# 概述

Java虚拟机是一个需要不断学习的、偏底层的知识，不断完善自己的学习笔记。

# 对象创建

本章节以HotSpot虚拟机为例，简单介绍对象的创建过程。

## 对象创建过程

* 判断类是否加载，如果没有加载，先加载，否则执行下一步
* 申请内存，根据类信息在堆空间申请指定大小的内存空间；
* 内存空间清零；
* 调用类的init方法（注意init和clinit的区别）；
* 调用构造方法（可选）；

## 对象内存布局

一个对象在内存中的布局分为三个部分：

* 对象头
* 实例数据
* 对齐填充（非必须）

# 垃圾回收（Garbage Collection）

## 垃圾回收区域

* 堆
* 方法区

## 判断对象是否可回收

* 引用计数法，缺点是无法解决循环引用问题；
* 可达性分析法，从gc root节点向下搜索，没搜索一个节点，则标记该对象不能被回收。

Gc root包括如下几类（分别来自于方法区、虚拟机栈区、本地方法栈）：

* + 方法区Static变量引用的对象
  + 方法区的常量引用的对象
  + 虚拟机栈中本地栈帧中引用的对象
  + 本地方法栈中引用的对象；

## 引用

* 强引用：如果一对象被强引用，JVM垃圾回收器将不会回收该对象，即使是内存不足，JVM宁可抛出异常OutOfMemory,也不会回收该对象；
* 软引用：被软引用的对象，如果JVM内存不够使用，JVM将把被软引用的对象放入垃圾回收器中进行重新回收，如果再次回收之后内存仍然不够使用，将抛出OutOfMemory异常。如果GC回收时，内存足够使用，则被软引用的对象不会有影响；
* 弱引用：被弱引用的对象，GC回收器回收垃圾对象时，无论当前内存是否够用，被弱引用的对象都会被回收；
* 虚引用：

## 回收算法

* 标记-清除：首先找出未被引用的对象并标记，接下来把所有标记的对象从内存清除（导致内存碎片）；
* 复制算法：将内存平分为两块，一块内存使用完之后进行对象标记，接下来将未被标记的对象复制到另一块内存（内存使用率最大为50%）；
* 标记-整理：先进行标记，然后将存活的对象往一侧移动，
* 分代收集：将内存分为老年代和新生代，新生代采用复制算法，老年代采用标记-清除或者标记-整理算法

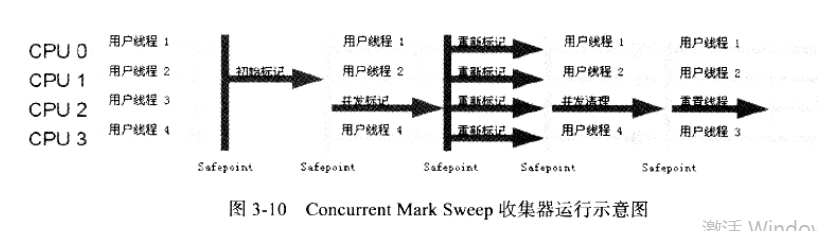
## 垃圾收集器

主要分新生代垃圾收集器，老年代垃圾收集器，没有万能的收集器，只有合适的收集器。重点掌握CMS（Concurrent Mark Sweep）和G1（Garbage First）收集器；

