**Thinkphp5 Restful Api接口设计理念及各种特殊情况处理**

**作者：孙博文**

**时间：2018年11月29号**

**接口设计理念：**

**一，参数请求处理方式**

1. **数据返回格式**

**namespace app\index\controller;**

**class Index {**

**public function index() {**

**$data = array(**

**'key' => 'value'**

**);**

**$code = 200;**

**$msg = 'ok';**

**return json\_encode(['data' => $data, 'code' => $code, 'message' => $msg]);**

**}**

**}**

**2,公共配置文件配置数据格式输出类型**

**'default\_return\_type' => 'html|xml|json'**

**3,获取参数方式**

**namespace app\index\controller;**

**use \think\Request;**

**class Receive {**

**public function index() {**

**$request = Request::instance();**

**echo '请求方法：' . $request->method() . '<br/>';**

**echo '访问地址：' . $request->ip() . '<br/>';**

**echo '请求参数：';**

**dump($request->param());**

**echo '请求参数：仅包含的key';**

**dump($request->only([‘key’]));**

**echo '请求参数：要排除的字段';**

**dump($request->except(['key ']));**

**}**

**}**

**4,判断请求方法**

**<?php**

**namespace app\index\controller;**

**use thinkphp\Request;**

**Class Test{**

**public function index(){**

**$request = Request::instance();**

**// 是否为 GET 请求**

**if (Request::instance()->isGet()) echo "当前为 GET 请求";**

**// 是否为 POST 请求**

**if (Request::instance()->isPost()) echo "当前为 POST 请求";**

**// 是否为 PUT 请求**

**if (Request::instance()->isPut()) echo "当前为 PUT 请求";**

**// 是否为 DELETE 请求**

**if (Request::instance()->isDelete()) echo "当前为 DELETE 请求";**

**// 是否为 Patch 请求**

**if (Request::instance()->isPatch()) echo "当前为 PATCH 请求";**

**}**

**}**

**5,参数验证规则**

**namespace app\index\controller;**

**use think\Validate;**

**class Verify{**

**public function index(){**

**$rule = [**

**//，这里举几个常用参数 utf-8 一个字符对应3个字母/数字 对应2个汉字(所以这里可以入3个字母/数字或者一个汉字)**

**'name' => 'require|max:3',**

**'age' => 'number|between:1,120',**

**'email' => 'email',**

**];**

**$msg = [**

**'name.require' => '名称必须',**

**'name.max' => '名称最多不能超过3个字符',**

**'age.number' => '年龄必须是数字',**

**'age.between' => '年龄只能在1-120之间',**

**'email' => '邮箱格式错误',**

**];**

**$data = input('post.');**

**$validate = new Validate($rule, $msg);**

**$result = $validate->check($data);**

**if (!$result) {**

**dump($validate->getError());**

**}**

**}**

**}**

**6，数据库连接（项目根目录下databnase.php）**

**7,原生sql查询**

**namespace app\index\controller;**

**use think\Db;**

**class Link{**

**public function index(){**

**$res = Db::query('select version()');**

**return $res;**

**}**

**}**

二、**权限验证**

**1,基于ThinkPHP5基础上开发的restful api ，首先有关权限验证的接口层级情况  
  
2,首先要在框架内定义中定义资源路由，比如定义一个文件router.php，**

**3,当访问相应的url地址时，例如：  
http://localhost/tp5/public/index.php/index/auth**

**定义的控制器是集成了Api类的,会在初始化检测控制器定义的访问限制  
访问相应的url，例如  
$restMethodList = 'get|post|put|patch|delete';**

**4，在Api类中，会有方法authenticate检测用户是否有权限调用接口**

**authenticate方法会调用BasicAuth类中的鉴权，**

**根据用户端换递过来的app\_key获取缓存中的access\_token,在进行对比,如果true,则可以调用控制器中的各种方法,否则返回不能调用原因**

**BasicAuth类中的具体请看代码**

**生成access\_token,缓存access\_tolen等相关逻辑在index/** **OauthAuth.php代码中，使用的时本地缓存，如果需要使用数据库或者redis请查询相关注释说明**

**5，开启授权认证**

**添加配置认证总开关**

'api\_auth' => true, //是否开启授权认证

**权限类实现AuthContract 接口**

**authenticate接口认证返回true则通过，否则不通过**

**getUser获取用户信息，Api控制可以调用**

self::$app['auth']->getUser();

/\*\*

\* 认证授权通过客户端信息和路由等

\* 如果通过返回true

\* @param Request $request

\* @return bool

\*/

**public** **function** **authenticate**(Request $request){

// TODO: Implement authenticate() method.

