1.app接口的介绍

客户端与服务端数据的交换.

2.app接口输出格式

code 错误码

msg 错误码对应的描述

data接口返回的数据

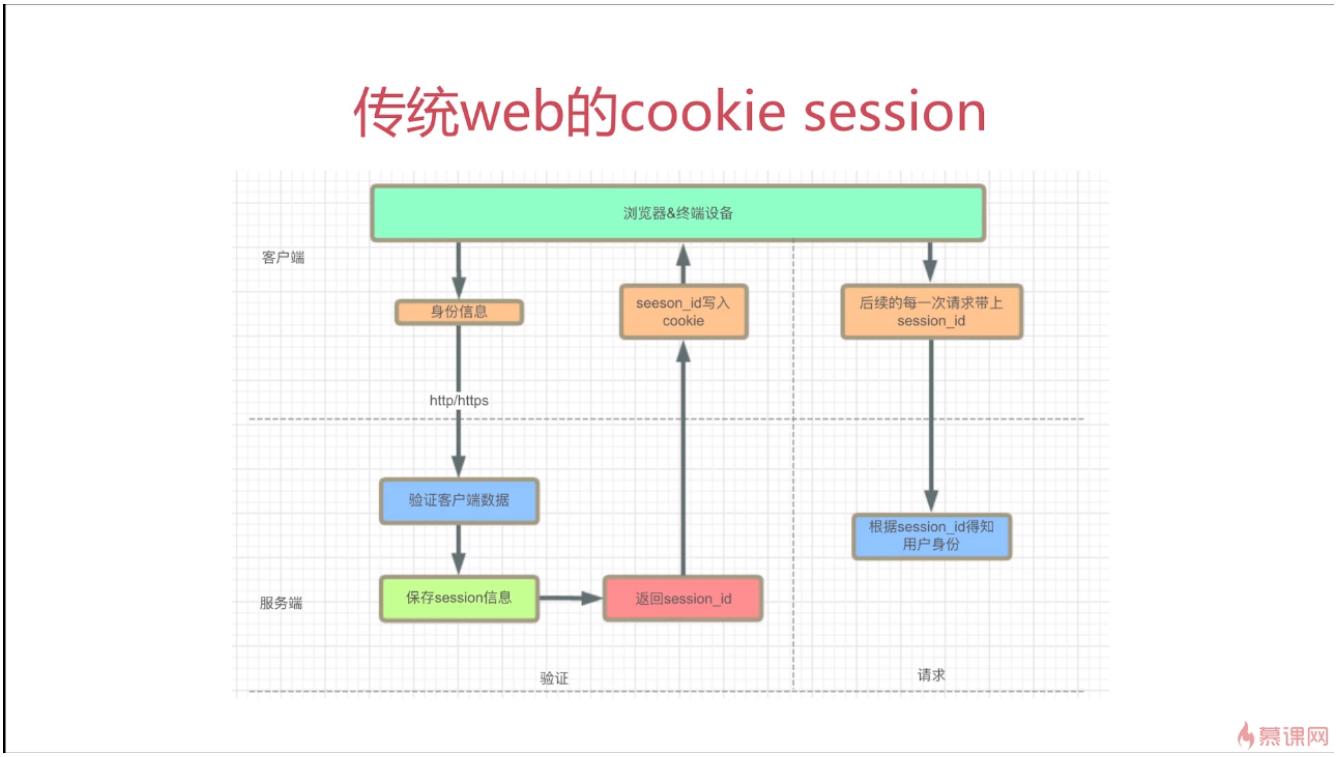
```
trait ResponseJson{
    //接口报错时调用
    public function jsonData($code,$message,$data){
        return $this->jsonResponse($code,$message,$data);
    }
    //接口成功时调用
    public function jsonSuccessData($data=[]){
        return $this->jsonResponse(0,'Success',$data);
    }
    private function jsonResponse($code,$message,$data){
        $content=[
            'code'=>$code,
            'msg'=>$message,
            'data'=>$data
        ]
    }
}
```

```
use ResponseJson;
return $this->jsonSuccessData(['abc']);
```

3.app接口的鉴权

客户端需要带着凭证来调用app接口

3.1传统web



3.2app接口的宠儿jwt(JSON Web Token)

JWT原理

服务器认证以后,生成一个json对象,返回给用户

3.2.1组成

- header通常包含了两部分:类型和采用的加密算法

```
{
"alg":"HS256",
"typ":"JWT",
}
```

  header需要经过Base64Url编码后作为JWT的第一部分

- payload包含了claim,三种类型reserved,public,private

  reserved这些claim是JWT预先定义的,不强制使用,常用的有:iss(签发者),exp(过期时间戳),sub(面向的用户),aud(接收方),iat(签发时间)

  payload需要经过base64url编码后作为JWT的第二部分

```
{

    "sub":"1234567890",

    "name":"John Doe",

    "admin":true

}
```

- signature创建签名使用编码后的header和payload以及一个密钥,使用header中指定的签名算法进行签名

```
HMACSHA256(base64UrlEncode(header)+"."+base64UrlEncode(payload),secret);
```

3.2.2完整的JWT(https://jwt.io/)

JWT格式输出的是以.分割的三段Basse64编码

```
eyjhbGciOJIUz11Nils1nR5cC161kpXVCJ9.eyjzdW1ioi1xMjMONTY3ODkw1iwibmFtZS11kpvaG4gRG911iw
iYWRtaW4iOnRydWV9.TJVA95ODcEfxj0YZgeFONFh7HgQ
```

3.2.3完整代码

请查看完整的laravel带啊

4.code data msg 定义

请查看app/common/Err/ApiErrDesc.php文件

5.异常处理

在使用jwt鉴权过程中不管是解码还是编码都可能抛出许多异常,为了避免在控制器里导出捕获异常(try cache),需要使用set_exception_handler set_error_handler方式统一捕获异常.

而在laravel框架中则是,由app/exceptions/handler.php(render方法)统一处理未捕获的异常.

首先自定义了一个app/exceptions/ApiException.php

```php
namespace App\Exceptions;

use Throwable;

class ApiException extends \RuntimeException
{
    public function __construct(array $apiErrConst, Throwable $previous = null)
    {
        $code=$apiErrConst[0];
        $message=$apiErrConst[1];
        parent::__construct($message, $code, $previous);
    }
}
```

然后再render方法中统一处理,异常

```php
public function render($request, Exception $exception)
    {
        //return parent::render($request, $exception);
        if($exception instanceof ApiException){
            $code = $exception->getCode();
            $message = $exception->getMessage();
        }else{
            $code = $exception->getCode();
            if(!$code || $code < 0){
                $code = ApiErrDesc::UNKNOWN_ERR[0];
            }
            $message = $exception->getMessage() ? : ApiErrDesc::UNKNOWN_ERR[1];
        }
        return $this->jsonData($code, $message);
    }
```

这样做后,我们在开发中我们就只需要抛出异常就可以了.

```php
$token = $request->input('token');
        if($token){
            $jwtAuth = JwtAuth::getInstance();
            $jwtAuth->setToken($token);

            if($jwtAuth->validate() && $jwtAuth->verify()){
                return $next($request);
            }else{
                //return $this-
>jsonData(ApiErrDesc::ERR_TOKEN[0],ApiErrDesc::ERR_TOKEN[1]);
                throw new ApiException(ApiErrDesc::ERR_TOKEN);
            }

        }else{
            //return $this->jsonData(ApiErrDesc::ERR_PARAMS[0],ApiErrDesc::ERR_PARAMS[1]);
            throw new ApiException(ApiErrDesc::ERR_PARAMS);
        }
```