# 职业入门--Java基础语法

## 1-1-1了解Java课程体系

## 1-2-1安装下载JDK搭建Java开发环境

## 1-3-1变量的声明与初始化

## 1-3-2变量值的交换-熟练掌握变量的赋值

## 1-4-1不同进制间的相互转化

## 1-5-1 Java基本数据类型1

## 1-5-2 Java基础数据类型2

## 1-6-1数据运算和比较-表达式-算数运算符

## 1-6-2数据的逻辑运算-条件运算-赋值运算

# 中级导学—贪吃蛇

## 贪吃蛇游戏-类设计

# 高级导学—能力提高

## 我赢独家提高课程之网站架构

# Java基础语法(一)

## 1-1-0梦茹告诉你-学Java你需要做什么

## 1-1-1了解Java课程体系



## 1-2-1安装下载JDK搭建Java开发环境

特点：

跨平台--🡪跨操作系统

类型CPU--🡪不同CPU含有指令不一样

--🡪JavaCPU

Jdk jre jvm

Jdk目录结构

JAVA\_HOME D:\java\jdk

bin | 命令 管理工具

src.zip java开发工具包提供的源码

第一个Java程序

1. java----编译----🡪二进制机器码.class

----🡪操作系统上安装JavaCPU(虚拟JavaCPU) ----🡪jvm

安装jdk java开发工具包

其他语言--🡪编译--🡪.exe

编译： a.进入java所在的目录

b.javac 环境变量----🡪操作系统中用到的变量就是环境变量

PATH JAVA\_HOME\bin

Javac 文件名.java

PATH环境变量要一直有效

2. 运行 java 文件名（.class的文件名，不要包含扩展名）

3. 开发工具

Eclipse/netBeans/intellij/记事本——>可编译的记事本EditPlus

注意：习惯把运行的class的名字作为文件名来保存

如果文件中类是public修饰，那么类名必须和文件名相同

一个java文件中可以放一个public的类

## 1-3-1变量的声明与初始化

## 1-3-2变量值的交换-熟练掌握变量的赋值

变量是用来标示一块内存

变量必须要有类型--🡪决定了内存区域的大小能放什么数据 int

命名：以字母or $ or \_ 开头后面可以跟 数字、字母、\_、$

变量赋值

注意：1.变量必须声明并且赋值之后才能使用，声明的时候必须有类型

赋值的时候必须类型匹配。

2.变量可以重复赋值，但不能重复定义，是指在同一个作用域内。

3.变量有作用域范围，就是离它最近的大括号，只在作用域内有效。

4.将来的话，在实际开发中，变量的命名建议：

a.首字母小写，有多个单词组成，驼峰式命名(除首个单词外首字母大写)

b.望文生义

案例：声明两个变量，并初始化，实现两个变量的交换。

思考：实现两个数的交换，不用第三个变量。

## 1-4-1不同进制间的相互转化

1. 十进制（0到9）

123 3\*10(0)+2\*10(1)+3\*10(2)

` 4356 6\*10(0)+5\*10(1)+3\*10(2)+4\*10(3)

n进制 第一位\*n(0)+第二位\*n(1)+…+第x位\*n(x-1)次方

1. 二进制
2. 0\*2(0)+1\*2(1)+0\*2(2)+1\*2(3)==🡺10
3. 八进制（0到7）
4. 7\*8(0)+6\*8(1)
5. 十六进制（0到9，A,B,C,D,E,F）

9EFA 10\*16(0)+15\*16(1)+14\*16(2)+9\*16(3)

1. 转换操作

二进制转十进制

十进制转二进制

十进制 二进制

1 0001

2 0010

3 0011

4 0100

5 0101

6 0110

7 0111

8 1000

9 1001

10 1010

11 1011

12 1100

13 1101

14 1110

15 1111

例：把35转成二进制

短除法 100011 1\*2(0)+1\*2(1)+0+0+0+1\*2(5)==🡺35 （正向的）

找次方数：2(5)+2(1)+2(0) ==🡺100011 （反向的）

137 = 128 + 8 + 1 ==🡺2(7)+2(3)+2(0) ==🡺10001001

179 = 128 + 32 + 16 + 2 + 1 ==🡺2(7)+2(5)+2(4)+2(1)+2(0) ==🡺10110011

Java中的进制问题----java采用二进制补码方式

1. 二进制补码

假设我现在只能用4位表示数字

0000

0001

0010

0011

0100

0101

0110

0111

1000

1001

1010

1011

1100

1101

1110

1111

没有负数 有牛人就发明了新的办法

0000 0

0001 1

0010 2

0011 3

0100 4

0101 5

0110 6

0111 7

把1开头的认为是负数

1000 -8

1001 -7

1010 -6

1011 -5

1100 -4

1101 -3

1110 -2

1111 -1

4位二进制补码能表示几个数字 2(4)

最大数2(4-1)-1 ==🡺7

最小数-2(4-1) ==🡺-8

假设某种语言是以8位二进制补码表示数字，那么其范围是

2(8-1)-1 到 -2(8-1)

总结1：二进制补码表示数据范围和位数有关，如果位数是n

那么范围就是 -2(n-1) 到 2(n-1)-1

Java中整数(int)是采用的32位二进制补码表示

Java int的范围是 -2(31) 到 2(31)-1

总结2：4位的看

7 0111 取反==🡺1000 +1 ==🡺1001

-7 1001 取反==🡺0110 +1 ==🡺0111

负数 = 正数取反 + 1

正数 = 负数取反 + 1

## 1-5-1 Java基本数据类型1

## 1-5-2 Java基础数据类型2

1 byte ---- 8 bit

1 kb ---1024 byte

1 M ---1024 kb

1 G ---1024 M

1 T ---1024 G

整数：

byte 1字节 8位 范围字节计算

short 2字节 16位 很少使用

int 4字节 32 位最常用

long 8字节 64 位长整形

小数：

float 4字节 32位

double 8字节 64位

其他：

char 2字节 16位 无符号(没有正负号)

boolean 1字节 8位 代表真 or 假

int类型(32位)的最大表示范围：-2(31) ~ 2(31)-1，即

-2147483647 ~ 2147483647，如表示更大的整数可以使用long。

直接量(literal)，即直接写入的常量，整数的直接量的类型默认是int类型，

整数直接量也经常写16进账的形式(以0X或0x开头)。

long类型

long(64位)的最大表示范围：-2(63) ~ 2(63)-1

(9223372036854775807)

如果要表示long直接量，需要以L或l结尾。

Long timeMillis = System.currentTimeMillis();

System.out.println(timeMillis);

System.currentTimeMillis();方法返回1970年1月1日零点

到此时此刻所经历的毫秒数，该方法经常用于计时操作。

浮点类型

浮点数，就是小数，包括：float和double

double的精度要大于float,因此，一般只使用

double计算浮点数。默认的浮点数字面量是double类型

由于舍入误差的原因，浮点数不能精确运算。

例：double money = 3.0;

System.out.println(money – 2.9);

//0.10000000000000009

注意舍入误差的问题，如果需要精确计算，可以使用

BigDecimal。

字符类型

字符类型事实上是一个16位无符号整数，这个值是对应字符的编码。

Java字符类型采用Unicode字符集编码。Unicode是世界通用的定长字符集，

所有的字符都是16位。

字符直接量可以采用诸‘中’的形式，也可以采用16进制的表示形式，

例如：‘\u4e2d’

char c1 = ‘中’;

char c2 = ‘\u4e2d’;

//‘4e2d’为‘中’所对应的16位Unicode编码的16进制表示形式

对于不方便输入的字符采用转义字符表示，如:

‘\n’ ‘\t’ ‘\\’ ‘\b’ ‘\r’ ‘\’’ ‘\”’ ‘\u4e2d’

数字编码：’0’：48，’1’：49 …

英文编码：’A’：65，’B’：66 …

字符是整数，可以参与运算

char c = ‘A’ + 1

int n = ‘6’ – ‘0’

整数字面量(不超范围)可以给char变量赋值

控制字符是没有显示效果

只有本地字符集中存在的字符才能显示

类型转换

自动类型转换(隐式类型转换)：从小类型到大类型可以自动完成：

byte short int long float double

char

强制类型转换：从大类型到小类型需要强制转换符，

会照成精度损失或者溢出。

long l = 1024L \* 1024 \* 1024 \*4;

int i = (int )l;//会造成溢出！

double pi = 3.14159265358979323846;

float f = (float)pi;//会造成精度损失

## 1-6-1数据运算和比较-表达式-算数运算符

## 1-6-2数据的逻辑运算-条件运算-赋值运算

Java运算符

算数运算符

+,-,\*,/,% ++,--

案例：有一个三位数(123)，对其中各个数位求和。

int n = 123;//对其中的数字求和

//取个位

int a1 = n % 10;

//取十位

int a2 = n / 10 % 10;

//取百位

int a3 = n / 100;

关系运算符 计算结果一定是true or false

>、<、>=(大于或等于)、<=(小于或等于)、==(判断是否相等)、!=(不等于)

逻辑运算符 计算结果true or false

逻辑运算符中的每个表达式的结果必须是true or false

&&逻辑与

表达式1 && 表达式2

1和2都为true结果为true，如果表达式1已经为false

表达式2不进行计算(这点称为短路)

||逻辑或

表达式1 || 表达式2

1,2都为fasle结果才为false，只要有一个为true，

结果就为true，如果表达式1为true，那么表达式2不计算。

！逻辑非

条件式运算符

表达式1 ？ 表达式2 ：表达式3

表达式1必须是条件

如果表达式1为true结果就是表达式2的结果

如果表达式1为false结果就是表达式3的结果

赋值运算符

“=”称为赋值运算符 用于对变量赋值。

赋值运算符表达式本身也有值，其本身之值即为所赋之值。

可以使用扩展赋值表达式(+=，-=，\*=，/=，%=)。

扩展表达式可以向表达式左边的数据类型强制转换。

例：有这样的面试题目：

short s1 = 1;

//a. 1 默认是int 能自动转换成short

1. s1 = s1 + 1; 2. s1 += 1;哪个对哪个错？

b. s1 = s1 + 1时，1为int类型 s1为 short类型 加起来结果为int

赋值给了short s1 所以错误

c.而s1 += 1 ====>(short)(s1 + 1);

字符串连接运算符

“+”可以实现字符串的连接。同时可以实现字符串与其他数据类型的”相连”。

## 1-6-3数据的移位运算和进制间的关系

位运算操作符(效率高，都是直接二进制运算)

位移运算符

右移操作 >>有符号 正数补0 负数补1

>>>无符号 正数补0 负数补0

左移操作 << 后面补0(不管正负数) 都相当于移动一位除以2

按位与& 按位或| 按位异或^ 操作数1& 操作数2

操作数1 | 操作数2

操作数1 ^ 操作数2

## 1-7-1 Java中的if分支语句

## 1-7-2 Java中的if-else分支语句

## 1-7-3 Java中的if-else-if语句

## 1-8-1 Java中的switch-case语句1

## 1-8-2 Java中的switch-case语句2

程序的结构

1. 顺序结构
2. 分支结构
3. 循环结构

分支结构

1. If(条件)

{

//条件成立执行的语句 如果条件成立要执行的语句只有一句，大括号可以省略

}

如何通过键盘输入数据

a.在程序开头import java.util.Scanner;

b.然后在主函数中

Scanner s = new Scanner(System.in);

案例：输入一个年龄，如果大于70，就输出老年人，然后bye-bye。

如果其他情况直接bye-bye。

案例：输入三个数字，找出其中最大值。

案例：通过键盘输入三个数字，要求从小到大排序输出这三个数字。

1. if(条件)

{

…语句

}else{

…语句

}

{}大括号中的语句只有一句，{}可以省略

案例：输入年龄,60以上免费(包含60)否则需要花钱买票

案例：输入年份,打印出该年是否为闰年

闰年的条件:y%4==0&&y%100!=0||y%400==0

1. if(条件){

}else if(条件){

}else if(条件)

{}

…

else{

}

案例:输入学生的考试成绩

如果90以上(包含) 优秀

>=80 并且<90 良好

>=70 并且<80 一般

>=60 并且<70 及格

<60 直接不及格

案例：请输入一个数字，判断是正数、负数、还是0。

练习：输入一个年龄 如果是6岁以下打印儿童，大于6岁到14岁打印少年

14到30打印青年，30到50打印中年，50到70中老年，70以上打印老人

1. 一些特殊的if语句

a.if(条件);

b.if()

else if()

else if()

…

无else

1. switch

case

案例:输入一个整数0,打印星期日,1打印星期一,2打印星期二…

注意:

1. break关键字是跳出switch{}执行后续语句
2. 如果匹配上某个case,该case段,没有break

则会默认继续往下匹配,直到遇到break为止.

