1.对框架进行评价,使用感受和经验

syslog

CREATE TABLE pos\_layout (

id int NOT NULL auto\_increment COMMENT '',

create\_id int NOT NULL default 0 COMMENT '',

created datetime NOT NULL default '2000-01-01' COMMENT '',

update\_id int NOT NULL default 0 COMMENT '',

updated datetime NOT NULL default '2000-01-01' COMMENT '',

cid int NOT NULL default 0 COMMENT '页面ID',

atype smallint NOT NULL default 0 COMMENT '类型',

aname varchar(60) NOT NULL default '' COMMENT '名称',

data varchar(6000) NOT NULL default '' COMMENT 'json参数',

memo varchar(1000) NOT NULL default '' COMMENT '说明',

status tinyint NOT NULL default 0 COMMENT '',

PRIMARY KEY (id)

) DEFAULT CHARSET=utf8 COLLATE=utf8\_general\_ci COMMENT='资源位置定义' AUTO\_INCREMENT=10000 ;

2.数据库高并发,行级锁和表锁

表锁:会锁定整张表.一个用户在对表进行写操作(插入,删除,更新等)前,需要先获得写锁,这会阻塞其他用户对该表的所有读写操作,只有没有写锁时,其他读取的用户才能获得读锁,读锁之间是不相互阻塞的.

行级锁:行级锁可以最大程度地支持并发处理(同时也带来了最大的锁开销)

4.查询优化,主从备份

5.搜索,排序

6.COOKIE、SESSION的联系和区别，多台web服务器如何共享SESSION？

允许服务器跟踪同一个客户端做出的连续请求。

web是通过http协议来实现的，由于HTTP协议是无状态的协议,所以服务器要记录用户状态时,需要用某种机制来识别具体用户,这种机制就是session,session保存在服务端,有一个唯一标识,在服务端保存session的方法很多,内存,数据库,文件都有.

cookie会根据从服务端发送的响应报文内的一个叫做Set-cookie的首部字段信息,通知客户端保存Cookie.当下次客户端再往该服务器发送请求时,客户端会自动在请求报文中加入Cookie值后发送出去.大多数应用都是用Cookie来实现Session跟踪的.如果客户端禁用了cookie,一般这种情况下,会使用一种叫做URL重写的技术来进行会话跟踪,即每次HTTP交互,URL后面都会被附上一个诸如sid=xxx这样的参数,服务端据此识别用户.

cookie可自动填写用户名

cookie的操作：

setcookie($name, $value. $expire, $path, $domain, $secure)

读取：$\_COOKIE

删除：setcookie($name, ‘’, time()-1000)

session的操作：

session\_start();

读取：$\_SESSION;

销毁：$\_SESSION=[]/session\_destory()

7.HTTP协议中的POST和GET有何区别？

|  | GET | POST |
| --- | --- | --- |
| 后退按钮/刷新 | 无害(同一个请求方法执行多次和仅执行一次的效果完全相同) | 数据会被重新提交 |
| 书签 | 可收藏为书签 | 不可收藏为书签 |
| 缓存 | 能被缓存 | 不能被缓存 |
| 编码类型 | application/x-www-form-urlenchded | application/x-www-form-urlencoded或multipart/form-data.为二进制数据使用多重编码 |
| 历史 | 参数保留在浏览器历史中 | 参数不会保留在浏览器历史中 |
| 对数据长度的限制 | 当发送数据时,get方法向url添加数据,url长度受限制(根据浏览器或服务器的配置参数) | 无限制(受服务器配置或内存大小) |
| 对数据类型的限制 | 只允许ascii字符 | 没有限制 |
| 安全性 | 较差,因为所发送的数据是url的一部分 | 较安全,因为参数不会存在浏览器历史或web服务器日志中 |
| 可见性 | 数据在url中对所有人都是可见的 | 数据不会显示在url中 |

8.reqiure的include都可包含文件，二者的区别何在？require与require\_once的区别?

require和include几乎完全一样，包含并运行指定文件，除了处理失败的方式不同之外。require在出错时将导致脚本中止，而include只产生警告，脚本会继续运行。

9.php中WEB上传文件的原理是什么，如何限制上传文件的大小？ is\_uploaded\_file(string $filename):如果filename所给出的文件是通过HTTP POST上传的则返回true.

move\_uploaded\_file(string $filename,string $destination):本函数检查并确保由filename指定的文件是合法的上传文件(即通过http post上传机制),将上传的文件移动到新位置.

10.写一个函数，可以遍历文件夹下的所有文件和文件夹.

<?php

function file\_rec($fileName)

{

if (is\_dir($fileName)) {

if ($dh = opendir($fileName)) {

while (($file = readdir($dh)) !== false) {

if ($file == '.' || $file == '..') {

continue;

}

file\_rec($fileName . '/' . $file);

}

}

} else {

echo "<pre>";

var\_dump($fileName);

echo "</pre>";

}

}

file\_rec('/home/tanglimei/pythontest');

PHP 获取url地址中文件的扩展名

$url = 'http://www.abc.com/ab/cd/e.php?id=1&data=2#laowen';  
echo pathinfo( parse\_url($url)['path'] )['extension'];

11.有mail.log的一个文档，内容为若干邮件地址，其中用'\n'将邮件地址分隔。要求从中挑选出sina.com的邮件地址（包括从文件读取、过滤到列印出来）。

mail.log内容如下：

james@sina.com.cn

jack@163.com

zhansan@sohu.com

lisi@hotmail.com

[wangwu@gmail.com](mailto:wangwu@gmail.com)

<?php

function loadLogByBuf($file, $bufsize)

{

$fHandle = fopen($file, 'rb');

if ($fHandle === false) return;

$count = true;

$end = "";

do {

$buf = fread($fHandle, $bufsize);

if ($buf === false) {

$count = true;

break;

}

$size = strlen($buf);

if ($size < $bufsize) {

$count = false;

}

$lines = explode("\n", $buf);

if (!$lines) {

$lines = [];

}

$lines[0] = $end . $lines[0];

$end = array\_pop($lines);

foreach ($lines as $row) {

yield $row;

}

} while ($count);

fclose($fHandle);

//yield $end;

}

$mails = loadLogByBuf('mail1.log', '1024');

foreach ($mails as $item) {

if (strpos($item, 'sina.com') !== false) {

echo "<pre>";

echo $item;

echo "</pre>";

}

}

12.js中网页前进和后退的代码

前进: javascript:history.forward();=javascript:history.go(1);

后退: javascript:history.back();=javascript:history.go(-1);

js中验证手机号

13.window（A）中用window.open（）打开了window（B），如何从窗口B调用窗口A中的内容？A、B仅仅是窗口的代号，不是窗口名字

window.opener.document.getElementById()

14.了解ajax吗？用过吗？……

15.什么是MVC？用过吗？使用时遇到了什么问题，怎么解决的

MVC是一种设计模式，强制使输入、处理、输出分开，MVC的三个核心部分：M模型，V视图，C控制器。 视图就是用户看到并与之交互的界面。 模型就是程序的数据业务规则。 控制器接收用户的数据调用模型和视图去完成用户需求。 使用MVC的优点：低耦合、高重用性、较低的生命周期成本、快速开发部署、可维护性、可扩展性，有利于软件工程化管理。 MVC的缺点：没有明确的定义，完全理解并不容易。小型项目不适合用MVC。

16.写出 访问 http://www.sina.com.cn 所需的报文头。

get\_headers($url, 1)

17.php的错误日志在哪里可以看到,一些框架是如何将所有的错误日志打到某个指定的文件?

