# 面试题3

### Php-fpm进程模型 如何处理重启，重启过程如何拒绝新请求，包括如何调度进程

https://segmentfault.com/a/1190000019790687

### 实现单例

申请一个私有的静态成员变量来保存该类的唯一实例

声明私有的构造方法，防止类外部创建对象

声明一个静态公共方法，供外部获取唯一实例

class SetSingleton{

private static $new; //申请一个私有的静态成员变量来保存该类的唯一实例

private function \_\_construct() {} //声明私有的构造方法，防止类外部创建对象

public static function instance () { //声明一个静态公共方法，供外部获取唯一实例

if (!(self::$new instanceof self)) {

self::$new = new self;

}

return self::$new;

}

private function \_\_clone() {} //声明私有的克隆方法，防止对象被克隆

public function \_\_sleep() { //重写\_\_sleep方法，将返回置空，防止序列化反序列化获得新的对象

return [];

}

}

### php有哪些模式 装饰器模式

1. 单例模式
2. 工厂模式

通过工厂来实例化类。

interface mysql{

public function connect();

}

class mysqli2 implements mysql{

public function connect(){

echo 'mysqli';

}

}

class pdo2 implements mysql{

public function connect(){

echo 'pdo';

}

}

class mysqlFactory{

public static function factory($class\_name){

return new $class\_name();

}

}

$obj = mysqlFactory::factory('pdo2');

$obj->connect();

1. 注册树模式

将类注册到树上去，需要的时候再取下来。（这个类可以是单例）

class a{}

class tree{

protected static $obj;

public static function set($alias,$obj){

self::$obj[$alias] = $obj;

}

public static function get($alias){

return self::$obj[$alias];

}

public static function \_unset(){

unset(self::$obj[$alias]);

}

}

tree::set('a',new a());

var\_dump(tree::get('a'));

1. 策略模式

定义一系列算法，将每一个算法封装起来，并让它们可以相互替换。策略模式让算法独立于使用它的客户而变化

1. 适配器模式

将各种截然不同的函数接口封装成统一的API。

PHP中的数据库操作有MySQL,MySQLi,PDO三种，可以用适配器模式统一成一致，使不同的数据库操作，统一成一样的API。

1. 观察者模式

观察者模式(Observer)，当一个对象状态发生变化时，依赖它的对象全部会收到通知，并自动更新。观察者模式实现了低耦合，非侵入式的通知与更新机制。

1. 装饰器模式

装饰器模式（Decorator Pattern）允许向一个现有的对象添加新的功能，同时又不改变其结构。这种类型的设计模式属于结构型模式，它是作为现有的类的一个包装。

### 4. 线程，进程、协程 原理及实现

### 5. 单点登录



### yii框架、ci框架、Laravel框架优点和缺点

**一、ThinkPHP**

ThinkPHP(FCS)是一个轻量级的中型框架，是从Java的Struts结构移植过来的中文PHP开发框架。它使用面向对象的开发结构和MVC模式，并且模拟实现了Struts的标签库，各方面都比较人性化，熟悉J2EE的开发人员相对比较容易上手，适合php框架初学者。 ThinkPHP的宗旨是简化开发、提高效率、易于扩展，其在对数据库的支持方面已经包括MySQL、MSSQL、Sqlite、PgSQL、 Oracle，以及PDO的支持。ThinkPHP有着丰富的文档和示例，框架的兼容性较强，但是其功能有限，因此更适合用于中小项目的开发。

优点

1.借助成熟的Java思想  
2.易于上手，有丰富的中文文档；学习成本低，社区活跃度高  
3.框架的兼容性较强，PHP4和PHP5完全兼容、完全支持UTF8等。  
4.适合用于中小项目的开发  
5.从thinkphp3.2.2引入composer包管理工具

缺点

1.对Ajax的支持不是很好；  
2.目录结构混乱，相比其他框架目录结构要差一点；  
3.上手容易，但是深入学习较难。

**二、Yii**

Yii 是一个基于组件的高性能php框架，用于开发大型Web应用。Yii采用严格的OOP编写，并有着完善的库引用以及全面的教程。从 MVC，DAO/ActiveRecord，widgets，caching，等级式RBAC，Web服务，到主题化，I18N和L10N，Yii提供了 今日Web 2.0应用开发所需要的几乎一切功能。事实上，Yii是最有效率的PHP框架之一。

优点

1.纯OOP  
2.用于大规模Web应用  
3.模型使用方便  
4.开发速度快，运行速度也快。性能优异且功能丰富  
5.使用命令行工具。  
6.支持composer包管理工具

缺点：

1.对Model层的指导和考虑较少  
2.文档实例较少  
3.英文太多  
4.要求PHP技术精通，OOP编程要熟练！  
5.View并不是理想view，理想中的view可能只是html代码，不会涉及PHP代码。

**三、laravel**

优点

1. laravel的设计思想是很先进的，非常适合应用各种开发模式TDD, DDD 和BDD
2. blade模板:渲染速度更快
3. artisan:命令行工具,很多手动的工作都自动化
4. 强大的rest router:用简单的回调函数就可以调用,快速绑定controller和router  
   2.支持composer包管理工具  
   3.集合了php 比较新的特性，以及各种各样的设计模式，Ioc 容器，依赖注入等。

缺点

1.基于组件式的框架，所以比较臃肿

### redis和memcached区别

Redis支持字符串，散列，列表，集合，有序集，位图，超级日志和空间索引；而Memcached支持字符串和整数。

Memcached的读写速度高于Redis

Memcached不支持复制。而，Redis支持主从复制，允许从属Redis服务器成为主服务器的精确副本；来自任何Redis服务器的数据都可以复制到任意数量的从属服务器。

Redis是单线程的；而，Memcached是多线程的。

### php垃圾回收原理

引用计数：指在value中增加一个字段 refcount 记录指向当前value的数量，变量复制、函数传参时并不直接硬拷贝一份value数据，而是将 refcount++ ，变量销毁时将refcount-- ，等到 refcount 减为0时表示已经没有变量引用这个value，将它销毁即可。