**return** **true**;

}

/\*\*

\* 获取用户信息 接口里可以直接获取用户信息

\* @return mixed

\*/

**public** **function** **getUser**(){

**return** ['app\_id'=>'111','name'=>'dawn-api'];

}

**6，接口类开启授权认证**

//是否开启授权认证

**public** $apiAuth = **true**;

| **配置(api\_auth)** | **类($apiAuth)** | **效用** |
| --- | --- | --- |
| true | true | 认证开启 |
| true | false | 认证关闭 |
| false | false | 认证关闭 |
| false | true | 认证关闭 |

1. **配置验证类**

'auth\_class' => \app\index\auth\Auth::class, //授权认证类

1. **改写get方法**

/\*\*

\* get

\*

\* @param Request $request

\* @return \think\Response|\think\response\Json|\think\response\Jsonp|\think\response\Redirect|\think\response\Xml

\*/

**public** **function** **get**(Request $request)

$user = self::$app['auth']->getUser();

// todo find

**return** $this->sendSuccess(['name' => 'dawn-api', 'id' => 1, 'user' => $user]);

}

**三，Oauth Client Credentials Grant(Oauth客户端凭据授予)**

1. **配置验证类**

'auth\_class' => \app\index\auth\Auth::class, //授权认证类

1. **获取access\_token**

**编写可访问的方法获取access\_token**

**namespace** app\index\controller;

**use** app\index\auth\OauthAuth;

**use** think\Request;

**class** **Auth**{

**public** **function** **accessToken**(){

$request = Request::instance();

$OauthAuth = **new** OauthAuth();

**return** $OauthAuth->accessToken($request);

}

}

配置路由

'accessToken'=>index/Auth/accessToken',//Oauth认证

**按需改写获取客户端信息**

/\*\*

\* 返回用户信息

\* @return array

\*/

**public** **static** **function** **getUserInfo**()

{

**return** [

'client\_id' => '20882088',//app\_id

'secret' => 'nGk5R2wrnZqQ02bed29rjzax1QWRIu1O',

'name' => 'test\_client'];

}

**请求获取 /accessToken?client\_id=&secret=或者 /accessToken headers Basic =**

**代码实现:**

**namespace app\index\auth;**

**use Dawnapi\contract\AuthContract;**

**use think\Request;**

**class Auth implements AuthContract{**

**/\*\***

**\* 认证授权通过客户端信息和路由等**

**\* 如果通过返回true**

**\* @param Request $request**

**\* @return bool**

**\*/**

**public function authenticate(Request $request){**

**// TODO: Implement authenticate() method.**

**return true;**

**}**

**/\*\***

**\* 获取用户信息 接口里可以直接获取用户信息**

**\* @return mixed**

**\*/**

**public function getUser(){**

**return ['app\_id'=>'111','name'=>'dawn-api'];**

**}**

**测试的基础验证**

**namespace app\index\auth;**

**use Dawnapi\auth\Basic;**

**use think\Request;**

**class BasicAuth extends Basic{**

**/\*\***

**\* 获取用户信息后 验证权限,**

**\* @param Request $request**

**\* @return bool**

**\*/**

**public function certification(Request $request){**

**return ($this->username == 'test' && $this->password == 'test') ? true : false;**

**}**

**/\*\***

**\* 获取用户信息**

**\* @return mixed**

**\*/**

**public function getUser(){**

**// TODO: Implement getUser() method.**

**}**

**}**

**Oauth客户端模式**

namespace app\index\auth;

use Dawnapi\auth\OAuth;

use RandomLib\Factory;

use think\Cache;

use think\Request;

class OauthAuth extends OAuth

{

/\*\*

\* 客户端获取access\_token

\* @param Request $request

\* @return \think\Response|\think\response\Json|\think\response\Jsonp|\think\response\Redirect|\think\response\Xml

