# 编程最重要的是理解思想

视屏看到10.13完

视频看到10.26

# 1.路由：

**固定的模式不能瞎写**：**（模块名/控制器名/方法名）**

例如： Route::get(‘banner/:id’, ‘api/Banner/getBanner’);

**注意：自己在controller文件夹下加入v1文件夹 该怎么写：**

**v1.Banner**

‘api/v1.Banner/getBanner’

**有混合模式，pathinfo模**式，强路由模式

默认是混合模式 url\_route\_on => true; 【route 和 pathinfo 路由都能用】

Pathinfo 模式 url\_route\_on => false [route 路由不能用]

强路由模式 url\_route\_must => true pathinfo 路由不能用；

## 推荐使用强路由模式

## Route::get( ‘形参’ , ’模块名/控制器名/方法名’, ‘array()[限制条件]’’);

1.Config.php 设置： url\_route\_must'=>true, url\_route\_on' => true,

2.Route.php 设置：引入Route类 think\Route;

调用Route类中方法：Route::rule()

**[GET POST PUT DELETE \* 代表支持所有]**

同时支持GET POST**[GET|POST]**

Route::rule(‘hello’,‘example/Index/index’, ‘POST|GET’);

例子： Route:: rule(‘hello/:id’, ‘example/Index/index’, ‘GET’, [‘https’ => true] );

**Route.php 文件中路由用 ‘/’ , 不能用‘\’**。

3.快捷方式**： Route::get(‘hello’, ‘example/Index/index’)**

Route::post

Route::any() any() 就相当于上面的 \* ，任意请求都行

# 2.获取参数：

引入请求类：Think\Request;

## 2.1实例化Request 类 ，调用param 方法

$id = Request : : instance() -> param(‘id’);

param不写参数就是获取所有值：

$all = Request::instance() -> param();

同理还有:

get() 获取get请求中？后的值。

route() 获取tp5路由设定的值

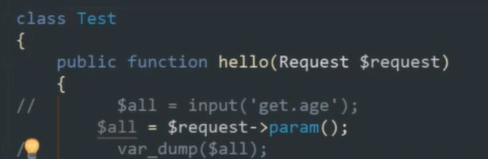
post() 获取post传值

## 2.2简化：

input()助手函数不用引入类，直接写input ,可以写成input(), input(get.id), input(‘post.id’)

## 3.依赖注入：

不用关心到底是谁实例化了Request , 只要你写了Request $request 就可以直接调用param 方法



# 3.验证validate：

验证器，独立验证

## 1.Check:

**Validate -> check 的时候 会调用 $rule 验证规则变量**

## 2.例子：

**new Validate( [‘name’ => ‘require|max:10’, ‘email’ => ‘email’ ] ) ->batch() -> check();**

1. 引入Validate 类
2. 实例化Validate ，调用批量验证batch()方法 ，再调用check()方法验证
3. 如果写require , url参数必须要传，不传会报错。如果不写require 可以不传参数， 用默认的参数。
   * 1. 例子：
        1. Portected $rule = ‘isPositiveInteager’; 和
        2. $rule = **‘require|** isPositiveInteager’;

# 4.全局异常处理:

## 1.普通异常处理方法：

**try** {  
 *BannerModel*::*getBannerInfo*($id);  
} **catch** (Exception $ex) {  
 $err = [  
 **'error\_code'** => **'10001'**,  
 **'msg'** => **$ex->getMessage(),**  
 ];  
 **return** json($err, **400**);  
};

## 2.分析:

### 由于用户导致的异常

返回异常信息，不写入日志

### 内部服务器错误导致的异常（代码错误， 调用外部api出错）

不返回异常，写入日志。

## 3.自定义全局异常：

**throw 抛出异常 到 handle 类**

**有开发模式 和 生产模式**

1.config.php 中 **exception\_handle = ‘’** 填写自己的全局异常处理类地址。

2. 自定义的全局异常处理类ExceptionHandler，继承handle 类， 重写 render 方法

3. 根目录创建 error\_code.txt 问价 用于记录错误码

4. 创建BaseException 类 继承 think\Exception 类， 覆盖4个成员变量 $error\_code ， $msg , $url , $code.

