Java static关键字以及Java静态变量和静态方法

static 修饰符能够与变量、方法一起使用，表示是“静态”的。

静态变量和静态方法能够通过类名来访问，不需要创建一个类的对象来访问该类的静态成员，所以static修饰的成员又称作类变量和类方法。静态变量与实例变量不同，实例变量总是通过对象来访问，因为它们的值在对象和对象之间有所不同。

请看下面的例子：

```
01.  public class Demo {
02.      static int i = 10;
03.      int j;
04.
05.      Demo() {
06.          this.j = 20;
07.      }
08.
09.      public static void main(String[] args) {
10.          System.out.println("类变量 i=" + Demo.i);
11.          Demo obj = new Demo();
12.          System.out.println("实例变量 j=" + obj.j);
13.      }
14.  }
```

运行结果：
类变量 i=10
实例变量 j=20

static 的内存分配

静态变量属于类，不属于任何独立的对象，所以无需创建类的实例就可以访问静态变量。之所以会产生这样的结果，是因为编译器只为整个类创建了一个静态变量的副本，也就是只分配一个内存空间，虽然有多个实例，但这些实例共享该内存。实例变量则不同，每创建一个对象，都会分配一次内存空间，不同变量的内存相互独立，互不影响，改变 a 对象的实例变量不会影响 b 对象。

请看下面的代码：

```
01.  public class Demo {
```

```
02.     static int i;
03.     int j;
04.
05.     public static void main(String[] args) {
06.         Demo obj1 = new Demo();
07.         obj1.i = 10;
08.         obj1.j = 20;
09.
10.         Demo obj2 = new Demo();
11.
12.         System.out.println("obj1.i=" + obj1.i + ", obj1.j=" + obj1.j);
13.         System.out.println("obj2.i=" + obj2.i + ", obj2.j=" + obj2.j);
14.     }
15. }
```

运行结果：
obj1.i=10, obj1.j=20
obj2.i=10, obj2.j=0

注意：静态变量虽然也可以通过对象来访问，但是不被提倡，编译器也会产生警告。

上面的代码中，i 是静态变量，通过 obj1 改变 i 的值，会影响到 obj2；j 是实例变量，通过 obj1 改变 j 的值，不会影响到 obj2。这是因为 obj1.i 和 obj2.i 指向同一个内存空间，而 obj1.j 和 obj2.j 指向不同的内存空间，请看下图：

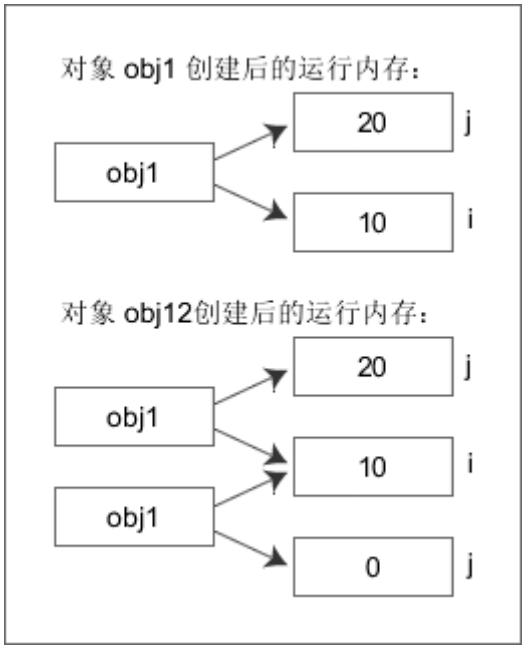

图1 静态变量内存分配

注意：static 的变量是在类装载的时候就会被初始化。也就是说，只要类被装载，不管你是否使用了这个static 变量，它都会被初始化。

小结：类变量(class variables)用关键字 static 修饰，在类加载的时候，分配类变量的内存，以后再生成类的实例对象时，将共享这块内存（类变量），任何一个对象对类变量的修改，都会影响其它对象。外部有两种访问方式：通过对象来访问或通过类名来访问。

静态方法

静态方法是一种不能向对象实施操作的方法。例如，Math 类的 pow() 方法就是一个静态方法，语法为 Math.pow(x, a)，用来计算 x 的 a 次幂，在使用时无需创建任何 Math 对象。

因为静态方法不能操作对象，所以不能在静态方法中访问实例变量，只能访问自身类的静态变量。

以下情形可以使用静态方法：

- 一个方法不需要访问对象状态，其所需参数都是通过显式参数提供（例如 Math.pow()）。
- 一个方法只需要访问类的静态变量。

读者肯定注意到，main() 也是一个静态方法，不对任何对象进行操作。实际上，在程序启动时还没有任何对象，main() 方法是程序的入口，将被执行并创建程序所需的对象。

关于静态变量和静态方法的总结：

- 一个类的静态方法只能访问静态变量；
- 一个类的静态方法不能够直接调用非静态方法；
- 如访问控制权限允许，静态变量和静态方法也可以通过对象来访问，但是不被推荐；
- 静态方法中不存在当前对象，因而不能使用 this，当然也不能使用 super；
- 静态方法不能被非静态方法覆盖；
- 构造方法不允许声明为 static 的；
- 局部变量不能使用static修饰。

静态方法举例：

```
01.  public class Demo {
02.      static int sum(int x, int y){
03.          return x + y;
04.      }
05.
06.      public static void main(String[] args) {
07.          int sum = Demo.sum(10, 10);
08.          System.out.println("10+10=" + sum);
09.      }
10.  }
```

运行结果：
10+10=20

static 方法不需它所属的类的任何实例就会被调用，因此没有 this 值，不能访问实例变量，否则会引起编译错误。

注意：实例变量只能通过对象来访问，不能通过类访问。

静态初始器（静态块）

块是由大括号包围的一段代码。静态初始器(Static Initializer)是一个存在于类中、方法外面的静态块。静态初始器仅仅在类装载的时候（第一次使用类的时候）执行一次，往往用来初始化静态变量。

示例代码：

```
01.  public class Demo {
```

```
02.     public static int i;
03.     static{
04.         i = 10;
05.         System.out.println("Now in static block.");
06.     }
07.     public void test() {
08.         System.out.println("test method: i=" + i);
09.     }
10.
11.     public static void main(String[] args) {
12.         System.out.println("Demo.i=" + Demo.i);
13.         new Demo().test();
14.     }
15. }
```

运行结果是：
Now in static block.
Demo.i=10
test method: i=10

静态导入

静态导入是 Java 5 的新增特性，用来导入类的静态变量和静态方法。

一般我们导入类都这样写：

```
01. import packageName.className;  // 导入某个特定的类
```

或

```
01. import packageName.*;  // 导入包中的所有类
```

而静态导入可以这样写：

```
01. import static packageName.className.methonName;  // 导入某个特定的静态方法
```

或

```
01. import static packageName.className.*;  // 导入类中的所有静态成员
```

导入后，可以在当前类中直接用方法名调用静态方法，不必再用 className.methodName 来访问。

对于使用频繁的静态变量和静态方法，可以将其静态导入。静态导入的好处是可以简化一些操作，例如输出语句 System.out.println(); 中的 out 就是 System 类的静态变量，可以通过 import static java.lang.System.*; 将其导入，下次直接调用 out.println() 就可以了。

请看下面的代码：

```
01. import static java.lang.System.*;
02. import static java.lang.Math.random;
03. public class Demo {
04.     public static void main(String[] args) {
```

```
05.             out.println("产生的一个随机数：" + random());
06.         }
07.     }
```

运行结果：

产生的一个随机数：0.05800891549018705