<https://www.cnblogs.com/dotgua/p/6354151.html?utm_source=itdadao&utm_medium=referral>

在java的关键字中，**static**和**final**是两个我们必须掌握的关键字。不同于其他关键字，他们都有多种用法，而且在一定环境下使用，可以提高程序的运行性能，优化程序的结构。下面我们先来了解一下static关键字及其用法。

**static关键字**

**1.修饰成员变量**

在我们平时的使用当中，static最常用的功能就是修饰类的属性和方法，让他们成为类的成员属性和方法，我们通常将用static修饰的成员称为类成员或者静态成员，这句话挺起来都点奇怪，其实这是相对于对象的属性和方法来说的。请看下面的例子：（未避免程序太过臃肿，暂时不管访问控制）

[复制代码](javascript:void(0);)

public class Person {

String name;

int age;

public String toString() {

return "Name:" + name + ", Age:" + age;

}

public static void main(String[] args) {

Person p1 = new Person();

p1.name = "zhangsan";

p1.age = 10;

Person p2 = new Person();

p2.name = "lisi";

p2.age = 12;

System.out.println(p1);

System.out.println(p2);

}

/\*\*Output

\* Name:zhangsan, Age:10

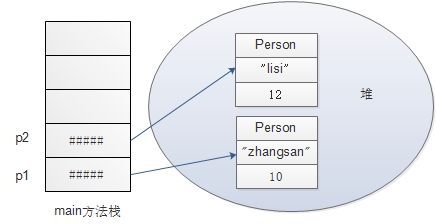
\* Name:lisi, Age:12

\*///~

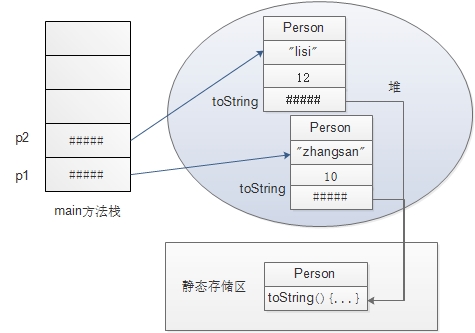
}

[复制代码](javascript:void(0);)

上面的代码我们很熟悉，根据Person构造出的每一个对象都是独立存在的，保存有自己独立的成员变量，相互不会影响，他们在内存中的示意如下:



从上图中可以看出，p1和p2两个变量引用的对象分别存储在内存中堆区域的不同地址中，所以他们之间相互不会干扰。但其实，在这当中，我们省略了一些重要信息，相信大家也都会想到，对象的成员属性都在这了，由每个对象自己保存，那么他们的方法呢？实际上，不论一个类创建了几个对象，他们的方法都是一样的：



从上面的图中我们可以看到，两个Person对象的方法实际上只是指向了同一个方法定义。这个方法定义是位于内存中的一块不变区域（由jvm划分），我们暂称它为静态存储区。这一块存储区不仅存放了方法的定义，实际上从更大的角度而言，它存放的是各种类的定义，当我们通过new来生成对象时，会根据这里定义的类的定义去创建对象。多个对象仅会对应同一个方法，这里有一个让我们充分信服的理由，那就是不管多少的对象，他们的方法总是相同的，尽管最后的输出会有所不同，但是方法总是会按照我们预想的结果去操作，即不同的对象去调用同一个方法，结果会不尽相同。

我们知道，static关键字可以修饰成员变量和方法，来让它们变成类的所属，而不是对象的所属，比如我们将Person的age属性用static进行修饰，结果会是什么样呢？请看下面的例子：

[复制代码](javascript:void(0);)

public class Person {

String name;

static int age;

/\* 其余代码不变... \*/

/\*\*Output

\* Name:zhangsan, Age:12

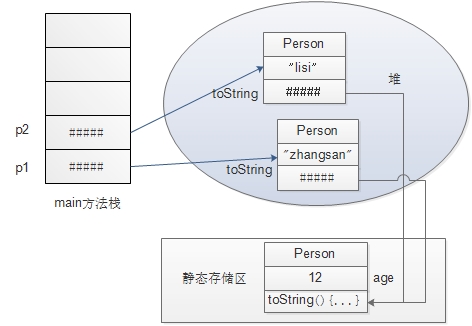
\* Name:lisi, Age:12

\*///~

}

[复制代码](javascript:void(0);)