\*/

public function accessToken(Request $request){

//获客户端信息

try {

$this->getClient($request);

} catch (UnauthorizedException $e) {

//错误则返回给客户端

return $this->sendError(401, $e->getMessage(), 401, [], $e->getHeaders());

} catch (\Exception $e) {

return $this->sendError(500, $e->getMessage(), 500);

}

//校验信息

if ($this->getClientInfo($this->client\_id)->checkSecret()) {

//通过下放令牌

$access\_token = $this->setAccessToken($this->clientInfo);

} else {

return $this->sendError(401, 'authentication Failed', 401, [], ['WWW-Authenticate' => 'Basic']);

}

return $this->sendSuccess([], 'success', 200, [], [

'access\_token' => $access\_token, //访问令牌

'expires' => self::$expires, //过期时间秒数

]);

}

/\*\*

\* 校验密码

\* @return bool

\*/

public function checkSecret(){

if ($this->secret == $this->clientInfo['secret']) {

return true;

} else {

return false;

}

}

/\*\*

\* 获取用户信息后 验证权限

\* @return mixed

\*/

public function certification(){

if ($this->getAccessTokenInfo($this->access\_token) == false) {

return false;

} else {

return true;

}

}

protected function getAccessTokenInfo($accessToken){

$keys = self::$accessTokenPrefix . $accessToken;

$info = Cache::get($keys);

if ($info == false || $info['expires\_time'] < time()) return false;

//验证索引是否正确

$client\_id = $info['client']['client\_id'];

if ($this->getAccessTokenAndClient($client\_id) != $accessToken) return false;

$this->clientInfo = $info['client'];

return $info;

}

protected function getAccessTokenAndClient($client\_id){

return Cache::get(self::$accessTokenAndClientPrefix . $client\_id);

}

/\*\*

\* 返回用户信息

\* @return array

\*/

public static function getUserInfo(){

return [

'client\_id' => '20882088',//app\_id

'secret' => 'nGk5R2wrnZqQ02bed29rjzax1QWRIu1O',

'name' => 'test\_client'];

}

/\*\*

\* 获取客户端所有信息

\* @param $client\_id

\* @return mixed

\*/

public function getClientInfo($client\_id){

// todo 通过客户端$client\_id 获取所有信息

// dump($client\_id);

$this->clientInfo = self::getUserInfo();

return $this;

}

/\*\*

\* 设置AccessToken

\* @param $clientInfo

\* @return int

\*/

protected function setAccessToken($clientInfo){

//生成令牌

$accessToken = self::buildAccessToken();

$accessTokenInfo = [

'access\_token' => $accessToken,//访问令牌

'expires\_time' => time() + self::$expires, //过期时间时间戳

'client' => $clientInfo,//用户信息

];

self::saveAccessToken($accessToken, $accessTokenInfo);

return $accessToken;

}

/\*\*

\* 生成AccessToken

\* @return string

\*/

protected static function buildAccessToken(){

//生成AccessToken

$factory = new Factory();

$generator = $factory->getMediumStrengthGenerator();

return $generator->generateString(32, '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ');

}

/\*\*

\* 存储

\* @param $accessToken

\* @param $accessTokenInfo

\*/

protected static function saveAccessToken($accessToken, $accessTokenInfo){

//存储accessToken

Cache::set(self::$accessTokenPrefix . $accessToken, $accessTokenInfo, self::$expires);

//存储用户与信息索引 用于比较

Cache::set(self::$accessTokenAndClientPrefix . $accessTokenInfo['client']['client\_id'], $accessToken, self::$expires);

}

/\*\*

\* 获取用户信息

\* @return bool

\*/

public function getUser(){

$info = $this->getAccessTokenInfo($this->access\_token);

if ($info) {

$this->client\_id = $info['client']['client\_id'];