$a = "time:" . time(); //$a -> zend\_string\_1(refcount=1)

$b = $a; //$a,$b -> zend\_string\_1(refcount=2)

$c = $b; //$a,$b,$c -> zend\_string\_1(refcount=3)

unset($b); //$b = IS\_UNDEF $a,$c -> zend\_string\_1(refcount=2)

字段 refcount在对应类型的结构zend\_refcounted\_h中，比如字符串结构：

struct \_zend\_string {

zend\_refcounted\_h gc;

zend\_ulong h; /\* hash value \*/

size\_t len;

char val[1];

};

引用：通过 & 操作符产生一个引用变量，也就是说不管以前的类型是什么， & 首先会创建一个 zend\_reference 结构，其内嵌了一个zval，这个zval的value指向原来zval的value(如果是布尔、整形、浮点则直接复制原来的值)，然后将原zval的类型修改为IS\_REFERENCE，原zval的value指向新创建的zend\_reference 结构。

struct \_zend\_reference {

zend\_refcounted\_h gc;

zval val;

};

$a = "time:" . time(); //$a -> zend\_string\_1(refcount=1)

$b = &$a; //$a,$b -> zend\_reference\_1(refcount=2) -> zend\_string\_1(refcount=1)

垃圾回收：如果一个变量value的refcount减少到0， 那么此value可以被释放掉，不属于垃圾，如果一个变量value的refcount减少之后大于0，那么此zval还不能被释放，此zval可能成为一个垃圾，通过 zval.u1.type\_flag 记录是否收集，只有包含 IS\_TYPE\_COLLECTABLE 的变量才会被GC收集（数组，对象）。

回收：放入一个缓冲buffer中，等这个buffer满了以后(10000个值)再统一进行处理，加入buffer的是变量zend\_value的 zend\_refcounted\_h，垃圾缓存区是一个双向链表，等到缓存区满了以后则启动垃圾检查过程：遍历缓存区，再对当前变量的所有成员进行遍历，然后把成员的refcount减1(如果成员还包含子成员则也进行递归遍历，其实就是深度优先的遍历)，最后再检查当前变量的引用，如果减为了0则为垃圾。

### trait 协程 反射 依赖注入

Trait 是为类似 PHP 的单继承语言而准备的一种代码复用机制。

<?php

class Base {

public function sayHello() {

echo 'Hello ';

}

}

trait SayWorld {

public function sayHello() {

parent::sayHello();

echo 'World!';

}

}

class MyHelloWorld extends Base {

use SayWorld;

}

$o = new MyHelloWorld();

$o->sayHello();

?>

以上例程会输出：

Hello World!

IOC（inversion of control）控制反转模式；控制反转是将组件间的依赖关系从程序内部提到外部来管理；

依赖注入：一个类的实例化需要依赖另一个类，需要注入依赖的类。

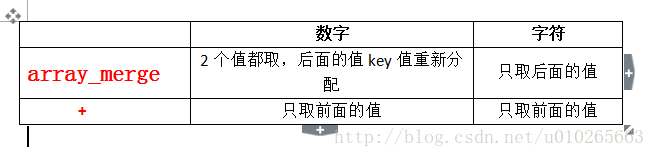
### 如何防止sql注入

使用sql语句预处理

### merge与+区别

array\_merge()是PHP语言中的一个函数，作用是将两个或多个数组的单元合并起来，一个数组中的值附加在前一个数组的后面。

主要区别是两个或者多个数组中如果出现相同键名，键名分为字符串或者数字，需要注意：



### 抽像类与接口区别

抽象类：不能被实例化，如果它里面至少有一个方法是被声明为抽象的，那么这个类就必须被声明为抽象的。被定义为抽象的方法只是声明了其调用方式（参数），不能定义其具体的功能实现。子类必须定义父类中的所有抽象方法。一个类只能继承一个抽象类。

接口：接口不能实例化，一个类可以同时实现多个接口，用逗号隔开。类必须实现接口的所有方法。方法只声明不实现，方法必须是public，

### Opcode

当解释器完成对脚本代码的分析后，便将它们生成可以直接运行的中间代码，也称为操作码（Operate Code，opcode）。Opcode cache的目地是避免重复编译，减少CPU和内存开销。

### php性能调优工具

OneAPM for PHP 是 OneAPM 推出的针对 PHP 语言的专项监控工具，能够监控 PHP 应用整体的响应时间、PHP 应用内部的执行时间以及数据库连接和第三方 API 的性能。并且能够通过精确到代码行级别的性能分析能力，轻松定位性能瓶颈，显著降低性能调优过程中定位问题的时间。

### Leravel job和event区别

Job:（同步\异步）消息队列

Event:允许你在应用中订阅和监听各种发生的事件。

### Static:: 与self::区别

Static:: 会强制使用当前调用对象所属类中的变量

Self:: 调用的是自身类的变量

class a{

static protected $test="class a";

public function static\_test(){

echo static::$test; //class b

echo self::$test; //class a

}

}

class b extends a{

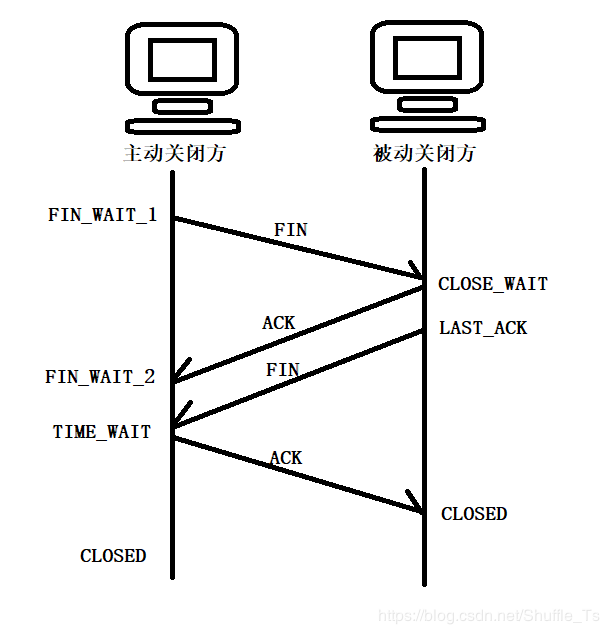
static protected $test="class b";

