# 环境的搭建

JRE ： java运行环境 jre = java虚拟机(jvm) + 核心类库(辅助java虚拟机运行的件)

JDK: java开发工具包 jdk = jre + java的开发工具(编译工具等...)

jdk目录介绍：

bin ： 存储的都是java的开发工具。

db : jdk7.0自带的一个数据。 Derby

include : 存放的调用系统资源的接口文件

jre : java运行环境

src.zip ：源代码

# 命名规范

1.类名与接口名： 所有的单词首字母都大写，其他字母小写。 LastTime 驼峰命名法

2.变量名与函数名: 首单词全部小写，其他单词的首字母大写，其他小写。 比如： getTime()

3.包名： 全部小写。

4.常量名： 全部单词的字母大写，单词与单词之间使用下划线分隔。 比如： UP\_DIRECTION

如何生成软件的开发文档：

使用javadoc开发工具即可生成软件的开发者文档。

javadoc的用法：

javadoc -d 存放文档的路径 java源文件

文档注释要注意的事项：

1. 如果一个类要生成软件开发文档，那么该类要使用public修饰。

2. 文档注释的内容一定要位于类或者是函数的上方，否则无法抽取。

# java中 的八种数据类型

整数 byte 字节 8个二进制 2^8 = 256 范围-128 ~ 127

short 短整型 16bit 2^16 = 65536

int 整型 32bit

long 长整型 64bit

小数

float 单精度浮点型 32bit 7个小数

double 双精度浮点型。 64bit 15个小数

布尔

boolean 1个字节或者4个字节 java白皮书(java规范2.0)

字符

char 16bit '\u0000'空字符

2种写法float d=20.0f

float d=20

String判断运算： "男".equals(g) !"男".equals(g)

String str1=null; str引用为空。

String str2=""; str2引用一个空串。

也就是null没有分配空间，""分配了空间，因此str1还不是一个实例化对象，而str2已经实例化。

null不是对象,""是对象，对象用equals比较，null用等号比较

if(str1==null||str1.equals(""))

打个比方null就像真空，""就像空气。

# 小数存储

float f=9.0f；

十进制9.0→二进制1001.0→1.001\*2^3

float有32位

正数放0 | 放指数3+127=130，即100000| 放小数0.001后面补零00100000000000000000000

第一位放符号位 | 用8位放小数 | 后23位放小数

# 解决小数损失精度的问题

import java.math.BigDecimal

System.out.println(new BigDecimal("2.0").subtract(new BigDecimal("1.1")));

如果用double的换2.0-1.1=0.999999998

如果用new BigDecimal("2.0").subtract(new BigDecimal("1.1"))

2.0-1.1=0.9

# 强制转换

int i=2；

byte b；

b=（int）i；

非基本数据类型的强制转换

String str="100";

int data=Integer.parseInt(str);

String str1="3.14";

double d=Double.parseDouble(str1);

// 二进制转十进制

**int** parseInt4 = Integer.*parseInt*(binaryString, 2);

System.*out*.println(parseInt4);

// 十六进制转十进制

**int** parseInt5 = Integer.*parseInt*(hexString, 16);

System.out.println(parseInt5);

// 八进制转十进制

**int** parseInt6 = Integer.*parseInt*(octalString, 8);

System.out.println(parseInt6);

# 三元运算符

c=a>b?1:2

# java输入

第一步创建扫描器

Scanner sc=new Scanner(System.in);

第二步导包

import java.util.Scanner;

第三步使用扫描器职业方法

int i=sc.nextInt();

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

String mess = br.readLine();

# 修饰符

public 公共修饰符：都能访问

protect保护修饰符：只能访问同一个类、包和子类

default默认修饰符：只能访问同一个类和包

private私有修饰符：只能访问同一个类

# 随机数的生成

产生0~0.99999.....的随机数

Math.random()

如果要产生1~13的随机数

(int)(Math.random()\*13)+1

随机生成4个不同的数

//验证码

**char**[] chars={'1','2','A','B'};

String s="";

Set<Integer> set=**new** HashSet<Integer>();

**int** size;

**while**(set.size()<4){

size=set.size();

**int** i=(**int**)(4\*Math.*random*());

set.add(i);

**if**(size<set.size()){

s+=chars[i];

}

}

System.*out*.println(s);

# 设置延时

Thread.sleep(1000);

throws Exception//抛出异常

# 格式化输出

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

# 数组

数组的创建：int[] arr=new int[3];

数组赋值：int[] arr=new int[]{1,2,3};

数组赋值简化：int[] arr={1,2,3};

二维数组 int[][] arr=new int[3][3];

int []arr[] = new int[3][3]

//错误写法int[] arr=new int[3]{1,2,3};

Arrays.sort(ch)；对char[] ch进行排序

Arrays.copy(Array[] arr,int i)复制数组到新数组i是新数组长度

扩容代码

**private** **void** ensureCapacity() {

**if** (size == arr.length) {

arr = Arrays.*copyOf*(arr, arr.length \* 3 / 2 + 1);

}

}

binarySearch(Array arr, Object o)查找元素o在arr数组内的角标

# 二进制输出

Integer.toBinaryString(i);

# Swich

switch(表达式|byte|short|int|char){

case 常量值1:

语句;

break;

case 常量值2:

语句;

break;

default:

语句;

break;

}

注意: switch (小括号中的表达式)可以处理的数据类型 byte short int char ,enum(jdk5.0) String(jdk7.0)

注意: case 后是常量值 常量值后是冒号(:),不要添加; (分号)

# 转义字符

\b 退格

\t 制表符

\r 回车键

\n 换行

\\

\'

\"

'\u0000'空格符

# 方法，函数。

class test {

public static void main(String[] args) {

System.out.println("9和8中最大值为："+max(9,8));

System.out.println("9和8是否相等："+compare(9,8));

System.out.println("2015年是否为闰年："+leapYear(2015));

}

public static int max(int i,int j){

return (i>j?i:j);

}

public static boolean compare(int i,int j){

return i==j;

}

public static boolean leapYear(int i){

return (i%4==0&&i%100==0||i%400==0);

}

}

# 类的封装

第一步: 类成员变量私有

private 修饰 -> 收回权限,禁止其他类随意的通过对象名.属性的形式访问属性.

第二步: 提供公共的set 方法

用于给对象的属性赋值.

在方法中加入逻辑判断,过滤掉非法数据.

第三步: 提供公共的get 方法

用于获取对象的属性值.

定义一个类时,是否提供get 或者set 方法,是需要根据实际开发来决定的...

# 类对象赋值给类对象

在java中，如引用一个对象：

Student cls1=new Student();

如果此时用如下方式定义cls2并赋值：

Student cls2=cls1;

则cls2只是cls1的一个别名，即cls1和cls2都是指向同一个内存地址，修改其中一个，另一个的值会同

时改变，如果想定义两个不同的引用，则定义cls2时，应为String cls2=new Student();

# 排序

## 冒泡排序: 适用于排序小列表

取出数组的前一个位置的元素和后一个位置的元素,比较大小, 如果前一个元素>后一个元素,就互换位置. ->最大值放在了数组的最后.

public static void bubbleSort(int[] arr){

for(int j=0;j<arr.length-1;j++){// 10个元素,只需要9次就可以排好序.

for(int i=0;i<arr.length-1-j;i++){ //-1 避免角标越界, -j 提高性能,避免重复的比较...

if(arr[i]>arr[i+1]){

int temp;

temp=arr[i];

arr[i]=arr[i+1];

arr[i+1]=temp;

}

}

}

}

## 选择排序: 适用于排序小列表

第一次: 取出数组的第一个位置的元素和后面的所有元素进行逐个的比较. 只要第一个元素>后面的元素,就互换位置,

结果: 第一个位置存储的最小的元素...

第二次: 取出数组的第二个位置的元素和后面的所有元素逐个比较,只要第二个元素>后面的元素,就互换位置

结果: 第二个位置就存储第二小的元素...

...

如果有5个元素,这个过程只需要4次.

public static void selectSort(int[] arr){

for(int j=0;j<arr.length-1;j++){

for(int i=j+1;i<arr.length;i++){

if(arr[j]>arr[i]){

int temp=arr[j];

arr[j]=arr[i];

arr[i]=temp;

}

}

}

}

## 快速排序: 适用于排序大列表

**public** **static** **void** sort(**int**[] arr,**int** minIndexOf,**int** maxIndexOf){

**int** i=minIndexOf; //最小角标

**int** j=maxIndexOf; //最大角标

**int** k=arr[i]; //自定义一个k值，作为中间数，通常用数组开头的元素

//把比k值大的数放在k的右边，把比k值小的数放在k的左边

//k从最左边开始找比k值小的数（用循环）

**while**(j>i){

**while**(j>i&&k<=arr[j]){

j--;

}

//找到这个值后把arr[i]的值和arr[j]的值交换

**if**(j>i&&k>=arr[j]){

**int** temp=arr[j];

arr[j]=arr[i];

arr[i]=temp;

}

//k从最右边开始找比k值小的数（用循环）

**while**(j>i&&k>=arr[i]){

i++;

}

//找到这个值后把arr[i]的值和arr[j]的值交换

**if**(j>i&&k<=arr[i]){

**int** temp=arr[j];

arr[j]=arr[i];

arr[i]=temp;

}

}

//循环跳出来时i=j

//4 2 3 1 |5| 9 6 5 8分开2组数然后用递归算法继续

**if**(i>minIndexOf){

*sort*(arr,minIndexOf,i-1);

}

**if**(j<maxIndexOf){

*sort*(arr,i+1,maxIndexOf);

}

}

# 成员变量与局部变量（封装）

在类中的位置不同

成员变量：直接定义在类中,和方法平级.

局部变量：在方法中或者方法上(参数列表)

在内存中的位置不同

成员变量：在堆内存(Heap)

局部变量：在栈内存(Stack)

生命周期不同

成员变量：随着对象的创建而出现在堆内存，随着对象的消失而消失

局部变量：随着方法的执行而存在在于栈内存，随着方法的消失而消失

初始化值不同

成员变量：有默认初始化值

局部变量：没有默认初始化值，使用之前必须初始化值.

# 静态变量，静态方法，静态代码块,构造代码块

## 静态变量

修饰成员变量.

静态变量.

被static 修饰的成员变量,称之为静态变量.

特点: 可以直接通过类名来访问...

类名.静态变量名;

静态变量可以被所有的对象所共享,在程序中只有1个...

普通成员变量

直接位于类中的,和方法平级的.

随着对象的创建,存在于堆内存中. 随着对象的消失而释放的.

只要创建对象,每一个对象内部都有一个.

普通成员变量依赖于对象的-> 堆内存

成员变量是有默认值的．　局部变量在使用之前必须初始化

静态变量

直接位于类中的,和方法平级的, 被static 修饰了.

随着类的加载,而初始化,存储于共享区.

由于类加载的动作只执行一次, 所以静态变量只有一个...

xxx.class (硬盘)-> 加载 xxx.class(内存) 以后再用... 直接读取内存的...

