**数据库相关**

**常见的数据库优化手段索引的优缺点**

​ 常见的优化手段有：

​ 1、在编写SQL语句时不使用\*，而是使用字段名，哪怕每一个字段名都写上去，

​ 2、添加索引，使查询效率变高，尤其是在where条件中的字段

​ 3、尽量不要在列上进行运算（函数操作和表达式操作）否则索引失效

​ 4、不使用not in 和 <> 以及！=否则索引失效

​ 5、在使用like语句时不要以%开头否则索引失效

​ 6、多表联查时多使用小表作为驱动表，因为（驱动表必须全表查询）

​

**什么字段上建立索引数据库**

​ 1、在作为where条件出现较多的表

​ 2、尽量建立索引时将顺序排好的字段（否则查询时中间一个字段没有用上后面的都用不上）

​

​ 3、尽量选择一些和其他表有关联的字段，如订单表的用户字段

**你们数据库的高可用架构是怎么样的？**

​ 一般来说数据库的备份我们采用主从复制（注：单纯的主从并不是高性能或者高可用的），而为了提高性能我们在主从之后会使用Mycat进行读写分离，但是他也只是做到了高性能并没有做到高可用，那么如何做到高可用呢，就是我们使用KeepAlive进行虚拟化ip，如果使用简单的高可用加高性能那么我们就直接通过KeepAlive，去连接主机然后再进行访问，如果主机挂掉了就直接使用丛机进行读写，等恢复主机之后将主机变为丛机，读取丛机的数据，或者恢复后同步数据，再使用指向主机，下面分别是高可用和一个高性能可用的架构图

​

**如何保证数据库主从一致性？**

​ **方案一：忽略**

​ 任何脱离业务的架构设计都是耍流氓，绝大部分业务，例如：百度搜索，淘宝订单，QQ消息，58帖子都允许短时间不一致。如果业务能接受，最推崇此法。别把系统架构搞得太复杂。

​ **方案二：选择性主读**

​ 将哪个库，哪个表，哪个主键三个信息拼装一个key设置到cache里，这条记录的超时时间，设置为“主从同步时延” ，画外音：key的格式为“db:table:PK”，假设主从延时为1s，这个key的cache超时时间也为1s。 这是要读哪个库，哪个表，哪个主键的数据呢，也将这三个信息拼装一个key，到cache里去查询， 在cache里记录哪些记录发生过写请求，来路由读主还是读从

