|  |
| --- |
| 易通JAVASE历险记 |
| JAVASE笔记 |



2016-5-16

目录

[Java语言的认识 0](#_Toc451206075)

[Java语言的历史 0](#_Toc451206076)

[Java语言的特性 1](#_Toc451206077)

[Java语言的应用 2](#_Toc451206078)

[Java开发环境搭建 3](#_Toc451206079)

[java关键字、标识符、变量、数据类型等 5](#_Toc451206080)

[关键字 5](#_Toc451206081)

[标识符 6](#_Toc451206082)

[Java中的名称命名规范： 7](#_Toc451206083)

[变量 7](#_Toc451206084)

[数据类型 9](#_Toc451206085)

[浮点型：float、double 9](#_Toc451206086)

[字符类型：char 9](#_Toc451206087)

[ASCII码 10](#_Toc451206088)

[Unicode 编码 11](#_Toc451206089)

[基本数据类型转换 12](#_Toc451206090)

[强制类型转换 12](#_Toc451206091)

[String类 13](#_Toc451206092)

[运算符 13](#_Toc451206093)

[算术运算符 13](#_Toc451206094)

[赋值运算符 15](#_Toc451206095)

[比较运算符（关系运算符） 16](#_Toc451206096)

[逻辑运算符 17](#_Toc451206097)

[位运算符 18](#_Toc451206098)

[三元运算符 18](#_Toc451206099)

[程序流程控制 19](#_Toc451206100)

[顺序结构 19](#_Toc451206101)

[分支结构 19](#_Toc451206102)

[if语句三种格式： 19](#_Toc451206103)

[循环结构 20](#_Toc451206104)

[特殊流程控制语句 21](#_Toc451206105)

[数组 22](#_Toc451206106)

[一维数组 23](#_Toc451206107)

[一维数组的初始化 23](#_Toc451206108)

[数组元素的引用 23](#_Toc451206109)

[多维数组之二维数组 24](#_Toc451206110)

[多维数组 25](#_Toc451206111)

[数据结构-排序方法 27](#_Toc451206112)

[数据结构 27](#_Toc451206113)

[逻辑结构 27](#_Toc451206114)

[OOP之面向对象基础 28](#_Toc451206115)

[面向对象的编程思想 29](#_Toc451206116)

[C语言的编程方式 29](#_Toc451206117)

[SP和OOP对比 29](#_Toc451206118)

[面向对象的三大原则 30](#_Toc451206119)

[对象 30](#_Toc451206120)

[类 31](#_Toc451206121)

[属性 31](#_Toc451206122)

[方法 31](#_Toc451206123)

[类和对象的区别 31](#_Toc451206124)

[类与结构 32](#_Toc451206125)

[多态 32](#_Toc451206126)

[抽象 32](#_Toc451206127)

[定义类的语法 35](#_Toc451206128)

[创建对象的语法 36](#_Toc451206129)

[成员运算符 36](#_Toc451206130)

[访问权限：public和private 36](#_Toc451206131)

[对象初始化 38](#_Toc451206132)

[构造方法 39](#_Toc451206133)

[总结 40](#_Toc451206134)

[OOP之方法重载 41](#_Toc451206135)

[回顾函数 41](#_Toc451206136)

[方法重载的三大原则 43](#_Toc451206137)

[方法重载之一 44](#_Toc451206138)

[方法重载之二 45](#_Toc451206139)

[重载构造方法 45](#_Toc451206140)

[对象在内存中的存放形式 48](#_Toc451206141)

[This关键字 48](#_Toc451206142)

[静态成员变量 49](#_Toc451206143)

# Java语言的认识

## Java语言的历史

Java是由Sun Microsystems公司推出的Java面向对象程序设计语言（以下简称Java语言）和Java平台的总称。由James Gosling和同事们共同研发，并在1995年正式推出。Java最初被称为Oak，是1991年为消费类电子产品的嵌入式芯片而设计的。1995年更名为Java，并重新设计用于开发Internet应用程序。用Java实现的HotJava浏览器（支持Java applet）显示了Java的魅力：跨平台、动态Web、Internet计算。从此，Java被广泛接受并推动了Web的迅速发展，常用的浏览器均支持Javaapplet。另一方面，Java技术也不断更新。Java自面世后就非常流行，发展迅速，对C++语言形成有力冲击。在全球云计算和移动互联网的产业环境下，Java更具备了显著优势和广阔前景。2009年Oracle公司收购Sun Microsystems。

它最初被命名为Oak，目标设定在家用电器等小型系统的编程语言，来解决诸如电视机、电话、闹钟、烤面包机等家用电器的控制和通讯问题。由于这些智能化家电的市场需求没有预期的高，Sun放弃了该项计划。就在Oak几近失败之时，随着互联网的发展，Sun看到了Oak在计算机网络上的广阔应用前景，于是改造了Oak，以“Java”的名称正式发布。

## Java语言的特性

**1.完全面向对象：**

**所有的设计都必须在类中实现，一个Java程序就是多个类的集合。**

**2.安全性：**

**良好的安全模型，摒弃了其它语言的不安全成分。如字节码验证器对字节码的验证，放止其它非法代码侵入；内存空间不够、程序异常终止等的处理。**

**3.分布式：**

**直接支持分布式的网络应用，可实现高效的网络编程。**

**4.多线程：**

**多线程机制使得程序能够并行处理多项任务。例如一个线程负责数据的检索，一个线程与用户进行交互，两个线程并行执行。**

**5.平台无关性：**

**借助虚拟机，程序不经修改即可在不同硬件或软件平台上运行。**

**6.动态内存管理机制：**

**采用自动垃圾回收机制进行内存管理。可以自动、安全回收不再使用的内存块，减少了内存管理方面出错的可能性。**

Java分为**三**个体系，分别为

**Java SE（J2SE，Java2 Platform Standard Edition，标准版）**

**Java EE（J2EE，Java 2 Platform, Enterprise Edition，企业版）**

**Java ME（J2ME，Java 2 Platform Micro Edition，微型版）**

Java的编译与运行

## Java语言的应用

* **网站**：主要使用JAVA EE开发，阿里巴巴、淘宝、支付宝，一号店，网易、搜狐、大众点评网，驴妈妈，人人网，去哪儿网,网银，电商 ，YY PC版等 ...
* **大型企业级应用**：主要使用**JAVA EE，**比如企业管理系统
* **移动互联网：** Android APP开发应用；

**最典型应用:**

<1> 有关通信及网络的企业：移动、联通、电信、网通主要的信息化

<2> 有关金融行的大型企业，所有的银行、证券公司，比如工商银行、建设银行

<3> 大型管理系统，如：供应链、客户管理系统……

<4> 电子政务，主要使用JAVA EE，相关的政府部门绝大多数的信息化系统

<5> 游戏，很多手机游戏都是用JAVA开发的。

<6> 嵌入式设备及消费类电子产品，主要用 JAVA ME,无线手持设备、通信终端、医疗设备、信息家电（如数字电视、机顶盒、电冰箱）、汽车电子设备等是比较热门的Java应用领域

## Java开发环境搭建

环境变量其中一种**用户变量**仅本用户适用，另一种即**系统变量**整个系统的用户都适用,两者都可以在使用应用程序时提供快捷。

简单的说就是，如果设置系统变量和用户变量，都叫做设置环境变量，设置系统变量时，该系统的所有帐号的用户都可以使用，但是设置用户变量时，其他的帐号登陆时就不一定可以使用。

