## 1.算法相关

#### 1.1快排

function quick\_sort($arr)  
{  
//判断参数是否是一个数组  
if(!is\_array($arr)) return false;  
//递归出口:数组长度为1，直接返回数组  
$length=count($arr);  
if($length<=1) return $arr;  
//数组元素有多个,则定义两个空数组  
$left=$right=array();  
//使用for循环进行遍历，把第一个元素当做比较的对象  
for($i=1;$i<$length;$i++)  
{  
//判断当前元素的大小  
if($arr[$i]<$arr[0]){  
$left[]=$arr[$i];  
}else{  
$right[]=$arr[$i];  
}  
}  
//递归调用  
$left=quick\_sort($left);  
$right=quick\_sort($right);  
//将所有的结果合并  
return array\_merge($left,array($arr[0]),$right);

#### 1.2二分查找

function getValue4($num,$arr,$start = 0,$end = 100){  
//采用二分法查找  
$middle = floor(($end + $start) / 2);  
  
//判断  
if($arr[$middle] == $num){  
//已经找到了，递归的出口  
return $middle + 1;  
}elseif($arr[$middle] < $num){  
//要查找的元素在数组的后半段  
$start = $middle + 1;  
//边界值  
if($start >= $end){  
//没有找到，但是已经超出边界值，递归出口  
return false;  
}  
//调用自己去查找：递归点  
return getValue4($num,$arr,$start,$end); //getValue4($num,$arr,51,100)  
}else{  
//要查找的元素在数组的前半段  
$end = $middle - 1;  
//判断边界值  
if($end < 0)return false;  
  
//调用自己：递归点  
return getValue4($num,$arr,$start,$end); //getValue4($num,$arr,0,49)  
}  
  
//都没有找到  
return false;  
}

#### 1.3桶排序了解一下

#### 1.4堆排序了解一下

#### 1.5最短路径优化

#### 推荐看一下《啊哈，算法》快速成型，两小时基本能搞定大部分问题。

## 2.mysql相关

#### 2.1sql注入

SQL注入，就是通过把SQL命令插入到Web表单提交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的SQL命令。具体来说，它是利用现有应用程序，将（恶意的）SQL命令注入到后台数据库引擎执行的能力，它可以通过在Web表单中输入（恶意）SQL语句得到一个存在安全漏洞的网站上的数据库，而不是按照设计者意图去执行SQL语句。

#### 2.2mysql存储引擎（b+树了解一下）

1）MyISAM：  
每个MyISAM在磁盘上存储成三个文件。第一个文件的名字以表的名字开始，扩展名指出文件类型。.frm文件存储表定义。数据文件的扩展名为.MYD (MYData)。  
MyISAM表格可以被压缩，而且它们支持全文搜索。不支持事务，而且也不支持外键。如果事物回滚将造成不完全回滚，不具有原子性。在进行updata时进行表锁，并发量相对较小。如果执行大量的SELECT，MyISAM是更好的选择。  
MyISAM的索引和数据是分开的，并且索引是有压缩的，内存使用率就对应提高了不少。能加载更多索引，而Innodb是索引和数据是紧密捆绑的，没有使用压缩从而会造成Innodb比MyISAM体积庞大不小  
MyISAM缓存在内存的是索引，不是数据。而InnoDB缓存在内存的是数据，相对来说，服务器内存越大，InnoDB发挥的优势越大。

优点：查询数据相对较快，适合大量的select，可以全文索引。  
缺点：不支持事务，不支持外键，并发量较小，不适合大量update

2）InnoDB：（参数说明：[Mysql存储引擎之Innodb重要参数说明](http://www.cnblogs.com/kevingrace/p/6133818.html)）  
这种类型是事务安全的。.它与BDB类型具有相同的特性,它们还支持外键。InnoDB表格速度很快。具有比BDB还丰富的特性,因此如果需要一个事务安全的存储引擎，建议使用它。在update时表进行行锁，并发量相对较大。如果你的数据执行大量的INSERT或UPDATE，出于性能方面的考虑，应该使用InnoDB表。  
优点：支持事务，支持外键，并发量较大，适合大量update  
缺点：查询数据相对较快，不适合大量的select  
对于支持事物的InnoDB类型的表，影响速度的主要原因是AUTOCOMMIT默认设置是打开的，而且程序没有显式调用BEGIN 开始事务，导致每插入一条都自动Commit，严重影响了速度。可以在执行sql前调用begin，多条sql形成一个事物（即使autocommit打开也可以），将大大提高性能。

