Java 语言中变量和方法的分析及其应用①

贺　军，李喜梅

(怀化职业技术学院 计算机与信息工程系，怀化 418000)

摘　要：详细分类说明了变量和方法的概念，分析了变量和方法在 java 程序设计应用过程中的内存分配问题、在继承时的覆盖问题以及静态变量和静态方法的应用问题，并给出了部分典型案例。本文的研究可以提供 java 程序开发人员在程序设计过程中恰当的使用变量和方法作为参考，保证开发出来的程序简洁和高效。

关键词：java 语言；变量；方法；覆盖；隐藏

Analysis and Application of Variables and Methods in Java Language

HE Jun, LI Xi-Mei

(Computer and Information Engineering Department, Huaihua Vocational and Technical College, Huaihua 418000, China)

Abstract: This paper makes a detailed classification and illustrates the concept of variables and method, analyzes the variables and methods of memory allocation problem in Java programming design application process, the covering problems in the inheritance and the application problems of static variables and static methods. It also sets partial typical examples. This research can provide Java program developers appropriate use of variables and methods as reference in programming design process, and ensure the developed program is concise and high efficiency.

Key words: Java language; variable; method; cover; hidden

1　引言

在程序运行时，有些数据的值会被改变，在使用前必须被临时存储，这就需要用于标识数据的存储单元，也就是变量。方法是描述实现某个特定功能所需的数据及进行的运算和操作。变量和方法是编程语言中非常重要的概念，对于 java 语言当然也不例外。由于 java 是面向对象的编程语言，java 的变量分为：局部变量、类变量（静态变量）和实例变量，java 的方法分为：类方法（静态方法）和实例方法。在学习 Java 语言和使用 Java 语言开发程序的过程中，只有了解和掌握好它们各自不同的特点和适用环境，我们才能充分利用好 java 这个工具，开发出性能更良好的系统。

2　变量

java 的变量有局部变量、类变量（静态变量）、实例变量和参数变量（方法参数、构造函数参数和异常处理参数）四种[1]。

（1）局部变量是定义在块内、方法体内的变量。这种变量的作用域是以块和方法为单位的，仅在定义该变量的块内或方法体内有效，而且要先定义赋值，然后再使用，即不允许超前使用。局部变量在方法每次被调用时重新初始化，与上次的调用无关。

（2）类变量是定义在类内或接口内、方法外且用 static 修饰符修饰的变量。类变量又称为静态变量。类变量是类的变量，不属于任何一个类的具体实例对象。不是保存在某个对象的内存空间中，而是保存在类的内存区域的公共存储单元中。换句话说：对于类的任何一个具体对象而言，类变量都是一个公共的存储单元，任何一个类的对象访问它，取到的都是相同的数值；同样，任何一个类的对象去修改它，也都是对同一内存单元进行操作。

类变量可以通过类名直接访问，也可以通过对象来引用，两种方法的结果是相同的。

（3）实例变量定义在类内、方法外但不用 static

① 收稿时间:2011-01-09;收到修改稿时间:2011-02-16

修饰符修饰的变量。实例变量是属于类创建的具体对象的。保存在具体对象的内存区域的存储单元中，对于不同的对象来说，实例变量的存储单元是不同的内存单元。同样，一个类的不同的对象去修改属于本对象的实例变量也是针对不同的内存单元进行操作的。实例变量只能通过对象来引用。

（4）参数变量分为方法参数、构造函数参数和异常处理参数。方法参数是用来传入方法体的，构造函数参数是用来传入构造函数的，异常处理参数是用来传入一个 try 语句中的 catch 块的[2]。

因为局部变量在查找时首先被查找，因此若某一局部变量与类变量或实例变量同名时，则该类变量或实例变量在方法体内（或程序块内）被暂时“屏蔽”起来，只有退出这个方法（或程序块内）后，类变量或实例变量才起作用。

局部变量、类变量和实例变量的应用和区别。请参看如下的例子。

```
class MyObject
{
static short s=400;//类变量，可以用类名和具体对象名两种方法来引用
int i=200;//实例变量，只能用具体的对象来引用
void f()
{
  System.out.println("s="+s);
  System.out.println("i="+i);
  short s=300;//局部变量与类变量同名，在方法体内，屏蔽类变量
  int i=100;//局部变量与实例变量同名，在方法体内，屏蔽实例变量
  double d=1E100;//局部变量
  System.out.println("方法体内局部变量 s="+s);
  System.out.println("方法体内局部变量 i="+i);
  System.out.println("d="+d);
  }
}
public class variables
{
  public static void main(String args[])
  {
  System.out.println("用类名 MyObject 调用的类变量 s="+MyObject.s);
  /* System.out.println("用类名 MyObject 调用实例变量 i="+MyObject.i);
  //这种书写方法是错误的，因为实例变量 i 只能通过对象来引用。
  */
  MyObject myobject=new MyObject();
  //用具体对象 myobject 来引用类变量 s
  System.out.println("用具体对象 myobject 调用的类变量 s="+myobject.s);
  //用具体对象 myobject 来引用实例变量 i，是正确的
  System.out.println("用具体对象 myobject 调用的实例变量 i="+myobject.i);
  myobject.f();//调用方法
  }
}
```

