离职理由  
  
业务介绍  
准入  
下单  
签署合同  
支付首付  
车辆扣减   
车辆服务  
车辆交付  
  
  
融租公司配置系统   
  
融租公司路由 目的  
  
额度扣减 有没有什么问题 优化的空间？  
  
并发量大 也有场景 比如天猫订单同步   
参与开发业务功能  
什么问题？  
录下通话 复盘   
  
mysql  
隔离级别 幻读/可重复读  
数据量怎么样 sql优化   
btree 索引原理   
explain  
  
分布式服务涉及到中间件和问题  
dubbo 内部会重试 注意写入接口的幂等性  
分布式锁： redis的实现，有一些问题，过期，释放时如何保证不会误操作到其他锁定记录，如何做到可重入  
单进程内：同步锁 cas原理  
事务 传播机制   
spring 结合使用 嵌套  
多线程使用的时候要使用线程池，原因  
  
spring aop 动态代理

# Mysql

## 事务

1.

## 索引

1. 定义：是一种高效获取数据的存储结构，例如hash，二叉，红黑数，b+tree
2. 根据索引值快速获取对应数据的行号，物理存储地址
3. 索引的缺点：需要额外的存储空间；在添加索引时会对表加锁。一般在晚上添加索引，添加字段时，比如order表时，要在晚上11点之后。
4. 聚簇索引，非聚簇索引
5. 联合索引一定要有最左边的https://www.jianshu.com/p/fd781d6e1158
6. B+ 所有的数据都存在叶子节点，并且没个叶子节点都有指向下一个叶子节点的指针，形成了一个有序的链表，方便范围查询。比如说select \* from Table where id > 1 and id < 100; 当找到1后，只需顺着节点和指针顺序遍历就可以一次性访问到所有数据节点，极大提到了区间查询效率。

一般情况下b+树的深度一般是2-4层,所以逻辑上找到某一个键值的行记录最多需要2-4层的io操作。

单一节点存储更多元素，减少io

所有查询都到叶子节点，查询比较稳定。

https://baijiahao.baidu.com/s?id=1628226562020721873&wfr=spider&for=pc

## 优化

的

# java

## Synchronize

底层原理时，线程通过获取的对象的监视器，这个过程时阻塞的，只能有一个线程获得，而其他线程只能获锁线程释放后，才能获取，在Javap -c class文件后，时monitorenter和monitorexit这两个命令控制