$this->user = $info['client'];

return $this->user;

} else {

return false;

}

}

}

**控制器Auth类获取acesstoken**

namespace app\index\controller;

use app\index\auth\OauthAuth;

use think\Request;

class Auth{

public function accessToken(){

$request = Request::instance();

$OauthAuth = new OauthAuth();

return $OauthAuth->accessToken($request);

}

}

## 四，自动生成文档

1. **创建Doc文档显示控制器 wiki 继承 Dawnapi\facade\Doc;**
2. **配置路由**

'wiki'=>index/Wiki/index',//文档

**3.配置文档显示目录**

/\*\*

\* 获取文档

\* @return mixed

\*/

**public** **static** **function** **getApiDocList**()

{

//todo 可以写配置文件或数据

$apiList = Config::get('api\_doc');

**return** $apiList;

}

**可以改写次方法以存储以无限级的方式，为了方便采用的是配置方式**

**tp5 增加额外配置 创建application/extra/api\_doc.php 文件**

**return** [

'1' => ['name' => '测试文档', 'id' => '1', 'parent' => '0', 'class'=>'','readme' =>''],//下面有子列表为一级目录

'2' => ['name' => '获取权限', 'id' => '2', 'parent' => '1', 'class'=>'','readme' => '/doc/md/auth.md'],//没有接口的文档，加载markdown文档

'3' => ['name' => '用户接口', 'id' => '3', 'parent' => '1', 'readme' => '','class'=>\app\test\controller\User::class],//User接口文档

];

| **参数** | **必须** | **备注** | **作用** |
| --- | --- | --- | --- |
| name | true | 接口列表名称 | 显示列表名称 |
| id | true | 主键 | 生成列表所用 |
| parent | true | 生成列表所用 |  |
| class | true | 接口位置 | 用于生成具体接口文档 |
| readme | true | markdown | 可以生成没有接口的文档，比如一些说明 module和controller为空,readme填文档路径 |

**3.具体接口文档配置**

**接口描述部分(类文件的注释)**

/\*\*

\* Class User

\* @title 用户接口

\* @url /v1/user

\* @desc 有关于用户的接口

\* @version 1.0

\* @readme /doc/md/user.md

\*/

**class** **User** **extends** Base{}

| **参数** | **必须** | **备注** | **作用** |
| --- | --- | --- | --- |
| title | true | 接口标题 | 显示列表名称 |
| url | true | 请求地址 | 用户显示 |
| desc | true | 接口描述 | 显示描述 |
| version | false | 版本号 | 版本号 |
| readme | false | markdown文档 | 可以编写信息文档 |

**五，Api请求处理**

1. **接口描述信息(注释填写)**

/\*\*

\* @title 获取用户信息

\* @desc 获取用户信息

\* @readme /doc/md/method.md

\*/

**public** **function** **getResponse**(\think\Request $request){}

| **参数** | **必须** | **备注** | **作用** |
| --- | --- | --- | --- |
| title | true | 接口标题 | 显示列表名称 |
| desc | true | 接口描述 | 显示描述 |
| readme | false | markdown文档 | 可以编写信息文档 |

**2.请求参数**

/\*\*

\* 参数规则

\* @name 字段名称

\* @type 类型

\* @require 是否必须

\* @default 默认值

\* @desc 说明

\* @range 范围

\* @return array

\*/

**public** **static** **function** **getRules**()

{

$rules = [

//共用参数

'all'=>[

'time'=> ['name' => 'time', 'type' => 'int', 'require' => 'true', 'default' => '', 'desc' => '时间戳', 'range' => '',]

],

'get'=>[

'id' => ['name' => 'id', 'type' => 'int', 'require' => 'true', 'default' => '', 'desc' => '用户id', 'range' => '',]

],

'post'=>[

'username' => ['name' => 'username', 'type' => 'string', 'require' => 'true', 'default' => '', 'desc' => '用户名', 'range' => '',],

