JVM | 类的初始化及新建过程

作者：xiaodaojava
原文链接：https://ld246.com/article/1580464223016
来源网站：链滴

本文知识点

- 类的状态变化
- <clinit> 方法
- 实例对象的创建

类的状态变化

类的初始化主要经历加载-> 链接(验证，准备，解析)-> 初始化这些阶段，与 JVM 中相对应的状态如图所示

instanceKlass.hpp

```
131   // See "The Java Virtual Machine Specification" section 2.16.2-5 for a detailed description
132   // of the class loading & initialization procedure, and the use of the states.
133   enum ClassState {
134     allocated,                          // allocated (but not yet linked)
135     loaded,                             // loaded and inserted in class hierarchy (but not linked ye
136     linked,                             // successfully linked/verified (but not initialized yet)
137     being_initialized,                  // currently running class initializer
138     fully_initialized,                  // initialized (successfull final state)
139     initialization_error                // error happened during initialization
140   };
141
```

allocated: 已分配，但尚未链接

loaded: 已加载，并插入到 JVM 内部类层次体系中，但尚未链接

linked: 已链接，但尚未初始化

being_initialized: 初始化中

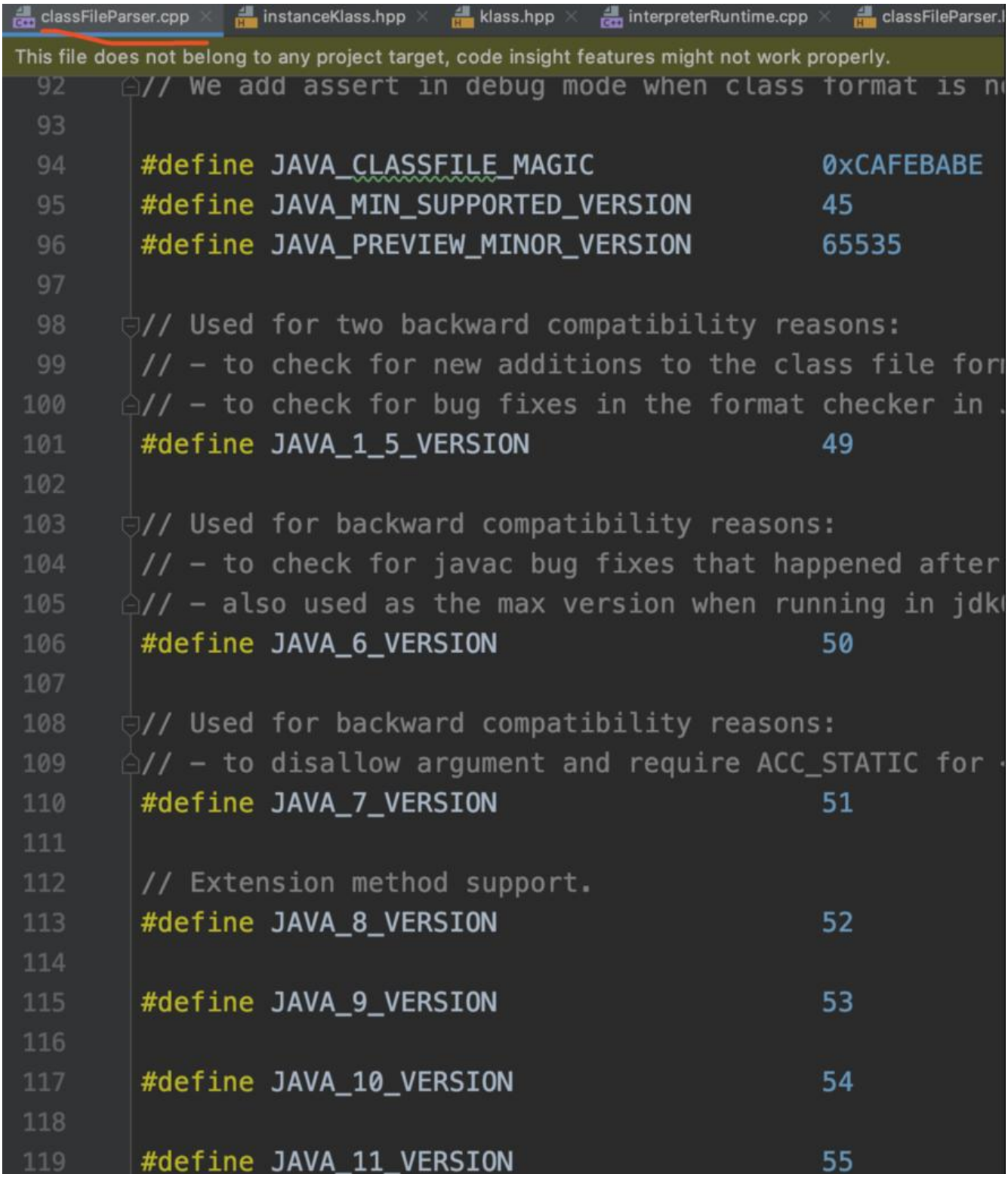
fully_initialized: 完成初始化

initialization_error: 初始化过程中出错

加载

.class 文件是个二进制文件，我们可以点开。class 文件，可以看到各种二进制信息， 右边转成的字不是很全， 有很多标识位，直接用数字表示的。 右边能看到的， 基本上都是常量池字符串里面的信息

加载。class 的源码在 classFileParser.cpp 中，如下图所示：



在上图中， 我们可以看到， 有 CAFEBABE 的定义，版本号的定义， 在往下， 我们可以看到对 class 文件中的常量池，附录表等解析方法，在此就不在赘述

链接

如我们在 out/build 或者别的输出目录中所看到的， class 文件都是单独的， class 文件中有本类用的各种静态常量池。 在 jvm 中还有一个运行时常量池，是各个 class 都可以访问的。 因为链接最主的就是把 class 文件中的静态常量池和运行时常量池关联起来， 把静态符号引用，转成直接内存引用