18.名词解释:CGI,fastcgi,php-cgi,php-fpm

CGI(common gateway intrface公共网关接口),用于帮助服务器与语言通信,这里就是nginx和php进行通信,因为nginx和php的语言不通,因此需要一个沟通转换的过程,而CGI就是这个沟通的协议.

nginx服务器在接收到浏览器传递过来的数据后,如果请求的是静态页面或者图片等无需动态处理的则会直接根据请求的url找到其位置然后返回给浏览器,这里无需PHP参与,但是如果是一个动态的页面请求,这个时候nginx就必须与PHP通信,这个时候就会需要用到cgi协议,将请求数据转换成php能理解的信息,然后PHP根据这些信息返回的信息也要通过cgi协议转换成nginx可以理解的信息,最后nginx接到这些信息再返回给浏览器.

FastCGI传统的cgi协议在每次连接请求时,会开启一个进程进行处理,处理完毕会关闭该进程,因此下次连接,又要再次开启一个进程进行处理,因此有多少个连接就有多少个cgi进程,这也是为什么传统的cgi会显得缓慢的原因,过多的进程会消耗资源和内存.而fast-cgi则是一个进程可以处理多个请求.

因为是多进程,所以比cgi多线程消耗更多的服务器内存,

php-cgi是PHP自带的FastCGI管理器,php-cgi变更php.ini配置后需重启php-cgi才能让新的php-ini生效,不可以平滑重启.直接杀死php-cgi进程,php就不能运行了.(php-fpm和Spawn-fcgi就没有这个问题,守护进程会平滑重新生成新的子进程)

php-fpm 是一个PHP FastCGI管理器,是只用于php的,它提供了更好的PHP进程管理方式,可以有效的控制内存和进程,可以平滑重载PHP配置.

19.empty返回为true的情况,”==”与”===”的异同

1. ""（空字符）
2. 0 （作为整数的0）
3. 0.0 （作为浮点数的0）
4. "0" （作为字符串的0）
5. NULL
6. FALSE
7. array()（一个空数组）
8. $var;(一个声明了，但是没有值的变量)

===检查表达式的值与类型，==只检查值

20.如何设置类的自动加载（spl\_autoload\_register()）

21.PHP实现n个无序数组top m(m<n)

22.PHP的序列化和反序列化函数是?何种场景下使用?

serialize(), unserialize()。serialize是将PHP中的变量如对象（object），数组（array）等的值序列化为字符串后存储起来，序列化的字符串可以存储在其他地方，如数据库，session，cookie等，序列化的操作并不会丢失这些值的类型和结构，这样这些变量的数据就可以在PHP页面，甚至是不同PHP程序间传递了，而unserialize()就是吧序列化的字符串转换回PHP的值。

23.PHP的编码转换函数?编码转换的准则或者注意的地方是什么?

mb\_convert\_encoding(),iconv

24.程序代码中修改某些php.ini的配置项

ini\_set()为一个配置选项设置值，在脚本运行时保持新的值，并在脚本结束时恢复

ini\_get()返回配置选项的值

25.PHP在进行数据库交互中如何防止sql注入

例：

$unsafe\_variable = $\_POST['user\_input'];

mysql\_query("INSERT INTO `table` (`column`) VALUES ('$unsafe\_variable')");

为什么会有注入漏洞呢？因为用户可以输入value'); DROP TABLE table;-- 然后查询语句就变成了这样

INSERT INTO `table` (`column`) VALUES('value'); DROP TABLE table;--')

使用预处理语句和参数化查询，预处理语句和参数分别发送到数据库服务器进行解析，参数将会被当作普通字符处理。

1.使用PDO：

$stmt = $pdo->prepare('SELECT \* FROM employees WHERE name = :name');

$stmt->execute(array('name' => $name));

foreach ($stmt as $row) {

// do something with $row

}

2.使用mysqli：

$stmt = $dbConnection->prepare('SELECT \* FROM employees WHERE name = ?');

$stmt->bind\_param('s', $name);

$stmt->execute();

$result = $stmt->get\_result();

while ($row = $result->fetch\_assoc()) {

// do something with $row

}

当你将sql语句发送给数据库服务器进行预处理和解析事发生了什么？通过指定占位符（一个？或者一个上面例子中命名的:name）,告诉数据库引擎你想在哪里进行过滤，当调用execute的时候，预处理语句将会与指定的参数值结合，参数值是和经过解析的sql语句结合到一起，而不是sql字符串，任何发送的参数值都将被当作普通字符串，而不会被数据库服务器解析。

3.addslashes（）使用反斜线引用字符串

26.有大量的ip和地域数据,如何实现类似ip 138的根据某个ip查数据

27.火车票售票系统数据库表设计及程序实现(需要考虑每个用户会购买不同车次的票,同一个车次的票有一定的数量,并发购买时的情况,要求有的功能两个,一个购票接口,一个是查询接口)

28.大数据量高并发,吞吐网站的设计(画示意图配合文字说明)

29.PHP中垃圾回收

##### PHP 5.3之前

采用引用计数的方式管理内存，PHP 所有的变量存在一个叫 zval 的变量容器中，当变量被引用的时候，引用计数会+1，变量引用计数变为0时，PHP 将在内存中销毁这个变量。  
但是引用计数中的循环引用，引用计数不会消减为 0，就会导致内存泄露。  
zval包含变量的类型和值,is\_ref(标识变量是否属于引用集合),refcount  
a: (refcount=1, is\_ref=0)='new string'

##### PHP 5.3之后(回收周期)

1. 并不是每次引用计数减少时都进入回收周期，只有根缓冲区满额后在开始垃圾回收
2. 可以解决循环引用问题
3. 可以总将内存泄露保持在一个阈值以下。

31.MySQL与MongoDB区别

Mysql和MongoDB都是开源的常用数据库，mysql是传统的关系型数据库，MongoDB则是非关系型数据库，也叫文档型数据库，是一种nosql的数据库。

1.关系型数据库-mysql

* 在不同的引擎上有不同的存储方式
* 查询语句是使用传统的sql语句，拥有较为成熟的体系，成熟度很高
* 开源数据库的份额在不断增加，mysql的份额也在持续增长
* 缺点是在海量数据处理时效率会显著变慢

