**application/config.php配置（模块配置文件）**

‘url\_convert’=>false， //可以实现url大小写敏感。

‘url\_route\_on’=>true, //是否开启路由

‘url\_route\_must’=>false, //是否强制使用路由

‘route\_complete\_match’=>false //路由使用完整匹配

‘app\_debug’=false, //关闭TP 5调试模式

‘type’=>’test’， //关闭TP 5默认记录日志的行为。

‘exception\_handle’=>’’ , // 异常处理handle类，留空使用\think\exception\Handle

‘exception\_handle’=>’app\lib\exception\ExceptionHandler’, //重新制定全局异常处理类

**/public/index.php配置（入口文件）**

define(‘LOG\_PATH’,\_\_DIR\_\_.’/../log/’); //修改日志路径。此方法提前定义了 thinkphp/base.php中LOG\_PATH的值。config.php中使用了LOG\_PATH。

**application/database.php配置（数据库配置）**

‘resultset\_type’=>’collection’，// 修改默认数据集返回类型为collection，而不是array

**第1章 不同的时代，不同的Web**

**1-1 前言与导语**

好的课程需要包含两方面：

1.整体的思路与编程思想（大局观，AOP，10-20%）

2.具体的编程知识与技巧（TP 5、小程序、数据库等80%）

**1-2 产品所使用的技术**

ThinkPHP 5

编写业务逻辑、访问数据库、向客户端提供数据

MySQL

数据存储、数据表设计、与业务紧密结合

微信

支付、善于借鉴与模仿，学习微信接口设计

小程序

直接与用户交互、体验很重要

**1-3 课程流程与体系**

1.服务端

ThinkPHP 5 + MySQL构建REST API

2.客户端

向服务端请求数据，完成自身行为逻辑

3.CMS

向服务端请求数据，实现发货与发送微信消息

本课程CMS的功能

1.基础数据的增删改查，比如添加商品、删除商品类目

2.特殊操作，比如我们要实现的发送微信消息

**1-4 扩展课程（三端分离）**

**1-5 项目特点**

我们想构建一个通用的、适合互联网公司的、有良好结构的产品

三端分离（客户端、服务器与数据管理分离）

基于REST API

基于Token令牌管理权限

一套架构适配iOS、Android、小程序与单页面

真正理解MVC

AOP面向切面编程思想在真实项目中的应用

使用ORM的方式与数据库交互

Object Relational Mapping对象关系映射

MySQL数据表设计与数据冗余的合理利用

用面向对象的思维构建前端代码（ES6 Class & Module）

**1-6 ThinkPHP 5 技术点简介**

Web框架三大核心知识（路由、控制器与模型）

验证器、读取器、缓存与全局异常处理

ORM：模型与关联模型

**1-7 微信技术点简介**

微信小程序

微信登陆

微信支付（预订单、库存量检测与回调通知处理）

微信模板消息（新版叫做订阅消息）

不同于订阅号、服务号，小程序不允许主动给用户发消息，只有用户在小程序中触发了某些选项的时候，才可以向用户下发模板消息

**1-8 MySQL技术点简介**

数据表设计

数据冗余的合理利用

事务与锁在订单（库存量）检测中的应用

**1-9 学习方式**

①代码本身并不难，静下心学习就好

②一关一关过，一级一级升，啃下这门课程

**1-10 前置知识需求**

① PHP与面向对象的相关知识

② ThinkPHP基本知识（对控制器和路由有一定的了解）

③ 了解关系数据库（MySQL）的基本使用，写过SQL语句

④ 小程序常用API

⑤ 要有一个小程序账号

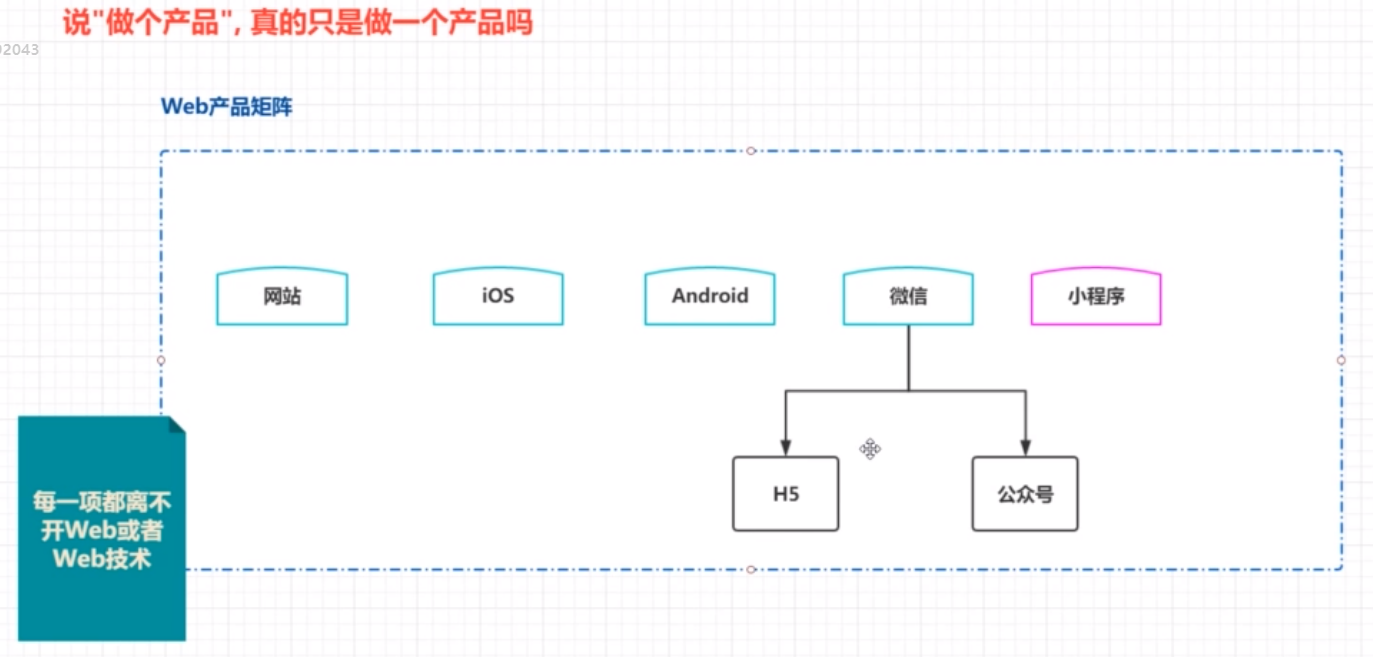
**1-11 扩展课程《理解Web与前端》**

语言与框架太多，关键是使用“它们”解决我们的问题。

先有想法，再寻找“工具”，而不是反过来。

这个思想贯穿我们的课程

① 泛化的Web。网站不是Web的全部，只是Web的一小部分



② 能写出代码和写出易维护的代码不是一回事儿

③ 前端绝不等于做界面和特效，前端与服务器在编程思维上的差异已经越来越小

④ 前端已经变得和服务器一样需要处理大量的业务逻辑

**1-12 扩展课程《第三方组件应该合理选择，特别是在学习阶段》**

依赖或者包管理

composer PHP包管理、包依赖关系管理工具

npm JavaScript/Node.js包管理工具

pip Python包管理工具

SDK（Software Development Kit）软件开发工具包

