http://blog.brucefeng.info/post/jvm-notes#toc_141

JVM学习笔记

于 2015 年 07 月 16 日

- 运行时数据区
  - 程序计数器（ProgramCounter Register）
    - 每个线程都有独立的程序计数器
    - 一块较小的内存空间，可以看作当前线程所执行的字节码的行号指示器
  - 虚拟机栈（VM Stack)
    - 线程私有，生命周期与线程相同
    - 每个方法执行时都会创建一个帧栈，用于存储局部变量表、操作数栈、动态链接、方法出口等信息
      - 局部变量表存放了编译期可知的各种基本数据类型，对象引用和returnAddress
  - 本地方法栈(Native Method Stack)
    - 与虚拟机栈类似，但只为Native方法服务
  - 方法区（Method Area)
    - 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
    - Permanent Generation（保存虚拟机自己的静态(refective)数据
  - 堆（Heap)
    - 被所有线程共享的一块内存区域
    - 也成为GC堆，是垃圾收集器管理的主要区域
    - 内存分配与回收策略
      - 堆内存布局
        - Eden:存放新生的对象
        - # 对象优先分配至Eden区，当空间不足时，将触发MinorGC
        - # MinorGC：新生代GC，指发生在新生代的拉圾收集动作，MinorGC非常频繁，一般回收速度也非常快
        - Survivor Space，主要用于存储垃圾回收之后的存活对

象
        - Old Generation：用于存放生命周期较长的大对象
        - # MajorGC/FullGC(老年代GC): 发生在老年代的GC,出现了MajorGC，通常伴随至少一次MinorGC,MajorGC速度通常比MinorGC慢10倍以上。
        - # 大对象
        - # 长期存活的对象
        - ## 对象每在Survivor经历一次MinorGC Age增加1，当增加到15时就直接晋升到老年代
        - ## 如果在Survivor空间中相同年龄所有对象大小的总和大于Suvivor空间的一半，年龄大于或等于该年龄段对象就可以直接进入老年代
    - 内存布局
- 执行引擎
    - 类加载
        - 类加载时机（必须进行初始化的时机）
            - 遇到new,getstatic,putstatic,invokestatic四条指令时，如果类没有进行初始化，需要初始化操作
                - 使用new关键字初始化对象
                - 读取或设置一个类的静态字段（被final修饰，或者编译器把结果放入静态池的静态字段除外）
                - 调用一个类的静态方法时
            - 当初始化一个类的时候，发现其父类还没有进行初始化
            - 使用java.lang.reflect包的方法对类进行反射调用的时候
            - 虚拟机启动时，用户需要指定要执行的主类，虚拟机会先初始化这个主类
            - 当使用JDK7以上的动态语言支持
        - 类加载过程
            - 加载（Loading)
                - 通过一个类的权限定名获取定义此类的二进制字节流
                - # zip包等
                - # 网络中获取
                - # 运行时计算生成，用的最多的为动态代理技术
                - # 其他文件生成，典型如JSP
                - # 数据库中读取
                - 将这个字节流所代表的静态存储结构转化为方法区的运行时数据结构

