# Java基本程序设计结构

在开始Java的基本程序设计结构之前，我们先来看一个最简单的Java应用程序，他是一个简单的控制台输出语句：

|  |
| --- |
| **package** cn.you;  /\*\* GitHub https://github.com/YorickYou/Java-SE.git \*/  **public** **class** FirstSample {  **public** **static** **void** main(String[] args) {  System.***out***.println("We will not use 'Hello World!'");  }  } |

这个程序看起来再简单不过了，但所有的Java应用程序都具有这种基本的结构，所以还是值得花一些时间来研究的。首先**，Java是区分大小写的**，所以如果出现拼写错误（如main拼写称Main）程序将无法运行。

下面逐行地查看一下这段源代码。关键字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。关于访问修饰符部分，我们会在面向对象部分再做详细介绍。关于关键字，会在下面2.1节中讲到。

关键字class表明Java程序中的全部内容都包括在类中。这里，只需要将类作为一个加载程序逻辑的容器，程序逻辑定义了应用程序的行为。关于类的具体内容我们同样会放在面向对象中概述。类是构建所有Java应用程序和applet的构建块。Java应用程序中的全部内容都必须放置在类中。

关键字class后面紧跟类名。Java中对类名是标识符，类名在编译后会成为Java的类文件名，所以尽管Java对于标识符的长度基本上没有限制，但是平台对于文件名的长度却是有限制的，Linux文件名的长度限制是255个字符，Windows先完全限定名必须少于260个字符，目录名必须少于248个字符。关于标识符，会放在2.2节介绍。

我们在对源代码编译后会形成字节码类文件.class。运行已编译的程序时，Java虚拟机将从指定类中的main方法开始执行（这里的方法和函数是同一个概念），因此为了代码能够执行，在类的源文件中必须包含一个main方法。当然，也可以将用户自定义的方法添加到类中，并且在main方法中调用它们。

**关于main函数：**根据Java语言规范，main方法必须声明为public（Java语言规范是描述是描述Java语言的官方文档）。不过当main方法不是public时，有些版本的Java解释器也可以执行Java应用程序。有个程序员报告了这个bug，但这个bug却被Sun注明“关闭，不予修复”。Sun公司的工程师解释说：Java虚拟机规范并没有要求main方法一定是public，并且“修复这个bug有可能带来其他隐患”。好在，这个问题最终得到了解决。在Java SE1.4及以后的版本中强制main方法是public的。注意这里对于编译器而言仅仅是一个普通方法，但是对于JVM而言却是一个程序的入口，所以main的特殊要求也是针对JVM规范 的。

从上面这段话可以发现一个问题的两个方面。一方面让质量保证工程师判断在bug报告中是否存在问题是一件很头疼的事情，这是因为其工作量巨大，并且工程师对Java的所有细节也未必了解的很清楚。另一方面，Sun公司在Java开源很久以前就把bug报告及其解决方案放到网站上让所有人监督检查，这是一种非常了不起的举动。某些情况下，Sun甚至允许程序员为他们最厌恶的bug投票，并用投票结果来决定发布的下一个JDK版本将修复哪些bug。

需要注意源代码中的大括号{}。在Java中，像在C/C++中一样，用大括号划分程序的各个部分（通常称为块，我们会在流程控制语句中介绍块语句和这个是一个概念）。Java中任何方法的代码都用{开始和}结束。大括号的使用风格也是一个需要注意的地方，这其实是规范编码的范畴，我们推荐括号前面不要换行。

至于static void我们放在后面静态和返回值这部分再说，我们现在只需要注意在一个程序中需要一个特定规范的main方法，作为程序入口。

**C++与Java之main函数异同：**作为一名C++程序员，一定知道类的概念。Java的类与C++的类很相似，但还是有些差异会使人感到困惑。例如，Java中的所有函数都属于某个类的方法（标准术语称之为方法，而不是成员函数）。因此，Java中的main方法必须有一个外壳类。C++的静态成员函数（Static member functions）定义在类的内部，且不对对象进行操作。Java中的main方法必须是静态的。最后，与C/C++一样，关键字void表示这个方法没有返回值，所不同的是main方法没有为推出系统返回“退出代码”。如果main方法正常退出，那么Java应用程序的退出代码为0，表示成功地运行了程序。如果希望在种植程序时返回其他的代码，那就需要调用System.exit方法。（exit（i）i为0正常退出，其他为非正常退出）

