**static**

static表示“全局”或者“静态”的意思，用来修饰成员变量和成员方法，也可以形成静态static代码块，但是Java语言中没有全局变量的概念。

被static修饰的成员变量和成员方法独立于该类的任何对象。它不依赖类特定的实例，被类的所有实例共享。只要这个类被加载，Java虚拟机就能根据类名在运行时数据区的方法区内定找到他们。因此，static对象可以在它的任何对象创建之前访问，无需引用任何对象。

用public修饰的static成员变量和成员方法本质是全局变量和全局方法，当声明它类的对象时，不生成static变量的副本，而是类的所有实例共享同一个static变量。

 static变量前可以有private修饰，表示这个变量可以在类的静态代码块中，或者类的其他静态成员方法中使用（当然也可以在非静态成员方法中使用），但是不能在其他类中通过类名来直接引用，这一点很重要。实际上你需要搞明白，private是访问权限限定，static表示不要实例化就可以使用。static前面加上其它访问权限关键字的效果也以此类推。

 static修饰的成员变量和成员方法习惯上称为静态变量和静态方法，可以直接通过类名来访问，访问语法为：  
类名.静态方法名(参数列表...)   
类名.静态变量名

用static修饰的代码块表示静态代码块，当Java虚拟机（JVM）加载类时，就会执行该代码块。

1、static变量按照是否静态的对类成员变量进行分类可分两种：一种是被static修饰的变量，叫静态变量或类变量；另一种是没有被static修饰的变量，叫实例变量。两者的区别是：对于静态变量在内存中只有一个拷贝（节省内存），JVM只为静态分配一次内存，在加载类的过程中完成静态变量的内存分配，可用类名直接访问（方便），当然也可以通过对象来访问（但是这是不推荐的）。对于实例变量，每次创建一个实例，就会为实例变量分配一次内存，实例变量可以在内存中有多个拷贝，互不影响（灵活）。

2、静态方法  
 静态方法可以直接通过类名调用，任何的实例也都可以调用，因此静态方法中不能用this和super关键字，不能直接访问所属类的实例变量和实例方法只能访问所属类的静态成员变量和成员方法。

3、static代码块

 static代码块也叫静态代码块，是在类中独立于类成员的static语句块，可以有多个，位置可以随便放，它不在任何的方法体内，JVM加载类时会执行这些静态的代码块，如果static代码块有多个，JVM将按照它们在类中出现的先后顺序依次执行它们，每个代码块只会被执行一次。例如：

public class Test5 {   
        private static int a;   
        private int b;   
  
        static {   
                Test5.a = 3;   
                System.out.println(a);   
                Test5 t = new Test5();   
                t.f();   
                t.b = 1000;   
                System.out.println(t.b);   
        }   
  
        static {   
                Test5.a = 4;   
                System.out.println(a);   
        }   
  
        public static void main(String[] args) {   
                // TODO 自动生成方法存根   
        }   
  
        static {                Test5.a = 5;   
                System.out.println(a);   
        }   
  
        public void f() {   
                System.out.println("hhahhahah");   
        }   
}

运行结果：  
3  
hhahhahah  
1000  
4  
5

利用静态代码块可以对一些static变量进行赋值。

4、static和final一块用表示什么  
static final用来修饰成员变量和成员方法，可简单理解为“全局常量”！对于变量，表示一旦给值就不可修改，并且通过类名可以访问。对于方法，表示不可覆盖，并且可以通过类名直接访问。

特别要注意一个问题：

  对于被static和final修饰过的实例常量，实例本身不能再改变了，但对于一些容器类型（比如，ArrayList、HashMap）的实例变量，不可以改变容器变量本身，但可以修改容器中存放的对象，这一点在编程中用到很多。

public class TestStaticFinal {   
        private static final String strStaticFinalVar = "aaa";   
        private static String strStaticVar = null;   
        private final String strFinalVar = null;   
        private static final int intStaticFinalVar = 0;   
        private static final Integer integerStaticFinalVar = new Integer(8);   
        private static final ArrayList<String> alStaticFinalVar = new ArrayList<String>();   
  
        private void test() {   
                System.out.println("-------------值处理前----------\r\n");   
                System.out.println("strStaticFinalVar=" + strStaticFinalVar + "\r\n");   
                System.out.println("strStaticVar=" + strStaticVar + "\r\n");   
                System.out.println("strFinalVar=" + strFinalVar + "\r\n");   
                System.out.println("intStaticFinalVar=" + intStaticFinalVar + "\r\n");   
                System.out.println("integerStaticFinalVar=" + integerStaticFinalVar + "\r\n");   
                System.out.println("alStaticFinalVar=" + alStaticFinalVar + "\r\n");   
  
