Java 的那点事

# 第三章 Java的基本程序设计结构

## 3.1 一个简单的 Java 应用程序

链接：[第一个java 程序](https://github.com/Swolf-GH/Java/blob/master/CoreJavaVolume/Chapter03/src/FirstSample.java)

public 称为访问修饰符（access modifier）这些修饰符用于控制程序的其他部分对这段代码的访问级别。

class Java应用程序中的全部内容都必须放置在类中。

类名的命名规范：类名是以大写字母开头的名词。如果名字由多个单词组成，每个单词的第一个字母都应该大写（骆驼命名法）。

源代码的文件名必须与公共类的名字相同，并用 .java 作为扩展名。

根据 Java 语言规范，main 方法必须声明为 public .

## 3.2 注释

Java 中有三种标记注释的方法：

1. // 其注释内容从//开始到本行结尾
2. /\* 需要注释的内容 \*/
3. /\*\* 可以自动生成文档的注释 \*/
4. 注释是不可以嵌套的

链接：[java中的各种注释](https://github.com/Swolf-GH/Java/blob/master/CoreJavaVolume/Chapter03/src/Annotation.java)

## 3.3 数据类型

Java 是一种强类型语言。这就意味着必须为每一个变量声明一种类型。

在 Java 中一共有 8 中基本数据类型（primitive type），其中有4中整形、2种浮点型、1种用于表示Unicode编码的字符单元的字符类型char和1种用于表示真值的boolean类型。

Java 有一个能够表示任意精度的算数包，通常称为“大数值”（big number）。虽然被称为大数值，但他并不是一种新的Java 类型，而是一个Java 对象。

3.3.1 整型

整型表示没有小数部分的数值，它允许是负数。Java 提供了4中整形。

java 中整形的范围与运行java代码的机器无关。

|  |  |  |
| --- | --- | --- |
| 类型 | 存储需求 | 取值范围 |
| int | 4字节 | 超过20亿 |
| short | 2字节 | -32768 ~ 32767 |
| long（有一个后缀L） | 8 字节 | 好大好大 so big😱 |
| byte | 1字节 | -128 ~ 127 |

进制

|  |  |  |
| --- | --- | --- |
| 进制 | 前缀 | 例子 |
| 二进制 | 0b或0B | 0b1001 就是十进制的9 |
| 八进制 | 0 | 010 就是十进制的8 |
| 十六进制 | 0x或0X | 0xCAFE |

从java7 开始可以为数字字面量加下划线，如用1\_000\_000 表示一百万。

3.3.2 浮点类型

浮点类型用于表示有小数部分的数值。

|  |  |  |
| --- | --- | --- |
| 类型 | 存储需求 | 取值范围 |
| float（有一个后缀F） | 4字节 | 好大好大（有效位数为6~7位） |
| double | 8字节 | 比好大还大（有效位数为15位） |

3.3.3 char 类型

char类型的字面量值要用单引号括起来。

特殊字符的转义序列

|  |  |  |
| --- | --- | --- |
| 转义序列 | 名称 | Unicode值 |
| \b | 退格 | \u0008 |
| \t | 制表 | \u0009 |
| \n | 换行 | \u000a |
| \r | 回车 | \u000b |
| \” | 双引号 | \u0022 |
| \’ | 单引号 | \u0027 |
| \\ | 反斜杠 | \u005c |

3.3.4 Unicode 和 char 类型

3.3.5 boolean 类型

boolean（布尔）类型有两个值：false 和true，用来判定逻辑条件。

整型值和布尔值之间不能进行相互转换。

## 3.4 变量

每个变量都有一个类型。

变量的声明：

3.4.1 变量的初始化

声明一个变量后，必须用赋值语句对变量进行显式初始化，千万不要使用为初始化的变量。

在Java 中，变量的声明尽可能地靠近变量第一次使用的地方，这是一种良好的程序编写风格。

变量的初始化：

### 3.4.2 常量

在Java中用关键字final指示常量

关键字final表示这个常量只能被赋值一次。一旦被赋值后，就不能再更改了。

习惯上常量名使用全大写。

常量的使用：

## 3.5 运算符

在Java中，使用算数运算符+、-、\*、/ 表示加、减、乘、除运算。当参与运算的两个数都是整数时，表示整数除法；否则，表示浮点数除法。整数的求余操作（有时称为取模）用%表示。例如：15/2等于7，15%2等于1，15.0/2等于7.5。