接下来我么年开始研究main方法的方法体，那句输出语句。在Java中，每个句子必须用分号结束，需要说明的是，回车不是语句的结束标志，因此，如果需要可以将一条语句写在多行上。在方法体中点号（.）用于调用方法。Java同C++一样，都是采用双引号“”分割字符串。

与其他程序设计语言中的函数一样，在Java的方法中可以没有参数，也可以有多个参数，在一个方法中既使没有参数也是需要使用空括号的。

## 2.1 关键字

在前面的一个小Demo中我们看到了一些关键字，这几我们就来简单看一下Java提供的关键字，有些关键字会在以后的学习中逐个的进行深入的探讨。首先我们先来讲一下什么是关键字，以及注意事项。

* 关键字是Java语言里事先定义的，有特别意义的单词，有时又叫[保留字](http://baike.baidu.com/view/421743.htm /t http://baike.baidu.com/_blank)，含有特别意义的变量。
* Java的关键字对Java的**[编译器](http://baike.baidu.com/view/487018.htm /t http://baike.baidu.com/_blank)**有特殊的意义，他们用来表示一种数据类型，或者表示程序的结构等
* **关键字不能用作变量名、方法名、类名、包名和参数。**
* **Java语言规定关键字不能作为标识符。**
* 目前共有50个Java关键字，其中，"const"和"goto"这两个关键字在Java语言中并没有具体含义。Java语言把它们列为关键字，只是因为"const"和"goto"是其他某些计算机语言的关键字。
* 关键字中所有字母都为小写。

### 2.1.2 关键字的分类

关键字根据用途可以分为一下几类：

#### 2.1.2.1 用于定义数据类型的关键字

* class：类，用来定义一个类
* interface：接口，用来定义一个接口
* byte：基本数据类型之一，字节类型，1字节
* short：基本数据类型之一,短整数类型，2字节
* int：基本数据类型之一，整数类型，4字节
* long：基本数据类型之一，长整数类型，8字节
* float：基本数据类型之一，单精度浮点数类型，2字节
* double：基本数据类型之一，双精度浮点数类型，8字节
* char：基本数据类型之一，字符类型，2字节
* boolean：基本数据类型之一，布尔类型，只有两个值true和false，用来做逻辑运算使用
* void：声明当前成员方法没有返回值

#### 2.1.2.2 用于定义数据类型值的关键字

* true：真
* false：假
* null：空

#### 2.1.2.3 用于定义流程控制的关键字

* if：条件语句的引导词
* while：用在循环结构中
* else：用在条件语句中，表明当条件不成立时的分支
* switch：分支语句结构的引导词
* case：用在switch语句之中，表示其中的一个分支
* default：默认，例如，用在switch语句中，表明一个默认的分支
* do：用在do-while循环结构中
* for：一种循环结构的引导词
* break：提前跳出一个块
* continue：回到一个块的开始处
* return：从成员方法中返回数据

#### 2.1.2.4 用于定义访问权限修饰符的关键字

* private：一种访问控制方式：私用模式,只能自己访问，用来修饰类中的字段或方法，以及内部类
* protected：一种访问控制方式：保护模式,允许子类和同包类访问，用来修饰类中的子弹或方法，以及内部类
* public：一种访问控制方式：共用模式,允许所有外部类访问，可以用来修饰所有
* 访问权限修饰符权限从高到低排列是public  ,protected   private。

#### 2.1.2.5 用于定义类，函数，变量修饰符的关键字

* abstract：表明类或者成员方法具有抽象属性
* final：用来说明最终属性，表明一个类不能派生出子类，或者成员方法不能被覆盖，或者成员域的值不能被改变
* static：表明具有静态属性
* synchronized：表明一段代码需要同步执行

