# Java 基础提升篇：Static 关键字

## Static 变量

**static 关键字**

- 在类中用 static 声明的成员变量为静态成员变量，它为该类的公用变量，在第一次使用时初始化，对于该类的所有对象来说，static 成员变量只有一份。
- 可以通过引用或者类名访问静态成员

原来一个类里面的成员变量，每 new 一个对象，这个对象就有一份自己的成员变量，因为这些成员变量都不是静态成员变量。对于 static 成员变量来说，这个成员变量只有一份，而且这一份是这个类所有的对象共享。

◆ 在类中，用 static 声明的成员变量为**静态变量**，或者叫:类属性、类变量。

（注意 :静态变量是从属于类 ,在对象里面是没有这个属性的 ;成员变量是从属于对象的，有了对象才有那个属性）

- 它为该类的公用变量，属于类，被该类的所有实例共享，在类被载入时被显示初始化。
- 对于该类所有对象来说，static 成员变量只有一份.被该类的所有对象共享！！
- 可以使用“对象.类属性”来调用。不过，一般都是用"类名.类属性”。
- static 变量置于方法区中。
- 在静态的方法里面不可以调用非静态的方法或变量 ;但是在非静态的方法里可以调用静态的方法或变量。

## Static 方法

◆ 用 static 声明的方法为静态方法。

- 不需要对象，就可以调用（类名.方法名）
- 在调用该方法时，不会将对象的引用传递给它，所以在 **static 方法中不可访问非 static 的成员。**

**【实例 1】**

student 类：

```
public class Student {
    //静态的数据
     String name;
     int id;     //学号
     int age;
     String gender;
     int weight;

     static int ss;

     public static void printSS(){
         System.out.println(ss);
     }
    //动态的行为
     public void study(){
         System.out.println(name+"在學習");
     }

     public void sayHello(String sname){
         System.out.println(name+"向"+sname+"說：你好！");
     }

}
```



测试：

```
public class Test {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Student.ss=321;
        Student.printSS();
    }

}
```



# 静态初始化块

静态初始化块是在类被加载的时候就执行的一块程序，并且一直存在直到程序关闭。也就是说当程序被执行，即 classloader 将该 java 程序编译后的 class 文件加载后，就能执行到静态初始化块这段程序；当程序关闭，我的个人理解也就是 java.exe 进程被结束的时候，静态初始化块结束（例如在静态初始化块里对一个类的静态变量进行赋值，该变量一直存在到程序关闭）。

下面我们来举例说明：

```
public class Test {
        //静态变量
```

```
        public static String testStatic = "testStatic";
        //静态初始化块
        static {
            System.out.println(testStatic);
            System.out.println("Proc begin");
            testStatic = "testProc";
            System.out.println("Proc end");
        }
        //主方法
        public static void main(String[] args) {
            System.out.println(testProc);
            System.out.println("main begin");
            System.out.println("main end");
        }
    }
```

执行 main 方法输出结果：

```
testStatic
Proc begin
Proc end
testProc
main begin
main end
```

也就是说当 JVM 将要执行 main 方法的时候，先要将 Test.class 加载到 JVM 中，此时就执行了静态初始化块中的程序；然后再执行执行 main 方法中的程序。这个例子没有将这个类实例化，所以没有用到构造函数。倘若需要实例化该类的时候，则构造方法的执行顺序也是在静态初始化块之后的。