**1-13 维护与提问**

课程代码长期维护

课程提问：提问区提问

代码维护、更改通知（知乎专栏：https://www.zhihu.com/people/qi\_yue）

公众号：林间有风

**第2章 环境、工具与准备工作**

**2-1 福利！新增38万字CMS专栏教程**

talelin.com Lin-CMS 前后端分离的开源CMS框架

http://doc.mini.7yue.pro/ Lin-UI 基于【微信小程序原生语法】实现的组件库

微信小程序两大改动

①获取用户信息

②订阅消息

**2-2 环境与开发工具**

Web框架：ThinkPHP 5.07

基础语言、环境：PHP 5.6、MySQL、Apache

XAMPP：集成安装包

客户端：小程序

开发工具：

① PHPStorm

② 微信Web开发者工具

③ PostMan（建议写 单元测试）

④ Navicat

**2-3 安装xampp**

ThinkPHP只是Web框架，还需要PHP运行环境和Web服务器。

常见Web服务器：Apache、Nginx。

Windows操作系统查看端口号：netstat -ano

**2-4 下载ThinkPHP 5（版本5.0.7）**

三种途径：Composer安装、Git安装、zip文件下载。

**2-5 三端命名与运行ThinkPHP 5**

服务器程序：Zerg

客户端小程序：Protoss

CMS：Terran

**2-6 PHPStorm安装及常见使用技巧**

切换文件：ALT + ←、→ 方向键

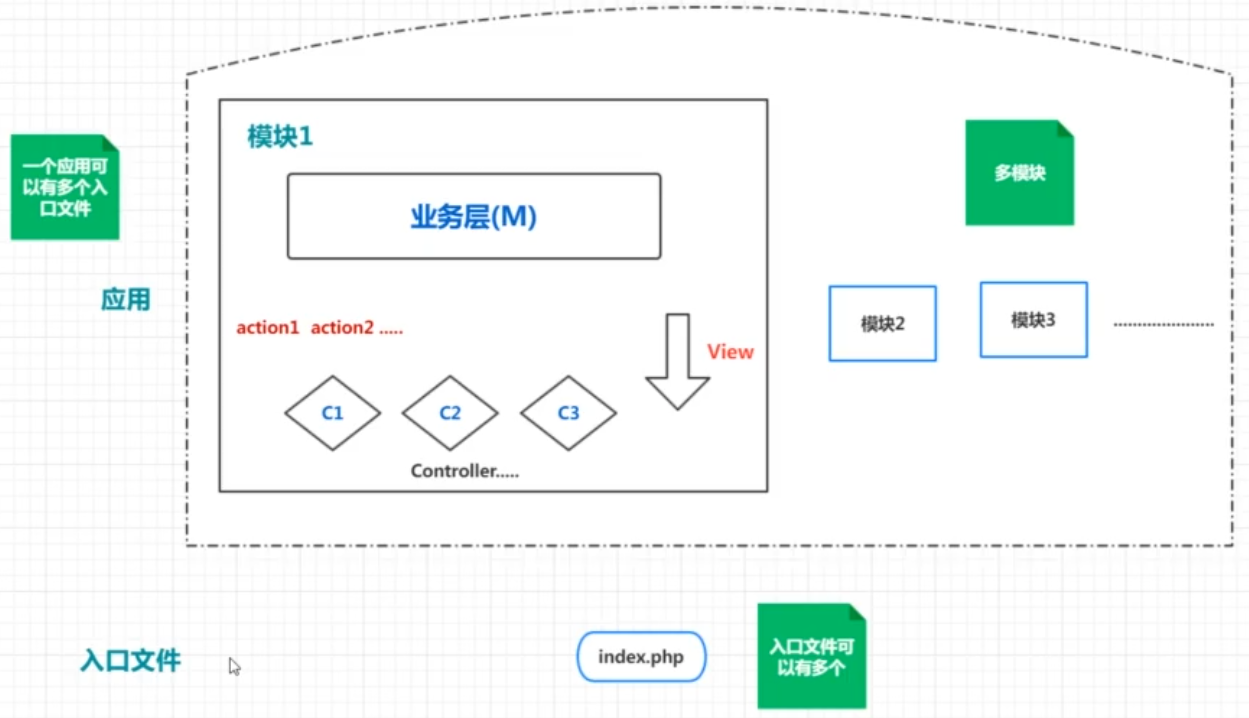
Recent Files：Ctrl + E

Navigate File：Ctrl + Shift +N

行注释：Ctrl + /

**2-7 ThinkPHP 5目录层次结构**

所有请求都要经过index.php，即使配置了【虚拟域名 or URL重写】，url中隐藏了index.php，thinkphp内部工作机制中，所有请求还是发送到index.php。



**2-8 扩展：TP5自带的Web Server**

php -S localhost:8080 router.php 可以监听8080端口

开发测试可以用这个Web Server，部署不能用。

**2-9 在PHPStorm下断点调试代码**

**2-10 PATH\_INFO URL路径模式解析**

http://serverName/index.php/module/controller/action/[param/value]

http://localhost/zerg/public/index.php/index/index/index

URL默认不区分大小写。application/config.php中url\_convert设为false，可以实现大小写敏感。

上述URL格式，官方称为：PATH\_INFO

兼容模式：http://serverName/index.php?s=module/controller/action/[param/value]

模块名/控制器名/操作方法

缺点：太长、URL路径暴露出了服务器文件结构、不够灵活、不能很好的支持URL语义化（最大的缺陷）

**第3章 模块、路由与获取请求参数**

**3-1 来编写一个简单的模块（多模块与模块命名空间）**

在TP5中，目录建议小写，类名大写。

配置Directories，将application设置为Sources，指定别名app。在项目中新建类时，不需要手动写命名空间，IDE会自动补齐。

**3-2 配置虚拟域名简化URL路径**

[**http://zhongyi.com/sample/test/hello**](http://zhongyi.com/sample/test/hello)**/[param/value]**

http://localhost/zhongyi/public/index.php/module/controller/action/[param/value]

**配置虚拟域名为了简化**localhost/zhongyi/public/index.php=> zhongyi.com

在Web服务器配置虚拟域名，监听端口，项目根下/public目录；

在c:\Windows\System32\drivers\etc\hosts设置域名指向Web服务器ip

**3-3 安装PostMan**

http://www.getpostman.com/

使用时可以收藏使用过的url，下次方便使用。

**3-4 三种URL访问模式**

①PATH\_INFO

②混合模式

③强制使用路由模式

application/config

‘url\_route\_on’=>true, //是否开启路由

‘url\_route\_must’=>false, //是否强制使用路由

application/route

use \think\Route;

