}

return t;

}

public void print()

{

System.out.println("hello");

}

}

* + 1. 09:47:18

1. 子类重写父类的方法时，子类方法的访问权限不能比父类低。

权限大小 public > default > private

比如 父类的方法为public 子类如果不写(default)或者写为private则报错。

1. 通过super关键字可以从子类访问父类发方法。用于子类重写了父类的方法的时候。如果不加super,则成为了自己调用自己，成为了递归的死循环。

使用super也不一定要复写之后才使用。加上它可以明确表示是由父类来的方法。

3． 如果父类中的方法是private，子类为default，则不存在复写，调用super会出错。父类和子类中的这两个方法分别属于各自的。

2015-9-22 20:37:49

1. final 关键字

final关键字修饰的类不能被继承，修饰的方法不能被复写，修饰的变量不能被修改。

1. 用static加final 声明的常量为全局常量。

2015年10月14日20:51:25

1. Instanceof关键字

Instanceof运算符是用来在运行时指出对象是否是特定类的一个实例。通过返回一个布尔值来指出。

用于增加验证，以保证转型时，不会发生ClassCastException

 用法：  
result = object instanceof class  
参数：  
Result：布尔类型。  
Object：必选项。任意对象表达式。  
Class：必选项。任意已定义的对象类。

如果 B类继承A类

A a= new B();

那么a即是A的实例，也是B的实例。

2015年12月4日22:05:02

1. 子类和父类间的转换

B类继承A类

A a = new B();

那么a这个对象只能访问A类中的方法，但是如果B类有同名方法，则会调用同名B类中的方法，B类中独有的，a这个对象是访问不了的。

2015年12月7日20:08:55

1. 关于引用传递和值传递

String和int都是值传递，将其作为参数传递时都会开辟空间创建副本，也就是在一个方法中改变它的值是不能改变的。

类是引用传递，在一个方法的参数中传递一个类的对象，会改变原来对象的变量的值。不管这个类中的变量是string还是int都会变。

class Ref2

{

String temp="hello";

}

**public** **static** **void** tell2(Ref2 r2)

{

r2.temp="world";

}

主方法：

Ref2 r2 = **new** Ref2();

r2.temp="java";

System.***out***.println(r2.temp);

*tell2*(r2);

System.***out***.println(r2.temp);

1. 和C#一样 private修饰的变量，在类外是不能访问的，包括对象
2. 关于static关键字

在C#中static关键字修饰的变量，只能通过类名来访问，而不能通过对象来访问

但是在java中就不一样了，类名和对象都可以访问static修饰的变量，不管创建了几个对象，一旦有对象访问了这个static变量，并且修改了值，或者类名直接放了这个变量，修改了了值，那么以最后一次修改为准，所有对象的这个static变量都是相同的值。因此在java中static修饰的变量具有全局性，任何一个对象修改了它，其余对象再访问都是修改后的，也就是说创建对象的时候，不会为它开辟新的空间，它不属于某个固定对象。

专业解释： 1. 使用static声明属性

Static声明全局属性

2 使用static声明方法

直接通过类名调用

3 注意点

使用static方法的时候，只能访问static声明的方法和属性，而非static声明的属性和方法不能访问。

Static修饰的属性和方法最好用类名来调用（在java中也可以用对象来调用），否则会报警告。

1. 关于this关键字

This关键字用于表示本类对象，在本类中调用变量，在java和C#中是没有区别的

但是this用于调用重载的构造方法时原理一样，写法不一样。

在C#中this写在构造方法的后面并加冒号：

在java中this写在构造方法内部首行

C#： public string name;

public int age;

public People()

{

}

public People(string name)

{

this.name = name;

this.age = 22;

}

public People(string name, int age) :this(name)

{

this.age = age;

}

Java：

**public** People(String name)

{

**this**.name=name;

}

**public** People(String name,**int** age)

{

**this**(name);

**this**.age=age;

}

2016年6月22日10:26:17

1. Java多线程中获取当前线程用的是Thread.currentThread()

注意是currentThread()是个方法要带括号。

1. Java多线程可以用哦getName和setName的方法读取和设置线程名称，线程运行前，如果没有设置名称，那么系统会设置一个默认的名称。

2016年6月22日23:11:51

* + - 1. 一个java程序运行的时候至少启动了两个线程，一个是主线程，另一个是GC线程，及垃圾回收线程。
      2. 通过isAlive()方法来判断线程是否执行完毕。在调用start方法之前返回false，在执行过程中返回true，线程执行完返回false。
      3. 线程的强制运行

调用线程对象的join方法，如果多个线程同时在运行，那么如果某个线程调用了join方法，那么它将被强制执行，即优先执行它，其余的线程等待，直到它运行完了才继续运行。

注意要捕获InterrupetedException异常。

4. 线程的休眠

调用sleep()方法

5. 线程的中断

调用interrupt()方法,进入中断异常

6.线程的礼让

调用yield()方法

2016年6月28日16:54:56

1、 StringBuffer和StringBuilder

主要区别是StringBuffer类的方法是线程安全的，而StringBuilder不是线程安全的。

String的对象一旦创建，其内容不能再被修改，而StringBuffer和StringBuilder可以修改。

2. 线程的停止等旧方法已经过时

比如suspend，stop，resume方法已经过时，不建议使用，因为使用它们会造成死锁。

一个线程的停止是run方法运行完毕就直接停止了，不需要别的干预。一般run方法里面如果是个死循环，控制循环标志来退出循环，即结束线程。

2016年6月28日23:40:59

1.定义泛型对象的时候注意前面也要指定类名称

Point<Integer> p = new Point<Integer>() ;

2016年6月29日22:21:13

1. 泛型的通配符是？

当给一个方法传递一个泛型对象的时候，在写形参的时候，形参可以直接用类名加对象名来表示，也可以用<>指定实际的泛型类型。

比如有泛型类

class Info<T>

{

T var;

public void setVar(T var)

{

this.var=var;

}

public T getVar()

{

return this.var;

}

public String toString()

{

return this.var.toString();

}

}

方法：

public static void fun(Info temp)

{

System.out.println(temp.getVar()) ;

System.out.println(temp) ;

}

public static void fun(Info<String> temp)

{

System.out.println(temp.getVar()) ;

System.out.println(temp) ;

}

这两种方法都可以，但都没有达到泛型的特点，可以用？表示任意类型

public static void fun(Info<?> temp)

{

System.out.println(temp.getVar()) ;

System.out.println(temp) ;

}

这样传递任意类型都可以

2. 泛型的上下限

可以设置泛型类型的上下限，即它的类型继承的最高等级和最低等级的类型，比如上限类型为Number，那么该类型只能是Number类和其子类。下限表示必须是该类是父类。声明上限用extends，下限用super。

设置上限：

声明对象：类名称<? extends 类> 对象名称

定义类：[访问权限] 类名称<泛型标示 extends 类>{}

设置下限：

声明对象：类名称<? super类> 对象名称

定义类：[访问权限] 类名称<泛型标示 super 类>{}

3. 子类的泛型无法使用父类的泛型接收

比如Info<String>不能用Info<Object>来接收

2016年7月12日12:06:48

1. StringBuffer类

类型可变的字符串

(1).连接字符串用append方法，在String中是用+号。append方法可以连续使用。

public static void main(String [] args)

{

StringBuffer buf = new StringBuffer();

buf.append("hello");

buf.append(" world").append("!!!");

buf.append('\n');

buf.append('C');

buf.append(true);

fun(buf);

System.out.println(buf);

}

public static void fun(StringBuffer s)

{

s.append("guo").append("zhang");

}

(2)插入字符串用insert

StringBuffer buf = new StringBuffer();

buf.append("hello");

buf.insert(0,"world");

(3)字符串反转功能reverse

StringBuffer buf1= buf.reverse();

buf和buf1都是反转后的结果。

(4)替换指定范围的内容replace

StringBuffer buf = new StringBuffer();

buf.append("hello").append("whorld");

buf.replace(6,11,"guozhang");

(5)字符串截取substring

该方法返回值直接是String

String res= buf.substring(2,4);

截取索引2,3，不截取4。

(6)删除指定范围的内容delete，该方法返回stringBuffer

buf.delete(2,4)

buf也被修改了。

(7)查找指定的内容是否存在

使用indexof来查找，和string是一样的，如果找到，返回指定的位置，没有的话返回-1.

int index = buf.indexOf("llo");

(8)stringbuffer的应用，用在需要频繁修改字符串内容的地方。比如很多次的for循环。

(9)StringBuffer中有而string中却没有的方法，比如：

insert，delete

2. Runtime类

运行时类，是一个封装了JVM进程的类，每一个java程序，实际上都是启动了一个JVM进程，那么每一个JVM进程都是对应的一个Runtime实例，此实例是又JVM为其实例化的。

(1)获得JVM信息

Runtime run = Runtime.getRuntime(); // 通过Runtime类的静态方法进行实例化操作

System.out.println("JVM最大内存量：" + run.maxMemory()) ; // 观察最大的内存，根据机器的不同，环境也会有所不同

System.out.println("JVM空闲内存量：" + run.freeMemory()) ; // 取得程序运行的空闲内存

run.gc() ; // 进行垃圾收集，释放空间

(2)打开系统程序

Runtime run = Runtime.getRuntime();

try{

run.exec("notepad.exe");

}catch(Exception e)

{

e.printStackTrace();

}

这段代码打开了系统的记事本。

exec方法返回进程Process类的对象

调用destroy方法可以结束进程。

3. java国际化

3.1 Locale类

3.2 ResourceBundle类

此类是专门完成属性文件的读取操作的，读取的时候指定文件名称即可(此文件名称一般不需要指定后缀，后缀统一为：\*.properties),可以根据Local所指定的区域码来自动选择所需要的资源文件。

public static final ResourceBundle getBundle(String baseName)此方法就是指定所有啊操作的资源文件，此方法找到的是默认的操作系统的语言的Local对象。

public static final ResourceBundle getBundle(String baseName，Locale local)此方法也是擦操作指定的资源文件，并传入Locale对象。

public static final String getString(Strig key) 根据key获取相应的value

例子：

新建一个文本文件，命名为Message.Properties，在里面写入inof=HELLO

在同一目录下新建java程序

ResourceBundle rb = ResourceBundle.getBundle("Message");

System.out.println(rb.getString("info"));

将输出:HELLO

3.3 利用ResourceBundle和Local实现国际化

不同的资源文件要建不同的properties文件，

中文：Message\_zh\_CN.properties

英文：Message\_en\_US.properties

法文：Message\_fr\_FR.properties

记得导入包；

import java.util.ResourceBundle;

import java.util.Locale;

Locale znloc = new Locale("zh","CN");

Locale enloc = new Locale("en","US");

ResourceBundle rb1 = ResourceBundle.getBundle("Message",znloc);

ResourceBundle rb2 = ResourceBundle.getBundle("Message",enloc);

System.out.println(rb1.getString("info"));

System.out.println(rb2.getString("info"));

在中文操作系统中，输出中文是乱码，因为zh\_CN的资源文件中直接 使用的是“你好”字符串，正确的应该是 用unicode编码，转换为java认识的16进制。

使用jdk自带的转换工具C:\Program Files\Java\jdk1.8.0\_65\bin\ native2ascii.exe,不同的语言环境转换结果不同，中文抽中系统下转换结果是：\u4f60\u597d

记住：只要是中文，就必须进行转码。

4. System类

4.1获取当前时间

long time1 = System.currentTimeMillis();

4.2 获取系统属性

获取全部属性：

System.getProperties().list(System.out);

-- listing properties --

java.runtime.name=Java(TM) SE Runtime Environment

sun.boot.library.path=C:\Program Files\Java\jre1.8.0\_92\bin

java.vm.version=25.92-b14

java.vm.vendor=Oracle Corporation

java.vendor.url=http://java.oracle.com/

path.separator=;

java.vm.name=Java HotSpot(TM) 64-Bit Server VM

file.encoding.pkg=sun.io

user.script=

user.country=CN

sun.java.launcher=SUN\_STANDARD

sun.os.patch.level=

java.vm.specification.name=Java Virtual Machine Specification

user.dir=D:\javatest

java.runtime.version=1.8.0\_92-b14

java.awt.graphicsenv=sun.awt.Win32GraphicsEnvironment

java.endorsed.dirs=C:\Program Files\Java\jre1.8.0\_92\lib...

os.arch=amd64

java.io.tmpdir=C:\Users\guozhang\AppData\Local\Temp\

line.separator=

java.vm.specification.vendor=Oracle Corporation

user.variant=

os.name=Windows 10

sun.jnu.encoding=GBK

java.library.path=C:\ProgramData\Oracle\Java\javapath;C...

java.specification.name=Java Platform API Specification

java.class.version=52.0

sun.management.compiler=HotSpot 64-Bit Tiered Compilers

os.version=10.0

user.home=C:\Users\guozhang

user.timezone=

java.awt.printerjob=sun.awt.windows.WPrinterJob

file.encoding=GBK

java.specification.version=1.8

user.name=guozhang

java.class.path=.;C:\Program Files\Java\jdk1.8.0\_65\l...

java.vm.specification.version=1.8

sun.arch.data.model=64

java.home=C:\Program Files\Java\jre1.8.0\_92

sun.java.command=SystemDemo02

java.specification.vendor=Oracle Corporation

user.language=zh

awt.toolkit=sun.awt.windows.WToolkit

java.vm.info=mixed mode

java.version=1.8.0\_92

java.ext.dirs=C:\Program Files\Java\jre1.8.0\_92\lib...

sun.boot.class.path=C:\Program Files\Java\jre1.8.0\_92\lib...

sun.stderr.encoding=ms936

java.vendor=Oracle Corporation

file.separator=\

java.vendor.url.bug=http://bugreport.sun.com/bugreport/

sun.cpu.endian=little

sun.io.unicode.encoding=UnicodeLittle

sun.stdout.encoding=ms936

sun.desktop=windows

sun.cpu.isalist=amd64

获取部分属性：类似键值对来查找

获取系统版本：

System.out.println("系统版本："+System.getProperty("os.name"));

os.version

os.arch

4.3 垃圾回收

一个对象如果不使用，则肯定要等待进行垃圾收集，垃圾收集可以自动调用也可以手工调用，手工调用的时候就是调用System.gc()或者Runtime.getRuntime().gc()

class Person{

private String name;

private int age;

public Person(String name,int age)

{

this.name=name;

this.age=age;

}

public String toString()

{

return "姓名："+this.name+"年龄："+this.age;

}

public void finalize() throws Throwable{

System.out.println("对象被释放-->"+this);

}

}

public class SystemDemo02

{

public static void main(String[] args)

{

Person pr = new Person("张三",20);

pr=null;

System.gc();

}

}

对象被释放-->姓名：张三年龄：20

5. 日期操作类

5.1 Date类

需要导入包 import java.util.Date;

Date date = new Date();

System.out.println(date);

输出：Mon Jul 18 17:21:10 CST 2016

5.2 Calendar类

使用此类可以直接将日期精确到毫秒

Calendar是一个抽象类

Calendar calendar = new GregorianCalendar(); // 实例化Calendar类对象

System.out.println("YEAR: " + calendar.get(Calendar.YEAR));

System.out.println("MONTH: " + (calendar.get(Calendar.MONTH) + 1));

System.out.println("DAY\_OF\_MONTH: " + calendar.get(Calendar.DAY\_OF\_MONTH));

System.out.println("HOUR\_OF\_DAY: " + calendar.get(Calendar.HOUR\_OF\_DAY));

// System.out.println("MINUTE: " + calendar.get(Calendar.MINUTE));

System.out.println("SECOND: " + calendar.get(Calendar.SECOND));

System.out.println("MILLISECOND: " + calendar.get(Calendar.MILLISECOND)); Calendar calendar = new GregorianCanlendar();

注意，取得月份的时候要加1.