最后我们可以得出这么一个结论：Java 类的执行**优先顺序。**

**该类的静态变量->该类的静态初始化块->该类的构造方法**

若存在父类的情况下则是：

**父类的静态变量->父类的静态初始化块->子类的静态变量->子类的静态初始化块**

# 内存分析 Static

静态成员变量与非静态成员变量的区别：

【示例 1】

```
package cn.galc.test;
```

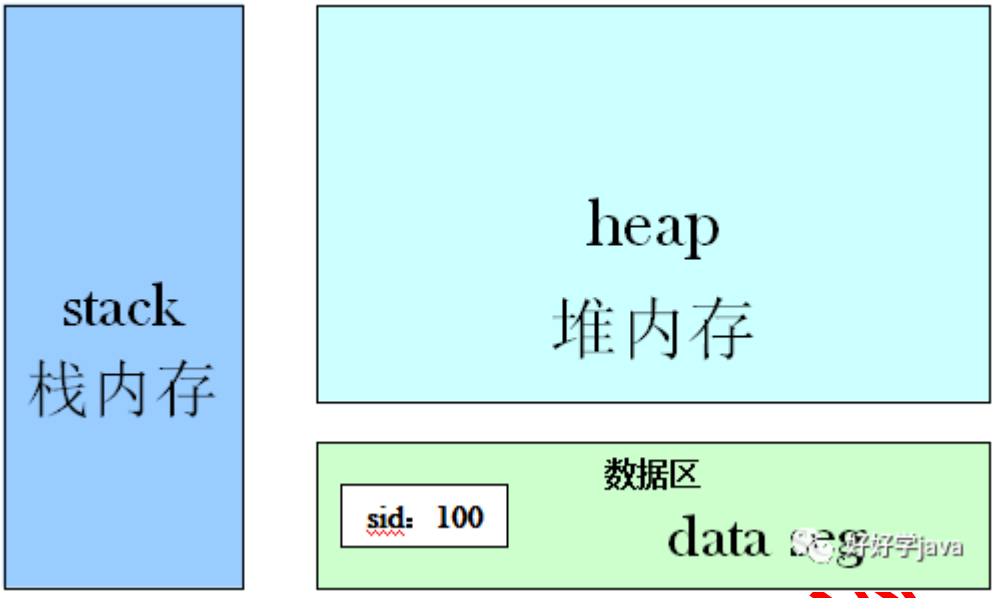
```
public class Cat {
    /**
     * 静态成员变量

     */
    private static int sid = 0;
    private String name;
    int id;
    Cat(String name) {
        this.name = name;
        id = sid++;
    }
    public void info() {
        System.out.println("My Name is " + name + ",NO." + id);
    }
    public static void main(String[] args) {
        Cat.sid = 100;
        Cat mimi = new Cat("mimi");
        Cat pipi = new Cat("pipi");
        mimi.info();
        pipi.info();
    }
}
```

通过画内存分析图了解整个程序的执行过程：

执行程序的第一句话：Cat.sid = 100;时，这里的 sid 是一个静态成员变量，静态变量存放在数据区(data seg)，所以首先在数据区里面分配一小块空间 sid，第一句话执行完后，sid 里面装着一个值就是 100。