  
                //strStaticFinalVar="哈哈哈哈";        //错误，final表示终态,不可以改变变量本身.   
                strStaticVar = "哈哈哈哈";                //正确，static表示类变量,值可以改变.   
                //strFinalVar="呵呵呵呵";                    //错误, final表示终态，在定义的时候就要初值（哪怕给个null），一旦给定后就不可再更改。   
                //intStaticFinalVar=2;                        //错误, final表示终态，在定义的时候就要初值（哪怕给个null），一旦给定后就不可再更改。   
                //integerStaticFinalVar=new Integer(8);            //错误, final表示终态，在定义的时候就要初值（哪怕给个null），一旦给定后就不可再更改。   
                alStaticFinalVar.add("aaa");        //正确，容器变量本身没有变化，但存放内容发生了变化。这个规则是非常常用的，有很多用途。   
                alStaticFinalVar.add("bbb");        //正确，容器变量本身没有变化，但存放内容发生了变化。这个规则是非常常用的，有很多用途。   
  
                System.out.println("-------------值处理后----------\r\n");   
                System.out.println("strStaticFinalVar=" + strStaticFinalVar + "\r\n");   
                System.out.println("strStaticVar=" + strStaticVar + "\r\n");   
                System.out.println("strFinalVar=" + strFinalVar + "\r\n");   
                System.out.println("intStaticFinalVar=" + intStaticFinalVar + "\r\n");   
                System.out.println("integerStaticFinalVar=" + integerStaticFinalVar + "\r\n");   
                System.out.println("alStaticFinalVar=" + alStaticFinalVar + "\r\n");   
        }   
  
        public static void main(String args[]) {   
                new TestStaticFinal().test();   
        }   
}

运行结果如下：

-------------值处理前----------

strStaticFinalVar=aaa

strStaticVar=null

strFinalVar=null

intStaticFinalVar=0

integerStaticFinalVar=8

alStaticFinalVar=[]

-------------值处理后----------

strStaticFinalVar=aaa

strStaticVar=哈哈哈哈

strFinalVar=null

intStaticFinalVar=0

integerStaticFinalVar=8

alStaticFinalVar=[aaa, bbb]

Process finished with exit code 0

        看了上面这个例子，就清楚很多了，但必须明白：通过static final修饰的容器类型变量中所“装”的对象是可改变的。这是和一般基本类型和类类型变量差别很大的地方。

[**Java基础(六) static五大应用场景**](https://www.cnblogs.com/LiaHon/p/11075178.html)

static是java中很常用的一个关键字，使用场景也很多，本文主要介绍了它的五种用法，static变量，static方法，static代码块，static内部类，static包内导入，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

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。上一个章节我们讲了[final关键字的原理及用法](https://www.cnblogs.com/LiaHon/p/11071861.html)，本章节我们来了解一下static关键字原理及其用法。

**一. static特点**

static是一个修饰符，通常用于修饰变量和方法，如开发过程中用到的字典类数据都会用到static来修饰，工具类方法，如Dateutils，Stringutils这类工具方法也会用到static来修饰，那么除了这两种最常用的场景外，是否还有其他场景呢，答案是：有的，总共五种：

1. static变量
2. static方法
3. static代码块
4. static内部类
5. static包内导入

static修饰的变量、方法、代码块、内部类在类加载期间就已经完成初始化，存储在Java Heap（JDK7.0之前存储在方法区）中静态存储区，因此static优于对象而存在。

static修饰的成员（变量、方法）被所有对象所共享，也叫静态变量或静态方法，可直接通过类调用（也建议通过类调用）。

**二. static 变量**

static变量随着类的加载而存在，随着类的消失而消失，当类被加载时，就会为静态变量在Java Heap中分配内存空间，可以通过【类.变量名】和【对象.变量名】的方式调用，建议直接使用【类.变量名】的方式，

public class Person {

private String name;

private static int eyeNum;

public static int legNum = 2;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public static int getEyeNum() {

return eyeNum;

}

public static void setEyeNum(int eyeNum) {

Person.eyeNum = eyeNum;

}

}

public static void main(String[] args) {

Person person = new Person();

person.setEyeNum(25);

