**一面（60分钟）**

1. Redis 里的有序列表底层实现
2. Redis 集群、哨兵机制
3. 消息中间件的选型问题，为什么要用 MQ
4. Linux 中能否删除一个正在运行的文件
5. Linux 常用的命令，如何排查 bug
6. JVM 垃圾回收如何解决对象循环引用的问题，垃圾回收算法
7. TCP 的四次挥手的详细过程，为什么会有 CLOSE\_WAIT 和 TIME\_WAIT
8. 输入一个 url 并回车，这个过程中涉及到哪些网络协议，具体描述
9. MySQL 索引一般会在什么情况下创建，讲讲存储引擎
10. 说说 B 树和 B+ 树的区别，为什么 InnoDB 用的 B+ Tree 而不是 B Tree
11. synchronized 和 Lock 的区别
12. 进程如何同步，页面置换算法
13. **三个编程题：题目记不起来了，涉及桶排序、层次遍历、动态规划**

**二面（60分钟）**

1. **实现一个阻塞队列（生产者消费者模型），编程**
2. wait/notify 机制，为啥要放在同步代码块中，判断条件为啥用 while 循环
3. CMS 垃圾回收算法的流程，标记清除算法
4. ConcurrentHashMap 的内部实现，JDK 1.7 与 1.8 有什么不同
5. **找出10000个数据中第 k 大的数，描述具体过程并编程（bug free）**
6. 聚簇索引和非聚簇索引，为什么要用 B + Tree
7. 一条 SQL 语句执行的流程，尽量说详细
8. AOP 的实现原理，动态代理和静态代理的区别
9. JVM 内存模型的结构
10. **输入一个字符串，包含数字、加减乘除和括号，输出结果，编程（调试并运行成功）**

**三面（50分钟）**

1. 登录注册系统怎么做，应该注意什么
2. 登录怎样校验密码，海量用户同时登录怎么优化
3. Cookie 和 Session 的区别，怎样存海量 Session
4. 分布式 Session 问题
5. 显示网站的用户在线数的解决思路
6. 一致性哈希
7. 怎样解决 Redis 和数据库一致性问题
8. Redis 集群、主从架构
9. 分布式事务
10. **给定一个数字 x，要求使用 k 个数字求和可以得到 x，数字从1-9中选择，不能重复。  
    例如 k = 3, x = 9时输出：[[1,2,6], [1,3,5], [2,3,4]]（刚开始结果重复，加了个 set 调试了两遍才写出来）**

**百度面经**

**一面（50分钟）**

1. 在线程 A 对 HashMap 进行迭代查询的时候，线程 B 对 HashMap 进行增删改会发生什么
2. ConcurrentHashMap 的数据结构与 HashMap 有什么区别与联系
3. Java 内存模型中哪一个区域不会发生 OOM 异常
4. 类加载流程，越详细越好
5. 为什么要有双亲委派，类加载器有哪些
6. MVCC 是什么，简述 MySQL 的隔离机制
7. MySQL 的锁机制
8. Java API 中 NIO 和 BIO 的区别
9. 消息中间件的好处，如何选型
10. Redis 的 IO 模型；常用的数据结构；zset 的内部实现，为什么用跳表不用红黑树，跳表的空间复杂度
11. AOP 的抽象属性之间的关系与联系
12. 有什么要问的问题（问了面试官对我的评价：主观能动性较强，基础很好，表达能力强，中等偏上，但离 top student 还有一段距离，对某些地方理解的深度不够）

**二面（40分钟）**

1. 介绍项目，秒杀平台如果想支持更大的并发量应该如何优化？如果把项目部署成多个服务，架构方面应该怎么做
2. 数据库的 ACID，MySQL 底层是如何保证原子性的
3. 设计一个实时监测 MySQL 的工具，可以监测每条 SQL 语句运行时间、事务的运行和数据库的连接信息等
4. MySQL 的 binlog、redo log 和 undo log
5. HTTP 和 HTTPS 的区别
6. Redis 的主从架构、哨兵机制
7. Redis 的持久化，RDB 和 AOF 的对比
8. MQ 如何保证消息传输的可靠性
9. 问了学校的研究方向、本科学习情况及获奖情况

**三面（40分钟）**

* 项目中的细节问题
* 对 Spring 的理解
* AOP 动态代理的实现
* 对并发的了解
* 场景设计题：实时更新街道的交通拥挤状况
* SQL 常见的优化手段
* 问了平时的学习方法以及三观问题
* 两道算法
* leetcode 56 merge intervals
* leetcode 173 binary search tree iterator
* 项目
* 设计模式
* 排序算法
* hashmap和concurrenthashmap
* 线程池
* CAP原理
* UDP与TCP
* Cookie与Session