Route::rule(‘xiaoyu’,’index/index/index’);

对同一个操作方法，设置路由之后，原PATH\_INFO访问方式失效。

**3-5 定义路由**

**3-6 获取请求参数**

use think\Request;

方式一：$all = Request::instance()->param();

方式二：$all = input(‘param.’);

TP 5中比较灵活的三个点：①路由 ②获取请求参数 ③数据库操作。需重点学习

**3-7 产品功能讲解与分析**

**3-8 Navicat安装以及数据库设计初步分析**

**第4章 构建验证层**

**4-1 Banner数据表设计分析**

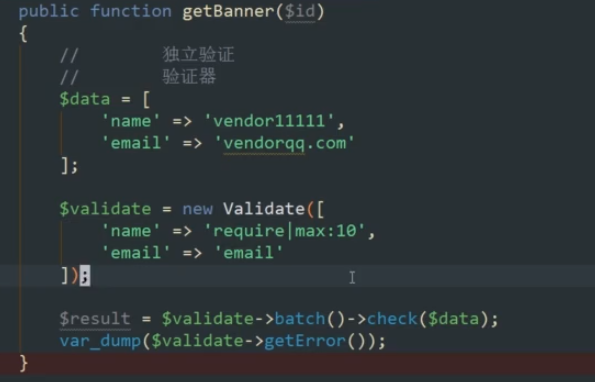
**4-2 Banner接口定义及自定义控制器多级目录**

路由规则：Route::rule(‘xiaoyu’,’index/index/index’);

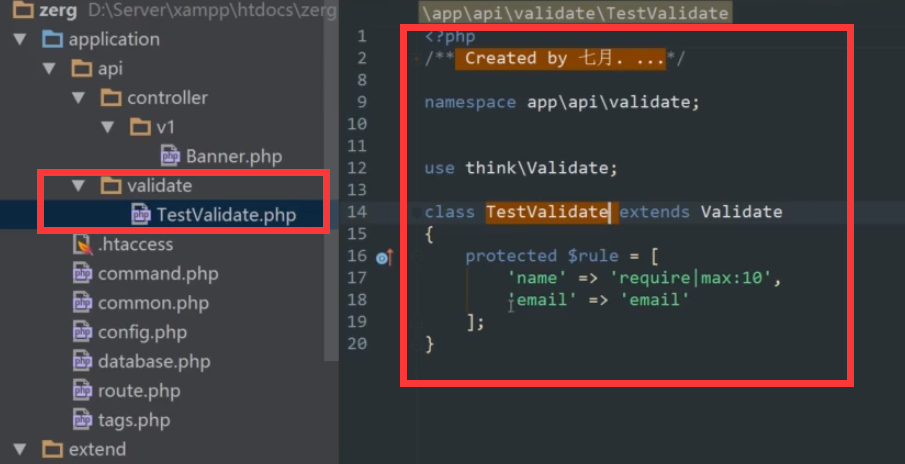
模块名/控制器名/操作方法

**4-3 Validate：独立验证**

客户端传过来的数据都是不可信的，接口参数校验非常重要。

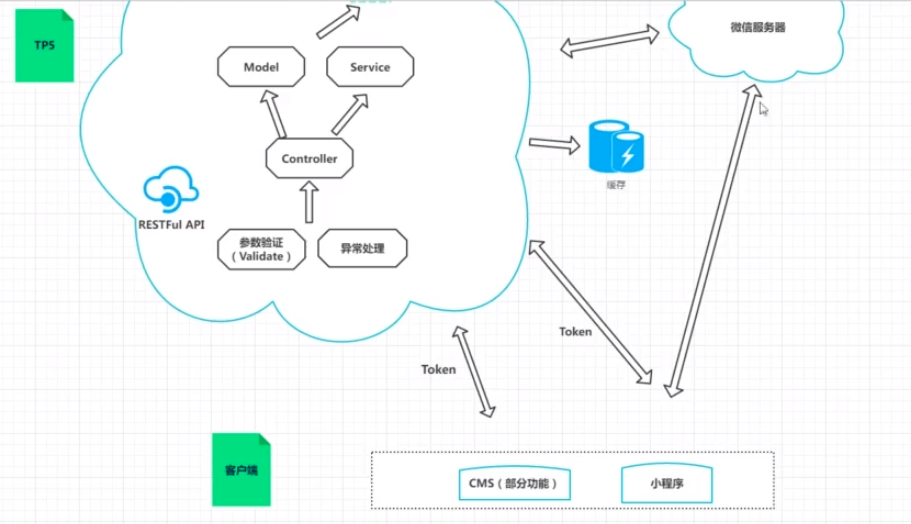
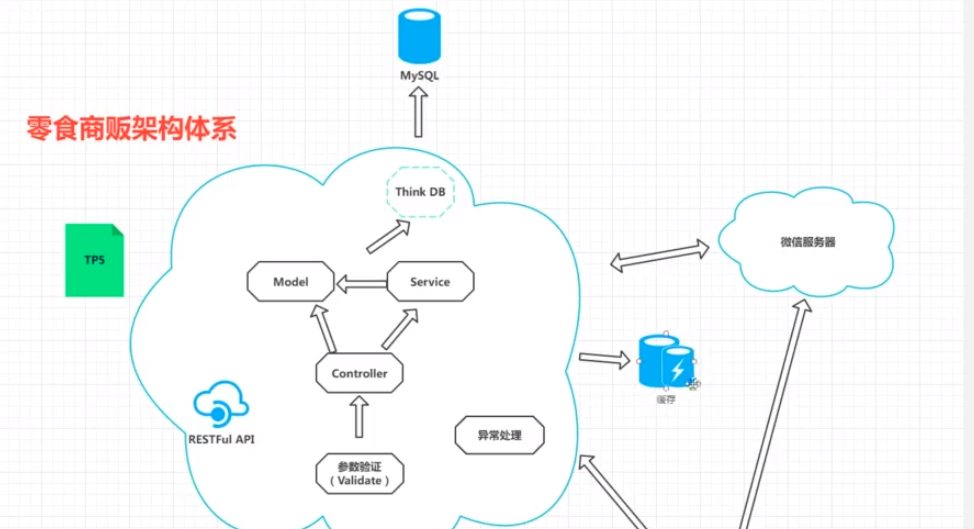