3）memory，表存在内存里面，可以用hash索引

#### 2.3Mysql数据事务隔离级别

一、事务的基本要素（ACID）

1、原子性（Atomicity）：事务开始后所有操作，要么全部做完，要么全部不做，不可能停滞在中间环节。事务执行过程中出错，会回滚到事务开始前的状态，所有的操作就像没有发生一样。也就是说事务是一个不可分割的整体，就像化学中学过的原子，是物质构成的基本单位。

2、一致性（Consistency）：事务开始前和结束后，数据库的完整性约束没有被破坏 。比如A向B转账，不可能A扣了钱，B却没收到。

3、隔离性（Isolation）：同一时间，只允许一个事务请求同一数据，不同的事务之间彼此没有任何干扰。比如A正在从一张银行卡中取钱，在A取钱的过程结束前，B不能向这张卡转账。

4、持久性（Durability）：事务完成后，事务对数据库的所有更新将被保存到数据库，不能回滚。

#### 2.4事务隔离级别

1）read uncommitted : 读取尚未提交的数据 ：哪个问题都不能解决

2）read committed：读取已经提交的数据 ：可以解决脏读 ---- oracle默认的

3）repeatable read：重读读取：可以解决脏读 和 不可重复读 ---mysql默认的

4）serializable：串行化：可以解决 脏读 不可重复读 和 幻读---相当于锁表

**1、脏读：事务A读取了事务B更新的数据，然后B回滚操作，那么A读取到的数据是脏数据**

**2、不可重复读：事务 A 多次读取同一数据，事务 B 在事务A多次读取的过程中，对数据作了更新并提交，导致事务A多次读取同一数据时，结果 不一致。**

**3、幻读：系统管理员A将数据库中所有学生的成绩从具体分数改为ABCDE等级，但是系统管理员B就在这个时候插入了一条具体分数的记录，当系统管理员A改结束后发现还有一条记录没有改过来，就好像发生了幻觉一样，这就叫幻读。**

**小结：不可重复读的和幻读很容易混淆，不可重复读侧重于修改，幻读侧重于新增或删除。解决不可重复读的问题只需锁住满足条件的行，解决幻读需要锁表**

#### 2.5MySQL优化

优化方案：

SQL和索引 > 数据库表结构 > 系统配置 > 硬件   
但成本从低到高。

<https://blog.csdn.net/sinat_23080035/article/details/52802569>

很详细。

#### 2.6MySQL防注入

应对方法：

1. mysql\_escape\_string() 转义特殊字符（(PHP 4 >= 4.3.0, PHP 5)）(mysql\_real\_escape\_string必须先链接上数据库，否则会报错)

下列字符受影响：

\x00 //对应于ascii字符的NULL

\n //换行符且回到下一行的最前端

\r //换行符

\ //转义符

'

"

\x1a //16进制数

如果成功，则该函数返回被转义的字符串。如果失败，则返回 false。

1. addslashes(): 函数返回在预定义字符之前添加反斜杠的字符串 (stripslashes()实现字符串还原)

预定义的字符有：

单引号（'）

双引号（"）

反斜杠（\）

NULL

1. prepared  statements(预处理机制)

<?php

$mysqli = new mysqli("example.com", "user", "password", "database");

if ($mysqli->connect\_errno) {

echo "Failed to connect to MySQL: (" . $mysqli->connect\_errno . ") " . $mysqli->connect\_error;

}

/\* Non-prepared statement \*/

if (!$mysqli->query("DROP TABLE IF EXISTS test") || !$mysqli->query("CREATE TABLE test(id INT)")) {

echo "Table creation failed: (" . $mysqli->errno . ") " . $mysqli->error;

}

/\* Prepared statement, stage 1: prepare \*/

if (!($stmt = $mysqli->prepare("INSERT INTO test(id) VALUES (?)"))) {

echo "Prepare failed: (" . $mysqli->errno . ") " . $mysqli->error;

}

/\* Prepared statement, stage 2: bind and execute \*/

$id = 1;

if (!$stmt->bind\_param("i", $id)) {

echo "Binding parameters failed: (" . $stmt->errno . ") " . $stmt->error;

}

if (!$stmt->execute()) {

echo "Execute failed: (" . $stmt->errno . ") " . $stmt->error;

}

?>

#### 2.7索引原理及优化慢查询案例

<https://www.cnblogs.com/bypp/p/7755307.html>

不幸碰见这种题，就算哔了狗吧。内容很多，主要是索引的原理。

很可能会问不同数据库之间的区别。

#### 2.8大量数据查询处理案例

1、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。   
  
2、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。   
  
3、应尽量避免在 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，如：   
select id from t where num is null   
可以在num上设置默认值0，确保表中num列没有null值，然后这样查询：   
select id from t where num=0   
  