什么是JDK？JVM？JRE

JDK全称**Java Se Development Kit**,即Java标准开发包，是sun公司提供一套用于开发Java应用程序开发包，它提供了编译、运行Java程序所需的工具和资源，包括Java编译器，Java运行时候的环境，以及常用的Java类库等。

**JRE是包含JVM**。JVM是运行Java核心的虚拟机。而运行Java核心程序不仅需要核心虚拟机，还需要其它的类加载器，字节码校验器以及大量的基础类库，故**JRE除了JVM外，还包含运行Java程序的其他环境**。

**Java环境搭建**

1.下载JDK www.java.com

2.安装JDK

3.配置环境变量

采用命令行 ----->**cmd**

**cd 目录名称，进入目录; cd.. 退回上一级目录**

**dir 显示文件或目录; del 文件名，删除文件**

1.配置JAVA\_HOME：

变量名：JAVA\_HOME

变量值：**D：\jdk1.7** (**你的jdk安装的目录文件夹名称**)

2.配置PATH：

变量名：PATH

变量值：**;%JAVA\_HOME%\bin** (**前面分号不要忘记！**)

3.配置CLASSPATH：

变量名：CLASSPATH

变量值：**.**

# java关键字、标识符、变量、数据类型等

## 关键字

* **关键字的定义和特点**
  + 定义：**被Java语言赋予了特殊含义，用做专门用途的字符串（单词）**
  + 特点：**关键字中所有字母都为小写**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 用于定义数据类型的关键字 | | | | |
| class | interface | enum | byte | short |
| int | long | float | double | char |
| boolean | void |  |  |  |
| 用于定义数据类型值的关键字 | | | | |
| true | false | null |  |  |
| 用于定义流程控制的关键字 | | | | |
| if | else | switch | case | default |
| while | do | for | break | continue |
| return |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 用于定义访问权限修饰符的关键字 | | | | |
| private | protected | public |  |  |
| 用于定义类，函数，变量修饰符的关键字 | | | | |
| abstract | final | static | synchronized |  |
| 用于定义类与类之间关系的关键字 | | | | |
| extends | implements |  |  |  |
| 用于定义建立实例及引用实例，判断实例的关键字 | | | | |
| new | this | super | instanceof |  |
| 用于异常处理的关键字 | | | | |
| try | catch | finally | throw | throws |
| 用于包的关键字 | | | | |
| package | import |  |  |  |
| 其他修饰符关键字 | | | | |
| native | strictfp | transient | volatile | assert |

保留字：

Java保留字：现有Java版本尚未使用，但以后版本可能会作为关键字使用。自己命名标记符时要避免使用这些保留字   
byValue、cast、future、 generic、 inner、 operator、 outer、 rest、 var 、 goto 、const

## 标识符

* 标识符：
  + Java 对各种**变量**、**方法**和**类**等要素命名时使用的字符序列称为标识符
  + **凡是自己可以起名字的地方都叫标识符**。
* 定义合法标识符规则：
  + **由26个英文字母大小写，0-9 ，\_或 $ 组成**
  + **数字不可以开头。**
  + **不可以使用关键字和保留字，但能包含关键字和保留字。**
  + **Java中严格区分大小写，长度无限制。**
  + **标识符不能包含空格。**
* 注意：在起名字时，为了提高阅读性，要尽量有意义，“见名知意”。

### Java中的名称命名规范：

**包名**：多单词组成时所有字母都**小写**：xxxyyyzzz

**类名、接口名**：多单词组成时，所有单词的首字母大写：XxxYyyZzz

**变量名、方法名**：多单词组成时，

第一个单词首字母小写，

第二个单词开始每个单词首字母大写：xxxYyyZzz

**常量名**：所有字母都大写。多单词时每个单词用下划线连接：XXX\_YYY\_ZZZ

## 变量

变量的概念：

内存中的一个存储区域

该区域有自己的名称（变量名）和类型（数据类型）

Java中每个变量必须先声明，后使用

该区域的数据可以在同一类型范围内不断变化

使用变量注意：

变量的作用域：一对{ }之间有效

初始化值

定义变量的格式：数据类型 变量名 = 初始化值

变量是通过使用变量名来访问这块区域的

整数类

**(byte,short,int,long)**

变量分类：

数组**([ ])**

接口**(interface)**

类**(class)**

浮点类型**(float,double)**

布尔型**(boolean)**

字符型**(char)**

数值型

**引用**

**数据类型**

**基本**

**数据类型**

数据类型

字符串在这里

所有

变量

代码块局部变量（在代码块内定义）

方法局部变量（在方法内定义）

形参（方法签名中定义的变量）

类变量（以static修饰）

实例变量（不以static修饰）

**局部变量**

**成员变量**

在**方法体外，类体内**声明的变量成为**成员变量**；

在**方法内部**声明的变量为**局部变量**

### 数据类型

Java各整数类型有固定的表数范围和字段长度，不受具体OS的影响，以保证java程序的可移植性。

java的整型常量默认为 int 型，声明long型常量须后加‘l’或‘L’

|  |  |  |
| --- | --- | --- |
| ***类 型*** | ***占用存储空间*** | ***表数范围*** |
| ***byte*** | ***1字节=8bit*** | ***-128 ~ 127*** |
| ***short*** | ***2字节*** | ***-215 ~215-1*** |
| ***int*** | ***4字节*** | ***-231 ~ 231-1*** |
| ***long*** | ***8字节*** | ***-263 ~ 263-1*** |

### 浮点型：float、double

与整数类型类似，Java 浮点类型也有固定的表数范围和字段长度，不受具体OS的影响。

**Java 的浮点型常量默认为double型**，**声明float型常量，须后加‘f’或‘F’。**

十进制数形式：如：5.12 512.0f .512 (必须有小数点）

|  |  |  |
| --- | --- | --- |
| ***类 型*** | ***占用存储空间*** | ***表数范围*** |
| ***单精度float*** | ***4字节*** | ***-3.403E38 ~ 3.403E38*** |
| ***双精度double*** | ***8字节*** | ***-1.798E308 ~ 1.798E308*** |

### 字符类型：char

char 型数据用来表示通常意义上“字符”(**2字节**)

字符型常量的三种表现形式：

字符常量是用单引号(‘ ’)括起来的单个字符，涵盖世界上所有书面语的字符。例如：char c1 = 'a'; char c2 = '中'; char c3 = '9';

Java中还允许使用转义字符‘\’来将其后的字符转变为特殊字符型常量。例如：char c3 = ‘\n’; // '\n'表示换行符

直接使用 Unicode 值来表示字符型常量：‘\uXXXX’。其中，XXXX代表一个十六进制整数。如：\u000a 表示 \n。

char类型是可以进行运算的。因为它都对应有Unicode码

注意：\r\n表示回车换行

我们在平时使用电脑时，已经习惯了回车和换行一次搞定，敲一个回车键，即是回车，又是换行

### ASCII码

在计算机内部，所有数据都使用**二进制**表示。每一个二进制位（bit）有 0 和 1 两种状态，因此 8 个二进制位就可以组合出 **256 种**状态，这被称为一个字节（byte）。一个字节一共可以用来表示 256 种不同的状态，每一个状态对应一个符号，就是 256 个符号，从 0000000 到 11111111。