}

$obj = new b();

### 三次握手 四次挥手 为什么

四次挥手：假设Client端发起中断连接请求，也就是发送FIN报文。Server端接到FIN报文后，意思是说"我Client端没有数据要发给你了"，但是如果你还有数据没有发送完成，则不必急着关闭Socket，可以继续发送数据。所以你先发送ACK，"告诉Client端，你的请求我收到了，但是我还没准备好，请继续你等我的消息"。这个时候Client端就进入FIN\_WAIT状态，继续等待Server端的FIN报文。当Server端确定数据已发送完成，则向Client端发送FIN报文，"告诉Client端，好了，我这边数据发完了，准备好关闭连接了"。Client端收到FIN报文后，"就知道可以关闭连接了，但是他还是不相信网络，怕Server端不知道要关闭，所以发送ACK后进入TIME\_WAIT状态，如果Server端没有收到ACK则可以重传。“，Server端收到ACK后，"就知道可以断开连接了"。Client端等待了2MSL后依然没有收到回复，则证明Server端已正常关闭，那好，我Client端也可以关闭连接了。Ok，TCP连接就这样关闭了！



### http协议状态码

|  |  |  |
| --- | --- | --- |
| 1XX | 100-101 | 信息提示 |
| 2XX | 200-206 | 成功 |
| 3XX | 300-305 | 重定向 |
| 4XX | 400-415 | 客户端错误 |
| 5XX | 500-505 | 服务器错误 |

200 OK 服务器成功处理了请求

304 Not Modified（未修改）客户的缓存资源是最新的， 要客户端使用缓存

403（禁止） 服务器拒绝请求。

404 Not Found 未找到资源

500（服务器内部错误） 服务器遇到错误，无法完成请求。

501 Internal Server Error服务器遇到一个错误，使其无法对请求提供服务

### http请求都有哪些方式及区别

get:使用给定的URI从给定服务器中检索信息，即从指定资源中请求数据

Post:向指定资源提交数据进行处理请求（例如提交表单或者上传文件

head:类似于 GET 请求，只不过返回的响应中没有具体的内容，用于获取报头

put:从客户端向服务器传送的数据取代指定的文档的内容。

delete:请求服务器删除指定的页面

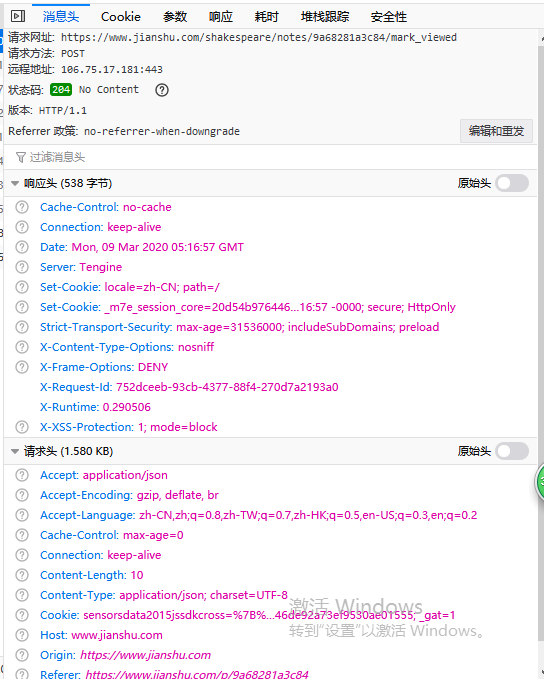
connect:HTTP/1.1 协议中预留给能够将连接改为管道方式的代理服务器。

options:允许客户端查看服务器的性能

trace:回显服务器收到的请求，主要用于测试或诊断。

patch:是对 PUT 方法的补充，用来对已知资源进行局部更新

### http表头



Accept:指定客户端能够接收的内容类型，内容类型中的先后次序表示客户端接收的先后次序。

Accept-Encoding : 指定客户端浏览器可以支持的web服务器返回内容压缩编码类型。表示允许服务器在将输出内容发送到客户端以前进行压缩，以节约带宽。而这里设置的就是客户端浏览器所能够支持的返回压缩格式。

Accept-Language : 指定HTTP客户端浏览器用来展示返回信息所优先选择的语言。

Connection ： 表示是否需要持久连接。如果web服务器端看到这里的值为“Keep-Alive”，或者看到请求使用的是HTTP 1.1（HTTP 1.1默认进行持久连接），它就可以利用持久连接的优点，当页面包含多个元素时（例如Applet，图片），显著地减少下载所需要的时间。要实现这一点， web服务器需要在返回给客户端HTTP头信息中发送一个Content-Length（返回信息正文的长度）头，最简单的实现方法是：先把内容写入ByteArrayOutputStream，然 后在正式写出内容之前计算它的大小。

Connec-Length : 请求头的长度。

Connect-Type : 显示此HTTP请求提交的内容类型。一般只有post提交时才需要设置该属性。

有关Content-Type属性值可以如下两种编码类型：

（1）“application/x-www-form-urlencoded”： 表单数据向服务器提交时所采用的编码类型，默认的缺省值就是“application/x-www-form-urlencoded”。 然而，在向服务器发送大量的文本、包含非ASCII字符的文本或二进制数据时这种编码方式效率很低。

（2）“multipart/form-data”： 在文件上载时，所使用的编码类型应当是“multipart/form-data”，它既可以发送文本数据，也支持二进制数据上载。