'age' => ['name' => 'age', 'type' => 'int', 'require' => 'true', 'default' => '18', 'desc' => '年龄', 'range' => '0-200',],

]

];

//合并父级类参数

**return** array\_merge(**parent**::getRules(),$rules);

}

## 请求字段

| **参数** | **必须** | **备注** | **d作用** |
| --- | --- | --- | --- |
| time | true | 接口标题 | 显示列表名称 |
| id | true | 接口描述 | 显示描述 |

1. **返回参数(注释填写)**

\* @**return** int id ID

\* @**return** string username 错误信息

\* @**return** int age 年龄

| **参数** | **必须** | **备注** |
| --- | --- | --- |
| 第一个参数 | true | 类型 |
| 第二个参数 | true | 参数名 |
| 第三个参数 | true | 参数说明 |

类型填写规则

'string' => '字符串',

'int' => '整型',

'float' => '浮点型',

'boolean' => '布尔型',

'date' => '日期',

'array' => '数组',

'fixed' => '固定值',

'enum' => '枚举类型',

'object' => '对象',

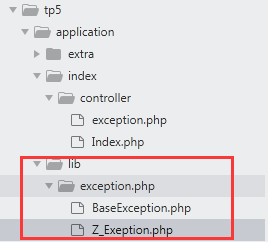
| **参数** | **类型** | **备注** |
| --- | --- | --- |
| id | 整型 | 类型 |
| usernmae | 字符串 | 参数名 |
| age | 整型 | 参数说明 |

**各种特殊情况处理方法：**

**一、自定义异常处理类**

**在项目的开发过程中异常抛出尤为重要不仅能够做出友好提示，还能做到提示开发人员代码白编写的错误，下面进行自定义异常抛出类**

**首先在框架中可以自定义目录结构用来做异常类的存储位置例如文件目录为以下红框中**



**定义目录结构后阐述一下异常类的工作流程，见名知意既然是异常抛出那么只有在代码出现问题的时候或者是逻辑出现异常的时候会进行抛出，那么一般就会出现两种情况，**

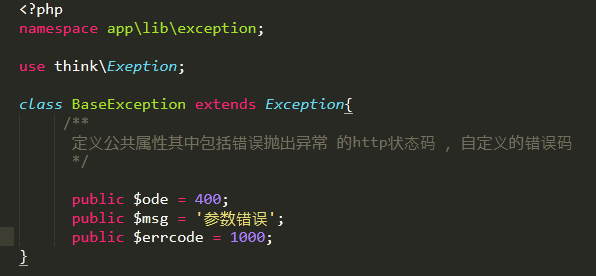
**第一种是用户传输的参数或者用户操作的流程有问题这种情况下并不是的开发的代码或者是业务逻辑出现的致命错误，直白的将这种情况就是用户的请求没有返回相应的数据**

**第二种是的代码存在问题，或者是服务器上面的问题，但是这个问题是不需要用户知道的，在这种情况下即需要给出用户的友好提示，还需要知道的代码具体是哪里出现了问题所以这种情况下就需要记录日志**

**分析了两种情况之后首先创建应对第一种情况的类，命名为BaseException.php在这个类中需要定义的很简单就是对应第一种情况的友好信息**

**信息中包括的是抛出异常的http状态码、异常的信息、自定义异常的状态码(可选)**

**但是BaseException.php需要继承Exception虽然是自己定义的异常类但是还是需要遵循规律，既然要抛出异常，就需要有捕获异常，BaseException.php代码如下**



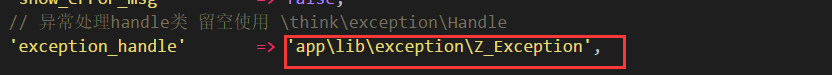


**这是官方手册告诉我们接下来要做的:**

**定义Z\_Exception.php用来继承Handle类并且重写render方法，这样就可以将自定义的信息进行抛出，并且在这个类中穿插判断用来应对第二种异常的情况**



**至此存在一个值得注意的问题在config.php中的问题**



**本身此处为空但已经自己定义了异常类，**



**处理之后继续下一步添加日志**

**通过入口文见index.php可以追踪到框架的核心文件start.php打开之后看到加载base.php然后打开base.php就可以看到文件的配置项找到log查看文件的路径**

https://images2017.cnblogs.com/blog/1001252/201708/1001252-20170816180223396-472847666.png