6. 日期格式化

6.1 DateFormat类

此类定义在java.text包中，是一个抽象类。

java.util.Date本身包含完整是日期信息，所以只需将这个日期格式化就可以。

DateFormat df1= null;

DateFormat df2= null;

df1 = DateFormat.getDateInstance();

df2= DateFormat.getDateTimeInstance();

System.out.println("Date "+df1.format(new Date()));

System.out.println("DateTime "+df2.format(new Date()));

输出：

Date 2016-7-18

DateTime 2016-7-18 23:58:33

此时采用的是默认的格式化操作，也可以使用Locale对象指定要显示的区域。

df1 = DateFormat.getDateInstance(DateFormat.YEAR\_FIELD,new Locale("zh","CN"));

df2=DateFormat.getDateTimeInstance(DateFormat.YEAR\_FIELD,DateFormat.ERA\_FIELD,new Locale("zh","CN"));

结果：

Date 2016年7月19日

DateTime 2016年7月19日 上午12时12分37秒 CST

注意导入包 import java.util.Locale;

6.2 SimpleDateFormat类

如果要实现转换，需要准备好一个模板，通过此模板进行日期数字的提取工作。

|  |  |  |
| --- | --- | --- |
| No | 标记 | 描述 |
| 1 | y | 表示年，年份是四位数字，所以要使用“yyyy”表示年。 |
| 2 | M | 表示月，月份是两位数字，所以要使用“MM”表示月。 |
| 3 | d | 表示日，日是两位数字，所以要使用“dd“表示日。 |
| 4 | H | 表示时，时是两位数字，所以要使用“HH“表示时。 |
| 5 | m | 表示分，两位数字“mm” |
| 6 | s | 表示秒，两位数字“ss” |
| 7 | S | 表示毫秒，三位数字“SSS” |

在SimpleDateFormat类使用的时候，必须注意的是在构造对象的时候要传入匹配的模板。

构造方法：public SimpleDateFormat（String pattern）

转换：public Date parse（String source）throws ParseException 此时取得的是全部的时间数

格式化；public final String format（Date date） 将时间格式化成字符串

String strDate= "2016-07-19 00:53:30.345";

String pat1 = "yyyy-MM-dd HH:mm:ss.SSS";

String pat2 = "yyyy年MM月dd日 HH时mm分ss秒SSS毫秒";

SimpleDateFormat sdf1 = new SimpleDateFormat(pat1);

SimpleDateFormat sdf2 = new SimpleDateFormat(pat2);

Date d = null;

try{

d = sdf1.parse(strDate);

}

catch(Exception e)

{

e.printStackTrace();

}

System.out.println(sdf2.format(d));

输出：

2016年07月19日 00时53分30秒345毫秒

其他：String timeStamp = **new** SimpleDateFormat(**"yyyyMMdd\_HHmmss"**).format(**new** Date());

7. Math类

静态类，调用静态方法

sqrt:平方根

max：求两数的最大值

min：求两数的最小值

pow：求幂

round：求四舍五入

Math.round(33.6) 输出34

8. Random类

导入包import java.util.Random;

先定义一Random对象

Random r = new Random();

public int nextInt(int n)

该方法的作用是生成一个随机的int值，该值介于[0,n)的区间，也就是0到n之间的随机int值，包含0而不包含n。

public double nextDouble()

该方法的作用是生成一个随机的double值，数值介于[0,1.0)之间

9. NumberFormat类

完成数字的格式化操作

NumberFormat nf = null;

nf = NumberFormat.getInstance();

System.out.println(nf.format(1234567));

输出：1,234,567

NumberFormat.getInstance();参数可以指定Locale区域，没有参数这调用默认本地locale

10 DecimalFormat

与simpleDateFormat类似，要指定模板

class FormatDemo

{

public void format1(String pattern,double value)

{

DecimalFormat df = null;

df = new DecimalFormat(pattern) ;

String str = df.format(value);

System.out.println(str);

}

}

FormatDemo demo = new FormatDemo();

demo.format1("###,###.###",111222.34567);

demo.format1("000,000.000",11222.34567);

demo.format1("000,000.000☆",11222.34567);

demo.format1("##.###%",0.34567);

demo.format1("00.###%",0.034567);

demo.format1("##.###\u2030",0.34567);

输出：

111,222.346

011,222.346

011,222.346☆

34.567%

03.457%

345.67‰

11. 大数字

正常情况下，一个数字最多只能放到long中。

操作整数：BigInteger

操作小数：BigDecimal

11.1 BigInteger

导入包：import java.math.BigInteger;

BigInteger bi1 = new BigInteger("123456789");

BigInteger bi2 = new BigInteger("987654321");

System.out.println("加法："+bi1.add(bi2));

System.out.println("减法："+bi1.subtract(bi2));

System.out.println("乘法："+bi1.multiply(bi2));

System.out.println("除法："+bi1.divide(bi2));

System.out.println("最大数："+bi1.max(bi2));

System.out.println("最小数："+bi1.min(bi2));

输出：

加法：1111111110

减法：-864197532

乘法：121932631112635269

除法：0

最大数：987654321

最小数：123456789

divide触发默认没有余数，如果要余数，要用divideAndRemainder方法

BigInteger result[]= bi2.divideAndRemainder(bi1); //求出余数

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

输出： 8 9

11.2 BigDecimal

12 对象克隆

在object类中存在clone方法。

如果一个对象能克隆，必须要继承自Cloneable接口。

为什么需要克隆：

在实际编程过程中，我们常常要遇到这种情况：有一个对象A，在某一时刻A中已经包含了一些有效值，此时可能会需要一个和A完全相同新对象B，并且此后对B任何改动都不会影响到A中的值，也就是说，A与B是两个独立的对象，但B的初始值是由A对象确定的。在Java语言中，用简单的赋值语句是不能满足这种需求的，要满足这种需求有很多途径。

观察一下Object类的clone()方法是一个native方法，native方法的效率一般来说都是远高于java中的非native方法。这也解释了为什么要用Object中clone()方法而不是先new一个类，然后把原始对象中的信息赋到新对象中，虽然这也实现了clone功能。Object类中的clone()还是一个protected属性的方法，重载之后要把clone()方法的属性设置为public。

Object类中clone()方法产生的效果是：先在内存中开辟一块和原始对象一样的空间，然后原样拷贝原始对象中的内容。对基本数据类型，这样的操作是没有问题的，但对非基本类型变量，我们知道它们保存的仅仅是对象的引用，这也导致clone后的非基本类型变量和原始对象中相应的变量指向的是同一个对象。

对象的序列化还有另一个容易被大家忽略的功能就是对象复制（Clone），Java中通过Clone机制可以复制大部分的对象，但是众所周知，Clone有深度Clone和浅度Clone，如果你的对象非常非常复杂，并且想实现深层 Clone，如果使用序列化，不会超过10行代码就可以解决。

class Person implements Cloneable{ // 实现Cloneable接口表示可以被克隆

private String name ;

public Person(String name){

this.name = name ;

}

public void setName(String name){

this.name = name ;

}

public String getName(){

return this.name ;

}

public String toString(){

return "姓名：" + this.name ;

}

public Object clone()

throws CloneNotSupportedException

{

return super.clone() ; // 具体的克隆操作由父类完成

}

};

public class CloneDemo01{

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

Person p1 = new Person("张三") ;

Person p2 = (Person)p1.clone() ;

p2.setName("李四") ;

System.out.println("原始对象：" + p1) ;

System.out.println("克隆之后的对象：" + p2) ;

}

};

12 数组Arrays类

**12.1 sort排序方法**

int temp[] ={4,6,2,6,8,22,53,24};

Arrays.sort(temp);

for(int i=0;i<temp.length;i++)

{

System.out.println(temp[i]);

}

sort有很多重载的方法，sort(Ojbect[] temp),能排序对象。

**12.2 toString转换为字符串**

System.out.println(Arrays.toString(temp));

输出: [2, 4, 6, 6, 8, 22, 24, 53]

**12.2 二分查询binarySearch**

int point = Arrays.binarySearch(temp,22); //查找22元素的位置

**12.3数组填充 fill**

Arrays.fill(temp,9);

System.out.println(Arrays.toString(temp));

输出 [9, 9, 9, 9, 9, 9, 9, 9]

注意数组是引用传递

13 Comparable接口

public interface Comparable<T>{

public int compareTo(T o);

}

class Student implements Comparable<Student> { // 指定类型为Student

private String name ;

private int age ;

private float score ;

public Student(String name,int age,float score){

this.name = name ;

this.age = age ;

this.score = score ;

}

public String toString(){

return name + "\t\t" + this.age + "\t\t" + this.score ;

}

public int compareTo(Student stu){ // 覆写compareTo()方法，实现排序规则的应用

if(this.score>stu.score){

return -1 ;

}else if(this.score<stu.score){

return 1 ;

}else{

if(this.age>stu.age){

return 1 ;

}else if(this.age<stu.age){

return -1 ;

}else{

return 0 ;

}

}

}

};

public class ComparableDemo{

public static void main(String args[]){

Student stu[] = {new Student("张三",20,90.0f),

new Student("李四",22,90.0f),new Student("王五",20,99.0f),

new Student("赵六",20,70.0f),new Student("孙七",22,100.0f)} ;

java.util.Arrays.sort(stu) ; // 进行排序操作

for(int i=0;i<stu.length;i++){ // 循环输出数组中的内容

System.out.println(stu[i]) ;

}

}

};

二叉树排序原理：

被排序的对象要实现Comparable接口，下面的例子Integer就实现了Comprable接口

Array.Sort就是使用这种排序方法

class BinaryTree

{

class Node{

private Comparable data;

private Node left;

private Node right;

public Node(Comparable data)

{

this.data=data;

}

public void add(Node newNode)

{

if(newNode.data.compareTo(this.data)<0)

{

if(this.left==null)

{

this.left= newNode;

}

else

{

this.left.add(newNode) ;

}

}

else

{

if(this.right==null)

{

this.right=newNode;

}

else

{

this.right.add(newNode);

}

}

}

public void printNode()

{

if(this.left!=null)

{

this.left.printNode();

}

System.out.print(this.data+"\t");

if(this.right!=null)

{

this.right.printNode();

}

}

} ;

private Node root;

public void add(Comparable data)

{

Node newNode = new Node(data);

if(root ==null)

{

root = newNode;

}

else

{

root.add(newNode);

}

}

public void print()

{

this.root.printNode();

}

};

public class ComparableDemo2{

public static void main(String[] args)

{

BinaryTree bt = new BinaryTree();

bt.add(8);

bt.add(3);

bt.add(8);

bt.add(3);

bt.add(83);

bt.add(32);

bt.add(48);

bt.add(13);

bt.add(58);

bt.add(39);

bt.print();

}

}

14 Comparator接口

如果一个已经定义好的类没有实现Comparable接口，可以定义一个Comparator类来排序

import java.util.\* ;

class Student{ // 指定类型为Student

private String name ;

private int age ;

public Student(String name,int age){

this.name = name ;

this.age = age ;

}

public boolean equals(Object obj){ // 覆写equals方法

if(this==obj){

return true ;

}

if(!(obj instanceof Student)){

return false ;

}

Student stu = (Student) obj ;

if(stu.name.equals(this.name)&&stu.age==this.age){

return true ;

}else{

return false ;

}

}

public void setName(String name){

this.name = name ;

}

public void setAge(int age){

this.age = age ;

}

public String getName(){

return this.name ;

}

public int getAge(){

return this.age ;

}

public String toString(){

return name + "\t\t" + this.age ;

}

};

class StudentComparator implements Comparator<Student>{ // 实现比较器

// 因为Object类中本身已经有了equals()方法

public int compare(Student s1,Student s2){

if(s1.equals(s2)){

return 0 ;

}else if(s1.getAge()<s2.getAge()){ // 按年龄比较

return 1 ;

}else{

return -1 ;

}

}

};

public class ComparatorDemo{

public static void main(String args[]){

Student stu[] = {new Student("张三",20),

new Student("李四",22),new Student("王五",20),

new Student("赵六",20),new Student("孙七",22)} ;

java.util.Arrays.sort(stu,new StudentComparator()) ; // 进行排序操作

for(int i=0;i<stu.length;i++){ // 循环输出数组中的内容

System.out.println(stu[i]) ;