2.非关系型数据库-MongoDB

非关系型数据库（nosql）属于文档型数据库，就是可以存放xml，json，bson等那些数据，

* 存储方式：虚拟内存+持久化
* 查询语句：是独特的MongoDB查询方式
* 适合场景：事件的记录，内容管理或者博客平台
* 架构特点：可以通过副本集，以及分片来实现高可用
* 数据处理：数据是存储在硬盘上的，只不过需要经常读取的数据会被加载到内存中，将数据存储在物理内存中，从而达到高速读写。
* 成熟度与广泛度：新兴数据库，成熟度较低，nosql数据库中最为接近关系型数据库，比较完善的db之一，适用人群不断增长。

MongoDB优势与劣势

优势：  
1、在适量级的内存的MongoDB的性能是非常迅速的，它将热数据存储在物理内存中，使得热数据的读写变得十分快。  
2、MongoDB的高可用和集群架构拥有十分高的扩展性。  
3、在副本集中，当主库遇到问题，无法继续提供服务的时候，副本集将选举一个新的主库继续提供服务。  
4、MongoDB的Bson和JSon格式的数据十分适合文档格式的存储与查询。  
劣势：  
1、 不支持事务操作。MongoDB本身没有自带事务机制，若需要在MongoDB中实现事务机制，需通过一个额外的表，从逻辑上自行实现事务。  
2、 应用经验少，由于NoSQL兴起时间短，应用经验相比关系型数据库较少。  
3、MongoDB占用空间过大。

| 数据库 | MongoDB | MySQL |
| --- | --- | --- |
| 数据库模型 | 非关系型 | 关系型 |
| 存储方式 | 以类JSON的文档的格式存储 | 不同引擎有不同的存储方式 |
| 查询语句 | MongoDB查询方式（类似JavaScript的函数） | SQL语句 |
| 数据处理方式 | 基于内存，将热数据存放在物理内存中，从而达到高速读写 | 不同引擎有自己的特点 |
| 成熟度 | 新兴数据库，成熟度较低 | 成熟度高 |
| 广泛度 | NoSQL数据库中，比较完善且开源，使用人数在不断增长 | 开源数据库，市场份额不断增长 |
| 事务性 | 仅支持单文档事务操作，弱一致性 | 支持事务操作 |
| 占用空间 | 占用空间大 | 占用空间小 |
| join操作 | MongoDB没有join | MySQL支持join |

32.MySQL中索引类型,B-tree,Hash索引怎么实现以及其特点

33.Mysql中什么时候会出现数据丢失,怎样预防数据丢失

34.PHP虚拟机怎样解析PHP代码的

35.Mysql的优化步骤

36.Mysql中索引的作用,如果不考虑其物理空间,索引是不是越多越好

37.Mysql中主从复制原理与搭建

**38.Mysql集群**

**39.socket与curl区别**

**40.php中图像处理**

**41.PHP为什么是弱类型**

一个变量的类型并不是一开始就确定不变，运行中能够才会确定并可能发生隐式或显式的类型转换。

**42.Apache与Nginx区别**

**43.web安全有哪些**

**44.描述从输入url到显示，中间发生了什么？主要阐述apache或者Nginx中发生了什么？**

**45.Hash结构**

**46.中国大概有16亿人口，你的名字的重复率最高，如何快速的找出叫你的名字的人？**

**51.PHP语言对字符串逆序的函数是什么，文件操作的函数名是什么**

**52.TCP的三次握手和四次挥手**

**53.LRU算法**

Least Recently Used最近最少使用，常用于页面置换算法。已经很久没有使用的页面很可能在未来较长的一段时间内不会被用到

# 54.什么是smarty? Smarty的优点是什么?

smarty是一个用php写出来的PHP模板引擎，目的是要使用PHP程序同美工分离，使得程序员改变程序的逻辑内容时不会影响到美工的页面设计，美工的重新修改页面时不会影响到程序的逻辑，这在多人合作的项目中显得尤为重要

优点：

1.速度快：相对其他模板引擎

2.编译型：采用smarty编写的程序在运行时要编译成一个非模板技术的PHP文件

3.缓存技术：它可以将用户最终看到的html文件缓存成一个静态的html页

4.插件技术：smarty可以自定义插件（function，modify，compiler）

不适合使用smarty的地方：

1.需要实时更新的内容（像股票显示）

2.小项目（小项目简单，美工和程序员兼于一人）

**55.魔术方法**

PHP将所有以\_\_（两个下划线）开头的类方法保留为魔术方法

\_\_construct()构造函数，对象创建完成后第一个被对象自动调用的方法，在每个类中都有一个构造方法，通常构造方法被用来执行一些有用的初始化任务，如对成员属性在创建对象时赋予初始值，如果没有显式地声明它，那么类中都会默认存在一个没有参数且内容为空的构造方法。

\_\_destruct()析构函数，与构造方法对应，析构方法允许在销毁一个类之前执行的一些操作或完成一些功能，比如说关闭文件，释放结果集等。

\_\_call()，该方法在调用的方法不存在时会自动调用，程序仍会继续执行下去

\_\_get() 获得一个类的成员变量时调用，通过它可以在对象的外部获取私有成员属性的值。

\_\_set()设置一个类的成员变量时调用，用来设置私有属性

\_\_isset()，当对不可访问属性调用isset()或empty()时调用

\_\_unset()，当对不可访问属性调用unset()时被调用。

\_\_sleep()，执行serialize()时，先会调用这个函数

\_\_clone()，当对象复制完成时调用

\_\_autoload()，尝试加载未定义的类

56.PHP数组底层是什么数据结构？

哈希表

57.hash 的时间复杂度是多少？如何解决冲突的？线性探测后hash的时间复杂度是多少？

Hash冲突是将两个不同的key映射到同一个索引的情况，当出现冲突时，php采用链接法解决冲突，也就是不同的key映射到同一个槽时，这些key将按链表的方式存储

58.事务

一组原子性的SQL查询,或者说一个独立的工作单元.(事务内的语句,要么全部执行成功,要么全部执行失败)

#### ACID:

##### 原子性(atomicity[ˌætəˈmɪsɪti]):

一个事务必须被视为一个不可分割的最小工作单元,整个事务中的所有操作要么全部提交成功,要么全部失败回滚.

##### 一致性(consistency):

数据库总是从一个一致性的状态转换到另一个一致性的状态

##### 隔离性(isolation):

通常来说,一个事务所做的修改在最终提交以前,对其他事务是不可见的.