案例:请输入年月日,判断这个日期在这一年当中已经过了多少天.

1. Switch当中default最多只能出现一次.

可以出现在和case并列的任意位置(一般在最后)

案例:请输入成绩(不判断输入是否合法) 90以上优秀,80-90良好,

70-80一般,60-70及格,0-60不及格.

1. switch(整数表达式的值)

()中的表达式的值只能是byte,short,char,int类型

将来还可以放的是枚举类型.

作业1:

托运计费问题:

当货物总量小于20公斤的时候,收费5元,大于20公斤小于100公斤

的时候超出20公斤的部分按每0.2元每公斤计费,如果超出100公斤

的时候,超出的部分按照每公斤0.15元计算.读入货物的重量,输出计算

之后货物的运费.

作业2:

读入一个三位数,计算其各位数字之和.

例如:123各位数字之和为6.

作业3:

输入一个年月日,求这一年还剩下多少天

(提示:总天数 – 已经过的)

作业4:

输入三个数字从大到小排序输出

# Java基础语法(二)循环

## 2-1-0和梦茹一起学Java中的循环

## 2-1-1 Java中for循环的使用1

## 2-1-2 Java中for循环的使用2

循环结构

1. for循环

for(表达式1;表达式2;表达式3){

//循环体 如果循环体只有一句,{}可以省略

}

表达式2是条件

案例:输出10次HelloWorld

案例:求1到100的和

案例:求1到100之间奇数的和

案例:打印1到100之间既能被3整除又能被5整除的数字

并记录有几个

案例:1!+2!+…+10!;

案例:输入一个数字,打印其所有因子,包含1不包含自己

并对所有因子求和

## 2-1-3根据指定条件跳出for循环-break和continue关键字

Continue语句用在循环体中,用于结束本次循环而开始下一次循环

例如:for(int i=0;i<10;i++){

if(i == 5) continue;

System.out.println(i+””);

}//输出的结果是:0 1 2 3 4 6 7 8 9,

//当i==5时,结束了当次循环,继续下一次循环,因此5没有输出.

Break语句是结束循环,直接跳出for循环结构,继续下面的语句

引入案例:吃包子

案例:输入一个数字,判断该数字是否为素数(素数就是质数)

只能被1和自己整除的数

## 2-1-4 for循环的嵌套使用

for结构的嵌套

遇到for结构,就按照for的执行原理执行完成.

引入案例:打印100到1000之间的所有素数

案例:

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*

嵌套循环可以控制行列

外层循环控制行,内层循环控制列,外层循环结束前之间换行

内外层的控制变量有一定的关系.

案例:

\*\*\*\*\*\*\*

\*\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*

\*\*\*

\*\*

\*

## 2-1-5双重for循环的使用

案例:

\*

\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*

\*\*\*\*\*

\*\*\*

\*

案例:百钱买百鸡

公鸡3元/只,母鸡2元/只,小鸡 1元/两只.

for的特殊结构

1. for(;;);循环体为空
2. for循环可以省略表达式
3. for循环嵌套中的break

for(;;){

for(;;){

break;

}

}//跳出内循环

outer:for(;;){

for(;;){

break outer;

}

}//直接跳出外循环

## 2-2-1 Java中的while循环使用1

while结构

while(条件){

//循环体

}

案例:求1到100的和

案例:请输入一个整数,要计算其各个数位之和

假设输入123 就计算1+2+3 和是6.

案例:输入一个数字,将该数字倒置

输入1234得到4321

案例:求10到1000之间各个数位之和为5的数字,打印输出

While for 互换

while语句和for都是”当型循环”（就是先判断条件，条件不成立不进入循环）可以互换

while(true)等价于for(;;)

do While循环

do

{循环体}(条件,boolean表达式);

循环首先会执行一次

Do-while与while结构的区别在于：如果循环条件一开始就不满足时while不会执行循环体；而do-while结构会执行一次循环体。

作业1：输入一个数，因式分解

如输入90得到90 = 2\*3\*3\*5

80得到80 = 2\*2\*2\*2\*5

作业2：求水仙花数。所谓水仙花数，是指一个三位数abc，

如果满足a3+b3+c3=abc(a的3次方)则abc是水仙花数，

打印所有水仙花数

作业3：如果一个数等于其所有因子之和，

我们就称这个数位“完数”，例如6的因子为1,2,3

6=1+2+3 那么6就是一个完数。

请编程打印出1000以内所有的完数

作业4：打渔还是晒网

中国有句老话叫“三天打渔，两天晒网”。

假设有一个人从2000年1月1日开始“三天打渔两天晒网”，读入三个整数，

分别表示年份、月份、日期，问这三个整数所表示的那天，这个人是在打渔还是在晒网？

求这个日期和2000年1月1日相差多少天

作业5：已知faibonacci(费波那契)数列的前几个数分别为0,1,1,2,3,5,… …。

从第3项开始，每一项都等于前两项的和。

读入一个整数n，编程求出此数列的前n项的和。

注意：这里的数列是从0开始的。

0 1 1 2 3 5 8 13 …

作业6：搬砖问题：36块砖，36人搬，男搬4，女搬3，

两个小孩抬1砖，要求一次全搬完，

问男、女和小孩各若干（要求三种都要有）

## 2-2-2 Java中的while循环使用2

## 2-2-3 Java中的while循环使用3

# Java基础语法(三)函数

## 3-1-0函数解决了梦茹的困惑

## 3-1-1 Java中函数的基本使用

函数 OR 方法

函数描述的就是一个过程或者是行为的描述,可以反复使用,

并通过某些手段,可以替换你需要改变的数据.

函数的引入

函数的声明

public static 返回值类型 函数名(形式参数列表)

{

//函数体

}

函数的返回值类型:

函数是一个过程的调用,调用这个过程,可以得到一些结果,

如果这些结果是一个具体的数值(可以用变量接收,

只能通过return关键字返回),那么返回值就是该数值的类型

如:把两个整数相加得到另外一个整数,把它写成一个函数,

必须通过return关键字返回这个和,那么返回值就是int.

如果只是一个过程的调用,没有任何数据得到(比如只是打印,

只是执行一个过程)

那么返回值类型是void.

函数名称:

习惯,首字母小写,望文生义,多个单词组成驼峰式命名.

形式参数列表:

可以根据用户需求,在这个功能上需要被代替的数字.

如:求两个数相加,哪两个数呢?任意的,那么我们就需要用

两个变量代替这两个任意的数字

如:我们要打印一个数字的因子,这个数字是哪个呢?任意的,

我们就需要用一个变量来代替这个数字.

这个代替就是形式参数,形式参数可以是多个,用逗号隔开.

函数的位置:和主函数并列

return关键字

参数列表

函数的调用

使用的时候直接 函数名(参数) =🡺具体的数据

为什么叫形式参数(形参):

调用函数传递的具体的值是实际参数(简称实参)

案例1:打印一个数字的因子,写成函数

返回值类型:只是打印 返回值void

函数名称:printSons

形式参数:一个数字,是任意的,可以替换的,用一个变量表示,

这个变量就代替了这个任意的数字,int a,变量名可以随意

public static void printSons(int a){//函数体}

案例:任意两个数求和写成函数

返回值类型:如果打印和,可以不用,直接void

如果希望得到和,然后做其他处理,必须是

通过函数能得到和,返回值是int

函数名称:add

函数的形式参数:任意两个整数的和,这两个整数可以替换,

用两个变量表示.

public static int add(int a,int b)

案例:打印出100到1000之间所有的素数

把判断素数写成函数(判断一个数字是否为素数写成函数)

返回值类型:a.是 打印是素数,不是 打印不是素数.--🡪void

b.是得到true,不是得到false--🡪boolean

函数的名称 isPrimeNumber

函数的参数:一个数字是任意的,应该用一个变量替代

函数的声明:public static void isPrimeNumber(int n)

案例:打印10到1000之间各个数位之和为5的数字

要求:求一个数字的各个数位之和先写成函数

函数的返回值:int(各个数位的和 返回)

函数的名称:getSum

函数的形式参数:一个数字,人一定,用一个变量替代

Public static int getSum(int n)

案例:求10到1000之间所有的可逆素数

该数字是素数,该数字倒置之和还是素数称为可逆素数

a判断一个数字是否为素数 -🡪写成函数

b把一个数字倒置 -🡪写成函数是

案例:计算两个日期之间相差的天数

案例:写一个函数,实现两个变量的交换(注意内存分配的问题).

## 3-1-2熟练使用Java中的函数1

## 3-1-3熟练使用Java中的函数2

## 3-1-4函数的递归调用

案例：写一个函数，求一个数据的阶乘。

案例：数列1,1,2,3,5,8,13,21,34… …求第n项的值

**Java基础语法之数组**

## 4-1-0梦茹告诉你什么时候用数组

## 4-1-1 Java数组的基本使用

数组：相同数据类型元素的集合

表示 一个组中所有学员的成绩 -🡪用普通智能表示一个数组。

数组的声明

第一种声明方式：

Int[] a = {1,2,3,…}; or int a[] = (1,2,3,…)

第二种声明方式：

int[] b;也可以是int b[];

b = new int[]{1,2,3,…} =🡺 int b = new int[]{1,2,3,…}等价

第三种声明方式：

int[] c = new int[10];然后一一初始化(也可以是：int c[] = new int[10])

数组的访问方式

数组名.length得到数组的长度，通过下标访问，从0开始

数组名[下标]

数组名，是对数组的引用（引用：数组名指向了那个数组-🡪其实就是地址）

数组是引用类型，数组开辟的是堆内存，将来释放时，需要jvm回收(后续讲解)

案例:创建一个整数数组,通过键盘输入进行初始化,求数组中的最大值,最小值

和.平均值.

案例: 创建一个数组并初始化,让这个数组倒置

案例: 把数组倒置案例写出函数

写一个函数把任意数组倒置

函数的返回值:要不要有返回值?有,没有都实验一下

函数名称:reverseArray

形式参数:任意的一个数组,数组作为参数

案例:求一个数的因子(包含1,也包含自己),放入数组并返回

案例:获取两个正int类型数组中相同且最大元素

案例:求两个数字的最大公约数和最小公倍数(辗转除法-🡪对数学特别清楚)

1. 分别找出两个数的公约数,并分别放入数组
2. 找出最大的,就是最大公约数
3. 两数相乘/最大公约数=最小公倍数

数组的扩容,复制

案例:有一个数组已经初始化完毕,现在向其下标3插入一个元素.