当提交为单单数据时，可以使用“application/x-www-form-urlencoded”；当提交的是文件时，就需要使用“multipart/form-data”编码类型。

在Content-Type属性当中还是指定提交内容的charset字符编码。一般不进行设置，它只是告诉web服务器post提交的数据采用的何种字符编码。

cookie : 浏览器端cookie。

Hose : 客户端地址

Origin : 目标地址

Referer : 包含一个URL，用户从该URL代表的页面出发访问当前请求的页面

Cache-control:缓存控制，public( http通信的过程中，包括请求的发起方（浏览器）、代理缓存服务器都可以进行缓存。)。private (只允许请求的发起方（浏览器）进行缓存。)

no-cache(可以在请求的发起方（浏览器）进行缓存，但是每次都需要去服务器进行资源验证。)。no-store(任何地方都不可以缓存。)。User-Agent (客户端信息)

x－Requested-With : 是否为同步请求 ，如果为XMLHttpRequest，则为 Ajax 异步请求。如果为null则为传统同步请求

### 从输入网站地址到打开经过哪些协议及过程 从底层去剖析

①浏览器输入地址，DNS解析查找ip，首先查看浏览器缓存>本地hosts文件>路由器缓存>ISP服务器缓存>递归搜索(根域名服务器缓存，顶级域名服务器缓存，主域名服务器缓存)

②TCP连接，HTTP协议是使用TCP作为其传输层协议的，当TCP出现瓶颈时，HTTP也会受到影响

③浏览器发送http请求

④服务器处理请求并返回HTTP报文

⑤浏览器解析渲染页面

⑥连接结束

### apache与nginx区别及底层实现

一般来说，需要性能的 web 服务，用 nginx 。如果不需要性能只求稳定，更考虑 apache

Apache:同步多进程模型，一个连接对应一个进程。

稳定。

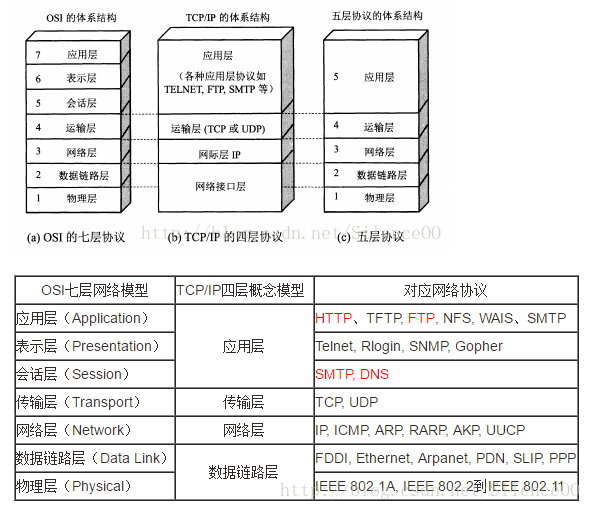
Nginx:异步的，多个连接（万级别）可以对应一个进程。

抗并发。

nginx 处理静态文件好。

nginx 作为负载均衡服务器，支持 7 层负载均衡

### 五层网络及各个层的协议



### http与https区别

http是超文本传输协议，信息是明文传输，https则是具有安全性的ssl/tls加密传输协议。

http和https使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。

### Redis为什么抗并发及性能好 从底层原理来解释

数据存在内存，存取快速。

采用单线程，避免了不必要的上下文切换和竞争条件，也不存在多进程或者多线程导致的切换而消耗 CPU，不用去考虑各种锁的问题，不存在加锁释放锁操作，没有因为可能出现死锁而导致的性能消耗；

### redis基本数据结构

字符串：键值对存储

哈希：存储多个键值对

列表：字符串列表，先进先出，可以重复

无序集合：不能出现相同值

有序集合：不能出现相同值，一个值对应有一个序号

### 线程池的实现

使用pthread 扩展。

### 连接池的实现

数据库连接池负责分配、管理和释放数据库连接，它允许应用程序重复使用一个现有的数据库连接，而不是再重新建立一个；释放空闲时间超过最大空闲时间的数据库连接来避免因为没有释放数据库连接而引起的数据库连接遗漏。这项技术能明显提高对数据库操作的性能。

laravel可以使用swoole实现连接池

### 线程与进程的区别 及通信方式和对应的同步方式？你用过吗？具体怎么用？

区别：

一个程序至少有一个进程,一个进程至少有一个线程.

他们之间根本区别在于 多进程中每个进程有自己的地址空间，线程则共享地址空间。所有其他区别都是因为这个区别产生的。比如说：

1. 速度。线程产生的速度快，通讯快，切换快，因为他们处于同一地址空间。

2. 线程的资源利用率好。

3. 线程使用公共变量或者内存的时候需要同步机制，但进程不用。

因为那个根本原因，实际上只有进程间需要通信,同一进程的线程共享地址空间,没有通信的必要，但要做好同步/互斥,保护共享的全局变量。

进程间通信方式：

管道（Pipe）及有名管道（named pipe）：管道可用于具有亲缘关系进程间的通信，有名管道克服了管道没有名字的限制，因此，除具有管道所具有的功能外，它还允许无亲缘关系进程间的通信；

信号（Signal）：信号是比较复杂的通信方式，用于通知接受进程有某种事件发生，除了用于进程间通信外，进程还可以发送信号给进程本身；linux除了支持Unix早期信号语义函数sigal外，还支持语义符合Posix.1标准的信号函数sigaction（实际上，该函数是基于BSD的，BSD为了实现可靠信号机制，又能够统一对外接口，用sigaction函数重新实现了signal函数）；

消息队列（Message）：消息队列是消息的链接表，包括Posix消息队列system V消息队列。有足够权限的进程可以向队列中添加消息，被赋予读权限的进程则可以读走队列中的消息。消息队列克服了信号承载信息量少，管道只能承载无格式字节流以及缓冲区大小受限等缺点。