#### 2.1.2.6 用于定义类与类之间关系的关键字

* extends：表明一个类型是另一个类型的子类型，这里常见的类型有类和接口
* implements：表明一个类实现了给定的接口

#### 2.1.2.7 用于定义建立实例及引用实例，判断实例的关键字

* new：用来创建新实例对象
* this：指向当前实例对象的引用
* super：表明当前对象的父类型的引用或者父类型的构造方法
* instanceof：用来测试一个对象是否是指定类型的实例对象

#### 2.1.2.8 用于异常处理的关键字

* try：尝试一个可能抛出异常的程序块
* catch：用在异常处理中，用来捕捉异常
* finally：用于处理异常情况，用来声明一个基本肯定会被执行到的语句块
* throw：抛出一个异常
* throws：声明在当前定义的成员方法中所有需要抛出的异常

#### 2.1.2.9 用于包的关键字

* import：表明要访问指定的类或包
* package：包

#### 2.1.2.10 其他修饰符关键字

* native：用来声明一个方法是由与计算机相关的语言（如C/C++/FORTRAN语言）实现的
* strictfp：用来声明FP\_strict（单精度或双精度浮点数）表达式遵循IEEE 754算术规范
* transient：声明不用序列化的成员域
* volatile：作为指令[关键字](https://baike.baidu.com/item/%E5%85%B3%E9%94%AE%E5%AD%97/7105697" \t "https://baike.baidu.com/item/volatile/_blank)，确保本条指令不会因[编译器](https://baike.baidu.com/item/%E7%BC%96%E8%AF%91%E5%99%A8/8853067" \t "https://baike.baidu.com/item/volatile/_blank)的优化而省略，且要求每次直接读内存中的值。
* assert：用来进行程序调试

## 2.2 标识符

在上面我们知道了关键字不能用于变量名、类名、方法名等，那么Java中的用来表示这些名词的单词是什么呢？有什么命名规则呢？答：标识符；下面我们来解读一下标识符的组成。

### 2.2.1 标示符的定义、特点及命名

Java 语言中，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。**标识符由任意顺序的字母大小写，数字，下划线‘\_’，美元符$ 组成，**Java语言中标识符中的字母是严格区分大小写的。Java语言使用Unicode标准字符集，最多可以标识65535个字符，因此Java语言中的字母不仅包括通常的拉丁文字a、b、c等，还包括汉字、日文以及其他许多语言中的文字。

Java 标识符有如下命名规则：

* **标识符应以字母、下划线 、美元符开头**【下面的标识符是合法的：myName，My\_name、Points，$points,\_sys\_ta，OK，\_23b，\_3\_下面的标识符是非法的：

#name，25name，class，&time，if】。

* 标识符不能是关键字。

注意：Java 标识符选取应注意“见名知意”且不能与 Java 语言的关键字重名。Java中严格区分大小写。如good和Good是不同的两个标识符。标识符中不能有空格；

### 2.2.2 Java中标识符的名称规范-驼峰命名

有很多时候提到命名规范，我们会联想到命名规约，其实这两个是两个概念，前者要求该类型名字该怎么命名为好，后者怎一种编程规范建立在命名规范之上的命名细化。Java中的命名规范遵循驼峰法则，下面我们来看一下。

* 包名：多单词组成时所有字母都小写。xxx.yyy.zzz   例如：cn.itcast.student,通常是域名反转
* 类名接口名：多单词组成时，所有单词的首字母大写。XxxYyyZzz
* 变量名和函数名：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、

### 2.2.3 命名规约

代码元素包括类、方法、参数、常量、变量等程序中的各种要素。合适的命名可以体现出元素的特征、职责，以及元素之间的差异性和协同性。为了统一代码风格，元素的命名要遵守以下约定。