一、JVM 整体组成  
JVM 整体组成可分为以下四个部分：  
类加载器（ClassLoader）  
运行时数据区（Runtime Data Area）  
执行引擎（Execution Engine）  
本地库接口（Native Interface）  
各个组成部分的用途：  
程序在执行之前先要把java代码转换成字节码（class文件），jvm首先需要把字节码通过一定的方式 类加载器（ClassLoader） 把文件加载到内存中 运行时数据区（Runtime Data Area） ，而字节码文件是jvm的一套指令集规范，并不能直接交个底层操作系统去执行，因此需要特定的命令解析器 执行引擎（Execution Engine） 将字节码翻译成底层系统指令再交由CPU去执行，而这个过程中需要调用其他语言的接口 本地库接口（Native Interface） 来实现整个程序的功能，这就是这4个主要组成部分的职责与功能

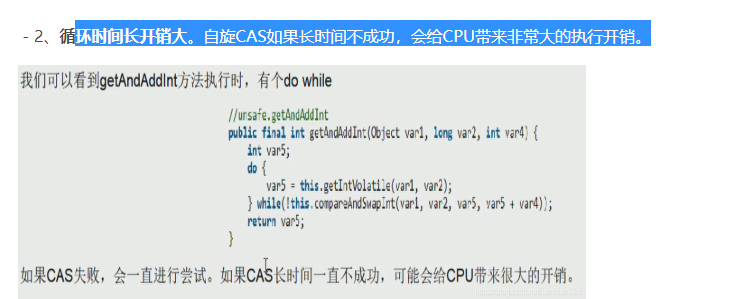
CAS的全称为Compare-And-Swap ,它是一条CPU并发原语.

CAS实现原理：包含三个值当前内存值(V)、预期原来的值(A)以及期待更新的值(B)。如果内存位置V的值与预期原值A相匹配，那么处理器会自动将该位置值更新为新值B,返回true。否则处理器不做任何操作，返回false

底层实现：JVM会帮我们实现出CAS汇编指令。这是一种完全依赖于硬件的功能，通过它实现了原子操作。由于CAS是一种系统原语，原语属于操作系统用语范畴，是由若干条指令组成的，用于完成某个功能的一个过程，并且原语的执行必须是连续的，在执行过程中不允许被中断，也就是说CAS是一条CPU的原子指令，不会造成所谓的数据不一致性问题。

三、CAS存在的问题 -- 虽然很高效的解决原子操作，但是CAS仍然存在三大问题。

1、ABA问题：当你获得对象当前数据后，在准备修改为新值前，对象的值被其他线程连续修改了两次，而经过两次修改后，对象的值又恢复为旧值，这样当前线程无法正确判断这个对象是否修改过



3、只能保证一个共享变量的原子操作。当对一个共享变量执行操作时，我们可以使用循环CAS的方式来保证原子操作，但是对多个共享变量操作时，循环CAS就无法保证操作的原子性，这个时候就可以用锁来保证原子性。

Java 类加载流程：

1.加载。 加载是指java虚拟机查找字节码文件即class文件。并且根据字节流创建java.lang.Class对象。这个过程，将类的.class文件中的二进制数据写入内存，放在运行时区域的方法区内。然后在堆中创建java.lang.class对象。用来封装类在方法区的数据结构。

* 1. java虚拟机将.class文件读入内存，放在运行时区域的方法区内，并为之堆上创建一个Class对象。
  2. 任何类被使用时系统都会为其创建一个且仅有一个Class对象。这个class对象描述这个类创建出来的对象的所有信息。比如有哪些构造方法，成员方法。都有哪些成员变量等。

2.链接

链接包括 验证，准备以及解析三个阶段。

* 1. 验证阶段：主要目的是确保被加载的类（.class文件的字节流）满足java虚拟机规范。比如会验证魔号
  2. 准备阶段：负责为类的静态成员分配内存。并设置默认初始值。
  3. 解析阶段：将类的二进制数据中的符号引用替换为直接引用。

3.初始化

初始化。则是为标记为常量值得字段赋值的过程。换句话说，只对static修饰的变量或语句块进行初始化。如果初始化一个类的时候，其父类尚未初始化，则优先初始化其父类。如果同时包含多个静态变量和静态代码块，则按照自上而下的顺序依次执行。