**4-4 Validate：验证器**





**4-5 零食商贩结构体系详解**



**4-6 自定义验证规则**

**4-7 工欲善其事必先利其器：构建接口参数校验层**

调试过程：

F6单步执行；

F8下一个断点；

F5进入函数；

**第5章 REST与RESTFul**

**5-1 介绍下REST之前的重要协议：SOAP**

REST（Representational State Transfer）：表述性转移。

七月理解：一种风格、约束、设计理念

SOAP VS REST

SOAP

重；

通常来说，使用XML描述数据；

**5-2 RESTFul API的特点解析**

RESTFul API（基于REST的API设计理论）

轻；

通常来说，使用JSON描述数据；

无状态

**基于资源，【增、删、改、查】都只是对于资源状态的改变；**

**使用HTTP动词来操作资源；**

**传统web开发中，选择get/post的依据：如果参数简单，就使用get；如果参数传递比较复杂（表单提交），使用post。**

**Restful API中，取决于操作是查询 or 新增 or 删除 or 更改。url表示资源，不允许出现动词，建议是名词。**

**/getmovie/:id => GET : /movie/:id**

**5-3 RESTFul API的最佳实践**

HTTP动词（幂等性、资源安全性）

POST：创建

PUT：更新

GET：查询

DELETE：删除

HTTP状态码：404、400、200、201、202、401、403、500

404：资源没有找到

400：参数错误

200：查询GET请求，执行成功

201：创建POST请求，执行成功

202：更新PUT请求，执行成功

401：未授权

403：当前资源被禁止

500：服务器的未知错误

错误码：自定义的错误ID号

**统一描述错误：错误码、错误信息、当前URL**

使用Token令牌来授权和验证身份

版本控制

测试与生产环境分开：api.xxx.com

dev.api.xxx.com

URL语义要明确，最好可以“望文知义”

最好是有一份比较标准的文档

**5-4 如何学习RESTFul API设计**

模仿

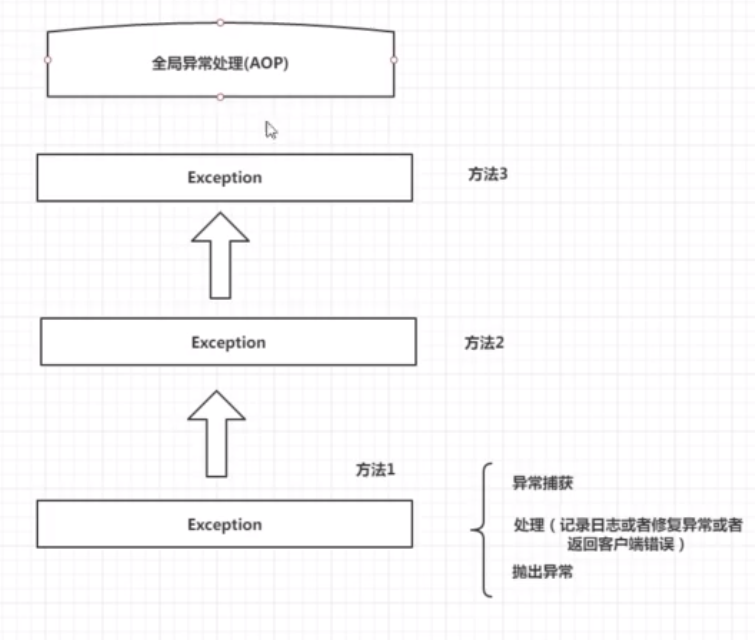
豆瓣开放API

GitHub开发者API

RESTFul API的合理使用（切勿盲目照搬标准REST）

**第6章AOP与全局异常处理**

**6-1 正确理解异常处理流程**



**6-2 固有的处理异常的思维模式与流程**

关闭TP 5调试模式

config.php中‘app\_debug’=false；

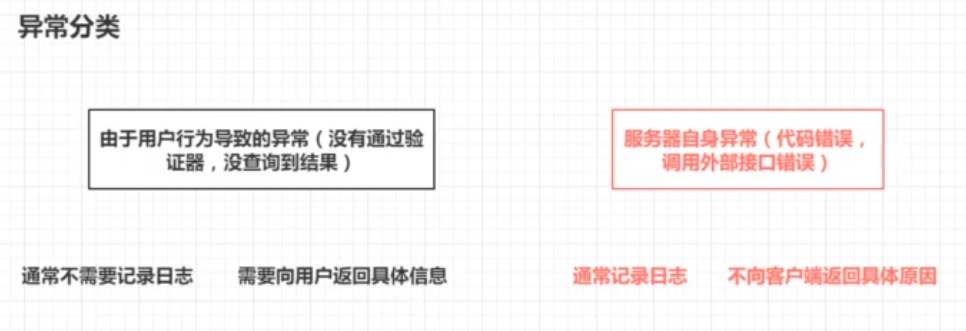
开发API与开发网页 不同。在处理异常时，尤其明显。

开发网页，服务器将错误页面返回，没问题；开发API，不能返回页面给前端。

Http中，200表示服务器成功响应。

RESTful中，200表示成功获取想要的结果。

**6-3 理清思路，总结异常的分类**



**6-4 实现自定义全局异常处理（上）**

继承 think\exception\Handle;

覆盖Handle中render方法后，要重新指定全局异常处理类。config.php中

// 异常处理handle类 留空使用 \think\exception\Handle  
'exception\_handle' => 'app\lib\exception\ExceptionHandler',

**6-5 实现自定义全局异常处理（下）**

如何判断异常是否需要记录日志？or客户端异常or服务器异常？

正推：我们就是新建了BaseException这个类和其他继承这个类的异常类去处理异常。

反推：所有需要返回给客户端的异常，我们都使用了自己新建的异常类；服务器异常没有使用。比如：使用MissException表示未找到，此时就是需要返回给客户端。

**6-6 ThinkPHP 5中的日志系统**

开发环境下，对日志的依赖不大；生产环境下，需要日志来查bug。

修改日志路径：在public/index.php中重新定义

define(‘LOG\_PATH’,\_\_DIR\_\_.’/../log/’);

此方法提前定义了 thinkphp/base.php中LOG\_PATH的值。

config.php中使用了LOG\_PATH。

**6-7 在全局异常处理中加入日志记录**

config.php中 ‘type’=>’test’，可以关闭TP 5默认记录日志的行为。

**6-8 全局异常处理的应用（上）**

config.php中‘app\_debug’，生产环境 建议关闭。

**6-9 全局异常处理的应用（中）**

**6-10 全局异常处理的应用（下）**

**6-11 本章小结与AOP思想**

写项目前考虑好处理异常、校验参数、生产环境关闭debug。

**第7张 数据库访问与ORM**

**7-1 数据库操作三种方式之原生SQL**

application/database.php配置数据库。

① 使用原生SQL语句查询数据库

② 使用构造器查询数据库

③ 使用模型、关联模型查询数据库

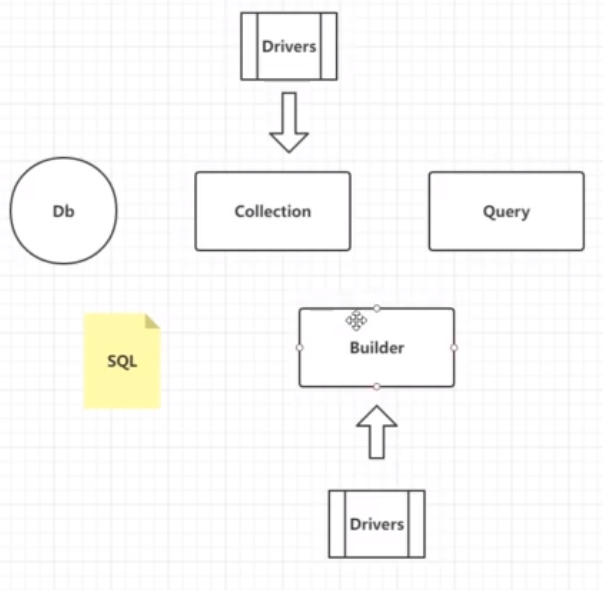
$result = Db::query(‘select \* from banner\_item where banner\_id=?’,[$id]);