此时的内存布局示意图如下所示



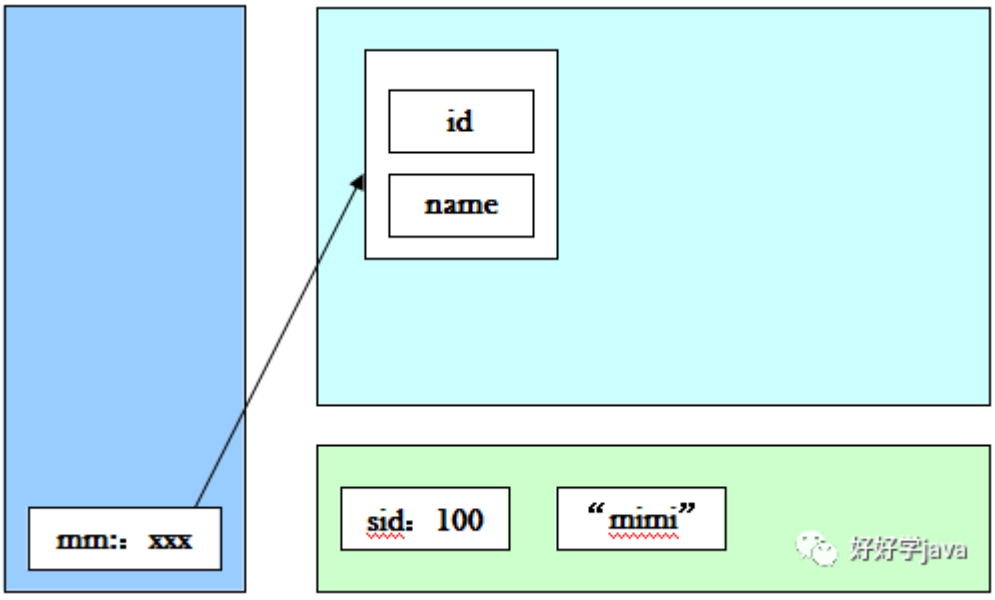

接下来程序执行到：

```
Cat  mimi = new Cat(“mimi”);
```

这里，调用 Cat 类的构造方法 Cat(String name)，构造方法的定义如下：

```
Cat ( String name){
   this.name = name;
   id=sid++;
}
```

调用时首先在栈内存里面分配一小块内存 mm，里面装着可以找到在堆内存里面的 Cat 类的实例对象的地址，mm 就是堆内存里面 Cat 类对象的引用对象。这个构造方法声明有字符串类型的形参变量,所以这里把“mimi”作为实参传递到构造方法里面，由于**字符串常量是分配在数据区存储**的，所以数据区里面多了一小块内存用来存储字符串“mimi”。此时的内存分布如下图所示：



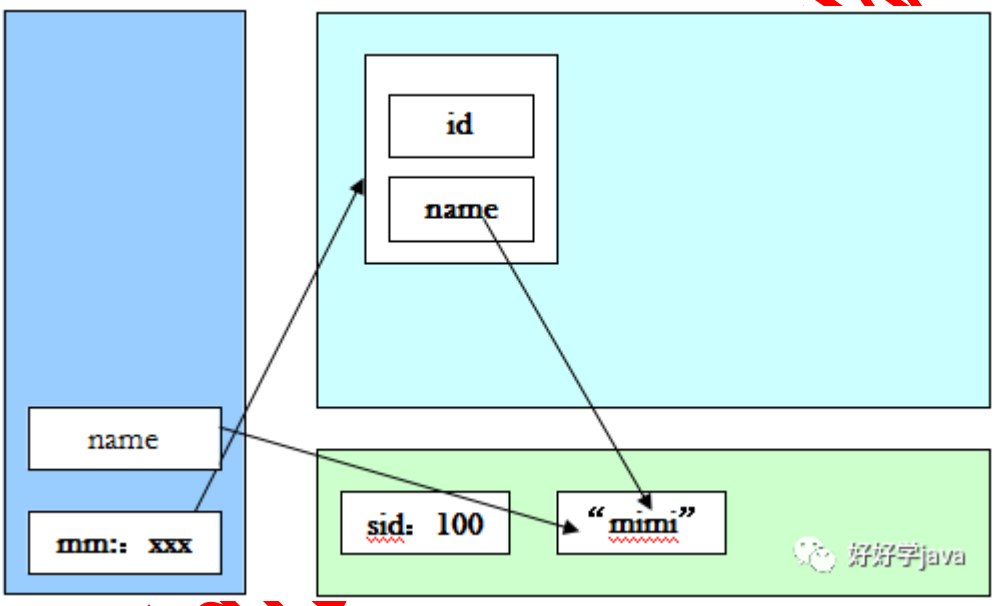

当调用构造方法时，首先在栈内存里面给形参 name 分配一小块空间，名字叫 name,接

下来把”mimi”这个字符串作为实参传递给 name，字符串也是一种引用类型，除了那四类 8 种基础数据类型之外，其他所有的都是引用类型，所以可以认为字符串也是一个对象。所以这里相当于把”mimi”这个对象的引用传递给了 name，所以现在 name 指向的是”mimi”。