5.在自定义全局异常处理类。ExceptionHandler.php， 判断是不是自定义异常类BaseException 的实例,如果是，写入自定义类定义的成员变量

**public function** render(Exception $e)  
{  
 *//判断是否是自定义异常处理类* **if** ($e **instanceof** BaseException) {  
 $this->**code** = $e->**code**;  
 $this->**error\_code** = $e->**error\_code**;  
 $this->**msg** = $e->**msg**;  
 }  
 $request = Request::*instance*();  
  
 $result = [  
 **'msg'** => $this->**msg**,  
 **'error\_code'** => $this->**error\_code**,  
 **'url'** => $request->url(),  
 ];  
 **return** json($result, $this->**code**);  
}

# 5.日志系统：

## 1.自定义日志系统：

1. 关闭tp内置日志系统初始话type值File改为test。 config.php -> log -> ‘type’ => ‘test’
2. 在index.php 入口文件 定义log地址 常量 LOG\_PATH ;
3. 自定义日志方法中要开启日志初始化。Log::init([ ‘type’=>’file’, ‘path’=>LOG\_PATH])

我们在开发模式中开启了sql 日志（线上模式关闭）：

在index.php 入口文件中设置：

Log::init(

‘type’ => ‘File’,

‘path’ => LOG\_PATH,

‘level’ => [‘sql’],

);

# 6.数据库操作：

## 6.1 Db类：

Db 类是模型类的基础，模型类处理业务逻辑比较强

Db 类返回数组， 模型返回对象，方便操作。

**Db类** -》 链式方法 - 》 执行方法

**链式方法** 顺序没有规定

例子：Db::table(‘user’)->where(‘name’ ,’’=, ’jack’)->select();

**非链式方法:**

Db::table(‘user’);

Db::where(‘’);

Db::select();

**闭包方法：**

Db::table(‘user’)

->where( function **($query) use ($id)**{

$query->where(‘id’ , ‘=’, ’$id’);

})

->find();

## 6.2 Orm:

DB可以调用: get all

模型可以调用： get find all select 例如： BannerModel :: find();

DB类是模型的基础 模型是处理业务 DB类是直接操作数据库

**get find select all**

**阅读**：

一般情况下：一个模型model 对应一张表名， 模型model 名和表名一样。

关联模型的时候： 一个模型对应多张数据表

如果模型名和数据表名不一样，可以在 模型model 中指定 数据表名 ：

Protected $table = ‘数据表名’;

1. model名 和数据表名一样，调用相对应模型方法等于直接操作对应的数据表。
2. model调用数据库返回 数据类型为**对象**。
3. Tp5 输出数据时会自动把**对象**转换成json体， 但是是类型还是 text/html;
4. Config.php 中更改数据返回类型 为json

## 6.3关联模型(连表查询)：

BelongsTo hasMany hasOne belongsToMany

### 表与表的关系：

#### 一对多：

hasMany()

#### 多对多关系：

嵌套关联查询：（三表以上查询）

一个model关联另一个model（model中定义方法关联另一个model）， 另一个model再关联第三个model，在controller中调用with 调用关联三个model的两个方法。

正确的做法：必须要用到中间表 ( 一共三张表 )。

错误的做法：在表的字段里写另一张表的id号用逗号或别的符号分割开。

例如 ‘1,2,3,4,5,6’

**belongsToMany**('关联模型名','中间表名','中间表和关联模型表的关联字段',

'在中间表中与当前模型关联的键名',['模型别名定义']);

#### 一对一的关系（几种关系中最难理解的）：

两张表，一对一关系，外键在哪张表上,在模型中关联用belongsTo( )。

当前表没有外键， 用 hasOne( );

## 6.4采用数据集还是数组：

模型操作数据库获取数据默认为array , 但是要变为‘数据集’可以有很多方法可以调 用，增加了可扩展性。

比如：只需要在某一个控制器接口隐藏某一字段。

方法：

更改database.php 配置;