##### **持久性(durability[ˌdjʊrəˈbɪlətɪ]):**

一旦事务提交,则其所做的修改就会永久保存到数据库中.

隔离级别:  
未提交读: 事务可以读取未提交的数据(脏读)  
提交读: 一个事务从开始直到提交之前,所做的任何修改对其他事务都是不可见的(不可重复读)  
可重复读: 保证了同一事务中多次读取同样记录的结果是一致的  
可串行化: 在读取每一行的数据上都加锁,可能导致大量的超时和锁争用问题

**59.cookie禁用后session是否还能使用？（可以使用url，表单进行提交）**

**60.获取中文字符串的长度，实现截取中文字符串**

**mb\_strlen($str, 'utf-8') mb\_substr($str, 0, 6,** 'utf8'**)**

61.如何设计数据库

(1)为数据库选择合适的存储引擎（2）遵循三范式建立合理的表结构（3）添加适当的索引

62.垂直分割和水平分割依据是什么

63.说出常用的数据库索引，并作相应的解释

64.说出常用的数据库引擎，MyISAM和INNODB的区别

1.MYISAM 不支持事务，innodb支持事务

2.myisam 是表级锁，innodb 是行级锁

3.msIsam 支持全文索引，innodb 不支持全文索引

4.myisam 常用于select较多的数据库，innodb 常用于更新较多的数据库

5.innodb支持外键

6.myisam不支持崩溃后的安全恢复，innodb支持崩溃后的安全恢复

7.innodb通过一些机制和工具支持真正的热备份

8.myisam表存储在两个文件，MYD和MYI，innodb数据存储在共享表空间

65.git常用命令，git的分支操作命令

git 如何回退版本

66.打印出昨天的日期

echo date('Y-m-d H:m:s', strtotime('-1 days'));

67.写出三种常用的三种开源的数据库，写出三种以上的数据库引擎

68.用js写出3种创建image标签的写法

var img = new image()

var img = document.createElementById(‘image’)

image.innerHTML(“<image src=’’>”)

69.解释以下协议：SMTP、POP3、HTTP、FTP、DNS

SMTP(simple mail transfer protocol):简单邮件传送协议 25

POP3：邮局协议 110

HTTP：超文本传输协议 80

FTP：文件传输协议 21  
DNS：域名解析 53

Telnet：远程登录 23

70.如何实现即时通信？给出至少给出两种方案

71.列举PHP的变量类型

四种标量：integer，string，float，boolean

两种复合：array，object

两种特殊：resource，null

72.PHP的常用函数

数组函数：

array\_column($input, $column\_key,[$index\_key])返回数组中指定的一列

array\_shift将数组开头的单元移出数组

array\_slice从数组中取出一段

array\_merge合并一个或多个数组

字符串函数：

substr，trim，strpos，explode，implode，strlen

日期函数：date，strtotime，mktotime

数学函数：max，min，rand，abs

73.变量的引用

不同的名字访问同一个变量的内容

<?php

$a="ABC";

$b =&$a;

echo $a;//这里输出:ABC

echo $b;//这里输出:ABC

$b="EFG";

echo $a;//这里$a的值变为EFG 所以输出EFG

echo $b;//这里输出EFG

?>

74.

server {

listen 80;

server\_name mxtrip.cn;

rewrite ^/(.\*) http://www.mxtrip.cn/$1 permanent;

}

server {

listen 80;

server\_name www.mxtrip.cn;

location ~ /.\*\.xml$ {

rewrite ^/(.\*)\.xml /$1.xml break;

proxy\_pass http://sitemap.corp.mxtrip.cn;

}

set $web\_root /data/deploy/manager2016-web/webroot/www;

index index.php;

error\_log /data/logs/nginx/logs/www.mxtrip.cn.error.log;

rewrite ^/gonglve/(list|list\_\d+).html /youji/$1.html permanent;

#rewrite ^/([^\/]+\.html$) http://www.mxtrip.cn/about/support permanent;

rewrite ^/c.html /i.php last;

include common-fastcgi;

include common-rewrite;

include common-base;

include common-php;

}

server {

listen 80;

server\_name www.api.mxtrip.cn;

set $web\_root /data/deploy/manager2016-web/webroot/www;

index index.php;

error\_log /data/logs/nginx/logs/www.api.mxtrip.cn.error.log;

include common-fastcgi;

include common-rewrite;

include common-base;

include common-php;

}

75.进程与线程的区别

76.打开一个url，发生了什么

77.长连接，短连接http1.0 http1.1

78.解释多态、

79.描述socket通信过程，每个函数

80.i++和++i哪个效率高

81.线程间的同步方式

82.C++多线程编程

83.扑克牌洗牌算法

84.C++内存管理方式

85.列举STL，解释vector，list

86.指针常量，常量指针

87.const修饰函数形参时的作用，const和define的区别

88.tcp协议头常见字段

89.http协议常见状态码

90.访问今日头条的一个链接，用到什么协议

91.sort(),asort(),ksort()有什么分别，在什么情况下使用。

bool sort ( array &$array [, int $sort\_flags = SORT\_REGULAR ] )

对数组排序，$array为要排序的数组，$sort\_flags为排序类型：SORT\_REGULAR 正常比较单元 SORT\_NUMERIC作为数字来比较 SORT\_STRING作为字符串来比较

bool asort ( array &$array [, int $sort\_flags = SORT\_REGULAR ] )

对数组进行排序并保持索引关系，成功时返回true，失败时返回false

bool ksort ( array &$array [, int $sort\_flags = SORT\_REGULAR ] )

对数组按照键名排序，保留键名到数据的关联，主要用于关联数组

92.mysql\_fetch\_row() 和 mysql\_fetch\_array()区别

mysql\_fetch\_row()返回数字数组，$row[0],$row[1]

而mysql\_fetch\_array()函数从结果集中取得一行作为关联数组，或数字数组，或二者兼有。依据第二个参数的值。返回不同结构的数组,$row[0],$row[1],$row[‘username’],$row[‘passwd’]

=========================Redis================================

（1）Redis是什么？

Redis是速度非常快的非关系型（NoSQL）内存键值数据库，可以存储键和五种不同类型的值之间的映射。

五种类型数据为字符串，列表，集合，有序集合，散列表。

redis支持很多特性，例如将内存中的数据持久化到硬盘中，使用复制来扩展读性能，使用分片来扩展写性能。

（2）五种基本类型

| 数据类型 | 可以存储的值 | 操作 |
| --- | --- | --- |
| STRING | 字符串、整数或者浮点数 | 对整个字符串或者字符串的其中一部分执行操作 对整数和浮点数执行自增或者自减操作 |
| LIST | 列表 | 从两端压入或者弹出元素 读取单个或者多个元素 进行修剪，只保留一个范围内的元素 |
| SET | 无序集合 | 添加、获取、移除单个元素 检查一个元素是否存在于集合中 计算交集、并集、差集 从集合里面随机获取元素 |
| HASH | 包含键值对的无序散列表 | 添加、获取、移除单个键值对 获取所有键值对 检查某个键是否存在 |
| ZSET | 有序集合 | 添加、获取、删除元素 根据分值范围或者成员来获取元素 计算一个键的排名 |