ASCII码：上个世纪60年代，美国制定了一套字符编码，对英语字符与二进制位之间的关系，做了统一规定。这被称为ASCII码。ASCII码一共规定了**128个**字符的编码，比如空格“SPACE”是32（二进制00100000），大写的字母A是65（二进制01000001）。这128个符号（包括**32**个不能打印出来的控制符号），只占用了一个字节的后面7位，最前面的1位统一规定为0。

A：65 a:97 0:48

### Unicode 编码

乱码：世界上存在着多种编码方式，同一个二进制数字可以被解释成不同的符号。因此，要想打开一个文本文件，就必须知道它的编码方式，否则用错误的编码方式解读，就会出现乱码。

Unicode：**一种编码，将世界上所有的符号都纳入其中。每一个符号都给予一个独一无二的编码，使用 Unicode 没有乱码的问题。**

#### Utf-8

UTF-8 是在互联网上使用最广的一种 **Unicode 的实现方式。**

UTF-8 是一种变长的编码方式。它可以使用 1-6 个字节表示一个符号，根据不同的符号而变化字节长度。

UTF-8的编码规则：

对于单字节的UTF-8编码，该字节的最高位为0，其余7位用来对字符进行编码（等同于ASCII码）。

对于多字节的UTF-8编码，如果编码包含 n 个字节，那么第一个字节的前 n 位为1，第一个字节的第 n+1 位为0，该字节的剩余各位用来对字符进行编码。在第一个字节之后的所有的字节，都是最高两位为"10"，其余6位用来对字符进行编码。

### 基本数据类型转换

自动类型转换：容量小的类型自动转换为容量大的数据类型。数据类型按容量大小排序为：

char-->int-->long-->float-->double

byte-->short-->int-->long-->float-->double

有多种类型的数据混合运算时，系统首先自动将所有数据转换成容量最大的那种数据类型，然后再进行计算。

byte,short,char之间不会相互转换，他们三者在计算时首先转换为int类型。

当把任何基本类型的值和字符串值进行连接运算时(+)，基本类型的值将自动转化为字符串类型。

### 强制类型转换

自动类型转换的逆过程，将容量大的数据类型转换为容量小的数据类型。使用时要加上强制转换符（()），但可能造成精度降低或溢出,格外要注意。

通常，字符串不能直接转换为基本类型，但通过基本类型对应的包装类则可以实现把字符串转换成基本类型。

如： String a = “43”; int i = Integer.parseInt(a);

boolean类型不可以转换为其它的数据类型。

//int a = Integer.parseInt("1010",2);//2进制转成10进制

String b = Integer.toString(8,2);//10进制转成2进制

### String类

值null可以赋值给任何引用类型（类、接口、数组）的变量，用以表示这个引用类型变量中保存的地址为空。

String类属于引用类型，可用null赋值。

String类是一个典型的不可变类，String对象创建出来就不可能被改变。创建出的字符串将存放在数据区，保证每个字符串常量只有一个，不会产生多个副本。

String s0 = "hello";

String s1 = "hello";

String s2 = "he" + "llo";

System.out.println(s0 ==s1);

System.out.println(s0 ==s2);

## 运算符

运算符是一种特殊的符号，用以表示**数据的运算、赋值和比较**等

### 算术运算符

|  |  |  |  |
| --- | --- | --- | --- |
| ***运算符*** | ***运算*** | ***范例*** | ***结果*** |
| ***+*** | ***正号*** | ***+3*** | ***3*** |
| ***-*** | ***负号*** | ***b=4; -b*** | ***-4*** |
| ***+*** | ***加*** | ***5+5*** | ***10*** |
| ***-*** | ***减*** | ***6-4*** | ***2*** |
| ***\**** | ***乘*** | ***3\*4*** | ***12*** |
| ***/*** | ***除*** | ***5/5*** | ***1*** |
| ***%*** | ***取模*** | ***7%5*** | ***2*** |
| ***++***  ***++*** | ***自增（前）：先运算后取值***  ***自增（后）：先取值后运算*** | ***a=2;b=++a;***  ***a=2;b=a++;*** | ***a=3;b=3***  ***a=3;b=2*** |
| ***- -***  ***- -*** | ***自减（前）：先运算后取值***  ***自减（后）：先取值后运算*** | ***a=2;b=- -a***  ***a=2;b=a- -*** | ***a=1;b=1***  ***a=1;b=2*** |
| ***+*** | ***字符串相加*** | ***“He”+”llo”*** | ***“Hello”*** |

除法的时候

整数相除在除不尽的时候，想要获取小数结果，必须一方转成float类型。

否则在计算的时候就会丢掉，造成精度丢失。(如文件上传)

int a = 1;

int b = 3;

float c = a/b;//c=0.0 12.0

System.out.println(a/b);

System.out.println(c);

int d = 100;float e = 5;

System.out.println(e/d);//0.05

### 赋值运算符

符号：=

当“=”两侧数据类型不一致时，可以使用自动类型转换或使用强制类型转换原则进行处理。

支持连续赋值。

扩展赋值运算符： +=, -=, \*=, /=, %=

short s = 3; s=s+2; ① s+=2; ② ①和②有什么区别？

1. 会报错；编译都通不过
2. 递增值为2；

**short** s=3;

s+=2;

s+=2;

System.***out***.println(s);//s=7

boolean b1 = false; //区分好==和=的区别。 双等号是比较，单等号是赋值

**boolean** b1 = **false**;

**if** (b1 = **true**){

System.***out***.println("结果为真");

}**else**{

System.***out***.println("结果为假");

}//结果为真

int i = 1;i \*= 0.1;

System.out.println(i);//0

i++;

System.out.println(i);//1

### 比较运算符（关系运算符）

比较运算符的结果都是boolean型，也就是要么是true，要么是false。

**比较运算符“==”不能误写成“=” 。**

|  |  |  |  |
| --- | --- | --- | --- |
| 运算符 | 运算 | 范例 | 结果 |
| == | 相等于 | 4==3 | False |
| ！= | 不等于 | 4!=3 | True |
| < | 小于 | 4<3 | False |
| > | 大于 | 4>3 | True |
| <= | 小于等于 | 4<=3 | False |
| >= | 大于等于 | 4>=3 | True |
| Instanc  eof | 检查是否是类的对象 | “hello” instanceof  String | True |

### 逻辑运算符

&—逻辑与 | —逻辑或 ！—逻辑非

&& —短路与 || —短路或 ^ —逻辑异或

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **a** | **b** | ***a&b*** | ***a|b*** | ***!a*** | ***a^b*** | ***a&&b*** | ***a||b*** |
| ***true*** | ***true*** | true | true | false | false | true | true |
| ***true*** | ***false*** | false | true | false | true | false | true |
| ***false*** | ***true*** | false | true | true | true | false | true |
| ***false*** | ***false*** | false | false | true | false | false | false |

逻辑运算符用于连接布尔型表达式，在Java中不可以写成3<x<6，应该写成x>3 & x<6 。

“&”和“&&”的区别：

单&时，左边无论真假，右边都进行运算；

双&时，如果左边为真，右边参与运算，如果左边为假，那么右边不参与运算。

“|”和“||”的区别同理，||表示：当左边为真，右边不参与运算。

异或( ^ )与或( | )的不同之处是：当左右都为true时，结果为false。

理解：异或，追求的是“异”!

