## 目前php使用的微服务框架

//RPC

//Consul、zookeeper

Hyperf、swoft、easySwoole

## 一．Hyperf基础

文档：<https://hyperf.wiki/2.0/#/README>

### 安装

A：使用docker镜像

B：单独安装，服务器满足条件：

* PHP >= 7.2
* Swoole PHP 扩展 >= 4.5，并关闭了 Short Name
* OpenSSL PHP 扩展
* JSON PHP 扩展
* PDO PHP 扩展 （如需要使用到 MySQL 客户端）
* Redis PHP 扩展 （如需要使用到 Redis 客户端）
* Protobuf PHP 扩展 （如需要使用到 gRPC 服务端或客户端）

#### Docker安装hyperf：

1.安装docker、composer create-project hyperf/hyperf-skeleton name(项目名称)

2.拉取、运行hyperf镜像

执行以下命令：

# 下载并运行 hyperf/hyperf 镜像，并将镜像内的项目目录绑定到宿主机的 /tmp/skeleton 目录

#将服务器上的项目挂载到镜像中，镜像内就可以共享服务器上的项目(将项目上传到服务器后，通过此命令运行最新的代码)

docker run -v /home/www/hyperf:/hyperf-skeleton -p 9501:9501 -it --entrypoint /bin/sh hyperf/hyperf:latest

# 镜像容器运行后，在容器内安装 Composer

wget <https://github.com/composer/composer/releases/download/1.8.6/composer.phar(官方给的命令不能用换成> wget https://getcomposer.org/composer.phar)

chmod u+x composer.phar

mv composer.phar /usr/local/bin/composer

# 将 Composer 镜像设置为阿里云镜像，加速国内下载速度

composer config -g repo.packagist composer https://mirrors.aliyun.com/composer

(并不是全局的每次安装新项目 先切换以下镜像源)

# 通过 Composer 安装 hyperf/hyperf-skeleton 项目

composer create-project hyperf/hyperf-skeleton

# 进入安装好的 Hyperf 项目目录

cd hyperf-skeleton

# 启动 Hyperf

php bin/hyperf.php start

参考：

<https://hyperf.wiki/2.0/#/zh-cn/quick-start/install?id=%e6%9c%8d%e5%8a%a1%e5%99%a8%e8%a6%81%e6%b1%82>

docker run -it -v /home/dock/Downloads:/usr/Downloads ubuntu64 /bin/bash

docker run -v /home/www/hyperf:/hyperf-skeleton -p 9501:9501 -it --entrypoint /bin/sh hyperf/hyperf:latest

#### 从docker容器复制项目到服务器

docker cp dockerid:/home/hyperf-skeleton /home/www/hyperf

### 生命周期

### 注解

Hyperf的注解类似于springboot的注解，但hyperf的注解必须写在注释块中才能被[doctrine/annotations](https://github.com/doctrine/annotations" \t "_blank) 包解析到。

/\*\*

\* @AnnotationClass()

\*/

和sprinboot一样可以对类、属性、方法使用注解

#### 3.1注解传递参数

传递主要的单个参数 @DemoAnnotation("value")

传递字符串参数 @DemoAnnotation(key1="value1", key2="value2")

传递数组参数 @DemoAnnotation(key={"value1", "value2"})

### 4.路由

#### 4.1普通路由定义

Router::get('/test','App\Controller\IndexController@test');

#### 4.2响应多种请求方式路由

Router::addRoute(['GET','POST'],'/test2','App\Controller\IndexController@test2');

路由组

Router::addGroup('/group',function(){  
 Router::get('index','App\Controller\IndexController@index');  
 Router::post('store','App\Controller\IndexController@store');  
 Router::get('update','App\Controller\IndexController@update');  
 Router::post('delete','App\Controller\IndexController@delete');  
});

#### 4.3路由参数 必填参数

Router::get('/test3/{id}','App\Controller\IndexController@test3');

#### 4.4路由参数 选填参数

Router::get('/test4/[id]','App\Controller\IndexController@test3');

#### 4.5注解定义路由

##### @AutoController注解



##### @Controller注解

@Controller 为满足更细致的路由定义需求而存在，使用 @Controller 注解用于表明当前类为一个 Controller 类，同时需配合 @RequestMapping 注解来对请求方法和请求路径进行更详细的定义。



