# 自我介绍

面试官好，我是从2018年开始参加工作的，至今差不多有3年时间，任职过一家公司，项目主要是一个企业全渠道管理的项目，主要是做经销商订货业务，和一些围绕着订货的相关业务，如促销返利库存等，用到的框架主要有 Spring，SpringMVC，数据库方面使用的是sqlserver，也了解一些oracle的，缓存方面用的是ehcache本地缓存，redis也有一点了解。有过一些并发编程的经验，MQ也有一点了解。。

Spring

SpringMVC的流程？

Spring MVC 提供了一种分离式的方法来开发 Web 应用，通过把模型-视图-控制器分离，将web层进行职责解耦。通过运用像 DispatcherServelet，MoudlAndView 和 ViewResolver 等一些简单的概念，开发 Web 应用将会变的非常简单。

（1）用户发送请求至前端控制器DispatcherServlet；

（2）DispatcherServlet收到请求后，调用HandlerMapping处理器映射器，请求获取Handler；

（3）处理器映射器根据请求url找到具体的处理器Handler，生成处理器对象及处理器拦截器(如果有则生成)，一并返回给DispatcherServlet；

（4）DispatcherServlet 调用 HandlerAdapter处理器适配器，请求执行Handler；

（5）HandlerAdapter 经过适配调用 具体处理器进行处理业务逻辑；

（6）Handler执行完成返回ModelAndView；

（7）HandlerAdapter将Handler执行结果ModelAndView返回给DispatcherServlet；

（8）DispatcherServlet将ModelAndView传给ViewResolver视图解析器进行解析；

（9）ViewResolver解析后返回具体View；

（10）DispatcherServlet对View进行渲染视图（即将模型数据填充至视图中）

（11）DispatcherServlet响应用户。

Springmvc 异常处理，自定义实现spring的全局异常解析器HandlerExceptionResolver，在异常处理器中添视图页面即可。

SpringBoot，主要特点是“默认大于配置”，不同于Spring和SpringMVC需要大量的配置，SpringBoot提供了默认的配置，通过一个自动配置和启动的项目解决这个问题。为了更快的构建产品就绪应用程序，Spring Boot 提供了一些非功能性特征。

优先加载 ；bootstrap.yml 其次 application.yml

Spring是一个开放源代码的设计层面框架，他解决的是业务逻辑层和其他各层的松耦合问题，因此它将面向接口的编程思想贯穿整个系统应用。可以接管web层，业务层，dao层，持久层的组件，并且可以配置各种bean,和维护bean与bean之间的关系。

IOC(控制反转，依赖注入)

在Java开发中，Ioc意味着将你设计好的对象交给容器控制，而不是传统的在你的对象内部直接控制。

IOC不是一种技术，只是一种思想，一个重要的面向对象编程的法则，它能指导我们如何设计出松耦合、更优良的程序。传统应用程序都是由我们在类内部主动创建依赖对象，从而导致类与类之间高耦合，难于测试；有了IOC容器后，把创建和查找依赖对象的控制权交给了容器，由容器进行注入组合对象，所以对象与对象之间是松散耦合，这样也方便测试，利于功能复用，更重要的是使得程序的整个体系结构变得非常灵活。

控制反转IOC(Inversion of Control)是说创建对象的控制权进行转移，以前创建对象的主动权和创建时机是由自己把控的，而现在这种权力转移到第三方，比如转移交给了IOC容器，它就是一个专门用来创建对象的工厂，你要什么对象，它就给你什么对象，有了 IOC容器，依赖关系就变了，原先的依赖关系就没了，它们都依赖IOC容器了，通过IOC容器来建立它们之间的关系。  
　　DI(依赖注入)其实就是IOC的另外一种说法

Aop：是一种设计思想，称为面向切面编程，利用横切技术剖析对象内部，将业务之间共同调用的逻辑提取并封装为一个可复用的模块，这个模块被命名为切面（Aspect），该模块减少系统中的重复代码，降低模块间的耦合度，可用于日志、权限认证、事务管理

1. ConcurretHashMap

ConcurrentHashMap可以做到读取数据不加锁，并且其内部的结构可以让其在进行写操作的时候能够将锁的粒度保持地尽量地小，允许多个修改操作并发进行，其关键在于使用了锁分段技术。它使用了多个锁来控制对hash表的不同部分进行的修改。对于JDK1.7版本的实现, ConcurrentHashMap内部使用段(Segment)来表示这些不同的部分，每个段其实就是一个小的Hashtable，它们有自己的锁。只要多个修改操作发生在不同的段上，它们就可以并发进行。JDK1.8的实现降低锁的粒度，JDK1.7版本锁的粒度是基于Segment的，包含多个HashEntry，而JDK1.8锁的粒度就是HashEntry（首节点）。

JAVA7之前ConcurrentHashMap主要采用锁机制，在对某个Segment进行操作时，将该Segment锁定，不允许对其进行非查询操作，而在JAVA8之后采用CAS无锁算法，这种乐观操作在完成前进行判断，如果符合预期结果才给予执行，对并发操作提供良好的优化.