\*\* 共享内存\*\*：使得多个进程可以访问同一块内存空间，是最快的可用IPC形式。是针对其他通信机制运行效率较低而设计的。往往与其它通信机制，如信号量结合使用，来达到进程间的同步及互斥。

信号量（semaphore）：主要作为进程间以及同一进程不同线程之间的同步手段。

套接口（Socket）：更为一般的进程间通信机制，可用于不同机器之间的进程间通信。起初是由Unix系统的BSD分支开发出来的，但现在一般可以移植到其它类Unix系统上：Linux和System V的变种都支持套接字。

进程中线程同步的四种常用方式：

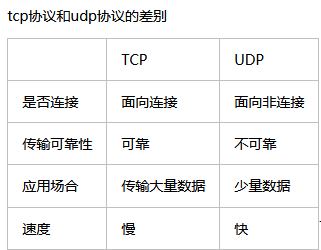
1. 互斥量： 采用互斥对象机制，只有拥有互斥对象的线程才有访问公共资源的权限。因为互斥对象只有一个，所以可以保证公共资源不会被多个线程同时访问。

2. 信号量： 它允许同一时刻多个线程来访问同一资源，但是需要控制同一时刻访问此资源的最大线程数量。

3. 事件（信号）：通过通知操作的方式来保持多线程同步，还可以方便实现多线程优先级的比较作。

4.临界区：临界区对象和互斥对象非常相似，只是互斥量允许在进程间使用，而临界区只限制与同一进程的各个线程之间使用，但是更节省资源，更有效率。

### TCP与UDP的区别



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、UDP具有较好的实时性，工作效率比TCP高，适用于对高速传输和实时性有较高的通信或广播通信。

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对系统资源要求较多，UDP对系统资源要求较少。

### 负载均衡实现原理

将同一应用部署到多台机器上，组成处理集群，将用户访问的请求，根据负载均衡算法，分发到集群中的一台处理服务器。

使用一台机器作为负载均衡服务器，nginx服务器配置负载均衡。

负责均衡下session数据共享：

①nginx中的ip\_hash技术能够将某个ip的请求固定到同一台后端应用服务器，这样一来这个ip下的某个客户端和某个后端就能建立起稳固的session。

②使用mysql数据库存储session。

③使用客户端的cookie作为存放登录信息的媒介

④使用memcache或者redis等缓存机制存放session

### RabitMQ怎么部署 架构是什么样的

实现了高级消息队列协议（AMQP）的开源消息代理软件

### 33. 从库如何做分布式

### 34. 微服务之间如何通信

1、客户端到服务端通信，API Gateway方法。

API Gateway是解决微服务通信的一个不错的方法。以客户端为例。一个客户端可以向多个微服务中的任意一个微服务发出请求。API Gateway负责请求转发、合成和协议转换。所有请求都要先经过API Gateway，然后再将请求转发到对应的微服务中。

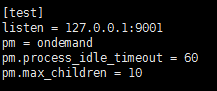
2、进程通信IPC

这种通信不但可以实现一对一、一对多。还可以实现同步和异步请求。

### Php-fpm如何管理进程及进程配置方法

php-fpm进程管理一共有三种模式：ondemand、static、dynamic。工作模式和nginx类似，都是一个master，多个worker模型。

**Ondemand**：在php-fpm启动的时候，不会给这个pool启动任何一个worker，是按需启动，当有连接过来才会启动。



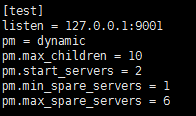
1秒定时器作用：

找到空闲worker，如果空闲时间超过pm.process\_idle\_timeout大小，关闭。这个机制可能会关闭所有的worker。

优点：按流量需求创建，不浪费系统资源（在硬件如此便宜的时代，这个优点略显鸡肋）

缺点：由于php-fpm是短连接的，所以每次请求都会先建立连接，建立连接的过程必然会触发上图的执行步骤，所以，在大流量的系统上master进程会变得繁忙，占用系统cpu资源，不适合大流量环境的部署

**dynamic**：在php-fpm启动时，会初始启动一些worker，在运行过程中动态调整worker数量，worker的数量受限于pm.max\_children配置，同时受限全局配置process.max



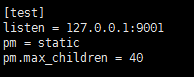
1秒定时器作用：

检查空闲worker数量，按照一定策略动态调整worker数量，增加或减少。增加时，worker最大数量<=max\_children· <=全局process.max；减少时，只有idle >pm.max\_spare\_servers时才会关闭一个空闲worker。

优点：动态扩容，不浪费系统资源，master进程设置的1秒定时器对系统的影响忽略不计；

缺点：如果所有worker都在工作，新的请求到来只能等待master在1秒定时器内再新建一个worker，这时可能最长等待1s；

Static：php-fpm启动采用固定大小数量的worker，在运行期间也不会扩容，虽然也有1秒的定时器，仅限于统计一些状态信息，例如空闲worker个数，活动worker个数，网络连接队列长度等信息。



如果配置成static，只需要考虑max\_children的数量，数量取决于cpu的个数和应用的响应时间，我司配置的是50。

### nginx怎么做负载均衡 原理是什么

nginx提供了以下三种负载均衡机制、方法：

round-robin — 请求以循环、轮转的方式分发到应用服务器。

least-connected — 下一个请求被分配到拥有最少活动连接数的服务器

ip-hash — 使用一个哈希函数，基于客户端ip地址判断下一个请求应该被分发到哪个服务器。

编辑nginx配置文件：

http {

upstream myapp1 {

ip-hash;

        server srv1.example.com;

        server srv2.example.com;

        server srv3.example.com;

    }

    server {

        listen 80;

        location / {

            proxy\_pass http://myapp1;