#### 2.2.3.1 命名符合本语言特性

当前主流的编程语言有50种左右，分为两大阵营----面向对象和面向过程，但是按照变量定义和赋值的要求，分为强类型语言和弱类型语言（Java\_001中有介绍）。每种语言都有自己的独特命名风格，有些语言在定义时提倡以前缀来区分局部变量、全局变量、控件类型。比如li\_count表示local int局部整型变量，dw\_report表示data window用于展示报表数据的控件。有些语言规定以下划线为前缀来进行命名。这些语言的命名风格，自成一派，也无可厚非，但是在同一种语言中，如果使用多种语言的命名风格，就会引起其他开发工程师的反感。比如，在Java中，所有代码元素的命名均不能以下画线或美元符号开始或结束。

#### 2.2.3.2 命名体现代码元素特性

命名上可体现出代码元素的特性，仅从名字上即可知道代码元素是什么，有利于快速沥青代码脉络。面向对象代码元素的命名行驶分为两大类，即首字母大写的UpperCamelCase和首字母小写的lowerCamelCase，前者俗称大驼峰，后者俗称小驼峰。类名采用大驼峰形式，一般为动词，与参数组成动宾结构，例如Object的wait（）、StringBuffer的append（String）、FileInputStream的read（）等。变量包括参数、成员变量、局部变量等，也采用小驼峰形式。常量的命名方式比较特殊，字母全部大写、单词之间用下划线连接。常量和变量是最基本的代码元素，就像血液中的红细胞一样无处不在。合理的命名，有利于保障代码机体的清爽、健康。

在命名时若能体现出元素的特征，则有助于快速识别命名对象的作用，有助于快速理解程序逻辑。我们推荐在Java命名时，以下列方式体现元素特征：

* 包名统一使用小写，点分隔符之间有且仅有一个自然语义的英文单词。包名统一使用单数形式，但是类名如果有复数含义，则可以使用复数形式。
* 抽象类命名使用Abstract或Base开头；异常类命名使用Exception结尾；测试类命名以它要测试的类名开始，以Test结尾。
* 类型与中括号紧挨相连来定义数组。
* 枚举类名带上Enum后缀，枚举成员名称需要全大写，单词间用下划线隔开。

#### 2.2.3.3 命名最好望文知意

望文知意是以在不需要额外解释的情况下，仅从名称上就能够理解某个词句的确切含义。在代码元素命名时做到望文知意，从而减少注释内容，达到自解释的目的。在实践中，望文知意的难度是最大的，就好像给孩子起名字一样需要反复斟酌的。文不对题的命名方式，肯定会加大理解成本，更大的罪过是把程序员引导到衣蛾错误的理解方向上。某些不规范的缩写会导致劣迹成本增加，比如condition缩写成condi，类似随意的缩写会严重降低代码的可理解性。再比如，以单个字母命名的变量，在上下文理解时，会带来很大的困扰。

主流的编程语言基本上以英语为基础，此处望文知意的“意”指的就是英文。随着开源社区的发展和繁荣，各国程序员踊跃参与开源项目的共建，国际交流与合作越来越频繁，英语能力已经成为程序员必备的技能之一。虽然有人认为命名方式应该符合本国语言习惯，拼音这种命名方式，应该是被允许的，但是在国际化项目或开源项目中，对于非汉语国家的开发工程师而言，拼音这种命名方式的可读性几乎为零。即使在汉语系国家，拼音也存在地区差异。中英文混合的方式，更不应该出现，比如在某业务代码中，曾经出现过DaZePromotion，踩了很久才被命名者告知是打折促销的类。最让人无法容忍的是拼音“首字母”简写的命名方式，即使发挥极致的想象力，也很难猜出具体的含义，比如PfmxBuilder，名称以上是评分模型的创建工厂类！这些命名方式，极大增加了程序的理解成本。所以，正确的英文拼写和语法可以让阅读者抑郁理解，避免歧义。Alibaba、taobao、hangzhou等国际通用的名称，可视同英文。某些复合语义的情况下，尽量使用完整的单词组合来达到望文知意的目的，比如KeyboardShortcutsHandler、AtomicReferenceFieldUpdater。

命名要符合语言特性、体现元素特征。命名做到望文知意、自解释是每个开发工程师的基本素质之一。我们在思量更好地代码元素命名的同时，也要干预修改已有的不合理的命名方式。