<https://www.jianshu.com/p/2ba154f275ea>

## Hashmap

Jdk7并发情况下扩容造成的链上的死循环 ，扩容时由于多个线程操作操作同一个一个链上的数据，在进行数据放到新数组时，指针引用造成了环，在get时，形成自旋。

<https://www.jianshu.com/p/1e9cf0ac07f4>

Aqs

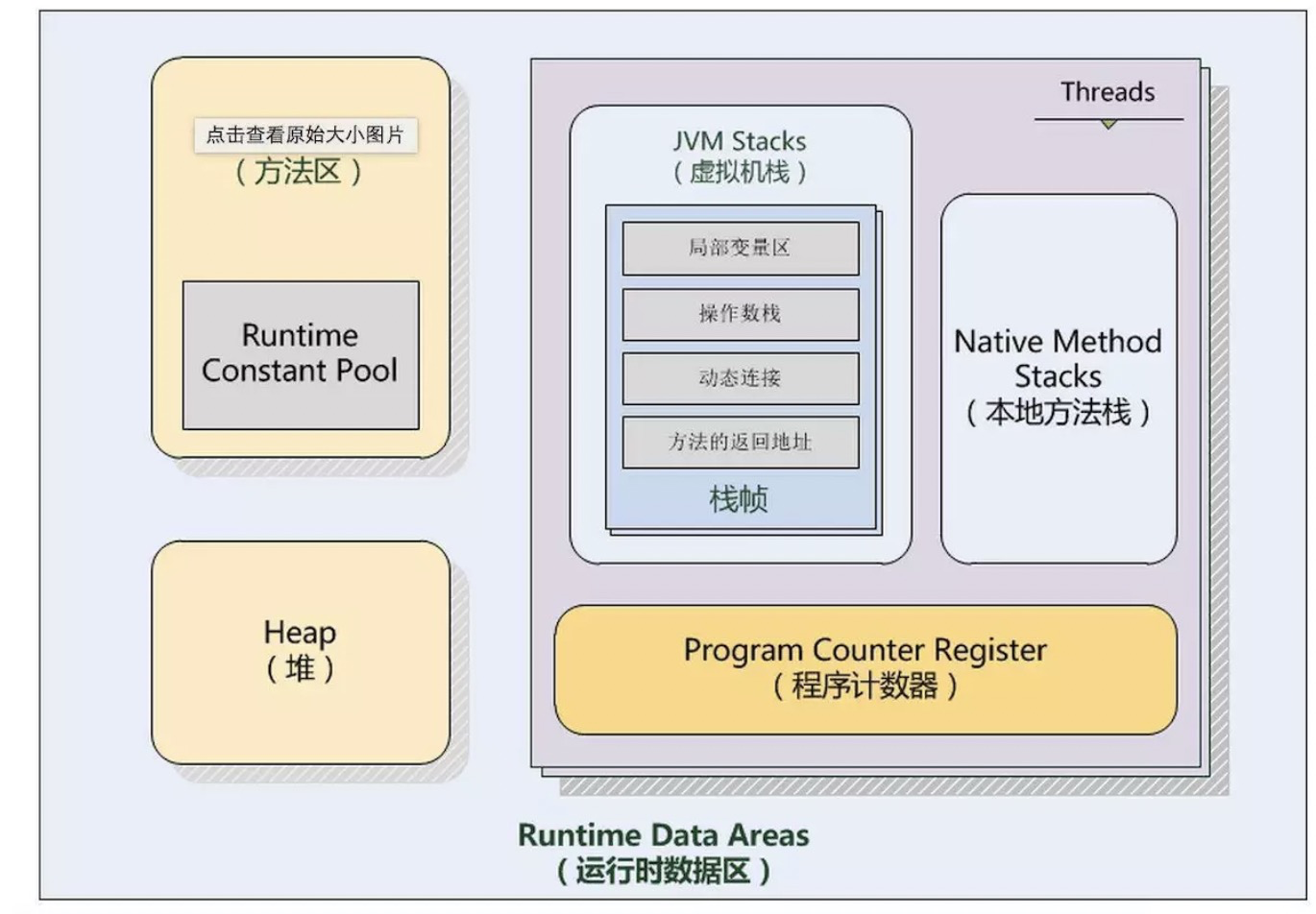
<https://www.jianshu.com/p/9d5cbadb6bc1>

<https://www.jianshu.com/p/e0fe0817b694>

# Jvm

## java内存模型

### 1.1运行时内存模型



程序计数器

线程私有，看作时当前线程执行时的字节码的行号执行器。java虚拟机是通过多线程轮流切换并非配处理器执行时间的方式实现的，一个处理器任意时刻都只有一条线程在执行，为了线程切换后恢复到切换前执行的字节码位置。

虚拟机栈

虚拟机栈描述的是Java方法执行的内存模型。每个方法在执行时会创建一个栈帧，用来存储局部变量表、操作数栈、动态链接、方法出口等信息。方法的调用和执行结束对应了一个栈帧在虚拟机栈中入栈和出栈的过程。

本地方法栈

本地方法栈为执行虚拟机使用到的native 方法服务，其他和虚拟机栈一样。Sun hotspot 虚拟机将本地方法栈和虚拟机合二为一。

堆

存放new出来的对象。线程共享

方法区

线程共享。存储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等数据。

存放Class信息。

运行时常量池

运行时常量池时方法区的一部分。String类的intern方法。在jdk1.6后常量池中存放的是第一次出现的字符串的引用，复制对象引用到常量池。

|  |
| --- |
| String s = "123";  new String("123").intern() //这两种方式都可以将字符串设置到常量池中。 |