整数被0除将会产生一个异常，而浮点数被0除将会得到无穷大或NaN结果。

### 3.5.1 数学函数与常量

在Math类中，包含了各种各样的数学函数。

Math提供了一些常用的三角函数：

sin

cos

tan

atan

atan2

还有指数函数以及它的反函数——自然对数以及10为底的对数：

exp

log

log10

Java还提供了两个用于表示 π 和 E 常量的近似值：

PI

E

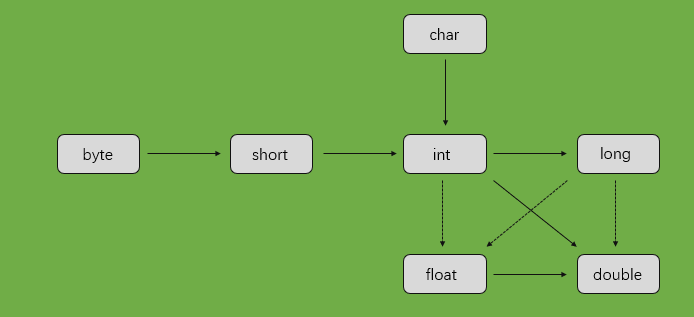
静态导入：

import static java.lang.Math.\*;

代码：

3.5.2 数值类型之间的转换

图



图中有6个实心箭头，表示无信息丢失的转换；有3个虚线头，表示可能有精度损失的转换。

**默认类型转换：**

1. 如果两个操作数中有一个是double 类型，另一个操作数就会转换为double 类型；
2. 否则如果其中一个操作数是float 类型，另一个操作数将会转换为float类型；
3. 否则，如果其中一个操作数是long 类型，另一个操作数将会转换为long 类型
4. 否则，两个操作数都将被转换为int 类型。

3.5.3 强制类型转换

语法：

double x = 12.3456;

int num1 = (int)x;

这样双精度浮点数x 就被强制转换为了整形num1（num1的值为12，因为强制类型转换通过截断小数部分将浮点值转换为整型）。

3.5.4 结合赋值和运算符

语法：

x += 5;

即：x = x + 5;

3.5.5 自增和自减运算符

a++;

++a;

a--;

--a;

3.5.6 关系和boolean 运算符

1==2; // 1等于2 false

1!=2; // 1不等于2 true

此外还有：< > <= >=

|  |  |  |
| --- | --- | --- |
| 逻辑“与” | && | 是按照**短路**方法来求值的 |
| 逻辑“或” | || |
| 逻辑“非” | ! |  |

java 支持三元运算符 ?:

语法：x < y? x : y ; //返回x 和 y 中较小的一个数

3.5.7 位运算符

3.5.8 括号与运算符级别

表

|  |  |
| --- | --- |
| 运算符 | 结合性 |
| [] . ()(方法调用) | 从左向右 |
| ! ~ ++ -- + - ()(强制类型转换) new | 从右向左 |
| \* / % | 从左向右 |
| + - | 从左向右 |
| << >> >>> | 从左向右 |
| < <= > >= instanceof | 从左向右 |
| == != | 从左向右 |
| & | 从左向右 |
| ^ | 从左向右 |
| | | 从左向右 |
| && | 从左向右 |
| || | 从左向右 |
| ?: | 从右向左 |
| = += -= \*= /= %= &= |= ^= <<= >>= >>>= | 从右向左 |

3.5.9 枚举类型

语法：

enum Size {SMAL,MEDIUM,LARGE,EXTRA\_LARGE};

变量声明：

Size s = Size.MEDIUM;

## 3.6 字符串

Java 没有内置的字符串类型，而是在标准Java类库中提供了一个预定义类，很自然地叫做String。每个用双引号括起来的字符串都是String类的一个实例。

### 3.6.1 子串

String类的substring方法可以从一个较大的字符串提取出一个字串。

代码：

### 3.6.2 拼接

Java语言允许使用 + 号连接（拼接）两个字符串。

当将一个字符串和一个非字符串的值进行拼接时，后者被转换为字符串（任何一个java对象都可以转换为字符串）。

如果需要将多个字符串放在一起，用一个界定符分割，可以使用静态join方法：

代码：

### 3.6.3 不可变字符串

### 3.6.4 检测字符串是否相等

可以使用equals方法检测两个字符串是否相等

语法：

str1.eauals(str2);

如果str1与str2相等返回true，否则返回false。

一定不要使用 == 检测两个字符串是否相等！

实际上只有字符串常量时共享的，而+或substring等操作产生的结果并不是共享的。因此千万不要使用==检测测的相等性。

3.6.5 空串与Null串

3.6.6 码点与代码单元

3.6.7 String API

3.6.8 阅读联机API 文档

### 3.6.9 构建字符串（StringBuilder）

如果需要使用许多小段的字符串构建一个字符串，应该按照以下步骤进行：

代码：

3.7 输入输出

### 3.7.1 读取输入

1. 首先构造一个Scanner对象，并于“标准输入流”关联

Scanner in = new Scanner(System.in);

2. 进行输入操作

Sting name = in.nextLine();

事例代码：

3.7.2 格式化输出