在所有代码元素中，常量和变量最为常见，优雅的定义与使用好它们，是开发工程师的基本功之一，下面我么你就来看看，命名规约在这两方面的要求。

#### 2.2.3.4 常量

#### 2.2.3.5 变量

# 变量与常量

在程序执行过程中，其值不能被改变的量称为常量，其值能被改变的量称为变量。变量与常量的命名都必须使用合法的标识符。

## 1 常量

在程序运行过程中一直不会改变的量称为常量（constant），在Java中利用关键字final指示常量,所以常量通常也被称为“final变量”。关键字final表示这个变量只能被赋值一次。一旦被赋值之后，就不能够再更改了。习惯上，常量名使用全大写。在为所有的对象共享值时，常量是非常有用的（如数学中的一些圆周率等）。常量通常会被修饰为final static，final意味着这是不能修改的值，static是静态，被static修饰的值可以直接通过类名调用，减少了创建对象在堆空间的内存分配，减轻了堆空间的压力。

### 1.1 常量的声明以及注意事项

在Java语言中声明一个常量，除了要指定数据类型外，还需要通过final关键字进行限定。声明常量的标准语法如下：

final 数据类型 常量名称[=值]

常量名通常使用大写字母，但这不是必须的。很多Java程序员使用大写字母表示常量，是为了清楚地表明正在使用的常量。但是用大写字母表示常量已经成为一种行业共识了。

当定义的final变量属于“成员变量”时，必须在定义时就设定他的初值，否则将会产生编译错误。当常量要被静态方法引用时，那么必须将常量静态化。

注意：在静态方法中不能调用非静态的方法和引用非静态的成员变量？如何理解？

答：因为static修饰的方法在装载class的时候首先加载初始化，比构造方法早，此时非static的属性和方法还没有完成初始化，故不能调用。若需调用要先静态化。

### 1.2 常量的分类

Java中常量主要分为两大类：字面值常量和引用常量

字面值常量又下面几种：

* 整数常量：所有整数，无穷多，正整数、负整数，例如：1、2、0等；对于整数有四种表现形式
* 小数常量：包含所有小数，也叫浮点数，例如：1.2、2.0、等；
* 布尔型常量：较为特殊的常量只包含两个数值（true\false）；
* 字符常量：将一个字符或符号用‘ ’标识；
* 字符串常量：将一个或多个字符用“ ”标识；
* null常量：只有一个数值就是null；

引用常量指的是由final修饰的引用对象，这里一定要注意，final修饰的引用，这个引用不能再指向其它对象，但是指向的对象本身状态确实可以发生改变的，如下代码所示：

|  |  |
| --- | --- |
| **package** cn.yorickyou.variable;  **public** **class** Test {  **private** **int** x;  **final** **int** y = 10;  **public** **void** get() {  System.***out***.println("y = " + y);  System.***out***.println("x = " + x);  }  **public** **void** set() {  x = y;  }  **public** **static** **void** main(String[] args) {  **final** Test test = **new** Test();  Test test2 = **new** Test();  test.get();  test.set();  test.get();  /\*\*\* The final local variable test cannot be assigned.  \* It must be blank and not using a compound assignment \*/  // test = test2;  }  } | 如代码所示：虽然使用final修饰了引用对象Test，但是test自身的状态还是可以改变的，只是引用test不能再指向别的引用了。 |

【例4.13】创建一个类ConstantPractice，被对其进行一些验证；

class ConstantPractice{

    static final double dou = 23.23;    //定义的final变量属于“成员变量”时，必须在定义时就设定他的初值，当它会在静态方法中被引用时,那么必须让它成为静态常量,否则将会产生编译错误"无法从静态上下文中引用非静态"

    final boolean boo = true ;

    public static void main(String[] args){

        final int a ;

        int s = 34 ;

        a = 23 ;

        // a = 23 ; //当我们已经声明一个常量并对其赋值,那么再次赋值会出现错误"可能已分配变量"

        System.out.println(a);