然后我们就可以通过地址调用相应的方法，完成操作

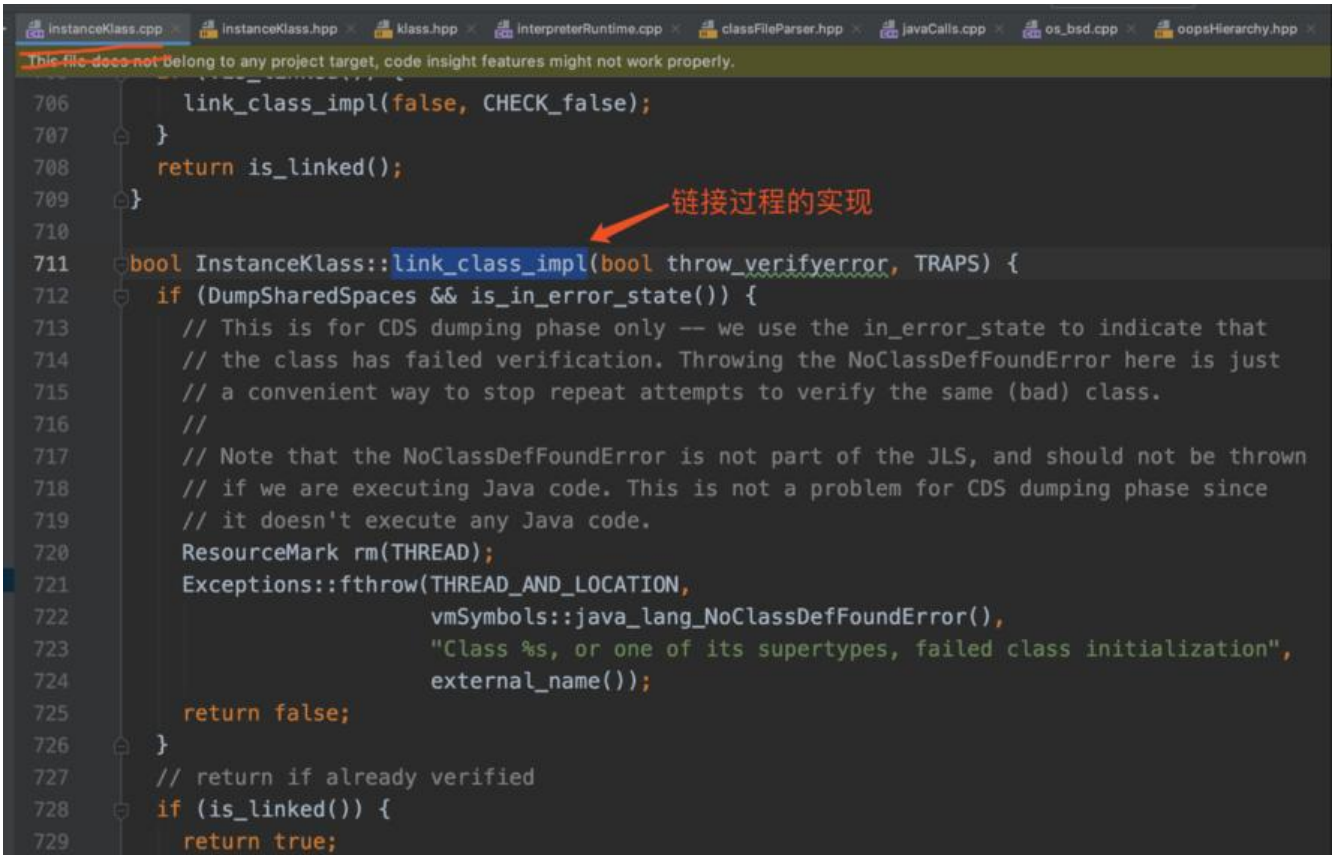
链接有三大步，验证， 准备，解析。

验证： 类或接口的二进制信息是否正确， 方法的访问控制， 变量是否初始化等。 通常来说， 只要们写代码时 ide 不报错， 基本上就没什么问题， 但有些会自己构造。class 文件，交由 jvm 运行，以要验证各种正确性

准备： 在类的准备阶段，将为类静态变量分配内存空间，和赋初始值，但是要注意， 这时候还没有行任何赋值的代码或者静态代码块！