当对象和对象之间出现了共享数据时,可使用静态变量,实现数据的共享...

## 静态方法

1.修饰成员方法.

被static 修饰的方法,称之为静态方法.

2. 特点:

静态方法可以直接通过类名来调用.

类名.静态方法名();

3. 注意静态方法的使用有局限性.

静态方法只能访问静态变量. 不能访问非静态变量

静态方法只能调用静态方法, 不不能调用非静态方法.

原因:

静态变量　：

随着类的加载而初始化. 存储于共享区.

普通成员变量:

随着对象的创建而初始化,存储于堆内存.

成员变量->对象 ->后出现.

new 操作符创建后才出现.

静态变量 ->类 ->先出现的

只要使用到这个类,即使不长久对象. 就会加载,就会出现了.

静态方法只能访问静态(变量,方法)

4. 如果是在本类中调用本类的静态方法,类名可以省略不写...

## 静态代码块:

直接定义在类中的.

static{

}

特点: 随着类的加载而执行,用于初始化类的.

由于类只加载一次,初始化只会执行一次,静态代码块只会执行一次...

## 构造代码块

：直接在类中定义且没有加static关键字的代码块称为{}构造代码块。构造代码块在创建对象时被调用，每次创建对象都会被调用，并且构造代码块的执行次序优先于类构造函数

# 继承extends

子类可以继承父类非私有的属性，子类可以继承父类非私有的方法。

子类无法继承父类的构造函数.

## super 关键字:

作用 : 可以在子类的构造函数中访问父类的构造函数.

用于初始化父类的属性.-> 给子类使用的.

创建的对象是子类的, 仅仅是希望子类的构造函数执行时,初始化到父类的属性...

作用一: 访问父类的构造函数.

1.在子类构造函数第一行存在默认隐式语句super

super(); ->该语句是编译器自动添加到子类构造函数中的.

问题: 如果父类没有提供默认无参数构造,子类的构造函数第一行的隐式语句 super() 就会报错...

解决问题: 程序员可以在子类构造函数第一行,使用super调用父类其有参数构造函数 -> 编译不会报错了...

2. 自行的在子类构造函数第一行访问父类的构造函数...

注意: 子类构造函数中访问父类的构造函数的super语句,必须在子类构造函数的第一行.

注意: 子类构造函数中, 不能即存在访问父类的构造函数语句(super),不能又存在访问本类构造函数的语句(this)

作用二:

子类中访问父类的成员变量

作用三:

子类中访问父类的成员方法.

在子类中,当子类中存在了和父类相同的函数(函数名,参数列表),子子类调用,执行的是子类的,如果此时需要使用父类的函数.

可以使用 super.函数名();

## 重写(override):

覆盖...

前提: 子类继承了父类, 子类中出现了和父类一样的函数(暂时理解为一模一样)

体现: 创建子类对象,调用该方法,执行的是子类的方法.

父类->顶层,只具备基本的属性和行为

子类->底层,基于父类,要比父类更强, 更多的属性,更丰富的行为...

一旦子类继承父类,父类的方法无法满足子类的需求,子类就可以重写父类的方法...

重写细节:

子类函数名和父类的函数名一致->基本

子类函数的参数列表要和父类的函数参数列表一致->基本...

子类函数的访问修饰符 和父类函数访问修饰符的关系.

重写的细节:

1. 子类无法重写父类的私有方法.

2. 子类无法重写父类的静态方法.

3. 子类无法重写父类的成员变量

-> 重写只涉及非静态的非私有的成员方法...

函数名:

子类重写父类方法,子类函数名必须和父类函数名一样.

参数列表:

子类重写父类方法, 子类函数参数列表必须和父类的一样

返回值类型:

初学者：子类重写父类方法，父类方法有返回值,子类方法的返回值类型和父类的一致... ->绝对ok ...

增强者: 子类重写父类方法, 子类方法的返回值类型可以和父类方法返回值类型不一致

但是子类方法的返回值类型应该是父类方法的返回值类型的子类.

子类方法返回值类型<= 父类方法返回值类型...

类型大: 父类

类型小: 子类

访问修饰符:

子类重写父类方法,子类方法的访问修饰符>=父类方法访问修饰符.

例如: 父类public 子类必须public

父类default 子类 default protected public

# final

作用一:修饰方法.

该方法不能被子类重写,但是子类还有有权利继承,并使用...

作用二:修饰类

该类不能有子类了. 不能被继承

但是该类仍然是可以创建对象的.

Integer St ring

Integer in1=new Integer(10); String s1=new String("jack");

作用三:修饰变量．

该变量变为了常量.

特点: 这种常量,可以通过名字访问,获取到常量值.

最显著,常量值是不能改变的...

总结: final 修饰变量,变量称之为常量, 常量值无法修改...

java 安全编码规范...

常量:

普通常量.

仅仅被final 修饰

静态常量

被static final 修饰.

实际开发时,最的的是被public static final 修饰的.

作用四：修饰方法的形参.

保证了函数的调用者参数的传递,不会被非法篡改.

# 抽象（继承）

1. 被abstract 修饰的类,叫做抽象类.

抽象类特点:

1. 抽象类不能创建对象.

抽象类有可能存在抽象方法,功能是不完整的, 理论上就可以调用这些抽象方法,没有意义

2. 抽象类中方法可以有抽象方法,也可以有非抽象方法.

非抽象方法->子类直接继承, 提高复用性.

抽象方法->子类去重写这些方法,规范子类的行为.

3. 抽象类可以有成员变量

给子类使用的.

4. 抽象类可以有构造函数

给子类使用的, 用于子类对象初始化父类属性,供子类用.

5. 抽象类中的可以有非抽象的成员方法

给子类使用,让子类继承.

6. 抽象类中也可以没有抽象方法.

如果一个类中出现一个抽象方法,那么这个类必须声明为抽象类.

但是抽象类可以没有抽象方法的. ->要注意...

总结: 抽象类和普通的类区别.

抽象不能创建对象

类的修饰多了abstract.

类中允许出现抽象方法.

普通类

可以创建对象.

类中不允许出现抽象方法.

使用抽象类.

抽象类通常作为父类存在的, 会有子类去继承抽象类.

子类继承了抽象类,如果抽象类中有抽象方法

1:子类重写父类中的抽象方法.

2:子类也声明为抽象类.

抽象方法不能有方法体

# 接口（继承）

1. interface创建接口 implements实现接口

特殊的类. 该类不再使用class 修饰,使用interface 修饰...

2. 接口是一个特殊的类:

成员变量:

公共的静态常量.

public static final 修饰的, 即使不写默认也是.

接口中不能有构造函数.

构造函数用于初始化对象的属性,给属性赋值,接收的成员变量都是常量,无法赋值.

接口不能创建对象.

接口中不能有普通成员方法.(非抽象方法)

成员方法:

公共的抽象方法.

总结: 接口内只有2块东西（常量，抽象方法）

接口成员变量,必须是公共静态常量.

接口中的方法,必须是公共抽象方法.

3.使用接口.

一个类实现接口.

这个类.

3.接口的作用?

1. 可以对类进行功能的拓展...

解决java 单继承的缺陷. 让类可以继承一个类,同时还可以实现接口...

核心: 规范类的功能...

# 多态

一个对象具备多种形态.

前提:

子类继承(实现接口)父类.子类重写了父类中的方法(实现类重写了接口中的方法).

体现:

创建子类对象. 并使用父类类型的对象名接收这个对象的地址值. ->父类类型接收了子类对象.

使用该对象名,调用子类重写过的方法, 执行的是子类重写后的方法...

多态作用:

面向对象3大特征之一:

封装

提高对象属性的安全性,过滤非法数据.

继承

描述类和类之间的关系,提高代码复用性.

多态

提高程序的可扩展性(可伸缩)

作用一: 多态可以用在方法的参数列表上.

可以保证该方法接收范围更多类型的数据.

作用二: 多态可以用在方法的返回值类型上.

可以保证该方法返回更多的类型的结果.

作用三: 多态可以用在数组中.

可以让数组存储不同类型的元素...

向上转型：多态使用的是父类类型接收子类对象. 虽然创建的是子类对象,但是使用的父类类型的对象名接收的.

向下转型：从父类转回子类去接受子类的对象

Instanceof

比较运算符. （我理解为属于）

对象名 instanceof 类名.

检测对象是否是该类类型的.

注意:

编译报错

对象和类之间没有关系.

返回false

对象和类之间有关系.

父类对象 instanceof 子类类型.

子类对象A(多态) instanceof 子类类型B .

返回true

本类对象 instanceof 本类类型

子类对象 instanceof 父类类型

实现类对象 instanceof 实现类实现的接口类型.

用好instanceof 可以避免类类型转换时出现ClassCastException 异常.

例子

if(an instanceof Dog){// 判断对象是否是Dog 类的.

Dog d=(Dog)an;

d.protectedHome();

}

# 内部类

定义比较特殊,出现在一个类的内部,或者方法的内部...

成员内部类:

成员内部类,直接位于类的内部, 和成员变量,成员方法平级的.

创建成员内部类对象:

外部类.内部类 对象名=new 外部类().new 内部类();

成员内部类好处:

成员内部类可以直接访问所在外部类的属性和方法. 注意即使是私有修饰的,内部类仍然可以访问...

成员内部类修饰符.

private 修饰:

封装思想了,成员内部类使用private 修饰,对外隐藏了内部类实现,其他类中就无法直接创建内部类对象了.

Private修饰后只能用get 访问，而且被private修饰后不能直接用内部类去接受对象，所以要用到多态，创建一个接口，内部类实现接口，用接口去接受对象。

public class Demo7{

public static void main(String[] args){

Out out=new Out();

InterA in=out.getIn();

in.testA();

}

}

interface InterA{

public abstract void testA();

}

class Out{

private class In implements InterA{

public void testA(){

System.out.println("testA...");

}

}

public In getIn(){

return new In();

}

}

static 修饰

成员内部类使用static 修饰. 创建静态内部类,不需要创建外部类对象了... 弊端: 静态内部类只能访问外部类静态属性,静态方法了

public class Demo8{

public static void main(String[] args){

Out.In in=new Out.In();

in.testIn();

}

}

class Out{

int x;

static int y=3;

static class In{

public void testIn(){

System.out.println("y:"+y);

}

}

}

局部内部类.

位于方法中. 局部内部类如果想要访问所在方法的局部变量,请使用final 修饰.

public class Demo1{

public static void main(String[] args){

Out out=new Out();

InterA in=out.testOut();

in.testA();

}

}

interface InterA{

public abstract void testA();

}

class Out{

public InterA testOut(){

final int x=1;

class In implements InterA{

int y=1;

public void testA(){

System.out.println("x:"+x);

System.out.println("局部内部类");

}

}

return new In();

}

}

匿名内部类

简化书写...

