JAVA 语言静态变量和静态方法的分析及其应用研究

An study on JAVA's static variables and methods

张素珍 （石家庄邮电职业技术学院 050021）
耿 磊 （石家庄市信息管理中心 050021）

摘要：本文说明了静态变量和方法的概念，并与非静态变量和方法进行比较说明静态变量的初始化、引用和内存分配等问题，并讨论了静态方法调用静态数据的问题。最后给出了如何使用静态变量和方法来简化程序、提高效率。

关键词：JAVA 静态变量 静态方法

1 引言

JAVA 语言已经与 Internet 的在线环境密不可分，但它首先是一种编程语言。而变量是编程语言中非常重要的概念，对于 JAVA 当然也不例外。由于 JAVA 是完全面向对象的，这个特点使得 JAVA 的变量可以分为类变量和实例变量两大类，即静态变量和非静态变量，同样地，方法也分成静态方法和非静态方法。了解和掌握它们各自不同的特点和适用环境，才能充分利用好 JAVA 这个工具，开发出性能更良好的系统。

2 静态变量

类是组成 JAVA 程序的基本要素，它有两种基本成分：变量和方法，称为成员变量和成员方法。成员变量的声明格式如下：

[public][protected|private][static][final][transient][volatile] type variableName;

其中，关键词 static 是用来限定该成员变量为类变量，也就是静态变量，而没有用 static 修饰的成员变量则是实例变量，即非静态变量。静态变量和非静态变量的本质虽然都是 JAVA 程序的基本存储单元，但是它们的使用有很大不同。

（1）当一个类被调用时（比如直接通过类名使用静态变量或者用 new() 方法创建一个实例），它首先被装载，这个时候静态变量会被初始化，而且以后不会再被初始化，而非静态变量只有在类创建实例的时候进行初始化，并且每创建一个实例都会进行一次初始化。具体情况看参看如下的例子：

```
class B
{
static int staticVarB;
}
class C
{
static int staticVarC;
int noStaticVarC;
}
class A
{
public static void main(String args[])
{
System.out.println(B.staticVarB);//类 B 被装载（未创建实例），静态变量 staticVarB 被初始化
C c1=new C();//类 C 被装载，静态变量 staticVarC 和非静态变量 noStaticVarC 都被初始化
System.out.println(C.staticVarC);
System.out.println(c1.noStaticVarC);
}
}
```

（2）JAVA 允许用特殊的"static 子句"把所有的静态初始化语句都组织起来，又称为静态块，它的形式是这样的：

```
class Example{
```


©《计算机系统应用》编辑部 http://www.c-s-a.org.cn

```
static int intvar;
static String strvar;
static
{
  intvar = 34;
  strvar = "java";
}
}
```

它看上去象一个方法，但实际上只是跟在 static 关键词后面的一段代码。同别的 static 的初始化一样，它只会被执行一次，在第一次创建这个类的对象或是第一次访问这个类的成员的时候。而如果不用 static 关键词来标识而直接写赋值语句，则编译时会认为是非法语句而发生错误。

(3) 如果一个类是其他类的子类，那么它的变量被初始化的顺序仍然会首先考虑静态变量，也就是说，不论静态变量在 JAVA 程序中写在什么位置，都会比实例变量先被初始化。以一个类创建实例的情况为例，具体如下：初始化父类的静态变量 - - - > 初始化子类的静态变量 - - >? 初始化父类的非静态变量 - - - > 初始化父类构造函数 - - - > 初始化子类非静态变量 - - - > 初始化子类构造函数。

(4) 由于初始化的不同，静态变量和非静态变量所占的内存空间和使用范围也不同。静态变量由于只会初始化一次，一个类的所有实例拥有同样的静态变量，所以只占一块内存空间，而且存在于类的整个生命周期中；而非静态变量在每个类的实例中都有一份拷贝，不同的类拥有各自的非静态变量，也就是说类创建了多少对象，非静态变量就占用多少块内存空间，但当一个对象的任务完成，JAVA 的垃圾自动收集器会把这个对象所占的内存空间回收(包括非静态变量所占的空间)，这一份非静态变量也就不存在了。