        System.out.println(dou);

    }

}

Image

### 1.3 常量中的注意事项

常量和变量有一点不同，变量可以在运行时赋值，所以成员字段可以未经初始化就被使用，但是常量不能在运行时改变，所以常量应该在声明时就初始化，否则编译不通过。

方法的参数和局部变量也都可以声明为final的，声明为final的参数在方法中不能改变，局部变量也如此。

声明为final的引用需要注意，引用不能再改变指向，但是指向的对象状态却是可以且可能发生改变的。

在Java中,经常希望某个变量可以在一个类中的多个方法使用,通常将这些常量称为类常量。可以使用关键字static final设置一个类常量。

要注意，Java中存在关键字const，但迄今为止这个关键字尚未起到作用，所以一些学习C++开发的要注意，在Java中修饰常量的是final。

## 2 变量

在程序开发中数据大多时候是以变量的形式存在的，定义和使用变量是一个开发人员最基本的技能了，下面我们来详细说说变量的相关内容。

### 2.1 变量的原理和声明

变量的使用是程序设计中一个十分重要的环节。定义变量就是要**告诉编译器（compiler）这个变量的数据类型，这样编译器才知道需要配置多少空间给它，以及它能存放什么样的数据**。在程序运行时空间内的值是变化的，这个内存空间就称为变量。为了便于操作给这个空间取个名字，称为变量名。变量的命名必须是合法的标识符。内存空间内的值就是变量值。在声明变量时可以是没有赋值，也可以是直接赋给初值。

从本质上讲，变量其实是内存中的一小块区域，使用变量名来访问这块区域，因此，每一个变量使用前必须要先申请（声明），然后必须进行赋值（填充内容），才能使用。简单来说我么可以将其总结为下面这几点：

|  |  |
| --- | --- |
| 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） 该区域的数据可以在同一类型范围内不断变化 |
| 为什么要定义变量 | 用来不断的存放同一类型的常量，并可以重复使用 |
| 使用变量注意 | 变量的作用范围（一对{}之间有效）    初始化值（成员变量不赋值是有默认值的，但局部变量必须初始化才能使用）  由于Java是强类型语言,所以是区分大小写的,如int A,a代表的是两个不同的变量. |
| 变量的声明（注：格式是固定的，记住格式，以不变应万变。） | 数据类型    变量名  =  初始化值；  由于声明是一条完整的Java语句,所以每个声明必须以分号结束，但在一条声明语句中可以同时声明多个相同类型的变量，但我们仍然提倡逐一声明每一个变量，这样可以提高程序的可读性。  变量名的长度基本上是没有限制的。 |
| 变量的初始化 | 成员变量是可以只声明不初始化的，这是由于类在加载的时候会对成员变量进行一次默认初始化，但局部变量不存在这种情况，所以在声明的时候必须初始化，但声明和初始化可以分为两条语句进行，成员变量的声明和赋值必须在一条语句中进行，否则编译不能通过。  在Java中变量的声明应尽可能的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。  Java中变量的声明和定义是一个含义指的是类型和变量名，初始化指的才是赋值操作，两个在一起是一条完整的声明赋值语句 |
|  | 理解：变量就如同数学中的未知数。 |

【例4.14】创建一个VariablePractice类，并进行一些操作，进而分析。

class VariablePractice{

    static int age ;        //当变量要被静态方法引用时,变量要静态化

    static char char1 = 'r' ;

    public static void main(String[] args){

        age = 78 ;

        int 年龄 = 26 ;

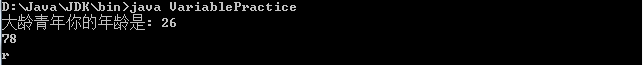
        System.out.println("大龄青年你的年龄是: "+年龄);

        System.out.println(age);

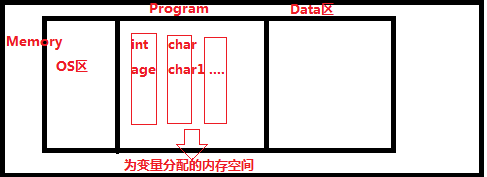
        System.out.println(char1);