看到的匿名内部类,很像是在new 一个接口,其实是在new 接口的实现类对象. 只不过该对象简化了书写...

public class Demo2{

public static void main(String[] args){

method1(new InterA(){

public void testA(){

System.out.println("testA...");

}

});

}

public static void method1(InterA in){

in.testA();

}

}

interface InterA{

public abstract void testA();

}

# 加载顺序

先加载main方法的类>父类 >父类静态构造代码块 >子类静态构造代码块>父类构造代码块 > 父类构造方法 >子类构造代码块 > 子类构造方法

静态构造代码块>Main

# 数组的扩充

private Object[] elementData=new Object[10];

private void ensureCapacity(){

//System.out.println("检测数组是否满了... ");

//1. 检测数组是否满了.

if(index==this.elementData.length){

System.out.println("数组已满,扩容...");

//2. 数组满了.

//2.1 创建一个新的数组 新数组的长度是旧数组的两倍.

Object[] newElementData=new Object[this.elementData.length\*2];

//2.2 遍历旧数组,取出旧数组每一个元素, 存储新数组中.

for(int i=0;i<index;i++){

newElementData[i]=this.elementData[i];

}

//2.3 使用新数组继续存储元素.

this.elementData=newElementData;

}

}

# 异常处理

java 异常体系:

|---Throwable（超类）

异常体系顶层父类

构造函数:

无参数构造

有参数构造: 可以接收一个字符串,用于定义异常信息的.

方法:

String toString(); 该方法返回一个字符串. 用于描述异常对象.

String getMessage(); 该获取异常对象表示的异常信息.

void printStackTrace();该方法是用于显示异常的栈信息.

|---Error (错误)

代表的是程序中出现的严重的错误, 不是程序员通过代码通过异常处理就可以解决... 通常是不关注...

java 中的错误类,有一个特点: 所有类的类名的后缀都有Error ,除了: ThreadDeath

|---Exception(异常)

这是程序员需要重点处理的.

异常:

如果程序中出现了异常,程序会提前终止...

如果没有进行专业的异常处理的话,程序提前终止了...

异常处理:

方式一: 使用专业的try catch 语句,进行捕获处理...

try{

可能出现异常的语句;

}catch(异常类 对象名){

专业的异常处理代码;

}

使用了异常处理语句后,程序不会再提前终止了...

注意: try 代码块中存放的是可能出现异常的语句.

catch 块中是设计专业的异常处理的语句. 根据实际情况设计.

先try 在catch

当try 语句中的代码出现了异常时,catch 会进行对应的捕获,如果捕获到了就进入异常处理...

例子

try{

System.out.println(i/j);

}catch(ArithmeticException e){

System.out.println("除数为0啦");

//System.out.println(e.getMessage()); // 获取异常信息.

//System.out.println(e.toString());

e.printStackTrace(); // 打印异常的栈信息...

}

throws和throw的区别

throws

用在方法声明后面，跟的是异常类名

可以跟多个异常类名，用逗号隔开

表示抛出异常，由该方法的调用者来处理

throws表示出现异常的一种可能性，并不一定会发生一些异常

throw

用在方法体内，跟的是异常对象名

只能抛出一个异常对象名

表示抛出异常，由方法体内的语句处理

throw则是抛出了异常，执行throw则一定抛出了某种异常

# String类

equals 方法. 比较字符串内容是否相等，== 比较对象内存地址值

查找:

char charAt(int index) 返回指定索引处的 char 值。

int length() 返回字符串的长度, 以字符为单位.

char charAt(int index); 获取字符串指定角标位的字符(就像数组一样,可以通过角标操作字符串)

int indexOf(char ch); 查找字符在字符串中第一次出现的角标. 找到返回1找不到返回-1，以下同理

int indexOf(String s); 查找字符串s 在指定字符串中第一次出现的角标.

int indexOf(char ch,int fromIndex); 可以从字符串的指定位置开始查找字符ch在字符串出现的角标.

int indexOf(String str,int fromIndex); 从字符串的指定位置开始查找str出现的角标.

int lastIndexOf(int ch); 也是存在4种解决方案的.

indexOf 默认是从字符串的 左->右.

lastIndexOf 是从字符串的 右->左.

concat(String st)用于字符串的拼接，

contains(String st)方法 检测字符串是否包含指定内容

isEmpty()检查是否为空””

replace(char c1,char c2) c2替换c1. 将字符串中的指定字符全部替换为新字符

split(String st)切割 ,按照字符串内出现的st就行切割，切割后返回String[]

如果st为正式表达式"."或者"|"需要改为"\\." 、"\\|"

substring(int start,int end) 截取子符串start(包含)开始,end(不包含)结束

toUpperCase()转大写

toLowerCase()转小写

trim() 去除字符串两端的空格

equalsIgnoreCase(String st)忽略大小写比较.

startWith(String st)开头是否为st，

endsWith(String st)结尾是否为

toCharArray() 字符串转字符

format(“%03d”,1) 格式化为001

字符数组转为字符串. 使用字符串构造函数.

char[] arr = { 'a', 'b', 'c', 'd' };

String s2 = new String(arr);

或者

String s2 = String.valueOf(arr);

**char**[] c=str.toCharArray();----常用

/基本数据类型转字符串(非常常见的操作) valueOf .

int i = 100;

方式一 使用字符串拼接 +"" (最为常用的)

String stringValue1 = i + "";

方式二 使用字符串的valueOf 方法..

String stringValue2 = String.valueOf(i);

面试题：

String s1="hello";

String s2="hello";

String s3=**new** String("hello");

System.*out*.println(s1); //hello

System.*out*.println(s3); //hello

System.*out*.println(s1==s2); //true

System.*out*.println(s1==s3); //false

System.*out*.println(s1.equals(s2)); //true

System.*out*.println(s1.equals(s3)); //true

# StirngBuilder类

StringBuilder 类概述:

该类被称之为,字符串缓冲区. java 的字符串拼接(+)底层使用的就是该类对象实现的.

查看源码: 所谓的字符串缓冲区,底层是一个字符数组,默认的StringBuilder 无参数构造,创建的缓冲区大小只有16个字符. 当然在缓冲区满了以后会自动扩容.

注意: 该类注意并不是String 类的子类. 该类的父类是Object .

StringBuilder 类的常用方法.

append(boolean b/ char c/ char[] str/ double d)添加, 一次可以增加一个

insert(int I,String st)插入内容，在角标为i的前面插入st

delete(int start,int end)删除内容，包含start不包含end

deleteCharAt(int i)删除指定角标下的字符

charAt(int i) 查找 指定角标位置的字符.

indexOf(String s) 查找字符串, 查找字符串在字符串缓冲区中出现的位置.返回要查找字符串的首字符在被查找字符串的角标

reverse() 反转字符串

getChars(int a,int b,char[] ch,int start) 取出字符串缓冲区中的部分字符,存储到指定的数组中.被取出数组从a角标开始，b角标结束，放在start角标开始的ch字符串数组中

# StringBuffer

StringBuffer和StringBuilder 异同点?

相同: 全部都表示字符串缓冲区的,这两个类的方法全部相同...

不同:StringBuffer ->jdk1.0 多线程是线程安全的,同步.(多线程,更加安全)

jdk 升级了. 升级后推出了StringBuilder ,java 内部都在使用StringBuilder ...

StringBuilder ->jdk1.5 多线程是不安全,不同步(单线程性能更高,更快)

什么时候使用,String

当你需要表示字符串时.

什么时候使用,StringBuilder

当程序中大量的出现了字符串,为了提升性能,建议使用StringBuilder . 大量的使用+ 进行字符串拼接时.

什么时候使用StringBuffer

出现多线程时. StringBuffer 是同步的,可以解决多线程安全问题.

# 集合

程序的组成是由众多的对象组成的.

在面向对象的程序中会出现大量的对象... 如何在程序有效的存储这些对象? 可以使用数组...

数组有缺陷:

1.定长.

数组长度一旦声明好之后,就不可变了.

2.类型固定

使用多态解决了... 弊端... 使用Object类型...

无法满足面向对象程序的开发了: 专业的对象做专业的事情. java 中是否存在一个专业的对象,这对象可以替代数组,比数组更强(解决 可变长,多类型)容器?

体验集合:

优点一: 可以存储任意多个元素

优点二: 可以存储任意类型的元素

集合并不一个单一的类,不像String类那样,集合是一个体系(继承父类,实现接口),父类,子类,接口,实现类...

为什么集合是一个体系?

顶层,底层...

集合,是用于替代数组存储的专业的容器. 专业的存储对象的容器...

容器: 如何存储,存储时的结构决定了程序的性能...

如果java 只设计一个集合类... 这个类到处使用... 这是可以的. 但是开发时: 有的时候,快速的查找... 有的时候,快速的增,删...

java 就设计了很多个集合类,用于满足开发者的不同使用场景的需求... 这些集合类,底层存储元素的结构不同,导致性能不同... 但是无论是何种性能的集合,都具备基本的核心的行为:

增,删,查,改元素的行为. java 就将这些集合类设计了集合体系... 抽取了接口... 规范了集合体系的行为...

数组 ->ArrayList

链表 ->LinkedList

哈希 ->HashSet

二叉树 ->TreeSet TreeMap

图 ->Map

都是数据结构,java 的集合类基于这些数据结构都有对应的实现类

学集合的三境界:

1.学会怎么用.

创建对象,调用方法,方法的作用...

2.明白集合实现原理

达到,合适的情况下找到合适的集合,保证程序性能是最优的...

3.自行设计集合类.

...

集合顶层接口:

## Collection

该接口是集合体系中的顶层父类.

重点关注接口中的方法:

分类:

增加

boolean add(Object e)

一次可以增加一个元素.

增加成功返回true,失败false(什么时候失败)

boolean addAll(Collection c)

批量增加,将集合c 中的所有元素添加到指定的集合中.

删除

void clear()

清空集合.

boolean remove(Object o)

删除元素. 删除指定的元素.

boolean removeAll(Collection c)

批量删除, 将指定集合中和集合c相同的元素全部删除.

boolean retainAll(Collection c)

批量删除, 将指定集合中和c 集合中不同的全部删除.

修改

查找

boolean contains(Object o)

检测集合是否包含元素.

boolean containsAll(Collection c)

检测指定的集合中是否包含集合c 中的所有元素...

boolean isEmpty()

检测集合是否为空,集合没有一个元素.

int size()

获取集合存储的元素的个数,也叫做集合的长度.

其他

Object[] toArray() 集合转为数组. 集合的元素作为了数组的元素...

可以改方法取出集合中存储每一个元素.

忽略的方法:

int hashCode()

boolean equals(Object o)

Iterator<E> iterator()

创建集合类对象,并测试这些方法,记录方法的作用

## List列表

collection的实现类

List的特有方法

增加

void add(int index, Object o)添加单个元素

boolean addAll(int index,Collect c)批量添加

修改

Object set(int index,Object o)按照角标修改，返回被修改的元素

删除

Object remove(int index) 按照角标删除，返回被删除的元素

查找

Object get(int index) 按照角标来修改，返回需要查看的元素

int indexOf(Object o) 查找元素，返回角标

int lastIndexOf(Object o) 查找元素，返回角标

其他

List subList(int start,int end) 截取，从start包含，end不包含

### ArrayList

|---Collection(接口)

|---List(接口)

Collection的子接口.