我们发现，结果发生了一点变化，在给p2的age属性赋值时，干扰了p1的age属性，这是为什么呢？我们还是来看他们在内存中的示意：



我们发现，给age属性加了**static**关键字之后，Person对象就不再拥有age属性了，age属性会统一交给Person类去管理，即多个Person对象只会对应一个age属性，一个对象如果对age属性做了改变，其他的对象都会受到影响。我们看到此时的age和toString()方法一样，都是交由类去管理。

虽然我们看到static可以让对象共享属性，但是实际中我们很少这么用，也不推荐这么使用。因为这样会让该属性变得难以控制，因为它在任何地方都有可能被改变。如果我们想共享属性，一般我们会采用其他的办法：

[复制代码](javascript:void(0);)

public class Person {

private static int count = 0;

int id;

String name;

int age;

public Person() {

id = ++count;

}

public String toString() {

return "Id:" + id + ", Name:" + name + ", Age:" + age;

}

public static void main(String[] args) {

Person p1 = new Person();

p1.name = "zhangsan";

p1.age = 10;

Person p2 = new Person();

p2.name = "lisi";

p2.age = 12;

System.out.println(p1);

System.out.println(p2);

}

/\*\*Output

\* Id:1, Name:zhangsan, Age:10

\* Id:2, Name:lisi, Age:12

\*///~

}

[复制代码](javascript:void(0);)

上面的代码起到了给Person的对象创建一个唯一id以及记录总数的作用，其中count由static修饰，是Person类的成员属性，每次创建一个Person对象，就会使该属性自加1然后赋给对象的id属性，这样，count属性记录了创建Person对象的总数，由于count使用了private修饰，所以从类外面无法随意改变。

**2.修饰成员方法**

static的另一个作用，就是修饰成员方法。相比于修饰成员属性，修饰成员方法对于数据的存储上面并没有多大的变化，因为我们从上面可以看出，方法本来就是存放在类的定义当中的。static修饰成员方法最大的作用，就是可以使用"**类名.方法名**"的方式操作方法，避免了先要new出对象的繁琐和资源消耗，我们可能会经常在帮助类中看到它的使用：

[复制代码](javascript:void(0);)

public class PrintHelper {

public static void print(Object o){

System.out.println(o);

}

public static void main(String[] args) {

PrintHelper.print("Hello world");

}

}

[复制代码](javascript:void(0);)

上面便是一个例子（现在还不太实用），但是我们可以看到它的作用，使得static修饰的方法成为类的方法，使用时通过“**类名.方法名**”的方式就可以方便的使用了，相当于定义了一个全局的函数（只要导入该类所在的包即可）。不过它也有使用的局限，一个static修饰的类中，不能使用非static修饰的成员变量和方法，这很好理解，因为static修饰的方法是属于类的，如果去直接使用对象的成员变量，它会不知所措（不知该使用哪一个对象的属性）。

**3.静态块**

在说明static关键字的第三个用法时，我们有必要重新梳理一下一个对象的初始化过程。以下面的代码为例：

[复制代码](javascript:void(0);)

package com.dotgua.study;

class Book{

public Book(String msg) {

System.out.println(msg);

}

}

public class Person {

Book book1 = new Book("book1成员变量初始化");

static Book book2 = new Book("static成员book2成员变量初始化");

public Person(String msg) {

System.out.println(msg);

}

Book book3 = new Book("book3成员变量初始化");

static Book book4 = new Book("static成员book4成员变量初始化");

public static void main(String[] args) {

Person p1 = new Person("p1初始化");

}

/\*\*Output

\* static成员book2成员变量初始化

\* static成员book4成员变量初始化

\* book1成员变量初始化

\* book3成员变量初始化

\* p1初始化

\*///~

}

[复制代码](javascript:void(0);)

 上面的例子中，Person类中组合了四个Book成员变量，两个是普通成员，两个是static修饰的类成员。我们可以看到，当我们new一个Person对象时，static修饰的成员变量首先被初始化，随后是普通成员，最后调用Person类的构造方法完成初始化。也就是说，在创建对象时，static修饰的成员会首先被初始化，而且我们还可以看到，如果有多个static修饰的成员，那么会按照他们的先后位置进行初始化。

实际上，static修饰的成员的初始化可以更早的进行，请看下面的例子：

[复制代码](javascript:void(0);)

class Book{

public Book(String msg) {

System.out.println(msg);