程序的运行结果：

```
用类名 MyObject 调用的类变量 s=400
用具体对象 myobject 调用的类变量 s=400
用具体对象 myobject 调用的实例变量 i=200
s=400
i=200
方法体内局部变量 s=300
方法体内局部变量 i=100
d=1.0E100
```

3 方法

java 中的方法有类方法（静态方法）和实例方法。

（1）类方法是定义在类中用 static 修饰并且属于整个类的方法。类方法可以使用类名直接调用，也可以使用某一个具体的对象名来调用。类方法只能处理类变量或调用类方法，类方法中不能访问实例变量和实例方法。如果类方法被允许处理实例变量和实例方法，当类方法被调用而实例不存在时，显而易见程序运行将会出现错误。

（2）实例方法是定义在类中不用 static 修饰属于某一个具体对象的方法。实例方法只能用某一个具体的对象名来调用。实例方法可以访问实例变量和实例方法，也可以处理类变量和调用类方法。

类方法和实例方法的应用和区别。请参看如下的

例子。

```
class A
{
  int i;                    //实例变量
  static int j;          //类变量
  void seti(int x)
  {
    i=x;
    System.out.println("i="+i);
    setj(45);
  }        //此处在实例方法 seti()中调用类方法 setj()，正确
  static void setj(int y)
  {
    j=y;
    System.out.println("j="+j);
    /*
    i=y;  //此处错误，类方法中不能处理实例变量
    seti(y);//此处错误，类方法中不能调用实例方法
    */
  }
  public static void main(String args[])
  {
      A a1=new A();   //创建类对象 a1
      A.j=2;           //用类名为类变量赋值
      a1.j=3;          //用对象名为类变量赋值
      A.setj(3);       //用类名直接调用类方法
      a1.i=4;          //用对象名为实例变量赋值
      a1.seti(4);      //用对象名调用实例方法
  }
}
```

程序的运行结果：

```
j=3
i=4
j=45
```

4 变量和方法的应用

4.1 变量和方法内存中的分配

(1) 类变量：在程序加载时系统就为它在堆中开辟了内存，堆中的内存地址存放于栈以便于高速访问。类变量的生命周期一直持续到整个“系统”关闭。

(2) 实例变量：当你使用 java 关键字 new 的时候，系统在堆中开辟并不一定是连续的空间分配给实例变量，然后根据零散的堆内存地址，通过哈希算法换算为一长串数字以表征这个变量在堆中的“物理位置”。实例变量的生命周期，当实例变量的引用丢失后，将被 GC（垃圾回收器）列入可回收“名单”中，但并不是马上就释放堆中内存。

(3) 局部变量：局部变量，由声明在某方法，或某代码段里（比如 for 循环），执行到它的时候在栈中开辟内存，当局部变量一但脱离作用域，内存立即释放。

(4) 方法：方法存储在栈区，生命周期只限于方法被放在栈上的这段时间（也就是方法调用直至执行完毕为止）。

4.2 变量和方法在继承时的覆盖

在类的继承中，如果子类新增的成员名与父类成员相同，则称为成员覆盖或成员隐藏。在子类中定义与父类同名成员的目的，是用来修改父类的属性和行为的。

(1) this 的使用

一个对象中的方法一般可以直接访问同一对象的成员变量。但是，有时候方法体内部定义的变量和成员变量名字相同，还有时方法的入口参数和对象的成员变量名字相同，那么就需要将三者区别清楚。因此，专门用 this 来指明当前对象的成员变量或当前对象的方法。由于 this 代表类的某个实例，所以静态成员不能用 this 来引用，静态成员可以通过类名前缀来引用[3]。

(2) super 的使用

在 Java 中，由父类派生子类，这样，子类的成员变量可能和父类的成员变量名字相同，子类的方法也可能和父类的方法一样。当需要调用父类的同名方法或使用父类的同名变量时，在子类中可用关键字 super 作前缀来指明父类的成员变量和方法。需要注意的是，super 不能用于静态方法中，在静态方法中，父类的静态成员可以通过类名前缀来引用[4]。