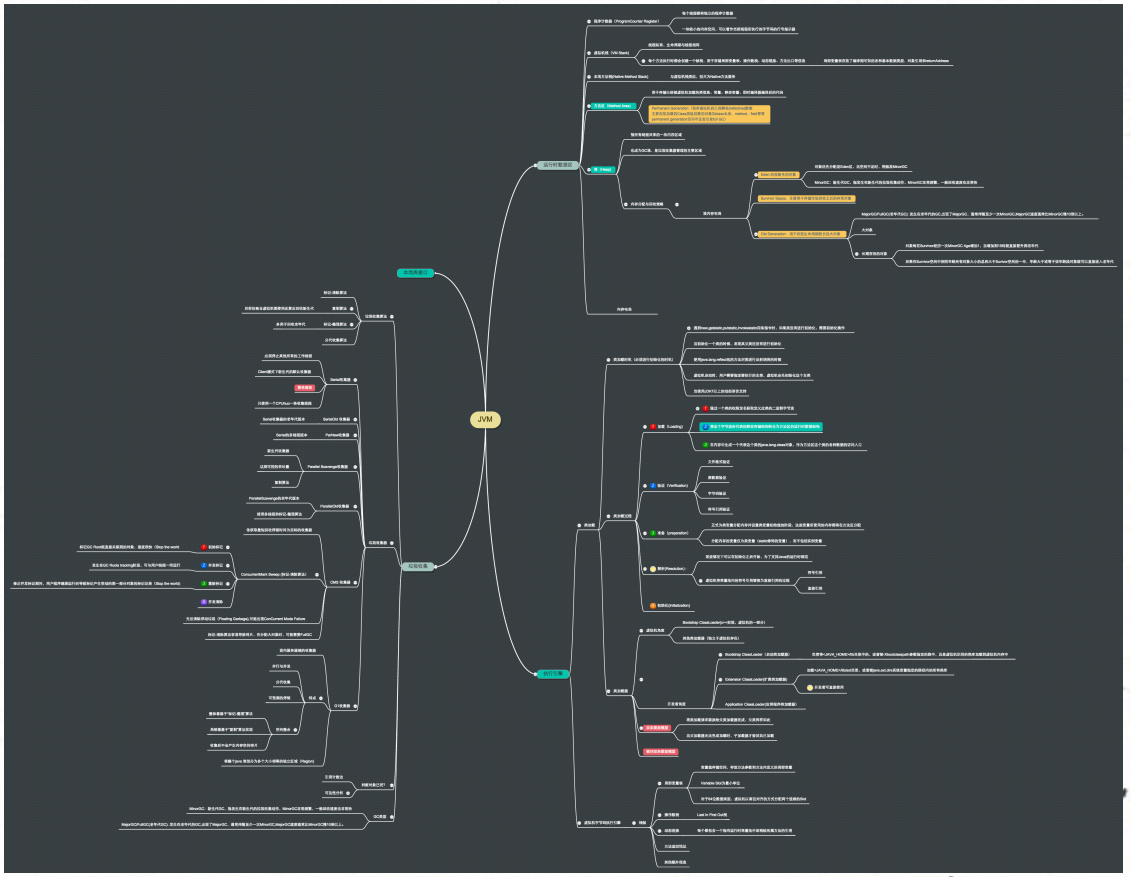
        - 在内存中生成一个代表这个类的java.lang.class对象，作为方法区这个类的各种数据的访问入口
    - 验证（Verification)
        - 文件格式验证
        - 原数据验证
        - 字节码验证
        - 符号引用验证
    - 准备（preparation）
        - 正式为类变量分配内存并设置类变量初始值的阶段，这些变量所使用的内存都将在方法区分配
        - 分配内存的变量仅为类变量（static修饰的变量），而不包括实例变量
    - 解析(Resolution) :
        - 某些情况下可以在初始化之后开始，为了支持Java的运行时绑定
        - 虚拟机将常量池内的符号引用替换为直接引用的过程
        - # 符号引用
        - # 直接引用
    - 初始化(Initialization)
- 类加载器
    - 虚拟机角度
        - Bootstrap ClassLoader(c++实现，虚拟机的一部分）
        - 其他类加载器（独立于虚拟机存在）
    - 开发者角度
        - Bootstrap ClassLoader（启动类加载器）
        - # 负责将/lib目录中的，或者被-Xbootclasspath参数指定的路中，且是虚拟机识别的类库加载到虚拟机内存中
        - Extension ClassLoader(扩展类加载器)
        - # 加载/lib/ext目录，或者被java.ext.dirs系统变量指定的路径内的所有类库
        - # 开发者可直接使用
        - Application ClassLoader(应用程序类加载器）
    - 双亲委派模型
        - 将类加载请求委派给父类加载器完成，父类同样如此
        - 当父加载器无法完成加载时，子加载器才尝试自己加载
    - 破坏双亲委派模型
- 虚拟机字节码执行引擎

        - 栈帧
            - 局部变量表
                - 变量值存储空间，存放方法参数和方法内定义的局部变量
                - Variable Slot为最小单位
                - 对于64位数据类型，虚拟机以高位对齐的方式分配两个连续的Slot
            - 操作数栈
                - Last In First Out栈
            - 动态连接
                - 每个都包含一个指向运行时常量池中该栈帧所属方法的引用
            - 方法返回地址
            - 其他额外信息
- 本地库接口
- 垃圾收集
    - 垃圾收集算法
        - 标记-清除算法
        - 复制算法
            - 目前的商业虚拟机都使用此算反回收新生代
        - 标记-整理算法
            - 多用于回收老年代
        - 分代收集算法
    - 垃圾收集器
        - Serial收集器
            - 必须停止其他所有的工作线程
            - Client模式下新生代的默认收集器
            - 简单高效
            - 只使用一个CPUhuo一条收集线程
        - SerialOld 收集器
            - Serial收集器的老年代版本
        - ParNew收集器
            - Serial的多线程版本
        - Parallel Scavenge收集器
            - 新生代收集器
            - 达到可控的吞吐量
            - 复制算法