ArrayIndexOutOfBoundsException,数组运行时的异常,数组下标越界

自己去实现 也可以

System.arraycopy()这个函数

Arrays.copyOf()函数

案例: 写一个函数,向整数数组的pos位置插入一个元素value

## 4-1-2函数和数组的综合运用1

## 4-1-3函数和数组的综合运用2

## 4-1-4深入学习Java中数组的内容

## 4-1-5数组的选择排序和冒泡排序-常见的排序

数组的排序

1. 选择排序

原理: a[0] a[1] a[2] a[3] a[4] a[5] 选择排序与三个数的比较排序是一样的算法

a[0]和a[1]进行比较 如果大 则交换 -🡪 a[0], a[1]中小的放入了a[0]中

a[0]和a[2]进行比较 如果大 则交换 -🡪 a[0], a[1], a[2]中小的放入了a[0]中

a[0]和a[3]进行比较 如果大 则交换 -🡪 a[0], a[1], a[2], a[3]中小的放入了a[0]中

… …

a[0]和a[a.length]进行比较 如果大 则交换 -🡪 所有元素中小的放入了a[0]中

---------------------------------------------------------------------------------------------------------------------

a[1]和a[2]进行比较 如果大 则交换,a[1]和a[3]…a[1]和a[a.length-1]比较,剩下的最小的放入了a[1]中

---------------------------------------------------------------------------------------------------------------------

… …

a[4]和a[a.length-1]进行比较-🡪最后一轮

1. 冒泡排序

原理: a[0] a[1] a[2] a[3] a[4] a[5]

a[0]和a[1]进行比较 如果大 则交换 -🡪 a[0], a[1]中大的放入了a[1]中

a[1]和a[2]进行比较 如果大 则交换 -🡪 a[0], a[1], a[2]中大的放入了a[2]中

a[2]和a[3]进行比较 如果大 则交换 -🡪 a[0], a[1], a[2], a[3]中大的放入了a[3]中

… …

a[4]和a[5]进行比较 如果大 则交换 -🡪 所有元素中最大的放入了a[5]中

------------------------------------------------------------------------------------------------------------------第一轮

a[0]和a[1],a[1]和a[2], a[2]和a[3], a[3]和a[4] ------------------------------------------------第二轮

… …

最后一轮a[0]和a[1]比较.

1. 插入排序

原理: a[0] a[1] a[2] a[3] a[4] a[5]

a[0]作为基本量 拿出a[1]进行插入工作 -🡪 a[0]和a[1]已经排序好了

a[0], a[1]作为基本量 拿出a[2]进行插入工作 -🡪 a[0],a[1] ,a[2]已经排序好了

a[0],a[1] ,a[2]作为基本量 拿出a[3]进行插入工作 -🡪 a[0],a[1] ,a[2],a[3]已经排序好

… …

引入:写一个函数,向一个已经从小到大排好序的数组中插入一个元素

插入之后,该数组还是从小到大排序.

1. 数组中本身也听过了api. Arrays.sort(数组)

Arrays.toString方法 用于字符串表示数组元素

Arrays.equals方法 用于比较两个数组的各个元素是否相等

Arrays.sort方法 用于实现数组的排序

案例:用一个整形数组,然后大小是100万,赋值之后进行排序.

## 4-1-6数组的插入排序-常见的算法及总结

## 4-1-7 Java自带的数组排序及多堆数组初步

多维数组

二维数组

二维数组,是由多个一维数组组成,每一个一维数组

都是二维数组一个元素.二维数组的长度就是其中

包含的一维数组的个数

int[][] a = {{},{}};

int[][] a = new int[][]{{},{}};

int[][] a = new int[10][10];//将来再初始化

案例:打印杨辉三角前8行

1

1 1

1 2 1

1 3 3 1 a[3][1] = a[2][1] + a[2][0] ==🡺a[i][j] = a[i-1][j-1] + a[i-1][j]

1 4 6 4 1

1 5 10 10 5 1

从第三行第二列(不包含最后一列)

三维数组

三维数组的长度就是其中含有二维数组的个数

字符串和数组//由多个字符组成用双引号

案例:双色球 6颗红球(33选1),1颗蓝球(16选1)

01,02,03,…33

## 4-1-8经典案例输出杨辉三角-深入掌握多维

## 4-1-9随机生成双色球号码-字符串和数组的应用

# 阶段性项目案例(一)

## 酒店前台客房管理系统1

## 酒店前台客房管理系统2

# Java面向对象(一)

## 5-1-0梦茹带你走进面向对象的世界

## 5-1-1了解Java中的面向对象-掌握Java中的类-对象-成员

面向对象-🡪描述现实生活

类,对象,引用

**类**:把具有相同属性和相似行为的一类事物称为类

相同的属性--🡪用数据来表示即可--🡪用一些数据类型就能表示

相似的行为--🡪用方法or函数表示即可

找出相似的行为和相同的属性的过程称为抽象

其中找出相同的属性称为数据抽象

找出相似的行为称为过程抽象

案例1:人类:

相同的属性--🡪身高,体重,年龄

相似的行为--🡪吃饭,走路,睡觉等 不需要写public static了.

通过封装行为构成类

class 类名{属性,行为}--🡪这个过程称为封装

案例2:钟类

相同的属性:时,分,秒

相似的行为:设置的时间,显示时间

案例3:点类

相同的属性:x,y

相似的行为:到圆心的距离,显示点的坐标位置

属性和行为谁先谁后,是没有关系的.每个属性和行为地位都是并列的,不分先后.

**对象**:类的实例就是对象(创建一个类的实例,创建一个对象)

如何创建对象:

就现在而言:类名 对象(引用)名 = new 类名();

**成员**

成员变量:类的属性

成员函数(方法):类的行为

暂时而言:类中只能放成员,成员都是并列的,不分先后

暂时而言:成员必须通过对象去访问 用.运算符

初始化(给成员变量进行赋值)

1. 成员变量有默认值.
2. 构造函数--🡪对象创建成功之后,就有初始值.

b1.构造函数是一个成员函数,该函数的函数名称与类名相同

b2.无任何返回类型,连void都没有

b3.创建对象的时候会自动调用构造函数,不用显示调用,

并且创建一个对象就会调用一次构造函数

b4.就现阶段而言:构造函数的主要作用是给成员赋初始值.

b5.如果一个类,没有构造函数,那么jvm会给类自动

添加一个无参数的构造函数 称之为默认构造函数

如下:public 类名(){}

一旦,人为的给了构造函数,那么这个默认的构造函数就不存在了.

函数重载

概念:同一个类中的一组函数,这组函数,函数名称都相同,参数不同,(类型不同,个数不同,类型个数都不同),与函数的返回值类型无关.

在函数调用的时候,会根据参数的类型和个数自动匹配,决定调用哪个.但是这种匹配未必是精确的,符合就近原则(找最精确的取匹配,如果没有最精确的找能够匹配得上的,不需要人为转换)

构造函数重载

定义初始化:给成员变量直接赋值就是定义初始化.什么时候用呢,当该类的所有对象的某个属性都一样的时候.

顺序,也就是说创建对象的时候,构造函数,定义初始化都被执行

谁先,谁后?

定义初始化块

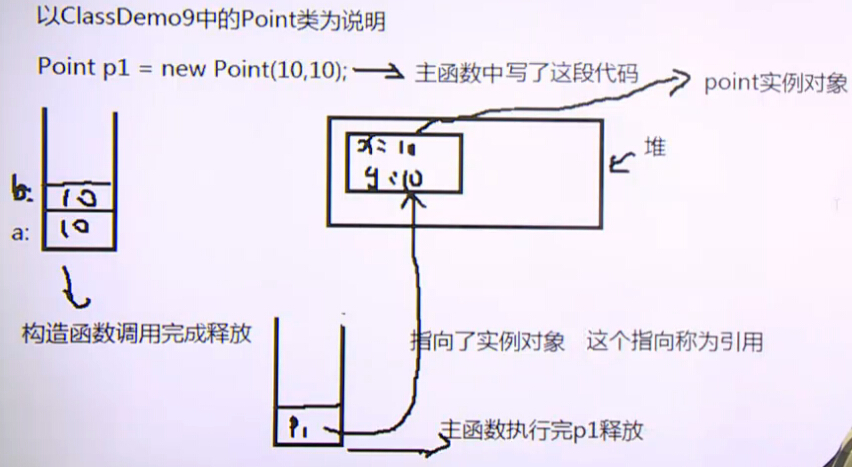
定义初始化先执行,然后再构造函数,如果存在多个定义初始化,那么这些定义初始化,从上到下依次执行,都在构造函数之前.

内存分配

1. 创建对象的过程内存分配的问题.

堆,栈,方法区 --🡪类的信息都存放于此

1. 对象和引用



this关键字

this引用 --🡪当前对象(的引用)

1. this在什么地方

一个成员函数(非静态的)访问另外一个成员(非静态的)

前面都省略了this

1. 当前对象(的引用) --🡪调用这个函数的那个对象(的引用)

案例:初始化的时候形式参数的变量名可以和成员一样.

成员前用this区分

案例:ClassDemo14.java

1. this(参数)调用另外一个构造函数,必须放在第一句,且只能调用一次

引入案例:写一个圆类 ClassDemo15.java

圆心是一个点(要用到点类)Point

4.以后有类名.this的用法(讲内部类的时候会讲这个问题)

静态成员

引入案例:ClassDemo16.java

1静态成员变量是属于类的,不属于某个对象

2访问时可以用类名.静态属性直接访问,也可以用对象.静态属性访问,后者不提倡

3静态成员变量,在用到类的时候就会完成初始化加载(验证:ClassDemo18)

4静态的成员方法

可以用对象.访问 不提倡,提倡用类名.访问

静态的成员方法只能访问静态的成员.(为什么我们在学面向对象之前,函数都写成了static?因为,主函数本身是静态的,只能访问静态成员)

命令行参数

运行时,给主函数的参数.

Java 类名 空格 给主函数的参数(空格隔开 列出.都会给主函数的字符串数组)

## 5-2-1掌握Java中的初始化操作

## 5-3-1掌握Java中的函数重载

## 5-4-1初步掌握Java中的内存分配

## 5-5-1掌握Java中的this关键字

## 5-6-1 this关键字补充及Java中的静态成员

## 5-7-1静态成员补充及面向对象知识点总结

# Java面向对象(二)

## 6-1-0什么时候用继承

## 6-1-1 Java中的继承

继承:主要是代码的复用(设计中意义会更多)

继承使用的关键子 extends

Java中的继承是全部继承,除构造函数之外

Class Circle extends Shape Circle这个类就称为Shape的子类,派生类

Shape类就称为Circle的父类(基类)

1. 继承中,子父类的构造函数的问题

a子类必须调用父类的构造函数(构造函数是不被继承的)

如果没有写调用,那子类的构造函数默认会调用父类

无参数的构造函数(该句默认就在子类构造函数的第一句)--🡪super();

如果要显示的调用,那么要用super(参数列表),而且该句必须在子类

构造函数中的第一句,需要给父类的构造函数传参数

b有子类,有父类初始化的顺序

1先父类静态,然后子类静态,//跟创建对象无关,只要用到类就会先加载其静态属性,并且只加载一次

2父类的定义初始化,父类的构造函数

3子类的定义初始化,子类的构造函数

## 6-1-2继承中构造函数的调用及初始化的顺序

## 6-1-3方法的重写

方法的重写

1父类中有一个方法,到子类中有重新的实现

2访问权限必须是非private,子类中还可以放大(后续详细讲解)

3super.方法()调用父类被重写的方法(因为如果不重写直接调用即可)

总结:super关键子, super(参数列表) --🡪调用父类的构造函数,必须在子类的第一句

Super.函数 --🡪在子类重写父类的方法里,调用父类的该方法

(Demo6,60行)