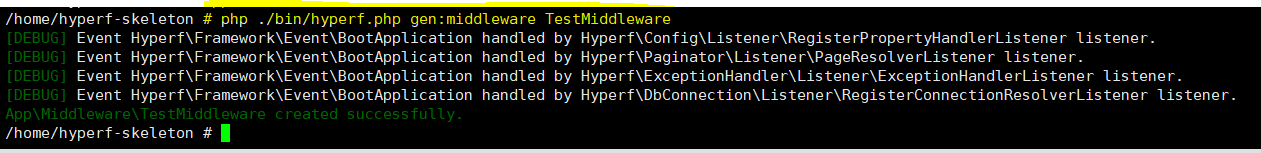
### 5中间件

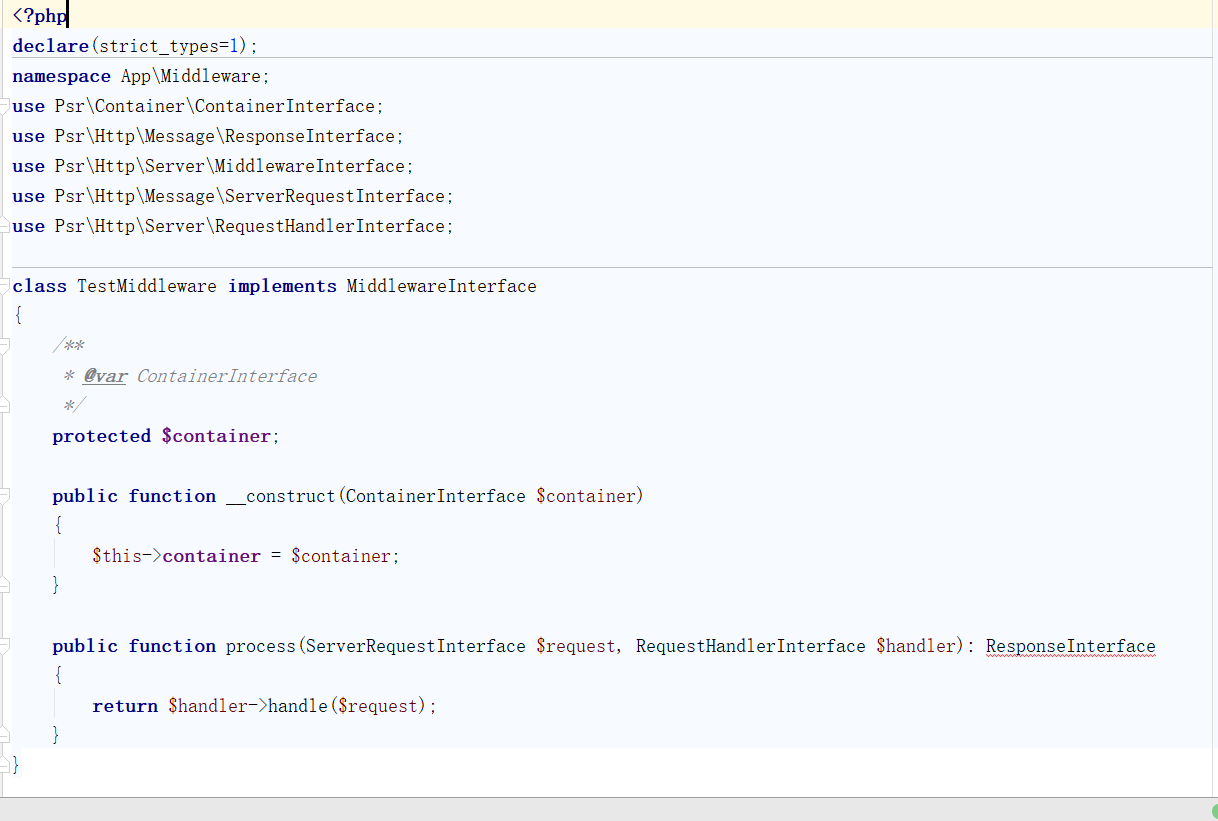
#### 5.1中间件类代码

指令生成中间件

php ./bin/hyperf.php gen:middleware TestMiddleware

(在项目目录运行， 默认会在app/Middleware目录下生成文件)





