

静态变量：

类变量也叫静态变量/静态属性，是该类的所有对象共享的变量，任何一个该类的对象去访问它时，取到的都是相同的值，同样任何一个该类的对象去修改它时，修改的也是同一个变量。这个从前面的图也可看出来。

定义静态变量：

访问修饰符 static 数据类型 变量名

访问静态变量：

类名. 变量名

注意事项：

1. 什么时候需要用类变量
当我们需要让某个类的所有对象都共享一个变量时，就可以考虑使用类变量(静态变量)：比如：定义学生类，统计所有学生共交多少钱。Student (name, static fee)
2. 类变量与实例变量（普通属性）区别
类变量是该类的所有对象共享的，而实例变量是每个对象独享的。
3. 加上static称为类变量或静态变量，否则称为实例变量/普通变量/非静态变量
4. 类变量可以通过 类名.类变量名 或者 对象名.类变量名 来访问，但java设计者推荐我们使用 类名.类变量名方式访问。【前提是满足访问修饰符的访问权限和范围】
5. 实例变量不能通过 类名.类变量名 方式访问。
6. 类变量是在类加载时就初始化了，也就是说，即使你没有创建对象，只要类加载了，就可以使用类变量了。【案例演示】
7. 类变量的生命周期是随类的加载开始，随着类消亡而销毁。
【举例，Monster.name】【案例演示】

静态方法：

定义静态方法：

访问修饰符 static 数据返回类型 方法名（） {}

使用静态方法：类名. 方法名

注意事项：

- 1) 类方法和普通方法都是随着类的加载而加载，将结构信息存储在方法区：
类方法中无this的参数
普通方法中隐含着this的参数
- 2) 类方法可以通过类名调用，也可以通过对象名调用。【举例】
- 3) 普通方法和对象有关，需要通过对象名调用，比如对象名.方法名(参数)，不能通过类名调用。【举例】

4) 类方法中不允许使用和对对象有关的关键字, 比如this和super。普通方法(成员方法)可以。
[举例]

5) 类方法(静态方法)中 只能访问 静态变量 或静态方法 。【如何理解】

6) 普通成员方法, 既可以访问 非静态成员, 也可以访问静态成员。

小结: 静态方法, 只能访问静态的成员, 非静态的方法, 可以访问静态成员和非静态成员
(必须遵守访问权限)

main 方法:

解释main方法的形式: **public static void main(String[] args){}**

1. main方法时虚拟机调用

2. java虚拟机需要调用类的main()方法, 所以该方法的访问权限必须是public

3. java虚拟机在执行main()方法时不必创建对象, 所以该方法必须是static

4. 该方法接收String类型的数组参数, 该数组中保存执行java命令时传递给所运行的类的参数, 案例演示, 接收参数.

5. java 执行的程序 参数1 参数2 参数3 [举例说明:]



注意:

1) 在 main() 方法中, 我们可以直接调用 main 方法所在类的静态方法或静态属性。

2) 但是, 不能直接访问该类中的非静态成员, 必须创建该类的一个实例对象后, 才能通过这个对象去访问类中的非静态成员。

代码块:

代码化块又称为**初始化块**, 属于类中的成员[即 是类的一部分], 类似于方法, 将逻辑语句封装在方法体中, 通过{}包围起来。

但和方法不同, 没有方法名, 没有返回, 没有参数, 只有方法体, 而且不用通过对象或类显式调用, 而是加载类时, 或创建对象时隐式调用。

语法:

(修饰符) {

代码

}

注意:

- 1、修饰符可选 static 或者没有
- 2、代码块分为静态代码块与非静态代码块
- 3、逻辑语句可以是任何逻辑语句

代码块注意事项:

1) static代码块也叫静态代码块, 作用就是对类进行初始化, 而且它随着**类的加载**而执行, 并且**只会执行一次**。如果是普通代码块, 每创建一个对象, 就执行。

2) 类什么时候被加载[重要背!]

- ① 创建对象实例时(new)
 - ② 创建子类对象实例, 父类也会被加载
 - ③ 使用类的静态成员时(静态属性, 静态方法)
- 案例演示: A 类 extends B类 的静态块

3) 普通的代码块, 在创建对象实例时, 会被隐式的调用。
被创建一次, 就会调用一次。

如果只是使用类的静态成员时, 普通代码块并不会执行。

小结:1. static代码块是**类加载**时, 执行, 只会执行一次

2. 普通代码块是在创建对象时调用的, 创建一次, 调用一次

3. 类加载的3种情况, 需要记住。

4) 创建一个对象时, 在**一个类**调用顺序是:(**重点, 难点**):