### 位运算符

* **& 表示按位与，两个同时为1返回1，其它都是0**
* 例如： System.out.println(5&9);//0101 1001
* System.out.println(**0b0101**&**0b1001**);
* **| 表示按位或，如果两个有一个1就是1，否则都是0**
* **例如： System.out.println(5&9);//0101 1001**

**^ 表示按位异或，两个相同返回0，否则返回1**

**<< 表示左移位（5<<3） >>表示右移位**

**~ 取反运算符 若n>0 ~n =- ~(n+1)**

### 三元运算符

格式:

(条件表达式)? 表达式1：表达式2；

为true，运算后的结果是表达式1；

为false，运算后的结果是表达式2；

int a=10;

a = a ==10?1:2;

System.out.println(a);

表达式1和表达式2为同种类型

三元运算符与if-else的联系与区别：

1）三元运算符可简化if-else语句

2）三元运算符要求必须返回一个结果。boolen

3）if后的代码块可有多个语句

## 程序流程控制

### 顺序结构

程序从上到下逐行地执行，中间没有任何判断和跳转。

Java中定义成员变量时采用合法的前向引用。如：

public class Test{

int num1 = 12;

int num2 = num1 + 2;

}

错误形式：

public class Test{

int num2 = num1 + 2；

int num1 = 12;

}

### 分支结构

根据条件，选择性地执行某段代码。

有***if…else***和***switch***两种分支语句。

### if语句三种格式：

1. if(true){

执行代码块；

}

2. if(条件表达式){

执行代码块；

}else{

执行代码块；

}

3. if(条件表达式){

执行代码块；

}else if (条件表达式){

执行代码块；

}

……

else{

执行代码块；

}

**分支结构-switch**

switch(变量){

case 常量1:

语句1;

break;

case 常量2:

语句2;

break;

… …

case 常量N:

语句N;

break;

default:

语句;

break;

}

**switch语句规则**

* switch(表达式)中表达式的**返回值**必须是下述几种类型之一：**byte，short，char，int，枚举，String**；
* case子句中的值必须是**常量**，且所有case子句中的值应是不同的；
* default子句是**可任选的**，当没有匹配的case时，执行default
* break语句用来在执行完一个case分支后使程序跳出switch语句块；如果没有break，程序会顺序执行到switch结尾

### 循环结构

根据循环条件，重复性的执行某段代码。

* 循环语句功能
  + *在某些条件满足的情况下，反复执行特定代码的功能*
* 循环语句的四个组成部分
  + *初始化部分（init\_statement）*
  + *循环条件部分（test\_exp）*
  + *循环体部分（body\_statement）*
  + *迭代部分（alter\_statement）*
* 循环语句分类
  + *for 循环*

for语法格式

for (初始化表达式; 布尔值测试表达式; 更改表达式)｛

语句或语句块；

｝

* + *while 循环*

while语法格式

[初始化语句]

while( 布尔值测试表达式)｛

语句或语句块;

[更改语句;]

}

* + *do/while 循环*

do while语法格式

[初始化语句]

do｛

语句或语句块;

[更改语句;]

｝while(布尔值测试表达式);

有***while、do…while、for***三种循环语句。

### 特殊流程控制语句

* **break 语句**
  + ***break语句用于终止某个语句块的执行***
* **continue 语句**
  + ***continue语句用于跳过某个循环语句块的一次执行***

**return语句**

并非专门用于结束循环的，它的功能是结束一个方法。当一个方法执行到一个return语句时，这个方法将被结束。

* ***与break和continue不同的是，return直接结束整个方法，不管这个return处于多少层循环之内***
* break只能用于**switch语句**和**循环语句**中。
* continue 只能用于**循环语句**中。
* 二者功能类似，但continue是终止**本次**循环，break是终止**本层**循环。
* break、continue之后不能有其他的语句，因为程序永远不会执行其后的语句。

注：***JDK1.5提供了foreach循环，方便的遍历集合、数组元素***。

If语句

int a = 100;

int b = 100;

if(a<=100 | (b/0)==1){

System.out.println("进入了");

}else{

System.out.println("没进入");

}

int a = 10;

if(!(a==10)){

System.out.println("进入");

}else{

System.out.println("不进入");

}

九九乘法表

**public** **class** Jiujiu {

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

System.***out***.println("九九乘法表");

**for** (**int** i=1;i<=9;i++ )

{

**for** (**int** j=1;j<=i;j++ )

{

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

}

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

}

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

**for** (**int** i=9;i>=1;i-- )

{

**for** (**int** j=1;j<=i;j++ )

{

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

}

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

}

}

}

## 数组

* 数组是多个相同类型数据的组合，实现对这些数据的统一管理
* 数组中的元素可以是任何数据类型，包括基本数据类型和引用数据类型
* 数组属引用类型，数组型数据是对象(object)，数组中的每个元素相当于该对象的成员变量

### 一维数组

/\*\*

\*

\* 一维数组的声明方式

\* 创建人:Yitong

\* 时间：2016年5月16日-下午5:44:52

\* **@version** 1.0.0

\*

\*/

**public** **class** Shuzu {

**int** a[];

**int**[] a1;

**double** b[];

Mydate[] c;//对象数组

* Java语言中声明数组时不能指定其长度(数组中元素的数)
* 例如： int a[5]; //非法

### 一维数组的初始化

//动态初始化：数组声明且为数组元素分配空间与赋值的操作分开进行

**int**[] arr = **new** **int**[3];

arr[0] = 3;

arr[1] = 9;

arr[2] = 8;

//静态初始化：在定义数组的同时就为数组元素分配空间并赋值

**int** a11[] = **new** **int**[]{ 3, 9, 8};

* **int**[] a = {3,9,8};

### 数组元素的引用

* 定义并用运算符**new**为之分配空间后，才可以引用数组中的每个元素；
* 数组元素的引用方式：数组名[数组元素下标]
* 数组元素下标可以是整型常量或整型表达式。如a[3] , b[i] , c[6\*i];
* 数组元素下标从**0**开始；长度为n的数组合法下标取值范围: **0** —>**n-1**；如int a[]=new int[3]; 可引用的数组元素为a[0]、a[1]、a[2]
* 每个数组都有一个属性***length***指明它的长度，例如：**a.length** 指明数组**a**的长度(元素个数)
* ***数组一旦初始化，其长度是不可变的***

### 多维数组之二维数组

* **格式1（动态初始化）：int**[][] arr = **new** **int**[3][2];
* **定义了名称为arr的二维数组**
* **二维数组中有3个一维数组 *arr[i]就是指向一维数组的内存地址***
* **每一个一维数组中有2个元素**
* **一维数组的名称分别为arr[0], arr[1], arr[2]**
* **给第一个一维数组1脚标位赋值为78写法是：arr[0][1] = 78;**
* **格式2（动态初始化）：int[][] arr = new int[3][];**
* **二维数组中有3个一维数组。**
* **每个一维数组都是默认初始化值null (注意：区别于格式1）**
* **可以对这个三个一维数组分别进行初始化**
* arr[0] = **new** **int**[3]; arr[1] = **new** **int**[1]; arr[2] = **new** **int**[2];
* **注：**
* **int[][]arr = new int[][3]; //非法**