4、尽量避免在 where 子句中使用 or 来连接条件，否则将导致引擎放弃使用索引而进行全表扫描，如：   
select id from t where num=10 or num=20   
可以这样查询：   
select id from t where num=10   
union all   
select id from t where num=20   
  
5、下面的查询也将导致全表扫描：(不能前置百分号)   
select id from t where name like ‘%c%’   
若要提高效率，可以考虑全文检索。   
  
6、in 和 not in 也要慎用，否则会导致全表扫描，如：   
select id from t where num in(1,2,3)   
对于连续的数值，能用 between 就不要用 in 了：   
select id from t where num between 1 and 3   
  
7、如果在 where 子句中使用参数，也会导致全表扫描。因为SQL只有在运行时才会解析局部变量，但优化程序不能将访问计划的选择推迟到运行时；它必须在编译时进行选择。然 而，如果在编译时建立访问计划，变量的值还是未知的，因而无法作为索引选择的输入项。如下面语句将进行全表扫描：   
select id from t where num=@num   
可以改为强制查询使用索引：   
select id from t with(index(索引名)) where num=@num   
  
8、应尽量避免在 where 子句中对字段进行表达式操作，这将导致引擎放弃使用索引而进行全表扫描。如：   
select id from t where num/2=100   
应改为:   
select id from t where num=100\*2   
  
9、应尽量避免在where子句中对字段进行函数操作，这将导致引擎放弃使用索引而进行全表扫描。如：   
select id from t where substring(name,1,3)=’abc’–name以abc开头的id   
select id from t where datediff(day,createdate,’2005-11-30′)=0–’2005-11-30′生成的id   
应改为:   
select id from t where name like ‘abc%’   
select id from t where createdate>=’2005-11-30′ and createdate<’2005-12-1′   
  
10、不要在 where 子句中的“=”左边进行函数、算术运算或其他表达式运算，否则系统将可能无法正确使用索引。   
  
11、在使用索引字段作为条件时，如果该索引是复合索引，那么必须使用到该索引中的第一个字段作为条件时才能保证系统使用该索引，否则该索引将不会被使 用，并且应尽可能的让字段顺序与索引顺序相一致。   
  
12、不要写一些没有意义的查询，如需要生成一个空表结构：   
select col1,col2 into #t from t where 1=0   
这类代码不会返回任何结果集，但是会消耗系统资源的，应改成这样：   
create table #t(…)   
  
13、很多时候用 exists 代替 in 是一个好的选择：   
select num from a where num in(select num from b)   
用下面的语句替换：   
select num from a where exists(select 1 from b where num=a.num)   
  
14、并不是所有索引对查询都有效，SQL是根据表中数据来进行查询优化的，当索引列有大量数据重复时，SQL查询可能不会去利用索引，如一表中有字段 sex，male、female几乎各一半，那么即使在sex上建了索引也对查询效率起不了作用。   
  
15、索引并不是越多越好，索引固然可以提高相应的 select 的效率，但同时也降低了 insert 及 update 的效率，因为 insert 或 update 时有可能会重建索引，所以怎样建索引需要慎重考虑，视具体情况而定。一个表的索引数最好不要超过6个，若太多则应考虑一些不常使用到的列上建的索引是否有 必要。   
  
16.应尽可能的避免更新 clustered 索引数据列，因为 clustered 索引数据列的顺序就是表记录的物理存储顺序，一旦该列值改变将导致整个表记录的顺序的调整，会耗费相当大的资源。若应用系统需要频繁更新 clustered 索引数据列，那么需要考虑是否应将该索引建为 clustered 索引。   
  
17、尽量使用数字型字段，若只含数值信息的字段尽量不要设计为字符型，这会降低查询和连接的性能，并会增加存储开销。这是因为引擎在处理查询和连接时会 逐个比较字符串中每一个字符，而对于数字型而言只需要比较一次就够了。   
  
18、尽可能的使用 varchar/nvarchar 代替 char/nchar ，因为首先变长字段存储空间小，可以节省存储空间，其次对于查询来说，在一个相对较小的字段内搜索效率显然要高些。   
  