- ParallelOld收集器
  - ParallelScavenge的老年代版本
  - 使用多线程和标记-整理算法
- CMS 收集器
  - 依获取最短回收停顿时间为目标的收集器
  - ConcurrentMark Sweep (标记-清除算法）
    - 初始标记
    - # 标记GC Root能直接关联到的对象，速度很快（Stop the world
    - 并发标记
    - # 发生在GC Roots tracking阶段，可与用户线程一同运行
    - 重新标记
    - # 修正并发标记期间，用户程序继续运行而导致标记产生变动的那一部分对象的标记记录（Stop the world)
    - 并发清除
  - 无法清除浮动垃圾（Floating Garbage),可能出现ConCurrent Mode Failure
  - 标记-清除算法容易导致碎片，在分配大对象时，可能需要FullGC
- G1收集器
  - 面向服务器端的收集器
  - 特点
    - 并行与并发
    - 分代收集
    - 可预测的停顿
    - 空间整合
    - # 整体看基于“标记-整理”算法
    - # 局部看基于“复制”算法实现
    - # 收集后不会产生内存空间碎片
  - 将整个java 堆划分为多个大小相等的独立区域（Region)
- 判断对象已死?
  - 引用计数法
  - 可达性分析
    - GCRoots?
      - 主要在全局性的引用（常量，或类静态属性）
      - 执行上下文中（如栈帧里的本地变量表）

        - 分析时，需要Stop the world,暂停所有线程的执行
    - GC类型
        - MinorGC：新生代GC，指发生在新生代的拉圾收集动作，MinorGC非常频繁，一般回收速度也非常快
        - MajorGC/FullGC(老年代GC): 发生在老年代的GC,出现了MajorGC，通常伴随至少一次MinorGC,MajorGC速度通常比MinorGC慢10倍以上。

这是学习「深入理解Java虚拟机:JVM高级特性与最佳实践(第2版)」时记录的笔记，共享出来。



图片可以直接下载： 下载地址。

运行时数据区

程序计数器（ProgramCounter Register）

每个线程都有独立的程序计数器

一块较小的内存空间，可以看作当前线程所执行的字节码的行号指示器

虚拟机栈（VM Stack)

线程私有，生命周期与线程相同

每个方法执行时都会创建一个帧栈，用于存储局部变量表、操作数栈、动态链接、方法出口等信息

局部变量表存放了编译期可知的各种基本数据类型，对象引用和returnAddress

本地方法栈(Native Method Stack)

与虚拟机栈类似，但只为Native方法服务

方法区（Method Area)

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

Permanent Generation（保存虚拟机自己的静态(refective)数据

主要存放加载的Class类级别静态对象如class本身，method，field等等

permanent generation空间不足会引发full GC）

堆（Heap)

被所有线程共享的一块内存区域

也成为GC堆，是垃圾收集器管理的主要区域

内存分配与回收策略

堆内存布局

Eden:存放新生的对象

对象优先分配至Eden区，当空间不足时，将触发MinorGC

MinorGC：新生代GC，指发生在新生代的拉圾收集动作，MinorGC非常频繁，一般回收速度也非常快

Survivor Space，主要用于存储垃圾回收之后的存活对象

Old Generation：用于存放生命周期较长的大对象

MajorGC/FullGC(老年代GC): 发生在老年代的GC,出现了MajorGC，通常伴随至少一次MinorGC,MajorGC速度通常比MinorGC慢10倍以上。

大对象

长期存活的对象

对象每在Survivor经历一次MinorGC Age增加1，当增加到15时就直接晋升到老年代

如果在Survivor空间中相同年龄所有对象大小的总和大于Suvivor空间的一半，年龄大于或等于该年龄段对象就可以直接进入老年代

内存布局

执行引擎

类加载

类加载时机（必须进行初始化的时机）

遇到new,getstatic,putstatic,invokestatic四条指令时，如果类没有进行初始化，需要初始化操作

使用new关键字初始化对象

读取或设置一个类的静态字段（被final修饰，或者编译器把结果放入静态池的静态字段除外）

调用一个类的静态方法时

当初始化一个类的时候，发现其父类还没有进行初始化

使用java.lang.reflect包的方法对类进行反射调用的时候

虚拟机启动时，用户需要指定要执行的主类，虚拟机会先初始化这个主类

当使用JDK7以上的动态语言支持

类加载过程

加载（Loading)