**7-2 从一个错误了解Exception的继承关系**

自定义异常层，参数使用php的基类\Exception，不使用thinkphp\Exception。

<https://www.php.net/manual/zh/class.exception.php>中定义：Exception是所有异常的基类。无论是TP 5 框架的think\Exception or 自定义的BaseException。

**7-3 TP 5数据库中间层架构解析**



**7-4 查询构造器 一**

$result = Db::table(‘banner\_item’)->where(‘banner\_id’, ’=’, $id)

->find(); //返回一条记录，是一维数组

$result = Db::table(‘banner\_item’)->where(‘banner\_id’, ’=’, $id)

->select(); //返回多条记录，是二维数组

没有find() / select () 时，table()、where()返回的是Query对象。

调用find() / select () 后，才会真正执行SQL语句。

**7-5 查询构造器 二**

查询方法：find()、select()、update()、delete()、insert()，会真正执行SQL语句，且结束掉Db的连接状态。

辅助方法/链式方法table()、where()等，可以写任意多个，而且都不会真正执行SQL语句，返回Query对象。

**7-6 查询构造器 三**

where条件编写有三种方式：

①表达式 where(‘字段名’, ‘表达式’, ‘查询条件’)

②数组（不建议使用）

③闭包

**7-7 开启SQL日志记录**

①辅助方法/链式方法 fetchSql()，不会执行具体SQL，会返回SQL语句。

②数据库配置database.php中‘debug’=>true, //数据库调试模式

config.php中开启debug调试，开启log记录。或者在public/index.php中开启log日志记录。

**7-8 ORM与模型**

ORM（Object Relation Mapping）对象关系映射，不是具体技术，不是一门语言，是一种思想。不同的编程语言、不同的框架会实现自己的ORM（对象关系映射模型）。

模型：与数据库表不是一一对应关系，是和业务逻辑相关的。在简单项目中，会造成一种假象：模型与数据库表是一一对应关系，原因是业务简单，模型设计造成。

**7-9 初识模型**

Db属于数据库访问层；Model不是数据库访问层，Model是建立在数据库访问层上面的、处理业务逻辑的模型层。

新建模型要extends Model。如 class BannerModel extends Model

$banner = BannerModel::get($id);// 此时$banner是一个对象，不同于之前原生SQL或者Db类，返回的是一维数组or二维数组。

return的时候，TP 5默认输出的是html。

所以数组需要 return json($banner);但是对象的返回不需要json一下， return $banner。

配置文件中进行修改：config中’default\_return\_type’ =>’html’ //默认输出类型

$banner = BannerModel::get($id);

return $banner; //此时BannerModel继承Model

$banner = BannerModel::getBannerByID($id);

return json($banner); //此时BannerModel自己实现getBannerByID

**7-10 模型定义总结**

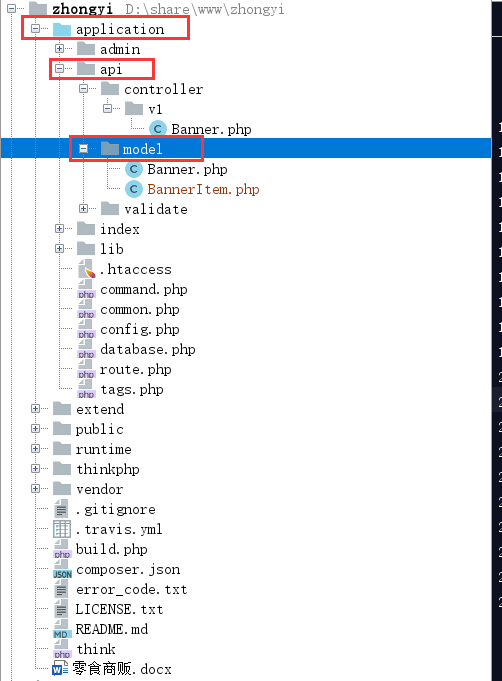
数据库表名和模型类名字不同时，可以在类中指定表名：protected $table = ‘表名’;

一般简单系统、简单业务时，数据库表名和模型类名一一对应。

命令行快速创建模型文件（在项目根目录）：php think make:model api/BannerItem

该命令自动在application下的api模块中的model层下新建BannerItem类。第一个参数为模块，第二个参数为模型类名。

目录结构为application/api/model



**7-11 静态调用还是实例对象调用**

模型进行查询时，推荐使用静态调用。

$banner = BannerModel::get($id); // 使用模型类的静态方法

$banner = new BannerModel(); // 实例化模型对象，调用对象的方法

$banner = $banner->get($id);

模型相当于数据库一张表，或者一张表的结构：比如BannerModel这个类。

模型实例化之后，就是模型对象，相当于表下面的一条记录：比如 $banner = new BannerModel();