19、任何地方都不要使用 select \* from t ，用具体的字段列表代替“\*”，不要返回用不到的任何字段。  
  
20、尽量使用表变量来代替临时表。如果表变量包含大量数据，请注意索引非常有限（只有主键索引）。   
  
21、避免频繁创建和删除临时表，以减少系统表资源的消耗。   
  
22、临时表并不是不可使用，适当地使用它们可以使某些例程更有效，例如，当需要重复引用大型表或常用表中的某个数据集时。但是，对于一次性事件，最好使 用导出表。   
  
23、在新建临时表时，如果一次性插入数据量很大，那么可以使用 select into 代替 create table，避免造成大量 log ，以提高速度；如果数据量不大，为了缓和系统表的资源，应先create table，然后insert。   
  
24、如果使用到了临时表，在存储过程的最后务必将所有的临时表显式删除，先 truncate table ，然后 drop table ，这样可以避免系统表的较长时间锁定。   
  
25、尽量避免使用游标，因为游标的效率较差，如果游标操作的数据超过1万行，那么就应该考虑改写。   
  
26、使用基于游标的方法或临时表方法之前，应先寻找基于集的解决方案来解决问题，基于集的方法通常更有效。   
  
27、与临时表一样，游标并不是不可使用。对小型数据集使用 FAST\_FORWARD 游标通常要优于其他逐行处理方法，尤其是在必须引用几个表才能获得所需的数据时。在结果集中包括“合计”的例程通常要比使用游标执行的速度快。如果开发时 间允许，基于游标的方法和基于集的方法都可以尝试一下，看哪一种方法的效果更好。   
  
28、在所有的存储过程和触发器的开始处设置 SET NOCOUNT ON ，在结束时设置 SET NOCOUNT OFF 。无需在执行存储过程和触发器的每个语句后向客户端发送 DONE\_IN\_PROC 消息。   
  
29、尽量避免向客户端返回大数据量，若数据量过大，应该考虑相应需求是否合理。   
  
30、尽量避免大事务操作，提高系统并发能力。

## 3.Redis相关

#### 3.1数据类型：

字符串，链表，哈希，集合，有序集合

注意不同场景下的使用数据类型的选择。

#### 3.2Redis单线程为什么快？

1、完全基于内存

Redis是纯内存数据库，相对于读写磁盘，读写内存的速度就不是几倍几十倍了，一般，hash查找可以达到每秒百万次的数量级。

2、多路复用IO

“多路”指的是多个网络连接，“复用”指的是复用同一个线程。采用多路 I/O 复用技术可以让单个线程高效的处理多个连接请求（尽量减少网络IO的时间消耗）。可以直接理解为：单线程的原子操作，避免上下文切换的时间和性能消耗；加上对内存中数据的处理速度，很自然的提高redis的吞吐量。

#### 3.3redis和memcached的区别（总结）

观点一：

1、Redis和Memcache都是将数据存放在内存中，都是内存数据库。不过memcache还可用于缓存其他东西，例如图片、视频等等；

2、Redis不仅仅支持简单的k/v类型的数据，同时还提供list，set，hash等数据结构的存储；

3、虚拟内存--Redis当物理内存用完时，可以将一些很久没用到的value 交换到磁盘；

4、过期策略--memcache在set时就指定，例如set key1 0 0 8,即永不过期。Redis可以通过例如expire 设定，例如expire name 10；

5、分布式--设定memcache集群，利用magent做一主多从;redis可以做一主多从。都可以一主一从；

6、存储数据安全--memcache挂掉后，数据没了；redis可以定期保存到磁盘（持久化）；

7、灾难恢复--memcache挂掉后，数据不可恢复; redis数据丢失后可以通过aof恢复；

8、Redis支持数据的备份，即master-slave模式的数据备份；

#### 3.4Redis怎么实现持久化

#### 3.5Redis主从同步

#### 3.6怎么去设计一个可配置的访问权限控制，利用Redis

## 4.Php相关

#### 4.1Php内存回收机制（感觉会常碰见）

#### 4.2Php是怎么实现实时类型的判断

#### 4.3Php怎么启动

#### 4.4Php-fpm是做啥的

## 5.Nginx相关

#### 5.1介绍nginx的工作原理

## 6.Rabbitmq相关

#### Rabbitmq消息怎么实现持久化

## Laravel相关

#### 为什么用laravel

#### Laravel中加载一个类的过程/原理

## 7.设计模式相关

了解两三个设计模式

单例模式

## 8.Linux相关

#### 经常使用的命令（查看内存使用情况，ps，nohup）

#### 进程间通信有哪些

#### Grep命令（不包含某某，查出来一个多少行）

#### Scp命令

## 9.网络通信相关

#### 正向代理和反向代理了解

#### 负载均衡的机制

#### 网络协议

## 10.零碎知识点

empty(‘0’) 返回true

正则表达式