**<?php  
declare**(strict\_types=1);  
**namespace** App\Middleware;  
  
**use** Psr\Container\ContainerInterface;  
**use** Psr\Http\Message\ResponseInterface;  
**use** Psr\Http\Server\MiddlewareInterface;  
**use** Psr\Http\Message\ServerRequestInterface;  
**use** Psr\Http\Server\RequestHandlerInterface;  
**use** Hyperf\HttpServer\Contract\ResponseInterface **as** HttpResponse;  
**use** Hyperf\HttpServer\Contract\RequestInterface;  
**class** TestMiddleware **implements** MiddlewareInterface  
{  
 */\*\*  
 \** ***@var*** *ContainerInterface  
 \*/* **protected $container**;  
  
 **public function** \_\_construct(ContainerInterface $container,HttpResponse $response,RequestInterface $request)  
 {  
 $this->**container** = $container;  
 $this->**response** = $response;  
 $this->**request** = $request;  
 }  
  
 **public function** process(ServerRequestInterface $request, RequestHandlerInterface $handler): ResponseInterface  
 {  
*// return $handler->handle($request);  
 // 根据具体业务判断逻辑走向，这里假设用户携带的token有效* $isValidToken = **true**;  
 **if** ($isValidToken) {  
 **return** $handler->handle($request);  
 }  
 **return** $this->**response**->json(  
 [  
 **'code'** => -1,  
 **'data'** => [  
 **'error'** => **'中间里验证token无效，阻止继续向下执行'**,  
 ],  
 ]  
 );  
 }  
}

#### 5.2中间件级别分类

全局中间件、类级别中间件、方法级别中间件

执行顺序：全局中间件—>类级别中间件—>方法级别中间件

#### 5.3定义全局中间件

全局docker配置文件路径：config/autoload/middlewares.php

<?php  
  
declare(strict\_types=1);  
/\*\*  
 \* This file is part of Hyperf.  
 \*  
 \* @link https://www.hyperf.io  
 \* @document https://hyperf.wiki  
 \* @contact group@hyperf.io  
 \* @license https://github.com/hyperf/hyperf/blob/master/LICENSE  
 \*/  
return [  
 'http' => [  
 \App\Middleware\TestMiddleware::class  
 ],  
];

#### 5.4定义局部中间件

*//在路由配置上定义 局部中间件*Router::*get*(**'/test5'**,**'App\Controller\IndexController@test5'**,  
 [**'middleware'**=>\App\Middleware\TestMiddleware::***class***]);  
*//在路由组上 定义局部中间件*Router::*addGroup*(**'/group'**,**function**(){  
 Router::**get**(**'index2'**,**'App\Controller\IndexController@index'**);  
 Router::**post**(**'store2'**,**'App\Controller\IndexController@store'**);  
 Router::**get**(**'update2'**,**'App\Controller\IndexController@update'**);  
 Router::**post**(**'delete2'**,**'App\Controller\IndexController@delete'**);  
}, [**'middleware'**=>\App\Middleware\TestMiddleware::***class***]);

#### 5.5注解定义中间件

使用 @Middleware 注解时需 use Hyperf\HttpServer\Annotation\Middleware; 命名空间；  
使用 @Middlewares 注解时需 use Hyperf\HttpServer\Annotation\Middlewares; 命名空间；

##### 单个中间件



##### 多个中间件



##### 注释定义到方法级别中间件



#### 5.6跨域中间件

**<?php**

declare(strict\_types=1);

namespace App\Middleware;

use Hyperf\Utils\Context;

use Psr\Http\Message\ResponseInterface;

use Psr\Http\Message\ServerRequestInterface;

use Psr\Http\Server\MiddlewareInterface;

use Psr\Http\Server\RequestHandlerInterface;

class CorsMiddleware implements MiddlewareInterface

{

public function process(ServerRequestInterface $request, RequestHandlerInterface $handler): ResponseInterface

{

$response = Context::get(ResponseInterface::class);

$response = $response->withHeader('Access-Control-Allow-Origin', '\*')

->withHeader('Access-Control-Allow-Credentials', 'true')

// Headers 可以根据实际情况进行改写。

->withHeader('Access-Control-Allow-Headers', 'DNT,Keep-Alive,User-Agent,Cache-Control,Content-Type,Authorization');

Context::set(ResponseInterface::class, $response);

if ($request->getMethod() == 'OPTIONS') {

return $response;

}

return $handler->handle($request);