代表了一类集合,这一类集合的特点是:

元素在集合中存储是有顺序序,元素在集合是有角标的,元素在集合是可以重复的.

|---ArrayList

ArrayList特点,实现原理,性能.

如何了解一个类的实现原理:

方案一：查看源码

方案二：可以看老师设计的mini版本的

只考虑核心的逻辑.

底层是一个数组,数组元素是类型是Object 类型,保证可以存储任意类型的元素.

特点: 底层是数组实现,内存地址连续的空间,查找时速度很快的...

底层的数组:默认空间为10，扩容的是以1.5倍形式: \*3/2+1

增加元素是:需要元素后移.

数组空间满了,还要创建新数组,拷贝旧元素,元素后移.

删除元素:需要元素前移

查找快，增、删很慢.

总结:ArrayList 性能: 数组实现,内存地址连续,查找快,增删慢.

注意: ArrayList集合中的contains 方法,如何正常工作的?

该方法是用于检测集合中是否包含指定的元素... 如何检测的?

boolean contains(Object e);

将元素e 和集合中存储的每一个元素,进行比较. 使用的是元素的equals 方法. 比较时.

只要equals 方法为true, 视为集合包含这个元素.

在jdk1.0用的是Vector，两者区别为：

ArrayList： 单线程效率高

Vector ： 多线程安全的，所以效率低

### LinkedList

|---Collection

|---List

特点: 元素有序,元素有角标,元素可重复.

|---ArrayList

特点: 数组实现,查找快,增删慢.

|---LinkedList

特点: 双向链表实现,查找慢,增删快.

LinkedList特点:

链接列表实现。 内部依靠节点存储元素的. 当使用LinkedList存储一个元素, 就增加了一个节点对象. 使用节点保存了一个元素. 如果增加新元素,就增加新的节点对象. 节点和节点之间记录地址.

由于节点和节点内存中的地址是不连续的... 导致了LinekdList 性能是查找慢,增删快.

LinkedList特有方法:

由于双向链表,提供众多操作集合头和尾的方法.

addFirst

addLast

getFirst

getLast

removeFirst

removeLast

实现数据结构:

#### 栈

Stack

先进后出. 后进先出.

方法

void push(Object o)堆栈,

Object pop()弹栈

#### 队列

Queue

先进先出.

方法

void offer(Object o)进队

Object poll()出队

## 去集合中的重复的部分

ArrayList list = **new** ArrayList();

list.add("jack");

list.add("rose");

list.add("lucy");

list.add("jack");

System.*out*.println(list);

// 去除ArrayList中重复的元素...

// 2. 创建新的集合...

ArrayList newList = **new** ArrayList();

// 3. 遍历旧集合.

**for** (**int** i = 0; i < list.size(); i++) {

Object e = list.get(i);

**if** (!newList.contains(e)) {// 使用contains 方法,检测新集合是否包含元素. 如果元素是类对象类型，必须在类中重新equals()方法

newList.add(e);

}

}

System.*out*.println(newList);

## 扩容代码

**private** **void** ensureCapacity() {

**if** (size == arr.length) {

arr = Arrays.*copyOf*(arr, arr.length \* 3 / 2 + 1);

}

}

## toString方法重写

**public** String toString() {

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

sb.append("[");

**for** (**int** i = size - 1; i >= 0; i--) {

**if** (i != 0) {

sb.append(arr[i]);

sb.append(",");

} **else** {

sb.append(arr[i]);

}

}

sb.append("]");

**return** sb.toString();

}

## java集合之迭代器:

问题:

有一个Collection集合.集合中并存储了元素. 想要取出集合中的每一个元素?

有一个List 集合, 集合中存储了元素,想要取出中每一个元素?

有一个Set 集合,想要取出集合中的每一个元素.

解决方案一:

toArray方法,集合转数组. 弊端: 不面向对象. 不专业

解决方案二:

get方法,缺陷: 不通用,只有List 集合才可以用.

根据面向对象思想,专业的对象的做专业的事情. java 设计了对象,用于取出集合中的每一个元素... ->该对象被称之为迭代器.

演示迭代器使用:

格式固定的,代码叫简单...

但是不易理解的...

迭代器的实现原理:

LinkedList link = new LinkedList();

link.add("001");

link.add("002");

link.add("003");

//取出集合中每一个元素.

Iterator it=link.Iterator();

while (it.hasNext()) {

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

}

System.out.println(">>>>>>>>>>>>>>>>");

for (Iterator it1 = list.iterator(); it1.hasNext();) {

System.out.println(it1.next());

}

System.out.println(">>>>>>>>>>>>>>>");

// 这些代码以后都不会在写了. 使用增强for 循环,简化了书写.

for (Object e : list) {// jdk5.0 foreach 增强for循环...

System.out.println(e);

}

1. 迭代器是一个类,该类是内部类

这个类存在于每一个集合实现类的内部(ArrayList内部有,LinkeList内部也有,HashSet内部也有).

原因: 设计为集合类的内部类的话,可以方便的访问所在外部类(集合类)的成员(变量,方法)

2. 该类被私有修饰

原因: 封装的思想,提高了内部的安全性,禁止在其他类中直接创建内部类对象.

对外隐藏了内部类的具体实现...

3. 获取私有成员对象.

集合类提供了一个公共的方法. 用于获取私有成员内部类对象的.

注意:为了保证每一个集合类都具备获取迭代器对象的能力.

在Collection 接口中,规范了一个方法. 保证了每一个集合实现类都要重写该方法,保证每一个集合类都具备获取迭代器的功能.

public abstract Iterator iterator();

只要是Collection 集合体系中的实现类,都具备获取迭代器的功能...

4. 如果和规范私有成员内部类的行为?

如何规范迭代器的行为?

java设计了一个接口,Iterator.

public interface Iterator{

public boolean hasNext();

public Object next();

public void remove();

}

该接口并不是集合类实现了,而是存在于集合类的私有的成员内部类实现了... 这个私有的成员内部类通常称之为迭代器类.

迭代器设计模式.

1. 设计接口,规范迭代器基本行为.

hasNext

next

remove

2. 在集合类内部,私有成员内部类实现该接口,重写接口方法.

3. 集合类提供一个公共的方法,用于获取该私有成员内部类对象

|---Iterator

|---ListIterator

List 集合特有迭代器.

3. 案例:

一句话:

"do you like java if you like please good good study java";

## Set集合

|---Collection

|---List

|---Set(接口)

作为集合体系Collection的一个重要分支.

特点: 元素不可重复(集合存储元素时,是不会存储重复元素的), 元素在集合中是无序(不会按照添加的先后顺序存储)

注意: 该接口没有特有的方法, 不必重复测试方法了.

重点: Set 集合的实现类上,不同的实现类,保证元素唯一性上实现方式是不同的...

|---HashSet

|---LinkedHashSet

|---TreeSet

### HashSet集合.

|---Collection

|---List

|---HashSet(实现类)

该类特点: 底层通过哈希表(hashtable),保证元素的唯一性的.

该类是如何确认元素的重复性,该集合类是如何保证元素的唯一性的.

方法:

hashCode equals

HashSet集合在存储元素时,先调用元素的hashCode方法,获取元素的哈希值. 和集合中已经存储的对象的哈希值进行比较.

如果新元素的哈希值不同,HashSet集合,直接把新元素视为不同元素,存储到集合中.

如果新元素和已经存储的元素的哈希值相同, 集合HashSet 会调用元素的的equals 方法,进行比较.

当equals 方法为true: 集合把新元素视为重复元素,无法添加.

当equals 方法为false : 集合把新元素视为不重复元素,把新元素和旧元素(哈希值相同)添加到了一个位置.

注意: 公共的约定....

如果两个对象的equals 为true,保证这两个对象的hashCode 结果应该相同...

如果不遵循该约定,导致使用导hashCode 和equals 的集合,无法正常的工作.

HashSet HashMap LinkedHasetSet ...

String 类的hashCode 和equals .

String 重写了hashCode 方法.

根据字符串每一个字符计算出总结果.int

String 重写了equals 方法.

只要字符串的内容相同,equals 就为true.

"jack" new String("jack");

设计汽车类.

Car 类.

名字,颜色,价格

创建汽车对象.

使用集合HashSet 集合存储.

HashSet 集合 不会存储重复的汽车. 当名字相同,颜色相同,价格一样的车,是相同的...

### TreeSet集合.

|---Collection

|---List

|---Set

|---HashSet(实现类)

哈希表.

hashCode

equals 方法.

|---TreeSet(实现)

二叉树.

TreeSet集合可以给集合中的元素自然排序, 还会去除重复的元素.

1. 使用TreeSet 集合存储元素时,要保证元素进行比较大小. 通常自定义类,不具备比较功能.

如果需要TreeSet存储自定义对象正常工作时, 自定义对象所属的这个类去实现Comparable接口,重写compareTo方法... 在方法内部建立自定义对象的比较规则...

2. 如果特殊的情况:

使用的这个类,不是自行设计的,是别人的...

这个类刚好有没有实现Comparable 接口. 表示该类不具备比较功能的?

该类的对象,还是TreeSet存储么? 解...

修改源码... TreeSet ...

如果你使用的是第三方的类, 你可以自定义类继承它... 子类中实现Comparable 接口,重写compareTo方法...

但是如果第三方类是final 修饰, 该类就无法有子类继承类.

可以直接给集合TreeSet 指定一个比较器...

Comparable是个接口，用于treeSet添加元素时，判断treeSet集合内是否有重复元素，如果类对象作为元素，那么该类需要实现Iterable接口，还有重写CompareTo方法。

Comparator也是个接口，但是它用于实现一个比较器的类，还要重写compare方法。

# 增强for循环

**public** **class** Demo3 {

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

HashSet<String> set = **new** HashSet<String>();

set.add("张三");

set.add("李四");

set.add("王五");

//遍历Set集合需要使用迭代器

/\*Iterator<String> it = set.iterator();

while(it.hasNext()){

String item = it.next();

System.out.println(item);

it.remove();

}

// 使用增强for循环遍历集合的元素

for(String item : set){

System.out.print(item+",");

set.remove(item);

}\*/

**int**[] arr = {19,2,80,10};

/\*//使用普通的for循环遍历

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

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

}

//使用增强for循环遍历数组的元素

for(int item : arr){

System.out.print(item+",");

}

//需求： 在控制台打印5句hello world.