**7-12 几种查询动词的总结与ORM性能问题的探讨**

**get、find返回一个模型对象/一条记录；**

**all、select返回一组模型对象/一组记录。**

**get、all是模型特有的方法（返回对象）；find、select是Db特有的方法（返回记录）；**

**使用Db类时，不能使用get、all；使用模型时，可以使用find、select。**

使用模型的几个原因：

①模型和数据库访问层是不同的，模型主要是用来处理业务的，数据库访问层是查询数据库的；模型是建立在Db的基础上的。

②不能因模型性能稍差，放弃使用模型。

③用面向对象的思维来使用、设计模型。

④模型的底层仍然是数据库访问抽象层。

**第8章 专题、分类、商品详情接口编写**

**8-1 Banner相关表分析（数据表关系分析）**

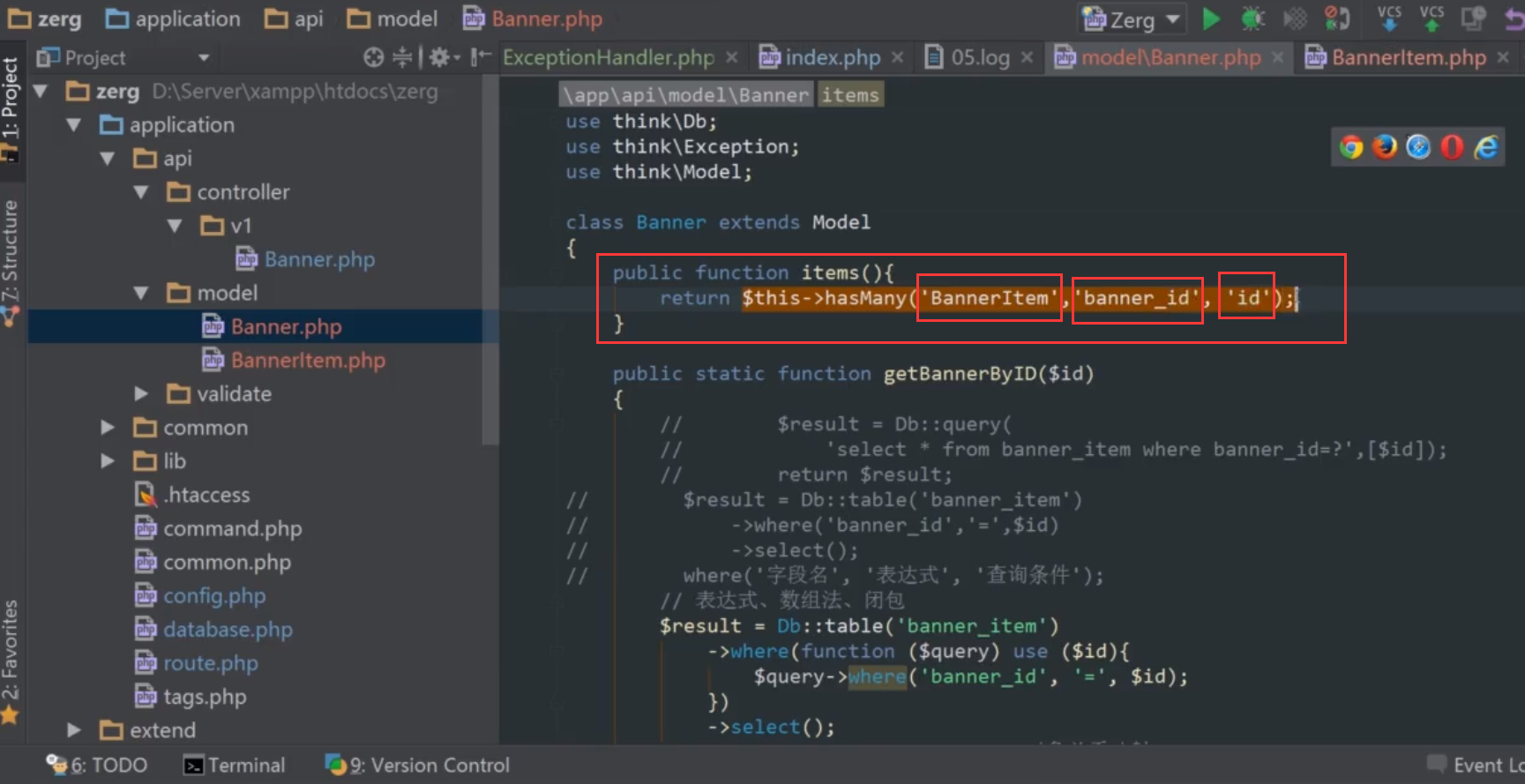
**8-2 模型关联-----用于关联与查询关联**

$this->hasMany(‘BannerItem’, ‘banner\_id', ‘id’);

第一个参数：关联模型的模型名

第二个参数：关联模型的外键

第三个参数：当前模型的主键



$banner = Banner::with(‘items’)->find($id);//此时使用find，不使用get、all。因为with返回的是Query

**8-3 模型关联-----嵌套关联查询**

**先定义关联关系，再使用。**

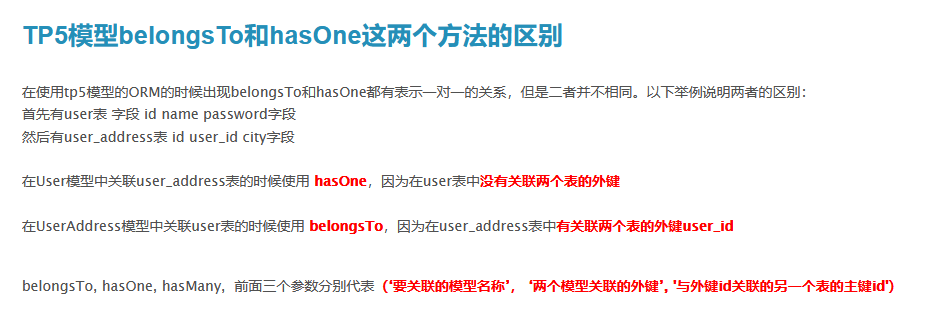
可以在模型中定义多个关联，BannerModel::with([‘items’,’items1’])，with中传入数组。

一对一关联关系：hasOne、belongsTo

一对多关联关系：hasMany

多对多关联关系：belongsToMany

一对一关系中，两方并不平等，存在主、从关系。关键看外键在哪张表。



**8-4 隐藏模型字段**

$banner = Banner::with(‘items’)->find($id);

$data = $banner->toArray();//将$banner模型对象转换为数组

unset($banner[‘delete\_time’]);

上述编写代码方式，强行将对象转换为数组，然后用数组的方法去处理这个将要返回给前端的数据。这样不合理，因为本身就是要使用ORM(对象关系映射)：我们就是要将数据库的一张表，当作模型；将数据库的表下一条记录，当作模型对象来去处理数据。模型提供了很多内置方法。

$banner->hidden([‘delete\_time’, ’update\_time’]);//隐藏字段

$banner->visible([‘id’, ‘update\_time’]); // 只返回某些字段

**8-5 在模型内部隐藏字段**

直接在模型内部，定义protected成员，就可以达到在控制器中$banner->visible([‘id’, ‘update\_time’]);这种效果。此时就不需要把这种类似的逻辑，放在控制器中编写，而是在模型层编写。

protected $hidden = [‘update’];

protected $visible = [‘id’];

隐藏字段的原因：

①安全性考虑，有些字段不应该返回给客户端；

②为了让返回的json数据，足够简洁、清晰、明了；

**8-6 图片资源URL配置**

存在本地的图片，无法在DB把URL写完整，因为在服务器配置项目目录时可能不同，所以要写相对目录；使用时，根据网站域名配置、网站根目录配置，拼接出完整URL。

存在互联网的图片，可以在DB保存完整URL。

**扩展配置目录，在这个目录下的配置文件，都可以被TP 5 默认加载。applicaton/extra目录，凡是放在这个目录下的配置文件，都可以被TP 5默认加载。**

在项目中使用助手函数config(‘文件名.数组键名’)就可以拿到自定义配置的某个值。比如：在extra下面有一个setting.php，定义

return [’ img\_prefix’ =>’http://z.cn/images’];

config(‘setting.img\_prefix’);就可以获取到<http://z.cn/images>

**8-7 读取器的巧妙应用**

读取器是属于模型的，使用模型的读取器，可以避免在Controller控制器中，编写应当属于model模型层的代码。侧面印证使用模型层的好处。

public function getUrlAttr()