        }

    }

}

### 索引的底层数据结构

索引存储在文件里。

存储结构有： 二叉树、红黑树、Hash、B-Tree、B+Tree（MySQL索引的真正存储结构）

https://blog.csdn.net/u010922732/article/details/82992920

### Sql语句是否走索引

在查询语句前面加上explain。

显示sql执行的类型，从最好到最差的类型为system > const > eq\_ref > ref > fulltext > ref\_or\_null > index\_merge > unique\_subquery > index\_subquery > range > index > ALL。一般来说，type至少要达到range级别，最好达到ref级别，低于range级别的sql必须进行优化。

Key：显示sql执行过程中实际使用的键或索引，如果为null则表示未使用任何索引，必须进行优化。

### sql优化方案

对大数据查询建立索引。

当只要一行数据时使用 LIMIT 1。

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，建表时尽量默认为0代替null

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。

应尽量避免在 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 allselect id from t where num=20

.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

下面的查询也将导致全表扫描：

select id from t where name like '%abc%'

应使用后模糊'abc%'

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

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

应改为:

select id from t where name like 'abc%'

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。

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。

### 事务的隔离级别及应用，包括结果输出

https://blog.csdn.net/m0\_38143375/article/details/102625510

### innodb与myisam区别及底层数据结构

InnoDB支持事务，MyISAM不支持

InnoDB支持外键，而MyISAM不支持

InnoDB是聚集索引，使用B+Tree作为索引结构，数据文件是和（主键）索引绑在一起的（表数据文件本身就是按B+Tree组织的一个索引结构），必须要有主键，通过主键索引效率很高。但是辅助索引需要两次查询，先查询到主键，然后再通过主键查询到数据。因此，主键不应该过大，因为主键太大，其他索引也都会很大。

MyISAM是非聚集索引，也是使用B+Tree作为索引结构，索引和数据文件是分离的，索引保存的是数据文件的指针。主键索引和辅助索引是独立的。

也就是说：InnoDB的B+树主键索引的叶子节点就是数据文件，辅助索引的叶子节点是主键的值；而MyISAM的B+树主键索引和辅助索引的叶子节点都是数据文件的地址指针。

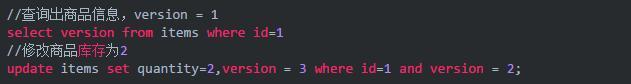
### 聚簇索引与非聚簇索引区别

41问有答

### 乐观锁 悲观锁

悲观锁：具有强烈的独占和排他特性，在整个数据处理过程中，将数据处于锁定状态。悲观锁的实现，往往依靠数据库提供的锁机制（mysql读锁，写锁）

乐观锁：采取了更加宽松的加锁机制，不会刻意使用数据库本身的锁机制，而是依据数据本身来保证数据的正确性，一般的实现乐观锁的方式就是记录数据版本。



### explain有哪些字段 分别是什么意思

table：显示这一行的数据是关于哪张表的

type：这是重要的列，显示连接使用了何种类型。从最好到最差的连接类型为const、eq\_reg、ref、range、indexhe和ALL

possible\_keys：显示可能应用在这张表中的索引。如果为空，没有可能的索引。可以为相关的域从WHERE语句中选择一个合适的语句。这个列表是在优化过程的早期创建的，因此有些列出来的索引可能对于后续优化过程是没有用的。

key： 实际使用的索引。如果为NULL，则没有使用索引。很少的情况下，MYSQL会选择优化不足的索引。这种情况下，可以在SELECT语句中使用USE INDEX（indexname）来强制使用一个索引或者用IGNORE INDEX（indexname）来强制MYSQL忽略索引

如果该索引没有出现在possible\_keys中，那么mysql选用的是出于另外的原因——例如，他可能选择了一个覆盖索引，哪怕没有where字句，换句话说，possible\_keys揭示了哪一个索引能有助于提高查询效率，而key显示的是优化采用哪一个索引可以最小化查询成本。

key\_len：使用的索引的长度。在不损失精确性的情况下，长度越短越好

ref：显示索引的哪一列被使用了，如果可能的话，是一个常数

rows：MYSQL认为必须检查的用来返回请求数据的行数

Extra：关于MYSQL如何解析查询的额外信息。将在表4.3中讨论，但这里可以看到的坏的例子是Using temporary和Using filesort，意思MYSQL根本不能使用索引，结果是检索会很慢

Type列的解释

Type：告诉我们对表使用的访问方式，主要包含如下集中类型。

all：全表扫描。

const：读常量，最多只会有一条记录匹配，由于是常量，实际上只须要读一次。

eq\_ref：最多只会有一条匹配结果，一般是通过主键或唯一键索引来访问。

fulltext：进行全文索引检索。

index：全索引扫描。

index\_merge：查询中同时使用两个（或更多）索引，然后对索引结果进行合并（merge），再读取表数据。

index\_subquery：子查询中的返回结果字段组合是一个索引（或索引组合），但不是一个主键或唯一索引。

rang：索引范围扫描。

ref：Join语句中被驱动表索引引用的查询。

ref\_or\_null：与ref的唯一区别就是在使用索引引用的查询之外再增加一个空值的查询。

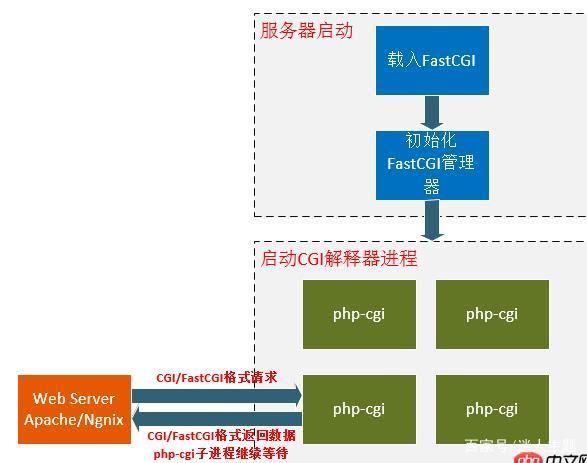
system：系统表，表中只有一行数据；

unique\_subquery：子查询中的返回结果字段组合是主键或唯一约束。

### Php-fpm与fastcgi是什么交互原理

**CGI**：公共网关接口，它是Web服务器与外部应用程序（CGI程序）之间传递信息的接口标准。通过CGI接口，Web服务器就能够获取客户端提交的信息，并转交给服务器端的CGI程序处理，最后返回结果给客户端。每当客户请求CGI的时候，WEB服务器就请求操作系统生成一个新的CGI解释器进程(如php-cgi.exe)，CGI 的一个进程则处理完一个请求后退出，下一个请求来时再创建新进程。当然，这样在访问量很少没有并发的情况也行。可是当访问量增大，并发存在，这种方式就不适合了。于是就有了fastcgi。