# 一、计算机网络

**基础部分**

1. TCP报头格式

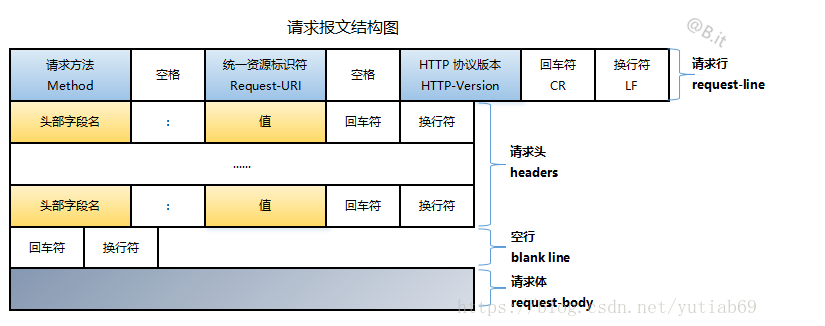
<https://www.cnblogs.com/shineyoung/p/10656914.html>

1. UDP报头格式

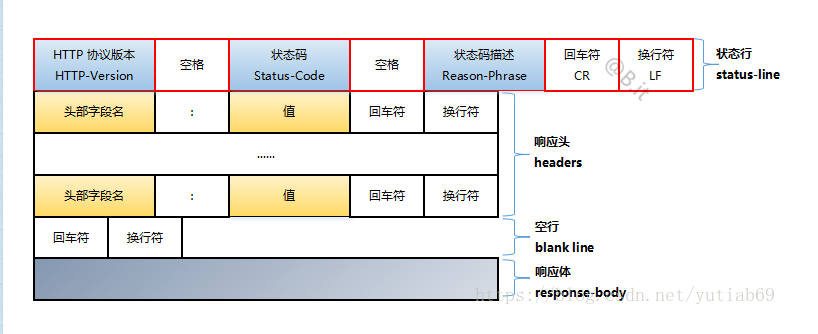
<https://www.cnblogs.com/rohens-hbg/articles/4680369.html>

1. TCP/UDP区别（不仅是宏观上的，最好能根据各自的机制讲解清楚）
2. HTTP状态码（最好结合使用场景，比如在缓存命中时使用哪个）
3. HTTP协议（一些报头字段的作用，如cace-control、keep-alive）

请求报文：



响应报文：

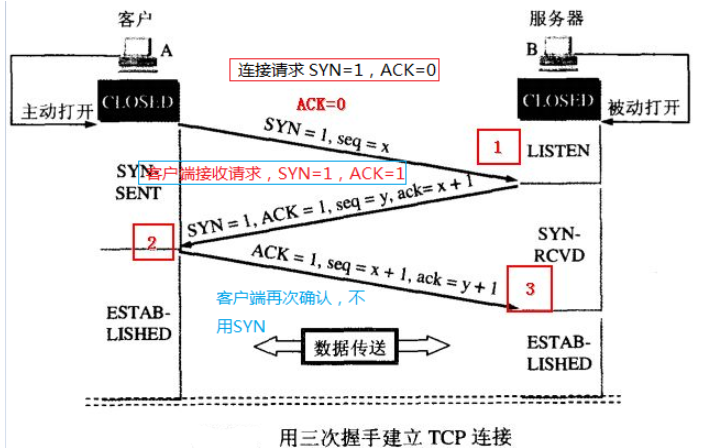


1. OSI协议、TCP/IP协议以及每层对应的协议。
2. SESSION机制、cookie机制,Token机制

* Cookie的作用就是为了解决http协议的无状态缺陷。Cookie机制采用客户端保持状态的方案。而sessioin机制采用的是服务器端保持状态的方案。同于由于采用服务器端保持状态的方案在客户端来需要保存一个标识。所以session机制可能需要借助于cookie机制来达到保存标识的目的，当然也有别的选择。
* 基于Token的身份验证是无状态的，我们不讲用户信息存在服务器或Session中。对数据进行了加密，签名算法.基于Token的身份验证过程
  + - 用户通过用户名和密码发送请求
    - 程序验证
    - 程序返回一个签名的token给客户端
    - 客户端储存token，并且每次用于请求
    - 服务端验证token并返回数据。