### 1.2对象

#### 对象创建

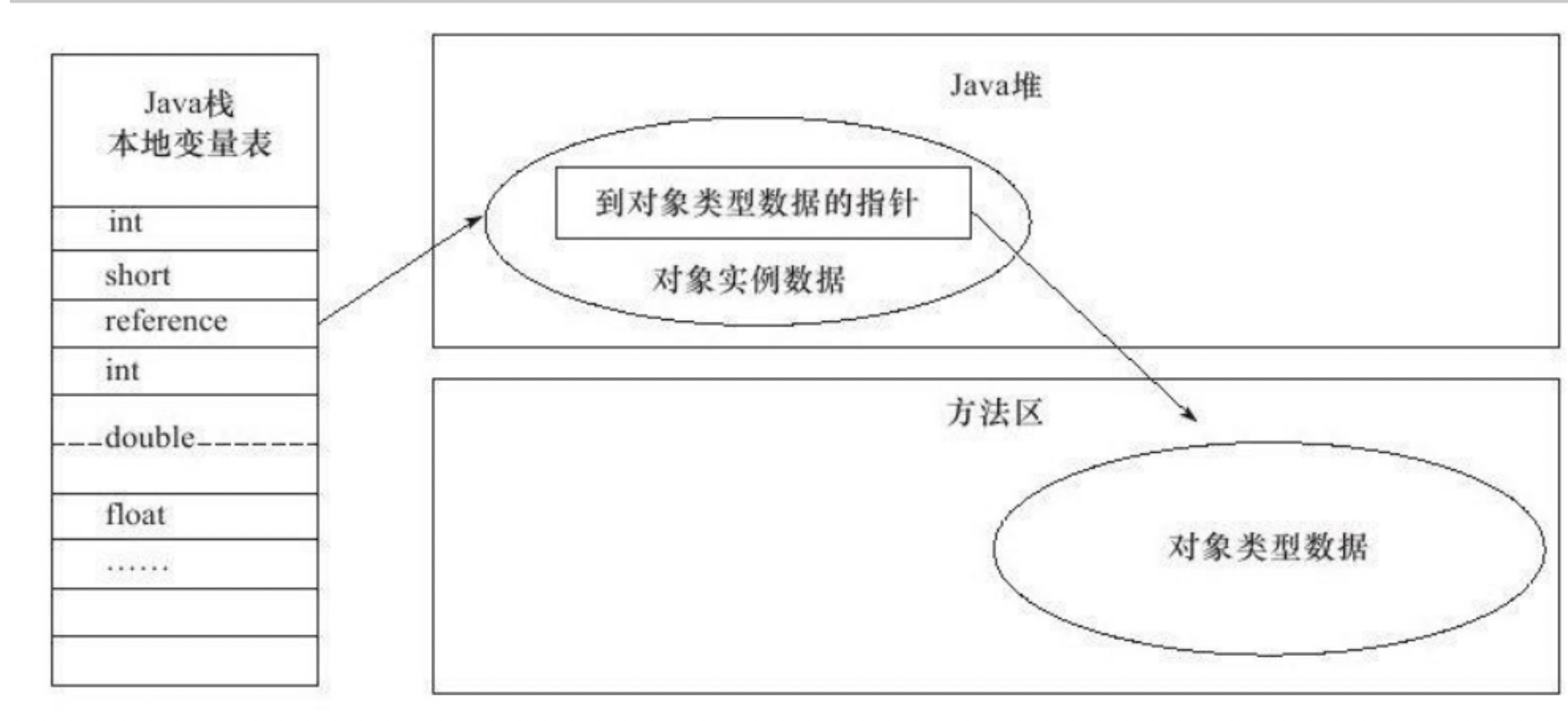
new关键自创建

对象的内存布局

对象{对象头、实例数据、对齐填充}

对象头{自身运行时数据【哈希码、gc分代年龄、锁状态标志、线程持有的锁、偏向线程ID、偏向时间戳】;指向class的指针}

#### 对象的访问定位



### 1.3内存溢出

堆内存溢出

一直创建对象

栈内存溢出

一个时超过栈深，持续压栈操作。方法无限递归调用。

创建过多线程，造成oom。

#### 方法区内存溢出

字符串常量一直生成，string.intern（）方法。

通过cglib无限生成代理类造成内存溢出。

## 垃圾收集

### 判断对象是否已死

**引用计数算法**

每当引用指向这个对象时，对象的引用器就加1，引用失效时对象的引用器就减1.当为0时表示可以被回收。

不能解决对象之间互相引用的问题

**可达性分析算法**

通过GCroot的方法判断。如果当前对象没有一个gcroot到当前对象的引用链，则认为当前对象可以被回收

**gcroot包括哪些：**

方法栈中引用的对象

方法区中静态变量引用的对象

方法区中常量引用的对象

### 引用类型

**强引用**

直接object o = new object new出来的对象并指向的对引用。

**软引用**

软引用描述的是有用但非必须的对象。软引用引用的对象只有在垃圾回收完成后，内存依然不够时才会对软引用引用的对象进行第二次垃圾回收。

**弱引用**

虚引用描述的是非必须对象。

被弱引用引用的对象，只要发生垃圾回收，就会对这部分对象进行回收。

**虚引用**

一个被虚引用引用的对象不会对对象的生存时间构成影响，也无法通过虚引用来取得一个对象实例。为一个对象设置虚引用关联的唯一目的就是能在这个对象被垃圾回收时收到一个系统通知。

### 方法区回收

主要回收无用的类和废弃的常量

无用的类必须满足：1.这个类所有的实例都被回收2.该类的classloader已经被回收3.该类对应的java.lang.Class对象没有在任何地方被引用。