**FastCGI**：像是一个常驻(long-live)型的CGI，它可以一直执行着，只要激活后，不会每次都要花费时间去fork一次.



Web Server启动时载入FastCGI进程管理器（Apache Module或IIS ISAPI等)

FastCGI进程管理器自身初始化，启动多个CGI解释器进程(可建多个php-cgi)，并等待来自Web Server的连接。

当客户端请求到达Web Server时，FastCGI进程管理器选择并连接到一个CGI解释器。Web server将CGI环境变量和标准输入发送到FastCGI子进程php-cgi。

FastCGI子进程完成处理后，将标准输出和错误信息从同一连接返回Web Server。当FastCGI子进程关闭连接时，请求便告处理完成。FastCGI子进程接着等待，并处理来自FastCGI进程管理器(运行在Web Server中)的下一个连接。 在CGI模式中，php-cgi在此便退出了。

**php-fpm**：是PHP内置的一个FastCGI 进程管理器。是一种master（主）/worker（子）多进程架构。

工作流程：

1).FPM的master进程接收到请求。

2).master进程根据配置指派特定的worker进程进行请求处理，如果没有可用进程，返回错误，这也是我们配合Nginx遇到502错误比较多的原因。

3).worker进程处理请求，如果超时，返回504错误。

### epoll与select区别

Select:时间复杂度O(n)，它仅仅知道了，有I/O事件发生了，却并不知道是哪那几个流（可能有一个，多个，甚至全部），我们只能无差别轮询所有流，找出能读出数据，或者写入数据的流，对他们进行操作。所以select具有O(n)的无差别轮询复杂度，同时处理的流越多，无差别轮询时间就越长。

while true

{

select(streams[]) //这一步死在这里，知道有一个流有I/O事件时，才往下执行

for i in streams[]

{

if i has data

read until unavailable

}

}

Epoll:时间复杂度O(1)，epoll可以理解为event poll，不同于忙轮询和无差别轮询，epoll会把哪个流发生了怎样的I/O事件通知我们。所以我们说epoll实际上是事件驱动（每个事件关联上fd）的，此时我们对这些流的操作都是有意义的。（复杂度降低到了O(1)）

while true

{

active\_stream[] = epoll\_wait(epollfd)

for i in active\_stream[]

{

read or write till

}

}

### linux如何设置进程数

进入 /etc/security/limits.conf 文件下，增加如下代码，并保存：

\* soft nproc 10240

\* hard nproc 10240

10240 为设置的进程数，表示对所有用户都生效。

### Linux 杀掉 php-cgi命令

用ps查看进程。

$ ps -aux

$ kill -s 9 17149等一系列信号

### redis队列与kafka rabiitMQ区别

redis： 轻量级，低延迟，高并发，低可靠性；

rabbitmq：重量级，高可靠，异步，不保证实时；

rabbitmq是一个专门的AMQP协议队列，他的优势就在于提供可靠的队列服务，并且可做到异步，而redis主要是用于缓存的，redis的发布订阅模块，可用于实现及时性，且可靠性低的功能。

52. 微信红包设计

### 秒杀设计

https://blog.csdn.net/m0\_38143375/article/details/103265494

### 中间件设计

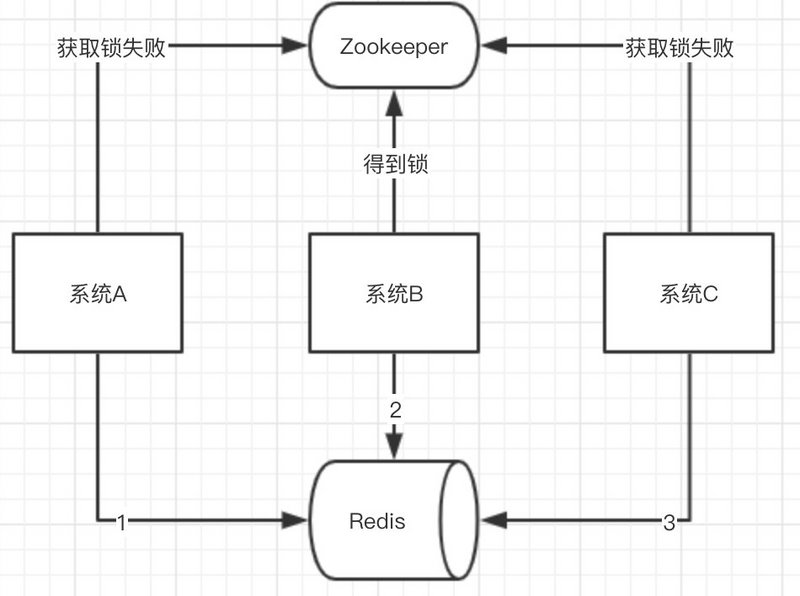
中间件，顾名思义存在于两个系统之间的，起到连接的设备。

### Redis 网络抖动如何处理

网络抖动是指最大延迟与最小延迟的时间差，如最大延迟是20毫秒，最小延迟为5毫秒，那么网络抖动就是15毫秒

系统A、B、C三个系统，分别去操作Redis的同一个Key，本来顺序是1，2，3是正常的，但是因为系统A网络突然抖动了一下，B，C在他前面操作了Redis，这样数据不就错了么。