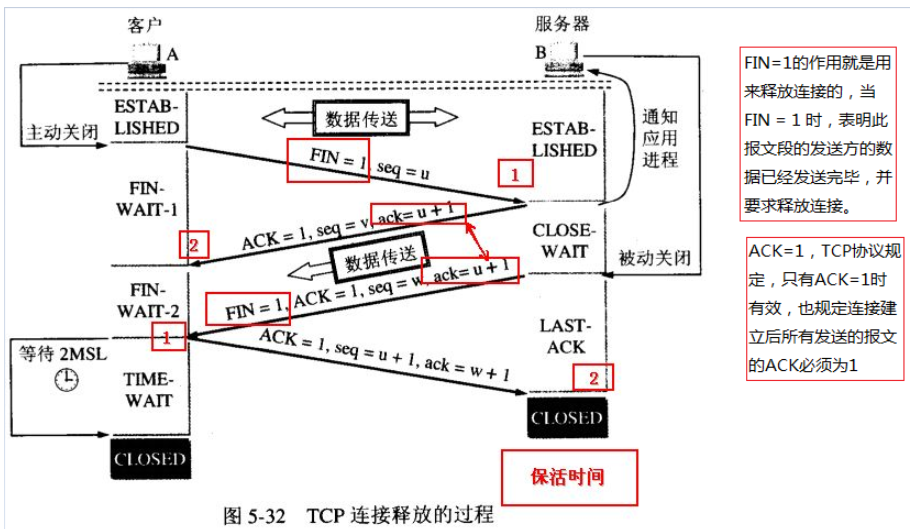
1. TCP三次握手、四次挥手（这个问题真的要回答吐了，不过真的是面试官最喜欢问的，建议每天手撸一遍，而且不只是每次请求的过程，各种FIN\_WAIT、TIME\_WAIT状态也要掌握）。

Tcp三次握手

* 首先由client发出请求连接，即SYN=1,ACK=0,seq=x;Tcp规定SYN=1时不能携带数据，但要消耗一个序号，因此声名自己的序号是seq=x；
* Server收到请假连接后进行回复确认，即SYN=1,ACK=1，seq=y；ack=x+1有两层意思，一是确认收到x序号，二是请求的下一个序号为x+1；
* 再然后client再进行一次，但不用SYN了，这时即ACK=1;seq=x+1;ack=y+1;
* 

TCP四次挥手：

* 当客户A没有东西要发送时就要释放A这边的连接，A会发送一个报文（没有数据）,其中FIN设置为1。FIN=1.seq=u; 随后进入FIN\_WAIT\_1状态；
* 服务器B收到后给应用程序一个信，即ACK=1, seq=v; ack=u+1;这时A那边的连接已经关闭。即A不再发送信息；但仍可接受信息；服务器B 进入close\_wait状态
* A收到B 的确认后进入FIN\_WAIT\_2状态；等待B请求释放连接。
* B发送完成后就向A请求连接释放，也是用FIN=1表示，并且用ack=u+1，随后B 进入LAST\_ACK状态；
* A收到后回复一个确认信息。并进入TIME\_WAIT状态，等待2MSL时间；



1. 打开网页到页面显示之间的过程（涵盖了各个方面，DNS解析过程，Nginx请求转发、连接建立和保持过程、浏览器内容渲染过程，考虑的越详细越好）。
2. http和https区别，https在请求时额外的过程，https是如何保证数据安全的
3. IP地址子网划分
4. POST和GET区别
5. DNS解析过程

**深入部分**   
13. TCP如何保证数据的可靠传输的（这个问题可以引申出很多子问题，拥塞控制慢开始、拥塞避免、快重传、滑动窗口协议、停止等待协议、超时重传机制，最好都能掌握）   
14. 地址解析协议ARP   
15. 交换机和路由器的区别

# 二、数据库

**基础部分**

1. 事务四大特性（ACID）

* 原子性：Atomicity 事务是一个原子操作。由一系列动作组成。事务的原子性确保动作要么全部完成，要么全部不完成。
* 一致性：consistency 事务在完成时，必须是所有的数据都保存一致状态。事务的一致性定义基本可以理解为是对数据完整性约束的遵循。这么约束可能包括主键约束，外键约束。然后也包括应用层次的一致性，比如银行的转账必须保证转账前后总额不变。
* 隔离性：isolation 并发事务的执行之间无影响。
* 持久性：durability 一旦事务完成，数据库的改变必须是持久化的。