    }

}



以上程序代码的内存状态如下图所示，系统的内存大略分为3个区域，即系统区（OS）、程序区（Program）和数据区（Data）。当程序执行时，程序代码会加载到内存中的程序区，数据暂时存储在数据区中。



假设上面两个变量定义在方法体中，则程序加载到程序区中。当执行此行程序代码时，会在栈中本地变量表加载存储这两个变量进行操作。

### 2.2变量的命名

变量虽然是由程序开发者所命名的，但是变量的命名却不是任意的，需要遵循一定的规则。Java中变量的命名规则如下：

* 变量名必须是一个有效的标识符。变量名必须使用Java语言中合法的标识符，即以字母、数字和下划线组成，且首字母不能是数字，还不可以使用关键字。
* 程序运行的生命周期中不能存在两个相同的变量名。如果存在两个相同的变量名，那么程序在运行时就不知道具体调用哪个变量了。
* 变量名称应该见名知意。命名最好能够通过变量名知道变量的内容，增加可读性；方便维护。
* 在Java语言中允许使用汉字或其他语言文字作为变量名，如“int 年龄 = 21”，在程序运行时并不会出错，但还是建议不要这么命名，最好使用英文单词，且遵循见名知意。
* 变量名实际上只是一个指针指向内存中变量的那块内存区域。

#### **2.3 变量的有效范围---成员变量、局部变量**

在程序运行中，变量被加载初始化后存放在内存中，等到程序执行到某一个点，该变量会被释放掉，也就是说变量有它的生命周期。因此，变量的有效范围是指程序代码能够访问该变量的区域，若超出该区域则在编译时出现错误（不能被识别的变量）。在程序中一般根据变量的“有效范围”将变量分为“成员变量”和“局部变量”。

##### **成员变量**

在类中方法或代码块外被定义的变量称为成员变量，成员变量在整个类（实例）中都有效。类的成员变量又可分为两种，即静态变量（类成员）和实例变量（实例成员）。下面我们通过一段代码来了解一下。

【例4.15】声明静态变量和实例变量，示例代码如下：

class var{

    int x = 45 ;

    static int y = 90 ;

}

其中，x为实例变量，y为静态变量（也称类变量）。如果在成员变量的类型前面加上关键字static，这样的成员变量称为静态变量。静态变量的有效范围可以跨类，甚至可以达到整个应用程序之内（关键在于封装程度即使用什么修饰符）。对于静态变量，除了能在定义它的类中使用，还能直接以“类名.静态变量”的方式在其他类中使用。

##### **局部变量**

在类的方法体或代码块中定义的变量（包括方法参数）均称为局部变量。局部变量只在当前定义变量的代码块中有效。通俗的理解就是在其所在的一对大括号中有效，出了这个大括号就没有效了，超出其生命周期调用编译器会提示“不能识别的变量”。局部变量的生命周期取决于方法或代码块，当方法或代码块被调用被加载时，Java虚拟机对局部变量进行内存分配和加载操作，当对局部变量的操作结束后，则会释放局部变量占用的内存空间，局部变量也将会被销毁。

局部变量可与成员变量的名字相同，此时遵循就近原则成员变量将被隐藏，即这个成员变量在此方法中暂时失效。举例证明如下：

【例4.16】在VariablePractice类中我们进行如下测试

class VariablePractice{

    static int age ;        //当变量要被静态方法引用时,变量要静态化

    static char char1 = 'r' ;

    final int test = 99 ;

    public static void main(String[] args){

        age = 78 ;

        int 年龄 = 26 ;

        int test = 87 ;

        System.out.println("大龄青年你的年龄是: "+年龄);

        System.out.println(age);

        System.out.println(char1);

        System.out.print(test);

    }

}

|  |  |
| --- | --- |
| Image  如上我们发现当存在一个成员变量时，我们在定义一个局部变量，且它们名字相同，这时在该局部变量所在方法中，成员变量暂时失效。成员变量和局部变量都有各自的有效范围： | Image |