后端面试总结

JS 中网页前进和后退的代码

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

后退：`history.back();=history.go(-1);`

PHP 中 WEB 上传文件的原理，如何限制上传文件的大小?

`$_FILES["file"]["tmp_name"]` `is_uploaded_file()` 和 `move_uploaded_file()` `php.ini: upload_max_filesize` `php.ini: memory_limit` `php.ini: post_max_size`

多台 web 服务器如何共享 SESSION

- 专门的一个数据库服务器用来存储 `session`
- 使用服务器脚本进行服务器之间 `session` 同步
- 使用 `cookie` 存储在客户端
- 使用 `cache server` 比如 `memcache`

禁用 COOKIE 后 SEESION

> *关于 session ID 的存取*

PHP 获取远程文件

获取：

- `curl`
- `readfile` 读一个文件到缓存中
- `ob_get_contents` 读缓存
- `file_get_contents` 保存到本地：

```
    $th = fopen($filepath, "w");
    fwrite($th, $file);
    fclose($fh);
```

POST 和 GET 有何区别

GET 传输时需要进行 url 编码

POST 和 GET 的最大容量

2MB / 1024B

三个数最大值

```
function($a, $b, $c){
    return $a > $b ? ($a > $c ? $a : $c) : ($b > $c ? $b : $c);
}
```

三范式

存储过程

数据库分区

- 范围分区

数据库分表

- 按日期分表
- …

数据库查询优化

- 避免会导致全表扫描的操作（null、%、判断…）
- 提前计算好表达式
- 适当合理的索引（考虑在 `where` 及 `order by` 涉及的列上建立索引）
- 不要返回不必要的行和列（避免返回不必要的数据）
- 避免大事务操作，提高系统并发能力
- 避免在 `where` 子句中使用 `or` 来连接条件，可使用 `union`，否则将导致引擎放弃使用索引而进行全表扫描
- 分表，分区

索引

- 普通索引
- 唯一索引
- 主键索引

```
ALTER TABLE tb_name ADD INDEX index_name (column_list)
```

B+树，B 树，B-树

单链表反转

> *存储当前节点（第一个）的 next，当前节点（第一个）指向第三个（第二个的 next），第二个指向（表头的 next）变为链表首，表头指向第二个（之前存储的当前节点（第一个）的 next）`current`，`nextNode`，`third`*

```
current = L->next;
while(current) {
    tmp = current->next;
    current->next = nextNode->next;
    nextNode->next = L->next;
    L->next = tmp;
}
```

两数交换

```
void swap(int &a, int &b)
{
    tmp = a ^ b; // a = a ^ b;
    a = a ^ tmp;
    b = b ^ tmp;
```

```
}
void swap(int &a, int &b)
{
    a ^= b;
    b ^= a;
    a ^= b;
}
```

---

8 个小球，其中一个比其它 7 个都重，其它 7 个一样重。给你一个天平，至少要称几次？

> *2 次 从 8 个球中任意取 2 组 3 个球放在天平上称，如果重量相等,那么重的球必然在余下的 2 个球中,故将那个 2 个球放在天平上称即可；如果有一边重,那么重的球必然在这一边的 3 个球里，再从这 3 个球中任意取 2 个出来称，如果一样重,那么重的球就是余下的那个球，如果一边更重,则便是这个球了.*

操作系统的线程与进程的区别

- 一个程序至少有一个进程，一个进程至少有一个线程
- 进程是一个独立的运行单位，也是系统进行资源分配和调度的基本单位
- 线程是操作系统进程中能够并发执行的实体，是处理器调度和分派的基本单位
- 线程高并发，进程并行

给你程序额外的内存块,你会把内存块分配给程序的堆区还是栈区

> *`malloc`、申请内存、内存块，都是分配到 `堆`*

---

include、require

- `incluce` 在用到时加载
- `require` 在一开始就加载
- `include` 的文件中出错了，主程序继续往下执行；`require` 的文件出错了，主程序也停止
- `require()` 和 `include()` 语句是语言结构，不是真正的函数，可以像 php 中其他的语言结构一样，例如 `echo()` 可以使用 `echo("ab")` 形式，也可以使用 `echo "abc"` 形式输出字符串 abc。`require()` 和 `include()` 语句也可以不加圆括号而直接加参数。

---

error_reporting(2047)