// 数据集返回类型

'resultset\_type' => 'collection',

在数据集中调用hidden 方法隐藏字段：

$result->hidden(‘update\_time’);

判空 isEmpty()

## 6.5隐藏字段：

$obj->toArray() 转为数组，再用unset删除

$obj->hidden()隐藏字段 , $obj->visible() 只显示字段

Model里重写$hidden 成员属性 $visible() 成员属性

# 7.定义自己的配置项：

Tp5会自动加载 extra[额外的] 文件夹下的配置文件；

5.0.1以上版本支持自动读取扩展配置文件（extra\_config\_list配置参数废 弃），只需要将扩展配置文件放入application/extra目录，即可自动读取。

用config()助手函数获取配置： config（配置文件名称.数组名称）

config(‘setting.prefix\_image’);

# 8.获取器：

## 方法：

get字段名称Attr()

例如url字段数据：getUrlAttr()

参数： getUrlAttr(‘当前url数值’, ‘整个data数据’);

定义：

获取器方法的第二个参数传入的是当前的所有数据数组

获取器只有当获取某个数据属性的时候自动触发

参考地址：

https://www.kancloud.cn/manual/thinkphp5/135192

# 9.[思想]：

## 多版本支持:

对老版本的支持：是给用户升级新版本的一个过渡。不会支持太多的版本，维护成本太 高

更改路由：[改成灵活路由]

* 1. Route::get(‘banner/**:version**/getbanner’, ‘api**/:version**/getbanner’);

## 关于嵌套关联模型的思考：

**$hidden $visible 的作用**

1. 当前模型关联另个一模型，另一个模型又关联第三个模型的时候，字段都归到第一 个模型下面， 在第一个模型中如果用了$visible方法，后面模型的字段就会不显示， 只显示$visible 定义的字段。**因此不用$visible 最好，改用 $hidden 成员属性**
2. **关联模型返回结果是model 类，不能用isEmpty() 判断是否为空**

## 参数过滤：

验证客户端传递的参数是否符合要求：

1. 客户端不能传递‘user\_id’ ‘uid’ 等user表在其他表的外键,如果传递即为恶意请求。抛出异常；
2. 按照validate $rule成员属性，验证的参数来取客户端传递的参数。

# Token：

存入缓存数据库中

规定token 放在heard 头中传递

1.客户端发送code 码到后台，后台拿到code码 请求微信Api 得到session\_key 和 open\_id。

2.数据库查询openid 是否存在，如果不存在新增数据， 存在记录uid

3.生成token , 将微信返回数据，以及uid openid , scope 权限范围 存入缓存数据.

4.返回客户端token

## 应用场景：

如果不用token而是用body 携带id 参数传参:

假如a,b 用户都是合法用户，a用户调用修改地址接口传b 用户的id ,就会将B用户的 数据修改。

# 11.公共文件：

位置：App\common.php

作用：在这里编写的方法可以被其它地方直接调用

## 12.不明白：

## 12.1闭包构建查询器，完成排序static public function getProductByID($id) { return self::*with*([ 'productImage' => function ($query) { $query->with('imgUrl') ->order('order', 'asc'); }]) ->with(['property']) ->find($id); }

## 12.2：模型的关联模型操作数据库

新增：$user->Address()->save();

更新：$user->Address->save();

# 13权限控制：

//@check 方法前写... python 中叫装饰器

TP5自带前置方法

beforeActionList = [];

参考地址：

https://www.kancloud.cn/manual/thinkphp5/118050

用法：

[‘前置方法名’=》 [‘only’ => ‘要限制的方法名’] ];

# 订单接口：【重点】

订单商品与数据库商品对比

检测库存量

生成订单快照

存入数据库

复杂的数据库操作 加入 try catch

## 模型新增：

Url : https://www.kancloud.cn/manual/thinkphp5/135188

save

save方法新增数据返回的是写入的记录数。不是主键id

获取主键id

$user->save();// 获取自增ID

echo $user->id;

$user = new User;

$user->name = 'thinkphp';

$user->email = 'thinkphp@qq.com';