1. 数据库隔离级别，每个级别会引发什么问题，mysql默认是哪个级别

* Read\_uncommitted: 这是事务最低的隔离级别，它允许一个事务可以看到这个事务未提交的数据。引发问题：脏读，读取了未提交的数据。
* Read\_committed : 保证一个事务修改的数据提交后才能被另外一个事务读取。引发问题：不可重复读。意味着我们在同一个事务中执行完全相同的select语句时可能看到一样的结果。（其他事务执行update和delete语句）
* Repeatable\_read: 这种事务隔离级别可以防止脏读，不可重复读。但是可能会出现幻读。引发问题：幻读。当用户读取某一范围的数据行时，另一个事务又在改范围内插入了新行。当用户再读取该范围的数据行时，会发现有新的幻影行。）（其他事务执行的Insert语句）。Mysql的默认隔离级别。
* Serializable: 这个是级别的事务，被要求顺序执行不要产生任何问题。这是最高的隔离级别。它是通过在每个读的数据行上加上共享锁。可能带来大量的损竞争和超时现象。

1. MYSQL的两种存储引擎区别（事务、锁级别等等），各自的适用场景

MyISAM:

* 1. 不支持事务，但是每次查询都是原子的；
  2. 支持表级锁，即每次操作都是对整个表加锁。
  3. 存储表的总行数;
  4. 一个MYISAM的表有三个文件：索引文件，表结构文件；
  5. 采用非聚集索引。索引文件的数据域存储指向数据文件的指针。辅索引与主索引基本一致；但是辅索引不用保证唯一性。

适用场景:MyISAM管理非事务表。它提供高速存储和检索以及全文搜素能力。如果应用中需要执行大量的select查询。那么MyISAM是最好的选择。

InnoDb：

* 1. 支持ACID的事务。支持事务的四种隔离级别；默认是可重复读级别
  2. 支持行级锁及外检约束；因此可以支持写并发；
  3. 不存储总行数；
  4. 主键索引采用聚集索引（索引的数据域存储数据文件本身），辅索引的数据域存储主键的值；因此从辅索引查找数据，需要先通过辅索引找到主键值，再访问主索引。

适用场景：innnodb具有事务。支持四个事务隔离级别。回滚，崩溃修复能力和多版本并发的事务安全。如果应用中需要执行大量的insert和update操作，则应该适用innodb。可以提高并发性能。

1. 数据库的优化（从sql语句优化和索引两个部分回答）

索引优化：

* 1. 建立聚集索引
  2. 常查询数据建立索引
  3. 最左前缀原则（如果索引不符合最左原则，会进行全索引查询然后进行where匹配）
  4. 不要建立无意义的索引

Sql语句优化

* 1. 查询语句尽可能使用索引。
  2. 常用的技巧：
* 保证不查询多余的行
* 慎用distinct关键字 distinct在查询一个字段或者很少字段的情况下使用，会避免重复数据的出现，给查询带来优化结果。
* 使用连接（join）来代替子查询
* 使用limit对查询结果的记录进行限定
* 一次提交多个Insert语句

1. 索引有B+索引和hash索引，各自的区别

* Hash索引不支持范围和排序查找
* Hahs索引只能用=的条件
* Mysql 中支持hash索引的数据库引擎有memeory引擎.hash索引的理论查询时间复杂度位O（1）

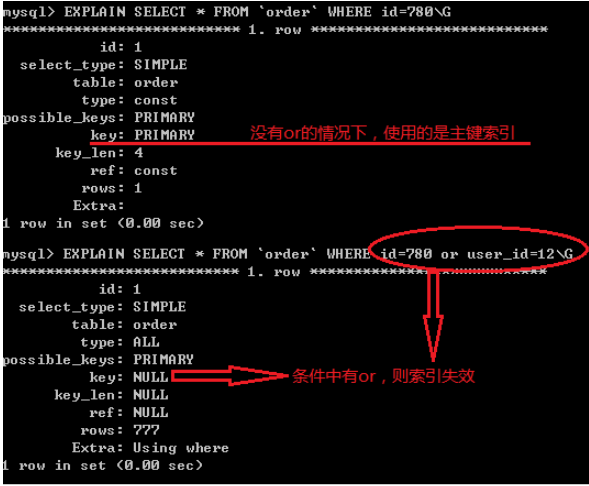
1. B+索引数据结构，和B树的区别
2. 索引的分类（主键索引、唯一索引），最左前缀原则，哪些情况索引会失效