**由此可以定位到log文件的路径**

**由此更改一下log的路径方便以后管理，在index.php中重新定义LOG\_PATH的路径**

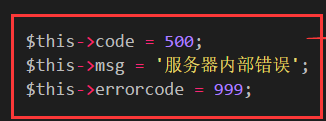
https://images2017.cnblogs.com/blog/1001252/201708/1001252-20170816181140490-809934478.png

**更改文件的路径后应该将框架中默认的记录日志的方式进行更改可以参考官方的文档**

**https://www.kancloud.cn/manual/thinkphp5/118126**

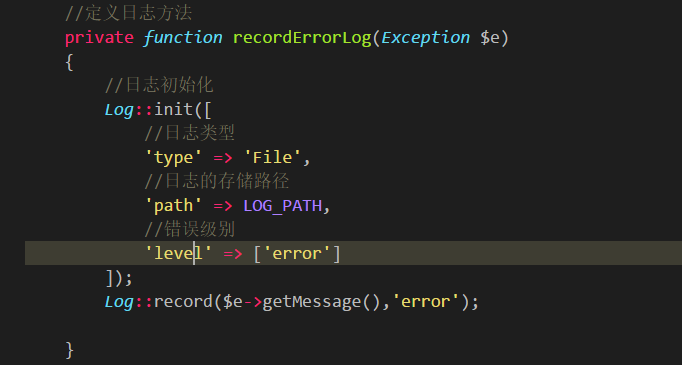
**更改框架中的日志的记录方式就是在config.php查找Log模块，将日志的type由File更改为test**

**既然日志是为了应对生产环境下的问题那么我在代码编写的时候就应该在**

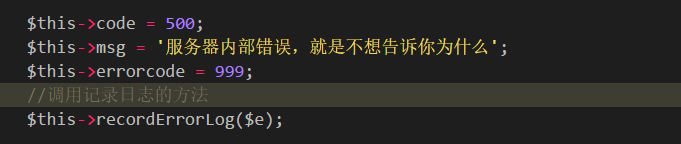
****

**之后进行日志的记录，但是要是进行日志的记录就应该使用框架中的Log类**

**在Z\_Exception.php中定义方法用来写入日志**



**然后再调用**



**这样就完成了自定义的类和自定义的日志文件**

二、**保障接口调用的安全**

**在tp5开放接口中，后台服务器在用户登录后如何去验证和维护用户的登陆有效性就需要开放接口安全验证，如token验证**

**这里我们就要用到加解密，签名认证**

## 签名设计:

**对于敏感的api接口，需使用https协议**

**https是在http超文本传输协议加入SSL层，它在网络间通信是加密的，所以需要加密证书。  
https协议需要ca证书，一般需要交费**

**关于签名设计实现方式：**

## 签名的设计

**首先用户登录后向服务器提供用户认证信息（如账户和密码），服务器认证完后给客户端返回一个Token令牌，用户再次获取信息时，带上此令牌，如果令牌正取，则返回数据。对于获取Token信息后，访问用户相关接口，客户端请求的url需要带上如下参数：  
时间戳：times  
Token令牌：token  
然后将所有用户请求的参数按照字母排序（包括times，token），然后更具MD5加密，全部大写，生成sign签名，这就是 所说的url签名算法。然后登陆后每次调用用户信息时，带上sign，times，token参数。  
  
例如：原请求https://www.hyhf.com/index.php/index/user/city/北京 （post和get都一样，对所有参数排序加密）  
  
加上时间戳和token**[**https://www.hyhf.com/index.php/index/user/city/北京&times=12312311233&token=wefkfjdskfjewfjkjfdfnc**](https://www.hyhf.com/index.php/index/user/city/北京&times=12445323134&token=wefkfjdskfjewfjkjfdfnc)**然后根据url参数生成sign  
  