\*/

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

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

}\*/

**int**[] arr2 = **new** **int**[5];

**for**(**int** item :arr){

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

}

}

}

# 枚举

枚举的定义格式：

enum 类名{

}

枚举要注意的事项：

1. 枚举其实也是一个类，只不过是一个特殊的类而已。

2. 枚举值默认的修饰符是： public static final

3. 枚举值的类型实际上就是 当前所属的枚举类的数据类型。

4. 枚举类的构造方法必须使用private修饰。

5. 枚举类是可以定义成员变量与普通的方法的。

6. 枚举值一定要是枚举类的第一个语句。

7. 如果一个枚举类出现了抽象的方法，那么枚举值就必须要实现抽象的方法。

**enum** Gender2{

*MAN*("男"){

@Override

**public** **void** test() {

System.*out*.println("这个是男的test方法....");

}

},*WOMAN*("女"){

@Override

**public** **void** test() {

System.*out*.println("这个是女的test方法....");

}

}; //枚举值

String info; //成员变量

**private** Gender2(String info){

**this**.info = info;

}

**public** **void** printInfo(){

System.*out*.println("print："+info);

}

**public** **abstract** **void** test();

}

**public** **class** Demo2 {

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

Gender2 g = Gender2.*WOMAN*; //获取到了枚举对象

g.printInfo();

g.test();

}

}

# 可变参数

可变参数的格式：

数据类型... 变量名

可变参数要注意的细节：

1. 如果一个函数使用了可变参数的时候，传递参数给可变参数的时候可以不传递也可以传递多个。

2. 可变参数实际上是一个数组对象，该数组对象是由jvm创建。

3. 可变参数必须要位于形式参数的最后一个位置。

4. 一个方法最多只能有一个可变参数，因为可变参数要位于形式参数的最后一个位置上，最后一个位置只有一个而已。

需求： 定义一个函数做加法功能。

**public** **class** Demo5 {

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

*add*(1,2,3); // jvm遇到可变参数的时候会把你的实参数据封装到数组中，然后才传递给可变参数的。

}

**public** **static** **void** add(**int** temp,**int**... arr){

**int** sum = 0;

**for**(**int** item : arr){

sum +=item;

}

System.*out*.println("总和："+sum);

}

# 自动装箱与自动拆箱

java是面向对象的计算机语言，任何事物都可以使用类进行描述，基本类型的数据也是一类事物，

所以sun就使用了类对基本类型的数据也进行了描述。

基本数据类型 --------------------> 基本数据类型的包装类

byte Byte

short Short

int Integer

long Long

float Float

double Double

boolean Boolean

char Character

基本类型数据出现了对应的包装类的好处： 可以让基本类型的数据也可以使用上了方法。

**public** **class** Demo6 {

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

String str = "10";

//功能一： 可以把字符串类型的数据转换成基本类型的数据

**int** num = Integer.*parseInt*(str);

System.*out*.println(num+1);

//功能二： 可以基本类型的数据转换成对应进制字符串形式

**int** num2 = 10;

System.*out*.println(10+""); //十进制的字符串表现现实

System.*out*.println("二进制："+Integer.*toBinaryString*(num2)); //1010

System.*out*.println("八进制："+ Integer.*toOctalString*(num2)); // 12

System.*out*.println("十六进制："+ Integer.*toHexString*(num2)); // a

//集合是可以存储任意对象类型的容器对象。

/\*

\* 自动装箱就是指把基本类型数据转换成引用类型数据，这个过程jvm帮我们完成了。 new Integer（）

\* ArrayList list = new ArrayList();

list.add(new Integer(1));

list.add(new Integer(2));

list.add(new Integer(3));

int num3 = 10 ; // 基本类型数据

Integer i = num3; //引用类型数据

\*/

/\*

自动拆箱： 把引用类型数据转换成基本类型的数据。---- jvm帮我们调用了.intValue()这个方法把引用类型数据转换成了基本类型数据。

Integer i = new Integer(10); //引用类型的数据

int num4 = i; // 基本类型的数据

\*/

/\*

Integer在一开始的时候就已经把-128~127这个范围的数据已经存储到内部维护的一个数组中，如果以后获取的数据在该为范围之内，

那么直接冲数组中返回即可，不会创建一个新的对象， 这样子做的目的是为了节省内存空间。---------->享元模式。

\*/

Integer num3 = 128;

Integer num4 = 128;

System.*out*.println("是同一个对象吗？"+ (num3==num4)); //引用类型数据使用==比较的时候比较的是内存地址值。

}

}

# Map集合

特点:

基于键值对的.

该集合可以一次存储2个对象, 可以保存这两个对象的关系.

在Map集合中这个两个对象,分别被称之为键和值.

一个键对应一个值. 在Map 集合中,键是唯一的(键不可重复).值可以重复.

你的学校:

学号->学生.

学号,在学校中是唯一的.

一个学号一个学生...

搜索学生的时候,可以通过学号,快速的找到对应的学生...

特殊的需求.

使用java 程序在存储对象. 一次存储2个对象... 并保存这两个对象的关系... 一一对应的关系...

学号<->学生

老公<->老婆

身份证<->人

请问使用集合来存储对象之间的映射关系?

Collection add 一次只能存储一个对象... 有一类集合一次存储2个对象,并且2个对象的一一对应的关系...

单列集合. 一次存储一个对象作为集合的一个元素.

双列集合, 一次存储2个对象...

键值对... key<->value

分类

增

V put(K key, V value);

一次存储两个对象,第一个参数称之为键(key), 第二个参数称之为值(value)

void putAll(Map m)

批量增加键值对. 将m 集合中的所有键值对,添加到指定的map 集合中.

删

V remove(Object key)

根据键删除Map集合中对应的键值对.并且返回键对应的值.

void clear()

清空集合, 清空键值对.

查

boolean containsKey(Object key)

检测map 集合是否包含指定的key

boolean containsValue(Object value)

检测map 集合是否包含指定的值.

V get(Object key)

获取map 集合中的键对应的值.

传入键,获取键对应的值. 如果键不存在,返回null .

boolean isEmpty()

检测集合是否为空. map集合中没有一个键值映射

int size()

获取集合中键值映射的个数的.

改.

V put(K key, V value);

当: key在map 集合中已经存在了. 重复的存储键值对. 键先和旧值解除绑定,键再和新值建立关系... 此时put 方法,返回旧值.

其他(取出Map集合的键值对)

Set<Map.Entry<K,V>> entrySet()

取出Map集合中所有的Entry对象,返回一个Set集合.

Entry 对象,提供的有getKey 和getValue 方法. 用于取出一个键和一个值.

Set<K> keySet()

取出Map 集合中所有的键, 返回一个Set集合.

Collection<V> values()

取出Map 集合中所有的值. 返回一个Collection集合.

**Map遍历**

Iterator it=map.entrySet().iterator();

**while**(it.hasNext()){

Map.Entry entry =(Map.Entry) it.next();

Object key = entry.getKey();

Object value = entry.getValue();

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

}

## HashMap

|---Map(接口)

|---HashMap(实现类)

实现原理. 哈希表

该集合类是如果检测键的唯一性的(识别什么样的键是重复的键?)

底层还是哈希表. 存储键时,调用键的hashCode 和equals 方法. 保证键的唯一性...

只要两个键的hashCode 方法一致,并且两个键的equals 比较结果为true. HashMap 就两个键视为相同的键.

键和旧值解决关系,键和新值建立关系.

## TreeMap

实现原理 二叉树.

不但可以保证键的唯一性,还可以给键进行排序.

方式一:

键具备比较性,键实现Comparable 接口,重写了compareTo 方法.

方式二:

给集合TreeMap指定一个比较器对象(通过TreeMap 的构造函数)

创建一个Comparator接口的实现类对象,重写compare 方法,创建实现类对象,通过TreeMap 的构造函数接收该对象.

注意: 无论是方式一还是方式二,都只能给TreeMap集合中的键排序.

TreeMap并不存在直接给值排序的方法...

列子：

练习一:请统计一个字符串中,每一个字符出现的次数.

"abcaaddczzzz"

a:3 b:1 c:2 d:2 z:4

字符<->次数

Map 集合实现, 很好做. 一次两个对象, 字符,次数,键值对,存储到Map 集合中...

练习二: 按照字符出现的次数升序排.

b:1 c:2 d:2 a:3 z:4

**public** **class** Test {

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

String s="abcaaddczzzz";

TreeMap<Character,Integer> set=**new** TreeMap<Character,Integer>();

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

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

**if**(!set.containsKey(c[i])){

set.put(c[i], 1);

}**else**{

set.put(c[i], set.get(c[i])+1);

}

}

System.*out*.println(set);

Set<Entry<Character,Integer>> entrySet=set.entrySet();

TreeSet<Entry<Character,Integer>> ts=**new** TreeSet<Entry<Character,Integer>>(**new** Comparator<Entry<Character,Integer>>() {

@Override

**public** **int** compare(Entry<Character, Integer> e1,

Entry<Character, Integer> e2) {

**if**(e1.getValue()==e2.getValue()){

**return** e1.getKey()-e2.getKey();

}**else**{

**return** e1.getValue()-e2.getValue();

}

}

});

ts.addAll(entrySet);

System.*out*.println(ts);

}

}

# 正则表达式

正则表达式是用于操作字符串的规则： 而正则表达式一般对字符串的常见的操作有：匹配 、切割、替换、查找

String str="123345";

str.matches("\\d");

## 匹配

. 任何字符（与行结束符可能匹配也可能不匹配）

\d 数字：[0-9]

\D 非数字： [^0-9]

\s 空白字符：[ \t\n\x0B\f\r]

\S 非空白字符：[^\s]

\w 单词字符：[a-zA-Z\_0-9]

\W 非单词字符：[^\w]

注意： 一个预定义字符只能匹配一个字符。

数量词

X? X，一次或一次也没有 非多次

X\* X，零次或多次 非一次

X+ X，一次或多次 有

X{n} X，恰好 n 次

X{n,} X，至少 n 次

X{n,m} X，至少 n 次，但是不超过 m 次

范围词：

[abc] a、b 或 c（简单类）

[^abc] 任何字符，除了 a、b 或 c（否定）

[a-zA-Z] a 到 z 或 A 到 Z，两头的字母包括在内（范围）

注意： 范围词没有配合数量词使用的时候，都只能匹配一个字符。

## 文件地址截取问题

String address=”F:\\aaa\\bbb\\ccc.txt”

if(address.contains("\\")){

String[] strs=s.split("\\\\");

}

# 泛型.

集合如果不使用泛型

增加元素,元素向上转型,变为了Object 类型. 保证集合集合可以存储任意类型的元素.

取出元素,需要将元素向下转型,转为具备的类型. 在转型的时候,容易出现ClassCastException.

将程序运行时出现的错误...

集合如果使用泛型

增加元素时,由于集合类上声明了泛型类型,强制性的要求你,必须添加指定类型的元素了... 保证了集合存储元素的类型.

取出元素时,不必在进行向下转型了,避免了ClassCastException.

1.泛型方法.

修饰符 声明泛型 返回值类型(参数列表){

方法体;

}

注意: 泛型方法中需要先声明泛型. 然后才能使用.