### 多维数组

* *格式3（静态初始化）：int[][] arr = new int[][]{{3,8,2},{2,7},{9,0,1,6}};*
* *定义一个名称为arr的二维数组，二维数组中有三个一维数组*
* *每一个一维数组中具体元素也都已初始化*
* *第一个一维数组 arr[0] = {3,8,2};*
* *第二个一维数组 arr[1] = {2,7};*
* *第三个一维数组 arr[2] = {9,0,1,6};*
* *第三个一维数组的长度表示方式：arr[2].length;*
* *注意特殊写法情况：int[] x,y[]; x是一维数组，y是二维数组。*

package com.yi.tong;

import java.util.Iterator;

import java.util.Scanner;

/\*\*

\*

\* 第四课作业Homework

\* 创建人:Yitong

\* 时间：2016年5月12日-上午10:02:22

\* @version 1.0.0

\*

\*/

public class Homework {

public static void main(String[] args) {

/\*

\* 要求通过控制台录入人员信息

\* （姓名、年龄、身高、性别）

\* 最后输出人员的信息

\*/

Scanner can = new Scanner(System.in);

System.out.println("请输录入人员个数：");

int num = can.nextInt();

//建立二维数组

String[][] person = new String[num][4];

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

System.out.println("请输入第"+(i+1)+"人员信息：");

System.out.println("请输入人员姓名");

person[i][0] = can.next();

System.out.println("请输入人员性别");

person[i][1] = can.next();

System.out.println("请输入人员年龄");

person[i][2] = can.next();

System.out.println("请输入人员身高");

person[i][3] = can.next();

}

//循环遍历

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

System.out.println("请输入第"+(i+1)+"个人员信息");

for (int j = 0; j < person[i].length; j++) {

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

}

System.out.println();

}

}

}

## 数据结构-排序方法

### 数据结构

* **编程的本质**就是对数据（信息以数据的形式而存在）的处理，实际编程中不得不处理大量数据，因此实际动手编程之前必须先分析处理这些数据，处理数据之间存在的关系。
* 现实的数据元素之间有着纷繁复杂的逻辑关系，需要采用合适的物理结构来存储这些数据，并以此为基础对这些数据进行相应的操作。同时，还要分析这些数据结构在时间、空间上的开销的优劣。这种专门研究应用程序中数据之间逻辑关系、存储方式及其操作的学问就是**数据结构**。

### 逻辑结构

Data-Structure = (D,R)

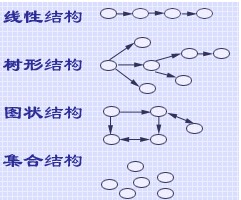
数据元素之间存在的关联关系被称为数据的逻辑结构。归纳起来，应用程序中的数据大致有如下四种基本的逻辑结构：

**集合：**数据元素之间只有“同属于一个集合”的关系

**线性关系：**数据元素之间存在一个对一个的关系

**树形结构：**数据元素之间存在一个对多个的关系

**图状结构或网状结构：**数据元素之间存在多个对多个的关系

****

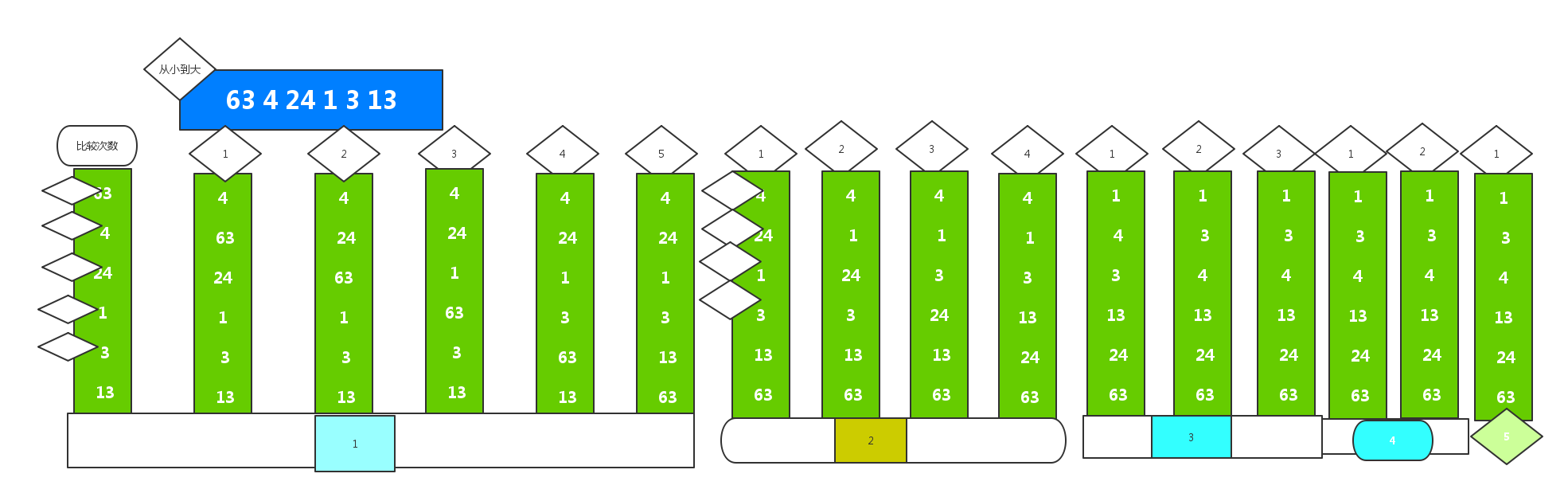
**常用排序之内部排序**

**选择排序**

**直接选择排序、堆排序**

**交换排序**

**冒泡排序、快速排序**

****

**插入排序**

**直接插入排序、折半插入排序、Shell排序**

**归并排序**

**桶式排序**

**基数排序**

**外部排序是在外部完成，数据量比较大，内部无法完成，在外部存储器进行排序（硬盘）。外部排序由多个内部排序组成的，拿一部分排完序放在硬盘里，所以相比内部排序，速度比较慢。**

# OOP之面向对象基础

关键词：

|  |  |
| --- | --- |
| **单 词** | **说 明** |
| **class** | **类，种类** |
| **public** | **公共的，公有的** |
| **private** | **私有的，私人的** |
| **object** | **对象，物体** |
| **encapsulation** | **封装，包装** |
| **attribute** | **属性，特征** |
| **method** | **方法** |
| **member** | **成员** |
| **constructor** | **构造方法，构造器** |

什么是面向对象？

面向对象就是**使用对象进行程序设计**，简写成OOP。



## 面向对象的编程思想

### C语言的编程方式

* C语言是结构化的编程语言（Structured Programming，简作SP）；
* 强调精巧的数据结构和优越的算法，关注程序的运行效率；
* 不适合于大规模的程序开发；
* 程序模块之间协调性差，程序的重用性、安全性、健壮性和可维护性都存在较大的缺陷。

### SP和OOP对比

|  |  |  |
| --- | --- | --- |
|  | **SP** | **OOP** |
| **设计思路** | **自顶向下、层次化、分解** | **自底向上、对象化、综合** |
| **程序单元** | **函数模块** | **对象** |
| **设计方法** | **程序 = 算法 + 数据结构** | **程序 = 对象 = 数据 + 方法** |
| **优点** | **相互独立，代码共享** | **接近人的思维方式**  **模拟客观世界** |
| **缺点** | **数据与程序不一致**  **维护困难** | **客观世界的无序性**  **概念不成熟** |

