# 类

## 类的加载机制

虚拟机吧描述类的数据冲class文件加载到内，并队数据进行校验、转换解析的和初始化，最终形成可以被虚拟机直接使用的Java类型。

## 类的生命周期

加载、验证、准备、解析、初始化、使用、卸载

## 类的加载过程

加载、验证、准备、解析、初始化

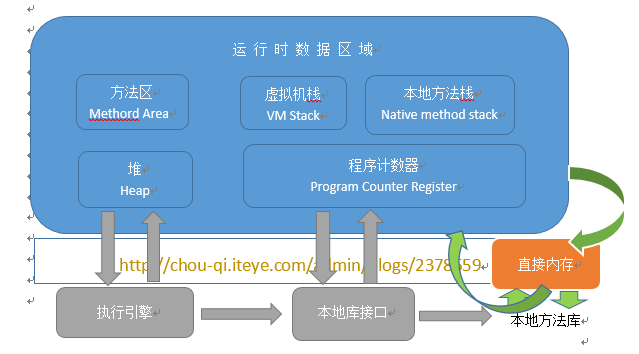
## 类加载器与双亲委派

如果一个类加载器收到了类加载的请求，它首先不会自己去尝试加载这个类，而是把这个请求委派给父类加载器去完成，每一个层次的类加载器都是如此，因此所有的加载请求最终都应该传送到顶层的启动类加载器中，只有当父类加载器反馈自己无法完成这个加载请求（它的搜索范围中没有找到所需的类）时，子加载器才会尝试自己去加载。

# JVM

## jvm内存

方法区、虚拟机栈、本地方法栈、堆、程序计数器



## 垃圾收集器

(1) Serial（串行GC）-XX:+UseSerialGC

(2) ParNew（并行GC）-XX:+UseParNewGC

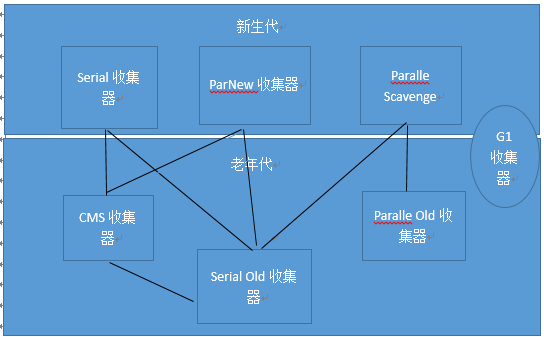
(3) Parallel Scavenge（并行回收GC）

(4) Serial Old（MSC）（串行GC）-XX:+UseSerialGC

(5) CMS（并发GC）-XX:+UseConcMarkSweepGC

(6) Parallel Old（并行GC）-XX:+UseParallelOldGC

(7) G1（JDK1.7update14才可以正式商用）



**二.1~3用于年轻代垃圾回收**：年轻代的垃圾回收称为minor GC **三.4~6用于年老代垃圾回收**（当然也可以用于方法区的回收）：年老代的垃圾回收称为full GC

G1独立完成"分代垃圾回收"

注意：并行与并发

并行：多条垃圾回收线程同时操作

并发：垃圾回收线程与用户线程一起操作

**八、调优方法**

8.1 新对象预留新生代

由于fullGC(老年代)的成本远比minorGC（新生代和老年代）的成本大，所以给应用分配一个合理的新生代空间，尽量将对象分配到新生代减小fullGC的频率  
8.2 大对象进入老年代

将大对象直接分配到老年代，保持新生代对象的结构的完整性，以提高GC效率， 以通过-XX:PretenureSizeThreshold设置进入老年代的阀值  
8.3 稳定与震荡的堆大小

稳定的对大小是对垃圾回收有利的，方法将-Xms和-Xmx的大小一致  
8.4 吞吐量优先

尽可能减少系统执行垃圾回收的总时间，故采用并行垃圾回收器

-XX:+UseParallelGC或使用-XX:+UseParallelOldGC  
8.5 降低停顿

使用CMS回收器,同时减少fullGC的次数

# 集合

## List

## Set

## Map

# 多线程

## 线程池

## 线程的作用

# Spring

## Ioc

## Aop

原理

## 事务传播

# Spring cloud

## 优点

# Dubbo

## 原理

与spring cloud 性能对比

# Redis

## 分布式锁

## 持久化

# 计算机网络

## http协议

## Tcp

## Udp

## Post、get

## Session

## Cookie

# NIO/IO

## NIO

## IO

# 数据库

## 性能优化

## 索引

## B+树

## 事务

# Neo4j

# ORM框架

## Mybatis

## Hibernate

## Jpa

# RocketMq

## 消息持久化

## 消息消费

## 分布式事务

# Zookeeper

## 服务一致问题

# 设计模式

## 单例模式

## 工厂模式

## 代理模式

## 建造者模式