}

}

总结：在使用中尽可能使用Comparable在需要排序的类上实现此接口，而Comparator需要单独建立一个排序的类，这样如果有很多的话，则排序规则类就会非常的多，操作起来非常麻烦。

15 Observer观察者设计模式

import java.util.\*;

class House extends Observable{

private float price;

public House(float price)

{

this.price=price;

}

public float getPrice()

{

return this.price;

}

public void setPrice(float price)

{

super.setChanged();

super.notifyObservers(price) ;

this.price=price;

}

public String toString()

{

return "房子价格为："+this.price;

}

}

class HousePriceObserver implements Observer

{

private String name;

public HousePriceObserver(String name)

{

this.name=name;

}

public void update(Observable o,Object arg){

if(arg instanceof Float)

{

System.out.print(this.name+"观察到价格更改为：");

System.out.println(((Float)arg).floatValue());

}

}

}

public class ObserDemo{

public static void main(String[] args)

{

House h= new House(10000);

HousePriceObserver hop1= new HousePriceObserver("购房者1");

HousePriceObserver hop2= new HousePriceObserver("购房者2");

HousePriceObserver hop3= new HousePriceObserver("购房者3");

h.addObserver(hop1);

h.addObserver(hop2);

h.addObserver(hop3);

System.out.println(h);

h.setPrice(6666);

System.out.println(h);

}

}

首先要定义观察的对象类，继承自Observerable类，然后定义观察者类，继承自Observer接口。

在Observer接口中的update方法里面的两个参数：

o：代表Observerable类的对象

arg：需要被观察的内容

**此模式是一种哦本身的机制，不使用Obser和Obserable也是可以实现的，只是比较复杂。**

2016年8月4日15:07:34

1.正则表达式

判断是否由数字组成：

import java.util.regex.Pattern;

public class RegexDemo

{

public static void main(String[] args)

{

String str="1234567890";

if(Pattern.compile("[0-9]+").matcher(str).matches())

{

System.out.println("数字组成");

}

}

}

1.1 Pattern和Matcher类

这两个类都定义在java.util.regex包中

1.1.1 判断日期格式

String str="1990-09-19";

String pat ="\\d{4}-\\d{2}-\\d{2}";

Pattern p = Pattern.compile(pat); //实例化Pattern类

Matcher m = p.matcher(str); //实例化Matcher类

if(m.matches())

{

System.out.println("日期格式合法");

}

else

{

System.out.println("日期格式不合法");

}

1.1.2 使用split拆分

String str= "A12B33Cfd3234E43R";

String pat="\\d+"; //按照数字拆分

Pattern p= Pattern.compile(pat);

String s[] = p.split(str);

for(int i=0;i<s.length;i++)

{

System.out.print(s[i]+"\t") ;

}

输出：A B Cfd E R

1.1.2 替换

String str= "A12B33Cfd3234E43R";

String pat="\\d+";

Pattern p= Pattern.compile(pat);

Matcher m = p.matcher(str);

String newString=m.replaceAll("--");

System.out.println(newString);

输出：A--B--Cfd--E—R

1.2 在JDK 1.4之后，java对正则进行了一些扩充，在String中开始直接支持正则操作。

替换

String str= "A12B33Cfd3234E43R".replaceAll("\\d+","--");

System.out.println(str)；

输出：A--B--Cfd--E—R

匹配：

boolean temp ="1991-09-19".matches("\\d{4}-\\d{2}-\\d{2}");

System.out.println(temp);

输出：true

拆分：

String s[] ="A12B33Cfd3234E43R".split("\\d+");

for(int i=0;i<s.length;i++)

{

System.out.print(s[i]+" ") ;

}

输出：A B Cfd E R

2. 定时调度Timer和TimerTask

Timer是一种线程设施，可以用来实现某一个时间或某一段时间后，安排某一个任务执行一次，或定期重复执行。

每一个Timer对象对应的上一个线程，因此计时器所执行任务应该迅速完成，否则可能会延迟后续任务的执行，而这些后续任务就有可能堆在一起，等到该任务完成后才能快速连续执行。

// 完成具体的任务操作

import java.util.TimerTask ;

import java.util.Date ;

import java.text.SimpleDateFormat ;

class MyTask extends TimerTask{ // 任务调度类都要继承TimerTask

public void run(){

SimpleDateFormat sdf = null ;

sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS") ;

System.out.println("当前系统时间为：" + sdf.format(new Date())) ;

}

};

import java.util.Timer ;

public class TestTask{

public static void main(String args[]){

Timer t = new Timer() ; // 建立Timer类对象

MyTask mytask = new MyTask() ; // 定义任务

t.schedule(mytask,1000,2000) ; // 设置任务的执行，1秒后开始，每2秒重复

}

};

**2016年8月4日23:02:09**

**1. JAVA IO操作**

**1.1.创建一个新文件**

创建文件：createNewFile

File f = new File("d:\\test.txt")；

try{

f.createNewFile();

}

catch(IOException e)

{

e.printStackTrace();

}

在各个操作系统中，实际上路径的分隔符是不一样的，在windows中使用反斜杠“\”,在linux中使用正斜杠“/”

如果想让java程序可移植性继续保持，则最好根据所在的操作系统自动使用分隔符。

pathSeparator和separator

System.out.println("pathSeparator:"+File.pathSeparator);

System.out.println("separator:"+File.separator);

输出:

pathSeparator:;

separator:\

File f = new File("d:"+File.separator+"test.txt");

try{

f.createNewFile();

}

catch(IOException e)

{

e.printStackTrace();

}

**1.2 删除一个指定的文件**

delete方法

File f = new File("d:"+File.separator+"test.txt");

f.delete();

1.3 判断一个文件是否存在

exists方法

if(f.exists())

{

System.out.println("存在");

}

**1.3创建文件夹**

makedir方法

File f = new File("d:"+File.separator+"hello");

f.mkdir();

1.4 列出指定目录的全部文件

以字符串数组的方式返回 public String[] list()

以File数组的形式返回 public File[] listFiles()

File f = new File("d:"+File.separator);

String str[] = f.list();

for(int i=0;i<str.length;i++)

{

System.out.println(str[i]);

}

列出文件名

File f = new File("d:"+File.separator);

File files[] = f.listFiles();

for(int i=0;i<files.length;i++)

{

System.out.println(files[i]) ;

}

列出完整路径

**1.4 判断一个给定的路径是否是路径**

public boolean isDirectory()

File f = new File("d:");

boolean temp =f.isDirectory();

System.out.println(f.getPath()+" "+temp);

输出：d: true

其中 f.getPath是获取路径字符串

2016年8月5日17:08:03

**1、RandAccessFile类**

完成随机读取功能，可以读取指定位置的内容。

import java.io.File ;

import java.io.RandomAccessFile ;

public class RandomAccessFileDemo01{

// 所有的异常直接抛出，程序中不再进行处理

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

File f = new File("d:" + File.separator + "test.txt") ; // 指定要操作的文件

RandomAccessFile rdf = null ; // 声明RandomAccessFile类的对象

rdf = new RandomAccessFile(f,"rw") ;// 读写模式，如果文件不存在，会自动创建

String name = null ;

int age = 0 ;

name = "zhangsan" ; // 字符串长度为8

age = 30 ; // 数字的长度为4

rdf.writeBytes(name) ; // 将姓名写入文件之中

rdf.writeInt(age) ; // 将年龄写入文件之中

name = "lisi " ; // 字符串长度为8

age = 31 ; // 数字的长度为4

rdf.writeBytes(name) ; // 将姓名写入文件之中

rdf.writeInt(age) ; // 将年龄写入文件之中

name = "wangwu " ; // 字符串长度为8

age = 32 ; // 数字的长度为4

rdf.writeBytes(name) ; // 将姓名写入文件之中

rdf.writeInt(age) ; // 将年龄写入文件之中

rdf.close() ; // 关闭

}

};

以上代码完成了写的操作

import java.io.File ;

import java.io.RandomAccessFile ;

public class RandomAccessFileDemo02{

// 所有的异常直接抛出，程序中不再进行处理

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

File f = new File("d:" + File.separator + "test.txt") ; // 指定要操作的文件

RandomAccessFile rdf = null ; // 声明RandomAccessFile类的对象

rdf = new RandomAccessFile(f,"r") ;// 以只读的方式打开文件

String name = null ;

int age = 0 ;

byte b[] = new byte[8] ; // 开辟byte数组

// 读取第二个人的信息，意味着要空出第一个人的信息

rdf.skipBytes(12) ; // 跳过第一个人的信息

for(int i=0;i<b.length;i++){

b[i] = rdf.readByte() ; // 读取一个字节

}

name = new String(b) ; // 将读取出来的byte数组变为字符串

age = rdf.readInt() ; // 读取数字

System.out.println("第二个人的信息 --> 姓名：" + name + "；年龄：" + age) ;

// 读取第一个人的信息

rdf.seek(0) ; // 指针回到文件的开头

for(int i=0;i<b.length;i++){

b[i] = rdf.readByte() ; // 读取一个字节

}

name = new String(b) ; // 将读取出来的byte数组变为字符串

age = rdf.readInt() ; // 读取数字

System.out.println("第一个人的信息 --> 姓名：" + name + "；年龄：" + age) ;

rdf.skipBytes(12) ; // 空出第二个人的信息

for(int i=0;i<b.length;i++){

b[i] = rdf.readByte() ; // 读取一个字节

}

name = new String(b) ; // 将读取出来的byte数组变为字符串

age = rdf.readInt() ; // 读取数字

System.out.println("第三个人的信息 --> 姓名：" + name + "；年龄：" + age) ;

rdf.close() ; // 关闭

}

};

以上代码完成了读的操作

2. 字节流与字符流

在java.io包中操作文件内容的主要有两大类：字节流、字符流，两类都分为输入和输出操作。在字节流中输出数据主要使用OutputStream完成，输入使用InputStream，在字符流中输出主要使用Writer类完成，输入主要使用Reader类完成。

操作流程：

使用File类打开一个文件

通过字节流或字符流的子类，指定输出位置

进行读写操作

关闭输入输出

**2.1字节流**

**写文件**

File f = new File("d:"+File.separator+"test.txt");

OutputStream out = new FileOutputStream(f);

String str ="Hello World";

byte b[] = str.getBytes();

out.write(b);

out.close();

在操作的时候，如果文件不存在，系统会自动创建文件。

以上方法，如果重新执行程序，则肯定会覆盖文件中的内容，那么此时可以通过FileOutputStream向文件中追加内容。

public FileOutputStream(File file,boolean append)throws FileNotFoundException

如果将append设置为true，则表示在文件末尾追加内容。

也可以按照一个一个字节来写

File f= new File("d:" + File.separator + "test.txt") ; // 声明File对象

// 第2步、通过子类实例化父类对象

OutputStream out = null ; // 准备好一个输出的对象

out = new FileOutputStream(f) ; // 通过对象多态性，进行实例化，第二个参数为true，表示追加

// 第3步、进行写操作

String str = "Hello World!!!" ; // 准备一个字符串

byte b[] = str.getBytes() ; // 只能输出byte数组，所以将字符串变为byte数组

for(int i=0;i<b.length;i++){ // 采用循环方式写入

out.write(b[i]) ; // 每次只写入一个内容

}

// 第4步、关闭输出流

out.close() ;

**读文件**

import java.io.File;

import java.io.InputStream;

import java.io.FileInputStream;

public class InputStreamDemo

{

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

{

File f = new File("d:"+File.separator+"test.txt");

InputStream input = new FileInputStream(f);

byte[] b= new byte[1024];

input.read(b); //返回值是实际读取到的长度

input.close();

System.out.println(new String(b));

}

}

上面会输出很多的空格

修改为：

File f = new File("d:"+File.separator+"test.txt");

InputStream input = new FileInputStream(f);

byte[] b= new byte[1024];

int len = input.read(b);

input.close();

System.out.println(len);

System.out.println(new String(b,0,len));

上面的代码会开辟多余的数组空间，应该根据文件大小来设置数组长度

byte[] b= new byte[(int)f.length()];

for(int i=0;i<b.length;i++){

b[i] = (byte)input.read() ; // 读取内容

}

// 第4步、关闭输出流

input.close() ;

如果事先不知道文件大小，可以使用读字节的方式，上面的read是读到一个byte数组里面，返回长度，如果read没有参数，则是读取一个字节，返回该字节内容，如果返回值为-1，则表示读到文件末尾了。

File f = new File("d:"+File.separator+"test.txt") ;

InputStream input = new FileInputStream(f);

int len=0;

int temp=0;

char c[]= new char[1024];

while((temp=input.read())!=-1)

{

c[len]=(char)temp;

len++;

}

System.out.println(new String(c,0,len));

**2.2字符流**

**字符流读取**

File f= new File("d:"+File.separator+"test.txt");

Reader rd = new FileReader(f);

char c[] = new char[1024];

int len = rd.read(c);

rd.close();

System.out.println(new String(c,0,len));

File f= new File("d:" + File.separator + "test.txt") ;

Reader input = null ;

input = new FileReader(f) ; // 通过对象多态性，进行实例化

char c[] = new char[1024] ; // 所有的内容都读到此数组之中

int temp = 0 ; // 接收每一个内容

int len = 0 ; // 读取内容

while((temp=input.read())!=-1){

// 如果不是-1就表示还有内容，可以继续读取

c[len] = (char)temp ;

len++ ;

}

// 第4步、关闭输出流

input.close() ; // 关闭输出流

System.out.println("内容为：" + new String(c,0,len)) ; // 把字符数组变为字符串输出

**字符流写入**

File f = new File("d:"+File.separator+"test2.txt");

Writer wr = new FileWriter(f);

String str="Hello world";

wr.write(str);

wr.close();

注意一定要调用close方法，否则不会讲内容写入文件，如果是用outputstream方法，则即使不close也能写入文件。

2016年8月7日16:25:38

**1.字节字符转换流**

读取：

File f = new File("d:"+File.separator+"test.txt");