1. CAS

是所有原子类的底层原理，乐观锁主要采用CAS算法。

CAS，比较并交换，是JDK提供的非阻塞原子性操作，通过硬件保证比较-更新操作的原子性。 通常结合volatile保证共享变量的原子性。

思想：获取当前变量最新值A（预期值），然后进行CAS操作。此时如果内存中变量的值V（内存值V）等于预期值A，说明没有被其他线程修改过，我就把变量值改成B（更新值）；如果不是A，便不再修改。

CAS操作利用CPU的特殊指令，由CPU保证原子性，完成一系列操作，不存在安全性问题。

CAS的变量需要用volatile修饰,以便在各线程之间保证可见。

CAS算法思想的使用场景

乐观锁

并发容器，例如ConcurrentHashMap

原子类 AtomicLong

缺点：

CAS是比较值，如果值相等则变换。此处可能存在这样的情况，线程1获取变量值为5，线程2将值改为10，线程3再将值改回5。对于线程1，变量的值没有变，但对于计数等后续操作是不正确的。

分析：ABA问题的产生是因为变量的状态值产生了环形变换。如果变量的值只能朝一个方向转化，便不会构成环形，不存在ABA问题。

解决方法：可以参考数据库乐观锁的处理，加版本号，变量更新时版本号会改变。通过比较版本号代替比较变量值。与集合的Fast-Fail机制类似，检查modCount值是否一致。

以AtomicLong为例，高并发场景下，如果线程一直无法进行CAS操作，内部是dowhile死循环，会一直自旋，消耗CPU

1. HashMap

HashMap : 数组+链表， 拉链发解决hash冲突

map.put(k,v)实现原理

(先扩容，判断是否需要扩容，实际容量 >容量\*加载因子时扩容， 容量为0时用默认初始容量16扩容)

（1）首先将k,v封装到Node对象当中（节点）。

（2）然后它的底层会调用K的hashCode()方法得出hash值。

（3）通过哈希表函数/哈希算法，将hash值转换成数组的下标，下标位置上如果没有任何元素，就把Node添加到这个位置上。如果说下标对应的位置上有链表。此时，就会拿着k和链表上每个节点的k进行equal。如果所有的equals方法返回都是false，那么这个新的节点将被添加到链表的末尾。如其中有一个equals返回了true，那么这个节点的value将会被覆盖。

如果是链表节点，且链表大于8时进行树化，树化时如果判断容量不超过64只会扩容，不会树化（红黑树）

# 红黑树

红黑树：（也是一种二分查找树，即左节点小于父节点，右节点大于父节点，高效增删改查，时间复杂度 O（logn），树的深度 = 查找次数）

1. 根节点为黑节点，2叶子节点为黑节点 null节点，3 红节点的下级节点必为黑节点4.根节点到叶子节点的最长路径不超过最短路径2倍(所有路径黑节点数目相同)5.节点非黑即红

优势：红黑树不用维持平衡二叉树的节点深度差不大于1的条件， 可以减少树翻转的次数

在做插入删除的时候比avl树要做的调整要少

1. Gc原理

MinoGC 和fullgc

java将堆内存分为3大部分：新生代、老年代和永久代，其中新生代又进一步划分为Eden、S0、S1(Survivor)三个区。结构如下图所示：

回收算法：对新生代，主要采用**复制算法**，而针对老年代，通常采用**标记-清除算法**或者**标记-整理算法**来进行回收。

**复制算法： 每次回收 将eden区和一个survivor区中存活的对象复制到另一个survivor区，然后清除eden区和一个survivor区的内存**

**标记-整理算法：让所有存活对象都向一端移动，然后直接清理掉边界以外的内存，标记-整理通常会在标记-清除算法里面作为备选方案，为了防止标记-清除后产生大量内存碎片而无法为大对象分配足够内存的情况。**

**引用计数法、可达性分析法**

1. B树

B树和b+树

B树每个节点都存储数据，所有节点组成这棵树。B+树只有叶子节点存储数据（B+数中有两个头指针：一个指向根节点，另一个指向关键字最小的叶节点），叶子节点包含了这棵树的所有数据，所有的叶子结点使用链表相连，便于区间查找和遍历，所有非叶节点起到索引作用。