接下来执行构造方法体里面的代码:

```
this.name=name;
```

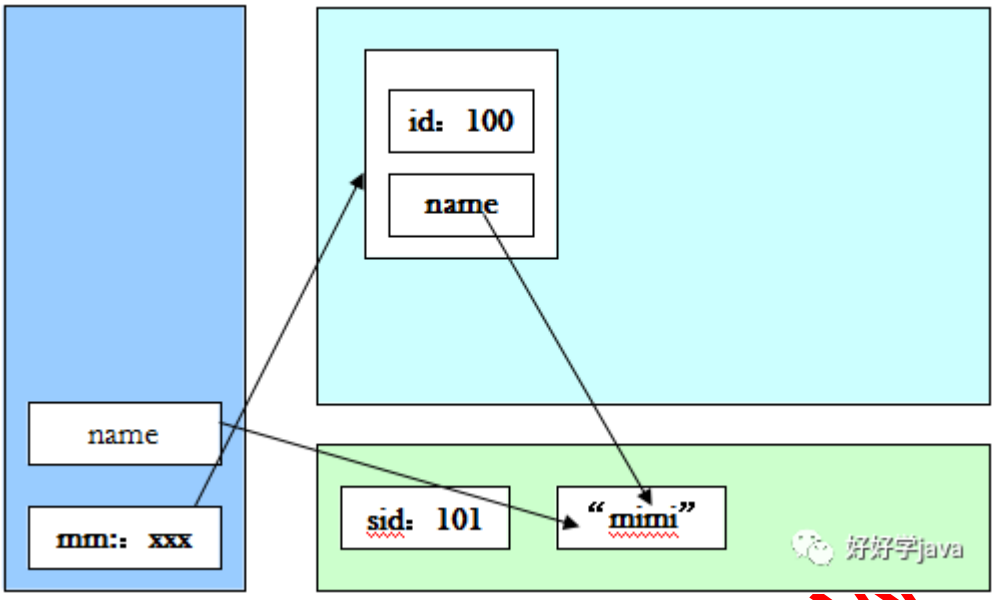
这里的 this 指的是当前的对象，指的是堆内存里面的那只猫。这里把栈里面的 name 里面装着的值传递给堆内存里面的 cat 对象的 name 属性，所以此时这个 name 里面装着的值也是可以找到位于数据区里面的字符串对象“mimi”的，此时这个 name 也是字符串对象“mimi”的一个引用对象，通过它的属性值就可以找到位于数据区里面的字符串对象“mimi”。此时的内存分布如下图所示：