唯一索引：索引列的值必须唯一，但允许有空值。

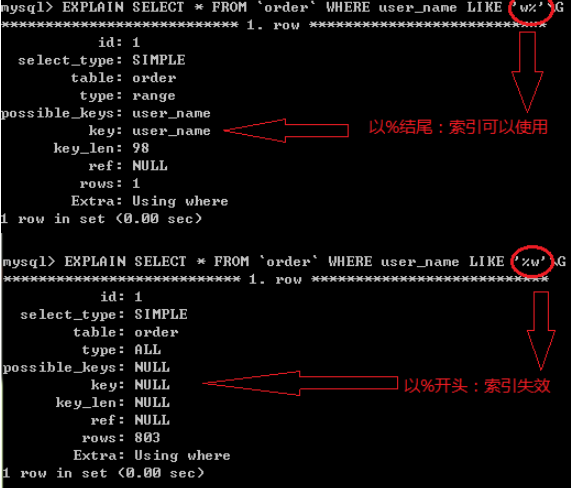
主键索引：它是一种特殊的唯一索引，不允许有空值。

索引失效：

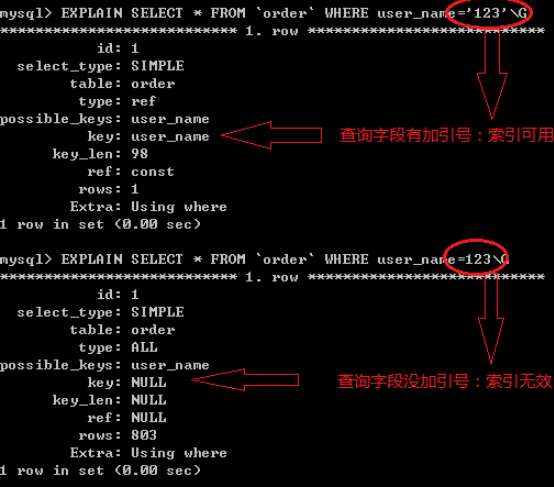
* 1. 组合索引中不符合最左原则。
  2. 如果条件中有or，即使其中有条件带索引也不会使用。这也是为什么尽量少用or的原因



* 1. Like查询以%开头



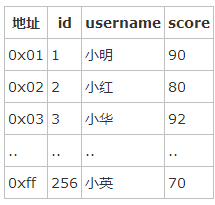
* 1. 如果列类型是字符串，那一定要在条件中将数据使用引号引用起来，否则使用索引。