> *相当于 `error_reporting('E_ALL');`*

JavaScript 子窗口调用父窗口

> *`window.opener` 是 `window.open` 打开的子页面调用父页面对象*

客户端 IP 与服务器 IP

- 客户端 `getenv('REMOTE_ADDR');`

- 服务器端

`getenv('SERVER_ADDR');`
`gethostbyname("www.baidu.com")`

对多个数组或多维数组进行排序

> *array_multisort*

类型转换

> *非数字字符串 -> 整型 0 数字字符串 -> 整型 数字*

打开 php.ini 中的 Safe_mode，会影响哪些参数

> *与文件、系统操作有关的函数，如：*

`ckdir,`
`move_uploaded_file,chgrp,parse_ini_file,chown,rmdir,copy,rename,fopenrequire,highlight_file,show_source,include,symlink,link,touch,mkdir,unlink`
`pathinfo,basename,fopen,system,exec,proc_open`

一些操作

`mkdir` 创建目录 `touch` 创建文件夹 修改权限 `chmod` 之类

大流量网站

1. pHp 缓存
2. 生成 html 静态页面
3. 使用主辅数据库，把数据库的读写分开
4. 使用负载均衡器和多台服务器

首先，确认服务器硬件是否足够支持当前的流量 其次，优化数据库访问。 第三，禁止外部的盗链。 第四，控制大文件的下载。 第五，使用不同主机分流主要流量 第六，使用流量分析统计软件

Apache

以 Apache 模块的方式安装 pHp，在文件 http.conf 中首先要用语句动态装载 pHp 模块， 然后再用语句使得 Apache 把所有扩展名为 php 的文件都作为 pHp 脚本处理。

> *LoadModule php5_module "c:/php/php5apache2.dll"*
> *AddType application/x-httpd-php-source .phps*
> *AddType application/x-httpd-php .php .php5 .php4 .php3 .phtml*

序列化

> *类的属性可以序列化后保存到 `session` 中，从而以后可以恢复整个类，*
> *`serialize()` /`unserialize()`*

函数的参数不能是对变量的引用

一个函数的参数不能是对变量的引用，除非在 php.ini 中
把 `allow_call_time_pass_reference` 设为 `on`

PHP 是什么

`Hypertext Preprocessor`，是一种用来开发动态网站的服务器脚本语言

PHP 不支持多继承

> *PHP 类只能继承一个父类，并用关键字 `extended` 标识*

PHP 获取图像尺寸

`getimagesize ()` 获取图片的尺寸 `Imagesx ()` 获取图片的宽度 `Imagesy ()` 获取图片的高度

PEAR

> *PHP 扩展与应用库(PHP Extension and Application Repository)，它是一个 PHP 扩展及应用的一个代码仓库*

魔术方法

`__construct()`， `__destruct()`， `__call()`， `__callStatic()`， `__get()`， `__set()`， `__isset()`， `__unset()` `__sleep()`， `__wakeup()`， `__toString()`， `__invoke()`， `__set_state()`， `__clone()`， `__debugInfo()`

__sleep

`serialize()` 函数会检查类中是否存在一个魔术方法 `__sleep()`。如果存在，该方法会先被调用，然后才执行序列化操作。此功能可以用于清理对象，并返回一个包含对象中所有应被序列化的变量名称的数组。

__wakeup

`unserialize()` 会检查是否存在一个 `__wakeup()` 方法。如果存在，则会先调用 `__wakeup` 方法，预先准备对象需要的资源。

魔术变量

`__LINE__`，`__FILE__`，`__DIR__`，`__FUNCTION__` `__CLASS__`，`__TRAIT__`，`__METHOD__`，`__NAMESPACE__`

表单提交

`<a href="javascript: document.myform.submit();">Submit Me</a>`

获取居中的位置

`mc.x = stage.StageWidth / 2` `mc.y = stage.StageHeight / 2`

OOP

`Object OrientedProgramming`

事件机制

1. 捕获阶段（即由根节点流向子节点,检测每个节点是否注册了监听器）
2. 目标阶段（激发在目标对象本身注册的监听程序）
3. 冒泡阶段（从目标节点到根节点,检测每个节点是否注册了监听器）当中涉及了两个属性 `target` 和 `currentTarget`，`target` 是事件的派发者，

`currentTarget` 是正在检测的对象，当开始了事件流，事件的 `currentTarget` 属性处于不断变化中

PHP 打印出前一天的时间

`echo date("Y-m-d H:i:s", time()-24*3600);`

修改 SESSION 的生存时间

1. `php.ini` 中 `session.gc_maxlifetime` （默认为 1440）
2. PHP

```
$savepath = "./session_save_dir/";
$lifeTime = 24 * 3600;
session_save_path($savepath);
session_set_cookie_params($lifeTime);
session_start();
```