## 6-1-4动态binding1

## 6-1-5动态binding2

动态binding

1.父类的引用可以引用任何子类的实例对象

2.如果父类的引用,引用了子类的实例

调用方法时,如果方法构成重写,那么

调用子类的方法,否则调用父类的方法 ---->动态绑定(binding);

对象数组

## 6-2-1 Java中的访问控制权限1

## 6-2-2 Java中的访问控制权限2

访问权限

private/protected/默认的/public

a.private 用来修饰成员的话:说明该成员只能在类的内部访问

b.包的概念 包是类名的一部分,主要用于命名的管理

之前我们写的类,都没有声明包,就是在默认包里

1 显示声明包 package aa 包对系统来说就是文件夹

但是对于java来说是类名的一部分

对于有包的类:直接编码,建立包(就是文件夹)然后把.class扔进去

javac -d . 类名(文件名).java

-d .代表默认在当前的目录下建立包

2 导入包 import 包名.类名; /import 包名.\*;

3 默认的,protected,public

默认的可以修饰成员和类 ---->代表可以在同一包下直接访问

protected 修饰成员 ---->在同一包中,或者不同包的子类中可以访问

public 修饰成员和类 ---->在不同包中可以访问

4 classpath 环境变量

寻找类的路径 默认是. 代表当前路径,多个设置的时候中间用;隔开

一般情况,成员变量private 函数public

5 构造函数private 如何创建对象

通过静态方法(Demo14.java)

## 6-3-1 eclipse开发工具的介绍

开发工具eclipse(绿色(解压就能用),开源(免费的))

1 解压,选择工作目录

2 file--new---->New Project---->Java Project---->输入项目名称(英文)---->finish

3 如果一不小心关了视图

Window--->Show View--->点击你想要的视图

4 注意:项目建立完成后,在项目管理的视图中,项目下有src目录

该目录就是监理我们的类的 直接右击src--->new--->Class 填写类名

(修改字体) Window--General--Colors and Fonts

运行:右击带主函数的类 Run As--->Run application

5 默认最常用的的快捷键alt + / 自动提示的功能

main alt+/

System.out.print() 敲 syso alt+/

更改快捷键:Window--偏好设置(prefeneces)--General--Keys--Content Assist--添加快捷键

6 学会建立包,包下建立类,直接在包中右击--->new---class

7 我们直接运行,并没有编译,eclipse自动帮我们完成了编译的工作

编译的.class在工作目录下的项目下的bin目录中

8 导入项目

file-->import-->General-->双击Existing Project into workspace-->Browse-->选中项目名称

## 6-4-1 JavaBean规范

JavaBean

JavaBean就是一个java类

标准的javaBean就是一个java类 属性private,有无参数的构造方法

对所用的属性都有getter/setter方法

方法重写 补充一下

在方法重写(函数覆盖)的时候 访问权限可以扩大

要想重写方法,父类中必须不能private,因为private 子类无法直接访问

访问权限可以扩大,不能缩小

## 6-5-1方法重写补充及Java中的Object类1

## 6-5-2 Java中的Object类2

java.lang.Object

1.很多类都没有导包 比如:String类,Math类

都是java.lang包下的类,是jdk提供的,

jdk提供的类中,java.lang包下的类是不用导包,是直接可用的

2.一个类如果没有父类,那么它的父类就是Object类

java中的继承是单一继承,就是一个类只有一个直接父类

java中的继承具有传递性

a.A继承了B,B继承了C,C没有父类

b.C的父类就是Object 那么Object类中的元素继承到了C

中,就继承到了B中,继承到了A中

可以认为B是A的父类,C也是A的父类,Object类也是A的父类

3.在eclipse中如何看java的源代码

一直按ctrl 让鼠标移入到你要看的函数or类上 第一次会出现

class界面上有一个选项--->Attach Source--->点击--->选择

文件所在的文件夹或者压缩包

4.Object类中

equals方法(注意重载,重写)

toString方法--->打印对象,就会打印该对象toString的返回内容

5.equals方法和hashCode方法

我们编程最终是需要和现实生活打交道,一般equals方法会作为

现实中两个对象的比较,如果返回true,我们就认为是同一个对象,

所有在使用equals方法的时候,需要注意 用唯一标识生成equals方法

和hashCode方法,直接利用eclipse工具帮我们生成即可.

变量的隐藏

方法有重写or覆盖,那么成员变量呢?---->变量(非private)不存在覆盖的

# Java面向对象(三)

## 7-1-0接口解决了梦茹心中的疑惑

## 7-1-1 Java中抽象类的基本运用

1.抽象类

a.抽象方法---->有些行为确实存在,但必须到子类中才会有具体实现

那么这个时候,父类中只需要有声明即可(abstract),不需要有实现

如何声明呢,abstract

b.如果一个类中有了抽象方法,该类就是抽象类

必须用abstract 关键字声明

c.抽象类就是用来继承的.

当一个类继承抽象类的时候,必须重写抽象类的抽象方法,如果不重写,

那么该类自己变成抽象类,必须用abstract声明

d.抽象类不能实例化,只能实例化其子类的对象

只要看到抽象类的引用,引用的一定是其子类的实例

如果看到函数的参数是抽象类,那么调用函数时传递的是该抽象类子类的对象

如果看到函数的返回值类型是抽象类,那么返回的一定是该抽象类子类的实例 对象

2.final 关键字

修饰变量:该变量只能赋值一次,如果作为成员变量必须有初始值(定义初始化or构 造函数初始化) (Demo4)

函数:final修饰函数,该函数不能被重写(Demo5)

类:final修饰类,该类不能被继承,该类是断子绝孙的

final和abstract是不能一起使用的

private和abstract也是不能组合

3.接口:接口是一个标准,更多体现能干什么 can-do

抽象类是体现的继承 is-a 关系

a.接口用interface关键字声明---->低位等价于类的低位,同样会产生.class文件

b.接口中所有的方法都是public abstract的,即使没有这样声明

所以所有方法在接口中都不能有实现

c.定义了一个CanFly接口,我们认为在编程中,实现了这个接口的

类的对象,就是具备了飞的功能

实现的时候用implements关键字

实现接口,必须重写接口中的抽象方法,否则自己变成了抽象类

需要用abstract声明

d.接口也是不能实例化的.接口中是不能有构造函数的,全部都是抽象的方法

看到接口的引用,引用的一定是实现了该接口的类的对象

e.如何理解接口

主板制造商 ---->主板需要插入内存,显卡,... ...

本来的原理是:主板制造商需要等待内存制造商,显卡制造商把内存,

显卡弄出来才知道如何去制造主板

但现实生活中并不是这样,内存,显卡都有标准,

那么主板制造商就不用等,直接根据标准,预留接口即可

那么这个标准就是接口.(MainBord/Memory/DisplayCard/Computer)

f.一个类可以实现多个接口(具备多个标准,具备多个功能,我即是u盘,有是MP3)

就需要实现接口中的所有方法

一个类可以继承一个类同时实现多个接口

g.java中是单一继承,单是接口与接口直接可以进行多继承,

就是一个接口可以由多个接口继承而来.

h.接口中甚至可以没有任何方法,就是一套标准.

Serializable,CloneAble

i.接口中可以声明变量,接口中声明的变量都是

public static final的,即使没这样声明也是这样的(Demo9)

k.接口中有非常多的方法,而我们再用的时候,每次只用个别的方法,

非常麻烦,可以写一个适配器类,见Demo10(MouseAdapter)

这种缺省适配器类一般都是抽象的,明确指定不需要对象

什么是抽象类:只要是abstract声明的类,就是抽象类.

4.内部类(只要是类,就会产生.class文件,内部类也是一样)

1.成员内部类

a.直接在外部类的其它成员中访问,创建内部类的对象(最简单,最常用的)

b.如果成员内部类访问权限为非private,直接访问就可以

Outer.Inner inner = outer.new Inner(参数);(用的较少)

c.类名.this 成员内部类中访问外部类的当前对象(Demo11)

2.静态内部类

a.直接在外部类成员中访问(简单,常用)

b.如果非private 直接创建对象

Outer.Inner inner = new Outer.Inner();

c.静态内部类中只能访问外部类的静态成员.

3.局部内部类

一般声明在某个函数的内部,只在该函数内有效,

一样会产生.class文件

4.匿名内部类--🡪(局部内部类的特殊情况)

a.什么时候用?下面的两种情况用

1已经知道父类,获取其子类的实例对象

2已经知道接口,获取实现了该接口的类的对象

匿名类在用的时候必须是直接获取该匿名类的对象

b.怎么用?怎么样获取对象呢?(Demo14,Demo15)

公式: new 父类 OR 接口 ( )

{

子类的实现 OR (实现了该接口的类的实现)

}

公式得到的是匿名类的对象,实现了该接口(继承了该抽象类的实例对象)

c.上面的公式看起来一样,但是要注意有区别

new 父类(可以给父类的构造函数传递参数) { 子类的实现 }(Demo16)

接口是不存在这个问题,因为接口是没有构造函数的

注意:在内部类中访问局部变量,该变量必须声明为final

## 7-1-2 Java中的final关键字

## 7-2-1 Java中的接口的基本运用

## 7-2-2深入理解接口的运用

## 7-2-3实现接口间的多继承

## 7-3-1 Java中成员内部类和静态内部类

## 7-3-2 Java中局部内部类和匿名内部类

## 7-3-3 Java内部类小结

# Java面向对象(四)

## 8-1-0和梦茹一起解决代码中的异常

## 8-1-1 如何处理程序的意外错误 - Java异常处理

Java的异常

1. 什么是异常

test(){

openFile();//文件可能打不开

readFile();//文件可能读取失败

writetoOtherFile();

closeFile();

}

你的程序当中可能出现意外,这些称为异常,如上面的程序,

当然也有内存不够等意外

Java中已经把异常or错误 都分类了

首先java中把错误or异常分成了两大类,都是java.lang.Throwable的子类

java.lang.Error 系统级的异常,一般程序很难调试

java.lang.Exception 程序级的异常,通过代码可以处理

1. java.lang.Exception是一切程序级异常的父类

a.java中把所有能产生的程序级异常都进行了分类,我们

程序中出现的任何异常都能在分类中找到.

b.Exception大的方向也分成了两种,

1.jvm能自动捕获的异常,java.lang.RuntimeException及其子类

2.jvm不能捕获的异常必须通过程序自己处理,如果不处理,程序编译不通过

3. test(){

openFile();

if(打开成功){

readFile();

if(readFile成功){

readFile();

if(writetoOtherFile成功){

}else{

//…处理异常

}

}

}else{

//…处理异常

}

}

以上的方式是太麻烦了,把本来的业务代码和异常处理都搅合在一起,非常乱

1. try catch关键子

try{

业务代码(可能会产生异常)

}catch(){

异常的处理

}

1. try可以有多个catch

多个catch的时候要注意,子类要放在父类的前面,因为父类的引用可以引用子类的对象

java.lang.Exception是一切异常的父类,它就可以处理任何异常(可能信息不太精确)

1. finally关键子

不管有没有异常最终都要执行的语句(有一些资源需要关闭)

try{}catch(){}finally{}

try{}finally{}(lesson4.Demo5)

4. throw throws关键字

throw 在某种情况下,抛出(制造出)一个异常

throws 是用来回避异常的(Demo7)

注意: 当子类重写父类方法的时候,不能throws比父类更多的异常,除非多throws的异常是RuntimeException(Demo8)

原理:java中,父类可以接收任何子类对象,如果子类比父类有更多的约束,

那么父类处理不了就出问题了.所以在java中子父类进行继承

子类不能比父类有更多的约束.面向对象的设计中,有一个里氏替换原则

说明的就是这个问题.