## 1. STRING

> set hello world

OK

> get hello

"world"

> del hello

(integer) 1

> get hello

(nil)

## 2. LIST

> rpush list-key item

(integer) 1

> rpush list-key item2

(integer) 2

> rpush list-key item

(integer) 3

> lrange list-key 0 -1

1) "item"

2) "item2"

3) "item"

> lindex list-key 1

"item2"

> lpop list-key

"item"

> lrange list-key 0 -1

1) "item2"

2) "item"

## 3. SET

> sadd set-key item

(integer) 1

> sadd set-key item2

(integer) 1

> sadd set-key item3

(integer) 1

> sadd set-key item

(integer) 0

> smembers set-key

1) "item"

2) "item2"

3) "item3"

> sismember set-key item4

(integer) 0

> sismember set-key item

(integer) 1

> srem set-key item2

(integer) 1

> srem set-key item2

(integer) 0

> smembers set-key

1) "item"

2) "item3"

## 4. HASH

> hset hash-key sub-key1 value1

(integer) 1

> hset hash-key sub-key2 value2

(integer) 1

> hset hash-key sub-key1 value1

(integer) 0

> hgetall hash-key

1) "sub-key1"

2) "value1"

3) "sub-key2"

4) "value2"

> hdel hash-key sub-key2

(integer) 1

> hdel hash-key sub-key2

(integer) 0

> hget hash-key sub-key1

"value1"

> hgetall hash-key

1) "sub-key1"

2) "value1"

## 5. ZSET

> zadd zset-key 728 member1

(integer) 1

> zadd zset-key 982 member0

(integer) 1

> zadd zset-key 982 member0

(integer) 0

> zrange zset-key 0 -1 withscores

1) "member1"

2) "728"

3) "member0"

4) "982"

> zrangebyscore zset-key 0 800 withscores

1) "member1"

2) "728"

> zrem zset-key member1

(integer) 1

> zrem zset-key member1

(integer) 0

> zrange zset-key 0 -1 withscores

1) "member0"

2) "982"

（3）键的过期时间

Redis可以为每个键设置过期时间，当键过期时，会自动删除该键。过期时间对于清理缓存数据非常有用。

（4）持久化

Redis是内存型数据库，为了保证数据在断电后不会丢失，需要将内存中的数据持久化到硬盘上.

1.快照持久化

将某个时间点的所有数据都存放到硬盘上。

可以将快照复制到其他服务器从而创建具有相同数据的服务器副本。

如果系统发生故障，将会丢失最后一次创建快照之后的数据。

如果数据量很大，保存快照的时间会很长。

2.AOF持久化

将写命令添加到AOF（Append Only File）文件的末尾。

对硬盘的文件进行写入时，写入的内容首先会被存储到缓冲区，然后由操作系统决定什么时候将该内容同步到硬盘，用户可以调用file.flush()方法请求操作系统尽快将缓冲区存储的数据同步到硬盘。

（5）复制

通过使用slaveof host port命令来让一个服务器成为另一个服务器的从服务器。一个从服务器只能有一个主服务器，并且不支持主主复制。

从服务器连接主服务器的过程：

主服务器创建快照文件，发送给从服务器，并在发送期间使用缓冲区记录执行的写命令。快照文件发送完毕之后，开始向从服务器发送存储在缓冲区的写命令。

从服务器丢弃所有旧数据，载入主服务器发来的快照文件，之后从服务器开始接受主服务器发来的写命令。

主服务器每执行一次写命令，就向从服务器发送相同的写命令。

（6）reids和memcache比较,使用方法,使用场景

两者都是非关系型内存键值数据库

|  | memcached | redis |
| --- | --- | --- |
| 数据类型 | 仅支持字符串类型 | 支持五种不同种类的数据类型 |
| 数据持久化 | 不支持持久化 | 支持RDB快照和AOF日志 |
| 分布式 | 不支持分布式，只能通过在客户端使用像一致性哈希这样的分布式算法来实现分布式存储，这种方式在存储和查询时都需要先在客户端计算一次数据所在的节点 | redis Cluster实现分布式的支持 |
| 内存管理机制 | 全部存在内存中，数据不能超过内存大小 | 将一些很久没用的value交换到磁盘 |

（7）Redis适用场景

缓存：将热点数据放到内存中

消息队列：List类型是双向链表。和适合用于消息队列。

计数器：Redis这种内存数据库能支持计数器频繁的读写操作。

好友关系：使用set类型的交集操作很容易就可以知道两个用户的共同好友。

（8）数据淘汰策略

可以设置内存的最大使用量。当内存使用量超过时施行淘汰策略

从已设置过期时间的数据集中挑选最近最少使用的数据淘汰，将要过期，任意选择数据，

从所有数据集中挑选最近最少使用的数据淘汰，任意选择数据，禁止驱逐数据

=============MySQL=============

INT(3):012 INT(5):00012

char适合存储很短的字符串，或者所有值都接近同一个长度

对于经常变更的数据，char比varchar更好，char不容易产生碎片。对于非常短的列，char比varchar在存储空间上更有效率

尽量使用TIMESTAMP，比DATETIME空间效率高

mysql -u -p -h -P

存储引擎：InnoDB，MyISAM，NDB，Memory，Archive，Blackhole，CSV，

InnoDB存储引擎

InnoDB默认事务型引擎，最重要，最广泛的存储引擎，性能非常优秀。数据存储在共享表空间，可以通过配置分开。对主键查询的性能高于其他类型的存储引擎。内部做了很多优化，从磁盘读取数据时自动在内存构建hash索引，插入数据时自动构建插入缓冲区

存储引擎（默认的存储引擎）支持事务。特点是行锁设计，支持外键，并支持非锁定读（即默认读取操作不会产生锁）。

InnoDB通过使用多版本并发控制（MVCC）来获得高并发性，并且实现了SQL标准的4种隔离级别，默认为REPEATABLE（重复读）级别，同时使用一种被称为next-key locking的策略来避免幻读。除此之外，InnoDB存储引擎还提供了插入缓存，二次写，自适应哈希索引，预读等高性能和高可用的功能。

对于表中数据的存储，InnoDB存储引擎采用了聚集的方式，因此每张表的存储都是按主键的顺序进行存放，如果没有显示地在表定义时指定主键，InnoDB存储引擎会为每一行生成一个6字节的rowid，并以此作为主键。