(3) 变量和方法的覆盖的使用

java 语言程序设计中，实例变量和类变量能被隐藏，被子类的同名变量成员隐藏；局部变量和各种参数永远不会被隐藏；实例方法被覆盖，静态方法被隐藏[2]。

请参看如下的例子。

```
class Base
{
   int x = 1;          //实例变量
   static int y=2;   //类变量
   int z=3;            //实例变量
   int method()     //实例方法
   {
   return x;
   }
   static String greeting()   //类方法
   {
   return "Goodnight";
   }
}
class Subclass extends Base
{
   int x = 4;          //与父类同名的实例变量
   int y=5;            //与父类同名的实例变量
   static int z=6;   //与父类同名的类变量
   int method()     //与父类同名的实例方法
   {
   return x;
   }
   static String greeting()   //与父类同名的类方法
   {
   return "Hello";
   }
}
public class Test
{
   public static void main(String[] args)
   {
   Subclass s=new Subclass();
   System.out.println(s.x + " " + s.y +" "+ s.z);     //隐藏与父类同名的变量
   System.out.println(s.method());          //使用子类中的实例方法，覆盖了父类中同名的实例方法
   System.out.println(s. greeting ());      //使用子类中的类方法，隐藏了父类中同名的类方法
   Base b =(Subclass)s;
   System.out.println(b.x + " " + b.y +" "+ b.z);   //使用父类中的变量
   System.out.println(b.method()); //使用子类中的实例方法，覆盖了父类中同名的实例方法
   System.out.println(b. greeting ());   //使用父类中的类方法
   }
}
```

程序的运行结果：

```
4  5  6
4
Hello
1  2  3
4
Goodnight
```

4.3 静态变量和静态方法的应用

java 语言中没有全局变量和全局函数的概念，但静态变量在 java 程序中充当了全局变量的角色，同样，静态方法类似于全局函数[5]。java 程序中应该什么时候用静态成员什么时候用非静态成员呢？这要看情况而定，比如，在 java API 的 Math 类中，几乎所有变量和方法都是静态的，而 System 类中大部分方法也是静态的。因为这样的类，通用性很强，重用性很高，而且关键的一点是改动的可能性很小，这种时候比较适合用静态变量和静态方法，这样节省了初始化所消耗的资源，也方便使用。比如 System 类中的 out 方法，使用时不必要在每次输出的时候创建一个 System 类的实例而消耗更多资源。因此，在 java 程序设计中应该将一些比较常用而且不会随他的实例而变化的变量和方法设置成静态的。尽管静态成员的使用有很多优点，但也不是所有的成员都适合设置成静态成员。比如，当某些类的通用性不强时不太适合使用静态成员，因为静态成员会在整个类的生命周期中一直存在从而影响系统的性能。

请参看如下的例子。

```
class circle
{
   int r;
   public circle (int r)
   {
      this.r=r;
      }
```

```
public int getr()
{
return r;
}          //返回当前圆的半径，根据每一个实例的不同，返回不同的值。所以此时不能使用静态方法，只能使用实例方法
}
```

5 结语

变量和方法是 java 语言中重要的概念，恰当的使用变量和方法可以保证程序的简洁和高效。本文详细讨论了变量和方法的分类及其使用情况，并给出了部分实例。希望本文的研究对于 java 程序设计人员准确使用 java 语言中变量和方法具有一定的帮助。

参考文献

1 李占波,姬莉霞,王海玲,欧研.程序设计基础(java 版).北京:中国铁道出版社, 2007.80–81.

2 CAD 教育网.Java 中的方法和变量在继承时的覆盖.北京:http://www.cadedu.com/编程开发.java 语言编程, 2010.02.06.

3 李尊朝,苏军.Java 语言程序设计.第 2 版.北京:中国铁道出版社, 2009.97–98.

4 刘培文.java 程序设计教程.北京:北京科海电子出版社, 2009.88–89.

5 张素珍,耿磊.Java 语言静态变量和静态方法的分析及其应用研究.计算机系统应用, 2006,15(5):84–86.

(上接第 227 页)

2 De Bra PME, Post RDJ. Information retrieval in the World Wide Web: Making client-based searching feasible. Proc. 1st International World Wide Web Conference (Geneva), 1994.

3 Hersovici M, Jacovi M, Maarek YS, et al. The shark-search algorithm An application: Tailored Web site mapping. Proc. 7th Intl. World-Wide Web Conference, 1998.

4 Chakrabarti S, van den Berg M, Dom B. Focused crawling: A new approach to topic-specific Web resource discovery. Computer Networks 1999,31:1623– 1640.

5 Aggarwal CC, Al-Garawi F, Yu PS. Intelligent crawling on the World Wide Web with arbitrary predicates. Proc. 10th International World Wide Web Conference.2001.96–105.

6 Michelangelo Diligenti, Frans Coetzee, Steve Lawrence et al. Focused crawling using context graphs. Proc. Very Large Data Bases 2000 (VLDB 2000).

7 Michael Chau, Hsinchun Chen . Comparison of three vertical search spiders. IEEE Computer, 2003,36(5).

8 Chakrabarti S. Integrating the Document Object Model with hyperlinks for enhanced topic distillation and information extraction. 10th International World Wide Web Conference, 2001.

9 Chen J, Zhou BY, Shi J, et al. Function-Based Object Model Towards Website Adaptation. Proc. of the 10th International World Wide Web Conference, 2001.

10 Zou J, Le D, Thoma GR. Online medical journal article layout analysi. Proc. SPIE, 2007, 6500: 1–12.

11 Cai D, Yu SP, Wen JR, et al. VIPS: A Vision-based Page Segmentation Algorithm. Microsoft Technical Report, 2003, MSR-TR-2003–79.