Reader reader = new InputStreamReader(new FileInputStream(f)) ;

char c[] = new char[1024];

int len= reader.read(c);

reader.close();

System.out.println(new String(c,0,len));

写入：

File f = new File("d:" + File.separator + "test.txt") ;

Writer out = null ; // 字符输出流

out = new OutputStreamWriter(new FileOutputStream(f)) ; // 字节流变为字符流

out.write("hello world!!") ; // 使用字符流输出

out.close() ;

**2. 内存操作流**

ByteArrayInputStream和ByteArrayOutputStream

String str = "HELLOWORLD" ; // 定义一个字符串，全部由大写字母组成

ByteArrayInputStream bis = null ; // 内存输入流

ByteArrayOutputStream bos = null ; // 内存输出流

bis = new ByteArrayInputStream(str.getBytes()) ; // 向内存中输出内容

bos = new ByteArrayOutputStream() ; // 准备从内存ByteArrayInputStream中读取内容

int temp = 0 ;

while((temp=bis.read())!=-1){

char c = (char) temp ; // 读取的数字变为字符

bos.write(Character.toLowerCase(c)) ; // 将字符变为小写

}

// 所有的数据就全部都在ByteArrayOutputStream中

String newStr = bos.toString() ; // 取出内容

try{

bis.close() ;

bos.close() ;

}catch(IOException e){

e.printStackTrace() ;

}

System.out.println(newStr) ;

**3. 管道流**

管道流主要作用是可以进行两个线程间通讯，分为管道输出流PipedOutputStream,管道输入流PipedInputStream，如果想进行管道输出，则必须把输出流连在输入流之上

import java.io.\* ;

class Send implements Runnable{ // 线程类

private PipedOutputStream pos = null ; // 管道输出流

public Send(){

this.pos = new PipedOutputStream() ; // 实例化输出流

}

public void run(){

String str = "Hello World!!!" ; // 要输出的内容

try{

this.pos.write(str.getBytes()) ;

}catch(IOException e){

e.printStackTrace() ;

}

try{

this.pos.close() ;

}catch(IOException e){

e.printStackTrace() ;

}

}

public PipedOutputStream getPos(){ // 得到此线程的管道输出流

return this.pos ;

}

};

class Receive implements Runnable{

private PipedInputStream pis = null ; // 管道输入流

public Receive(){

this.pis = new PipedInputStream() ; // 实例化输入流

}

public void run(){

byte b[] = new byte[1024] ; // 接收内容

int len = 0 ;

try{

len = this.pis.read(b) ; // 读取内容

}catch(IOException e){

e.printStackTrace() ;

}

try{

this.pis.close() ; // 关闭

}catch(IOException e){

e.printStackTrace() ;

}

System.out.println("接收的内容为：" + new String(b,0,len)) ;

}

public PipedInputStream getPis(){

return this.pis ;

}

};

public class PipedDemo{

public static void main(String args[]){

Send s = new Send() ;

Receive r = new Receive() ;

try{

s.getPos().connect(r.getPis()) ; // 连接管道

}catch(IOException e){

e.printStackTrace() ;

}

new Thread(s).start() ; // 启动线程

new Thread(r).start() ; // 启动线程

}

};

**4. 打印流**

字节打印流PrintStream和字符打印流PrintWriter

可以打印任何的数据类型，例如：小数、整数、字符串等

PrintStream ps = null ; // 声明打印流对象

// 如果现在是使用FileOuputStream实例化，意味着所有的输出是向文件之中

ps = new PrintStream(new FileOutputStream(new File("d:" + File.separator + "test.txt"))) ;

ps.print("hello ") ;

ps.println("world!!!") ;

ps.print("1 + 1 = " + 2) ;

ps.close() ;

构造方法：public PrintStream(OutputStream out) 接收OutputStream的子类

PrintStream可以更加方便的输出数据，这就好比将OutputStream类重新包装了一下，使之输出更加方便。

格式化输出，在JDK1.5后对printstream进行了扩充，增加了格式化输出，直接使用printf()方法可以完成操作，类似c语言的printf。

%s 表示内容为字符串

%d 表示内容为整数

%f 表示内容为小数

%c 表示内容为字符

PrintStream ps = null ; // 声明打印流对象

// 如果现在是使用FileOuputStream实例化，意味着所有的输出是向文件之中

ps = new PrintStream(new FileOutputStream(new File("d:" + File.separator + "test.txt"))) ;

String name = "李兴华" ; // 定义字符串

int age = 30 ; // 定义整数

float score = 990.356f ; // 定义小数

char sex = 'M' ; // 定义字符

ps.printf("姓名：%s；年龄：%d；成绩：%f；性别：%c",name,age,score,sex) ;

ps.close() ;

**5.System类对IO的支持**

System对io的三种支持

System.out.

sytem.err

system.in

System.out输出的时候是将输出的位置定义在了显示器之中

Fileoutputstream是定位在文件里。

import java.io.OutputStream ;

import java.io.IOException ;

public class SystemDemo01{

public static void main(String args[]){

OutputStream out = System.out ; // 此时的输出流是向屏幕上输出

try{

out.write("hello world!!!".getBytes()) ; // 向屏幕上输出

}catch(IOException e){

e.printStackTrace() ; // 打印异常

}

try{

out.close() ; // 关闭输出流

}catch(IOException e){

e.printStackTrace() ;

}

}

};

System.err是标准的错误输出

public class SystemDemo02{

public static void main(String args[]){

String str = "hello" ; // 声明一个非数字的字符串

try{

System.out.println(Integer.parseInt(str)) ; // 转型

}catch(Exception e){

System.err.println(e) ;

}

}

};

**System.in**

完成键盘数据读取的功能

InputStream input = System.in;

byte b[] = new byte[1024];

System.out.print("请输入内容：");

int len = input.read(b);

System.out.println("输入内容为："+new String(b,0,len));

input.close();

上面的代码有问题：  
1. 如果字节数组比较小，超出的内容就不会写入

2. 如果输入的内容有中文，数组长度为奇数，则会乱

码，因为汉字两个字节

修改1：

InputStream input = System.in;

StringBuffer buf = new StringBuffer();

int temp=0;

while((temp=input.read()) !=-1)

{

char c = (char) temp;

if(c=='\n') //如果回车，则退出

{

break;

}

buf.append(c);

}

input.close();

System.out.println(buf);

修改后输入英文内容，没有问题，但是输入汉字就全是乱码了

要想实现汉字的读入，只能用**BufferedReader**类来完成

**6. 输入、输出重定向**

**为System.out输出重定向**

使用setOut方法

System.setOut(new PrintStream(new FileOutputStream("d:"+File.separator+"read.txt")));

System.out.println("hello world");

输出到文件了

通过此操作可以完成错误重定向

System.out是希望用户看得到的信息，一旦有错误，最好保存

String str = "hello";

try{

System.out.println(Integer.parseInt(str));

}

catch(Exception e)

{

try{

System.setOut(new PrintStream(new FileOutputStream("d:"+File.separator+"err.txt")));

}

catch(Exception e1)

{

}

System.out.println(e);

}

**为System.err重定向**

1.

System.setErr(new PrintStream(new FileOutputStream("d:"+File.separator+"err.txt")));

System.err.println("hello world");

2.

ByteArrayOutputStream bos = null ; // 声明内存输出流

bos = new ByteArrayOutputStream() ; // 实例化

System.setErr(new PrintStream(bos)) ; // 输出重定向

System.err.print("www.mldnjava.cn") ; // 错误输出，不再向屏幕上输出

System.err.println("李兴华") ; // 向内存中输出

System.out.println(bos) ; // 输出内存中的数据

**为System.in重定向**

默认情况下System.in是指屏幕键盘输入，也可以通过setIn()方法，将其输入流的位置改变。

System.setIn(new FileInputStream("d:"

+ File.separator + "demo.txt")) ; // 设置输入重定向

InputStream input = System.in ; // 从文件中接收数据

byte b[] = new byte[1024] ;// 开辟空间，接收数据

int len = input.read(b) ; //接收

System.out.println("输入的内容为：" + new String(b,0,len)) ;

input.close() ; // 关闭输入流

上面的代码让从文件中接收数据，而不是接收键盘的数据

**7. BufferedReader类**

为了可以从键盘读取任意长度，且可以是中文的字符，要用到这个类，还用用到字符字节转换流InputStreamReader

BufferedReader = null;

buf = new BufferedReader(new InputStreamReader(System.in));

将字节流转换为字符流，然后用readLine方法读数据

BufferedReader buf =null;

buf = new BufferedReader(new InputStreamReader(System.in)) ;

String str =null;

try{

str= buf.readLine();

}

catch(IOException e)

{

e.printStackTrace();

}

System.out.println(str);

}

上面的代码完成了读取任意长度，即使是中文也可以读取的功能。

2016年8月9日17:13:15

1. Scanner类

Scanner scan= new Scanner(System.in) ;

scan.useDelimiter("\n");

String str= scan.next();

System.out.println(str);

完成键盘输入功能

可以用hasNextInt判断输入的是否是整数，用hasNextFloat判断输入的是否是小数

这个类可以接收各种类型，但是对于日期型的数据却无法接收，

如果要接收Date类型的数据，则只能通过字符串转型，但是在接收的时候依然可以使用Scanner类中提供的方法进行验证，因为可以接收正则表达式

验证：public String hasNext(Pattern pattern)

接收：public String next(String pattern)

scanner还可以从文件中读取数据

File f = new File("d:"+File.separator+"test.txt");

Scanner scan= new Scanner(f) ;

String str=null;

if(scan.hasNext())

{

str= scan.next();

}

System.out.println(str);

以上的代码只能读取第一行且第一个空格前的数据

File f = new File("d:"+File.separator+"test.txt");

Scanner scan= new Scanner(f) ;

StringBuffer str= new StringBuffer();

while(scan.hasNext())

{

str.append(scan.next()).append("\n");

}

System.out.println(str);

**2016年8月9日22:55:37**

**1. 数据操作流**

DataInputStream和DataOutputStream

import java.io.DataInputStream ;

import java.io.File ;

import java.io.FileInputStream ;

public class DataInputStreamDemo{

public static void main(String args[]) throws Exception{ // 所有异常抛出

DataInputStream dis = null ; // 声明数据输入流对象

File f = new File("d:" + File.separator + "order.txt") ; // 文件的保存路径

dis = new DataInputStream(new FileInputStream(f)) ; // 实例化数据输入流对象

String name = null ; // 接收名称

float price = 0.0f ; // 接收价格

int num = 0 ; // 接收数量

char temp[] = null ; // 接收商品名称

int len = 0 ; // 保存读取数据的个数

char c = 0 ; // '\u0000'

try{

while(true){

temp = new char[200] ; // 开辟空间

len = 0 ;

while((c=dis.readChar())!='\t'){ // 接收内容

temp[len] = c ;

len ++ ; // 读取长度加1

}

name = new String(temp,0,len) ; // 将字符数组变为String

price = dis.readFloat() ; // 读取价格

dis.readChar() ; // 读取\t

num = dis.readInt() ; // 读取int

dis.readChar() ; // 读取\n

System.out.printf("名称：%s；价格：%5.2f；数量：%d\n",name,price,num) ;

}

}catch(Exception e){}

dis.close() ;

}

};

2. 合并流

SequenceInputStream

将两个文件合并到一个文件

import java.io.File ;

import java.io.SequenceInputStream ;

import java.io.FileInputStream ;

import java.io.InputStream ;

import java.io.FileOutputStream ;

import java.io.OutputStream ;

public class SequenceDemo{

public static void main(String args[]) throws Exception { // 所有异常抛出

InputStream is1 = null ; // 输入流1

InputStream is2 = null ; // 输入流1

OutputStream os = null ; // 输出流

SequenceInputStream sis = null ; // 合并流

is1 = new FileInputStream("d:" + File.separator + "a.txt") ;

is2 = new FileInputStream("d:" + File.separator + "b.txt") ;

os = new FileOutputStream("d:" + File.separator + "ab.txt") ;

sis = new SequenceInputStream(is1,is2) ; // 实例化合并流

int temp = 0 ; // 接收内容

while((temp=sis.read())!=-1){ // 循环输出

os.write(temp) ; // 保存内容

}

sis.close() ; // 关闭合并流

is1.close() ; // 关闭输入流1`

is2.close() ; // 关闭输入流2

os.close() ; // 关闭输出流

}

};

2016年8月10日10:56:29

1.压缩流

java支持三种压缩格式zip、jar、gzip

zip需要导入java.uitl.zip包，使用ZipFile、ZipOutputStream、ZipInputStream、ZipEntry几个类完成操作。

jar需要导入java.util.jar包，使用JarOutputStream、JarInputStream、JARFile、JAREntry

2016年8月10日22:44:36

对象序列化

import java.io.Serializable ;

一个对象要想序列化，必须实现Serializable接口

要想完成序列化，还必须依靠对象输出流ObjectOutputStream和对象输入流ObjectIntputStream

使用对象输出流叫序列化，使用对象输入流叫反序列化

File f = new File("D:" + File.separator + "test.txt") ; // 定义保存路径

ObjectOutputStream oos = null ; // 声明对象输出流

OutputStream out = new FileOutputStream(f) ; // 文件输出流

oos = new ObjectOutputStream(out) ;

oos.writeObject(new Person("张三",30)) ; // 保存对象

oos.close() ; // 关闭

File f = new File("D:" + File.separator + "test.txt") ; // 定义保存路径

ObjectInputStream ois = null ; // 声明对象输入流

InputStream input = new FileInputStream(f) ; // 文件输入流

ois = new ObjectInputStream(input) ; // 实例化对象输入流

Object obj = ois.readObject() ; // 读取对象

ois.close() ; // 关闭

System.out.println(obj) ;

如果某个属性不希望被序列化，可以使用transient关键字

private transient Stirng name;