$user->save();

新增多条数据：saveAll

V5.0.13+版本开始，saveAll方法的返回类型受模型的resultSetType属性影响（可能返回数据集对象）。

saveAll方法新增数据默认会自动识别数据是需要新增还是更新操作，当数据中存在主键的时候会认为是更新操作，如果你需要带主键数据批量新增，可以使用下面的方式：

$user = new User;

$list = [

['id'=>1, 'name'=>'thinkphp', 'email'=>'thinkphp@qq.com'],

['id'=>2, 'name'=>'onethink', 'email'=>'onethink@qq.com'],];

$user->saveAll($list, false);

关于一对多的新增操作：

多对多分拆成一对多，先保存一，再保存多。

# 自动写入时间戳：

1.TP5自动检测是删除操作还是新增，在delete\_time ,create\_time 中自动写入

2.只有在模型中操作才会有

3.自定义创建时间字段

在model中定义 protected $create\_time = ‘Create\_timeStamp’;

4.在model 中写 protected $autoWriteTimeStamp = true;

# 事务：

在TP5中使用事务

开启事务 Db::startTrans();

提交事务： Db::commit();

回滚事务： Db::rollback();

# 微信支付：

准备：

商户号

商户密码

参数检测：

订单不存在

订单用户和当前用户不是同一人

该订单已经被支付过

检测订单库存

下载微信支付服务器sdk

1.网页搜索

微信支付开发者文档

2.地址url :

<https://pay.weixin.qq.com/wiki/doc/api/external/jsapi.php?chapter=11_1>

**请求微信Api获取微信预订单：**

$wxOrderData = **new** \WxPayUnifiedOrder();  
$wxOrderData->SetTotal\_fee($orderPrice \* 100);  
$wxOrderData->SetOut\_trade\_no($this->**orderNO**);  
$wxOrderData->setTrade\_type(**'MWEB'**);  
$wxOrderData->SetBody(**'全能宝商城'**);  
$wxOrderData->SetOpenid($openid);  
$wxOrderData->SetNotify\_url(**''**);

**return** $this->getPaySignature($wxOrderData);

请求微信API 获取微信签名

$jsApiPayData = **new** \WxPayJsApiPay();  
$jsApiPayData->SetAppid(config(**'wx.app\_id'**));  
$jsApiPayData->SetTimeStamp((string)*time*());  
$rand = *md5*(*time*(). *mt\_rand*(0,1000));  
$jsApiPayData->SetNonceStr($rand);  
$jsApiPayData->SetPackage(**'prepay\_id='**.$wxOrder[**'prepay\_id'**]);  
$jsApiPayData->SetSignType(**'md5'**);  
$sign = $jsApiPayData->MakeSign();  
$rawValues = $jsApiPayData->GetValues();  
**return** $rawValues;

返回客户端签名数据，客户端拉起微信支付弹窗

支付结果：

微信异步发送消息 给客户端，和服务器（服务器回调URL在请求微信API预订单的时 候填写）

微信回调通知处理：

微信返回的是xml 类型数据，我们直接调用WxPay sdk 可以转换成array 类型，更加 方便

回调通知方法做4件事

0.检测订单状态是否为 ”1” [未支付] 如果true ,走下面流程,如果不是不用检测库存， 节省服务器资源；

1.检测库存 [有两种情况：1.有库存更改订单状态，减去库存。2.没有库存更改订单状 态]

2.更改订单状态 [未支付，已支付，已发货，已支付没货]

3.减去库存

4.加入事务防止重复添加数据

**分页**

**接口： 我的订单接口**

**Tp5**

**paginate()**

**第三方登录：**

**Cotroller->service->model**

**验证账号密码是否正确**

**AppToken**

**getAppToken**

## 类库：

TP5 定义类库地址在

根目录 \ Extend 文件夹下 \配置文件夹

例如： \Extend\WxPay

**自动加载类：**

例如引入：获取\extend\Wxpay\Wxpay.Api.php 文件

就照着这种模式写，没有为什么

Loader：：import(‘目录名称 **.** 文件开头名称’，‘extend的更目录’，‘后面的文件名’);