声明泛型时, 一个尖括号 <单词字母|建议大写>

泛型方法的泛型类型什么时候确定了,当调用方法时,确定.

2.泛型类

3.泛型成员

4.泛型继承

5.泛型接口.

6.泛型通配符.

?

? extends 类类型

明确的泛型类型进行限定, 只能是 子类或者本类.

? super 类类型

只能是父类或者本类.

# 集合工具类

Collections

该工具类提供了众多的静态方法,用于操作集合.

很好,很强大...

应用一:

实现List集合的元素的排序. sort

应用二:

取最值max

应用三:

反转集合，reverse

应用四:

反转比较器. reverseOrder

应用五:

解决集合的同步问题.

# I/O File类

该类的出现是对文件系统的中的文件以及文件夹进行对象的封装。可以通过对象的思想来操作文件以及文件夹。

可以用面向对象的处理问题，通过该对象的方法，可以得到文件或文件夹的信息方便了对文件与文件夹的属性信息进行操作。

File file=new File(“f:\\a.txt”);新建File对象

exists() File对象是否存在

isDirectory() File对象是否为目录

isFile() File对象是否为文件

创建：

|  |
| --- |
| createNewFile() 在指定位置创建一个空文件，成功就返回true，如果已存在就不创建然后返回false  mkdir() 在指定位置创建目录，这只会创建最后一级目录，如果上级目录不存在就抛异常。  mkdirs() 在指定位置创建目录，这会创建路径中所有不存在的目录。  renameTo(File dest) 重命名文件或文件夹，参数为新的File对象，也可以操作非空的文件夹，文件不同时相当于文件的剪切。移动/重命名成功则返回true，失败则返回false。 |

删除：

|  |
| --- |
| delete() 删除文件或一个空文件夹，如果是文件夹且不为空，则不能删除，成功返回true，失败返回false。  deleteOnExit() 在虚拟机终止时，请求删除此抽象路径名表示的文件或目录，保证程序异常时创建的临时文件也可以被删除 |

判断：

|  |
| --- |
| exists() 文件或文件夹是否存在。  isFile() 是否是一个文件，如果不存在，则始终为false。  isDirectory() 是否是一个目录，如果不存在，则始终为false。  isHidden() 是否是一个隐藏的文件或是否是隐藏的目录。  isAbsolute() 测试此抽象路径名是否为绝对路径名。 |

获取：

|  |
| --- |
| getName() 获取文件或文件夹的名称，不包含上级路径。  getPath() 将此抽象路径名转换为一个路径名字符串。  getAbsolutePath() 获取文件的绝对路径，与文件是否存在没关系  length() 获取文件的大小（字节数），如果文件不存在则返回0L，如果是文件夹也返回0L。  getParent() 返回此抽象路径名父目录的路径名字符串；如果此路径名没有指定父目录，则返回null。  lastModified() 获取最后一次被修改的时间。  文件夹相关：  staic File[] listRoots() 列出所有的根目录（Window中就是所有系统的盘符）  list() 返回目录下的文件或者目录名，包含隐藏文件。对于文件这样操作会返回null。返回String[]然后用for输出  list(FilenameFilter filter) 返回指定当前目录中符合过滤条件的子文件或子目录。对于文件这样操作会返回null。  listFiles() 返回目录下的文件或者目录对象（File类实例），包含隐藏文件。对于文件这样操作会返回null。返回File[]然后用for输出  listFiles(FilenameFilter filter) 返回指定当前目录中符合过滤条件的子文件或子目录。对于文件这样操作会返回null。 |

# 路径分隔符

1. windows系统下 默认的是 \(反斜线),也兼容 / (正斜线)

2. linux 系 统 使用的 /(正斜线)

保证File可以跨平台. 可以用File类中提供的一个( public static final String separator)

File.separator替换斜线

# 编码解码

URLEncoder.encode(String s,”utf-8”)编码

URLDecoder.decode(String s,”utf-8”)解码

//系统默认码表

String System.getProperty("file.encoding")

//用gbk进行编码

**byte**[] bytes2 = “中国”.getBytes("gbk");

System.*out*.println(Arrays.*toString*(bytes2));

//用gbk进行解码

System.out.println(new String(bytes2, "gbk"));

// FileReader用的是系统的码表解码，不能去解utf-8格式的文本文件

// fis.available() 适用于文本文件(小文件),不适用大文件...用于缓冲

**byte**[] buffer = **new** **byte**[fis.available()];

# IO流.

字节输入流

|---InputStream -> java.io包中.

核心方法:

int read(); 一次读取一个字节,正常: 返回读取到的字节,特殊: 返回-1.

int read(byte[] b); 连续的读取多个字节,使用字节数组缓冲, 正常:返回字节数组存储的字节数,特殊: 返回-1

close(); 关闭流释放资源.

|---FileInputStream

用于操作文件的字节输入流

|---BufferedInputStream

自带字节数组作为缓冲区的字节输入流,默认的缓冲区8192byte->8k

|---SequenceInputStream

顺序流, 合并流...

按照顺序(串联)先后读取字节输入流. 合并多个字节输入流. 通常用于,需要将多个文件中的数据合并到一个文件中时,考虑该流.

read

|---ObjectInputStream

反序列化流.

读取文件,获取数据,恢复一个对象.

注意: 不能直接关联文件的,需要关联普通的字节输入流的.

核心方法: Object readObject();

字节输出流

|---OutputStream

核心方法:

void write(int b); 一次写出一个字节. 无论传递的是什么值(int byte short char) 只写出最低8位.

写出...

void write(byte[] b); 一次写出一个字节数组.

void write(byte[] b,int start,int len); 将字节数组部分字节写出.

close();关闭流.

|---FileOutputStream

用于操作文件的字节输出流.

|---BufferedOutputStream

自带字节数组作为缓冲区的字节输出流

|---ObjectOutputStream

序列化流, 将对象存储到文件中(本质,将对象的属性存储到文件中,二进制格式)

核心方法: writeObject(Object obj);

注意: 要想成功序列化一个对象,必须对象实现接口(Serializable)

注意: 类实现了Serializable ,该类有出现黄色警告线. 怎么处理?

注意:序列化流在序列化一个对象时,会将该对象的所有的属性都序列化,但是如果有的属性无法序列化? 那么该对象就无法序列化了. ->抛出

transient关键字

字节流:

1.读取文件.

1.普通

2.高性能.

2.写出数据到文件.

3.拷贝文件.

高性能拷贝.

字符输入流

|---Reader

核心方法:

int read(); 一次读取一个字符.

底层字节流,连续的读取多个字节.(根据码表) 将字节,根据码表解码为字符.

正常:返回读取的字符

特殊: 读取到文件末尾返回-1

int read(char[] ch);

使用字符数组作为缓冲区,一次读取多个字符.

正常: 返回缓冲区中存储的字符数.

特殊: 读取到文件末尾返回-1.

close();

|---FileReader

用于操作字符文件的一个便捷类. 使用的是系统默认码表...

FileReader ->windows 简体中文下: gbk .

->linux 系统下 utf-8 .

|---BufferedReader

字符输入流的缓冲流

核心方法:

readLine(); 一次读取一行,读取到文件末尾返回null .

注意: 虽然正常情况下返回读取的一行,但是不包含(行的结束符 \r\n

字符输出流

|---Writer

核心方法:

wirte(int c); 一次写出一个字符. int->char(强转)

如何写出字符? 'a'-> 文件97->1100001

字符输出流在写出字符时,根据码表,将字符编码为字节 ,将字节写到文件中. 符号->数字

write(char[] ch); 一次写出一个字符数组.

write(String[] s); 一次写出一个字符串.

flush(); 刷新缓冲区.如果有好的习惯,关闭了close flush 就可以省略.

close();

|---FileWriter

用于操作字符文件的便捷类.

FileWriter ->windows 简体中文下: gbk .

->linux 系统下 utf-8 .

|---BufferedWriter

核心方法:

newLine(); 新建换行符.

应用:

一:读取文本文件.

普通读取,高效读取.

一次一行的.

二: 写出数据到文本文件

一次写出一个字符,一次写出多个字符,一次写出一行字符串.

写出一个换行符.

# 集合Properties

该集合,用于操作配置文件.

配置文件:

配置名=配置值

配置名 配置值

配置名:配置值

该集合可以和IO流进行关联.

load;

load(InputStream in)

集合Properties 可以通过load 方法, 获取配置文件中的配置信息,并将配置信息,以键值对的形式存储到了集合中...

store

store(OutputStream out);

集合Properties store 方法,将集合中的键值对数据,写到配置文件中,写成配置名=配置值的形式...

4.字节流操作中文的问题(解决)

码表 人可以阅读的...

# 转换流.

|---Reader

|---InputStreamReader

字节流通向字符流的桥梁... 该流的构造函数接收一个字节流,该流是一个字符流. 底层使用字节流的读取方法(逐个字节的读取),返回的确实一个字符...

本质上是一个字符流.

read() 一次读取一个字符.

构造函数:

InputStreamReader(InputStream in, 码表)

关联字节输入流,在关联码表.

底层就会使用字节输入流读取多个字节,再按照指定的码表,将字节自动解码为字符.

通用和便捷,但是一旦读取使用系统默认码表的文件.

windows:

new InputStreamReader(new FileInputStream(文件路径),"gbk");

linux:

new InputStreamReader(new FileInputStream(文件路径),"utf-8");

为了更加方便的读取使用系统默认码表的文本文件. java 设计了FileReader

|---FileReader

读取文本文件的便捷类. 当文件使用的码表是系统默认码表时. 使用该流就特别的简单,快捷...

new FileReader(文件);

等同于:

windows 下:

new InputStreamReader(new FileInputStream(文件路径),"gbk");

linx下:

new InputStreamReader(new FileInputStream(文件路径),"utf-8");

|---Writer

|---OutputStreamWriter

本质上是一个字符流.

write(int c); 一次写一个字符.

write(String s); 一次写出一个字符串.

优点: 使用不同码表的文件中写出信息...

构造函数:

OutputStreamWriter(OutputStream out,码表);

关联了字节输出流, 将写出的字符(字符串)按照指定的码表,编码为字节. 再使用字节输出流将字节写出. -> 自动的将字符编码为字节.

|---FileWriter

写出字符信息的便捷类, 当文件使用的码表是系统默认码表时. 使用该流写出文本信息,非常的简单.

new FileWriter(文件);

等同于:

windows 下:

new OutStreamWriter(new FileOutputStream(文件路径),"gbk");

linx下:

new OutStreamWriter(new FileOutputStream(文件路径),"utf-8");

FileWwriter: 适用于使用系统默认码表的文本文件.

OutputStreamWriter : 使用与使用非系统码表的文件...

思考题:

1.读取文本文件,一次读取一行的操作.

BufferedReader

readLine.

控制台录入用户的输入,一次录入一行

new Scanner();

# 标准的输入输出和错误流.

系统提供的流,这些流不是程序员,直接创建出的...

注意: 用户创建的普通流,需要关联路径(文件), 这些标准的流,关联的不是普通的文件.