2016年8月11日18:46:41

**1. List接口**

Collection下分为很多子接口，其中一个List借口哦，List接口中可以存放任意数据。而且在List接口中内容是允许重复的。

List接口的功能要比Collection接口强大很多，因为大量扩充了Collection接口的操作。

List接口的方法：

public void add(int index,E element)

public boolean addAll(int index,Collection<?extends E>c)

E get(int index)

public int indexOf(Object o)

public int lastIndexOf(Object o)

public ListIterator<E>listIterator()

public E remove(int index) 按指定位置删除元素

public List<E> subList(int fromIndex,int toIndex) 取出集合中的子集

public E set(int index,E element) 替换指定位置的元素

List接口的常用子类：ArrayList

常用操作：

判断集合是否为空： boolean isEmpty()

截取部分集合：List<E> subList(int fromIndex,int toIndex)

查找指定对象是否存在： int indexOf(Object o) 如果找到返回位置，否则返回-1

查找是否存在：boolean contains(Object o)

挽救的子类：Vector

在list接口中还有一个子类：Vectro，Vector属于一个挽救的子类，和之前的操作区别不大。

ArrayList与Vector的区别，ArrayList采用异步处理方式，性能高，Vector采用同步处理方式，性能低。

ArrayList属于非线程安全的操作类，Vector属性线程安全的操作类。

**2. LinkedList类**

LinkedList表示一个链表操作类，继承List接口和Queue接口

**Queue接口定义的方法：**

public E element() 找到链表的表头

public boolean offer(E o) 将指定的元素增加到链表的结尾

public E peek() 找到但并不删除链表的头

public E poll() 找到并删除此链表的头

public E remove() 检索并删除表头

**LinkedList中操作链表的部分方法：**

public void addFirst(E o) 在链表开头增加元素

public void addLast(E o) 在链表结尾增加元素

public boolean offer(E o) 将指定元素增加到链表结尾

public E removeFirst() 删除链表的第一个元素

public E removeLast() 删除链表的最后一个元素

**2. Set接口**

和List接口不同的是Set接口中的元素不能重复，而List接口可以重复

Set接口也是Collection接口的子类

set接口的主要方法与Conllection接口是一致的

Set接口的实例无法向List接口那样进行双向输出

Set接口常用子类：

HashSet ：散列存放

TreeSet ： 有序存放

import java.util.HashSet ;

import java.util.Set ;

public class HashSetDemo01{

public static void main(String args[]){

Set<String> allSet = new HashSet<String>() ;

allSet.add("A") ; // 增加内容

allSet.add("B") ; // 增加内容

allSet.add("C") ; // 增加内容

allSet.add("C") ; // 重复内容

allSet.add("C") ; // 重复内容

allSet.add("D") ; // 增加内容

allSet.add("E") ; // 增加内容

System.out.println(allSet) ;

}

};

输出: [A, B, C, D, E]

import java.util.TreeSet ;

import java.util.Set ;

public class TreeSetDemo01{

public static void main(String args[]){

Set<String> allSet = new TreeSet<String>() ;

allSet.add("C") ; // 增加内容

allSet.add("C") ; // 重复内容

allSet.add("C") ; // 重复内容

allSet.add("D") ; // 增加内容

allSet.add("B") ; // 增加内容

allSet.add("A") ; // 增加内容

allSet.add("E") ; // 增加内容

System.out.println(allSet) ;

}

};

输出：[A, B, C, D, E]

TreeSet是可以排序的

**i**mport java.util.Set ;

import java.util.HashSet ;

class Person{

private String name ;

private int age ;

public Person(String name,int age){

this.name = name ;

this.age = age ;

}

public String toString(){

return "姓名：" + this.name + "；年龄：" + this.age ;

}

};

public class RepeatDemo01{

public static void main(String args[]){

Set<Person> allSet = new HashSet<Person>() ;

allSet.add(new Person("张三",30)) ;

allSet.add(new Person("李四",31)) ;

allSet.add(new Person("王五",32)) ;

allSet.add(new Person("王五",32)) ;

allSet.add(new Person("王五",32)) ;

allSet.add(new Person("赵六",33)) ;

allSet.add(new Person("孙七",33)) ;

System.out.println(allSet) ;

}

};

上面的代码不能排序，因为Person没有实现Comparable接口

**import java.util.Set ;**

**import java.util.TreeSet ;**

**class Person implements Comparable<Person>{**

**private String name ;**

**private int age ;**

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

**this.name = name ;**

**this.age = age ;**

**}**

**public String toString(){**

**return "姓名：" + this.name + "；年龄：" + this.age ;**

**}**

**public int compareTo(Person per){**

**if(this.age>per.age){**

**return 1 ;**

**}else if(this.age<per.age){**

**return -1 ;**

**}else{**

**return 0 ;**

**}**

**}**

**};**

**public class TreeSetDemo03{**

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

**Set<Person> allSet = new TreeSet<Person>() ;**

**allSet.add(new Person("张三",30)) ;**

**allSet.add(new Person("李四",31)) ;**

**allSet.add(new Person("王五",32)) ;**

**allSet.add(new Person("王五",32)) ;**

**allSet.add(new Person("王五",32)) ;**

**allSet.add(new Person("赵六",33)) ;**

**allSet.add(new Person("孙七",33)) ;**

**System.out.println(allSet) ;**

**}**

**};**

结果：[姓名：张三；年龄：30, 姓名：李四；年龄：31, 姓名：王五；年龄：32, 姓名：赵六；年龄：33]

上面的代码 孙七没有加进来

import java.util.Set ;

import java.util.TreeSet ;

class Person implements Comparable<Person>{

private String name ;

private int age ;

public Person(String name,int age){

this.name = name ;

this.age = age ;

}

public String toString(){

return "姓名：" + this.name + "；年龄：" + this.age ;

}

public int compareTo(Person per){

if(this.age>per.age){

return 1 ;

}else if(this.age<per.age){

return -1 ;

}else{

return this.name.compareTo(per.name) ; // 调用String中的compareTo()方法

}

}

};

public class TreeSetDemo04{

public static void main(String args[]){

Set<Person> allSet = new TreeSet<Person>() ;

allSet.add(new Person("张三",30)) ;

allSet.add(new Person("李四",31)) ;

allSet.add(new Person("王五",32)) ;

allSet.add(new Person("王五",32)) ;

allSet.add(new Person("王五",32)) ;

allSet.add(new Person("赵六",33)) ;

allSet.add(new Person("孙七",33)) ;

System.out.println(allSet) ;

}

};

import java.util.Set ;

import java.util.HashSet ;

class Person{

private String name ;

private int age ;

public Person(String name,int age){

this.name = name ;

this.age = age ;

}

public String toString(){

return "姓名：" + this.name + "；年龄：" + this.age ;

}

};

public class RepeatDemo01{

public static void main(String args[]){

Set<Person> allSet = new HashSet<Person>() ;

allSet.add(new Person("张三",30)) ;

allSet.add(new Person("李四",31)) ;

allSet.add(new Person("王五",32)) ;

allSet.add(new Person("王五",32)) ;

allSet.add(new Person("王五",32)) ;

allSet.add(new Person("赵六",33)) ;

allSet.add(new Person("孙七",33)) ;

System.out.println(allSet) ;

}

};

输出： [姓名：王五；年龄：32, 姓名：赵六；年龄：33, 姓名：孙七；年龄：33, 姓名：王五；年龄：32, 姓名：张三；年龄：30, 姓名：李 四；年龄：31, 姓名：王五；年龄：32]

上面的代码并没有去掉重复的元素，如果要想去掉重复元素，则需要Object类中的两个方法帮助：

hashCode() 表示一个唯一的编码，一般通过计算表示

equals(): 进行对象的比较操作

import java.util.Set ;

import java.util.HashSet ;

class Person{

private String name ;

private int age ;

public Person(String name,int age){

this.name = name ;

this.age = age ;

}

public boolean equals(Object obj){ // 覆写equals，完成对象比较

if(this==obj){

return true ;

}

if(!(obj instanceof Person)){

return false ;

}

Person p = (Person)obj ; // 向下转型

if(this.name.equals(p.name)&&this.age==p.age){

return true ;

}else{

return false ;

}

}

public int hashCode(){

return this.name.hashCode() \* this.age ; // 定义一个公式

}

public String toString(){

return "姓名：" + this.name + "；年龄：" + this.age ;

}

};

public class RepeatDemo02{

public static void main(String args[]){

Set<Person> allSet = new HashSet<Person>() ;

allSet.add(new Person("张三",30)) ;

allSet.add(new Person("李四",31)) ;

allSet.add(new Person("王五",32)) ;

allSet.add(new Person("王五",32)) ;

allSet.add(new Person("王五",32)) ;

allSet.add(new Person("赵六",33)) ;

allSet.add(new Person("孙七",33)) ;

System.out.println(allSet) ;

}

};

输出：[姓名：赵六；年龄：33, 姓名：王五；年龄：32, 姓名：张三；年龄：30, 姓名：李四；年龄：31, 姓名：孙七；年龄：33]

总结：

1. 一个好的类应该覆写Object类中的equals()、hashCode()、toString()方法，实际上再String中已经全部覆写完成了。

2. Set接口依靠hashCode()和euals()完成重复元素的判断，关于这一点在日后的Map接口中也有体现。

3. TreeSet依靠Comparable接口完成排序操作。

2016年8月13日18:04:24

**1. Iterator接口**

在集合的操作中支持以下几种方式：

Iterator

ListIterator

foreach输出

Enumeration输出

集合输出的标准操作：

在使用集合输出的时候必须形成以下的一个思路：只要是喷到了集合的输出操作，就一定使用Iterator接口。

原理：Iteartor是专门的迭代输出接口，所谓迭代输出就是将元素一个个进行判断，判断其是否有内容，如果有内容则把内容取出。

**Iterator本身是一个接口，所以要想实例化必须依靠Collection接口完成。**

**Iterator<E> iterator()**

import java.util.List;

import java.util.ArrayList;

import java.util.Iterator;

public class IteratorDemo

{

public static void main(String[] args)

{

List<String> all = new ArrayList<String>();

all.add("a");

all.add("b");

all.add("c");

all.add("d");

all.add("e");

Iterator<String> iter= all.iterator(); //实例化

while(iter.hasNext()) //判断是否有内容

{

System.out.println(iter.next()) ;

}

}

}

Iterator接口提供了remove方法，此方法删除当前的对象。

在实际中，很少调用删除操作，本身的功能就是输出，集合本身就具有删除功能。

import java.util.List ;

import java.util.ArrayList ;

import java.util.Iterator ;

public class IteratorDemo02{

public static void main(String args[]){

List<String> all= new ArrayList<String>() ; //

all.add("hello") ;

all.add("\_") ;

all.add("world") ;

Iterator<String> iter = all.iterator() ; // 为Iterator接口实例化

while(iter.hasNext()){ // 判断是否有内容

String str = iter.next() ;

if("\_".equals(str)){

iter.remove() ; // 删除元素

}else{

System.out.println(str) ; // 输出内容

}

}

System.out.println("删除之后的集合：" + all) ;

}

};

List接口中本身存在删除方法：remove

如果在使用迭代输出的过程中使用了List中的remove()方法执行删除操作，则代码将出现问题。

import java.util.List ;

import java.util.ArrayList ;

import java.util.Iterator ;

public class IteratorDemo03{

public static void main(String args[]){

List<String> all= new ArrayList<String>() ; //

all.add("hello") ;

all.add("\_") ;

all.add("world") ;

Iterator<String> iter = all.iterator() ; // 为Iterator接口实例化

while(iter.hasNext()){ // 判断是否有内容

String str = iter.next() ;

if("\_".equals(str)){

all.remove(str) ; // 删除元素

}else{

System.out.println(str) ; // 输出内容

}

}

System.out.println("删除之后的集合：" + all) ;

}

};

输出：

hello

删除之后的集合：[hello, world]

使用list的删除后循环就不执行了

总结：

1. Iterator接口的功能是从前向后输出，属于单向输出

2.Iteratro的主要功能就是完成迭代输出操作

3.在使用Iterator的时候最好不要删除数据

2016年8月13日18:30:47

1.ListIterator接口

Iterator接口的主要功能是由前向后输出，而此时如果想要实现由后向前或者由前向后输出，则就必须使用Itertor的子接口ListIterator

虽然此接口可以进行双向输出，但是Collection接口中并没有为此类可以实例化的操作，只有List接口中才存在ListIterator接口的实例化操作。

import java.util.ArrayList ;

import java.util.List ;

import java.util.ListIterator ;

public class ListIteratorDemo01{

public static void main(String argsp[]){

List<String> all = new ArrayList<String>() ;

all.add("hello") ;

all.add("\_") ;

all.add("world") ;

ListIterator<String> iter = all.listIterator() ;

System.out.print("由前向后输出：") ;

while(iter.hasNext()){

String str = iter.next() ;

System.out.print(str + "、") ;

}

System.out.print("\n由后向前输出：") ;

while(iter.hasPrevious()){

String str = iter.previous() ;

System.out.print(str + "、") ;

}

}

};

必须先由前向后输出，才能由后向前输出，否则不能由后向前输出。

ListIterator还可以进行增加和修改操作add 和set

import java.util.ArrayList ;

import java.util.List ;

import java.util.ListIterator ;

public class ListIteratorDemo02{

public static void main(String argsp[]){

List<String> all = new ArrayList<String>() ;

all.add("hello") ;

all.add("\_") ;

all.add("world") ;

ListIterator<String> iter = all.listIterator() ;

System.out.print("由前向后输出：") ;

while(iter.hasNext()){

String str = iter.next() ;

System.out.print(str + "、") ;

iter.set("LI - " + str) ; // 修改内容

}

System.out.print("\n由后向前输出：") ;

iter.add("LXH");

while(iter.hasPrevious()){

String str = iter.previous() ;

System.out.print(str + "、") ;

}

}

};

总结：

要想使用ListIterator接口则只能依靠List接口完成

如果要进行由后向前的输出，则只能先进行由前向后输出

对于此接口中的增加及修改操作了解即可

**2. foreach**

foreach可以输出数组，同样也支持集合的输出

List<String> all = new ArrayList<String>() ;

all.add("hello") ;

all.add("\_") ;

all.add("world") ;

for(String str:all){

System.out.print(str + "、") ;