解析： 如上所述， 把 class 文件中的静态变量池和 jvm 内部的运行池给关联起来， 把符号引用换成接引用

源码位置如下图所示：



clinit 方法

clinit 方法是**初始化**的关键所在

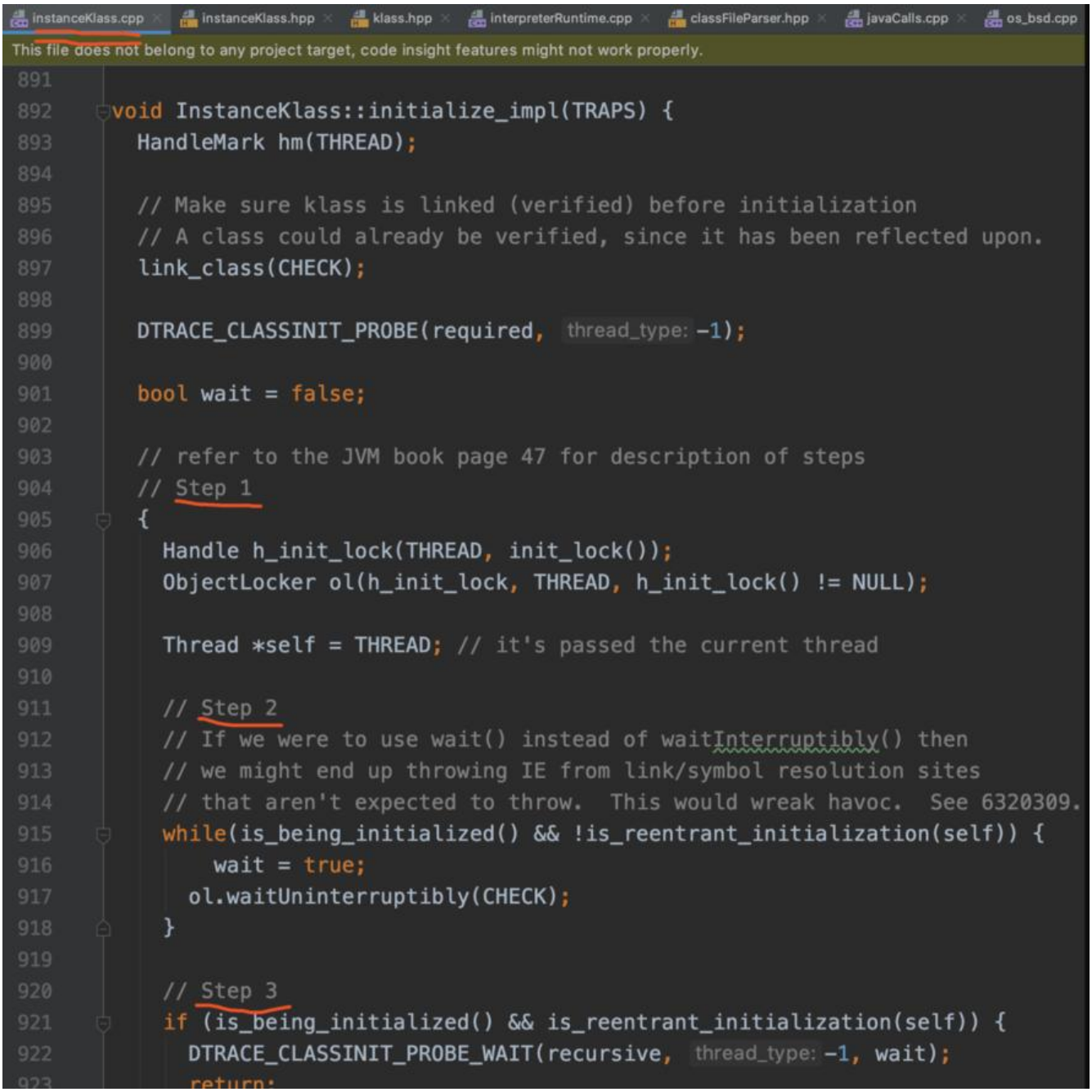
这个方法， 我们在 Java 源代码中没有看到过，该方法只能由 javac 编译器自动生成和命名，然后自插入到 Class 文件中。