## 面向对象的三大原则

## 对象

* 对象是面向对象编程的核心部分，是实际存在的具体实体，具有明确定义的状态和行为；
* 对象其实就是“数据”和“函数”的封装体，其中：

数据表示自身的状态，也称作“属性”或“成员数据”；

函数表示自身的功能，也称作“方法”或“成员函数”

### 类

* 人们为了更好地认识世界，将现实生活中的事物（对象）划分成类；
* 同一类中的事物总是具有一些共性；
* 类以共同的特性和行为定义实体；
* 类是具有相同属性和和行为的一组对象的集合。

### 属性

* 事物的特性在类中用变量表示；
* 每个对象的每个属性都拥有其特定的值；
* 属性名称由类的所有对象共享；
* 对象或实体拥有的特征在类中表示时称为属性。

### 方法

* 事物的行为和动作在类中用函数表示；
* 每个对象都有相同的动作和行为；
* 对象执行的操作在类中表示为方法。

### 类和对象的区别

* 类是用来描述实体的“模板”或“原型”
* 对象是实际的实体，每一个对象都是类的一个具体实例；
* 类用来定义对象所有的属性和方法，同一类的所有对象都拥有相同的特征和操作；
* 可以将类理解成生产产品的模具，而对象则是根据此模具生产的一个个产品。

### 类与结构

区别在于：结构体一般只有数据（属性）而没有函数（方法）

### 多态

* 多态是指同一函数在不同的类中有不同的实现；
* 多态的好处就是使类更灵活，更便于扩充。

## 抽象

* 把相同的或相似的对象归为一类的这个过程就是抽象，所以，抽象就是分析问题的方法；
* 抽象的基本原则：
  + 只关心主要问题，而不关心次要问题；
  + 只关心主要矛盾，而不关心次要矛盾；
  + 只关心相同的东西，而不关心不同的东西；
  + 只关心问题是什么，能够完成什么，而不关心怎样去完成。
* 抽象的过程其实就是面向对象编程的核心思想。

**package** com.yi.tong;

/\*\*

\*

\* StudentName学生类

\* 创建人:Yitong

\* 时间：2016年5月12日-下午9:17:07

\* **@version** 1.0.0

\*

\*/

**public** **class** StudentName {

//属性 姓名 年龄 地址

//方法 吃饭 睡觉 上课

//姓名

**private** String name;

//年龄

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

//体重

**public** **float** weight;

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

**this**.name = name;

}

//获取年龄

**private** String getName() {

// **TODO** Auto-generated method stub

**return** name;

}

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

**return** age;

}

**public** **float** getWeight() {

**return** weight;

}

**public** **void** setWeight(**float** weight) {

**this**.weight = weight;

}

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

**this**.age = age;

}

//吃饭

**public** **void** eat(){

System.***out***.println("吃饭了！！！！");

weight++;

}

//跑步

**public** **void** run(){

System.***out***.println("跑步");

weight--;

}

}

**package** com.yi.tong;

/\*\*

\*

\* Test

\* 创建人:Yitong

\* 时间：2016年5月12日-下午9:44:26

\* **@version** 1.0.0

\*

\*/

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

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

StudentName student = **new** StudentName();

//student.name = "易通";

student.setName("易通");

student.setAge(24);

student.setWeight(42.5f);

student.eat();

System.***out***.println(student.weight);

student.run();

System.***out***.println(student.weight);

StudentName2 studentName2 = **new** StudentName2("易通",22,120F);

studentName2.printMessage();

studentName2.eat();

studentName2.eat();

studentName2.eat();

studentName2.eat();

studentName2.eat();

studentName2.eat();

studentName2.run();

studentName2.printMessage();

}

}

## 定义类的语法

class 类名

{

成员列表

//包括成员变量和成员函数

}

struct 结构体名

{

成员列表

//只有成员变量，没有成员函数

};

习惯上，类名的首字母大写，如：Student（学生类）

**package** com.yi.tong;

**public** **class** Test

{

/\*main方法，程序入口\*/

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

{

Student std; //声明Student类型的引用

std = **new** Student(); //创建Student类型的对象

std.name = "张三"; //为姓名赋值

std.age = 18; //为年龄赋值

std.weight = 50; //为体重赋值

std.dining(); //调用成员方法

System.***out***.println(std.weight);

std.walk();

System.***out***.println(std.weight);

}

}

### 创建对象的语法

* 和数组相似，对象也是引用数据类型，只能使用new运算符从堆中分配内存；
* 创建对象的一般语法：

类名 引用名 = new 类名();注意：这里要有括号；

* 使用已经定义好的类，创建该类对象的过程称为“实例化”。

### 成员运算符

* 在C语言中，必须要先声明结构体变量，才可以访问结构体中的成员；
* 同样的，只有先实例化类的对象，才可以访问到类中的成员（属性和方法）；
* 使用成员运算符（.）来访问成员属性或成员方法
* 一般语法是：

对象名.成员名

* 如：std.age = 18;  **//为成员属性赋值**

std.dining(); //调用成员方法

### 访问权限：public和private

* C语言中结构体的成员可以从任何地方进行访问，这将给数据的安全留下极大的隐患；
* 为了避免从类外部直接访问类成员而造成的数据损坏，Java对类成员的访问制定了约束；
* 关键字public和private是访问修饰符，用来说明某个成员是否可以从类外部进行访问；
* public修饰的成员可以在任何地方进行访问，不受任何约束；
* private修饰的成员只能够被本类中的其它成员访问，而不能从类的外部进行访问。
* 无法从类的外部访问私有成员；
* 其它类的私有成员对于当前类也是隐藏的
* 加上访问修饰符有时候可能会给操作数据带来不便，但可以在很大程度上保证数据的安全；
* 一般地，我们会将成员属性声明为private，而将成员方法声明为public，但这样做并不是绝对的
* 有时候，类外部可能要操作到某些私有数据成员，那么就可以增加一个公有的方法，再由这个方法来操作私有数据，避免因类外部的误操作而造成的数据损坏；
* 因为main方法要由类外部的虚拟机来调用，所以main方法必须声明成public。

class Student { //定义学生类

private String name; //姓名，私有

private int age; //年龄，私有

private float weight; //体重，私有

public void setName(String name) { //为姓名赋值的方法，公有

this.name = name;

}

public void setAge(int age) { //为年龄赋值的方法，公有

this.age = age;

}

public void setWeight(float weight) { //为体重赋值的方法，公有

this.weight = weight;

}

public void display() { //将所有信息打印出来的方法，公有

System.out.println("姓名：" + name + ",年龄：" + age + ",体重：" + weight);

}

public void dining() {……} //吃饭的方法，公有，代码略

public void walk() {……} //走路的方法，公有，代码略

}

public class Test {

public static void main(String[] args) {

Student std = new Student(); //实例化学生类对象

std.setName("张三"); //为姓名赋值

std.setAge(18); //为年龄赋值

std.setWeight(55); //为体重赋值

std.dining(); //调用吃饭的方法

std.display(); //将信息打印出来

}

}

### 对象初始化

* 在上例中，只能逐个地为数据成员赋值，如果想在对象实例化的同时就初始化成员属性，就使用到了构造方法；
* 构造方法是特殊的成员方法，它与类同名，在对象实例化时由虚拟机自动调用；
* 请注意：构造方法没有返回值类型，也不能有返回值。