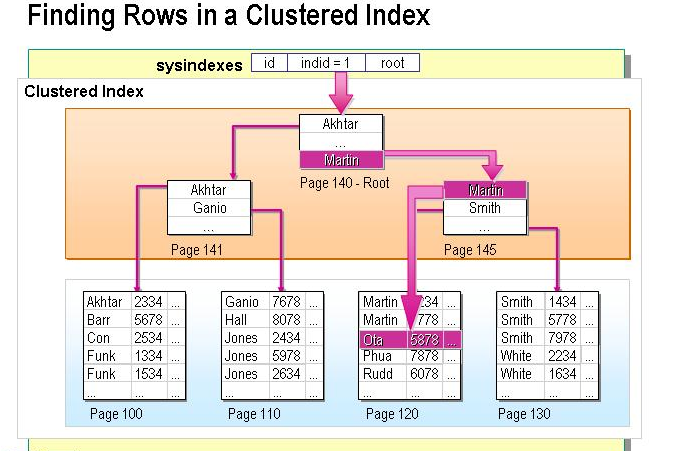
1. 聚集索引和非聚集索引区别。

聚集索引指的是索引的一种组织形式，而且是一种SQL规范。

聚集索引的定义：数据行在硬盘上的物理存储顺序和数据表中逻辑存储顺序相同，一个表只能有一个聚集索引。



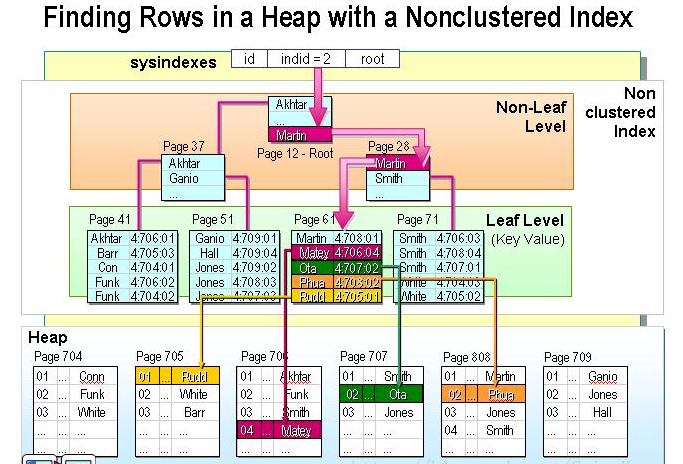
上图模拟了聚集索引的存储形式，如果我们在数据行id上建立聚集索引，那么整个表在磁盘上的存储顺序，就是按照表中id的顺序进行存储的。也是由于这个原因，所以一张表中只能有一个聚集索引。在MySQL中，innodb引擎默认按照主键索引进行聚集，当没有主键时，innodb会以唯一的非空索引来代替，如果没有主键也没有唯一的非空索引，innodb会生产一个隐藏的主键索引，然后在上面进行聚集。所以主键的创建（聚集索引的创建），尽量在建立表的时候创建，如果后续添加，数据库会根据聚集索引将磁盘的数据重新排列，以确保数据的物理顺序和索引列所在表中的逻辑顺序相同。



聚集索引的叶子结点就是对应的数据结点，可以直接查找到全部列的数据。也就是说聚集索引的叶子结点存储结构就是数据在硬盘上的存储结构。

非聚集索引和聚集索引是相对立的概念。

非聚集索引的索引逻辑顺序和磁盘上存储的物理顺序不同，一个表中可以有多个非聚集索引。



从上图可以看出，非聚集索引生成B+树的叶子结点就是索引的数据列和相应的数据行所在的地址。也就是说如果建立非聚集索引，要查询建立索引列外的数据，需要进行两次检索而聚集索引就不会出现这种情况。因为聚集索引的叶子结点就是全部的数据内容。

对于MySQL来说，平时对于某列或多列建立的索引就是非聚集索引，而innodb引擎中，必须有聚集索引。所以非聚集索引中包括聚集索引的列。Innodb的二次查询，就是进行的聚集索引查询。

1. 有哪些锁（乐观锁悲观锁），select时怎么加排它锁
2. 关系型数据库和非关系型数据库区别

* 关系型数据库通过外键关联来建立表与表之间的关系。Mysql 表
* 非关系数据库通茶以对象的形式存储在数据库中。而对象与对象的关系通过每个对象自身的属性来决定。 Mongodb 键值对存储，一般不支持ACID。

1. 了解nosql
2. 数据库三范式，根据某个场景设计数据表（可以通过手绘ER图）
3. 数据库的主从复制
4. 使用explain优化sql和索引
5. long\_query怎么解决
6. 内连接、外连接、交叉连接、笛卡儿积等

**深入**

1. MVCC机制

Muti\_version concurrency control 多版本并发控制。

客观上讲，Mvcc是一种乐观锁的一套实现方式。就是每行都有版本号，保存时根据版本号决定是否保存成功。

<https://blog.csdn.net/w2064004678/article/details/83012387>

1. 根据具体场景，说明版本控制机制
2. 死锁怎么解决
3. varchar和char的使用场景。

Varchar 往往用来保存长度可变的字符串。我们在设置的时候只是给你设置一个最大值。然后系统会根据实际存储的数据量来分配。Varchar数据类型能够节约磁盘空间。但是由于其长度是可变的，当进行数据更新时，可能会导致一些额外的工作。

Char 采用固定长度的存储方式，当保存数据时，即使没有达到最大的长度，系统也会为其分配这么多的存储空间。会造成磁盘的浪费，但是数据更改时，不会造成存储空间的扩容之类的。

* 1. 在字符串长度相近时，比如身份证号，MD5值得的时候可以采用固定长度存储char类型。在字符串长度比较小，可以采用固定长度，虽然可能会产生碎片，但是在进行数据更新时，会减少开销。
  2. 在考虑减少碎片问题的时候，可以采用varchar去减少磁盘碎片。

1. mysql并发情况下怎么解决（通过事务、隔离级别、锁）

**Redis**

1. redis数据结构有哪些
2. redis队列应用场景
3. redis和Memcached（支持数据持久化）
4. 分布式使用场景（储存session等）
5. 发布/订阅使用场景

# 三、操作系统

1. 内存的页面置换算法

地址映射过程中，若在页面中发现所要访问的页面不在内存中，则产生缺页中断。而发生缺页中断时，如果操作系统内存中没有空闲页面，则操作系统必须在内存选择一个页面将其移出内存，以便为即将调入的页面让出空间，而用选择淘汰哪一个页面的算法叫做页面置换算法。