最终的请求:**[**https://www.hyhf.com/index.php/index/user/city=北京&times=12312311233& token=wefkfjdskfjewfjkjfdfnc&sign=FDK2434JKJFD334FDF2**](https://www.hyhf.com/index.php/index/user/city=北京&times=12312311233&%20token=wefkfjdskfjewfjkjfdfnc&sign=FDK2434JKJFD334FDF2)**其最终的原理是减小明文的暴露次数；保证数据安全的访问。  
具体实现如下：  
1. api请求客户端想服务器端一次发送用户认证信息（用户名和密码），服务器端请求到该请求后，验证用户信息是否正确。  
如果正确：则返回一个唯一不重复的字符串（一般为uid），然后在redis（任意缓存服务器）中维护Token-uid的用户信息关系，以便其他api对token的校验。  
如果错误：则返回错误码。**

**2.服务器设计一个url请求拦截规则  
（1）判断是否包含times，token，sign参数，如果不含有返回错误码。  
（2）判断服务器接到请求的时间和参数中的时间戳是否相差很长一段时间（时间自定义如半个小时），如果超过则说明该 url已经过期（如果url被盗，他改变了时间戳，但是会导致sign签名不相等）。  
（3）判断token是否有效，根据请求过来的token，查询redis缓存中的uid，如果获取不到这说明该token已过期。  
（4）根据用户请求的url参数，服务器端按照同样的规则生成sign签名，对比签名看是否相等，相等则放行。（自然url签名 也无法100%保证其安全，也可以通过公钥AES对数据和url加密，公钥加密私钥签名，这样如果可以确保公钥最大可能不会丢失，所以签名是很大程度上保证安全）。  
（5）此url拦截只需对获取身份认证的url放行（如登陆url），剩余所有的url都需拦截。  
  
3.Token和uid关系维护  
对于用户登录需要创建token\_uid的关系，用户退出时需要需删除token\_uid的关系。**

**上面的设计模式可以保障session根据token加密后传输，为接口数据传输提供了安全高效的保障。**

**三、sql注入**

**当所有用于建立一个SQL语句的数据被正确过滤和转义时，实际上也就避免了SQL注入的风险。如果你正在使用支持参数化查询语句和占位符的数据库操作类（如PEAR::DB, PDO等），数据就会多得到一层保护。**

**四、安全攻击(cc/dos/ddos)**

**1，在程序里面通过设置时间间隔和刷新次数来避免cc攻击。还有就是判断通过ip判断行为符合cc行为就exit，就算已经识别到了目标ip是cc源，程序也exit了，但是依然经过web服务器到fpm再到程序，资源也会消耗，这样做会相对少一些。所以最好就是结合服务器，比如识别到cc的ip，再添加到ip tables的黑名单，这样就从服务器入口屏蔽。**

**2，防止dos和ddos攻击要在主机和网络设备上限制syn链接数量并且使用大带宽和存储量的服务器，数据和程序的定时备份，版本补丁更新，漏洞修改，可以通过版本的更新迭代来防止攻击。**

**3，使用负载均衡来把服务器数据分层处理，减少单一服务器的压力。**

**4，在服务器资源充足的情况下采用服务器集群分布式部署，这种接力式的数据处理机制以及反弹式处理方法可以更好地址ddos攻击。**

**五，如何防止输入安全**

**1,设置public目录为唯一对外目录，不要把资源文件放入应用目录；**

**2,开启表单令牌验证避免数据的重复提交，能起到CSRF防御作用；**

**3,使用框架提供的请求标量获取方法(Request类param方法及input 助手函数)而不是原生系统变量获取用户输入；**

**4,对不同的应用需求设置default\_filter过滤规则，行间的安全过滤函数包括stripslashes、htmlentities、htmlspecialchars和strip\_tags等，请根据业务场景选择最合适的过滤方法；**

**5,使用验证类或者验证方法对业务数据设置必要的验证规则；**

**6,如果可能开启强制路由或者设置MISS路由规则，严格规范每个URL请求;**