Loader::import(‘WxPay.Wxpay’, EXTEND\_PATH , ‘.Api.php’ );

**CORS 跨域**

Google 认为请求分为两种 复杂请求 和 简单请求

1.复杂请求： google浏览器会先发送一个options 请求 ， 服务器正确相应之后会发送get[post] 请求。如果 options 请求不能正确响应，就会报错。Get[post] 请求不能正确响应和支持 也会报错。

2.简单请求使用正常的 get .post 请求 ， 不用处理options 响应，支持header头跨域即可。

处理跨域和options 请求

Tp5自带 “行为” 功能

<https://www.kancloud.cn/manual/thinkphp5/118130>

可以在controller 下创建 behavior\CORS.php 文件

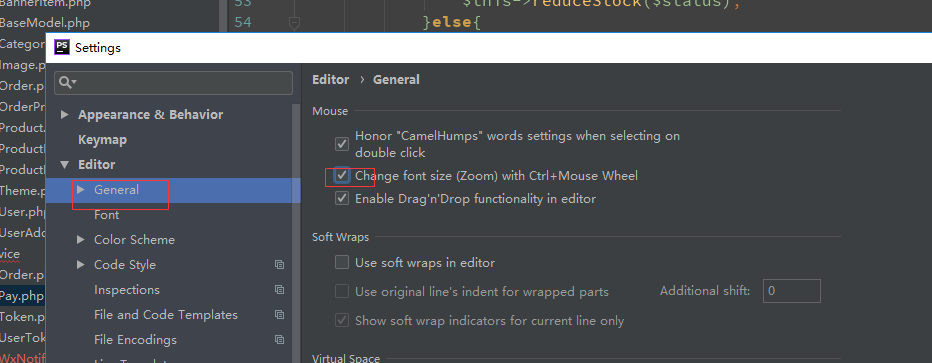
文件内容：

header('Access-Control-Allow-Origin: \*');  
header("Access-Control-Allow-Headers: token,Origin, X-Requested-With, Content-Type, Accept");  
header('Access-Control-Allow-Methods: POST,GET');  
**if**(request()->isOptions()){  
 **exit**();  
}