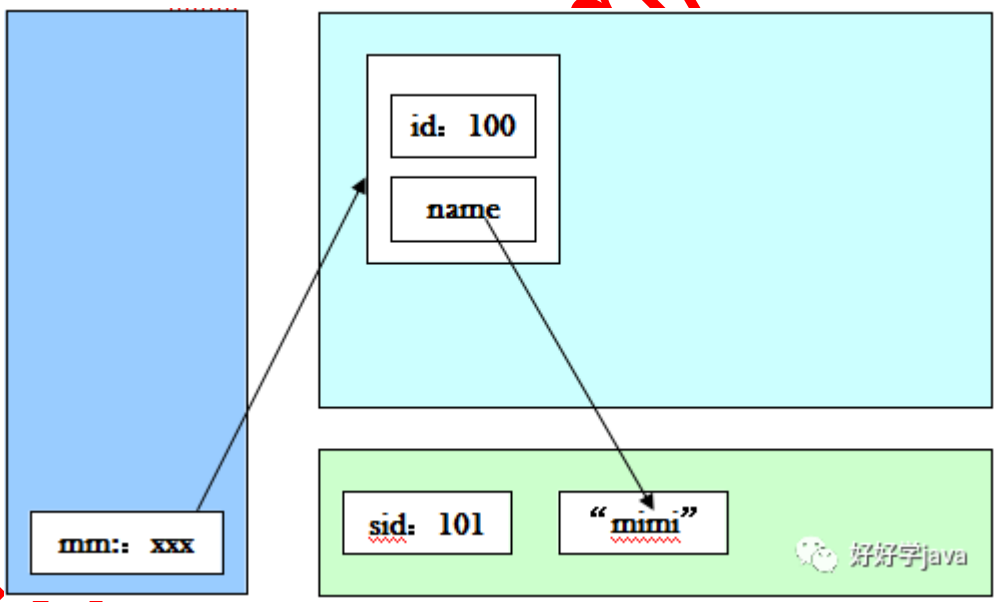

接下来执行方法体内的另一句代码：

```
id=sid++;
```

这里是把 sid 的值传递给 id，所以 id 的值是 100，sid 传递完以后，自己再加 1，此时 sid 变成了 101。此时的内存布局如下图所示。



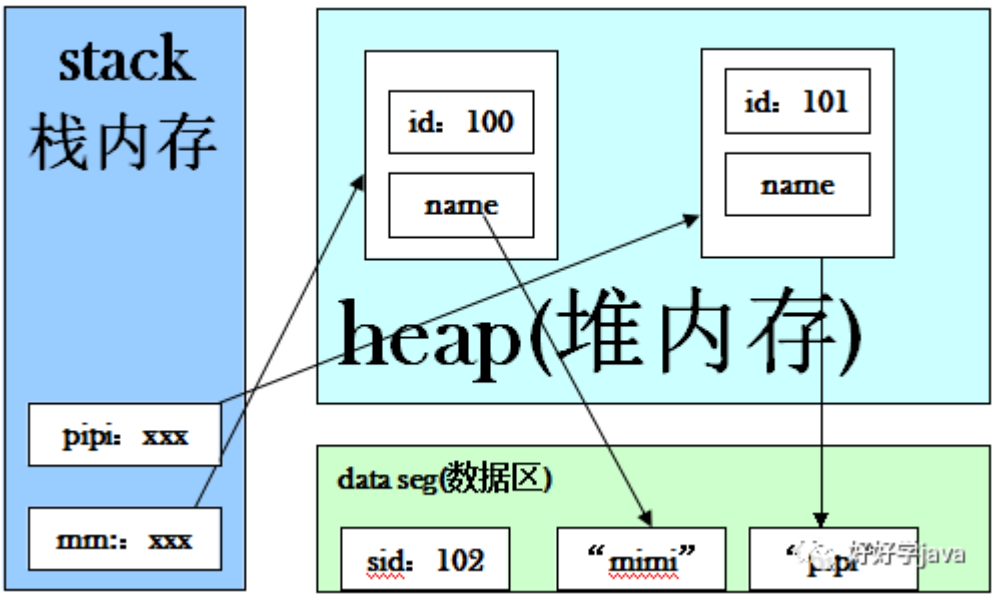

到此，构造方法调用完毕，给这个构造方法分配的局部变量所占的内存空间全部都要消失，所以位于栈空间里面的 name 这块内存消失了。栈内存里面指向数据区里面的字符串对象“mimi”的引用也消失了，此时只剩下堆内存里面的指向字符串对象“mimi”的引用没有消失。此时的内存布局如下图所示：




接下来执行：

```
Cat  pipi = new Cat(“pipi”);
```

这里是第二次调用构造方法 Cat()，整个调用过程与第一次一样，调用结束后，此时的内存布局如下图所示：




最后两句代码是调用 info()方法打印出来，打印结果如下：

```
My name is mimi,NO.100
My name is pipi,NO.101
```