B树中叶节点包含的关键字和其他节点包含的关键字是不重复的，B+树的索引项只包含对应子树的最大关键字和指向该子树的指针，不含有该关键字对应记录的存储地址。

B树优点 ：B树的每一个节点都包含key和value，因此经常访问的元素可能离根节点更近，因此访问也更迅速。

B+树优点：所有的叶子结点使用链表相连，便于区间查找和遍历。B树则需要进行每一层的递归遍历。相邻的元素可能在内存中不相邻，所以缓存命中性没有B+树好。

b+树的中间节点不保存数据，能容纳更多节点元素。

共同优点：考虑磁盘IO的影响，它相对于内存来说是很慢的。数据库索引是存储在磁盘上的，当数据量大时，就不能把整个索引全部加载到内存了，只能逐一加载每一个磁盘页（对应索引树的节点）。所以我们要减少IO次数，对于树来说，IO次数就是树的高度，而“矮胖”就是b树的特征之一，m的大小取决于磁盘页的大小。

1. Nginx

Nginx ，是一个 Web 服务器和反向代理服务器，用于 HTTP、HTTPS、SMTP、POP3 和 IMAP 协议。

目前使用的最多的 Web 服务器或者代理服务器，像淘宝、新浪、网易、迅雷等都在使用。

Nginx 的主要功能如下：

作为 http server (代替 Apache ，对 PHP 需要 FastCGI 处理器支持)

FastCGI：Nginx 本身不支持 PHP 等语言，但是它可以通过 FastCGI 来将请求扔给某些语言或框架处理。

反向代理服务器 ：反向代理隐藏了真实的服务端，反向代理服务器会帮我们把请求转发到真实的服务器那里去。Nginx就是性能非常好的反向代理服务器，用来做负载均衡。

实现负载均衡：

虚拟主机

1. 线程

多线程是为了同步完成多项任务，不是为了提高运行效率，而是为了提高资源使用效率来提高系统的效率。线程是在同一时间需要完成多项任务的时候实现的。

1. 集成Thread类，run()
2. 实现runable类，start()
3. 实现callable类，start()
4. 线程池创建

线程状态

（1）新建（New）：创建后尚未启动的线程处于这种状态

（2）运行（Runable）：Runable包括了操作系统线程状态的Running和Ready，也就是处于此状态的线程有可能正在执行，也有可能正在等待着CPU为它分配执行时间。

（3）等待（Wating）：处于这种状态的线程不会被分配CPU执行时间。等待状态又分为无限期等待和有限期等待，处于无限期等待的线程需要被其他线程显示地唤醒，没有设置Timeout参数的Object.wait()、没有设置Timeout参数的Thread.join()方法都会使线程进入无限期等待状态；有限期等待状态无须等待被其他线程显示地唤醒，在一定时间之后它们会由系统自动唤醒，Thread.sleep()、设置了Timeout参数的Object.wait()、设置了Timeout参数的Thread.join()方法都会使线程进入有限期等待状态。

（4）阻塞（Blocked）：线程被阻塞了，“阻塞状态”与”等待状态“的区别是：”阻塞状态“在等待着获取到一个排他锁，这个时间将在另外一个线程放弃这个锁的时候发生；而”等待状态“则是在等待一段时间或者唤醒动作的发生。在程序等待进入同步区域的时候，线程将进入这种状态。

（5）结束（Terminated）：已终止线程的线程状态，线程已经结束执行。

# sql优化原子性（atomicity，或称不可分割性）、一致性（consistency）、隔离性（isolation，又称独立性）、持久性（durability）。

* + - 1. 对查询进行优化，应尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引
      2. 应尽量避免在 where 子句中使用!=或<>操作符，否则将引擎放弃使用索引而进行全表扫描。
      3. 尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描
      4. 应尽量避免在 where 子句中使用 or 来连接条件
      5. 模糊查询，考虑用全文本检索
      6. 应尽量避免在 where 子句中对字段进行表达式操作/函数操作
      7. 不要写一些没有意义的查询