* 最佳置换算法0PT：(理想置换算法)从主存中移出永远不再需要的页面；如无这样的页面存在，则选择最长时间不要需要访问的页面。这样可以保证最低的缺页率。
* 先进先出置换算法（FIFO）
* 最近最久未使用算法（LRU）
* 时钟置换算法（CLOCK）

1. 进程调度算法

* 先来先服务调度算法：FCFS first come first served:谁第一个排队谁就先执行。在它执行过程中，不会被中断。
* 短作业优先算法：对预计执行时间短的进程优先分配处理机。通常后来的短进程不会抢先正在执行的进程；对长进程非常不利，可能长时间得不到执行。
* 时间片轮转法：
* 优先级算法:  
  多级反馈队列:不同的队列应用不同的进程调度算法。

1. 进程间通信方式

* 信号量
* 共享内存
* 消息队列、
* 管道

1. 进程线程区别
2. 进程之间的通信
3. 父子进程、孤儿进程
4. fork进程时的操作，   
   这个部分我回答的都不好，只能是死记硬背，建议基础好的同学多看看操作系统这部分，能大大加分。

# 四、算法

**基础**

1. 剑指OFFER的各个题目是最常见的，即使不是原题也是题目的变体，因为面试不像笔试，一般不会出特别困难的题目，所以剑指OFFER上小而精的题目就非常适合。建议手刷一遍。PHP的同学可以参考专栏[剑指OFFER](http://blog.csdn.net/column/details/15795.html" \t "_blank)
2. 二叉树相关（层次遍历、求深度、求两个节点距离、翻转二叉树、前中后序遍历）
3. 链表相关（插入节点、链表逆置、使用链表进行大数字的加减，双向链表实现队列、寻找链表中的环）
4. 堆（大量数据中寻找最大N个数字几乎每次都会问，还有堆在插入时进行的调整）
5. 排序（八大排序，各自的时间复杂度、排序算法的稳定性。快排几乎每次都问）
6. 二分查找（一般会深入，如寻找数组总和为K的两个数字）
7. 两个栈实现队列。
8. 图（深度广度优先遍历、单源最短路径、最小生成树）
9. 动态规划问题。

**深入**

1. 红黑树性质
2. 分治法和动态规划的区别
3. 计算时间复杂度
4. 二叉树和哈希表查找的时间复杂度

栈和链表是面试算法的时候经常用到的工具，多考虑怎么用数据结构的性质解决，因为面试不像笔试，对基础数据结构关注的比较多一些，一般问题也比较简单。然后取模也是常用的工具（比如有一次问怎么让100个进程按规定的权重被调用，就可以用取模的方式）。   
面试官一般会先出简单的问题，然后深入地问下去，最好是根据他的思路走，因为能听懂他的提示也是需要考察的能力。

# LINUX

1. 硬链接和软连接区别
2. kill用法，某个进程杀不掉的原因（进入内核态，忽略kill信号）
3. linux用过的命令
4. 系统管理命令（如查看内存使用、网络情况）
5. 管道的使用 |
6. grep的使用，一定要掌握，每次都会问在文件中查找
7. shell脚本
8. find命令
9. awk使用

**面向对象、设计模式**

1. 接口和抽象类区别
2. 单继承
3. construct的调用顺序（子类父类之间）
4. 设计模式（工厂模式、策略模式、单例模式、装饰模式比较常见）
5. OOP特性，通过哪些机制实现的
6. 重写和重载区别
7. 静态类静态方法
8. 根据某个需求设计一个类（主要考虑类之间的继承关系和属性的权限设置）
9. 设计模式 包括工厂模式 单例模式

# 项目

1. 项目中遇到的困难（提前想好，并且把实现或者优化方法说清楚）
2. 系统的量级、pv、uv等
3. 应对高并发的解决办法（分布式）
4. 在项目中主要负责了哪些工作。
5. nginx的负载均衡
6. 分布式缓存的一致性，服务器如何扩容（哈希环）

总之要把写在简历上的项目部分熟悉一遍，技术栈、项目功能、难点都要考虑好。

String 字符串常亮

StringBuffer 字符串常量（线程安全）

StringBuilder 字符串常量（非线程安全）

String和StringBuffer的主要性能区别其实在于String是不可变对象，因此每次对String类型进行改变的时候，都是在创建新的对象，将指针指向新的对象。

StringBuffer和StringBuilde的主要区别就线程安全和非线程安全