}

3. Enumeration接口

Iterator接口是一比较新的接口，最早使用Enumeration接口，

但是在java中因为存在发展的历史问题，所以有些地方还会使用到Enumeration输出

二期必须注意的是在使用Enumeration输出的时候一般都是直接操作Vector类完成的

import java.util.Vector;

import java.util.Enumeration;

public class EnumerationDemo01{

public static void main(String args[]){

Vector<String> all = new Vector<String>() ;

all.add("hello") ;

all.add("\_") ;

all.add("world") ;

Enumeration<String> enu = all.elements() ;

while(enu.hasMoreElements()){ //判断是否有内容，hasNext()

System.out.print(enu.nextElement() + "、") ; // 输出元素：next()

}

}

};

2016年8月13日22:09:53

**1. Map接口**

Map接口与Collection接口的不同：

Collection接口每次保存的是一个对象，但是在Map接口中保存的是一对对象，以键值对的形式保存。

public interface Map<K,V>

Map接口常用子类：

HashMap: 无序存放的，是新的操作类，key不允许重复

HashTable：无序存放的，是旧的操作类，key不允许重复

TreeMap：可以排序的Map集合，按集合中的key排序，key不允许重复

WeakHashMap： 弱引用的Map集合，当集合中的某些内容不再使用时，可以清除掉无用的数据，可以使用gc进行回收。

IdentityHashMap： key可以重复的Map集合

增加内容用put方法，取出内容用get方法。

import java.util.HashMap ;

import java.util.Map ;

public class HashMapDemo01{

public static void main(String args[]){

Map<String,String> map = null; // 声明Map对象，其中key和value的类型为String

map = new HashMap<String,String>() ;

map.put("mldn","www.mldn.cn") ; // 增加内容

map.put("zhinangtuan","www.zhinangtuan.net.cn") ; // 增加内容

map.put("mldnjava","www.mldnjava.cn") ; // 增加内容

String val = map.get("mldn") ; // 根据key取出值

System.out.println("取出的内容是：" + val) ;

}

};

判断键是否存在用containsKey()方法

判断值是否存在用containsValue方法

import java.util.HashMap ;

import java.util.Map ;

public class HashMapDemo02{

public static void main(String args[]){

Map<String,String> map = null; // 声明Map对象，其中key和value的类型为String

map = new HashMap<String,String>() ;

map.put("mldn","www.mldn.cn") ; // 增加内容

map.put("zhinangtuan","www.zhinangtuan.net.cn") ; // 增加内容

map.put("mldnjava","www.mldnjava.cn") ; // 增加内容

if(map.containsKey("mldn")){ // 判断key是否存在

System.out.println("搜索的key存在！") ;

}else{

System.out.println("搜索的key不存在！") ;

}

if(map.containsValue("www.mldn.cn")){ // 判断value是否存在

System.out.println("搜索的value存在！") ;

}else{

System.out.println("搜索的value不存在！") ;

}

}

};

**如果要输出全部的key，可以使用keySet方法**

Set<K> keySet()

Map<String,String> map = null;

map= new HashMap<String,String>();

map.put("a","1") ;

map.put("b","2");

Set<String> keys = map.keySet();

Iterator<String> iter = keys.iterator();

while(iter.hasNext())

{

String str = iter.next();

System.out.println(str) ;

};

如果要输出全部的value，可以用vaules方法

Collection<V> valuse()

Collection<String> values = map.values() ; // 得到全部的value

Iterator<String> iter = values.iterator() ;

while(iter.hasNext()){

String str = iter.next() ;

System.out.print(str + "、") ;

}

HashTable类，属于旧类

HashTble与HashMap的区别：

(1) 继承不同

public class Hashtable extends Dictionary implements Map  
public class HashMap extends AbstractMap implements Map

(2)

Hashtable 中的方法是同步的，而HashMap中的方法在缺省情况下是非同步的。在多线程并发的环境下，可以直接使用Hashtable，但是要使用HashMap的话就要自己增加同步处理了。

(3)

Hashtable中，key和value都不允许出现null值。

在HashMap中，null可以作为键，这样的键只有一个；可以有一个或多个键所对应的值为null。当get()方法返回null值时，即可以表示 HashMap中没有该键，也可以表示该键所对应的值为null。因此，在HashMap中不能由get()方法来判断HashMap中是否存在某个键， 而应该用containsKey()方法来判断。

(4)

两个遍历方式的内部实现上不同。

Hashtable、HashMap都使用了 Iterator。而由于历史原因，Hashtable还使用了Enumeration的方式

(5)

哈希值的使用不同，HashTable直接使用对象的hashCode。而HashMap重新计算hash值。

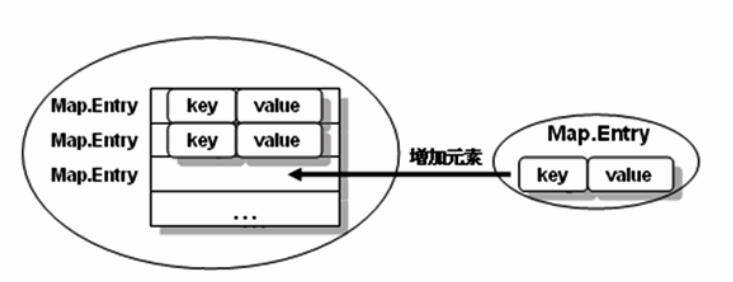
(6)

Hashtable和HashMap它们两个内部实现方式的数组的初始大小和扩容的方式。HashTable中hash数组默认大小是11，增加的方式是 old\*2+1。HashMap中hash数组的默认大小是16，而且一定是2的指数。

2. Map.Entry接口

public static interface Map.Entry<K,V>

Map与Map.Entry



一个例子：

Map<Marker, Collectpoint\_Attribute> markerlist = CollectpointActivity.*markerlist*;  
Iterator iter = markerlist.entrySet().iterator();  
**while** (iter.hasNext()) {  
 Map.Entry entry = (Map.Entry) iter.next();  
 Object key = entry.getKey();  
 Object val = entry.getValue();  
  
 temp\_attr = (Collectpoint\_Attribute) val;  
 **if** (temp\_attr.getUuid\_attr().equals(temp\_uuid)) { *//字符串的比较要equals，不能用==* temp\_marker = (Marker) key;  
 **break**;  
 }  
}

2. TreeMap子类

此类属于排序类

import java.util.TreeMap ;

import java.util.Map ;

import java.util.Set ;

import java.util.Iterator ;

import java.util.Collection ;

public class TreeMapDemo01{

public static void main(String args[]){

Map<String,String> map = null; // 声明Map对象，其中key和value的类型为String

map = new TreeMap<String,String>() ;

map.put("A、mldn","www.mldn.cn") ; // 增加内容

map.put("C、zhinangtuan","www.zhinangtuan.net.cn") ; // 增加内容

map.put("B、mldnjava","www.mldnjava.cn") ; // 增加内容

Set<String> keys = map.keySet() ; // 得到全部的key

Iterator<String> iter = keys.iterator() ;

while(iter.hasNext()){

String str = iter.next() ;

System.out.println(str + " --> " + map.get(str)) ; // 取出内容

}

}

};

输出：

A、mldn --> www.mldn.cn

B、mldnjava --> www.mldnjava.cn

C、zhinangtuan --> www.zhinangtuan.net.cn

3. 弱引用类WeakHashMap

之前所讲解的Map子类中的数据都是强引用保存的，即里面的内容哦哦不管是否使用都始终在集合中保留，如果希望集合可以自动清理暂时不用的数据就可以使用WeakHashMap类

import java.util.WeakHashMap ;

import java.util.Map ;

import java.util.Set ;

import java.util.Iterator ;

import java.util.Collection ;

public class WeakHashMapDemo01{

public static void main(String args[]){

Map<String,String> map = null; // 声明Map对象，其中key和value的类型为String

map = new WeakHashMap<String,String>() ;

map.put(new String("mldn"),new String("www.mldn.cn")) ;

map.put(new String("zhinangtuan"),new String("www.zhinangtuan.net.cn")) ;

map.put(new String("mldnjava"),new String("www.mldnjava.cn")) ;

System.gc() ; // 强制性进行垃圾的收集操作

map.put(new String("lxh"),new String("lixinghua")) ;

System.out.println(map) ;

}

};

**Map接口输出**

1. 对于Map接口来说，其本身是不能直接使用迭代(如：Iterator、foreach)进行输出，因为Map中每个位置存放的是一对值，而Iterator中每次只能找到一个值。所以如果非要使用迭代输出的话，得按照以下方法(iterator为例)

(1) 将Map的实例通过entrySet()方法变为Set接口对象

(2) 通过Set接口实例为Iterator实例化

(3) 通过Iterator迭代输出，每个内容都是Map.Entry的对象

(4) 通过Map.Entry进行key->valuse的分离

Map map = new HashMap();  
map.put("Rajib Sarma","100");  
map.put("Rajib Sarma","200");//The value "100" is replaced by "200".  
map.put("Sazid Ahmed","200");  
  
Iterator iter = map.entrySet().iterator();  
while (iter.hasNext()) {  
 Map.Entry entry = (Map.Entry) iter.next();  
 Object key = entry.getKey();  
 Object val = entry.getValue();  
}

因为Set接口是Collection的子接口，因此可以使用Iterator接口来输出

Map<String,String> map = null; // 声明Map对象，其中key和value的类型为String

map = new HashMap<String,String>() ;

map.put("mldn","www.mldn.cn") ; // 增加内容

map.put("zhinangtuan","www.zhinangtuan.net.cn") ; // 增加内容

map.put("mldnjava","www.mldnjava.cn") ; // 增加内容

Set<Map.Entry<String,String>> allSet = null ;

allSet = map.entrySet() ;

Iterator<Map.Entry<String,String>> iter = null ;

iter = allSet.iterator() ;

while(iter.hasNext()){

Map.Entry<String,String> me = iter.next() ;

System.out.println(me.getKey() + " --> " + me.getValue()) ;

}

**2.使用foreach输出**

Map<String,String> map = null; // 声明Map对象，其中key和value的类型为String

map = new HashMap<String,String>() ;

map.put("mldn","www.mldn.cn") ; // 增加内容

map.put("zhinangtuan","www.zhinangtuan.net.cn") ; // 增加内容

map.put("mldnjava","www.mldnjava.cn") ; // 增加内容

for(Map.Entry<String,String> me:map.entrySet()){

System.out.println(me.getKey() + " --> " + me.getValue()) ;

}

如果使用非系统类作为Map的key，则此类必须覆写Object类的hashCode和equals方法

import java.util.Map ;

import java.util.HashMap ;

class Person{

private String name ;

private int age ;

public Person(String name,int age){

this.name = name ;

this.age = age ;

}

public String toString(){

return "姓名：" + this.name + "；年龄：" + this.age ;

}

public boolean equals(Object obj){

if(this==obj){

return true ;

}

if(!(obj instanceof Person)){

return false ;

}

Person p = (Person)obj ;

if(this.name.equals(p.name)&&this.age==p.age){

return true ;

}else{

return false ;

}

}

public int hashCode(){

return this.name.hashCode() \* this.age ;

}

};

public class HashMapDemo08{

public static void main(String args[]){

Map<Person,String> map = null ;

map = new HashMap<Person,String>() ;

map.put(new Person("张三",30),"zhangsan"); // 增加内容

System.out.println(map.get(new Person("张三",30))) ;

}

};

如果没有覆写equals和hashcode方法，直接使用匿名对象，则new出来的对象即使和之前的对象内容完全一样，也不是同一个对象，那么map中get到的只能是null。

hashcoe代表一个编码，如果编码相等并且属性相等，则认为是同一个对象。

作为对象的时候，实际上是依靠hashCode()和equls()来判断两个匿名对象是否相等的，这一点由系统内部自动完成。

复习：

==和equals都表示地址是否相等，但是在String中重写了equals，那么在String中，equals表示内容是否相等，如果两个字符串内容相等，那么equals为true，==为false。

总结：

Map可以使用迭代输出：

map🡪 entrySet🡪Set🡪Iterator🡪Map.Entry🡪key和value

如果使用非系统类作为key，则一定保证覆写equals和hashCode方法，否则无效。

2016年8月14日01:30:00

1. IdentityHashMap类

在正常的map中，key是不允许重复的。如果想要重复，可以使用该类，这个类在实际中用的比较少。

import java.util.IdentityHashMap ;

import java.util.Set ;

import java.util.Iterator ;

import java.util.Map ;

class Person{

private String name ;

private int age ;

public Person(String name,int age){

this.name = name ;

this.age = age ;

}

public boolean equals(Object obj){

if(this==obj){

return true ;

}

if(!(obj instanceof Person)){

return false ;

}

Person p = (Person)obj ;

if(this.name.equals(p.name)&&this.age==p.age){

return true ;

}else{

return false ;

}

}

public int hashCode(){

return this.name.hashCode() \* this.age ;

}

public String toString(){

return "姓名：" + this.name + "，年龄：" + this.age ;

}

};

public class IdentityHashMapDemo02{

public static void main(String args[]){

Map<Person,String> map = null ; // 声明Map对象

map = new IdentityHashMap<Person,String>() ;

map.put(new Person("张三",30),"zhangsan\_1") ; // 加入内容

map.put(new Person("张三",30),"zhangsan\_2") ; // 加入内容

map.put(new Person("李四",31),"lisi") ; // 加入内容

Set<Map.Entry<Person,String>> allSet = null ; // 准备使用Set接收全部内容

allSet = map.entrySet() ;

Iterator<Map.Entry<Person,String>> iter = null ;

iter = allSet.iterator() ;

while(iter.hasNext()){

Map.Entry<Person,String> me = iter.next() ;

System.out.println(me.getKey() + " --> " + me.getValue()) ;

}

}

};