**为什么mongodb的索引用了B树，而mysql用B+树？**

​ 我们首先来看一下B树的结构

​ B树的特点就是每一层的节点数据非常多，而层数比较少，目的就是为了减少IO次数

​ B树的每个节点都是有data域的（指针），这无疑增大了节点大小

​ 再来看一下B+树的结构

​ B+树的特点就是，B+树所有的Data域在叶子节点 ，非叶子节点存储索引数据

​ B+树只要遍历叶子节点就可以实现整棵树的遍历

**用mysql过程中，有遇到什么问题么？**

**你们生产用的是哪种事务隔离级别，为什么？**

**什么情况下会造成索引失效？**

​ 1、使用or关键字，如果条件中有or，即使其中有条件带索引也不会使用

​ 注意：使用or，又想索引生效，只能将or条件中的每个列都加上索引

​ 2、索引顺序不一致，例如索引为（name，age，email），而查询时（name，email，age）则会造成只有name生效，其他两个字段用不到索引

​ 注意：尽量避免索引顺序不一致，如果实在避免不了，在设计时就应先考虑顺序

​ 3、以like %开头的关键字，如果以like %name查询，会使索引失效

​ 注意：尽量使用 like name%这种，因为%号在后面还是能实用索引

​ 4、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，否则不会使用索引

​ 注意：字符串类型列一定要使用引号（虽然情况很少）

​ 5、 where语句中使用 <>和 != 还有not in，如果使用了不等于那么他会全表扫描

​ 注意：尽量不要使用不等于或者not

​ 6、 where语句中对字段表达式操作，例如where age \* 2 = 100

​ 注意：不要使用表达式，尽量使用age = 50

**数据结构**

**计算机网络**

**TCP，UDP区别**

1、TCP面向连接（如打电话要先拨号建立连接）;UDP是无连接的，即发送数据之前不需要建立连接

2、TCP提供可靠的服务。也就是说，通过TCP连接传送的数据，无差错，不丢失，不重复，且按序到达;UDP尽最大努力交付，即不保证可靠交付

3、TCP面向字节流，实际上是TCP把数据看成一连串无结构的字节流;UDP是面向报文的

UDP没有拥塞控制，因此网络出现拥塞不会使源主机的发送速率降低（对实时应用很有用，如IP电话，实时视频会议等）

4、每一条TCP连接只能是点到点的;UDP支持一对一，一对多，多对一和多对多的交互通信

5、TCP首部开销20字节;UDP的首部开销小，只有8个字节

6、TCP的逻辑通信信道是全双工的可靠信道，UDP则是不可靠信道

总结：

​ TCP面向连接，即：需要连接才能发送数据，并且TCP提供的服务是可靠的有序的，不重复，不丢失，他将数据看成字节流进行传输，点对点的连接，TCP头部开销20字节，TCP是逻辑通信的可靠信道

​

​ UDP是无连接的直接发送，UDP不保证可靠地将消息交付，UDP没有拥堵或阻塞状态，不会让发送效率变低，UDP可以一对一，一对多，多对多，多对一，首部8个字节，UDP是不可靠信道

​ 优缺点对比：

​ TCP慢，但是消息传送可靠，并且无差错不丢失，但是效率并不高，并且首部开销大，还有可能阻塞或者拥堵

​ UDP是无连接的，不需要询问对方直接发送，发送速度快，并且没有阻塞拥堵状态，还能一对多多对多，首部开销少，但是不保证消息的可靠性，容易丢失，

​ 在根据不同的场合使用不同的技术

**TCP核心**

​ TCP的Flags有几个

​ URG：紧急指针标志

​ ACK：确认序号编制

​ PSH：push标志

​ RST：重置连接标志

​ SYN：同步序列好，用于建立连接过程

​ FIN：finish标志，用于释放链接

​ 这些flags就是用于TCP的通信的核心

**三次握手**

​ TCP如何进行三次握手的呢？

​ 假设我们通过A去访问B那么，A就会带着一个SYN进行连接请求B，这是第一次握手，如果B收到了并且统同意连接，那么B就会向A发送一个ACK+SYN用于确认并且请求连接，这里就是第二次的握手，然后A收到了B的ACK表示B已经同意了连接，那么A就会响应一个ACK表示收到了请求，这个时候就是真正的连接并且传输了，最后的连接就是第三次握手，这个就是TCP的三次握手

​

下面是完整的示例图：

\*\* LISTEN \*\* 表示socket已经处于listen状态了，可以建立连接；  
​  
\*\* SYN\_SENT \*\* 表示socket在发出connect连接的时候，会首先发送SYN报文，然后等待另一端发送的确认报文(ACK)，表示这端已经发送完SYN报文了  
​  
\*\* SYN\_RCVD \*\* 表示一端已经接收到SYN报文了；  
​  
\*\* ESTABLISHED \*\* 表示已经建立连接了，可以发送数据了  
​

​

**四次挥手**

​ 什么是TCP的四次挥手呢？

​ TCP的四次挥手就是A连接到了B，并且完成传输数据之后，这个连接关闭的过程我们称之为四次挥手，

​ TCP的四次挥手的过程

​ 那么我们又来假设A已经连接到了B，那么我们需要断开连接，首先我们的A会向B发起一个FIN来表示他传输完成，需要进行资源的释放，这里A向B发送了一个FIN表示第一次挥手，然后B收到了这个FIN，表示他知道了需要关闭资源，那么B就会给A发送一个ACK表示他收到了，然后进入到半关闭的CLOSE\_WAIT状态，这个时候就是第二次挥手，然后B虽然处于了半关闭的状态，但是还是可以发送消息的，这个时候他会再次给A发送一个FIN表示，我已经关闭，你也关闭吧，这里就是第三次的挥手，那么A就会给B发送一个ACK表示他也收到了会进行关闭（但是不会立即关闭），这里的ACK表示A收到了然后给B发送确认，B收到之后才会进行关闭

