**交付应用开发技术规范**

v1.7

修改记录：

|  |  |  |  |
| --- | --- | --- | --- |
| 版本号 | 修改时间 | 修改内容说明 | 修改人 |
| v1.1 | 201906 | 初版 | 周龙 |
| v1.2 | 201907 | 数据库规范补充 | 葛兵 |
| v1.4 | 201908 | PHP版本和框架限定，类名规范修订 | 葛兵 |
| v1.5 | 201909 | PHP集成环境和框架用法补充 | 葛兵 |
| V1.6 | 201910 | PHP代码注释规范补充(12)(13)(14) | 吴治俭 |
| V1.7 | 201911 | 前端框架补充vue环境搭建说明  PHP http状态码和抛异常规范补充(17)(18)(19) | 王欢  赵金辉 |

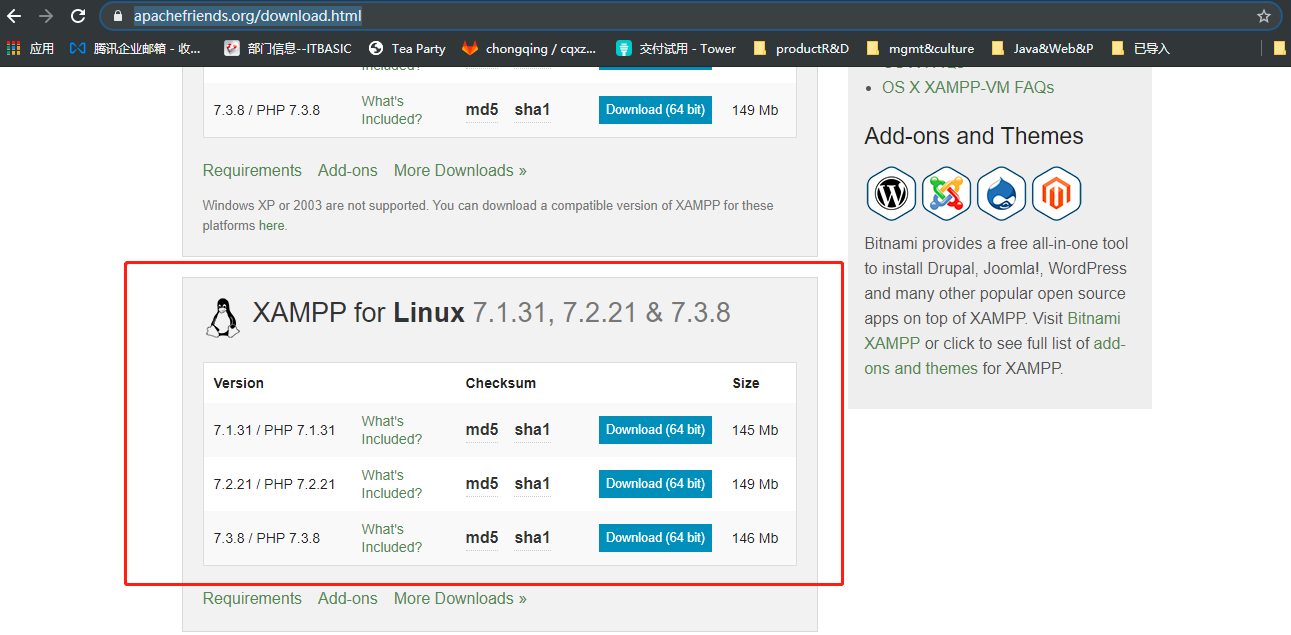
# PHP规范

## 版本：

PHP语言版本建议使用7.1及以上，见下集成环境。

## 集成环境版本和部署说明：

PHP集成环境建议使用XAMPP 7.1及以上，<https://www.apachefriends.org/download.html>。



xampp服务配置

1.如果启动xampp时80端口被占用了。则需要在XAMPP目录下的\lampp\etc\httpd.conf文件中搜索到Listen 80一句，改成服务器没被占用的端口即可。例如：Listen 81

2.将\lampp\etc\httpd.conf 里面 第488行 #Include etc/extra/httpd-vhosts.conf去掉前面#，目的是为了引用 httpd-vhosts.conf 文件。

3.配置项目站点

在\lampp\etc\extra\目录下的httpd-vhosts.conf文件

打开添加如下代码

<VirtualHost \*:81>

DocumentRoot "D:\xampp\htdocs\thinkphp5\_demo\public"