通过这个程序，看出来了这个静态成员变量 sid 的作用，它可以计数。每当有一只猫 new 出来的时候，就给它记一个数。让它自己往上加 1。

程序执行完后，内存中的整个布局就如上图所示了。一直持续到 main 方法调用完成的前一刻。

这里调用构造方法 Cat(String name) 创建出两只猫，首先在栈内存里面分配两小块空间 mimi 和 pipi，里面分别装着可以找到这两只猫的地址，mimi 和 pipi 对应着堆内存里面的两只猫的引用。这里的构造方法声明有字符串类型的变量，字符串常量是分配在数据区里面的，所以这里会把传过来的字符串 mimi 和 pipi 都存储到数据区里面。所以数据区里面分配有存储字符串 mimi 和 pipi 的两小块内存，里面装着字符串“mimi”和“pipi”，字符串也是引用类型，除了那四类 8 种的基础数据类型之外，其他所有的数据类型都是引用类型。所以可以认为字符串也是一个对象。

这里是 new 了两只猫出来，这两只猫都有自己的 id 和 name 属性，所以这里的 id 和 name 都是非静态成员变量，即没有 static 修饰。所以每 new 出一只新猫，这只新猫都有属于它自己的 id 和 name，即非静态成员变量 id 和 name 是每一个对象都有单独的一份。但对于静态成员变量来说，只有一份，不管 new 了多少个对象，哪怕不 new 对象，静态成员变量在数据区也会保留一份。如这里的 sid 一样，sid 存放在数据区，无论 new 出来了多少只猫在堆内存里面，sid 都只有一份，只在数据区保留一份。

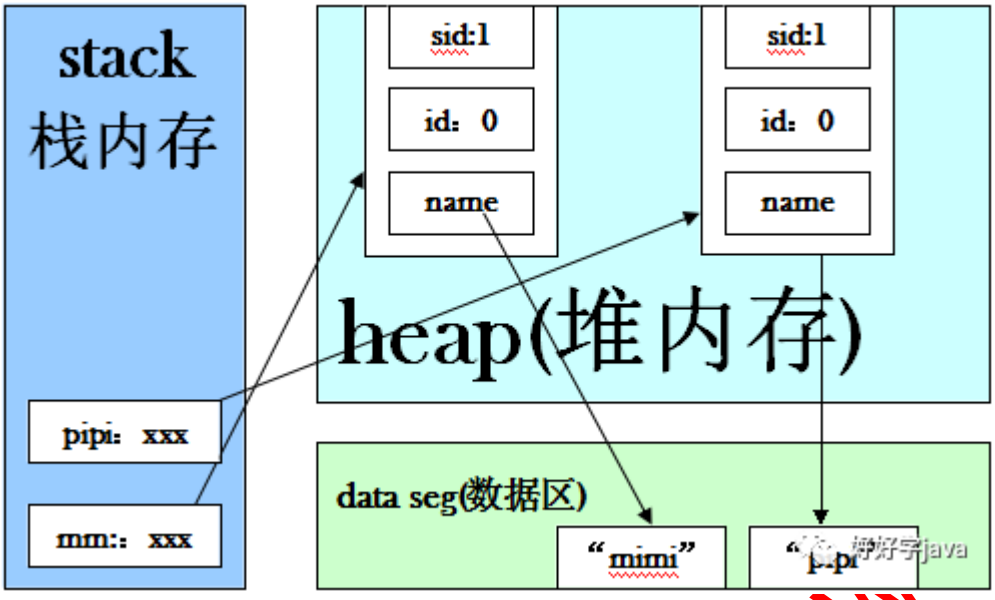
静态成员变量是属于整个类的，它不属于专门的某个对象。那么如何访问这个静态成员变量的值呢？首先第一点，任何一个对象都可以访问这个静态的值，访问的时候访问的都是同一块内存。第二点，即便是没有对象也可以访问这个静态的值，通过“类名.静态成员变量名”来访问这个静态的值，所以以后看到某一个类名加上“.”再加上后面有一个东西，那么后面这个东西一定是静态的，如“System.out”，这里就是通过类名(System 类)再加上“.”