2016年8月14日12:35:43

1. SortedMap类

回顾：SortedSet是TreeSet的实现接口，那么此类可以进行排序操作

sortedMap也是排序操作，之前学过treemap，那么此类也可以排序。

import java.util.Map ;

import java.util.SortedMap ;

import java.util.TreeMap ;

public class SortedMapDemo{

public static void main(String args[]){

SortedMap<String,String> map = null ;

map = new TreeMap<String,String>() ; // 通过子类实例化接口对象

map.put("D、jiangker","http://www.jiangker.com/") ;

map.put("A、mldn","www.mldn.cn") ;

map.put("C、zhinangtuan","www.zhinangtuan.net.cn") ;

map.put("B、mldnjava","www.mldnjava.cn") ;

System.out.print("第一个元素的内容的key：" + map.firstKey()) ;

System.out.println("：对应的值：" + map.get(map.firstKey())) ;

System.out.print("最后一个元素的内容的key：" + map.lastKey()) ;

System.out.println("：对应的值：" + map.get(map.lastKey())) ;

System.out.println("返回小于指定范围的集合：") ;

for(Map.Entry<String,String> me:map.headMap("B、mldnjava").entrySet()){

System.out.println("\t|- " + me.getKey() + " --> " + me.getValue()) ;

}

System.out.println("返回大于指定范围的集合：") ;

for(Map.Entry<String,String> me:map.tailMap("B、mldnjava").entrySet()){

System.out.println("\t|- " + me.getKey() + " --> " + me.getValue()) ;

}

System.out.println("部分集合：") ;

for(Map.Entry<String,String> me:map.subMap("A、mldn","C、zhinangtuan").entrySet()){

System.out.println("\t|- " + me.getKey() + " --> " + me.getValue()) ;

}

}

};

输出：

第一个元素的内容的key：A、mldn：对应的值：www.mldn.cn

最后一个元素的内容的key：D、jiangker：对应的值：http://www.jiangker.com/

返回小于指定范围的集合：

|- A、mldn --> www.mldn.cn

返回大于指定范围的集合：

|- B、mldnjava --> www.mldnjava.cn

|- C、zhinangtuan --> www.zhinangtuan.net.cn

|- D、jiangker --> http://www.jiangker.com/

部分集合：

|- A、mldn --> www.mldn.cn

|- B、mldnjava --> www.mldnjava.cn

2. 集合工具类Collections

Collections与Collection的区别

没有直接关系，但是与集合中的各个接口都有操作的方法支持。

空集合操作

import java.util.Collections ;

import java.util.List ;

import java.util.Set ;

public class CollectionsDemo01{

public static void main(String args[]){

List<String> allList = Collections.emptyList() ; // 返回空的 List集合

Set<String> allSet = Collections.emptySet() ; // 返回空的 List集合

allList.add("Hello") ; // 加入数据

}

};

上面的代码不能添加数据

import java.util.Collections ;

import java.util.List ;

import java.util.ArrayList ;

import java.util.Iterator ;

public class CollectionsDemo02{

public static void main(String args[]){

List<String> all = new ArrayList<String>() ; // 返回空的 List集合

Collections.addAll(all,"MLDN","LXH","mldnjava") ;

Iterator<String> iter = all.iterator() ;

while(iter.hasNext()){

System.out.print(iter.next() + "、") ;

}

}

};

一般直接使用list的添加方法，很少直接使用Collections的addAll方法

**反转操作**

import java.util.Collections ;

import java.util.List ;

import java.util.ArrayList ;

import java.util.Iterator ;

public class CollectionsDemo03{

public static void main(String args[]){

List<String> all = new ArrayList<String>() ; // 返回空的 List集合

Collections.addAll(all,"MLDN","LXH","mldnjava") ;

Collections.reverse(all) ; // 内容反转

Iterator<String> iter = all.iterator() ;

while(iter.hasNext()){

System.out.print(iter.next() + "、") ;

}

}

};

输出：mldnjava、LXH、MLDN、

**二分检索方法 binarySearch**

import java.util.Collections ;

import java.util.List ;

import java.util.ArrayList ;

import java.util.Iterator ;

public class CollectionsDemo04{

public static void main(String args[]){

List<String> all = new ArrayList<String>() ; // 返回空的 List集合

Collections.addAll(all,"MLDN","LXH","mldnjava") ;

int point = Collections.binarySearch(all,"LXH") ; // 检索数据

System.out.println("检索结果：" + point) ;

}

};

返回位置：1

**替换指定内容 replaceAll**

import java.util.Collections ;

import java.util.List ;

import java.util.ArrayList ;

import java.util.Iterator ;

public class CollectionsDemo05{

public static void main(String args[]){

List<String> all = new ArrayList<String>() ; // 返回空的 List集合

Collections.addAll(all,"MLDN","LXH","mldnjava") ;

if(Collections.replaceAll(all,"LXH","李兴华")){// 替换内容

System.out.println("内容替换成功！") ;

}

System.out.print("替换之后的结果：") ;

System.out.print(all) ;

}

};

输出：

内容替换成功！

替换之后的结果：[MLDN, 李兴华, mldnjava]

排序操作:

import java.util.Collections ;

import java.util.List ;

import java.util.ArrayList ;

import java.util.Iterator ;

public class CollectionsDemo06{

public static void main(String args[]){

List<String> all = new ArrayList<String>() ; // 返回空的 List集合

Collections.addAll(all,"1、MLDN","2、LXH","3、mldnjava") ;

Collections.addAll(all,"B、www.mldn.cn") ;

Collections.addAll(all,"A、www.mldnjava.cn") ;

System.out.println("排序之前的集合：" + all) ;

Collections.sort(all) ;

System.out.println("排序之后的集合：" + all) ;

}

};

输出：

排序之前的集合：[1、MLDN, 2、LXH, 3、mldnjava, B、www.mldn.cn, A、www.mldnjava.cn]

排序之后的集合：[1、MLDN, 2、LXH, 3、mldnjava, A、www.mldnjava.cn, B、www.mldn.cn]

排序操作也需要依靠Comparable接口

**交换指定位置的内容：**

import java.util.Collections ;

import java.util.List ;

import java.util.ArrayList ;

import java.util.Iterator ;

public class CollectionsDemo07{

public static void main(String args[]){

List<String> all = new ArrayList<String>() ; // 返回空的 List集合

Collections.addAll(all,"1、MLDN","2、LXH","3、mldnjava") ;

System.out.println("交换之前的集合：" + all) ;

Collections.swap(all,0,2) ;

System.out.println("交换之后的集合：" + all) ;

}

};

输出：

交换之前的集合：[1、MLDN, 2、LXH, 3、mldnjava]

交换之后的集合：[3、mldnjava, 2、LXH, 1、MLDN]

**2. Stack类**

栈是采用现金后出的操作方式

Stack类是Vector的子类

public class Stack<E> extends Vector<E>

常用方法：

public boolean empty() 测试栈是否为空

public E peek() 查看栈顶，但不删除

public E pop() 出栈，同时删除

public E push(E item) 入栈

public int search(Object o)在栈中查找

import java.util.Stack ;

public class StackDemo{

public static void main(String args[]){

Stack<String> s = new Stack<String>() ;

s.push("A") ; // 入栈

s.push("B") ; // 入栈

s.push("C") ; // 入栈

System.out.print(s.pop() + "、") ;

System.out.print(s.pop() + "、") ;

System.out.println(s.pop() + "、") ;

System.out.println(s.pop()) ;

}

};>

输出：

C、B、A、

Exception in thread "main" java.util.EmptyStackException

at java.util.Stack.peek(Unknown Source)

at java.util.Stack.pop(Unknown Source)

at StackDemo.main(StackDemo.java:11)

**3. Properties类**

public class Properties extends HashTable<Object,Object>

Properties 是HashTable的子类，则也是Map的子类，可以使用Map的全部操作，但是一般情况下属性类是单独使用的。

设置属性：

public Object setProperty(String key,String value)

得到属性：

public String getProperty(String key,String defaultValue)

import java.util.Properties;

public class PropertiesDemo01{

public static void main(String args[]){

Properties pro = new Properties() ; // 创建Properties对象

pro.setProperty("BJ","BeiJing") ; // 设置属性

pro.setProperty("TJ","TianJin") ;

pro.setProperty("NJ","NanJing") ;

System.out.println("1、BJ属性存在：" + pro.getProperty("BJ")) ;

System.out.println("2、SC属性不存在：" + pro.getProperty("SC")) ;

System.out.println("3、SC属性不存在，同时设置显示的默认值：" + pro.getProperty("SC","没有发现")) ;

}

};

将属性保存到文件中

import java.util.Properties;

import java.io.File;

import java.io.FileOutputStream;

import java.io.FileNotFoundException;

import java.io.IOException;

public class PropertiesDemo02{

public static void main(String args[]){

Properties pro = new Properties() ; // 创建Properties对象

pro.setProperty("BJ","BeiJing") ; // 设置属性

pro.setProperty("TJ","TianJin") ;

pro.setProperty("NJ","NanJing") ;

File file = new File("D:" + File.separator + "area.properteis") ; // 指定要操作的文件

try{

pro.store(new FileOutputStream(file),"Area Info") ; // 保存属性到普通文件

}catch(FileNotFoundException e){

e.printStackTrace() ;

}catch(IOException e){

e.printStackTrace() ;

}

}

};

从文件中读取属性

import java.util.Properties;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.IOException;

public class PropertiesDemo03{

public static void main(String args[]){

Properties pro = new Properties() ; // 创建Properties对象

File file = new File("D:" + File.separator + "area.properteis") ; // 指定要操作的文件

try{

pro.load(new FileInputStream(file)) ; // 读取属性文件

}catch(FileNotFoundException e){

e.printStackTrace() ;

}catch(IOException e){

e.printStackTrace() ;

}

System.out.println("1、BJ属性存在：" + pro.getProperty("BJ")) ;

System.out.println("2、SH属性存在：" + pro.getProperty("SH")) ;

}

};

将属性保存到xml中

import java.util.Properties;

import java.io.File;

import java.io.FileOutputStream;

import java.io.FileNotFoundException;

import java.io.IOException;

public class PropertiesDemo04{

public static void main(String args[]){

Properties pro = new Properties() ; // 创建Properties对象

pro.setProperty("BJ","BeiJing") ; // 设置属性

pro.setProperty("TJ","TianJin") ;

pro.setProperty("NJ","NanJing") ;

File file = new File("D:" + File.separator + "area.xml") ; // 指定要操作的文件

try{

pro.storeToXML(new FileOutputStream(file),"Area Info") ; // 保存属性到普通文件

}catch(FileNotFoundException e){

e.printStackTrace() ;

}catch(IOException e){

e.printStackTrace() ;

}

}

};

从xml中读取属性

import java.util.Properties;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.IOException;

public class PropertiesDemo05{

public static void main(String args[]){

Properties pro = new Properties() ; // 创建Properties对象

File file = new File("D:" + File.separator + "area.xml") ; // 指定要操作的文件

try{

pro.loadFromXML(new FileInputStream(file)) ; // 读取属性文件

}catch(FileNotFoundException e){

e.printStackTrace() ;

}catch(IOException e){

e.printStackTrace() ;

}

System.out.println("1、BJ属性存在：" + pro.getProperty("BJ")) ;

}

};

**4. 一对多关系**

使用类集可以表示出一下的关系：一个学校可以包含多个学生，一个学生属于一学校，那么这就是一个典型的一对多关系，此时就可以通过类集进行关系的表示。

import java.util.List ;

import java.util.ArrayList ;

public class School{

private String name ;

private List<Student> allStudents ;

public School(){

this.allStudents = new ArrayList<Student>() ;

}

public School(String name){

this() ;

this.setName(name) ;

}

public void setName(String name){

this.name = name ;

}

public String getName(){

return this.name;

}

public List<Student> getAllStudents(){

return this.allStudents ;

}

public String toString(){

return "学校名称：" + this.name ;

}

};

public class Student{

private String name ;

private int age ;

private School school; // 一个学生属于一个学校

public Student(String name,int age){

this.setName(name) ;

this.setAge(age) ;

}

public void setSchool(School school){

this.school = school ;

}

public School getSchool(){

return this.school ;

}

public void setName(String name){

this.name = name ;

}

public void setAge(int age){

this.age = age ;

}

public String getName(){

return this.name;

}

public int getAge(){

return this.age ;

}

public String toString(){

return "学生姓名：" + this.name + "；年龄：" + this.age ;

}

};

import java.util.Iterator ;

public class TestDemo{

public static void main(String args[]){

School sch = new School("清华大学") ; // 定义学校

Student s1 = new Student("张三",21) ;

Student s2 = new Student("李四",22) ;

Student s3 = new Student("王五",23) ;

sch.getAllStudents().add(s1) ;

sch.getAllStudents().add(s2) ;

sch.getAllStudents().add(s3) ;

s1.setSchool(sch) ;

s2.setSchool(sch) ;

s3.setSchool(sch) ;

System.out.println(sch) ;

Iterator<Student> iter = sch.getAllStudents().iterator() ;

while(iter.hasNext()){

System.out.println("\t|- " + iter.next()) ;

}

}

};

5. 多对多关系

import java.util.List ;

import java.util.ArrayList ;

public class Course{

private String name ;

private int credit ;

private List<Student> allStudents ;

public Course(){

this.allStudents = new ArrayList<Student>() ;

}

public Course(String name,int credit){

this() ;

this.name = name ;

this.credit = credit ;

}

public List<Student> getAllStudents(){

return this.allStudents ;

}

public void setName(String name){

this.name = name ;

}

public void setCredit(int credit){

this.credit = credit ;

}

public String getName(){

return this.name ;

}

public int getCredit(){

return this.credit ;

}

public String toString(){

return "课程名称：" + this.name + "；课程学分：" + this.credit ;

}

};

import java.util.List ;

import java.util.ArrayList ;

public class Student{

private String name ;

private int age ;

private List<Course> allCourses ;

public Student(){

this.allCourses = new ArrayList<Course>() ;

}

public Student(String name,int age){

this() ;

this.name = name ;

this.age = age ;

}

public List<Course> getAllCourses(){

return this.allCourses ;