5.Exception有几个常用的方法

getMessage()获取异常的信息

toString()

printStackTrace();

6.自定义异常

自定义的异常需要继承已经提供的异常类

7.有了异常之后一些细节需要注意

如:函数如果有返回值,那么不管有何异常产生都要保证函数有返回值Demo12

函数中如下:try{}catch(){}finally{} return 返回值

try{return …}catch(){return …}finally{}

是finally先执行还是return先执行?

不管怎么写,函数遇到return就结束,所以finally一定在return之前执行

## 8-1-2 捕获处理代码中的异常

## 8-1-3 抛异常处理 - throw和throws关键字

## 8-1-4自定义异常

# 阶段性项目案例(二)

## 影片租赁系统-根据需求搭建项目

需求描述:

1:完成影片连锁企业租赁管理系统,可以计算每一位客户的消费金额和影片的详细信息,

金额根据影片的类型和租赁的日期来进行计算.

2:客户把影片分成3类进行管理,A:最新电影 B:普通电影 C:儿童电影.

3:费用计算规则如下

a:是普通电影,如果租赁期小于2天,费用为2元,如果租期大于两天,费用是租期减去2,每天1.5元.

b:如果是新片电影,费用为每天3元

c:是儿童电影,如果租期小于3天,费用为1.5元.如果租期大于三天,费用是租期减去3,

每天1.5元

4:每次客户租赁电影可以为客户积累积分,规则是每次累计增加1分.如果是新片,并且

租期大于1天,再增加1分.

5:暂且不要考虑系统界面和系统的存储问题.

## 项目代码重构-容器的细致处理

## 项目代码重构-解决类中代码臃肿实现类功能

## 项目代码重构-使用迭代器遍历容器

## 项目代码重构-解决版本二中代码臃肿问题

## 项目代码重构-使用面向对象思想设计代码

## 项目代码重构-将项目进行抽象设计

# JavaSE Core

## 9-1-0梦茹告诉你如何使用java工具类

## 9-1-1 JavaString类

Java常用类介绍

1. java.lang.String(学会使用帮助文档)

1.1final修饰,不能被继承

1.2 String的构造函数

直接赋值 String s = “hello”;字符串常量的问题

* 1. 字符串连接的问题 +
  2. 常用方法

charAt(),length(),indexOf,lastIndexOf,substring等

1. StringBuilder,StringBuffer

字符串拼接时用这两个类,效率高,节约内存,如果用

字符串 + 的话,性能差,并且浪费很多空间,产生很多垃圾

StringBuilder线程安全,StringBuilder线程不安全,在多线程场景下

性能比StringBuffer要高.

案例:String getString(int length) 产生如下字符串:

(?,?,?) ?的个数是length个.(StringBuilderDemo)

1. 正则表达式(Pattern)

3.1用一些特殊的有意义的字符组成的字符串

验证输入的邮箱是否合法

Java JavaScript PHP等语言都有正则表达式,稍有区别

java.util.Patten static compile(正则表达式r)

Split(字符串) 对当前字符串用r分隔

java.util.Matcher p.matcher(字符串)

find()看是否有匹配 如果有返回true 会继续往下找

group()

String matches(判断一个字符串是否匹配一个正则表达式)、split

3.2原子:正则表达式的最基本的组成单位

只要一个正则表达是可以单独使用的字符就是原子

a 所有可以显示的字符 或 非打印的字符

b .、？、\*、+等都是有特殊意义的字符

想作为原子使用必须转义

\\.、\\？、\\\*、\\+

C 在正则表达式可以直接使用一些范围的原子

[\\d](file:///\\d) 表示任意的一个十进制数字[0-9]

[\\D](file:///\\D) 表示任意的一个除数字之外的字符[^0-9]

[\\s](file:///\\s) 表示任意的一个空白字符空格[\n\r\t\f]

[\\w](file:///\\w) 表示任意一个字a-zA-Z0-9\_[a-zA-Z0-9\_]

[\\W](file:///\\W) 表示非a-zA-Z0-9\_[^a-zA-Z0-9\_]

d 自己定义个原子

[0-9]

[5-8]

[a-z5-8]

[^0-9] ^代表取反 不能是0-9必须放在[]里面的第一个字符

[^a-zA-Z0-9]

3.3元字符

用来修饰原子用的,不能单独出现

\*:表示原子可以出现0次1次或者多次{0,}

+:表示原子可以出现1次或者多次{1,}

?:表达原子可以出现0次或者1次{0,1}

{}用于自定义原子出现的次数

{m} 出现m次

{m,n}m到n次,包含m和n

{m,}最少出现m次,做多无限

.:表示除换行符之外的任意一个字符

^:表示必须以什么什么开头

$:表示必须以什么什么结尾

|:表示或的关系

1. java.lang.Math
2. java.util.Date(DateDemo)

格式化日期

字符串到日期的转换

1. java.util.Calendar

1. java.util.System

## 9-1-2 StringBuffer类和StringBuilder类

## 9-2-1正则表达式1

## 9-2-2正则表达式2

## 9-3-1 Math类 - Date类

## 9-3-2 Calendar类 - System类

## 9-4-1 Java中的Class类

Class类的使用

1. java.lang.Class (ClassType)类类型

面向对象的原理中,一般认为万事万物皆对象

那么类也是对象,这个对象是哪个类的呢,这个对象该如何表示呢

class A{}

A类的对象如何表示 A a1 = new A(); a1即是这个对象

A类本身也是对象,是java.lang.Class类的对象.

这个对象称为类的类类型(class type),如何表示呢?

有三种表示方式

1. 类名.class
2. 如果有对象存在可以直接对象.getClass()
3. Class.forName(“类的全称”);//代表了动态加载类(Office案例)

1.2理解动态加载类的方法(Office案例)

将来功能性的类往往都是动态加载而非静态加载

1.3Class类的常用方法(ClassDemo3)

我们可以通过Class类,就可以获取类的所有信息

包括方法,成员变量,构造函数等等

1. 方法的反射(java.lang.reflect.Method类的使用)

如何获取某一个方法(MethodReflectDemo)

方法的反射操作 method.invoke(object,参数)

很多时候我们需要根据方法名称调用方法.( MethodReflectDemo2)

案例:根据标准javaBean的属性名获取其属性值

1. 成员变量的反射

如何获取某个成员变量

成员变量的反射操作 set/get 两个方法

案例:写一个方法,public static void changeValue(Object obj)

把obj字符串属性的值都变成大写,int属性的值全部加100

(MethodReflectDemo3)

案例:写一个方法,public static String getSql(Object object)

该方法返回的是”insert into 类名(属性1,属性2,…)value(?,?,…)”

1. 构造函数的反射

如何获取某一个构造函数

如果通过构造函数的反射创建对象

1. 数组的反射

java中数组都是对象

数组的类类型的表示,对象.getClass, 类型维数.class

java.lang.reflect.Array(ArrayReflectDemo4)

## 9-4-2如何动态加载类

## 9-4-3 Class类的常用方法

## 9-4-4方法的反射

## 9-4-5成员变量的反射

## 9-4-6构造函数的反射

## 9-4-7数组的反射

# JavaSE Core – 集合框架

## 10-1-0和梦茹一起学集合

Collect

List

ArrayList,LinkedList

Set

HashSet,TreeSet

Map

HashMap

TreeMap

Hashtable

Properties

集合综述

## 10-1-1了解ArrayList使用

1. 集合操作(java.util.\*)
2. Java.util.Collection
3. ---🡪java.util.List
4. ---🡪java.util.ArryList(Vector)
5. ---🡪java.util.LinkedList
6. ---🡪java.util.Set 集合中不能放重复对象,是否重复由equals方法决定
7. ---🡪java.util.HashSet
8. ---🡪 java.util.TreeSet
9. java.util.ArrayList

1.1基本使用(底层操作的就是数组,对数组添加元素,

移除元素,扩容等操作)

添加add(),有效元素size(),遍历的三种方式

contains() 注意和实际场景的的使用

remove() 移除元素

clear()清空所有元素

… …

* 1. 集合中泛型的使用

集合容器可以容纳任何对象

但是在实际应用中,往往都是容纳同一个类型的对象

注意:泛型只是在编译阶段有效,防止错误输入,

如果绕过编译那么就可以绕过泛型

1.3 ArrayList源代码

底层是数组(我们在电影租赁小项目中其实已经模拟过)

重要的成员:Object[] elementData;

int size;有效元素的个数

其实数组的初始大小是10

扩容、后移的操作

移除元素要做前移操作

注意:ArrayList这种有数组作为底层的操作

最好估算元素的多少来指定底层数组的大小

ArrayList list = new ArrayList(大小给定);

来避免扩容操作,因为扩容操作会产生垃圾,

占用内存,性能较差.需要垃圾回收等操作

* 1. java.util.Vector

底层也是数组的操作,所以大部分功能和ArrayList一样

遍历中有一种老的方式Enumeration,

ArrayList存在的遍历方式也都有

ArrayList是线程不安全的,Vector是线程安全的

jdk7之前的版本,扩容大小稍有差异

1. LinkedList(链表结构)
   1. 基本使用和 ArrayList 类似

提供了更为丰富的头尾的操作addFirst,addLast,getFirst,getLast

* 1. 头尾操作较多的场景适合使用LinkedList

比如:贪吃蛇游戏

(控制台操作贪吃蛇Node,Worm,WormPanel,WormTest)

* 1. 数据结构(链表结构--🡪双向链表)

内部包含了一个静态内部类Node(Object,Node next,Node prev)

1. java.util.HashSet
   1. 基本使用

注意:容纳的对象必须根据自己的唯一标识来重写equals和HashCode

方法.

注意:对象一旦放入了HashSet容器中,那么对象的唯一标识属性的值

不能再修改,否则会导致对象移除不了.

* 1. HashSet的数据结构

HashSet底层用的就是HashMap

所以很多底层的结构我们学完HashMap之后,再看.

1. java.util.TreeSet

TreeSet中容纳的对象,必要要能够排序(排序方式由自己来定义).

两种排序的比较器:

* 1. java.util.Comparable

TreeSet使用无参数的构造函数,那么容纳的对象必须实现Comparable接口

* 1. java.util.Comparator

TreeSet构造的时候用Comparator作为构造函数的参数

比较方法如果返回0,那么对象不能重复加入.

* 1. 数据结构底层是TreeMap

关于Tree的结构我们可以不用深入了解,熟悉使用即可

## 10-1-2 深度学习ArrayList

## 10-1-3分析ArrayList底层源代码

## 10-1-4使用LinkedList开发贪吃蛇游戏

## 10-1-5 完成贪吃蛇游戏-分析LinkedList数据

## 10-1-6深入学习HashSet

## 10-1-7深入学习TreeSet

## 10-1-8深入学习HashMap

java.util.Map

--🡪java.util.HashMap

--🡪java.util.TreeMap

--🡪java.util.HashTable

5.java.util.HashMap(key/value)

5.1 基本操作HashMapDemo1

5.2 用HashMap的泛型实现基本操作,避免强制类型转换. HashMapDemo2

5.3 map.put(key,value)如果key相同,后面的value会覆盖前面的.

这个特点特别适用于做计数. HashMapDemo3

按照上面的案例计数后,再进行排序. HashMapDemo4

* 1. HashMap的数据结构(数组+链表的结构)

分析HashMap的源代码

a.HashMap包含了一个Entry(key,value,next,hash)的内部类

Key/value放入HashMap的时候都会被包装成Entry的对象

b.HashMap成员中就有Entry的数组,该数组的大小默认是16,

永远是2的次方数,如果自己给出的不是2的次方数,会转换成

大于并最接近自己给出的2的次方数.

