解析Java类和对象的初始化过程 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/145/2021_2022__E8_A7_A3_E6_9E_90Java_c104_145334.htm 类的初始化和对象初始化是JVM 管理的类型生命周期中非常重要的两个环节，Google 了一遍网络，有关类装载机制的文章倒是不少，然而类初始化和对象初始化的文章并不多，特别是从字节码和 JVM 层次来分析的文章更是鲜有所见。 本文主要对类和对象初始化全过程进行分析，通过一个实际问题引入，将源代码转换成 JVM 字节码后，对 JVM 执行过程的关键点进行全面解析，并在文中穿插入了相关 JVM 规范和 JVM 的部分内部理论知识，以理论与实际结合的方式介绍对象初始化和类初始化之间的协作以及可能存在的冲突问题。 问题引入 近日我在调试一个枚举类型的解析器程序，该解析器是将数据库内一万多条枚举代码装载到缓存中，为了实现快速定位枚举代码和具体枚举类别的所有枚举元素，该类在装载枚举代码的同时对其采取两种策略建立内存索引。由于该类是一个公共服务类，在程序各个层面都会使用到它，因此我将它实现为一个单例类。这个类在我调整类实例化语句位置之前运行正常，但当我把该类实例化语句调整到静态初始化语句之前时，我的程序不再为我工作了。 下面是经过我简化后的示例代码：［清单一］package com.ccb.framework.enums.import java.util.Collections.import java.util.HashMap.import java.util.Map.public class CachingEnumResolver { //单态实例 一切问题皆由此行引起 private static final CachingEnumResolver SINGLE_ENUM_RESOLVER = new CachingEnumResolver().

/*MSGCODE->Category内存索引*/ private static Map CODE_MAP_CACHE. static { CODE_MAP_CACHE = new HashMap(). //为了说明问题,我在这里初始化一条数据 CODE_MAP_CACHE.put("0","北京市"). } //private, for single instance private CachingEnumResolver() { //初始化加载数据 引起问题，该方法也要负点责任 initEnums(). } /** * 初始化所有的枚举类型 */ public static void initEnums() { // ~~~~~~~~~~问题从这里开始暴露 ~~~~~~~~~~~~// if (null == CODE_MAP_CACHE) { System.out.println("CODE_MAP_CACHE为空,问题在这里开始暴露."). CODE_MAP_CACHE = new HashMap(). } CODE_MAP_CACHE.put("1", "北京市"). CODE_MAP_CACHE.put("2", "云南省"). //..... other code... } public Map getCache() { return Collections.unmodifiableMap(CODE_MAP_CACHE). } /** * 获取单态实例 * @return */ public static CachingEnumResolver getInstance() { return SINGLE_ENUM_RESOLVER. } public static void main(String[] args) { System.out.println(CachingEnumResolver.getInstance().getCache()). }} 想必大家看了上面的代码后会感觉有些茫然，这个类看起来没有问题啊，这的确属于典型的饿汉式单态模式啊，怎么会有问题呢？ 是的，他看起来的确没有问题，可是如果将他 run 起来时，其结果是他不会为你正确 work。运行该类，它的执行结果是：［清单二］ CODE_MAP_CACHE为空,问题在这里开始暴露.{0=北京市} 我的程序怎么会这样？为什么在 initEnum() 方法里 CODE_MAP_CACHE 为空？为什么我输

出的 CODE_MAP_CACHE 内容只有一个元素，其它两个元素呢？？？？！！看到这里，如果是你在调试该程序，你此刻一定觉得很奇怪，难道是我的 Jvm 有问题吗？非也！如果不是，那我的程序是怎么了？这绝对不是我想要的结果。可事实上无论怎么修改 initEnum() 方法都无济于事，起码我最初是一定不会怀疑到问题可能出在创建 CachingEnumResolver 实例这一环节上。正是因为我太相信我创建 CachingEnumResolver 实例的方法，加之对 Java 类初始化与对象实例化底层原理理解有所偏差，使我为此付出了三、四个小时--约半个工作日的大好青春。 那么问题究竟出在哪里呢？为什么会出现这样的怪事呢？在解决这个问题之前，先让我们来了解一下JVM的类和对象初始化的底层机制。 类的生命周期图展示的是类生命周期流向；在本文里，我只打算谈谈类的"初始化"以及"对象实例化"两个阶段。 类初始化 类"初始化"阶段，它是一个类或接口被首次使用的前阶段中的最后一项工作，本阶段负责为类变量赋予正确的初始值。 Java 编译器把所有的类变量初始化语句和类型的静态初始化器通通收集到 方法内，该方法只能被 Jvm 调用，专门承担初始化工作。 除接口以外，初始化一个类之前必须保证其直接超类已被初始化，并且该初始化过程是由 Jvm 保证线程安全的。另外，并非所有的类都会拥有一个 () 方法，在以下条件中该类不会拥有 () 方法： 该类既没有声明任何类变量，也没有静态初始化语句；该类声明了类变量，但没有明确使用类变量初始化语句或静态初始化语句初始化；该类仅包含静态 final 变量的类变量初始化语句，并且类变量初始化语句是编译时常量表达式。 对象初始化 在类被装载、连接和初始化，这个类

就随时都可能使用了。对象实例化和初始化是就是对象生命的起始阶段的活动，在这里我们主要讨论对象的初始化工作的相关特点。Java 编译器在编译每个类时都会为该类至少生成一个实例初始化方法--即 "()" 方法。此方法与源代码中的每个构造方法相对应，如果类没有明确地声明任何构造方法，编译器则为该类生成一个默认的无参构造方法，这个默认的构造器仅仅调用父类的无参构造器，与此同时也会生成一个与默认构造方法对应的 "()" 方法. 通常来说，() 方法内包括的代码内容大概为：调用另一个 () 方法；对实例变量初始化；与其对应的构造方法内的代码。 如果构造方法是明确地从调用同一个类中的另一个构造方法开始，那它对应的 () 方法体内包括的内容为：一个对本类的 () 方法的调用；对应用构造方法内的所有字节码。 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com