1. `setcookie()` 或 `session_set_cookie_params($lifeTime)`

多个进程同时写入一个文件成功

```
function processWriteInFile($path, $mode, $data){
    fopen($path, $mode);
    $times = 0;
    while (true) {
        if ($times > 30) {
            echo "file locked";
            return 0;
        }
        if (flock($path, LOCK_EX)) {
            fwrite($path, $data);
            break;
        } else {
            $times++;
            usleep(1000);
        }
    }
    flock($path, LOCK_UN);
    fclose($path);
}
```

代理模式

```
class A{
    public function printer()
    {
        print_r("I'm print!");
    }
}class B{
```

```
    private $print;
    function __construct()
    {
        $this->print = new A();
    }
    function __call($method, $args)
    {
        if (!method_exists($this->print, $method)) {
            exit("the method didn't exist");
        }
        $this->print->$method();
    }
}$kof = new B();$kof->printer();
```

执行效率分析

- xdebug 统计函数执行次数和具体时间进行分析，配合 winCacheGrind 分析
- explain(mysql)，启用 slow query log 记录慢查询

---

mysql

```
CREATE TABLE test(
    id int not null primary key,
    name char(20) not null default ""
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

---

linux 静态库 动态库

> *静态库文件 .a 动态库文件 .so*

---

vi / vim 3 种模式

- 命令模式 esc
- 插入模式 i
- 命令行模式 esc + :

---

vi / vim 删除行

`1,10d`

Linux 下网络配置文件目录

`/etc`

Linux 下权限修改

修改/usr/local 目录下的文件 file 的权限为 755，修改/home/th1 下的所有文件的所有者和组为 th1 和 ggv？ `chmod 755 /usr/local/file` `chown th1:ggv /home/th1/*`

常用的字符串操作

substr(),strtolower(),ucwords(),ucfirst(),strtoupper()implode(),explode(),str_replace(),strpos(),strrev()

PHP 缓存机制

- APC opcode 缓存
- memcache
- redis

TDD

测试驱动开发的基本过程如下：

1） 明确当前要完成的功能。可以记录成一个 TODO 列表。

2） 快速完成针对此功能的测试用例编写。

3） 测试代码编译不通过。

4） 编写对应的功能代码。

5） 测试通过。

6） 对代码进行重构，并保证测试通过。

7） 循环完成所有功能的开发。

原则

（1）测试隔离。不同代码的测试应该相互隔离。对一块代码的测试只考虑此代码的测试，不要考虑其实现细节（比如它使用了其他类的边界条件）。

（2）一顶帽子。开发人员开发过程中要做不同的工作，比如：编写测试代码、开发功能代码、对代码重构等。做不同的事，承担不同的角色。开发人员完成对应的工作时应该保持注意力集中在当前工作上，而不要过多的考虑其他方面的细节，保证头上只有一顶帽子。避免考虑无关细节过多，无谓地增加复杂度。

（3）测试列表。需要测试的功能点很多。应该在任何阶段想添加功能需求问题时，把相关功能点加到测试列表中，然后继续手头工作。然后不断的完成对应的测试用例、功能代码、重构。一是避免疏漏，也避免干扰当前进行的工作。

（4）测试驱动。这个比较核心。完成某个功能，某个类，首先编写测试代码，考虑其如何使用、如何测试。然后在对其进行设计、编码。

（5）先写断言。测试代码编写时，应该首先编写对功能代码的判断用的断言语句，然后编写相应的辅助语句。

（6）可测试性。功能代码设计、开发时应该具有较强的可测试性。其实遵循比较好的设计原则的代码都具备较好的测试性。比如比较高的内聚性，尽量依赖于接口等。

（7）及时重构。无论是功能代码还是测试代码，对结构不合理，重复的代码等情况，在测试通过后，及时进行重构。关于重构，我会另撰文详细分析。

（8）小步前进。软件开发是个复杂性非常高的工作，开发过程中要考虑很多东西，包括代码的正确性、可扩展性、性能等等，很多问题都是因为复杂性太大导致的。极限编程提出了一个非常好的思路就是小步前进。把所有的规模大、复杂性高的工作，分解成小的任务来完成。对于一个类来说，一个功能一个功能的完成，如果太困难就再分解。每个功能的完成就走测试代码－功能代码－测试－重构的循环。通过分解降低整个系统开发的复杂性。这样的效果非常明显。几个小的功能代码完成后，大的功能代码几乎是不用调试就可以通过。一个个类方法的实现，很快就看到整个类很快就完成啦。本来感觉很多特性需要增加，很快就会看到没有几个。你甚至会为这个速度感到震惊。（大幅度减少调试、出错的时间产生的这种速度感）