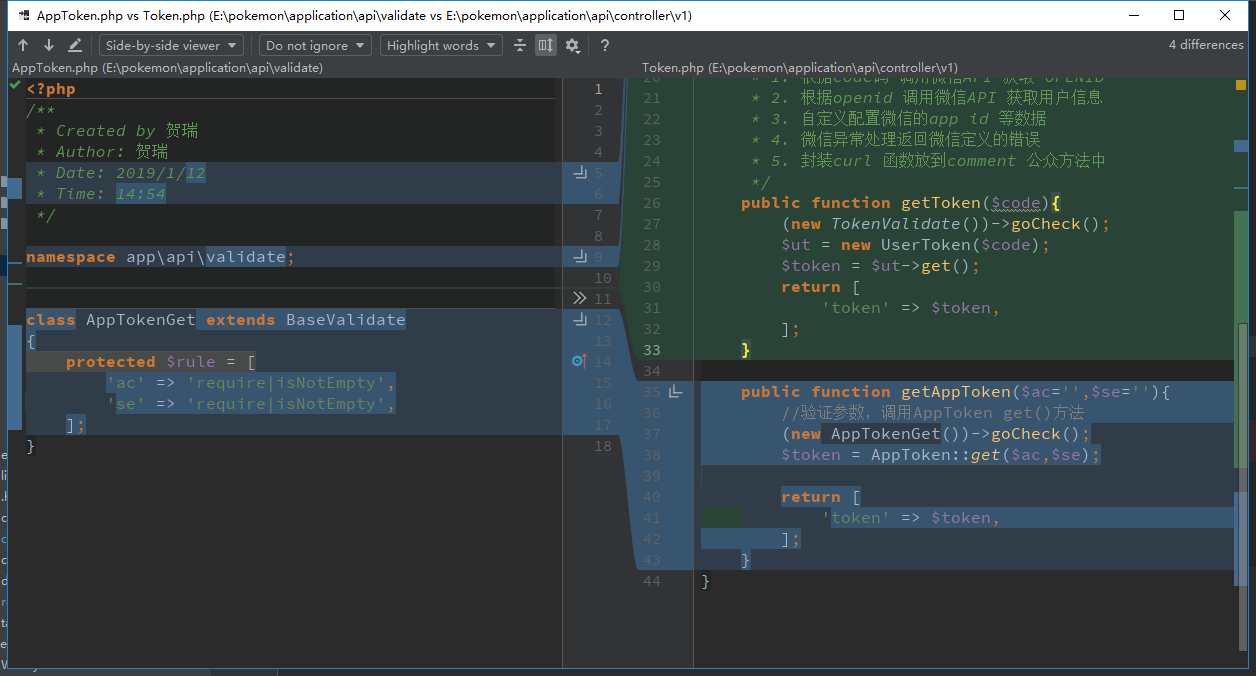
App\tag.php 中设置：

**return** [  
 // 应用初始化  
 'app\_init' => [  
 'app\\api\\behavior\\CORS'  
 ],

PhpStorm 用滚轮调整字体大小：



选中文件名 右键+m



# .细节：

Exception 成员属性要写public , 应为handle 要调用

validate 成员属性可以是protected 不能是private 。因为父类BaseValidate 要调用。不能是私有的，私有就调用不到

三次库存检测：

点击下订单时候

点击支付时候

支付完成时候

Requrie 写在类外面

没有命名空间 新建对象的时候要加 “\” 代表根目录

Input()助手函数 获取数组, 要在后面加 “/a”

示例： input( ‘post.id/a ’ );

做检测 把最可能发生的错误写在最外层能节约服务器性能。

数据库的写的速度大于读的速度

Uml 产品经理建模

避免循环查询数据库， 如 在foreach 中查询数据库。

一个方法最好不要超过50行，超过就是不合理。

Tp5 自带缓存 cache(‘key’, ’value’, ‘过期时间’);

JSON 用的是双引号 “”

开启路由完整匹配：**'route\_complete\_match'** => **true**,

重构代码 = 优化代码

模型对象转成数组 $user = User::find(1);dump($user->toArray());

注释的代码就删除

**代码的可读性**

**模型 比原生sql 慢**

**Tp5快速创建model小技巧**

**打开Terminal ， 输入代码： php think make:model api/模型名称**

Config() 助手函数 读取配置文件

1/0 分母不能为零，会报错

**代码的复用性**

堆栈信息

**Controller 中不处理业务**

**AOP 思想**

**ORM对象关系映射**

**面向对象， 面向过程**

**思想导图：**

验证层

异常处理

Controller

Model

Service

**RESTFul API 实践：**

**REST 提倡用json描述数据。**

**Json可以在不同的语言之间交换数据**

**意义： 可以用一种语言调用另外一种语言。**

**REST是基本思想和理论 。**

**RESTFul REST应用在web API 接口的设计里面，产生了RESTFul 风格的API**

**RESTFul 是REST在web 接口的应用和延伸**

**数据库代表资源**

**http 动词[GET, POST, PUT ,DELETE] 采用什么样的动词 取决于是对资源查询 还是新增**

**Url 通向资源**

**Url 不用动词**

**HTTP动词：**

**POST 创建**

**PUT 修改**

**DELETE 删除**

**GET 查询**

REST表示：

**状态码：读法不是五百，读五零零**

200 GET 成功

201 POST 成功

202 PUT 成功

401 没有权限

403 资源[接口] 被禁止 一个用户操作另一个用户的资源

500 服务器未知错误（不想让客户端知道具体错误）

错误码：

自定义错误id号

统一描述错误：

错误码

错误信息（概要信息）

当前URL

用token 来验证来授权和验证身份

版本控制 （v1）

测试和生产环境分开： api.xxx.com

dev.api.xxx.com

URL 语义要明确

有标准的文档

学习RESTFul API 的最好方式

模仿： 豆瓣API github