clinit 方法由编译器收集类变量(静态非 final),static 代码块

clinit 方法没有任何虚拟机字节码指令可以调用， 它**只能**在类型初始化阶段被虚拟机隐式调用，全程调用一次

如果有继承的话，会先初始化父类

其源码如下：



如上图所示，有多个步骤，每个步骤的注释也十分清晰， 强烈建议小伙伴们把源码拉下来阅读一下

其实父类优先于子类初始化，可以步骤 7 和步骤 8 中看到，如下图所示：

```
954     // Step 7
955     // Next, if C is a class rather than an interface, initialize it's super class and super
956     // interfaces.
957     if (!is_interface()) {...}
984
985
986     // Look for aot compiled methods for this klass, including class initializer.
987     AOTLoader::load_for_klass(this, THREAD);
988
989     // Step 8
990     {
991       assert(THREAD->is_Java_thread(), "non-JavaThread in initialize_impl");
992       JavaThread* jt = (JavaThread*)THREAD;
993       DTRACE_CLASSINIT_PROBE_WAIT(clinit,  thread_type: -1, wait);
994       // Timer includes any side effects of class initialization (resolution,
995       // etc), but not recursive entry into call_class_initializer().
996       PerfClassTraceTime timer(ClassLoader::perf_class_init_time(),
997                                ClassLoader::perf_class_init_selftime(),
998                                ClassLoader::perf_classes_inited(),
999                                jt->get_thread_stat()->perf_recursion_counts_addr(),
1000                               jt->get_thread_stat()->perf_timers_addr(),
1001                               PerfClassTraceTime::CLASS_CLINIT);
1002      call_class_initializer(THREAD);
1003    }
```