/\*定义ConstructorDemo类，对构造方法进行测试\*/

class ConstructorDemo

{

/\*构造方法，方法名与类名完全一致

无需指定返回值类型，也不能有返回值\*/

public ConstructorDemo()

{

System.out.println("这是构造方法");

}

}

/\*Test类，用来容纳main方法

一般将包含有main方法的类声明为public\*/

public class Test

{

/\*main方法，程序入口\*/

public static void main(String[] args)

{

/\*实例化ConstructorDemo类的对象\*/

ConstructorDemo cd = new ConstructorDemo();

}

}

### 构造方法

* 正是由于在实例化对象的同时会自动调用构造方法，所以构造方法一般用来给数据成员分配资源或初始化数据成员；
* 构造方法的一般形式：

访问权限 类名（形参列表） {

方法体

}

* 因为是由虚拟机来调用构造方法，所以构造方法一般应定义成public。

class Student { //定义学生类

private String name; //姓名，私有

private int age; //年龄，私有

private float weight; //体重，私有

//构造方法，根据传递进来的参数，为数据成员赋值

public Student(String n, int a, float w) {

//分别为每个数据成员赋初始值

name = n;

age = a;

weight = w;

}

public void setName(String n) {……} //为姓名赋值的方法，公有，代码略

public void setAge(int a) {……} //为年龄赋值的方法，公有，代码略

public void setWeight(float w) {……} //为体重赋值的方法，公有，代码略

public void display() {……} //将所有信息打印出来的方法，公有，代码略

public void dining() {……} //吃饭的方法，公有，代码略

public void walk() {……} //走路的方法，公有，代码略

}

public class Test {

public static void main(String[] args) {

//利用构造方法，为数据成员指定初始值

Student std = new Student("张三", 18, 55);

std.display(); //将信息打印出来

}

}

* 每个对象在生成时都必须执行构造方法，而且只能执行一次；
* 如果构造方法调用失败，那么对象也无法创建；
* 不可以显式地直接调用构造方法；
* 在没有定义构造方法的情况下，类会自动产生一个无参数的默认构造方法，这个默认的构造方法什么都不做；
* 一旦显式地定义了构造方法，默认构造方法自动消失。

### 总结

* 对象由状态（属性）和行为（方法）组成；
* 类是具有相同属性和方法的对象的集合；
* 封装可以隐藏对象实现的具体细节；
* 必须先实例化类的对象，才可以访问到其中的成员；
* 成员运算符用于访问对象的成员；
* 成员可以定义为公有，也可以定义为私有；
* 构造方法一般用来初始化对象中的数据成员；
* 如果不定义构造方法，将有默认的构造方法，一旦定义，默认的构造方法自动消失。

## OOP之方法重载

关键词：

|  |  |
| --- | --- |
| **单 词** | **说 明** |
| **this** | **这，这个** |
| **static** | **静态的** |
| **package** | **包裹，包** |
| **import** | **引入，导入** |
| **overloaded** | **重载，超载** |
| **overloaded method** | **方法重载** |

* 方法重载
* 重载构造方法
* 对象在内存中的存放形式
* this关键字
* 静态成员，static关键字
* 打包，package关键字
* 导入包，import关键字

### 回顾函数

* 在Java中，函数无需C语言中的前向声明，只要直接定义即可；
* Java中，函数被称为方法；
* 定义一个方法的一般形式：

[访问权限] 返回值类型 方法名称(形参列表) {

方法主体;

}

* 我们更关注的是方法原型，对于方法原型的三个主要组成部分应当明确区分：
  + 返回值类型
  + 方法名称
  + 形式参数

**package** com.yi.tong;

**import** sun.net.www.content.audio.x\_aiff;

/\*\*

\*

\* 定义求平方值

\* 创建人:Yitong

\* 时间：2016年5月16日-下午10:38:17

\* **@version** 1.0.0

\*

\*/

**public** **class** Yi02 {

**public** **int** square(**int** x){

**return** (x\*x);

}

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

Yi02 a = **new** Yi02();

**int** b=8;//8的平方=64

**int** c = 0;

c= a.square(b);

System.***out***.println(b+”8的平方”+c);

}

}

**package** com.yi.tong;

/\*\*

\* 求绝对值

\* Yi01

\* 创建人:Yitong

\* 时间：2016年5月14日-下午8:44:39

\* **@version** 1.0.0

\*

\*/

**public** **class** Yi01 {

**public** **int** abs(**int** x){

**return** (x>=0) ? x : -x;

}

**public** **float** abs(**float** x){

**return** (x>=0) ? x : -x;

}

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

Yi01 obj = **new** Yi01();

**int** a=-8;

**int** b=0;

b=obj.abs(a);

System.***out***.println(a+"=int类型的绝对值=="+b);

**float** c=-3.0f;

**float** d= 0f;

d = obj.abs(c);

System.***out***.println(c+"==float类型的绝对值="+d);

}

}

### 方法重载的三大原则

* 方法名相同
* 参数不同（可以有三方面的不同）
  + 数量不同
  + 类型不同
  + 顺序不同（参数类型有不一样的）
* 同一作用域
* 温馨提示：方法重载跟方法的返回值类型没有任何关系。也就是说，只有返回值不同的方法不能构成重载。

**package** com.yi.tong;

/\*\*

\*

\* 重载方法

\* 创建人:Yitong

\* 时间：2016年5月16日-下午11:09:11

\* **@version** 1.0.0

\*

\*/

**public** **class** AbsoluteValue {

**public** **int** abs(**int** x) { //重载方式1，求整数的绝对值

**return** ((x >= 0) ? x : -x);

}

**public** **float** abs(**float** x) { //重载方式2，求浮点数的绝对值

**return** ((x >= 0) ? x : -x);

}

**public** **double** abs(**double** x) { //重载方式3，求双精度数的绝对值

**return** ((x >= 0) ? x : -x);

}

**public** **long** abs(**long** x) { //重载方式4，求长整型数的绝对值

**return** ((x >= 0) ? x : -x);

}

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

AbsoluteValue obj = **new** AbsoluteValue();

**int** a = -8, b;

b = obj.abs(a);

System.***out***.println(a + "的绝对值是" + b);

**float** c = -3.14f, d;

d = obj.abs(c);

System.***out***.println(c + "的绝对值是" + d);

}

}

### 方法重载之一

利用不同数据类型的参数重载；

对于带有相同数量参数但参数类型不同的重载方法，编译器可以区分它们，如：

int square(int x) {……}

float square(float x) {……}

double square(double x) {……}

只要所有参数式样都是唯一的，就可以使用多个重载；

很多编程语言已经重载了输出方法，以便您可以用相同的函数名称输出任何类型的数据。

### 方法重载之二

* 利用不同数量的参数重载，如：

/\*函数定义\*/

int fun(int n1) {……}

int fun(int n1, int n2, int n3) {……}

/\*函数调用\*/

fun(a);

fun(x, y, z)

* 将调用与参数最匹配的方法，如果没有方法得出最佳匹配，那么编译器将报错；
  + 编译器解决重载的方式与方法定义的顺序无关；
  + 重载不考虑方法的返回值类型。

