理解多线程

为何用多线程? 多个任务需要执行(机器是按顺序执行 一个阻塞全面停止) 多线程即如果电脑开5个软件,其中一个卡了,然后你去弄另一个然后那个弄完了再去弄原来的那个.

单线程会是等那个卡完了再接着下一个.

但是多线程的问题在于代码写了加了几个线程,但是代码处理的时候是抢占式的,就是如果不同线程之间代码存在关联的话,那么很容易造成程序结果奔溃.所以要实现线程同步. 即出去买菜, 需要去菜市场,拿菜,给钱,三步,如果分为三个线程则容易导致一线程去拿菜然后二线程也去拿菜,一已经把菜拿在手上了,二线程还在拿一已经拿在手上的菜,即因为线程执行速度过快,调用全局变量时,某个资源并不会按正常所想的方式执行,就是买菜这个方法执行到一半还没有完全执行结束,所以多个线程同时进入了会导致很大的问题所以要线程同步.

代码执行顺序

非静态代码块

直接由 { } 包起来的代码，称为非静态代码块

1

静态代码块

直接由 static { } 包起来的代码，称为静态代码块

1形参

比如你定义一个函数void add（int a, int b），这里的a和b就是形参。

当你进行函数调用的时候，add（1, 2），这里的1和2就是实参。

向前引用

所谓向前引用，就是在定义类、接口、方法、变量之前使用它们。

成员变量

在类体里面定义的变量称为成员变量；

如果该成员变量有 static 关键字修饰，则该成员变量称为 静态变量 或 类变量；

如果该成员变量没有 static 关键字修饰，则该成员变量被称为 非静态变量 或 实例变量。

局部变量

形参、方法内定义的变量、代码块中定义的变量，都属于局部变量。

类变量 （静态变量）

可以向前引用

变量属于类本身

类变量不依赖类的实例，类变量只在初始化时候在方法区中被分配一次空间，无论类的实例被创建几次，都不再为类变量分配空间

通过类的任意一个实例来访问类变量，底层都将将其转为通过类本身来访问类变量，它们的效果是一样的

一旦类变量的值被改变，通过类或类的任意一个实例来访问类变量，得到的都将是被改变后的值

将在类的初始化之前初始化

1

2

3

4

5

6

实例变量（非静态变量）

不能向前引用，如果向前引用，则称为非法向前引用，这是不允许的

变量属于类的实例对象

随着类的实例被创建而分配内存空间

java代码初始化顺序

由 static 关键字修饰的（如：类变量(静态变量)、静态代码块）将在类被初始化创建实例对象之前被初始化，而且是按顺序从上到下依次被执行。静态（类变量、静态代码块）属于类本身，不依赖于类的实例。

没有 static 关键字修饰的（如：实例变量(非静态变量)、非静态代码块）初始化实际上是会被提取到类的构造器中被执行的，但是会比类构造器中的代码块优先执行到，非静态（实例变量、非静态代码块）的地位是相等的，它们将按顺序被执行。

类变量（静态变量）、实例变量（非静态变量）、静态代码块、非静态代码块的初始化时机

由 static 关键字修饰的（如：类变量[静态变量]、静态代码块）将在类被初始化创建实例对象之前被初始化，而且是按顺序从上到下依次被执行；

没有 static 关键字修饰的（如：实例变量[非静态变量]、非静态代码块）初始化实际上是会被提取到类的构造器中被执行的，但是会比类构造器中的 代码块优先执行到，其也是按顺序从上到下依次被执行。

容易混淆的一个知识点

静态方法只允许直接访问静态成员，而实例方法中可以访问静态成员和实例成员，原因是类还没有实例化，所实例成员也没有被创建，静态方法中因此也不能用this。

泛型