(5) 静态变量和非静态变量的使用方法也不相同。静态变量可以用通过类名来使用，也可以通过实例来使用，他们都是对同一份静态变量操作。而非静态变量则只能通过实例来使用，每个实例使用各自不同的非静态变量。例如把上例的类 A 改为如下的代码：

```
class A
{
public static void main(String args[])
{
  C c1 = new C();//类 C 被装载，静态变量 staticVarC 和非静态变量 noStaticVarC 都被初始化
  System.out.println(C.staticVarC);//通过类名直接访问静态变量，输出结果是 0
  System.out.println(c1.staticVarC);//通过实例访问静态变量，输出结果也是 0
  c1.staticVarC++;//通过实例把静态变量加 1
  C c2 = new C();
  c2.noStaticVarC++;//通过实例 c2 把非静态变量加 1
  System.out.println(C.staticVarC);//通过类名访问静态变量，输出结果是 1
  System.out.println(c1.staticVarC);// 通过实例 c1 访问静态变量，输出结果是 1
  System.out.println(c2.staticVarC);// 通过实例 c2 访问静态变量，输出结果是 1
  System.out.println(c2.noStaticVarC);// 通过实例 c2 访问静态变量，输出结果是 1
  System.out.println(c2.noStaticVarC);// 通过实例 c1 访问静态变量，输出结果是 0
}
}
```

上述例子说明通过类实例可以对类的静态变量进行操作，其他类实例再访问时会操作已修改的静态变量值，非静态变量则是各类实例独立操作。

另外，非静态变量还可以通过 this 来引用。this 是一个关键词，它只能用于方法内部，代表当前对象的引用。当方法的参数和这个类的数据成员同名时(假设为 data)，这时产生了二义性，为了区分它们就可以用 this.data，来表明要使用类的数据成员。由上述概念可知，this 是代表类的某个实例的，因此，静态变量不能用 this 来引用。

(6) 静态类。通常一个普通类不允许声明为静态的，只有一个内部类才可以声明为静态的，它可以看作是特殊的静态变量。这时这个声明为静态的内部类可以直接作为一个普通类来使用，就是说要创建一个 static 内部类的对象不需要一个外部类对象，因此不能从一个 static 内部类的一个对象访问一个外部类对象。如下代码所示：

©《计算机系统应用》编辑部 http://www.c-s-a.org.cn

```
public class Static Cls{
public static void main(String[ ]args){
    OuterClass. InnerClass outinner = new OuterClass.
InnerClass( ); }
    }
class OuterClass{ public static class InnerClass{ In-
nerClass( ){ System. out. println( " InnerClass" ); } }}
```

它的输出结果是:InnerClass

3 静态方法

静态方法的概念类似于静态变量,是属于类的方法,声明时也用 static 关键词进行修饰。静态方法的引用方法和静态变量也类似,不再过多讨论。下面主要论述静态方法在实际应用中的问题。

(1) 静态方法只能直接处理静态数据。比如

```
class MyClass
{
    int number = 1000;
    static int addNumber( )
    { return number + + ;}
}
```

以上这个类编译会出错,必须把 number 声明为静态变量,即 static int number = 1000;但是如果是如下的程序则不会出错

```
class MyClass
{
    static int number = 1000;
    int addNumber( )
    { return number + + ;}
}
```

这是因为静态方法是整个类公用的方法,可以通过类名直接调用,如果静态方法被允许处理实例变量,当静态方法被调用时而实例不存在时,显而易见程序运行将会出现错误,因此,静态方法只能处理静态数据。但是反过来,实例方法处理静态变量是不会出现上述错误的。