来访问这个 out 的，所以这个 out 一定是静态的。

**【示例 2】**

```
package cn.galc.test;
public class Cat {
    /**
     * 这里面的 sid 不再是静态成员变量了，因为没有 static 修饰符，
     * 此时它就是类里面一个普通的非静态成员变量，和 id，name 一样，
     * 成为每一个 new 出来的对象都具有的属性。
     */
    private  int sid = 0;
    private String name;
    int id;
    Cat(String name) {
        this.name = name;
        id = sid++;
    }
    public void info() {
        System.out.println("My Name is " + name + ",NO." + id);
    }
    public static void main(String[] args) {
        //Cat.sid = 100;这里不能再使用“类.静态成员变量”的格式来访问 sid 了，因为 sid 现在变成了非静态的成员变量了。所以必须要把这句话注释掉，否则无法编译通过。
        Cat mimi = new Cat("mimi");
        Cat pipi = new Cat("pipi");
        mimi.info();
        pipi.info();
    }
}
```

这段代码与上一段代码唯一的区别是把声明 sid 变量的 static 修饰符给去掉了，此时的 sid 就不再是静态成员变量，而是非静态成员变量了，此时每一个 new 出来的 cat 对象都会有自己单独的 sid 属性。所以这段代码执行完成后，内存中的布局如下图所示：




由于 sid 变成了非静态成员变量，所以不再有计数的功能了。sid 与 id 和 name 属性一样，成为每一个 new 出来的对象都具有的属性，所以每一个 new 出来的 cat 都加上了一个 sid 属性。由于不能再使用”类名.静态成员对象名”的格式访问 sid，所以代码的第一句”Cat.sid =100;”不能这样使用，否则编译会出错，必须把这句话注释掉才能编译成功。既然无法访问得到 sid 的值，所以 sid 的值就一直都是初始化时赋给的值 0。直到调用构造方法时，执行到方法体内的代码 id=sid++;时，sid 首先把自身的值 0 赋值给 id，所以 id 的值是 0，然后 sid 自己加 1，所以 sid 变成了 1。

所以**静态变量和非静态变量的区别就在于静态变量可以用来计数，而非静态变量则不行。**

理解了内存，就理解了一切，就理解了各种各样的语言。所有的语言无非都是这样：**局部变量分配内存永远在栈里面，new 出来的东西分配内存永远是在堆里，静态的东西分配内存永远是在数据区。**剩下的代码肯定是在代码区。所有的语言都是这样。

在一个静态方法里，如果想访问一个非静态的成员变量，是不能直接访问的，必须在静态方法里 new 一个对象出来才能访问。如果是加了 static 的成员变量，那么这个成员变量就是一个静态的成员变量，就可以在 main 方法里面直接访问了。

main 方法是一个静态的方法，main 方法要执行的时候不需要 new 一个对象出来。

动态方法是针对于某一个对象调用的，静态方法不会针对某一个对象来调用，没有对象照样可以用。所以可以使用”classname.method()”.的形式来调用静态方法。所以想在 main 方法里面访问非静态成员变量是不可以的，想在 main 方法里面访问非静态方法也是不可以的，因为非静态方法只能针对于某个对象来调用，没有对象，就找不到方法的执行者了。

成员变量只有在 new 出一个对象来的时候才在堆内存里面分配存储空间。局部变量在栈内存里面分配存储空间。

静态方法不再是针对某一个对象来调用，所以不能访问非静态的成员。

非静态成员专属于某一个对象，想访问非静态成员必须 new 一个对象出来才能访问。

静态的变量可以通过对象名去访问，也可以通过类名去访问，两者访问的都是同一块内存。

# 总结

类只能使用类的方法和属性，对象既可以使用类的方法，又可以使用对象的方法。

static 的方法只能调用 static 的变量和方法，非 static 的方法既可以调用 static 的，又可以调用非 static 的。