处理：



某个时刻，多个系统实例都去更新某个 key。可以基于 Zookeeper 实现分布式锁。每个系统通过 Zookeeper 获取分布式锁，确保同一时间，只能有一个系统实例在操作某个 Key，别人都不允许读和写。

http://www.imooc.com/article/284956?block\_id=tuijian\_wz

### 如何实现主从同步，如何配置

主从数据库实现读写分离缓解网站并发访问。

https://blog.csdn.net/qq\_15092079/article/details/81672920

### 如何防止重复下单

在前端下单按钮防多次点击

### 如何避免redis 抖动打穿缓存

缓存穿透：缓存穿透是指查询一个一定不存在的数据，由于缓存是不命中时需要从数据库查询，查不到数据则不写入缓存，这将导致这个不存在的数据每次请求都要到数据库去查询，造成缓存穿透。

解决办法：对所有可能查询的参数以hash形式存储，在控制层先进行校验，不符合则丢弃。

缓存雪崩：如果缓存集中在一段时间内失效，发生大量的缓存穿透，所有的查询都落在数据库上，造成了缓存雪崩。

解决办法：在缓存失效后，通过加锁或者队列来控制读数据库写缓存的线程数量。比如对某个key只允许一个线程查询数据和写缓存，其他线程等待。可以通过缓存reload机制，预先去更新缓存，再即将发生大并发访问前手动触发加载缓存

不同的key，设置不同的过期时间，让缓存失效的时间点尽量均匀

### 分布式架构怎么搭建（消息队列 redis）

https://www.jianshu.com/p/d32b16f12f09

### 上亿条的数据库查询方案 按商品购买数量排序 取前100

首先需要对数据库进行优化

①.优化你的sql和索引

②.加缓存，memcached,redis

③.主从复制，读写分离

④.垂直拆分:将一个属性较多，一行数据较大的表，将不同的属性拆分到不同的表中。简单说就是把表的属性拆分成多个表存。

⑤.水平拆分：水平切分算法有“范围法”和“哈希法”。

范围法：以业务主键id为划分依据，将数据水平切分到两个数据库实例上去，例如：表1：存储0到1千万的uid数据。表2：存储1到2千万的uid数据。

哈希法：也是以业务主键id为划分依据，模=id%2，表1：存储id取模得1的uid数据。表2：存储id取模得0的uid数据。

### 高并发 如何按顺序给不同的服务器写文件

（没有找到标准答案）

使用redis消息队列。

### 二叉树的遍历

https://www.jianshu.com/p/45d75aeb3b01

### 判断链表是否有交叉

https://blog.csdn.net/qq\_22313585/article/details/78953555

### 无限级分类

使用pid，一级0，二级是上一级主键...

### 插入排序

每一步将一个待排序的数据插入到前面已经排好序的有序序列中，直到插完所有元素为止

//插入排序

$a = [31,41,59,26,41,58];

$count = count($a);

for($i=1;$i<$count;$i++){

$key = $a[$i];

$j=$i-1;

while($j>=0 && $a[$j]>$key){

$a[$j+1] = $a[$j];

$j--;

}

$a[$j+1] = $key;

}

var\_dump($a);

### 快排（二分，递归排序）

通过设置一个初始中间值，来将需要排序的数组分成3部分，小于中间值的左边，中间值，大于中间值的右边，继续递归用相同的方式来排序左边和右边，最后合并数组。

function fastSort($arr){

if(count($arr)<1){

return $arr;

}

$middle = $arr[0];

$left = [];

$right = [];

for($i=1;$i<count($arr);$i++){

if($arr[$i]<$middle){

$left[] = $arr[$i];

}else{

$right[] = $arr[$i];

}

}

$left = fastSort($left);

$right = fastSort($right);

return array\_merge($left,(array)$middle,$right);

}

$a = array(2,13,42,34,56,23,67,365,87665,54,68,3);

print\_r(fastSort($a));

### 堆和栈的区别并举例

栈（操作系统）：由操作系统自动分配释放 ，存放函数的参数值，局部变量的值等。其操作方式类似于数据结构中的栈；

堆（操作系统）： 一般由程序员分配释放， 若程序员不释放，程序结束时可能由OS回收，分式倒是类似于链表。

堆（数据结构）：堆可以被看成是一棵树，如：堆排序；

栈（数据结构）：一种先进后出的数据结构。

### 哈希算法及如何解决哈希冲突

定义：把任意长度的输入（又叫做预映射pre-image）通过散列算法变换成固定长度的输出，该输出就是散列值。

常用HASH函数：

①.直接寻址法：取关键字或关键字的某个线性函数值为散列地址。即

H(key)=key或H(key) = a·key + b，其中a和b为常数。

②.数字分析法：取关键字的中的若干位作为哈希地址。

③.平方取中法：取关键字平方后的中间几位作为散列地址。

④.将关键字分割成位数相同的几部分（最后一部分可以不同），然后取这几

部分的叠加和作为哈希地址。

⑤.随机数法。选择一随机函数，取关键字作为随机函数的种子生成随机值作

为散列地址，通常用于关键字长度不同的场合。

⑥.除留余数法。取关键字被某个不大于散列表表长m的数p除后所得的余

数为散列地址。即 H(key) = key MOD p,p<=m。不仅可以对关键字直接取模，

也可在折叠、平方取中等运算之后取模。对p的选择很重要，一般取素数或

m，若p选的不好，容易产生碰撞。

最常用的是除留余数法

冲突：对不同的关键字可能得到同一散列地址，即key1≠key2，而f(key1)=f(key2)，这种现象称碰撞

### 反转字符串

$a = "rfhdsdfh";

$fan = "";

for($i=0;$i<strlen($a);$i++){

$fan = $a[$i].$fan;

}

echo $fan;

### 反转二叉树.

public static TreeNode InvertTree1(TreeNode root)

{

if (root == null) return root;

TreeNode tmp = root.Left;

root.Left = InvertTree1(root.Right);

root.Right = InvertTree(tmp);

return root;

}