放入对象时,put(key,value)时,其实级亅转换成Entry对象并放入数组中

c.put方法的实现

c1.根据key的hashCode进行hash运算(hash算法不用关心)得到值hash

c2.根据hash值去确定数组的位置

hash&(table.length-1) 等价与 hash%table.length length是2的次方数该公式成立,正好获取了数组的位置.

c3.如果这个位置没有元素存在,直接包装Entry的实例,并给数组元素赋值.

如果计算出的位置已经有元素存在,就要判断是否key相同,如果相同则覆盖,

并且要遍历整个链表.如果都不覆盖,插入到链表的头部.

如果计算出来的位置相同,这就是冲突率,

注意1:我们一定要减少冲突率,因为一旦放入链表中,以后总是要遍历链表,效率差,要尽量把元素放入直接放入数组,而非链表,根据实际要求重写hashCode和equals方法

注意2:底层是数组,尽量减少扩容,所以HashMap放入元素的时候也应该估算数组的大小,避免扩容操作

HashMap中有加载因子默认是0.75

比如:默认大小16,那么数组已经有12个元素,会自动扩容.

c4. get方法的实现

通过key查找元素的算法和放入是一样的.

所以一旦key放入HashMap就不应该修改跟hashCode和equals方法生成相关的属性的值了.否则就找不到了.

在HashSet中我们提过这个,那么我们看一下HashSet的源码.

HashSet中操作 就是HashMap

add方法的时候只用HashMap的key,value是死得,所以数据结构是一样的.

1. TreeMap(key,value)

Key必须要能够根据某种规则排序

通过两种比较器,TreeSet中用到就是TreeMap

1. Hashtable

7.1和HashMap相比类的层次结构不一样,当然都实现了Map接口,基本操作类似.

7.2HashMap中key/value都可以是null

而Hashtable不能是null,会抛出NullPointException的异常

7.3Hashtable是线程安全的(线程并发时效率低,但是数据一致性高)

HashMap是线程不安全的(线程并发时效率高,数据可能会存在安全问题)

7.4Hashtable还有一个子类

java.util.Properties类,这个类在将来用的比较多,用来加载资源文件

讲IO操作的时候会详细讲解

1. 其它

还有一些集合类,大家可以查阅帮助文档,最常用的的我们已经介绍

java.util.Collections类

该类封装了一些对集合的操作,都是静态方法.

面试有时也会问和Collection类有什么关系,其实没什么关系,介绍两个类的用途即可.Collection是一个接口,所有集合框架的父接口,下面有两个List接口,两个Set接口作为子类.而Collections类只是封装了一些关于集合的简单操作.CollectionsDemo

## 10-1-9 HashMap源代码分析

## 10-1-10 HashMap – HashSet源代码分析及

## 10-1-11深入学习HashTable

# JavaSE Core – IO专题

## 11-1-0如何使用程序对磁盘文件进行操作

## 11-1-1了解常用的编码

编码

java.io包

1. 编码问题
2. 一个字符串放入文本文件--🡪可以认为字符串的序列化

从文本文件读取字符串--🡪可以认为是字符串的反序列化

文本文件中的内容都是以字节的方式呈现的.做序列化的时候需要

把字符串转换成字节.

其实在字符串放入文本文件的时候需要把字符串转换成字节,

转换成字节的时候可以选择很多种编码.

以什么样的编码方式转换成字节,将来再转换成字符串也需要用该编码方式转换

1. 常用的编码

Gbk,utf-8,utf-16be

1. java.io.File表示文件或者目录

2.1文件常用的方法 FileDemo1.java

2.2文件的过滤器 FileDemo2.java

List(),listFiles(),list(FilenameFilter),listFile(FileFilter or FilenameFilter)

注意递归的问题

2.3文件的递归删除,目录必须为空才能被删除,所以删除一个目录先要删除目录中的内容

文件的读写操作

3.字节流--🡪读写以字节为单位

输入流:InputStream(读)

--🡪FileInputStream把文件作为字节流进行读操作

--🡪BufferedInputStream(缓冲区的,先读入缓冲区)

--🡪PipedInputStream管道流

--🡪DataInputStream 分析读写的原理

--🡪System.in 从键盘的读

输出流:OutputStream(写)

--🡪FileOutputStream把文件作为字节流进行写操作

--🡪BufferedOutputStream

--🡪PipedOutputStream

--🡪DataOutputStream分析读写的原理

--🡪System.out 🡪PrintStream

PipedOutputStream(PipedInputStream())

PipedInputStream(PipedOutputStream())

out写出的数据正好被in读到

4.字符流--🡪读写以字符为单位,只适合文本类的文件,不适合二进制文件

都是由字节流构造而来(适配器模式)

输入流Reader

--🡪InputStreamReader

等价于--🡪FileReader

--🡪BufferedReader 能够读字符了,希望能一次读一行,

就是一个字符串 readLine

输出流Writer

--🡪OutputStreamWriter

等价于--🡪FileWriter

--🡪BufferedWriter writer(String,int,int) newLine方法用于换行

--🡪PrintWriter

5.RandomAccessFile 随机访问文件流

6.java.util.Properties类的使用(并不是IO中的类)

实际使用中需要和IO集合在一起

资源文件 以properties作为文件的扩展名

文件的内容是以key=value的方式进行存放的

资源文件将来在设计中可以用来解耦+反射结合使用

Properties文件中文的问题:较新版本的eclipse都直接能转换

如果不能转换可以通过native2ascii –encoding gbk 原文件 新文件

7.序列化--🡪对象的序列化(对象进行存储or在网络中进行传递,该对象必须进行序列化)

7.1 java.io.ObjectOutputStream

java.io.ObjectInputStream

java.io.Serializable //该接口只是一个规范,没有任何方法

7.2 transient关键字

如果一个对象的属性用tranisent关键字来修饰,那么该属性不会做

Jvm默认的序列化工作

可以通过如下方式来进行自己的序列化和反序列化工作

Private void readObject(java.io.ObjectInputStream stream) throws IOException,ClassNotFoundException{}

private void writeObject(java.io.ObjectOutputStream stream) throws IOException{}

7.3集合中的类是如何序列化的,如ArrayList

ArrayList中有数组,且数组不一定放满元素,所以没有必要把整个数组序列化.只需要序列化有效元素,用了以上两个方法签名来进行序列化和反序列化的工作.这样的话,性能比较高.

7.4java实现接口,继承都有传递性,所以,如果父类实现了序列化接口(Serializable),那么其子类都可以序列化

需要注意的是:

如果父类是可序列化的,那么在反序列化子类的时候不会调用父类的构造函数.

如果父类是不可序列化的,而子类是可以序列化的,那么在子类进行反序列化时,会调用父类的构造函数.

SeriDemo2案例进行了测试

7.5java中有克隆的功能,Object类中已经提供了该功能

一个类的对象要进行克隆可以直接实现Cloneable接口

浅克隆or浅拷贝

深克隆or 深拷贝 通过序列化和反序列化的方式

这其实是一个设计模式--🡪原型模式

7.6和集合一起操作,如下的文本文件

1,zhangsan,28

2,lisi,29

3,wangwu,20

4,zhaoliu,33

5,zhangsan,28

6,zhangsan,28

7,lisi,29

8,zhangsan,28

9,tianqi,30

要求统计姓名出现的次数,并按照次数降序输出,

如果次数相同,按照姓名字母顺序升序.

思路:先一次读一行,然后进行字符串分割,把名字放入Map集合当中统计次数,

再用TreeSet排序

## 11-1-2 File类中的常用方法

## 11-1-3文件的过滤操作

## 11-1-4文件的过滤及递归删除

## 11-1-5以字节为单位的输入输出流1

## 11-1-6以字节为单位的输入输出流2

## 11-1-7以字节为单位的输入输出流3

## 11-1-8以字节为单位的输入输出流4

## 11-1-9以字符为单位的输入输出流1

## 11-1-10以字符为单位的输入输出流2

## 11-1-11 Properties类和IO结合使用

## 11-1-12 JavaIO的序列化问题

## 11-1-13 Java中浅拷贝(浅克隆)

## 11-1-14 Java中深拷贝(深克隆)

## 11-1-15案例-文件的读取(集合和IO结合使用)

# JavaSE Core – 多线程专题

## 12-1-0多线程解决梦茹心中的疑惑

## 12-1-1初步了解Java中的线程

进程:进程就是计算机的运行线索.

线程:进程的运行线索,计算机运行的最小单位.

1. Java中如何创建一个线程

1.1以实现了Runnable接口的类的实例做为创建Thread类对象的构造函数的参数.

主函数本身就是一个线程.

1.2直接创建类继承Thread,重写run方法.

两种方式都可以:建议使用第一种,更加面向对象.

一个Runnable的实例我们认为就是一个任务.

1.3直接用匿名类对象的方式

2.Java线程的生命周期

Newborn新生状态

Pause 阻塞状态

Runnable 可执行状态(Running,Ready)

Dead (不是stop--🡪死不能复生)

多线程的程序经常会每次运行结果可能不一样.(Why?)

Runnable(Running/Ready)跟这个状态有关 切换有关系

3.Java线程中常用的方法

3.1常用的方法 setName(),getName(),Thread.currentThread,优先级设置的方法等

3.2重要的方法

Thread.sleep()让线程睡,线程就会进入阻塞状态. ThreadDemo6

Interrupt()ThreadDemo5

Join()方法 一个线程等待另外一个线程运行结束ThreadDemo7

Thread.yield() 主动放弃CPU把机会让给别的线程,然后参与竞争.

Running----🡪ready ThreadDemo8

4.线程的互斥操作.--🡪主要解决的是:多个线程之间共享数据的问题.

4.1同步块 synchronized

锁(java中任何对象都是一把锁,有且只有一把钥匙) ThreadDemo9

4.2同步函数

也是由锁存在的 --🡪普通方法加同步,锁就是当前对象.

静态函数同步也是由锁的 --🡪锁是当前类的类类型

注意:多个线程共享数据要保证安全,一定要保证用的是同一个对象锁.

注意:死锁的问题,死锁程序就卡死,无法调试 ----🡪可以通过好的工具来检查死锁

要慎用synchronized关键字

注意:之前我们讲StringBuilder/StringBuffer

还有集合类:ArrayList/Vertor

4.3 Lock

synchronized 很繁琐 比较不好理解而且不面向对象

java5版本开始做了很好的改进,有了Lock对象

扩展内容:缓存操作,读写锁操作,见案例:CacheDemo OR 查看帮助文档.

5.线程的通讯

5.1通过管道流的方式进行简单的通讯(性能操作比较差,只能交互简单数据)

ThreadDemo13

5.2 Thread.yield()方式

生产者,消费者的模型.ThreadDemo14

效率也比较差.yield()是主动放弃了CPU,但又去参与竞争.

5.3 wait/notify

任何一个对象有拥有一个线程等待池,

挂在同一个对象的线程等待池中的线程,它们之间可以互相唤醒

wait()/notify()是属于Object类的.ThreadDemo15

wait方法的使用必须放入synchronized同步块中

先学会搭模型.

线程间的通讯的模型:ThreadDemo16

面试题:

先A线程运行10次,然后B线程运行20次

如此反复50次.

理解:A线程生产者 生产食物需要循环10次

B线程消费者 消费食物需要循环20次.(ThreadDemo15,16)

改进:A10次 B20次,C30次 如此50次(ThreadDemo17,18)

5.4 java5中引入了新的线程通讯方式(了解)

用的是锁机制.

Java5开始引入的线程机制java.util.concurrent

java.util.concurrent.locks

6.线程范围内数据共享.

6.1 HashMap<Thread,Object>

6.2 java中提供了ThreadLocal这个类,已经完成了类似的功能,

我们可以直接使用. ThreadDemo20

6.3 写一个类,使得该类在创建对象时,创建完之后,

直接就是同一个线程,同一个对象,不同线程,对象不同.(ThreadDemo21)

不管是单例模式还是这种情况,我们都使得构造函数private,这种情况

对反射来说是无效的.