废弃的常量：例如一个“abc”的字符串被推进了常量池，只有在当前jvm进程中没有任何“abc”时才会被回收。

### 垃圾收集算法

**标记清除**

**标记整理**

在对象被标记后，进行清理的过程中让存活的对象向一端移动，垃圾回收完成后会形成连续的内存空间

**复制算法**

将内存分成大小相等的两块，每次只使用其中的一块，在垃圾回收时将存活的对象复制到另一块内存上，然后一次性清理那个需要回收的内存区域。不会产生空间碎片。

**分代收集**

新生代采用复制算法，老年代采用标记清除或者标记整理

### 垃圾算法实现

### 垃圾收集器

**serial收集器**

新生代收集器，采用复制算法。

单线程完成垃圾收集工作，在垃圾收集时，必须暂停其他所有的工作线程。

**Serial old收集器**

老年代收集器，采用标记整理的算法。

Serial old是serial收集器的老年代版本。也是采用单线程收集，在收集过程中必须暂停所有的用户线程。

**parNew 收集器**

新生代收集器，采用复制算法。

serial的多线程版本，可以多个线程完成垃圾收集工作。

在垃圾收集期间，暂停所有的用户线程。

在单cpu情况下由于线程切换，性能不如serial收集器。

默认开启的垃圾收集线程等于cpu的数量，可以使用-XX:ParallerGCThreads 参数指定

**Parallel Scavenge 收集器**

新生代收集器，采用复制算法

多个线程完成收集工作和parnew 类似

Parallel scavenger收集器目标是达到一个可控的吞吐量。

吞吐量=运行用户代码时间/运行用户代码时间+垃圾收集时间

可以用-xx:MaxGCPauseMillis设置最大垃圾收集停顿时间；用-XX:GCTimeRatio设置吞吐量大小。

gc停顿时间的缩短需要牺牲吞吐量和新生代空间换取。

可以设置-XX:+userAdaptiveSizePolicy这个开关参数，开启自适应调节策略。不需要指定新生代的大小（-xmn）、eden与survivor区的比例、晋升老年代的对象大小等细节参数。