只有属于同一作用域范围内的方法才能构成重载。

### 重载构造方法

* 方法重载的一个重要应用就是重载构造方法；
* 可以定义一组构造方法，使其构成重载关系，从而方便使用者能够灵活机动地以各种不同的方式来实例化对象。

**package** com.yi.tong;

/\*\*

\*

\* 重载方法

\* 创建人:Yitong

\* 时间：2016年5月16日-下午11:30:02

\* **@version** 1.0.0

\*

\*/

**class** MyTime { //定义时间类

**private** **static** **int** *mHour*; //三个成员变量，时、分、秒

**private** **static** **int** *mMinute*;

**private** **static** **int** *mSecond*;

**public** MyTime() { //构造方法重载1

*mHour* = 0;

*mMinute* = 0;

*mSecond* = 0;

}

**public** MyTime(**int** hour) { //构造方法重载2

*mHour* = hour;

*mMinute* = 0;

*mSecond* = 0;

}

**public** MyTime(**int** hour, **int** minute) { //构造方法重载3

*mHour* = hour;

*mMinute* = minute;

*mSecond* = 0;

}

**public** MyTime(**int** hour, **int** minute, **int** second) { //构造方法重载4

*mHour* = hour;

*mMinute* = minute;

*mSecond* = second;

}

**public** **static** **void** display() { //显示时间信息的方法

System.***out***.println(*mHour* + ":" + *mMinute* + ":" + *mSecond*);

}

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

*display*();

}

}

**package** com.yi.tong;

/\*测试时间类\*/

/\*\*

\*

\* 重载方法

\* 创建人:Yitong

\* 时间：2016年5月16日-下午11:32:39

\* **@version** 1.0.0

\*

\*/

**public** **class** Test

{

/\*main方法，程序入口\*/

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

{

//分别调用不同的构造方法来初始化时间对象

MyTime t1 = **new** MyTime(); //调用重载方式1

t1.*display*();

MyTime t2 = **new** MyTime(8); //调用重载方式2

t2.*display*();

MyTime t3 = **new** MyTime(8, 30); //调用重载方式3

t3.*display*();

MyTime t4 = **new** MyTime(8, 30, 30); //调用重载方式4

t4.*display*();

}

}

/\*使用重载构造方法可以使初始化对象的方式灵活机动，大

大方便类的使用者。\*/

### 对象在内存中的存放形式

每个对象都有自己独立的数据成员，但是所有的对象共享成员方法。

### This关键字

* 既然所有的对象都共用相同的成员方法，那么在不同的对象都调用同一方法时，它是怎么确定要使用哪个对象的数据成员呢？
* 每个成员方法都有一个隐含的this引用，它总是指向调用它的对象；
* 关键字this给出用于调用成员方法的对象的地址
* 每当调用成员方法时，编译器会向this分配调用该方法的对象的地址；
* 可以像使用任何引用那样使用this。

/\*this示例，代码片段\*/

public class Student //定义学生类

{

private String mName; //姓名

private int mAge; //年龄

public Student(String name, int age)

{

//隐式调用，等同于this.mName = name;

mName = name;

//显式调用，等同于mAge = age;

this.mAge = age;

}

……

}

## 静态成员变量

* 在成员变量前加static关键字，可以将其声明为静态成员变量；
* 如果类中成员变量被定义为静态，那么不论有多少个对象，静态成员变量只有一份内存拷贝，即所有对象共享该成员变量；
* 静态成员变量的作用域只在类内部，但其生命周期却贯穿整个程序。

**静态成员变量的注意事项**

* 和程序同生命周期；
* 在没有实例化对象时，可以通过类名访问静态成员变量；
* 也可以通过对象访问静态成员变量，但不论使用的是哪个对象，访问到的都是同一个变量；
* 静态成员变量在声明时最好初始化，如果不进行初始化，系统会默认初始化为0。
* 静态成员在编译的时候已经确定内存空间的大小，和当前类本身进行捆绑，存放入静态区中（栈区）；
* 非静态成员，在new创建对象的时候，将属性copy一份作为自己的，确定内存空间大小（堆区）
* 注意：既然静态方法是属于类的，就不允许在该方法中采用this关键字，因为我们一般是通过类去调用，出于合理，this应该指向对象，而不是指向类的。

**静态变量成员方法**

* **在成员方法前加static关键字，可以将其声明为静态成员方法；**
* **静态成员方法只能对类的静态成员变量进行操作；**
* **静态成员方法没有this引用；**
* **在没有实例化对象时，可以通过类名访问静态成员方法。**
* **非静态方法可以调用静态成员也可以调用非静态成员。**

class Dog {

private static int count = 0; //静态成员变量

public Dog() {

count++;

}

//显示数量的方法，静态成员方法

public static void displayCount() {

System.out.println("当前狗的数量是：" + count);

}

}

public class Test {

public static void main(String[] args) {

//没有实例化对象之前，直接通过类名调用静态成员方法

Dog.displayCount();

Dog d1 = new Dog();

Dog d2 = new Dog();

Dog.displayCount();

}

}

**静态变量小结**

* **静态成员包括静态数据成员和静态成员方法；**
* **静态成员属于整个类而不是属于某个对象，它被该类的所有对象共享；**
* **访问静态成员时一般通过类名访问，也可以通过对象访问；**
* **静态成员也受访问权限的约束；**
* **静态数据成员在使用之前应尽量初始化；**
* **静态成员方法不能操作非静态成员；**
* **静态成员方法没有this引用。**

## 包

* 有时候，类和类的名称可能发生冲突；
* Java提供了把类名空间划分为更容易管理的块的机制，这就是包；
* 包允许将类组合成较小的单元，类似于文件夹；
* 有助于避免命名冲突，分属不同包的类即便名称相同也不会引起误会；
* 能在包与包之间对于类的访问权限提供更有力的约束。

### 使用package关键字打包

* 可以使用package关键字将源文件中的类打入某个包中，语法是：

package 包名;

* 该语句必须是整个源文件的第一条语句；

//将本源文件中的所有类打到mypkg包中

package mypkg;

class Student //定义学生类

{

……

}

* 包中还可以包含下一级子包，这与文件目录体系非常相似

### 缺省包

* 如果省略了package语句，类将保存在一个缺省的没有名称的包中；
* 尽管缺省包很方便，但对于大型的程序，它是不恰当的；
* 请尽量为自己编写的类定义具体的包。

### 使用import关键字导入包

* 如要使用到某个包中的类，就需要使用import关键字将包或类显式地包括到程序中，有如下两种情况：

/\*导入java包下的io子包中的所有类\*/

import java.io.\*;

/\*导入mypkg包下的Student类，但并不导入该包中的其它类\*/

import mypkg.Student;

……

* 一个程序中允许有多条import语句，导入多个包或多个类。

### 静态导入

例如：import **static** com.tz.demo.Content.**\***

ent

静态导入：就是把一个类中的静态成员通过静态导入的方式，可以当做自己的静态成员一样使用。

（针对于***静态成员---->静态方法与静态常量***）

***ctrl+shift+o 快速导入***

### 总结

* 方法重载可以方便方法的调用者，但方法重载应遵循三大原则：
  + 方法名相同
  + 参数列表不同
  + 属于同一作用域
* 可以适当地重载构造方法，使初始化对象的方式更加灵活；
* this引用总是指向调用方法的对象本身；
* 静态成员属于类而不是属于某个对象；
* 总是将自己定义的类打入到具体的包中是良好的编程习惯；
* 可以使用import语句导入需要的包或类。