下面是完整示例

\*\* FIN\_WAIT\_1 \*\* 表示在等待另一方的FIN报文，和FIN\_WAIT\_2的区别是，FIN\_WAIT\_1表示socket现在要主动关闭连接，在发送完FIN报文后socket进入FIN\_WAIT\_1状态，当收到另一方发送FIN的ACK之后立即进入FIN\_WAIT\_2状态；  
​  
\*\* FIN\_WAIT\_2 \*\* 同上，此时需要做的事情是可能还会接收数据，然后等待另一方的FIN；  
​  
\*\* TIME\_WAIT \*\* 存在主动关闭的一方，表示收到了对方的FIN报文，并发送出了ACK报文，就等2MSL(Max Segment Lifetime))后即可回到CLOSED可用状态了，需要等一段时间时原因是网络是不可靠的，不能保证这个ACK发送成功了，如果失败了，对端会超时重传FIN；  
​  
\*\* CLOSING \*\* 表示在发送FIN之后，没有收到对方的ACK，而是收到了对方的FIN，这中情况很少见，只有在两端几乎同时关闭同一个socket的时候才会出现CLOSING状态；  
​  
\*\* CLOSE\_WAIT \*\* 表示收到对方的FIN之后，回给对方ACK，此时处于CLOSE\_WAIT状态，等待关闭，要看自己是否还有数据要发送；  
​  
\*\* LAST\_ACK \*\* 表示收到对方的FIN之后，回给对方ACK，然后自己也要关闭发送FIN，等待另一方的ACK时候的状态；  
​  
\*\* CLOSED \*\* 这个状态表示连接已经断开。  
​

**为什么要三次握手**

​ 因为我们要知道TCP的通信是一定要保证数据的投递的可靠性，还有一致性，所以他建立连接的时候必须要确定双方的状态，要保证一定的可靠的投递，所以需要三次握手保证双方的数据传输环境，然后再进行通信

**为什么要四次挥手**

​ 由于TCP连接是全双工的，因此每个方向都必须单独进行关闭。这个原则是当一方完成它的数据发送任务后就能发送一个FIN来终止这个方向的连接。收到一个 FIN只意味着这一方向上没有数据流动，一个TCP连接在收到一个FIN后仍能发送数据。首先进行关闭的一方将执行主动关闭，而另一方执行被动关闭。

**长连接和短连接**

​ HTTP的长连接和短连接本质上是TCP长连接和短连接。HTTP属于应用层协议，在传输层使用TCP协议，在网络层使用IP协议。 IP协议主要解决网络路由和寻址问题，TCP协议主要解决如何在IP层之上可靠地传递数据包，使得网络上接收端收到发送端所发出的所有包，并且顺序与发送顺序一致。TCP协议是可靠的、面向连接的。

​ 在HTTP/1.0中默认使用短连接。也就是说，客户端和服务器每进行一次HTTP操作，就建立一次连接，任务结束就中断连接。当客户端浏览器访问的某个HTML或其他类型的Web页中包含有其他的Web资源（如JavaScript文件、图像文件、CSS文件等），每遇到这样一个Web资源，浏览器就会重新建立一个HTTP会话。

​ 而从HTTP/1.1起，默认使用长连接，用以保持连接特性。使用长连接的HTTP协议，会在响应头加入这行代码：

Connection:keep-alive

在使用长连接的情况下，当一个网页打开完成后，客户端和服务器之间用于传输HTTP数据的TCP连接不会关闭，客户端再次访问这个服务器时，会继续使用这一条已经建立的连接。Keep-Alive不会永久保持连接，它有一个保持时间，可以在不同的服务器软件（如Apache）中设定这个时间。实现长连接需要客户端和服务端都支持长连接。

HTTP协议的长连接和短连接，实质上是TCP协议的长连接和短连接。

**连接池适合长连接还是短连接**

​ 连接池适合长连接，因为在使用连接池的时候我们需要快速的连接，并且持续性的，所以我们使用长连接，在一般的连接池中也会设定最小连接数，和空闲连接数为的就是能快速的持续的传输数据所以连接池适合使用长连接

​