(2) 特殊的静态方法 main( )。在编写 JAVA 程序过程中,总有一个类含有这样的方法 public static void main( ),这是因为 JAVA 程序运行时需要有一个入口,而某一个类的 main 方法就是提供这个功能的。当这个含有 main( )方法的类被装载之后程序才可以启动运行了。而在装载时没有任何类的实例存在,因此实例方法是不可能被调用的,所以,main( )方法必须是类的方法而不属于任何实例,也就是说它是静态方法。

(3) 静态方法与非静态方法如何使用内存。在内存中,一个静态变量只占用一块内存,所有该类的实例都共用这一块内存来保存静态变量,实例变量则不同,每个实例化后的实例都会为实例变量占用一块内存来保存变量。而方法则不同,类方法和实例方法,即静态方法和非静态方法都共用一块内存,每个实例并不会为一个实例方法单独开辟一块空间,这是因为每个方法的 java 语句都是一样的,不同的只是方法中变量的值,这个在编译时利用一个隐藏 this 关键字就可以把不同的值区别开来,所以就没有必要再开空间。这使得静态方法的使用具有了更大的优势。

(4) 子类中创建的静态方法不会覆盖父类中相同名字的静态方法。静态方法是在编译的时候把静态方法和类的引用类型进行匹配,而不是在运行的时候和类引用进行匹配。因此在子类中创建的静态方法,它并不会覆盖父类中相同名字的静态方法。例如以下程序:

```
    class Parent{
    public static void staticMethod( ) {
    System. out. println ( parents static method is
called); }
    }
    class Child extends Parent{
    public static void staticMethod( ) {
    System. out. println ( childs static method is
called); }
    }
public class Test{
    public static void main(String args[ ]) {
    Parent p = new Child( );
    p. staticMethod( );
      }
```

这个程序的输出结果是:parents static method is called。这说明虽然 p 实际上是一个 Child 的类型的引用,然而在调用静态方法的时候,它执行的却是父类的静态方法,而不是 Child 的静态方法。

©《计算机系统应用》编辑部 http://www.c-s-a.org.cn

4 静态变量和静态方法的应用

由以上论述，我们可以暂时这样认为，尽管 JAVA 中并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。这个设计并不是面向对象语言的思想，因为，我们没有办法通过 static 方法向对象发送消息。但是在实际应用中，静态变量和静态方法非常实用，可以使 JAVA 程序编写更加简洁、高效。那么 JAVA 程序中应该什么时候用静态成员什么时候用非静态的成员呢？这要看情况而定，比如，在 java API 的 Math 类中，几乎所有的变量和方法都是静态的，而 System 类中的大部分？方法也是静态方法。因为这样的类，通用性很强，重用性高，而且关键的一点是改动的可能性很小，几乎没有，这种时候比较适合用静态变量和方法，这样节省了初始化所消耗的资源，也便于使用；比如 System 中 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 程序设计中应该一些比较常用而且不会随他的实例而变化的变量和方法可以设置成为静态的。尽管静态成员的使用有很多优点，但也并不是所有的成员都适合，因为设置成为静态成员的代价是失去了灵活性，当每个类的实例对数据成员有自己的要求时，数据成员就不能设置成静态的。以静态方法为例，以下方法不适合设置成静态方法：

```
class Circle
{
  int r;
  public Circle(int r)
  {this.r=r;}
  public int getR()
    { return r; }//返回当前圆的半径，根据每个实例不同，返回值不同
}
```

另外，当某些类的通用性不强时也不太适合使用静态变量和方法，因为静态数据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。

5 结束语

本文中讨论了主要静态变量和方法并给出了实例。静态变量和方法是 JAVA 语言中一个重要的、特殊的特性，解决了 JAVA 语言没有全局变量和函数的问题。静态变量和方法使用的恰当可以保证程序的简洁、高效。

参考文献

1 Bruce Eckel. thinking in java 3rd. 2002

©《计算机系统应用》编辑部 http://www.c-s-a.org.cn