{

}

get是固定的；Url是数据库中字段名（驼峰命名法）；Attr是固定的；

使用时，模型层这个Url字段有多少个返回，getUrlAttr就会被调用多少次。

**图片的地址处理，包含在Image模型中，Controller控制器的代码就可以写的很简洁。使用某个类时，复杂的方法应该封装在这个类内部，体现了面向对象的封装性。而不应该在控制器（类的外部）去处理应当属于这个类的处理。**

$img = new Image();

$img->url //此时就会调用Image模型的getUrlAttr()方法

框架的return，也会调用$img->url。TP 5的官方文档，叫做获取器。

**8-8 自定义模型基类**

共有特性的抽象问题，新建基类。

可以在BaseModel新建getUrlAttr()方法，但是所有子类都会继承这个方法，并且去处理自己模型，如果子类模型涉及到Url字段，但是该字段不是Image模型的Url字段意义，此时就不合适。

所以在BaseModel基类新建prefixUrlImg()方法，在子类中使用getUrlAttr()方法去调用。因为框架会把getUrlAtrr()方法识别为一个读取器。但是我们并不需要所有子类都获取BaseModel基类自己定义的读取器，而是在子类中定义并使用。如果一个读取器是很多子类都共同需要的，那可以定义在基类中。

有几下几种方案去处理相同问题：

①把getUrlAttr()方法提取成一个函数，有需要的地方调用；

②把getUrlAttr()方法的函数内容，在需要的地方复制一遍代码；

③在基类中写一个公共方法，在子类中写自己的getUrlAttr()方法去调用prefixUrlImg()公共方法。

方法①、③差不多，方法③更具有面向对象的思想。如果数据库中涉及到url字段，都是表示相同意义，那就不需要prefixUrlImg()方法，而是直接把getUrlAttr()写在基类中。

**8-9 定义API版本号**

**开闭原则：代码对扩展是开放的，对修改是封闭的。**

**凡是修改代码，都会涉及到影响以前代码正常调用的风险。**

**下述方法动态支持前端调用。**

[**http://zhongyi.com/api/v1/banner/1**](http://zhongyi.com/api/v1/banner/1)

**路由到**[**http://zhongyi.com/api/v1/Banner/getBanner 传参 1**](http://zhongyi.com/api/v1/Banner/getBanner%20传参%201)

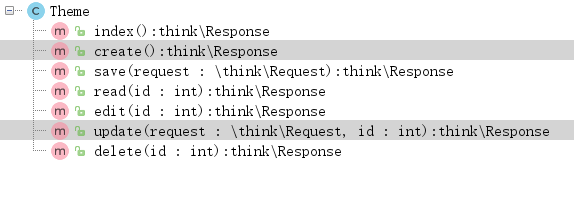
**api是模块；v1/Banner 是控制器；getBanner是操作方法。**

//Route::rule('banner/:id','api/v1.Banner/getBanner');  
Route::rule('api/:version/banner/:id','api/:version.Banner/getBanner');

**8-10 专题接口模型分析**

php think make:controller api/v1/Theme

TP 5框架新建的controller，更适合Web网页开发，不适合api开发。



**save()、delete()、update()、read() 、index() 对应数据库的【增、删、改、查】操作。**

**create()、 edit() 显示对应的表单页**

**删除操作，不需要表单页。**

**8-11 一对一关系解析**

**8-12 Theme接口验证与重构**

**8-13 完成Theme简要信息接口**

**8-14 开启路由完整匹配模式**

TP 5的route.php，默认会从上往下匹配。所以前端调用的时候，会匹配27行的路由，并不会匹配28行的路由，那就会出问题。需要修改配置config.php中‘route\_complete\_match’=>false（路由使用完整匹配）

**RESTful服务中，每个url应当仅对应一个接口。不应该出现模糊的情况。**



**8-15 编写Theme详情接口**

**RESTFul是基于资源的，请求Banner，就会把Banner所有字段/属性返回。**开放接口设计中，不清楚前端要用数据做什么，所以会返回所有属性。针对自己项目做开发，不要过度使用RESTFul，针对性的去除一些前端不需要的字段。

**8-16 数据库字段冗余的合理利用**

pivot，模型关联多对多关系的中间表属性。

不可控的数据集，尽量不要去和其他表做关联查询，会导致很多次循环，故我们在product表设计数据库字段冗余main\_img\_url。

但是不能滥用数据冗余！新增操作，需要多个地方新增；删除、更新操作如果不做相应处理，会导致数据不一致。

**8-17 REST的合理利用**

①如果前端要什么，服务器给什么，当前端业务需求变化时，服务器代码也要做相应调整，不具有兼容性。RESTFul基于资源，返回内容比较多，适应前端业务能力强。

②不能一味的过度使用RESTFul，还是要甄别一下，有些字段前端确实不需要，那就不必要返回给前端。

既不能只按照前端要求来返回数据，也不能一味全部返回。（根据自己项目合理修改）

**8-18 最新新品接口编写**

**8-19 使用数据集还是数组？**

可以使用助手函数，把数组变为数据集。

$collection = collection($products)

$products = $collection->hidden([‘summary’]); // 临时隐藏summary字段，就不需要写在Model中。写在Model中的弊端是，所有查询都会隐藏。

如果使用数据集，就不能使用 !$product 来判断是否为空，应当使用对象的方法，$product->isEmpty()。

database.php中，设置‘resultset\_type’=>’collection’，可以修改默认数据集返回类型。

不过我个人还是喜欢返回array，然后在控制器中，决定是否转为collection。

**需要再研究一下get()、all()、select()、find()**

**8-20 分类列表接口编写**

$categories = CategoryModel::all( [ ], ‘img’);

等价于

$categories = CategoryModel::with(‘img’)->select();

**8-21 扩展：接口粒度与接口分层**

粒度过大、过粗，复用性不好、不够灵活；

粒度过于细小，客户端调用起来不方便。

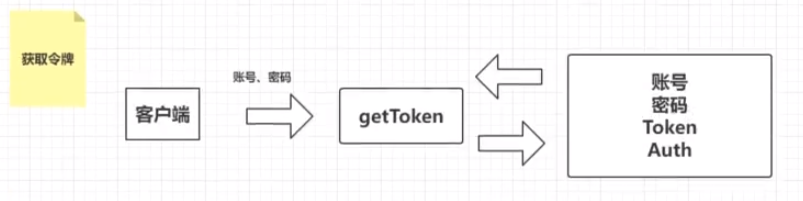
如果项目比较复杂，考虑对后台代码进行分层，建立基数数据层和业务层。业务层去调用基础数据层的api，组装数据后返回给客户端，此时客户端不需要发送多次http请求。