}

}

public class Person {

Book book1 = new Book("book1成员变量初始化");

static Book book2 = new Book("static成员book2成员变量初始化");

public Person(String msg) {

System.out.println(msg);

}

Book book3 = new Book("book3成员变量初始化");

static Book book4 = new Book("static成员book4成员变量初始化");

public static void funStatic() {

System.out.println("static修饰的funStatic方法");

}

public static void main(String[] args) {

Person.funStatic();

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Person p1 = new Person("p1初始化");

}

/\*\*Output

\* static成员book2成员变量初始化

\* static成员book4成员变量初始化

\* static修饰的funStatic方法

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* book1成员变量初始化

\* book3成员变量初始化

\* p1初始化

\*///~

}

[复制代码](javascript:void(0);)

在上面的例子中我们可以发现两个有意思的地方，第一个是当我们没有创建对象，而是通过类去调用类方法时，尽管该方法没有使用到任何的类成员，类成员还是在方法调用之前就初始化了，这说明，当我们第一次去使用一个类时，就会触发该类的成员初始化。第二个是当我们使用了类方法，完成类的成员的初始化后，再new该类的对象时，static修饰的类成员没有再次初始化，这说明，static修饰的类成员，在程序运行过程中，只需要初始化一次即可，不会进行多次的初始化。

回顾了对象的初始化以后，我们再来看static的第三个作用就非常简单了，那就是当我们初始化static修饰的成员时，可以将他们统一放在一个以static开始，用花括号包裹起来的块状语句中：

[复制代码](javascript:void(0);)

class Book{

public Book(String msg) {

System.out.println(msg);

}

}

public class Person {

Book book1 = new Book("book1成员变量初始化");

static Book book2;

static {

book2 = new Book("static成员book2成员变量初始化");

book4 = new Book("static成员book4成员变量初始化");

}

public Person(String msg) {

System.out.println(msg);

}

Book book3 = new Book("book3成员变量初始化");

static Book book4;

public static void funStatic() {

System.out.println("static修饰的funStatic方法");

}

public static void main(String[] args) {

Person.funStatic();

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Person p1 = new Person("p1初始化");

}

/\*\*Output

\* static成员book2成员变量初始化

\* static成员book4成员变量初始化

\* static修饰的funStatic方法

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* book1成员变量初始化

\* book3成员变量初始化

\* p1初始化

\*///~

}

[复制代码](javascript:void(0);)

我们将上一个例子稍微做了一下修改，可以看到，结果没有二致。

**4.静态导包**

 相比于上面的三种用途，第四种用途可能了解的人就比较少了，但是实际上它很简单，而且在调用类方法时会更方便。以上面的“PrintHelper”的例子为例，做一下稍微的变化，即可使用静态导包带给我们的方便：

[复制代码](javascript:void(0);)

/\* PrintHelper.java文件 \*/

package com.dotgua.study;

public class PrintHelper {

public static void print(Object o){

System.out.println(o);

}

}

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

/\* App.java文件 \*/  
  
import static com.dotgua.study.PrintHelper.\*;

public class App

{

public static void main( String[] args )

{

print("Hello World!");

}

/\*\*Output

\* Hello World!

\*///~

}

[复制代码](javascript:void(0);)

上面的代码来自于两个java文件，其中的PrintHelper很简单，包含了一个用于打印的static方法。而在App.java文件中，我们首先将PrintHelper类导入，这里在导入时，我们使用了static关键字，而且在引入类的最后还加上了**“.\*”**，它的作用就是将PrintHelper类中的所有类方法直接导入。不同于非static导入，采用static导入包后，在不与当前类的方法名冲突的情况下，无需使用“**类名.方法名**”的方法去调用类方法了，直接可以采用"**方法名**"去调用类方法，就好像是该类自己的方法一样使用即可。

**总结**

static是java中非常重要的一个关键字，而且它的用法也很丰富，主要有四种用法：

1. 用来修饰成员变量，将其变为类的成员，从而实现所有对象对于该成员的共享；
2. 用来修饰成员方法，将其变为类方法，可以直接使用**“类名.方法名”**的方式调用，常用于工具类；
3. 静态块用法，将多个类成员放在一起初始化，使得程序更加规整，其中理解对象的初始化过程非常关键；
4. 静态导包用法，将类的方法直接导入到当前类中，从而直接使用**“方法名”**即可调用类方法，更加方便。