7.java线程池 --🡪java5引入的.(了解)

线程并不是越多越好,如果无限制的创建线程,那么线程的创建,销毁等都有很大的

消耗.

希望不管执行多少任务,都用固定的线程数来执行.

7.1 如何创建线程池. ThreadDemo23

Executors类

newFixedThreadPool()

newSingleThreadExecutor()

newCachedThreadPool()

ExecutorService

ScheduledExecutorService

8.Callable & Future java5开始才有--🡪了解

## 12-1-2多线程下的程序运行及常用的线程

## 12-1-3常用的线程方法及线程的互斥操作

## 12-1-4解决多个线程间的共享数据问题

## 12-1-5 Lock对象

## 12-1-6线程之间的通讯1

## 12-1-7线程之间的通讯2

## 12-1-8强化学习线程间的通讯

## 12-1-9线程范围内数据共享

## 12-1-10 Java线程池操作

## 12-1-11 Callable&Future

# JavaSE Core – 网络编程专题

## 13-1-0和梦茹一起实现网络通讯

## 13-1-1客户端及服务器端的搭建

Java网络编程 java.net

1. java.net.InetAddress类
2. java.net.ServerSocket

java.net.Socket

服务器端

端口(port):计算机上的服务都会运行在一个端口上,端口用整数表示,

计算机最多有65536个端口,1024以下的基本被占用,我们

使用1024以上的.

在我们的计算机上开启一个服务,需要占用一个端口,如何开启一个服务?

在java中使用ServerSocket类即可.

ServerSocket s = new ServerSocket(port);

Socket socket = s.accept();

客户端

Socket socket = new Socket(ip,port);

Socket.getInputStream()

Socket.getOutputStream()

1. 聊天室(群聊,私聊的功能)

客户端需要做什么?

1. 随时要接收从服务器端发送过来的数据,读操作.
2. 随时从键盘读数据,发送给服务器.

客户端连接上服务器之后,需要开启两个线程来执行a,b.

聊天时.私聊的时候,ip/内容

群聊的时候,直接内容

服务器端需要做什么?

1. 随时需要读取客户端发来的数据,然后再往外写.
2. 服务器要能够保存产生的所有的Socket对象,能够遍历能够查找.

客户端连接上服务器之后,需要开启两个线程来执行a,b.

1. 在网络中传输对象.(如果一个对象要在网络间传递,必须要实现序列化接口)

## 13-1-2客户端和服务器之间的数据传递

## 13-1-3网络聊天室1

## 13-1-4网络聊天室2

## 13-1-5网络聊天室3

# JavaSE Core – 用户图形化界面

## 14-1-0抛弃控制台实现图形界面

## 14-1-1常用的简单组件和布局

GUI

java.awt.\*;//创建用户界面和绘制图形图像的所有类.

java.awt.event.\*;//事件

javax.swing.\*;// 创建用户界面的轻量级组件(不建议使用awt包下的组件,

swing包下的组件大部分都继承了awt包下的组件.)

java.awt.geom.\*;//2D几何图形

1. 常用的简单组件和布局

javax.swing.JFrame

javax.swing.JButton

窗体上可以做布局.FlowLayout水平布局

从上边的中间开始放

BorderLayout边界布局

东南西北中

GridLayout网格布局

x\*y的网格

javax.swing.JPanel --🡪面板

1. 面板比窗体提供了更丰富的操作.(比如画图等)
2. 面板是没有边界的窗体(没有最大最小化等)
3. 习惯是在窗体上布局面板,面板布局组件.
4. java事件机制

java中已经给所有的事件都分类了(和异常的原理很像)一个事件一旦产生,就产生了

事件类的对象.

给某个组件加事件操作就是要给一个组件加监听器.

如何加监听器

事件类型 监听器类型 加监听器的方式

ActionEvent ActionListener addActionListener(ActionListener)

XXXEvent XXXListener addXXXListener(XXXListener)

监听器都是接口,实现接口的方法,就是事件要做的事情.

2.1按钮的事件 ActionEvent

2.2鼠标的事件 MouseEvent

MouseListener addMouseListener 五个事件

MouseMotionListener addMouseMotionListener

2.3键盘事件

KeyEvent KeyListener addKeyListener

给组件加键盘事件指的是当组件获得焦点时操作键盘产生事件

1. 面板中的绘图操作

3.1 paintComponent重写该方法可以进行画图的操作,

该方法显示时会自动调用一次.

* 1. 如果要调用paintComponent只需要调用repaint即可.
  2. 和事件结合使用.

鼠标事件的案例

键盘事件的案例

## 14-1-2 Java事件机制

## 14-1-3 Java常用事件

## 14-1-4面板中的画图操作

## 14-1-5画图操作和事件相结合

# 阶段性项目-贪吃蛇游戏

## 贪吃蛇游戏-类设计

## 贪吃蛇游戏-控制器设计

## 贪吃蛇游戏-蛇身的实现

## 贪吃蛇游戏-实现蛇的移动

## 贪吃蛇游戏-实现障碍物

## 阶段性项目-完成游戏开发

# JavaSE Core – JDK新特性

## 15-1-1 JavaSE Core – JDK新特性之静态导入

Jdk新特性(java5.0后引入的)

1. 静态导入

导入类的静态成员,访问时就不需要类名.操作 用的比较少.

1. 自动装箱,拆箱

针对的是基本数据类型和封装类

基本的数据类型可以直接转换成封装类的实例--🡪装箱

封装类的实例可以直接转换成基本的数据类型--🡪拆箱

1. 可变参数

注意1:

int add(int[] a)

int add(int … a) jvm认为这两个是同一个方法.

注意2:一个函数的参数只能有一个可变参数,并且放在最后一个.

注意3:可变参数的函数可以跟其他函数构成重载.

1. 增强for循环(我们已经一直在用)

数组

集合

1. 枚举类型

如果一个类的对象是固定的,并且可列出的,并且大家都知道的

比如:季节类--🡪4个对象春,夏,秋,冬

星期类--🡪只有7个对象,周一到周日.

自己写一个星期类应该如何设计 WeekDay.java--🡪模拟枚举

enum 关键字 地位等级与类

1. 注解(原数据--🡪修饰数据的数据)
2. 泛型

泛型是提供给javac编译器使用的,可以限定集合中的输入类型.

让编译器挡住源程序中的非法输入.

编译器编译来类型说明的集合时会去掉”类型”信息,使程序运行效率不收影响.对于参数化的泛型类型getClass()方法的返回值和原始类型完全一样.由于编译生产的字节码会去掉泛型的类型信息,只要能跳过编译器.就可以往某个泛型集合中加入其它类型的数据.例如:用反射得到集合,再调用其add方法即可.

ArrayList<E>类定义和ArrayList<Integer>类引用中涉及术语如下:

ArrayList<E>泛型类型

ArrayList<E>中的E成为类型变量或者类型参数.

ArrayList<Integer>成为参数化的类型

ArrayList<Integer>中的Integer成为类型参数的实例,或者实际类型参数

ArrayList<Integer>中的<>念typeOf

参数化类型与原始类型的兼容性:

参数化类型可以引用一个原始类型.

Collection<String> c = new Vector();//会有警告

原始类型可以引用一个参数化类型的对象.

Collection c = new Vector<String>()//会有警告

参数化类型不考虑类型参数的继承关系

Vector<String> v = new Verctor<Object>();//错

Vector<Object> v = new Vector<String>();//也错

在创建数组实例时,数组的元素不能使用参数化的类型

Vector<String>[] v = new Vector<String>[10];//错

Vector v1 = new Vector<String>();

Vector<Object> v = v1;

泛型中的?通配符

问题:定义一个方法,该方法用于打印出任意参数化类型的集合(Collection)中的所有数据,该方法如何定义.

使用?通配符可以引用其他各种参数化的类型,?通配符定义的变量主要用作引用,可以调用与参数化无关的方法,不能调用与参数化有关的方法.

泛型中的?通配符的扩展

限定通配符的上边界

正确 Vector<? extends Number> x = new Vector<Integer>();

错误Vector<? extends Number> x = new Vector<String>();

限定通配符的下边界

正确 Vector<? super Integer> x = new Vector< Number>();

错误Vector<? super Integer> x = new Vector<Byte>();

限定通配符总是包括自己.