}

}

### 6.控制器

### 7.请求

#### 7.1请求相关方法

##### 7.1.1获取请求路径

path() 方法返回请求的路径信息。也就是说，如果传入的请求的目标地址是 http://domain.com/foo/bar?baz=1，那么 path() 将会返回 foo/bar：

$url=$request->path();

##### 7.1.2请求路径验证

If($request->is(‘user/\*’){

}

##### 7.1.3获取完整请求路径

//没有查询参数

$url=$request->url()

//携带参数参数

$url=$request->fullUrl();

##### 7.1.4获取请求方法

getMethod() 方法将返回 HTTP 的请求方法。你也可以使用 isMethod(string $method) 方法去验证 HTTP 的请求方法与指定规则是否匹配：

$method=$request->getMethod();

if ($request->isMethod('post')) {

// ...

}

#### 7.2获取输入

##### 7.2.1获取所有输入

$name=$request->all();

##### 7.2.2获取指定输入值

//存在则返回，不存在则返回null

$name=$request->input();

//存在则返回，不存在则返回Hyperf

$name=$request->input(‘name’,’Hyperf’)

**如果传输表单数据中包含「数组」形式的数据，那么可以使用「点」语法来获取数组：**

$name = $request->input('products.0.name');

$names = $request->input('products.\*.name');

##### 7.2.3从查询字符串获取输入

使用input,inputs方法可以从整个请求中获取输入数据(包括Query参数)，而 query(?string $key = null, $default = null) 方法可以只从查询字符串中获取输入数据：

// 存在则返回，不存在则返回 null

$name = $request->query('name');

// 存在则返回，不存在则返回默认值 Hyperf

$name = $request->query('name', 'Hyperf');

// 不传递参数则以关联数组的形式返回所有 Query 参数

$name = $request->query();

##### 7.2.4获取json输入信息

如果请求的 Body 数据格式是 JSON，则只要 请求对象(Request) 的 Content-Type Header 值 正确设置为 application/json，就可以通过 input(string $key, $default = null) 方法访问 JSON 数据，你甚至可以使用 「点」语法来读取 JSON 数组：

// 存在则返回，不存在则返回 null

$name = $request->input('user.name');

// 存在则返回，不存在则返回默认值 Hyperf

$name = $request->input('user.name', 'Hyperf');

// 以数组形式返回所有 Json 数据

$name = $request->all();

##### 7.24判断是否有输入值

要判断请求是否存在某个值，可以使用 has($keys) 方法。如果请求中存在该值则返回 true，不存在则返回 false，$keys 可以传递一个字符串，或传递一个数组包含多个字符串，只有全部存在才会返回 true：

// 仅判断单个值

if ($request->has('name')) {

// ...

}

// 同时判断多个值

if ($request->has(['name', 'email'])) {

// ...

}

##### 7.2.5从请求中获取cookie值

使用 getCookieParams() 方法从请求中获取所有的 Cookies，结果会返回一个关联数组。

$cookies = $request->getCookieParams();

如果希望获取某一个 Cookie 值，可通过 cookie(string $key, $default = null) 方法来获取对应的值：

// 存在则返回，不存在则返回 null

$name = $request->cookie('name');

// 存在则返回，不存在则返回默认值 Hyperf

$name = $request->cookie('name', 'Hyperf');

#### 7.3文件

##### 7.3.1获取上传文件

// 存在则返回一个 Hyperf\HttpMessage\Upload\UploadedFile 对象，不存在则返回 null

$file = $request->file('photo');

##### 7.3.2检查文件是否存在

if ($request->hasFile('photo')) {

// ...

}

##### 7.3.3验证是否上传成功

if ($request->file('photo')->isValid()) {

// ...

}

##### 7.3.4获取文件路劲&扩展名

UploadedFile 类还包含访问文件的完整路径及其扩展名方法。getExtension() 方法会根据文件内容判断文件的扩展名。该扩展名可能会和客户端提供的扩展名不同：

// 该路径为上传文件的临时路径

$path = $request->file('photo')->getPath();

// 由于 Swoole 上传文件的 tmp\_name 并没有保持文件原名，所以这个方法已重写为获取原文件名的后缀名

$extension = $request->file('photo')->getExtension();

##### 7.3.5存储上传文件

上传的文件在未手动储存之前，都是存在一个临时位置上的，如果您没有对该文件进行储存处理，则在请求结束后会从临时位置上移除，所以我们可能需要对文件进行持久化储存处理，通过 moveTo(string $targetPath): void 将临时文件移动到 $targetPath 位置持久化储存，代码示例如下：

$file = $request->file('photo');

$file->moveTo('/foo/bar.jpg');

// 通过 isMoved(): bool 方法判断方法是否已移动

if ($file->isMoved()) {

// ...