}

public void setName(String name){

this.name = name ;

}

public void setAge(int age){

this.age = age ;

}

public String getName(){

return this.name ;

}

public int getAge(){

return this.age ;

}

public String toString(){

return "学生姓名：" + this.name + "；年龄：" + this.age ;

}

};

import java.util.Iterator ;

public class TestMore{

public static void main(String args[]){

Course c1 = new Course("英语",3 ) ; // 第一门课程

Course c2 = new Course("计算机",5) ; // 第二门课程

Student s1 = new Student("张三",20) ;

Student s2 = new Student("李四",21) ;

Student s3 = new Student("王五",22) ;

Student s4 = new Student("赵六",23) ;

Student s5 = new Student("孙七",24) ;

Student s6 = new Student("钱八",24) ;

// 第一门课程有三个学生参加

c1.getAllStudents().add(s1) ;

c1.getAllStudents().add(s2) ;

c1.getAllStudents().add(s6) ;

s1.getAllCourses().add(c1) ;

s2.getAllCourses().add(c1) ;

s6.getAllCourses().add(c1) ;

// 第二门课程有六个学生参加

c2.getAllStudents().add(s1) ;

c2.getAllStudents().add(s2) ;

c2.getAllStudents().add(s3) ;

c2.getAllStudents().add(s4) ;

c2.getAllStudents().add(s5) ;

c2.getAllStudents().add(s6) ;

s1.getAllCourses().add(c2) ;

s2.getAllCourses().add(c2) ;

s3.getAllCourses().add(c2) ;

s4.getAllCourses().add(c2) ;

s5.getAllCourses().add(c2) ;

s6.getAllCourses().add(c2) ;

// 输出一门课程的信息，观察一门课程有多少个学生参加\

System.out.println(c1) ;

Iterator<Student> iter1 = c1.getAllStudents().iterator() ;

while(iter1.hasNext()){

Student s = iter1.next() ;

System.out.println("\t|- " + s) ;

}

// 通过学生找到学生参加的课程

System.out.println(s6) ;

Iterator<Course> iter2 = s6.getAllCourses().iterator() ;

while(iter2.hasNext()){

Course c = iter2.next() ;

System.out.println("\t|- " + c) ;

}

}

};

2016年8月14日19:10:04

**1. 枚举 enum**

所谓的枚举就是规定好了指定的取值范围，所有的内容只能从指定的范围中取得。

public enum Color{

RED,GREEN,BLUE ; // 定义三个枚举的类型

};

public class GetEnumContent{

public static void main(String args[]){

Color c = Color.BLUE ; // 取出蓝色

System.out.println(c) ;

}

};

public class PrintEnum{

public static void main(String args[]){

for(Color c:Color.values()){ // 输出枚举中的全部内容

System.out.println(c) ;

}

}

};

**2. enum关键字和Enum类的关系**

使用enum关键字可以定义一个枚举，实际上此关键字表示的是java.lang.Enum类型，即：使用enmu声明的枚举类型，就相当于定义一个类，而此类则默认继承java.lang.Enum类。

public abstract class **Enum<E extends Enum<E>>**

extends [Object](http://tool.oschina.net/uploads/apidocs/jdk_7u4/java/lang/Object.html)

implements [Comparable](http://tool.oschina.net/uploads/apidocs/jdk_7u4/java/lang/Comparable.html)<E>, [Serializable](http://tool.oschina.net/uploads/apidocs/jdk_7u4/java/io/Serializable.html)

此类定义的时候使用了泛型机制，而且实现了Comparable接口以及Serialzable接口，证明此种类型是可以比较，可以被序列化的。

构造方法：

protected Enum([String](http://tool.oschina.net/uploads/apidocs/jdk_7u4/java/lang/String.html) name, int ordinal)

不能调用这个构造方法

public class GetEnumInfo{

public static void main(String args[]){

for(Color c:Color.values()){

System.out.println(c.ordinal() + " --> " + c.name()) ;

}

}

};

输出：

0 --> RED

1 --> GREEN

2 --> BLUE

enum Color{

RED("红色"),GREEN("绿色"),BLUE("兰色") ;

private Color(String name){

this.setName(name) ;

}

private String name ; // 定义name属性

public void setName(String name){

this.name = name ;

}

public String getName(){

return this.name ;

}

}

public class ConstructorEnum{

public static void main(String args[]){

for(Color c:Color.values()){

System.out.println(c.ordinal() + " --> " + c.name()

+"(" + c.getName() + ")") ;

}

}

};

输出：

0 --> RED(红色)

1 --> GREEN(绿色)

2 --> BLUE(兰色)

enum Color{

RED,GREEN,BLUE ;

private String name ; // 定义name属性

public void setName(String name){

switch(this){ // 判断操作的是那个枚举对象

case RED:{

if("红色".equals(name)){

this.name = name ; // 允许设置名字

}else{

System.out.println("设置内容错误。") ;

}

break ;

}

case GREEN:{

if("绿色".equals(name)){

this.name = name ; // 允许设置名字

}else{

System.out.println("设置内容错误。") ;

}

break ;

}

case BLUE:{

if("蓝色".equals(name)){

this.name = name ; // 允许设置名字

}else{

System.out.println("设置内容错误。") ;

}

break ;

}

}

this.name = name ;

}

public String getName(){

return this.name ;

}

}

public class SetEnum{

public static void main(String args[]){

Color c = Color.BLUE ; // 得到兰色

c.setName("兰色") ; // 名字错误

c.setName("蓝色") ; // 名字正确

System.out.println(c.getName()) ;

}

};

排序：

import java.util.Iterator;

import java.util.Set ;

import java.util.TreeSet ;

enum Color{

RED,GREEN,BLUE ;

}

public class ComparableEnum{

public static void main(String args[]){

Set<Color> t = new TreeSet<Color>() ; // 设置类型

t.add(Color.GREEN) ; // 加入绿色

t.add(Color.RED) ; // 加入红色

t.add(Color.BLUE) ; // 加入蓝色

Iterator<Color> iter = t.iterator() ;

while(iter.hasNext()){

System.out.print(iter.next() + "、") ;

}

}

};

输出：

RED、GREEN、BLUE、

**3.类集对Enum的支持**

EnumMap和EnumSet

在java.util.EnumMap包中

public class **EnumMap<K extends** [**Enum**](http://docs.oracle.com/javase/7/docs/api/java/lang/Enum.html)**<K>,V>**

extends [AbstractMap](http://docs.oracle.com/javase/7/docs/api/java/util/AbstractMap.html)<K,V>

implements [Serializable](http://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html), [Cloneable](http://docs.oracle.com/javase/7/docs/api/java/lang/Cloneable.html)

键必须是枚举类型

import java.util.EnumMap ;

import java.util.Map ;

enum Color{

RED , GREEN , BLUE ;

}

public class EnumMapDemo{

public static void main(String args[]){

Map<Color,String> desc = null ; // 定义Map对象，同时指定类型

desc = new EnumMap<Color,String>(Color.class) ; // 实例化EnumMap对象

desc.put(Color.RED,"红色") ;

desc.put(Color.GREEN,"绿色") ;

desc.put(Color.BLUE,"蓝色") ;

System.out.println("====== 输出全部的内容 ======") ;

for(Color c:Color.values()){

System.out.println(c.name() + " --> " + desc.get(c)) ;

}

System.out.println("====== 输出全部的键值 ======") ;

for(Color c:desc.keySet()){

System.out.print(c.name() + "、") ;

}

System.out.println() ;

System.out.println("====== 输出全部的内容 ======") ;

for(String s:desc.values()){

System.out.print(s + "、") ;

}

}

};

====== 输出全部的内容 ======

RED --> 红色

GREEN --> 绿色

BLUE --> 蓝色

====== 输出全部的键值 ======

RED、GREEN、BLUE、

====== 输出全部的内容 ======

红色、绿色、蓝色、

* public abstract class EnumSet<E extends [**Enum**](file:///D:\jdk-8u101-docs-all\docs\api\java\lang\Enum.html)<E>>
* extends [AbstractSet](file:///D:\jdk-8u101-docs-all\docs\api\java\util\AbstractSet.html)<E>

implements [Cloneable](file:///D:\jdk-8u101-docs-all\docs\api\java\lang\Cloneable.html), [Serializable](file:///D:\jdk-8u101-docs-all\docs\api\java\io\Serializable.html)

EnumSet是Set的子类，所以里面的内容是无法重复的，在使用EnumSet的时候是不能直接使用关键字new为其进行实例化的，所以在此类中提供了很多的静态方法。

**将全部的集合设置到EnumSet集合中**

import java.util.EnumSet ;

enum Color{

RED , GREEN , BLUE ;

}

public class EnumSetDemo01{

public static void main(String args[]){

EnumSet<Color> es = null ; // 声明一个EnumSet对象

System.out.println("======== EnumSet.allOf(Color.class) =====") ;

es = EnumSet.allOf(Color.class) ; // 将枚举的全部类型设置到EnumSet对象之中

print(es) ;

}

public static void print(EnumSet<Color> temp){ // 专门的输出操作

for(Color c:temp){ // 循环输出EnumSet中的内容

System.out.print(c + "、") ;

}

System.out.println() ;

}

};

**只设置一个内容到集合**

import java.util.EnumSet ;

enum Color{

RED , GREEN , BLUE ;

}

public class EnumSetDemo02{

public static void main(String args[]){

EnumSet<Color> es = null ; // 声明一个EnumSet对象

System.out.println("======== EnumSet.of(Color.BLUE) =====") ;

es = EnumSet.of(Color.BLUE) ; // 将枚举的全部类型设置到EnumSet对象之中

print(es) ;

}

public static void print(EnumSet<Color> temp){ // 专门的输出操作

for(Color c:temp){ // 循环输出EnumSet中的内容

System.out.print(c + "、") ;

}

System.out.println() ;

}

};

创建只能放入指定枚举类型的集合

import java.util.EnumSet ;

enum Color{

RED , GREEN , BLUE ;

}

public class EnumSetDemo03{

public static void main(String args[]){

EnumSet<Color> es = null ; // 声明一个EnumSet对象

System.out.println("======== EnumSet.noneOf(Color.class) =====") ;

es = EnumSet.noneOf(Color.class) ; // 将枚举的全部类型设置到EnumSet对象之中

es.add(Color.RED) ; // 增加内容

es.add(Color.GREEN) ; // 增加内容

print(es) ;

}

public static void print(EnumSet<Color> temp){ // 专门的输出操作

for(Color c:temp){ // 循环输出EnumSet中的内容

System.out.print(c + "、") ;

}

System.out.println() ;

}

};

创建不包含指定元素的集合

import java.util.EnumSet ;

enum Color{

RED , GREEN , BLUE ;

}

public class EnumSetDemo04{

public static void main(String args[]){

EnumSet<Color> esOld = null ; // 声明一个EnumSet对象

EnumSet<Color> esNew = null ;

System.out.println("======== EnumSet.complementOf(Color.class) =====") ;

esOld = EnumSet.noneOf(Color.class) ; // 将枚举的全部类型设置到EnumSet对象之中

esOld.add(Color.RED) ; // 增加内容

esOld.add(Color.GREEN) ; // 增加内容

esNew = EnumSet.complementOf(esOld) ; // 不包含指定内容

print(esNew) ;

}

public static void print(EnumSet<Color> temp){ // 专门的输出操作

for(Color c:temp){ // 循环输出EnumSet中的内容

System.out.print(c + "、") ;

}

System.out.println() ;

}

};

输出新的集合只包含蓝色，不包含esOld中的红和绿。

拷贝一个集合的内容

import java.util.EnumSet ;

enum Color{

RED , GREEN , BLUE ;

}

public class EnumSetDemo05{

public static void main(String args[]){

EnumSet<Color> esOld = null ; // 声明一个EnumSet对象

EnumSet<Color> esNew = null ;

System.out.println("======== EnumSet.copyOf(Color.class) =====") ;

esOld = EnumSet.noneOf(Color.class) ; // 将枚举的全部类型设置到EnumSet对象之中

esOld.add(Color.RED) ; // 增加内容

esOld.add(Color.GREEN) ; // 增加内容

esNew = EnumSet.copyOf(esOld) ; // 从已有的集合拷贝过来

print(esNew) ;

}

public static void print(EnumSet<Color> temp){ // 专门的输出操作

for(Color c:temp){ // 循环输出EnumSet中的内容

System.out.print(c + "、") ;

}

System.out.println() ;

}

};

总结：

EnumMap是符合Map的操作形式的，而EnumSet基本上都是使用静态方法完成的。

在操作中大量使用了枚举.class,这属于java反射机制。

4. 枚举的其他应用

**2016年8月9日10:46:52**

**1. Annotation注释**

系统自带的annotation

@Override : 覆写

@Deprecated :不赞成使用

@ SuppressWarnings:压制安全警告

1.1 override的作用是保证覆写的正确性

覆写父类的方法的时候，有可能方法写错了，加上@Override后，如果覆写错误，编译会提示错误。

1.2 添加Deprecated注解后，在调用这个方法编译会出现安全警告

该注解不仅可以用在方法前，还可以用在类前面，表示类也不能使用了

1.3 SuppressWarnings

@SuppressWarnings("unchecked")

public static void main(String[] args)

{

Demo d= new Demo(); //没有指明泛型类型，会出现警告

d.setVar("hello") ;

System.out.println(d.getVar());

}

关键字:

(1). deprecation 使用了不赞成的类或方法的警告

(2). unchecked 执行了未检查的转换时警告，例如：泛型操作中没有指定泛型类型

(3).fallthrough 当switch程序块直接通往下种情况而没有break时的警告

(4).path 在类的路径、源文件路径中有不存在的路径时的警告

(5). serial当在可序列化的类上缺少serialVerisonUID定义是的警告

(6). finally 任何finally子句不能正常完成时的警告

(7). all 关于以上所有情况的警告

使用数组传递更多的关键字

@ SuppressWarnings(value={“unchecked”,”deprecation”})

2016年8月9日17:13:05