1. Java线程框架--🡪java分布式课程中讲解

## 15-1-2 JavaSE Core – JDK新特性之可变参数

## 15-1-3 JavaSE Core – JDK新特性之枚举

## 15-1-4 JavaSE Core – JDK新特性之注解

## 15-1-5 JavaSE Core – JDK新特性之注解

## 15-1-6 JavaSE Core – JDK新特性之泛型

# JavaSE阶段项目-我赢在线考试系统

## 登陆界面的实现

## 功能选择界面的实现

## 系统答题界面的实现

## 使用MVC管理各个界面

## 模拟数据层-装载用户信息

## 模拟数据层-装载题库信息

## 实现用户登陆

## 完善用户登录的操作

## 完成开始考试功能

## 显示考题相关信息

## 完成题目的显示功能

## 完善答题界面的显示

## 完成交卷功能

## 完成倒计时功能

## 完成网络版考试系统1

## 完成网络版考试系统2

## 完成网络版考试系统3

## 完成网络版考试系统4

## 完成网络版考试系统5

# Oracle数据库专题

## 16-1-1 Oracle下载安装登陆

## 16-1-2 sqlplus常用命令

## 16-1-3基本的SQL查询语句

## 16-1-4限制条件的查询

## 16-1-5限制条件的查询的练习

## 16-1-6函数查询1

## 16-1-7函数查询2

## 16-1-8函数查询3

## 16-1-9函数查询4

## 16-1-10函数查询5

## 16-1-11数据表之间的关联1

## 16-1-12数据表之间的关联2

## 16-1-13数据表之间的关联3

## 16-1-14数据表之间的关联4及表的创建

## 16-1-15数据表的创建2

## 16-1-16数据表的创建3

## 16-1-17数据表的相关操作

## 16-1-18序列的相关操作

## 16-1-19索引的相关操作

# JDBC专题

## 17-1-1 Java连接数据库

## 17-1-2将常用的方法-资源包装

## 17-1-3将针对数据表的操作进行包装

## 17-1-4使用代码对数据库进行操作1

## 17-1-5使用代码对数据库进行操作2

## 17-1-6解决SQL注入的问题

## 17-1-7分页操作

## 17-1-8事物操作1

## 17-1-9事物操作2

## 17-1-10事物操作3

## 17-1-11代理模式

## 17-1-12使用代理模式完成实际业务1

## 17-1-13使用代理模式完成实际业务2

## 17-1-14抽象DAO操作

## 17-1-15 JDBC操作补充1

## 17-1-16 JDBC操作补充2

## 17-1-17 JDBC操作补充3

# XML专题

## 18-1-1 XML初步介绍

## 18-1-2如何使用dom4j解析XML文件

## 18-1-3完成XML文件的创建

## 18-1-4将XML文件中的数据导入到数据库中

# MySQL数据库专题

## 19-1-1 MySQL数据库介绍及操作使用

## 19-1-2 MySQL在操作中与Oracle的不同点

# JDBC阶段实战项目

## BBS项目演示机数据表设计

## BBS论坛数据表设计2

## BBS论坛数据表设计3

## BBS论坛数据表设计4

## 提取数据库通用方法

## 实现项目中的业务操作1

## 实现项目中的业务操作2

## 实现项目中的业务操作3

## 实现项目中的业务操作4

## 实现项目中的业务操作5

# 网页前端– HTML专题

## 20-1-1初步了解HTML

## 20-1-2 HTML常用标签1

## 20-1-3 HTML常用标签2

## 20-1-4 HTML常用标签3

## 20-1-5 HTML常用标签4

# 网页前端– CSS专题

## 21-1-1 CSS样式的初步介绍

## 21-1-2 CSS样式表的使用

## 21-1-3 常用的样式属性

## 21-1-4企业实战案例-数据挖掘系统的页面

## 21-1-5企业实战案例-数据挖掘系统的页面

## 21-1-6企业实战案例-数据挖掘系统的页面

# 网页前端– JavaScript专题

## 22-1-1 JavaScript基础介绍

## 22-1-2 JavaScript基础语法

## 22-1-3 JavaScript常用函数

## 22-1-4 JS中的window对象

## 22-1-5 JS中的Date对象

## 22-1-6 BOM编程1

## 22-1-7 BOM编程2

## 22-1-8 BOM编程3

## 22-1-9 BOM编程4

## 22-1-10 BOM编程5

## 22-1-11 DOM编程

## 22-1-12事件操作1

## 22-1-13事件操作2

## 22-1-14级联下拉框的实现

## 22-1-15数据的验证操作1

## 22-1-16数据的验证操作2

## 22-1-17数据的验证操作3

## 22-1-18数据的验证操作4

## 22-2-1 JQuery基本介绍

## 22-2-2案例-可编辑表格1

## 22-2-3案例-可编辑表格2

## 22-2-4案例-窗口的淡入淡出效果

## 22-2-5案例-菜单的关闭和隐藏1

## 22-2-6案例-菜单的关闭和隐藏2

## 22-2-7案例-显示在中间的窗口1

## 22-2-8案例-显示在中间的窗口2

## 22-2-9案例-显示在中间的窗口3

# Servlet专题

## 23-1-1 Servlet专题 – tomcat安装配置

## 23-1-2 Servlet专题 – 手动开发第一个Web项

## 23-1-3 Servlet专题 – 配置servlet

## 23-1-4 Servlet专题 – eclipse中创建Web项目

## 23-1-5 Servlet专题 – servlet生命周期

## 23-1-6 Servlet专题 – servlet获取配置信息

## 23-1-7 Servlet专题 – 请求数据

## 23-1-8 Servlet专题 – 中文乱码处理

## 23-1-9 Servlet专题 – 重定向、分页(上)

## 23-1-10 Servlet专题 – 分页(下)

## 23-1-11 Servlet专题 – 会话管理

## 23-1-12 Servlet专题 – 会话管理

## 23-1-13 Servlet专题 – 会话管理之请求范围

## 23-1-14 Servlet专题 – 避免页面直接被访问

## 23-1-15 Servlet专题 – Session使用范围

## 23-1-16 Servlet专题 – Session原理

## 23-1-17 Servlet专题 – Session创建、销毁

## 23-1-18 Servlet专题 – Session总结

## 23-1-19 Servlet专题 – 过滤器 - Cookie(下)

# JSP专题

## 24-1-1 JSP专题1

## 24-1-2 JSP专题2

## 24-1-3 JSP专题3

## 24-1-4 JSP专题4

## 24-1-5 JSP专题5

## 24-1-6 JSP专题6

## 24-1-7 JSP专题7

## 24-1-8 JSP专题8

## 24-1-9 JSP专题9

## 24-1-10 JSP专题10

## 24-1-11 JSP专题11

## 24-1-12 JSP专题12

## 24-1-13 JSP专题13

## 24-1-14 JSP专题14

## 24-1-15 JSP专题15

# Ajax专题

## 25-1-1 Ajax专题1

## 25-1-2 Ajax专题2

## 25-1-3 Ajax专题3

## 25-1-4 Ajax专题4

## 25-1-5 Ajax专题5

## 25-1-6 Ajax专题6

## 25-1-7 Ajax专题7

## 25-1-8 Ajax专题8

## 25-1-9 Ajax专题9

## 25-1-10 Ajax专题10

## 25-1-11 Ajax专题11

## 25-1-12 Ajax专题12

# 企业实战项目-BBS论坛系统

## 26-1-1 BBS论坛用户登陆1

## 26-1-2 BBS论坛用户登陆2

## 26-1-3 BBS论坛用户登陆3

## 26-1-4 BBS论坛用户登陆4

## 26-1-5 BBS论坛用户登陆5

## 26-2-1 BBS论坛用户注册1

## 26-2-2 BBS论坛用户注册2

## 26-3-1 BBS论坛用户自动登陆1

## 26-3-2 BBS论坛用户自动登陆2

## 26-3-3 BBS论坛用户自动登陆3

## 26-4-1 BBS论坛版面管理模块1

## 26-4-2 BBS论坛版面管理模块2

## 26-4-3 BBS论坛版面管理模块3

## 26-4-4 BBS论坛版面管理模块4

## 26-4-5 BBS论坛版面管理模块5

## 26-4-6 BBS论坛版面管理模块6

## 26-4-7 BBS论坛版面管理模块7

## 26-4-8 BBS论坛版面管理模块8

## 26-5-1 BBS论坛角色管理模块1

## 26-5-2 BBS论坛角色管理模块2

## 26-5-3 BBS论坛角色管理模块3

## 26-5-4 BBS论坛角色管理模块4

## 26-5-5 BBS论坛角色管理模块5

## 26-5-6 BBS论坛角色管理模块6

## 26-6-1 BBS论坛会员管理模块1

## 26-6-2 BBS论坛会员管理模块2

## 26-6-3 BBS论坛会员管理模块3

## 26-6-4 BBS论坛会员管理模块4

## 26-6-5 BBS论坛会员管理模块5

## 26-6-6 BBS论坛会员管理模块6

## 26-6-7 BBS论坛会员管理模块7

## 26-6-8 BBS论坛会员管理模块8

## 26-7-1 BBS论坛权限控制1

## 26-7-2 BBS论坛权限控制2

## 26-8-1 BBS论坛论坛帖管理1

## 26-8-2 BBS论坛论坛帖管理2

## 26-8-3 BBS论坛论坛帖管理3

## 26-8-4 BBS论坛论坛帖管理4

## 26-8-5 BBS论坛论坛帖管理5

## 26-8-6 BBS论坛论坛帖管理6

## 26-8-7 BBS论坛论坛帖管理7

## 26-8-8 BBS论坛论坛帖管理8

# Struts2框架专题

## 27-1-1 Struts2框架专题1

## 27-1-2 Struts2框架专题2

## 27-1-3 Struts2框架专题3

## 27-1-4 Struts2框架专题4

## 27-1-5 Struts2框架专题5

## 27-1-6 Struts2框架专题6

## 27-1-7 Struts2框架专题7

## 27-1-8 Struts2框架专题8

## 27-1-9 Struts2框架专题9

## 27-1-10 Struts2框架专题10

## 27-1-11 Struts2框架专题11

## 27-1-12 Struts2框架专题12

## 27-1-13 Struts2框架专题13

## 27-1-14 Struts2框架专题14

## 27-1-15 Struts2框架专题15

## 27-1-16 Struts2框架专题16

## 27-1-17 Struts2框架专题17

## 27-1-18 Struts2框架专题18

## 27-1-19 Struts2框架专题19

## 27-1-20 Struts2框架专题20

## 27-1-21 Struts2框架专题21

## 27-1-22 Struts2框架专题22

# Hibernate框架专题

## 28-1-1 Hibernate框架专题1

## 28-1-2 Hibernate框架专题2

## 28-1-3 Hibernate框架专题3

## 28-1-4 Hibernate框架专题4

## 28-1-5 Hibernate框架专题5

## 28-1-6 Hibernate框架专题6

## 28-1-7 Hibernate框架专题7

## 28-1-8 Hibernate框架专题8

## 28-1-9 Hibernate框架专题9

## 28-1-10 Hibernate框架专题10

## 28-1-11 Hibernate框架专题11

## 28-1-12 Hibernate框架专题12

## 28-1-13 Hibernate框架专题13

## 28-1-14 Hibernate框架专题14

## 28-1-15 Hibernate框架专题15

## 28-1-16 Hibernate框架专题16

## 28-1-17 Hibernate框架专题17

## 28-1-18 Hibernate框架专题18

## 28-1-19 Hibernate框架专题19

## 28-1-20 Hibernate框架专题20

## 28-1-21 Hibernate框架专题21

## 28-1-22 Hibernate框架专题22

# Spring框架专题

## 29-1-1 Spring框架专题1

## 29-1-2 Spring框架专题2

## 29-1-3 Spring框架专题3

## 29-1-4 Spring框架专题4

## 29-1-5 Spring框架专题5

## 29-1-6 Spring框架专题6

## 29-1-7 Spring框架专题7

## 29-1-8 Spring框架专题8

## 29-1-9 Spring框架专题9

## 29-1-10 Spring框架专题10

## 29-1-11 Spring框架专题11

## 29-1-12 Spring框架专题12

## 29-1-13 Spring框架专题13

## 29-1-14 Spring框架专题14

## 29-1-15 Spring框架专题15

## 29-1-16 Spring框架专题16

# 能力提高

## 30-1-1能力提高- 网站架构

## 30-1-2能力提高- 分布式数据存储架构-垂直

## 30-1-3能力提高- 水平切分-数据切分及整合

## 30-1-4能力提高- NoSql

## 30-1-5能力提高- NoSql之Memcached Java

## 30-1-6能力提高- NoSql之Memcached集群

## 30-1-7能力提高- NoSql之redis安装及常用

## 30-1-8能力提高- NoSql之redis详解

## 30-1-9能力提高- 异构数据源下数据层设计

## 30-1-10能力提高- 异构数据源下设计问题

## 30-1-11能力提高- 设计原则1

## 30-1-12能力提高- 设计原则2

## 30-1-13能力提高- 创建型设计模式之工厂模式1

## 30-1-14能力提高- 创建型设计模式之工厂模式2

## 30-1-15能力提高- 结构型设计模式之适配器

## 30-1-16能力提高- 结构型设计模式之装饰模式

## 30-1-17能力提高- 结构型设计模式之享元模式

# 彩票系统

## 1-1-1彩票系统的简介

## 1-1-2搭建环境

## 1-1-3用户注册

## 1-1-4 mvc三层架构

## 1-1-5整合hibernate

## 1-1-6登陆界面的实现

## 1-1-7修改密码

## 1-1-8退出功能的实现

## 1-1-9查询会员操作

## 1-1-10分页查询会员

## 1-1-11对会员的删除操作

## 1-1-12会员管理模块总结

## 1-1-13用户留言

## 1-1-14 JavaBean实体类

## 1-1-15添加留言

## 1-1-16分页查询留言

## 1-1-17删除留言

## 1-1-18全选功能

## 1-1-19全部删除

## 1-1-20查询消息

## 1-1-21通过会员名称查询会员信息

## 1-1-22查询所有留言信息

## 1-1-23 where语句

## 1-1-24代码优化

## 1-1-25代码封装

## 1-1-26 Dom4j解析

## 1-1-27 Map集合封装解析的数据

## 1-1-28控制层和业务层的解耦

## 1-1-29处理中文乱码问题

## 1-1-30提高代码的可维护性

## 1-1-31公告管理

## 1-1-32添加公告管理

## 1-1-33显示公告管理

## 1-1-34删除公告管理

## 1-1-35查看详细公告

## 1-1-36添加彩票

## 1-1-37查询彩票

## 1-1-38图片上传(上)

## 1-1-39图片上传(下)

## 1-1-40将彩票信息显示在前台页面

## 1-1-41随机选择双色球

## 1-1-42购买彩票

## 1-1-43显示购买信息

## 1-1-44彩票订单

## 1-1-45彩票系统总结