MyISAM存储引擎

5.1版本前，MyIsam是默认的存储引擎，MyISAM存储引擎不支持事务，表锁设计，支持全文索引，另一个与众不同的地方是它的缓冲池只缓存索引文件，而不缓冲数据文件。MyISAM存储引擎表由MYD和MYI组成，MYD用来存放数据文件，MYI用来存放索引文件。select count(\*) 性能好

NDB存储引擎

是一个集群存储引擎，特点是数据全部放在内存中，因此主键查找的速度极快，并且通过添加NDB数据存储节点可以线性地提高数据库性能，是高可用，高性能的集群系统。

NDB存储引擎的连接操作（JOIN）实在MySQL数据库层完成的，而不是在存储引擎层完成的，这意味着，复杂的连接操作需要巨大的网络开销，因此查询速度很慢。

事务

事务会把数据库从一种一致状态转换为另一种一致状态。在数据库提交工作时，可以确保要么所有修改都已经保存了。要么所有修改都不保存。（a：原子性c：一致性i：隔离性d：持久性）

索引

存储引擎先去索引当中找到对应的值，然后根据匹配的索引找到对应的数据行。

大大减少服务器需要扫描的数据量。帮助服务器避免排序和临时表。将随机I/O变顺序I/O。大大提高查询速度，降低写的速度，占用磁盘。

对于非常小的表，大部分情况下全表扫描效率更高

中到大型表，索引非常有效

特大型的表，建立和使用索引的代价将随之增长，可以使用分区技术来解决

Mysql各种索引区别：

普通索引：最基本的索引，没有任何限制

唯一索引：与普通索引类似，不同是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

（不常用）全文索引：仅可用于MyISAM表，并且只能对英文进行全文检索，针对较大的数据，生成全文索引很耗时耗空间

组合索引：将多个列组合在一起创建索引，可以覆盖多个列，遵循“最左前缀”原则。

（不常用）外键索引：只有Innodb类型的表才可以使用外键索引，保证数据的一致性，完整性和实现级联操作

主键索引与唯一索引的区别：

一个表只能有一个主键索引，可以有多个唯一索引

主键索引一定是唯一索引，唯一索引不是主键索引

主键可以与外键构成参照完整性约束，防止数据不一致

索引创建原则：

1.最适合索引的列是出现在where子句中的列，或连接子句中的列而不是出现在select关键字后的列

2.索引列的基数越大，索引的效果越好

3.对字符串进行索引，应该制定一个前缀长度，可以节省大量的索引空间

4.根据情况创建复合索引，复合索引可以提高查询效率

5.避免创建过多索引，索引会额外占用磁盘空间，降低写操作效率

6.主键尽可能选择较短的数据类型，可以有效减少索引的磁盘占用，提高查询效率

复合索引（前缀原则）

KEY(a,b,c)

1.生效：where a = 1 and b = 2 and c = 3

where a = 1 and b = 2

where a= 1

不生效：where b = 2 and c = 3（跳过a）

where a=1 and c = 3（跳过b）

2.like查询，%不能在前，可以使用全文索引

3.column is null 可以使用索引

4.如果mysql估计使用索引比全表扫描更慢，会放弃使用索引

5.如果or前的条件中的列有索引，后面的没有，索引都不会被用到

6.列类型是字符串，查询时一定要给值家引号，否则索引失效

B-索引

CREATE TABLE People(

last\_name varchar(50) not null,

first\_name varchar(50) not null,

dob date not null,

gender enum(’m’,’f’)not null,

key(last\_name,first\_name,dob)

);

全值匹配：和索引中所有列进行匹配，例查找姓名为Cuba Allen，出生于1960-01-01的人

匹配最左前缀：例查找所有姓为Allen的人，即只使用索引的第一列

匹配列前缀：可以只匹配某一列的值的开头部分。例查找所有以J开头的姓的人，这里也只使用了索引的第一列。

匹配范围值：例可用于查找姓在Allen和Barrymore之间的人。这里也只使用了索引的第一列。

精确匹配某一列并范围匹配另外一列：例可用于查找所有姓为Allen并且名字是字母K开头（Kim，Karl）的人，即第一列last\_name全匹配，第二列first\_name范围匹配。

只访问索引的查询：即查询只需要访问索引，而无须访问数据行

B-索引的限制：

如果不是按照索引的最左列开始查找，则无法使用索引。例无法用于查找名字为Bill的人。也无法查找某个特定生日的人，因为这两列都不是最左数据列。类似地，也无法查找姓氏以某个字母结尾的人。

不能跳过索引中的列，例无法用于查找姓为Smith并且在某个特定日期出生的人。如果不指定名（first\_name），则MySQL只能使用索引的第一列。

如果查询中有某个列的范围查询，则其右边所有列都无法使用索引优化查找。例有查询where last\_name=’Smith’and first\_name like ‘J%’and dob = ‘1976-12-23’这个查询只能使用索引的前两列，因为这里like是一个范围条件。

哈希索引：

哈希索引基于哈希表实现，只有精确匹配索引所有列的查询才有效。对于每一行数据，存储引擎都会对所有的索引列计算一个哈希码，哈希码是一个较小的值，并且不同键值的行计算出来的哈希码也不一样。哈希索引将所有的哈希码存储在索引中，同时在哈希表中保存指向每个数据行的指针。

优化

1.慢查询日志(pt-query-digest)

2.set profiling=1; show profiles; show profile for query 2;

EXPLAIN

只能解释select查询，并不会对存储程序调用和insert，update，delete或其他语句做解释。

id列：这一列总是包含一个编号，标识select所属的行，如果在语句中没有子查询或联合，那么只会有唯一的select，于是每一行在这个列都将显示一个1.

select\_type列：显示了对应行是简单还是复杂select，simple值意味着查询不包括子查询和UNION，如果查询有任何复杂的子部分，则最外层部分标记为PRIMARY，其他部分标记如下：

