`JVM学习

JAVA的JVM的内存可分为3个区：堆(heap)、栈(stack)和方法区(method)

栈区:

1. 每个线程包含一个栈区,栈中只保存方法中(不包括对象的成员变量)的基础数据类型和自定义对象的引用(不是对象),对象都存放在堆中
2. 每个栈中的数据(原始类型和对象引用)都是私有的,其它栈不能访问
3. 栈分为三个部分:基本类型变量区,执行环境上下文,操作指令区(存放操作指令)

堆区:

1. 存储的全部是对象实例,每个对象都包含一个与之对应的class信息(class信息存放在方法区中)
2. jvm只有一个堆区(heap)被所有线程共享,堆中不存放基本类型和对象引用,只存放对象本身,几乎所有的对象实例和数组都在堆中分配

方法区:

又叫静态区,跟堆一样,被所有线程共享.它用于存储已经被虚拟机加载的类信息,常量,静态变量,即时编译器编译后的代码等数据.

虚拟机类加载机制:

初始化:

1.new 实例化对象,调用一个类的静态方法.

2.reflect反射的时候.

3.初始化类的时候先初始化父类,对于接口则是在使用到父接口的时候才会初始化.

4.启动方法main

对于静态字段,只有直接定义了这个字段的类才会被初始化,static,子类继承并不会.

定义数组并不会初始化类,详情见项目dailyDemo.设置成final常量的类,调用常量时不会初始化,因为编译的时候已经放到常量池中.

JVM启动准备阶段:

在方法区中分配类变量的内存,被static修饰的变量,并赋值为变量类型的零值,如下

public static int value = 123; --------🡪 赋值0 具体值需要在类初始化时执行.

如果类变量被final修饰成为constantValue 则在准备阶段赋值.

类加载器的双亲委派模型:

类加载器的继承关系: 启动类加载器🡪扩展类加载器🡪应用程序类加载器🡪自定义加载器

子类想要加载某一个class文件的时候会将请求委派给父级加载器最后传送到顶层,如果顶层加载器无法加载才会提示子类加载器去加载.这样可以避免应用程序的混乱.

变量的初始值:

类变量在准备阶段的时候会给变量赋初始值int为0,boolean为false等,在初始化阶段则会赋值代码定义的值.

局部变量如果不赋初始值则会编译失败,需要初始化赋值.

方法的退出:

一个方法的退出有两种方式,一个是返回到上层调用,是否有返回值都属于正常退出.

另一种方式是异常退出,如果方法执行产生异常并且没有在方法中得到处理,则导致方法退出,并且不会给上层调用者任何返回信息.

退出过程:

退出当前栈帧---🡪回复上层局部变量表和操作数栈-🡪返回值压如调用者操作数栈

虚方法和非虚方法:

非虚方法意思是在类加载的时候就会把符号引用解析为该方法的直接引用.

非虚方法包括: 静态方法,私有方法,实例构造器,父类方法,还有final修饰的不可变方法.其它则为虚方法.

分派(dispatch)

静态分派 ---- 重载(overload) 编译期确定

Human man = new Man(); Human 静态类型, Man 实际类型,编译阶段只根据静态类型判断.

动态分派 --- 重写(override) 运行期确定

Human man = new Man(); 根据man类型判断.

静态分派属于多分派,动态分派属于单分派.

运行期优化:

即时编译器(JIT) 先编译针对调用过多的方法以及执行更多的循环体.采用方法调用计数器和回边计数器统计.

编译优化:方法内联,将需要调用的方法放到内部进行.对于非虚方法操作简单,但对于虚方法则无法确定,需要进行继承关系分析,如果有多个版本的目标方法可供选择,则用内联缓存,直到缓存到不一致取消内联.

逃逸分析:

1. 方法逃逸:对象是否作为调用参数传递到其它方法中.
2. 线程逃逸:是否会被外部的线程访问到.

如果确定不会发生方法逃逸,则可以是用栈上分配,这样对象随着栈帧出栈而销毁,减少消耗.

如果确定不会发生线程逃逸,则可以取消对象的线程同步.