SetEnv APPLICATION\_ENV "development"

ServerName crm2.abc.com

<Directory "D:\xampp\htdocs\thinkphp5\_demo\public">

DirectoryIndex index.php index.html

AllowOverride All

Allow from All

Order allow,deny

</Directory>

ErrorLog "logs/localhost-error.log"

CustomLog "logs/localhost-access.log" common

</VirtualHost>

说明:( DocumentRoot "项目的入口文件index.php所在的目录，例如：D:\XAMPP\htdocs\thinkphp5\_demo\public")，注意:<VirtualHost \*:81>要保持和httpd.conf文件中监听端口一致。

## 框架：

1. 对于已有项目，不建议跟换框架；
2. 对于新开项目，建议采用交付统一的ThinkPHP5.0框架，见下附件：

使用注意：PHP语言版本请使用php7.\*版本。

请将\thinkphp\library\think\db\connector\pgsql.sql放到数据库执行，执行之后可以使用TP函数的Insert和InsertAll。

Linux环境下需要手动创建当前的runtime空目录文件并且赋予777权限(参见下述目录结构说明)。

thinkphp5\_demo 目录下子目录

├─application 应用目录

├--admin 项目后台管理模块

├--index 项目前台管理模块（如果是前后端分离项目,建议只保留index目录,admin目录可以删除，或者保留不用即可）

├─extend 扩展类库目录

├─public 网站对外访问目录

├─runtime 运行时日志目录（项目启动时在Windows环境下自动生成，**如果是Linux环境，需要手动创建当前的runtime空目录文件并且给777权限即可**）

├─vendor 第三方类库目录（Composer）

├─thinkphp 框架核心目录

├─build.php 自动生成定义文件

├─tp5.sql 框架测试数据库sql语句

可参阅<[**ThinkPHP5.0完全开发手册**](https://www.kancloud.cn/manual/thinkphp5)> <https://www.kancloud.cn/manual/thinkphp5/118127>

## 编码：

1、PHP文件统一采用UTF-8编码；

## 缩进：

1、代码必须使用**「Tab 键」**进行缩放，Tab是**4**个空格，且在编码过程中，不同的代码层级采用不同的缩进，保持代码的工整；  
 2、示例：

class Foo extends implements FooInterface

{

public function sampleFunction($a, $b = null)

{

if ($a === $b) {

bar();

} elseif ($a > $b) {

$foo->bar($arg1);

} else {

BazClass::bar($arg2, $arg3);

}

}

final public static function bar()

{

//方法的内容

}

}

## 行：

1、行的长度禁止有硬性的约束，软性的长度约束应该要限制在120个字符以内；   
 2、行后禁止有多余的空格符；  
 3、每行禁止存在多于一条语句；

## 命名空间：

1、namespace声明后**最好**插入一个空白行；  
2、所有use**必须**在namespace后声明；  
3、示例：

namespace Verdor\Package;

use FooCass;

use BarClass as Bar;

use OtherVendor\OtherPackage\BazClass;

## 类名与文件名：

1、类名采用驼峰命名，首字母大写，以.php结尾；默认不需要添加后缀，例如UserController应该直接命名为User  
2、文件名与类名保持一致；  
3、示例：  
文件名：Base.php  
类:class Base extends Controller

## 类属性名与方法名：

1、类属性与方法名采用驼峰命名，首字母小写；  
 2、类属性与方法**必须**添加访问修饰符（private、protected以及public）；  
 3、私有类属性与方法，以**\_**开头；  
 4、示例：

//类方法

getNickname()；

//私有类方法

\_getNickname()；

## 公共函数名：

1、公共函数采用小写字母和下划线的方式；  
 2、例如：

//公共方法

get\_now\_time()；

## 常量与配置参数名：

1. 常量以大写字母和下划线命名，例如APP\_PATH；
2. 配置参数以小写字母和下划线命名，例如url\_route\_on

## 关于按照apidoc规范及其工具写注释和用注释的说明：

后端开发中，核心重要的技术环节，是把业务逻辑封装成类、及其函数/方法，体现在对外服务的api接口上，给前端或后端其他模块使用，因此，描述清晰、内容详细的api文档相当重要，这不论对开发人员自己条理化“围绕业务逻辑的对外服务接口”展开实现，还是对于开发工作的前后端和模块间的高效协同，都是十分必要和有价值的。因此，如下以规范化标签化的类注释和函数/方法注释为基础，利用apidoc工具生成api文档，就作为各项目后端应用开发的、重要且必须全面贯彻推行的规范。