**Parallel old收集器**

老年代收集器，采用标记整理算法。

Parallel old是parallel scavenge的老年代版本。

吞吐量优先、多个线程完成垃圾收集工作

**cms收集器**

老年代收集器，采用标记-清除算法。是一种获取最短停顿时间为目标的收集器。

初始标记-并发标记-重新标记-并发清除

初始标记和重新标记会暂停其他工作线程。

初始标记只是标记一下gc root能直接关联到的对象，并发标记似乎gcroot tracing的过程。重新标记是为了标记在并发标记阶段程序继续运行产生变动的一部分记录，并发清除阶段是和用户线程一起执行的，这个时间段最长，降低了用户线程的停顿时间。

缺点：

**产生浮动垃圾，**在并发清除阶段，用户线程还在执行，这期间产生的垃圾。由于在并发阶段用户线程还在执行，老年代就需要额外的存储空间，如果存储空间不够，就会出现“concurrent mode failure”失败，这时会启用serial old收集器。

**产生空间碎片。**由于采用了标记-清除的算法，所以在并发清除阶段会产生空间碎片。cms收集器为了解决这个问题，在要进行full gc时会开启空间碎片整理过程，这个过程也是需要暂停其他工作线程的。通过-xx:CmsFullGcsBeforeCompaction指定几次不压缩的full gc后，跟着来一次带压缩的full gc 默认是0，每次full gc也就是并发清除完成后都要碎片整理。

**G1收集器**

对整个堆进行收集，综合采用标记整理，局部两个region之间采用复制算法。

G1收集器将整个堆划分为多个大小相等的独立区域region，仍保留新生代老年代的感念，但都是region的一部分，不需要联系。

可预测停顿，并行与并发，分代收集，空间整合

为了避免全堆扫描，将每个region关联一个remembered set。每次有reference类型的数据写操作时，检查是否其他的region有引用整个这个对象，如果有将引用信息记录到被引用对象的remembered set 里面。

收集过程也是分为4部分：

初始标记-》标记gcroot 能直接关联到的对象，需要停顿其他工作线程，时间短

并发标记-》对堆中的对象进行可达性分析，找出存活的对象，和工作线程并发执行，时间长。

最终标记-》记录在并发标记期间产生变动的记录，将这段时间对象变动记录在线程的remembered set log 里面，并合并到remembered set里面。这段期间需要暂停其他工作线程，但是收集线程并行执行。

筛选回收-》将各个region的回收价值和成本进行排序，根据用户所期望的gc停顿时间来制定回收计划。这段时间也是需要暂停其他用户线程。

### 内存分配和回收策略

对象在分配时，在eden区分配，当eden区内存不足时，触发minor gc，将eden区和一块survivor中还存活的对象复制到另一块survivor内存中。

如果对象在分配时，遇到**大对象**时（数组，字符串）超过-XX:PretenureSizeThreshold这个设置的值时，直接分配到老年代。

**长期存活的对象**，每次gc后这个对象的age就加1，超过-XX:MaxTenuringThreshold的设置的值就放到老年代。

**空间分配担保**：新生代中有大量的对象存活，survivor空间不够，当出现大量对象在MinorGC后仍然存活的情况（最极端的情况就是内存回收后新生代中所有对象都存活），就需要老年代进行分配担保，把Survivor无法容纳的对象直接进入老年代.只要老年代的连续空间大于新生代对象的总大小或者历次晋升的平均大小，就进行Minor GC，否则FullGC

在每次minor gc之前，虚拟机会先检查老年代最大可用的连续空间时否大于新生代所有对象的总空间，如果大于就说明时担保成功，可以确定minor gc 是成功的，如果小于就要先进性full gc，清除老年代死去的对象。

# Dubbo

<https://www.cnblogs.com/h-c-g/p/11209756.html>

# 题

<https://blog.csdn.net/weixin_43495390/article/details/86533482>

Fifo lru lfu

<https://blog.csdn.net/notOnlyRush/article/details/80158703>

# Spring