① 调用静态代码块和静态属性初始化(注意: 静态代码块和静态属性初始化调用的优先级一样, 如果有多个静态代码块和多个静态变量初始化, 则按他们定义的顺序调用) [举例说明]

② 调用普通代码块和普通属性的初始化(注意: 普通代码块和普通属性初始化调用的优先级一样, 如果有多个普通代码块和多个普通属性初始化, 则按定义顺序调用)

③ 调用构造方法。

新写一个类演示【**CodeBlockDetail02.java**】

5) 构造器 的最前面其实隐含了 super()和 调用普通代码块, 新写一个类演示【截图+说明】, 静态相关的代码块, 属性初始化, 在类加载时, 就执行完毕, 因此是优先于 构造器和普通代码块执行的 **CodeBlockDetail03.java**

```
class A {  
    public A() { //构造器  
        //这里有隐藏的执行要求  
        //(1) super(); //这个知识点, 在前面讲解继承的时候, 老师说  
        //(2) 调用普通代码块的  
        System.out.println("ok");  
    }  
}
```

6) 我们看一下创建一个子类对象时(继承关系), 他们的静态代码块, 静态属性初始化, 普通代码块, 普通属性初始化, 构造方法的调用顺序如下:

- ① 父类的静态代码块和静态属性(优先级一样, 按定义顺序执行)
- ② 子类的静态代码块和静态属性(优先级一样, 按定义顺序执行)
- ③ 父类的普通代码块和普通属性初始化(优先级一样, 按定义顺序执行)
- ④ 父类的构造方法
- ⑤ 子类的普通代码块和普通属性初始化(优先级一样, 按定义顺序执行)
- ⑥ 子类的构造方法 // 面试题

AAAAA extends BBBB 类 演示 [10Min]55 [CodeBlockDetail04.java](#)

7) 静态代码块只能直接调用静态成员(静态属性和静态方法), 普通代码块可以调用任意成员。学习比较麻烦, 工作轻松

单例设计模式:

单例(单个的实例)

1. 所谓类的单例设计模式, 就是采取一定的方法保证在整个的软件系统中, 对某个类只能存在一个对象实例, 并且该类只提供一个取得其对象实例的方法
2. 单例模式有两种方式: 1) 饿汉式 2) 懒汉式

演示**饿汉式**和**懒汉式**单例模式的实现。

步骤如下:

- 1) 构造器私有化 =》防止直接 new
- 2) 类的内部创建对象
- 3) 向外暴露一个静态的公共方法。getInstance
- 4) 代码实现 [SingleTon01.java](#) [SingleTon02.java](#)

饿汉式与懒汉式:

- 1、饿汉式在类加载时创建了对象实例, 懒汉式是在使用时才创建
- 2、饿汉式不存在线程安全问题
- 3、饿汉式存在资源浪费问题。

Final 关键字:

Final01.java

final 中文意思:最后的, 最终的.

final 可以修饰类、属性、方法和局部变量.

在某些情况下,程序员可能有以下需求, 就会使用到final:

- 1) 当不希望类被继承时,可以用final修饰.【案例演示】
- 2) 当不希望父类的某个方法被子类覆盖/重写(override)时,可以用final关键字修饰.【案例演示: 访问修饰符 final 返回类型 方法名】
- 3) 当不希望类的某个属性的值被修改,可以用final修饰.【案例演示: public final double TAX_RATE=0.08】
- 4) 当不希望某个局部变量被修改, 可以使用final修饰【案例演示: final double TAX_RATE=0.08】

注意:

- 1) final修饰的属性又叫常量,一般用 XX_XX_XX 来命名
- 2) final修饰的属性在定义时,必须赋初值,并且以后不能再修改,赋值可以在如下位置之一【选择一个位置赋初值即可】:
 - ① 定义时: 如 public final double TAX_RATE=0.08;
 - ② 在构造器中
 - ③ 在代码块中.
- 3) 如果final修饰的属性是静态的,则初始化的位置只能是
 - ① 定义时
 - ② 在静态代码块不能在构造器中赋值。
- 4) final类不能继承,但是可以实例化对象.[A2类]
- 5) 如果类不是final类,但是含有final方法,则该方法虽然不能重写,但是可以被继承.[A3类]

- 5) 一般来说,如果一个类已经是final类了,就没有必要再将方法修饰成final方法.
- 6) final不能修饰构造方法(即构造器)
- 7) final 和 static 往往搭配使用,效率更高,不会导致类加载.底层编译器做了优化处理.

```
class Demo{  
    public static final int i=16; //  
    static{  
        System.out.println("韩顺平教育~");  
    }  
}
```

- 8) 包装类(Integer,Double,Float, Boolean等都是final),String也是final类.