### CMS收集器

四个过程：

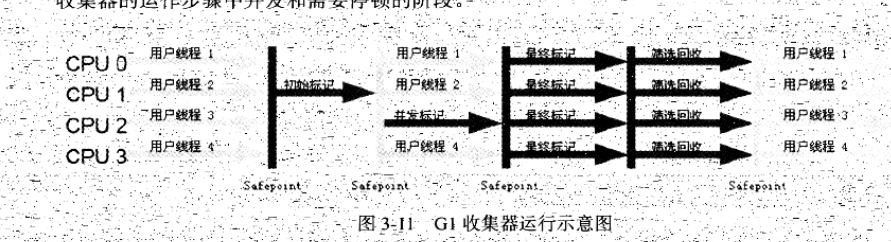
* 初始标记
* 并发标记
* 重新标记
* 并发清除



## G1收集器

收集过程：

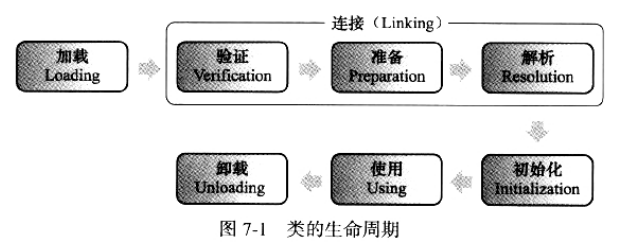
* 初始标记
* 并发标记
* 最终标记
* 筛选回收



# 类加载

## 概述

类生命周期7个主要过程如下图：



* 加载
* 验证
* 准备
* 解析
* 初始化
* 使用
* 卸载

类的加载按照这个顺序逐个进行，但是不代表后续阶段必须在前一个阶段执行完之后才开始，各个阶段也可能交替进行。接下来需要分别阐述各个生命期。

## 加载

对于加载阶段，虚拟机规范没有明确指定该什么时候进行加载阶段，这个由虚拟机具体实现决定，但是加载阶段必须处于初始化之前。

加载过程：

* 通过一个类的全限定名来获取对应的class文件；
* 将class文件（特殊的数据结构）中的内容转换为方法区的数据结构内容；
* 在内存（不一定是堆区）中生成一个java.lang.Class的对象，该对象作为访问该类在方法区中所有内容的入口

## 验证

验证的目的是为了使加载进虚拟机的类文件是安全的，以免破坏虚拟机系统。

## 准备

准备阶段主要完成类中静态变量内存的分配，并且为类变量赋“零值”。但是也存在例外，如果类变量被final修饰，该静态变量在分配内存时直接赋值为被指定的值；

## 解析

解析阶段主要完成常量池中的符号引用替换为直接引用。比如讲常量符号替换为具体的值。

## 初始化

首先需要理解初始化所做的工作，初始化就是对类中static变量的初始化、static代码的执行。

虚拟机规范明确指出如下五种情况将进行初始化阶段，除了这5中情况外，其他情况都不会触发初始化：

* 遇到new，getstatic，putstatic和invokestatic字节码指令时，如果类没有进行初始化，则需要触发初始化。对应这些指令的java代码为：java中的new操作，读取或者设置一个static field、调用static方法；
* 通过java.lang.reflect包中的方法对类进行调用时，如果该类没有进行初始，则需要先对其初始化；
* 当初始化一个类时，如果发现其父类还没有初始化，则需要先初始化其父类；
* 项目主类（包含main方法的类），虚拟机会初始化这个类；
* 当使用动态语言（暂时没理解）

初始化的具体实现：初始化阶段就是执行类的<clinit>构造方法的过程。

* <clinit>方法由编译器生成，其中的代码是类变量的赋值动作，静态语句块按照在源码中的顺序构造而成的。静态语句块只能访问在该语句块之前的静态变量，定义在静态语句块之后的类变量只能赋值，不能访问。
* <clinit>构造器不用调用父类的<clinit>，虚拟机会保证在执行一个<clinit>之前，父类的<clinit>已经执行过了；这一点和构造方法（或者<init>方法）不同，构造方法可以执行多次，并且是在创建类对象时执行，同时执行构造方法时，需要调用父类的构造方法。
* <clinit>构造器是线程安全的，虚拟机会保证在多线程时，<clinit>只被执行一次；

## 使用

通过加载好的类创建实例并调用其中的方法；

## 卸载

如果一个类不在由任何实例化对象，java垃圾回收器才可能把该类在方法区对应的类结构卸载，包括加载时创建的类对象Class；

# 类加载器

## 双亲委派模型

## 破坏双亲委派

# 参考

《深入理解Java虚拟机：JVM高级特性与最佳实践》