1. 标准输入.

System.in

键盘,

2. 标准输出

System.out

显示器

System.out. 获取的是一个字节打印流.

注意: 该字节打印流关联的不再是普通文件,而是控制台.

3. 标准错误

System.err

显示器

# 打印流

字节打印流.

|---OutputStream

|---PrintStream

核心方法:

write(int b); 一次写出一个字节.

需要手工刷新

print(); 重载版本的.

可以写出各种不同数据类型的数值.

自动刷新.

println(); 重载版本的.

不但可以写出各种不同数据类型的数值.

还会自动换行.

使用场景一:

System.out 获取的标准输入流.

关联的流向不是普通文件,而是控制台.

使用场景二:

可以创建普通字节打印流,关联普通文本文件,当做普通的流用也是可以的.

# RandomAccessFile

随机访问流.

唯一的一个可以读和写的流.

同时具备了.

字节输入流和字节输出流的功能.

既可以写出数据.

也可以读取数据.

通过seek 设计内部的指针. 可以从指定的位置开始读,可以从指定的位置开始写...

不向想普通的流,写出数据要么写在文件的开头,要么文件的末尾...

案例: 并发的

多用户版的文件上传程序.

客户端

服务端.

上网. 网站... 并发支持n 个人同时访问... 技术支持...

普通读取

System.out.print((char) fis.read());

高效读取.

int data;

while ((data = fis.read()) != -1) {

System.out.print((char) data);

}

普通写入

fos.write(20013);// 20013 最低8位.覆盖原来内容

FileOutputStream fos = new FileOutputStream(file, true);

fos.write("hello,kitty".getBytes()); 不覆盖，在后面接着写

高效写入

byte[] bytes = "hello,world".getBytes();

fos.write(bytes);

fos.write(bytes, 0, 3); // 高级方法 write(byte[] b,int start,int len);

高效拷贝

private static void copyFile(File srcFile, File destFile)

throws IOException {

// 3. 创建字节输入流.

FileInputStream fis = new FileInputStream(srcFile);

// 4.创建字节输出流.

FileOutputStream fos = new FileOutputStream(destFile);

// 5. 定义字节数组作为缓冲区.

byte[] buffer = new byte[1024];

int len;

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

fos.write(buffer, 0, len);

}

// 6.关闭流.

fis.close();

fos.close();

}

# 多线程

Thread 类.

该类就是表示线程的类.

java 中创建线程的两种方式.

方式一:

1.定义子类继承Thread 类.

2.子类重写Thread 类的run 方法.

3.创建子类对象.

4.调用start 方法, 可以开启线程.

方式二:

# 文件切割

FileInputStream fis = **new** FileInputStream("d:\\a.mp3");

// 3. 定义字节数组作为缓冲区.

**byte**[] buffer = **new** **byte**[1024 \* 1024];// 1m

**int** len;

**for** (**int** i = 1; i <= 4; i++) {

len = fis.read(buffer);

FileOutputStream fos = **new** FileOutputStream("d:\\part" + i + ".mp3");

fos.write(buffer, 0, len);

fos.close();

}

fis.close();//

# 合并文件

FileInputStream fis1 = **new** FileInputStream("d:\\part1.mp3");

FileInputStream fis2 = **new** FileInputStream("d:\\part2.mp3");

FileInputStream fis3 = **new** FileInputStream("d:\\part3.mp3");

FileInputStream fis4 = **new** FileInputStream("d:\\part4.mp3");

// 创建集合,存储4个字节输入流.

ArrayList<InputStream> list = **new** ArrayList<InputStream>();

list.add(fis1);

list.add(fis2);

list.add(fis3);

list.add(fis4);

// 获取集合迭代器.

**final** Iterator<InputStream> it = list.iterator();

// 创建1个字节输出流.

FileOutputStream fos = **new** FileOutputStream("d:\\merge.mp3");

// 创建SequenceInputStream 流()

SequenceInputStream seq = **new** SequenceInputStream(**new** Enumeration<InputStream>() {

@Override

**public** **boolean** hasMoreElements() {

**return** it.hasNext();

}

@Override

**public** InputStream nextElement() {

**return** it.next();

}

});

// 一定要按照顺序先后读取.

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

**int** len;

// 使用SequenceInputStream 读取每一个文件.

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

fos.write(buffer, 0, len);

}

// 关闭流.

seq.close();

fos.close();

# 对象的序列化.

## 序列化:

将对象存储到文件中.

ObjectOutputStream 流

方法 writeObject .

序列化的本质: 好像是将对象存储到文件中... 仅仅是将对象的属性值存储到文件中...

## 反序列化:

根据已有的序列化文件.文件->恢复出对象.

ObjectInputStream 流

方法 readObject ...

反序列化本质: 好像读取文件,恢复 了对象,好像是同一个对象... 其实,读取文件获取文件中存储的属性值,根据值创建了一个新的对象. 新的对象和原有的对象属性值是相同的...

注意: 如果想要成功的序列化一个对象,保证这对象实现 Serializable,该接口是一个标示接口.

注意: 序列化流在序列化一个对象时,会将该对象的所有的属性都序列化,但是如果有的属性无法序列化? 那么该对象就无法序列化了. ->抛出

transient关键字

用于序列化时,用于忽略成员的...

ArrayList<String> list = **new** ArrayList<String>();

list.add("jack");

list.add("rose");

list.add("lucy");

*writeList*(list);

// 反序列化 读取文件恢复对象.

*readList*();

}

**private** **static** **void** readList() **throws** Exception {

ObjectInputStream ois = **new** ObjectInputStream(**new** FileInputStream("d:\\list.txt"));

// 2. 恢复对象. 创建了一个新的对象.

ArrayList list1 = (ArrayList) ois.readObject();

// 3. 关闭流.

ois.close();

System.*out*.println(list1);

}

**private** **static** **void** writeList(ArrayList<String> list) **throws** IOException {

// 1. 序列化流,可以将ArrayList类的对象序列化到文件中. -> ArrayList 实现了Serializable

ObjectOutputStream oos = **new** ObjectOutputStream(**new** FileOutputStream("d:\\list.txt"));

// 2. 序列化集合 序列化流是仅仅将list 对象存储到了文件中,还是也考虑了list中存储的元素呢?

oos.writeObject(list);

oos.close();

# 配置文件

Properties 类

该类是Map 集合. 该类可以和IO流进行关联,用于操作配置文件.

由于配置文件中的信息都是字符串,该集合的不用设置键值对. 都是String .

核心方法:

load(InputStream in) -> 读取配置文件,获取配置信息. 硬盘(配置文件)->内存(集合)

load 方法,调用字节输入流,读取配置文件,获取配置信息. 配置信息键值对的形式存储到Properties集合中

store(OutputStream string) ->将集合键值对写到文件中 内存(集合)->硬盘(配置文件)

调用字节输出流,把集合中的键值对写到配置文件中. 一个键值对占用一行.

getProperty(String key);

根据键获取对应的值.

根据配置名称,获取对应配置值.

setProperty(String key,String value);

修改键值对.

新值覆盖旧值.

|---Map

|---Hashtable

该类基于哈希表实现.

jdk1.0出现. 线程同步的,该集合不允许存储null 键,null 值

|---Properties

是一个Map 集合.

该集合可以和IO流进行关联. 可以用于专业的操作配置文件.

|---HashMap

基于哈希表实现... 保障键的唯一: hashCode equals 方法.

jdk1.2出现,线程不同步. 允许存储null 键,null 值.

练习题:

编写小程序,统计软件运行的次数.

当软件运行次数超出了3次,提示费.

实际开发:

设计软件. 1k 个类.. 1w 个类.

10类,需要获取到配置文件中的数据...

10用一次,写一次... load ... 性能... 下降...

类A, load

读取硬盘. I/O

类B, load

读取硬盘 I/O

设计专业的类,只读取一次... 以后在需要用到配置信息,找这个类要...

System.*out*.println("软件启动了...");

// 1 如果软件是第一次运行. 必要的初始化工作.

File file = **new** File("d:\\count.txt");

**if** (!file.exists()) {

FileOutputStream fos = **new** FileOutputStream(file);

fos.write("count=1".getBytes());

fos.close();

}

// 创建集合Properties .

Properties pro = **new** Properties();

// 创建字节输入流,关联配置文件

FileInputStream fis = **new** FileInputStream(file);

// 集合和流进行关联,集合通过流获取键值对的信息.

pro.load(fis);

// 获取指定的配置信息.

String strCount = pro.getProperty("count");

**int** count = Integer.*parseInt*(strCount);

System.*out*.println("第:" + count + "运行程序...");

// 判断次数.是否能够继续免费运行.

**if** (count > 3) {

System.*out*.println("你已经超出免费使用的次数,请续费");

System.*exit*(0);

}

// 如果没有超出.

System.*out*.println("软件运行中...");

// 更新配置文件...

count++;

pro.setProperty("count", count + "");

FileOutputStream fos = **new** FileOutputStream(file);

pro.store(fos, "ok");

fis.close();

fos.close();

# 多线程

程序运行时,程序至少为存在一个以上的线程.

如果程序内部只有一个线程,就是典型的单线程程序

如果程序内部存在多个线程,就是所谓多线程程序,这个程序是并发的,这个同时执行多个任务...

多线程意义?

提高了系统资源的利用率(cpu利用率),提高用户体验...

提高程序运行的速度嘛? 不能. 提升硬件的性能. cpu.

线程是开的越多越好么? 也不是的. cpu 需要在多个线程之间进行切换,切换需要消耗...

1cpu,100线程 如果线程之间没有任何阻塞,,,

多线程下的程序: 其中一个线程进入了阻塞, cpu 不会闲置的. 切换到了其他线程...

提升用户体验, 并发的访问. cpu 在多个线程快速的切换. 跟人感觉100个线程在同时执行...

2.java中的多线程.

1. java 支持多线程. java提供类Thread,表示线程...

java创建线程的方式一:

1.定义子类继承Thread类.

2.子类重写父类(Thread)的run方法.

3.创建子类对象.

4.子类对象调用start 方法(继承自Thread)

java创建线程的方式二:

1.定义类实现Runnable 接口.

2.重写接口中的run 方法.

3.创建实现类对象.

4.创建Thread 类对象. 并将实现类对象通过Thread 类的构造函数传参...

5.调用Thread 对象的start 方法.

推荐使用方式二,实现Runnable接口的方式...

好处一: 更加符合面向对象编程的思想... 专业对象做专业事情.

Runnable 接口的实现类中,定义的是线程需要执行的任务.

Thread 类的对象,执行任务的线程...

将任务封装为了对象,

将线程封装成了对象, 任务和执行任务的线程

可以更加方便的开启多线程:

如果Runnable 接口实现类对象,只交给一个线程执行, ok .

如果Runnable 接口实现类对象,交给多个线程同时执行.

Runnable 接口的实现类对象,工作.

Thread 类对象,理解为工人. 工作可以交给一个人做,也可以交给多个人同时做...

