计算机等级二级java变量 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/138/2021_2022__E8_AE_A1_E7_AE_97_E6_9C_BA_E7_c97_138595.htm 变量是Java 程序的一个基本存储单元。变量由一个标识符，类型及一个可选初始值的组合定义。此外，所有的变量都有一个作用域，定义变量的可见性，生存期。接下来讨论变量的这些元素。 3.8.1 声明一个变量 在Java 中，所有的变量必须先声明再使用。基本的变量声明方法如下：type identifier [= value][，identifier [= value] ...] . type 是Java 的基本类型之一，或类及接口类型的名字（类和接口类型在本书第1部分的后部讨论）。标识符（identifier ）是变量的名字，指定一个等号和一个值来初始化变量。请记住初始化表达式必须产生与指定的变量类型一样（或兼容）的变量。声明指定类型的多个变量时，使用逗号将各变量分开。 以下是几个各种变量声明的例子。注意有一些包括了初始化。 int a, b, c. // declares three ints, a, b, and c.int d = 3, e, f = 5. // declares three more ints, initializing // d and f.byte z = 22. // initializes z. double pi = 3.14159. // declares an approximation of pi.char x = ’x’. // the variable x has the value ’x’. 你选择的标识符名称没有任何表明它们类型的东西。许多读者记得FORTRAN 预先规定从I到N的所有标识符都为整型，其他的标识符为实型。Java 允许任何合法的标识符具有任何它们声明的类型。 3.8.2 动态初始化 尽管前面的例子仅将字面量作为其初始值，Java 也允许在变量声明时使用任何有效的表达式来动态地初始化变量。 例如，下面的短程序在给定直角三角形两个直角边长度的情况下，求其斜边长度。

// Demonstrate dynamic initialization. class DynInit { public static void main(String args[]) { double a = 3.0, b = 4.0. // c is dynamically initialized double c = Math.sqrt(a * a b * b). System.out.println("Hypotenuse is " c). } } 这里，定义了3个局部变量a，b，c。前两个变量a和b初始化为常量。然而直角三角形的斜边c被动态地初始化（使用勾股定理）。该程序用了Java 另外一个内置的方法sqrt()，它是Math类的一个成员，计算它的参数的平方根。这里关键的一点是初始化表达式可以使用任何有效的元素，包括方法调用、其他变量或字面量。 3.8.3 变量的作用域和生存期 到目前为止，我们使用的所有变量都是在方法main() 的后面被声明。然而，Java 允许变量在任何程序块内被声明。在第2章中已解释过了，程序块被包括在一对大括号中。一个程序块定义了一个作用域（scope ）。这样，你每次开始一个新块，你就创建了一个新的作用域。你可能从先前的编程经验知道，一个作用域决定了哪些对象对程序的其他部分是可见的，它也决定了这些对象的生存期。 大多数其他计算机语言定义了两大类作用域：全局和局部。然而，这些传统型的作用域不适合Java 的严格的面向对象的模型。当然将一个变量定义为全局变量是可行的，但这是例外而不是规则。在Java 中2个主要的作用域是通过类和方法定义的。尽管类的作用域和方法的作用域的区别有点人为划定。因为类的作用域有若干独特的特点和属性，而且这些特点和属性不能应用到方法定义的作用域，这些差别还是很有意义的。因为有差别，类（以及在其内定义的变量）的作用域将被推迟到第6章当讨论类时再来讨论。到现在为止，我们将仅仅考虑由方法或在一个方法内定义的作用域。 方法定

义的作用域以它的左大括号开始。但是，如果该方法有参数，那么它们也被包括在该方法的作用域中。本书在第5章将进一步讨论参数，因此，现在可认为它们与方法中其他变量的作用域一样。 作为一个通用规则，在一个作用域中定义的变量对于该作用域外的程序是不可见（即访问）的。因此，当你在一个作用域中定义一个变量时，你就将该变量局部化并且保护它不被非授权访问和/或修改。实际上，作用域规则为封装提供了基础。 作用域可以进行嵌套。例如每次当你创建一个程序块，你就创建了一个新的嵌套的作用域。这样，外面的作用域包含内部的作用域。这意味着外部作用域定义的对象对于内部作用域中的程序是可见的。但是，反过来就是错误的。内部作用域定义的对象对于外部是不可见的。 为理解嵌套作用域的效果，考虑下面的程序： // Demonstrate block scope. class Scope { public static void main(String args[]) { int x. // known to all code within main x = 10. if(x == 10) { // start new scope