SUBQUERY：包含在select列表中的子查询中的select（换句话说，不在from子句中）标记为SUBQUERY

DERIVED：表示包含在from子句的子查询中的select，mysql会递归执行并将结果放到一个临时表中。

UNION：在UNION中的第二个和随后的select被标记为union。

UNION RESULT：用来从UNION的匿名临时表检索结果的select被标记为UNION RESULT。

table列：显示了对应行正在访问哪个表

type列：访问类型（mysql决定如何查找表中的行，从最差到最优）

all：全表扫描

index：跟全表扫描一样，只是mysql扫描表时按索引次序进行而不是行

range：范围扫描就是一个有限制的索引扫描，它开始于索引里的某一点，返回匹配这个值域的行。（between， >）

ref：这是一种索引访问，它返回所有匹配某个单个值的行。

eq\_ref：mysql知道最多只返回一条符合条件的记录，这种访问方法可以在mysql使用主键或者唯一性索引查找时看到，它会将它们与某个参考值做比较。

const，system：当mysql能对查询的某部分进行优化并将其转换成一个常量时。

null：mysql能在优化阶段分解查询语句，在执行阶段甚至用不着再访问表或者索引。例从一个索引列里选取最小值可以通过单独查找索引来完成，不需要在执行时访问表。

possible\_keys列：显示了查询可以使用哪些索引

key列：显示了mysql决定采用哪个索引来优化对该表的访问。

key\_len：显示了mysql在索引里使用的字节数

ref列：

rows列：为了找到所需的行而要读取的行数

复制

复制解决的基本问题是让一台服务器的数据与其他服务器保持同步。一台主库的数据可以同步到多台备库上，备库本身也可以被配置成另外一台服务器的主库。

复制的工作原理：

1.主服务器（master）把数据更改记录到二进制日志（binlog）中。

2.从服务器（slave）把主服务器的二进制日志复制到自己的中继日志（relay log）中。首先，备库会启动一个工作线程。称为I/O线程，I/O线程跟主库建立一个普通的客户端连接，然后在主库上启动一个特殊的二进制转储线程，这个二进制转储线程会读取主库上二进制日志中的事件，备库I/O线程会将接收到的事件记录到中继日志中。

3.从服务器重做中继日志中的日志，把更改应用到自己的数据库上，以达到数据的最终一致性。

Mysql的安全性

1.使用预处理语句防SQL注入

delete from user where id = 1 or 1=1

delete from user where id = ?;

2.写入数据库的数据要进行特殊字符的转义

3.查询错误信息不要返回给用户，将错误记录到日志

其他安全设置：定期做数据备份。不给查询用户root权限，合理分配权限。关闭远程访问数据库权限。修改root口令，不用默认口令，使用较复杂的口令。删除多余的用户。改变root用户的名称。限制一般用户浏览其他库。限制用户对数据文件的访问权限

MySQL数据库优化

====================排序算法========================================

1.冒泡（时间复杂度：最优O(n)最坏O(n2)平均(n2)，空间复杂度O(1)，稳定排序算法）

for (int i = 0; i < len - 1; i++)

for(int j = i + 1; j < len; j++){

if (a[i] > a[j]) {

int temp = a[i];

a[i] = a[j];

a[j] = temp;

}

}

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 排序方法 | 时间复杂度（平均） | 时间复杂度（最坏) | 时间复杂度（最好) | 空间复杂度 | 稳定性 |
| 冒泡排序 | *O*(*n*2) | *O*(*n*2) | *O*(*n*) | *O*(1) | 稳定 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

==============HTTP==============

http协议用于客户端和服务端之间的通信。工作原理：客户端发送请求给服务端，创建一个TCP连接，指定端口号，默认80，连接到服务器，服务器监听浏览器请求，一旦监听到客户端请求，分析请求类型后，服务器会向客户端返回状态信息和数据内容。

1.URI和URL

URI（统一资源标识符）：由某个协议表示的资源定位标识符

URL（统一资源定位符）：使用Web浏览器等访问Web页面时需要输入的网页地址。

uri用字符串标识某一互联网资源，而url表示资源的地点（互联网上所处的位置）



使用 http: 或 https: 等协议方案名获取访问资源时要指定协议类型。不

区分字母大小写,最后附一个冒号(:)。

也可使用 data: 或 javascript: 这类指定数据或脚本程序的方案名。

登录信息(认证)

指定用户名和密码作为从服务器端获取资源时必要的登录信息(身份

认证)。此项是可选项。

服务器地址

使用绝对 URI 必须指定待访问的服务器地址。地址可以是类似

hackr.jp 这种 DNS 可解析的名称,或是 192.168.1.1 这类 IPv4 地址

名,还可以是 [0:0:0:0:0:0:0:1] 这样用方括号括起来的 IPv6 地址名。

服务器端口号

指定服务器连接的网络端口号。此项也是可选项,若用户省略则自动

使用默认端口号。

带层次的文件路径

指定服务器上的文件路径来定位特指的资源。这与 UNIX 系统的文件

目录结构相似。

查询字符串

针对已指定的文件路径内的资源,可以使用查询字符串传入任意参

数。此项可选。

片段标识符

使用片段标识符通常可标记出已获取资源中的子资源(文档内的某个

位置)。但在 RFC 中并没有明确规定其使用方法。该项也为可选

项。

2.请求报文：请求方法，请求的uri，协议版本，可选的请求首部字段，内容实体构成

3.响应报文：协议版本，状态码，用以解释状态码的原因短语，可选的响应首部字段，实体主体。

4.http方法

GET(主要用于数据的读取)，POST(向指定资源提交数据，请求处理,可能会创建新的资源，修改现有资源)，PUT，HEAD(服务器只返回头信息)，DELETE，OPTIONS，TRANCE，CONNECT