好处二: java 只支持单继承,如果使用继承方式创建线程, 就必须定义类继承Thread 类, 导致该类无法继承其他类...

如果使用实现的方式,创建线程的话,保留了一个继承,该类还可以继承其他类...

语法层面的一个理解...

可以将聊天,视频,语言,交给3个线程分别执行. 并同时开启3个线程,多个线程同时执行. 实现所谓并发...

## 多线程安全问题:

1.前提(充分).

多线程.

2.提现(必要)

多个线程修改了同一个共享数据. 很有可能,就出现了线程安全问题...

使用同步解决多线程安全问题.

核心程序... 涉及到多线程修改共享数据,就需要同步.

有些程序,没有涉及到多线程修改共享数据,此时为了提供性能,可以不同步...

同步方案一:

同步代码块.

synchronized(锁对象){

需要同步的代码;

}

锁对象: 可以是任意类型的对象,必须保证锁的唯一性...

开发中,会使用到this 作为同步代码块的锁对象, this 不是随意使用的,使用this 的前提,保证这个this 只有1个...

同步方案二:

同步方法:

修饰符 synchronized 返回值类型 函数名(参数列表){

方法体;

}

如果方法非静态的方法,默认的锁对象是this.

理论上同步方法可以实现的,同步代码块也可以实现. 同步代码块要比同步方法更加的灵活...

同步方案三:

使用jdk5.0出现的新的同步解决方案.

Lock

lock();

unlock();

结束线程:

1. 使用方法stop .

注意: 过时了.

2. 使用布尔标记控制线程.

想要结束线程时,改变标记.即可...

线程优先级.

如果没有修改线程的优先级,默认是5.

setPriority()

设置线程优先级 1~10

getPriority()

获取线程优先级

join()

设置线程插入，变最优先执行

将线程设置为后台线程

setDaemon(true)

Lock lock = new ReentrantLock();

**try** {

lock.lock();

要上锁的代码块

} **finally** {

// }

// 释放锁. 保证锁一定释放...用try

lock.unlock();

}

多线程使用同步以后,就变为单线程程序了, 多线程有什么意义?

软件: 并不是真个软件都会出现线程安全问题,只是部分核心内容需要同步,其他部门不需要同步的...

同步:

注意: 静态方法内部的同步代码块,不能使用this.

静态同步方法,默认的锁对象是 方法所在类的 类名.class(类的字节码对象) 是唯一的.

## 设计模式:

1. 一些代码用来决绝一类问题,比较经典...

遇到一类问题,总结,使用一些代码适合解决这类问题.

20年... 代码进行了总结... 设计模式了...

棋谱,拳谱

单例设计模式.

作用:可以保证对象的唯一性. 在程序内,对象只有1个...

1.构造私有

2.本类中维护一个本类对象.

3.提供公共静态方法获取该本类对象.

模式一:

懒汉式 ->延时加载. 调用方法时,才创建对象.

模式二:

饿汉式 ->直接创建对象...

其中的一个模式,在多线程下,会出现线程安全问题...

// 饿汉式.

**class** Single {

**private** **final** **static** Single *sin* = **new** Single();

**private** Single() {

}

**public** **static** Single getInstance() {

**return** *sin*;

}

}

// 懒汉式.(延时加载) ->多线程下会出现线程安全问题. T1,T2,T3,T4.

**class** SingleA {

**private** **static** SingleA *sin*;

**private** SingleA() {

}

**public** **static** SingleA getInstance() {

**if** (*sin* == **null**) {// 高效的判断.

**synchronized** (SingleB.**class**) {

**if** (*sin* == **null**) { // if 判断,保证对象的唯一性...

*sin* = **new** SingleB();

}

}

}// 只要sin 不会空,有线程已经创建了一个单例对象,直接返回是sin ,即可...

**return** *sin*;

}

}

## 死锁

哲学家就餐的问题.

## 数据迁移.

数据库.

MySql

Oracle

# 守护线程

守护线程是是运行在后台的一种特殊进程，它独立于控制终端并且周期性地执行某种任务或等待处理某些发生的事件，作用是为其他线程的运行提供服务。

设置守护线程要在线程对象创建之前调用线程对象的setDaemon(boolean b)方法。true为Daemon模式，false为User模式

# 网络编程

1.ip地址.

互联网协议(公共的,非私用的). 可以用来唯一的标识互联网上的计算机

192.168.10.1(只能在局域网内部使用的) 如果真正的接入互联网,需要真正的读公共的ip地址.

IPv4. 4个字节, 32bit 组成,无符号的. 理论上,可以有2(32)个ip地址... 40多亿...

ip 地址使用二进制表示,不便于记忆.点分发,每8位二进制转为一个十进制.

192.168.10.1

IPv6 128个bit,表示一个ip地址

通过ip地址,通过网络找到指定那台计算机. 就可以进行通信了...

2.端口

0~65535之间.

0~1024 不建议使用.

建议使用1024以后的端口.

端口一旦被其他程序占用,我们的程序就无法再使用了...

3.域名

baidu.com

浏览器地址栏 输入 http://ip地址:端口号; 客户网络中指定的服务器...

ping 域名|ip 地址. 测试网络.

互联网的协议之UDP

（User datagram protocol） (用户数据报协议)

特点: 1.数据打包,一个数据包最大64k.

2.发送数据, 不会确认接收方是否在线... 直接发送,如果接收方不再线? ,数据丢失

3.该协议是不可靠的,速度快...

不面向连接,不可靠,速度快.

应用场景: 屏幕广播,视频会议, 视频...

凌波: udp ...

udp 不能保证数据的可靠性,不适合进行文件的的传输... (文本,图片,.... )

互联网的协议之TCP

（Transmission control protocol ）（传输控制协议）

特点: 1.使用该协议,进行数据通信时,通过3次握手,建立连接通道.

2.发送数据前,先确认接收方是否在线.

3.该协议是可靠的, 速度稍慢...

面向连接,可靠,速度稍低.

2.java 中的网络编程.

io流,只能实现单机数据传输... 现在实现网络的数据传输. 通过网络,实现计算机和计算机之间的数据的传输...

1)InetAddress 类

ip地址在java中如何表示.

"192.168.10.1" ->面向对象...

java 中,通过类InetAddress 表示ip 地址.

2) java 中支持UDP协议的类.

java 支持UDP 协议.

类:

DatagramPacket

表示数据报包

构造函数:

DatagramPacket(byte[] buf, int length)

构造 DatagramPacket，用来接收长度为 length 的数据包。-> 该构造函数,可以创建数据报包对象,该对象是作用接收数据

DatagramPacket(byte[] buf, int length, InetAddress address, int port)

构造数据报包，用来将长度为 length 的包发送到指定主机上的指定端口号。 ->该构造函数,可以创建数据报包对象,该对象用于 发送数据的.

DatagramSocket

用于发送和接收数据报包.

发送数据报包

send(DatagramPacket p);

接收数据报包

receive(DatagramPacket p);

3. java中支持tcp 相关的类.

Socket 是 TCP 客户端 API，通常用于连接远程主机。

ServerSocket 是 TCP 服务器 API，通常接受源于客户端套接字的连接。

java tcp 基于 c/s模式的.

C->Client 客户端

S->Server 服务器...

网络应用程序典型: 游戏... cf 网络...

-> B/S

B-> 浏览器

S->服务器...

只要有浏览器,有网络,就可以访问b/s模式的应用...

# java包机制.

1. 是避免类名冲突,导致.class 文件保存的问题(覆盖)

可以使用包名. 添加在源码的第一行...

package 公司域名反写.类的用途;

2. 编译带包名的源码

javac -d 路径名 xxx.java

-d 生成文件夹

路径名 生成文件夹存储的位置,当前路径.

路径名可以是当前路径,也可以是指定的路径.

注意: 在当前路径下想要运行在其他路径中的.class程序时,记得配置classpath ...

3. 运行带包名的class 字节码文件.

不能切换到.class 字节码所在的目录中直接运行了.

java 包名.类名

注意: 一定不要切到

4. 打.jar 包.

发布java 程序时, 不会直接发布 文件夹. 将文件夹打包为.jar 文件.

jar -cvf 自定义包名.jar 文件夹名称(包含有.class的文件夹->通常是由包名生成的)

5. 运行.jar 包.

.jar 必须要修改一个文件,才能保证jar 的运行.

使用 winrar 工具,打开xxx.jar 包,找到META-INT 文件夹.

修改MANIFEST.MF 文件(记事本即可), 添加主类:

Main-class: 包名.类名

该类,必须要有主函数,作为整个程序入口的一个类...

保存即可.

注意: 如果.jar 中的程序时命名行.

无法直接双击运行的. 需要在命令中运行.

java -jar xxx.jar

6. 注意,你需要运行.jar 包时,必须保证你的电脑上配置的有jre ...

# 反射

就业班. 知识储备.

破坏面向对象的一些特性: 封装... 破坏封装...

// 方式一: 类名.class

Class clazzA = Person.class;

// 方式二: 对象名.getClass();

Person p = new Person();

Class clazzB = p.getClass();

// 3.方式三: Class类的静态方法 forName

Class clazzC = Class.forName("cn.itcast.a\_reflect.Person");

获取字节码对象所表示的类的父类. getSuperclass()

获取字节码对象所表示的类实现的接口. getInterfaces()

获取类的修饰符. getModifiers()

获取类的所有的字段(公共的字段) getFields()

获取类的所有的字段(包括私有的) getDeclaredFields()

获取类中所有的方法(公共的),还可以父类的公共方法. getMethods() 返回Method[]

getDeclaredMethod(方法名,参数1，参数2……);

方法的调用invoke(clazz, "jack");

暴力反射setAccessible(true)

获取类中所有的方法(包括非公共的) getDeclaredMethods()

取出类中的构造函数(只能获取公共) getConstructors()

取出类中的构造函数(包括私有的) getDeclaredConstructors()返回Constructor[]

取出类中的一个无参构造函数getConstructor(null) 返回Constructor

取出类中的一个有参构造函数getConstructor(String.class,int.class) 返回Constructor

构造方法的方法创建对象newInstance()

# 输出系统当前时间

**import** java.text.SimpleDateFormat;

**import** java.util.Calendar;

**import** java.util.Date;

**public** **class** ADFU {

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

//使用默认时区和语言环境获得一个日历

Calendar cale = Calendar.*getInstance*();

//将Calendar类型转换成Date类型

Date tasktime=cale.getTime();

//设置日期输出的格式

SimpleDateFormat df=**new** SimpleDateFormat("yyyy年MM月dd日 HH时mm分ss秒");

//格式化输出

System.*out*.println(df.format(tasktime));

}

}

//获取目前时间

SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");

String time=sdf.format(new Date());

# 求阶乘（简单粗暴）

**public** **static** **int** mul(**int** num){

**return** num\*((num>1)?*mul*(num-1):1);

}

**获取系统时间 long** startTime=System.*currentTimeMillis*();