}

### 8.响应

#### 返回json格式

$data = [

'key' => 'value'

];

return $response->json($data);

$data = [

'key' => 'value'

];

return $response->xml($data);

return $response->raw('Hello Hyperf.');

#### 重定向

// redirect() 方法返回的是一个 Psr\Http\Message\ResponseInterface 对象，需再 return 回去

return $response->redirect('/anotherUrl');

#### 设置cookie

$cookie = new Cookie('key', 'value');

return $response->withCookie($cookie)->withContent('Hello Hyperf.');

### 9.异常处理

#### 9.1自定义异常处理

## 二．hyperf数据库

### docker安装配置mysql

见docker word

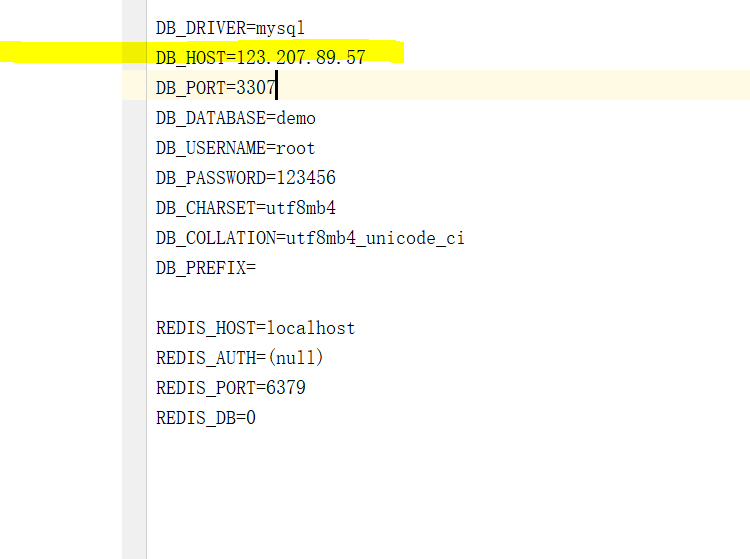
·

<https://www.cnblogs.com/followyou/p/10315717.html>

一开是按的mysql8 但是数据库一直连接不成功，保下面错误

[SQLSTATE[HY000] [2054] The server requested authentication method unknown to the client](https://www.cnblogs.com/liyiran/p/12573765.html)

https://www.cnblogs.com/liyiran/p/12573765.html



数据库配置文件 host一开始写的是loaclhost一直报错 SQLSTATE[HY000] [2002] No such file or directory

Host要写成ip

Hyperf也支持数据库出迁移

## 三．核心架构

### 1.协程

#### 什么是协程

协程是一种轻量级线程，由用户代码来调度和管理，而不是由操作系统内核来进行调度。

Swoole 的每个 Worker 进程 会存在一个协程调度器来调度协程，协程切换的时机就是遇到 I/O 操作或代码显性切换时，进程内以单线程的形式运行协程，也就意味着一个进程内同一时间只会有一个协程在运行且切换时机明确

单个协程内的代码运行仍是串行的，放在一个 HTTP 协程服务上来理解就是每一个请求是一个协程，举个例子，假设为请求 A 创建了 协程 A，为 请求 B 创建了 协程 B，那么在处理 协程 A 的时候代码跑到了查询 MySQL 的语句上，这个时候 协程 A 则会触发协程切换，协程 A 就继续等待 I/O 设备返回结果，那么此时就会切换到 协程 B，开始处理 协程 B 的逻辑，当又遇到了一个 I/O 操作便又触发协程切换，再回过来从 协程 A 刚才切走的地方继续执行，如此反复，遇到 I/O 操作就切换到另一个协程去继续执行而非一直阻塞等待。

#### Aop面向切面编程

## 四.分布式配置中心

将配置与项目脱离开，配置中心根据不同的应用返回不同的配置。

### apollo

文档：<https://github.com/ctripcorp/apollo>

Nocas