Person person1 = new Person();

person1.setEyeNum(28);

System.out.println(person.getEyeNum());//28

System.out.println(person1.getEyeNum());//28

int legNum = person.legNum;

System.out.println(legNum);//2

}

从上面的例子可以看出静态变量是对所有对象共享，一个对象对其值的改动，直接就会造成另一个对象取值的不同。

**什么时候使用static变量？**

作为共享变量使用，通常搭配final关键字一起使用，比如我们常用的字典类数据；

private static final String GENERAL\_MAN = "man";

减少对象的创建，比如在类开头的部分，定义Logger方法，用于异常日志采集

private static Logger LOGGER = LogFactory.getLoggger(MyClass.class);

始终返回同一变量，比如我们的单例模式。

**三. static 方法**

静态方法只能访问静态成员（静态变量、静态方法），而非静态方法既可访问静态方法也可访问非静态方法；因为静态成员优于对象而存在，因此无法调用和对象相关的关键字，如this,super，无法通过关键字访问对象资源。

public class Person {

private String name;

private static int eyeNum;

public static int legNum = 2;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public static int getEyeNum() {

System.out.println(name);//编译出错，name不可用

return eyeNum;

}

public static void setEyeNum(int eyeNum) {

Person.eyeNum = eyeNum;

this.name = "";//编译出错，this不可用

}

}

**什么时候使用static方法？**

static方法一般用于与当前对象无法的工厂方法、工具方法。如Math.sqrt()，Arrays.sort()，StringUtils.isEmpty()等。

**四. static 代码块**

static代码块相对于static变量和static方法来说使用不是那么广泛，但也算是比较常见的用法了，static代码块在加载一个类的时候最先执行，且只执行一次。

public static Map<String, String> timeTypes;

static {

timeTypes = new HashMap<>();

timeTypes.put("year", "年");

timeTypes.put("quarter", "季");

timeTypes.put("month", "月");

timeTypes.put("day", "日");

System.out.println("初始化1");

}

public static void main(String[] args) {

System.out.println("初始化2");

}

执行结果是：

初始化1；

初始化2；

**什么时候使用static代码块？**

一般在进行初始化操作时，比如读取配置文件信息，获取当前服务器参数等

**五. static内部类**

定义一个内部类，加上static，就成为了一个static内部类，static只能修饰内部类，不能修饰顶级类，静态内部类在业务应用系统开发中使用的不多。

public class StaticCouter {

private String str0 = "hi"; //非静态变量

private static String str1 = "hello"; //静态变量

static class StaticInnerClass{//静态内部类

public void getMessage(){

System.out.println(str0);//编译出错

System.out.println(str1);

}

}

class NormalInnerClass{//非静态内部类

public void getMessage(){

System.out.println(str0);

System.out.println(str1);

}

}

}

**静态内部类与非静态内部类有何异同？**

| **静态内部类** | **非静态内部类** |
| --- | --- |
| 不需要有指向外部类的引用 | 必须通过外部类的new关键字引用 |
| 可定义普通变量和方法，也可定义静态变量和方法 | 可定义普通变量和方法，不可定义静态变量和方法 |
| 可以调用外部类的静态成员，不能调用外部类的普通成员 | 可调用外部类的普通成员和静态成员 |

public static void main(String[] args) {

//创建静态内部类实例

StaticInnerClass staticInnerClass = new StaticInnerClass();

//调用静态内部类方法

staticInnerClass.getMessage();

//创建静态内部类实例

StaticCouter.StaticInnerClass staticInnerClass1 = new staticCouter.StaticInnerClass();

//调用静态内部类方法

staticInnerClass1.getMessage();

//创建普通内部类实例

StaticCouter.NormalInnerClass normalInnerClass = new StaticCouter().new NormalInnerClass();

//调用普通内部类方法

normalInnerClass.getMessage();

}

**六. static包内导入**

这个概念不太好理解，举个例子

public static void main(String[] args) {

int[] arra = {1,4,5,7};

Arrays.sort(arra);

Arrays.asList(arra);

Arrays.fill(arra, 6);

}

static包导入目的就是去掉重复的Arrays类名调用

通过在顶部引入

import static java.util.Arrays.\*

即可把Arrays类中所有的静态变量，方法，内部类等都引入当前类中，调用时直接调用sort(arra)，asList(arra)，

java5后引入的，不常用，调用类方法时会比较简单，但可读性不好，慎用。