通过一个类的权限定名获取定义此类的二进制字节流

zip包等

网络中获取

运行时计算生成，用的最多的为动态代理技术

其他文件生成，典型如JSP

数据库中读取

将这个字节流所代表的静态存储结构转化为方法区的运行时数据结构

在内存中生成一个代表这个类的java.lang.class对象，作为方法区这个类的各种数据的访问入口

验证（Verification)

文件格式验证
原数据验证
字节码验证
符号引用验证
准备（preparation）
正式为类变量分配内存并设置类变量初始值的阶段，这些变量所使用的内存都将在方法区分配
分配内存的变量仅为类变量（static修饰的变量），而不包括实例变量
解析(Resolution)：
某些情况下可以在初始化之后开始，为了支持Java的运行时绑定
虚拟机将常量池内的符号引用替换为直接引用的过程
符号引用
直接引用
初始化(Initialization)

类加载器

虚拟机角度
Bootstrap ClassLoader(c++实现，虚拟机的一部分）
其他类加载器（独立于虚拟机存在）
开发者角度
Bootstrap ClassLoader（启动类加载器）
负责将/lib目录中的，或者被-Xbootclasspath参数指定的路中，且是虚拟机识别的类库加载到虚拟机内存中
Extension ClassLoader(扩展类加载器)
加载/lib/ext目录，或者被java.ext.dirs系统变量指定的路径内的所有类库
开发者可直接使用
Application ClassLoader(应用程序类加载器）
双亲委派模型
将类加载请求委派给父类加载器完成，父类同样如此
当父加载器无法完成加载时，子加载器才尝试自己加载
破坏双亲委派模型

虚拟机字节码执行引擎

栈帧

局部变量表
变量值存储空间，存放方法参数和方法内定义的局部变量
Variable Slot为最小单位
对于64位数据类型，虚拟机以高位对齐的方式分配两个连续的Slot
操作数栈
Last In First Out栈
动态连接
每个都包含一个指向运行时常量池中该栈帧所属方法的引用
方法返回地址
其他额外信息

本地库接口

垃圾收集

垃圾收集算法

标记-清除算法

复制算法

目前的商业虚拟机都使用此算反回收新生代

标记-整理算法

多用于回收老年代

分代收集算法

垃圾收集器

Serial收集器

必须停止其他所有的工作线程
Client模式下新生代的默认收集器
简单高效
只使用一个CPUhuo一条收集线程

SerialOld 收集器

Serial收集器的老年代版本

ParNew收集器

Serial的多线程版本

Parallel Scavenge收集器

新生代收集器
达到可控的吞吐量
复制算法

ParallelOld收集器

ParallelScavenge的老年代版本
使用多线程和标记-整理算法

CMS 收集器

依获取最短回收停顿时间为目标的收集器
ConcurrentMark Sweep (标记-清除算法)
初始标记
标记GC Root能直接关联到的对象，速度很快 (Stop the world
并发标记
发生在GC Roots tracking阶段，可与用户线程一同运行
重新标记
修正并发标记期间，用户程序继续运行而导致标记产生变动的那一部分对象的标记记录 (Stop the world)
并发清除
无法清除浮动垃圾 (Floating Garbage),可能出现ConCurrent Mode Failure
标记-清除算法容易导致碎片，在分配大对象时，可能需要FullGC

G1收集器

面向服务器端的收集器
特点
并行与并发
分代收集
可预测的停顿
空间整合
整体看基于“标记-整理”算法
局部看基于“复制”算法实现
收集后不会产生内存空间碎片
将整个java 堆划分为多个大小相等的独立区域 (Region)

判断对象已死？

引用计数法

可达性分析

GCRoots?
主要在全局性的引用 (常量，或类静态属性)
执行上下文中 (如栈帧里的本地变量表)
分析时，需要Stop the world,暂停所有线程的执行

GC类型

MinorGC：新生代GC，指发生在新生代的拉圾收集动作，MinorGC非常频繁，一般回收速度也非常快
MajorGC/FullGC(老年代GC): 发生在老年代的GC,出现了MajorGC，通常伴随至少一次MinorGC,MajorGC速度通常比MinorGC慢10倍以上。
#Java#JVM