调用 clinit

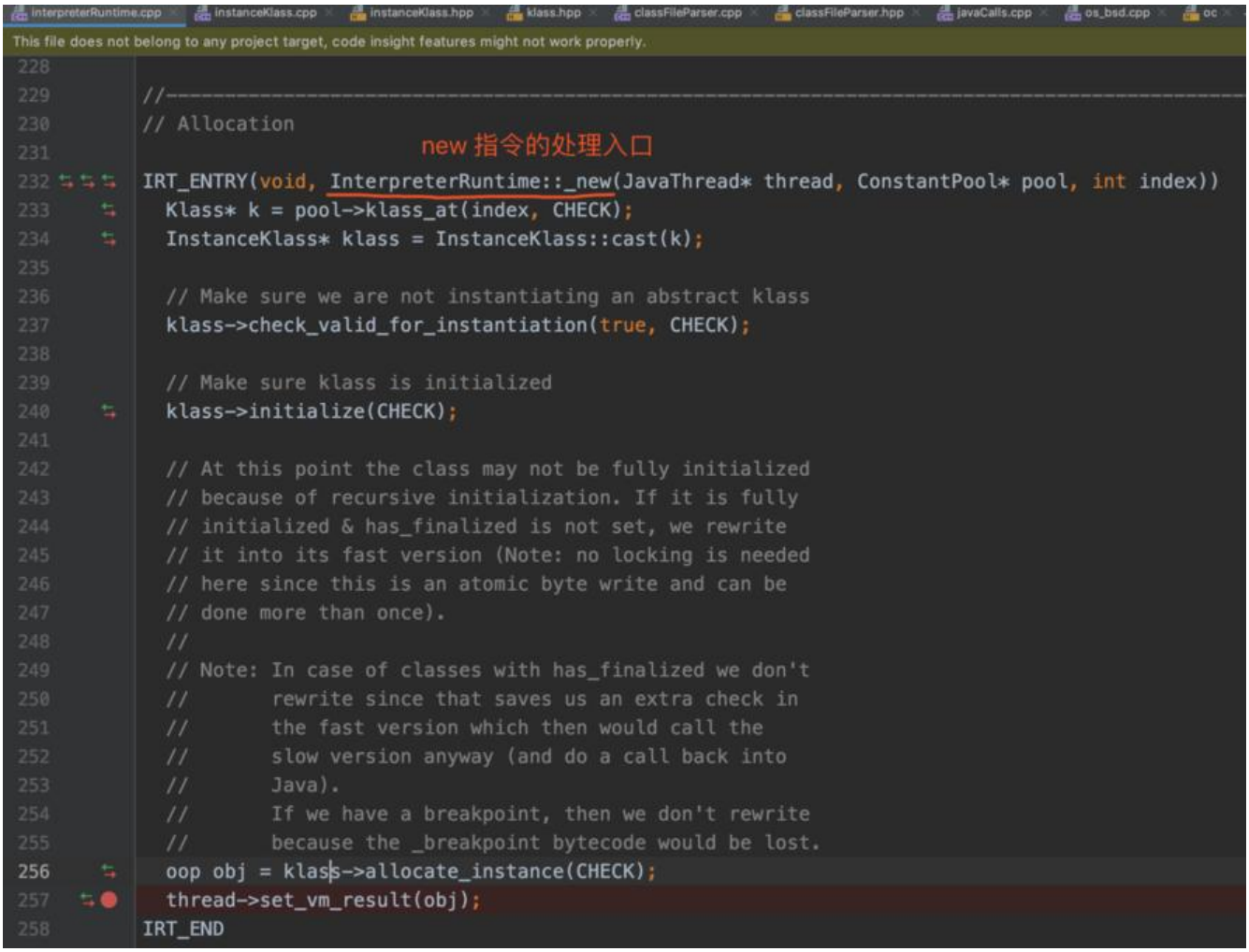
实例对象的创建

实例对象的创建， 这一块相对来说就简单了， 虚拟机遇到 new 的时候， 从栈顶取得目标对象在常池中的索引，接着定位到目标类型的类型，接下来，虚拟机看是否已加载采用 tlabs/慢速分配(Eden)一块空地， 然后完成实例数据和对象头的初始化。

流程就是上面个流程，其实也没啥复杂的， 就像我们买东西， 在京东上看了图片(klass) ,然后就买了个回来(有自己的实例), 如果快递配送的很快，还没来得及想好放哪(还没加载这个类),那就先丢到仓库(den 区),已经想好怎么放的话(已加载了这个类),那就顺手就给安排了(使用 TLABS 来分配).

其中要注意的一点就是。一但选好放哪里之后， 就开始在自己的小本本上更新，XXX 东西被我放在了 XXX 地址。 即使现在还没有走过去把东西放下， 别人问的时候，已经可以用那个地址去回答别人了。

源码入口如下图所示，有兴趣的小伙伴，可以沿着这个入口，深入跟踪下去， 小刀后面也会和大家一再次看这些地方的！ 加油



总结

说一点我自己的学习小结， 就是背概念只能记一时， 比如以前静态变量， 静态赋值，构造函数的执顺序， 以前看了很多博客，也背了很多次， 也就这次跟着源码一步步走下来， 印象更觉深刻，忘肯还会忘的，但我相信这次的记忆会更清晰，会更深刻，一起加油!