**8-22 分类商品接口编写**

**第9章 微信登录与令牌**

**9-1 初识Token-----意义与作用**

在api里面，不存在用户登陆这样的概念，系统用令牌管理身份。





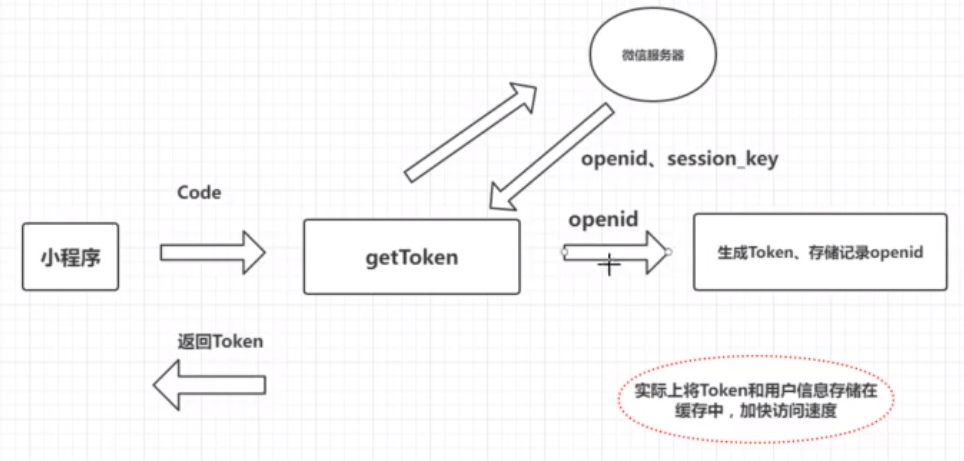
**访问接口时，进行三层验证：Token是否合法、是否有效、对应权限。**

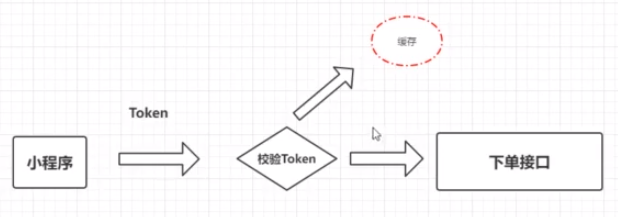
**①有些Token是伪造的，肯定不合法；**

**②Token需要有效期，不可能设置一个Token，过了三、五年，还能访问接口；**

**③Token对应着身份和权限，有些Token虽然合法、有效，但是权限不足，也不能获取对应接口。**

**9-2 微信身份体系设计**





**9-3 实现Token身份权限体系**

model层进行分成，分为model和service。

service层用于处理较为复杂的业务逻辑；model层处理细粒度、简单的业务逻辑，还负责调用数据库访问层，实现对数据库表的增、删、改、查。

**9-4 实现Token身份权限体系二-----获取openid**

**9-5 实现Token身份权限体系三**

**json中，为双引号**

**9-6 实现Token身份权限体系四**

**公共函数，写在application/common.php中。全局可以使用。**

**9-7 实现Token身份权限体系五**

**9-8 API小程序测试工具**

**9-9 商品详情接口分析与初步编写**

**9-10 路由变量规则与分组**

TP 5的route.php，默认会从上往下匹配。所以前端调用的时候，会匹配27行的路由，并不会匹配28行的路由，那就会出问题。需要修改配置config.php中‘route\_complete\_match’=>false（路由使用完整匹配）

**RESTful服务中，每个url应当仅对应一个接口。不应该出现模糊的情况。**

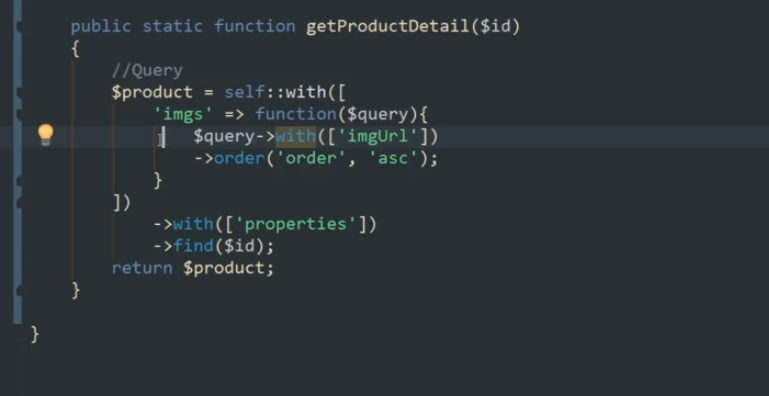
即使使用了路由完整匹配，在如下情况下，33行依旧会匹配失败。原因是TP 5会把它当作32行的路由进行匹配，发现recent不符合id字段的校验规则，报错。





通过设定路由变量规则，使得api/v1/product/:id中id为正整数时才去匹配32行的路由。

**9-11 闭包函数构建查询器**



**9-12 用户收货地址-----通过令牌获取用户标识**

**9-13 用户收货地址-----面向对象的方式封装获取UID方法**

**9-14 用户收货地址-----模型新增和更新**

**9-15 用户收货地址-----参数过滤**

**9-16 用户收货地址-----接口测试**

**第10章 微信支付**

**10-1 Scope权限作用域的应用**

**10-2 前置方法**

**10-3 对Address接口做权限控制**

**10-4 下单与支付的业务流程（库存量检测）**

**10-5 下单与支付详细流程**

**10-6 重构权限控制前置方法**

**10-7 编写一个复杂的验证器**

**10-8 下单接口业务模型一**

**10-9 下单接口业务模型二**

**10-10 订单快照**

**10-11 订单快照的实现**

**10-12 订单创建**

**10-13 一对多关系的新增操作**

**10-14 测试订单接口**

**10-15 TP 5自动写入时间戳**

**10-16 在TP 5 中使用事务**

**10-17 关于微信支付**

**10-18 支付的服务器端编写一**

**10-19 支付的服务器端编写二**

**10-20 支付的服务器端编写三**

**10-21 支付的服务器端编写四**

**10-22 支付的服务器端编写五**

**10-23 支付的服务器端编写六**

**10-24 支付的服务器端编写七**

**10-25 支付的小程序端讲解（含签名的作用讲解）八**

**10-26 支付的服务器端编写九**

**10-27 统一测试订单与支付接口**

**10-28 支付的服务器端编写（支付结果回调机制）十一**

**10-29 支付回调通知处理一**

**10-30 支付回调通知处理二**

**10-31 事务与锁防止多次减库存**

**10-32 接收微信回调**

**10-33 测试支付全流程**

**10-34 可以断点调试微信支付的回调吗？**

**10-35 TP5中的分页查询与获取用户历史订单数据**

**10-36 测试与验证历史订单数据接口**

**10-37 订单详情接口**