5.持久连接：只要任意一端没有明确提出断开连接，则保持TCP连接状态

6.管线化：不用等待响应亦可直接发送下一个请求，可做到同时并行发送多个请求

7.响应状态码

200 OK：从客户端发来的请求在服务器端被正常处理了

204 No Content：服务器接收的请求已成功处理，但在返回的响应报文中不含实体的主体部分，一般在只需要从客户端往服务器发送信息，而对客户端不需要发送新信息内容的情况下使用

301 永久性重定向

302 临时性重定向

400 请求报文中存在语法错误

403 Forbidden：对请求资源的访问被服务器拒绝了

404 Not Found：服务器上无法找到请求的资源

500 服务器端在执行请求时发生了错误

503 服务器暂时处于超负载或正在进行停机维护

8.首部字段

通用首部字段：

Cache-Control：控制缓存行为

Connection：1.控制不再转发给代理的首部字段 2.管理持久连接

Date：创建HTTP报文的日期和时间

Transfer-Encoding:规定了传输报文主体时采用的编码方式

请求首部字段：

Accept：通知服务器，用户代理能够处理的媒体类型及媒体类型的相对优先级

Accept-Charset：通知服务器用户代理支持的字符集及字符集的相对优先顺序

Accept-Encoding：用户代理支持的内容编码及内容编码的优先级顺序

Accept-Language：用户代理能够处理的自然语言集

Authorization：用户代理的认证信息

Host：请求的资源所处的互联网主机名和端口号

Referer：请求的原始资源的URI

User-Agent：创建请求的浏览器和用户代理名称等信息传达给服务器

Cookie：服务器接收到的Cookie信息

响应首部字段：

Age：告知客户端，源服务器在多久前创建了响应

Server：当前服务器上安装的http服务器应用程序的信息

Set-Cookie

实体首部字段：

Content-Encoding：告知客户端服务器对实体的主体部分选用的内容编码方式

Content-Language：告知客户端，实体主体使用的自然语言

Content-Type：说明了实体主体内对象的媒体类型

Expires：将资源失效的日期告知客户端

常考：

Content-type：响应的HTTP内容类型，默认TEXT/HTML

Accept:指定客户端能接收的内容，类型

Origin：最初的请求来源于哪，用于post请求

Cookie：http请求发送给服务端的cookie的值

Cache-Control：

User-Agent

Referrer：上一级请求路径

X-Forwarded-For：请求端真实的ip

Access-Control-Allow-Origin：允许用特定的域名进行访问，跨域使用，

Last-Modified：资源最终修改的时间

导航的第一步是通过访问的域名找出其IP地址。DNS查找过程如下：

* 浏览器缓存 –浏览器会缓存DNS记录一段时间。 有趣的是，操作系统没有告诉浏览器储存DNS记录的时间，这样不同浏览器会储存个自固定的一个时间（2分钟到30分钟不等）。
* 系统缓存– 如果在浏览器缓存里没有找到需要的记录，浏览器会做一个系统调用（windows里是gethostbyname）。这样便可获得系统缓存中的记录。
* 路由器缓存– 接着，前面的查询请求发向路由器，它一般会有自己的DNS缓存。
* ISP DNS 缓存– 接下来要check的就是ISP缓存DNS的服务器。在这一般都能找到相应的缓存记录。
* 递归搜索– 你的ISP的DNS服务器从跟域名服务器开始进行递归搜索，从.com顶级域名服务器到Facebook的域名服务器。一般DNS服务器的缓存中会有.com域名服务器中的域名，所以到顶级服务器的匹配过程不是那么必要了。

==========设计模式==================

=============高并发和大流量================

QPS：每秒请求或查询的数量

吞吐量：单位时间内处理的请求数量(通常由QPS与并发数决定)

响应时间：从请求发出到收到响应花费的时间

带宽：计算带宽大小需关注两个指标，峰值流量和页面平均大小

性能测试工具：ab

ab工作原理：创建多个并发访问线程，模拟多个访问者同时对某一url地址进行访问，用来测试apache的负载压力，也可以测试其他web服务器的压力。

ab的使用：模拟并发请求100次 ，总共请求5000次

ab -c 100 -n 5000待测试网站

高并发解决案例

流量优化：防盗链处理(对恶意的请求拒之门外)

前端优化：减少http请求(把css文件进行合并，js文件)

添加异步请求

启用浏览器缓存和文件压缩

CDN加速(选择最近的CDN)

建立独立图片服务器

服务端优化：页面静态化

并发处理

队列处理

数据库优化：数据库缓存

分库分表，分区操作

读写分离

负载均衡

web服务器优化：负载均衡

1.Web资源防盗链

盗链：在自己的页面上展示一些并不在自己服务器上的内容。获得他人服务器上的资源地址，绕过别人的资源展示页面，直接在自己的页面上向最终用户提供此内容

防盗链工作原理：通过referer或者签名，网站可以检测目标网页访问的来源网页，如果是资源文件，则可以跟踪到显示它的网页地址。一旦检测到来源不是本站即进行阻止或者返回指定的页面

2.减少http请求次数

(1)图片地图：我们可以通过使用五个分开的图片，然后每个图片对应一个超链接产生了5个http请求，我们的目标是要减少http请求，将5个图片合并为一张图片，然后以位置信息定位超链接，把http请求减少为一个，可以保证设计的完整性和功能的齐全性。

(2)合并脚本和样式表

使用外部的js和css文件引用的方式，因为这要比直接写在页面中性能要更好一点。

独立的一个js比用多个js文件组成的页面载入要快38%

(3)图片使用Base64编码减少页面请求数

3.浏览器缓存和数据压缩

缓存分类：http缓存模型中，如果请求成功会有三种情况

200 from cache：直接从本地缓存中获取响应，最快速，最省流量，因为根本没有向服务器发送请求

304 Not Modified：协商缓存，浏览器在本地没有命中的情况下，请求头中发送一定的校验数据到服务端，如果服务端数据没有改变，浏览器从本地缓存响应，返回304

200 OK:以上两种缓存全都失败，服务器返回完整响应，没有用到缓存，相对最慢

http缓存机制：

Cache-Control：告知浏览器缓存过期的时间间隔而不是时刻

no-store 禁止浏览器缓存响应

no-cache 不允许直接使用本地缓存，先发起请求和服务器协商

max-age=3600 告知浏览器该响应本地缓存有效的最长期限，以秒为单位

Last-Modified：通知浏览器资源的最后修改时间

适合缓存的内容：不变的图像，如logo，图标等。js，css静态文件，可下载的内容，媒体文件

建议使用协商缓存(经常改变)：html文件，经常替换的图片，经常修改的js，css文件(可以加入文件的签名来拒绝缓存，index.css?签名)

不建议缓存的内容：用户隐私等敏感数据，经常改变的api数据接口

===========PHP==============

PHP7.0新特性

1.变量类型

参数和返回值增加了类型限定，增加类型后PHP JIT可以准确判断变量类型，生成最佳的机器指令

function test(int $a, string $b, array $c) : int {

//code

}

2.错误异常

PHP程序出错后过去Zend引擎会发生致命错误并终止程序运行，PHP7可以使用try/catch捕获错误

try {

non\_exists\_func();

} catch (EngineException $e ) {

echo “Exception :{$e->getMessage()}\n”;

}

3.zval使用栈内存（节省了大量内存分配和内存管理等操作）

在Zend引擎和扩展中，经常要创建一个PHP的变量，底层就是一个zval指针。之前的版本都是通过MAKE\_STD\_ZVAL动态的从堆上分配一个zval内存。而PHP7可以直接使用栈内存

4.zend\_string存储hash值，array查询不再需要重复计算hash

5.hashtable桶内直接存数据，减少了内存申请次数，提升了Cache命中率和内存访问速度

=================XSS================

反射型：发出请求时，XSS代码出现在URL中，作为输入提交到服务器端，服务器端解析后响应，XSS代码随响应内容一起传回给浏览器，最后浏览器解析执行XSS代码。这个过程像一次反射，故叫反射型XSS。

存储型：存储型XSS和反射型XSS的差别仅在于，提交的代码会存储在服务器端（数据库，内存，文件系统等），下次请求目标页面时不用再提交xss代码

xss防御措施：

1.过滤

移除用户上传的dom属性，如onerror

移除用户上传的style节点，script节点，iframe节点

2.校正

避免直接对html entity解码

使用dom parse转换，校正不配对的dom标签

跨站脚本攻击，dns劫持