首先，是如下两个参考文档，需要各项目在实施中参阅。

然后，是如下两个小节“类注释”和“函数/方法注释”，明确了注释的必填项及其标签，需要遵照实施。

## 类注释：

1. 每个类**必须**进行注释；  
    2、注释内容包括：~~操作系统(必填)、php运行环境(必填)、~~类名称(必填，@controller)、简要说明（必填，@Description）、作者(必填，@author)、版本号（必填，@version）、创建/修改时间(必填，@createtime)；  
    3、示例：

/\*\*~~Linux版本centos7.3 windows环境使用~~

[~~\*@php 7.0.2~~](mailto:*@php版本7.0.2) ~~+ Apache2.2.22 + pgsql 10.0 版本~~

\*@controller 个人中心控制器

\*@Description包含登陆者信息展示、修改资料、消息推送等功能

\*@author zhou.long

\*@version 1.0

\*@createtime 2019-07-04

\*/

## 函数/方法注释：

1. 每个函数/方法**必须**进行注释；  
   2、注释内容包括：简要说明（@apiDescription）、作者(@author)、版本号（@apiVersion）、创建/修改时间(@apiCreatetime)、导航称号（@apiGroup）、api名称（@apiName）、请求参数（@apiParam）、返回数据描述（@apiSuccess）、接口成功返回样例（@apiSuccessExample）、接口失败描述（@apiError）、接口失败返回样例（@apiErrorExample）、测试地址（@apiSampleRequest）;**代码修改的同时，注释也要进行相应的修改，尤其是参数、返回值、异常、核心逻辑等的修改**

【参考】注释要求：能够准确反应设计思想和代码逻辑；能够描述业务含义。注释是给自己看的，即使隔很长时间，也能清晰理解当时的思路；注释也是给其他人看的，例如代码走查者、开发交接者，使其能够理解代码逻辑或接续维护。

【参考】好的命名、代码结构是自解释的，注释力求精简准确、表达到位。避免出现注释的一个极端：过多过滥的注释，代码的逻辑一旦修改，修改注释是相当大的负担。

反例：

/\* put elephant into fridge \*/

put($elephant, $fridge)

方法名 put，加上两个有意义的变量名 elephant 和 fridge，已经说明了这是在干什么，语义清晰的代码不需要额外的注释。

【参考】特殊注释标记，请注明标记人与标记时间。注意及时处理这些标记，通过标记扫描， 经常清理此类标记。线上故障有时候就是来源于这些标记处的代码。 1） 待办事宜（TODO）:（ 标记人，标记时间，[预计处理时间]） 表示需要实现，但目前还未实现的功能。 2） 错误，不能工作（FIXME）:（标记人，标记时间，[预计处理时间]） 在注释中用 FIXME 标记某代码是错误的，而且不能工作，需要及时纠正的情况。

3、示例:

1、每个方法**必须**进行注释；

2、注释内容说明：

**一**、**HTTP接口调用方法、路径及名称**

**(1) 格式 @api {method} path [title]**

例: **method**请求方式:get/post/put…（必填）

**path** 请求路径:User/GetUser （必填）

**title** 接口标题:获取用户信息 （选填）

其中**{}或省略不写必填项**,**[]**为**选填项**

格式为: @api {POST}(请求方式为POST) http://aaa.com/User/GetUser (请求路径 PATH)

title(标题可以省略不写)

1. **@apiExample**

@apiExample[{type}] title example //使用API方法的示例。 作为预格式化代码输出。

示例:1 [get请求,请求参数单条]

/\*\*

\* @api {get} /user/getUserById/:id

\* @apiExample {curl} Example usage:

\* curl -i http://127.0.0.1/user/getUserById/1

\*/

示例:2[get请求,请求参数多条]

/\*\*

\* @api {get} /user/getUserById/:id/:goodsid/:gid

\* @apiExample {request} Example usage:

\* @apiSampleRequest http://127.0.0.1/user/getUserById/1/goodsid/5

\*/

或者

/\*\*

\* @api {get} /user/getUserById

\* @apiParamExample {string} 请求参数格式:

\* ?getUserById=1&goodsid=20

\* @apiExample {request} Example usage:

\* @apiSampleRequest http://127.0.0.1/user/getUserById?=1&goodsid=20

\*/

示例:3 [post,请求参数单条]

/\*\*

\* @api {post} /user/getUserList/

\* @apiExample {request} Example usage:

\* @apiParam [(group)] [{type}] [field=defaultValue] [description]

\* @apiSuccessExample {json} 正确返回值:

\* {

\* "code": ,

\* "msg": "",

\* "data": [

\* { }

\* ]

\* "status": ""

\* }

\* @apiSampleRequest http://127.0.0.1/user/getUserList

\*/

示例:4 [post,请求参数多条]

/\*\*

\* @api {post} /user/getUserList2/

\* @apiExample {request} Example usage:

\* @apiParam [(group)] [{type}] [field=defaultValue] [description]

\* @apiParamExample {json} 请求参数格式:

\* {

Field1=”xx”,

Field2=”xx”,

...

}

\* @apiSuccessExample {json} 正确返回值:

\* {

\* "code": ,

\* "msg": "",

\* "data": [

\* { }

\* ]

\* "status": ""

\* }

\* @apiSampleRequest http://127.0.0.1/user/getUserList2

\*/

示例:5 [请求成功后的返回字段参数]

/\*\*

\* @apiSuccess [(group)] [{type}] field [description]

\*/

(group) 可选 参数归属组名，不填写组名，则默认设为Success 200

{type} 可选 返回的数据类型，如{String}、{Number}等等

field 返回的标示符（返回成功的状态码）

description 可选 描述

/\*\*请求成功后返回的字段参数例子

\* @apiSuccessExample [{type}] [title] example

\*/

{type} 可选 请求数据结构

title 可选 一个简短的标题

example 例子的详细信息，可多个例子并存

@apiSuccessExample {json} Success-Response:

HTTP/1.1 200 OK

{

"Field1": ""

}

示例:6 [请求失败后的返回字段参数,用法跟@apiSuccess、@apiSuccessExample的用法相类似]

\* @apiError (400){Int} status\_code 状态码

\* @apiError (400){String} msg 返回消息

\* @apiErrorExample {json} 失败:

\* {

\* "msg": "",

\* "status\_code": 201, //状态码

\* }

**二**、**注释常用到的注解说明**

@apiPermission API的访问权限,没有权限要求的话，此项可以省略。

@apiVersion 版本

@apiDescription 详细描述

@apiCreatetime 创建日期

@apiGroup 定义的方法的文档块属于哪个组，该组的名称。也用作导航称号。

@apiName api名称

@apiParam 请求参数

@apiSuccess 返回数据描述

@apiSuccessExample 接口成功返回样例

@apiError 接口失败描述

@apiErrorExample 接口失败返回样例

@apiSampleRequest 文档提供的API Sample测试的地址

实例1 get请求方式,接收参数为单条

/\*\*

\*@author zhou.long

\*@apiVersion 1.0

\*@apiDescription根据用户id查询用户有效信息，请求方式为get请求

\*@apiCreatetime 2019-07-04

\*@api {get} http://aaa.com/User/userinfo/:id 根据id获取用户信息

\*@apiName userinfo

\*@apiGroup User

\*@apiParam {Number} id 用户唯一id

\*@apiSuccess {Array} 返回相应id的用户信息

\*@apiSampleRequest http://aaa.com/User/userinfo/1

\*@apiSuccessExample Success-Response:

\* HTTP/1.1 200 OK

\* {

\* "firstname": "John",

\* "lastname": "Doe"

\* "telephone": "18500901123"

\* }

\*@apiError (Error 4xx) 404 对应id的用户信息不存在

\*@apiErrorExample Error-Response:

\*HTTP/1.1 404 对应id的用户信息不存在

\*{

\* "error":err

\*}

\*/

public function userinfo(){//函数体}

实例2 get请求方式,（入参）是多条

/\*\*

\* @api {get} /getcode接口测试

\* @apiDescription 根据ID（id）获取列表信息

\* @apiGroup code APIs

\* @apiParam {Number} id 任务ID

\* @apiParam {Number} [page] 页数

\* @apiParam {Number} [perpage] 每页的条数

\* @apiParamExample {string} 请求参数格式:

\* ?id=123&page=1&perpage=20

\* @apiVersion 1.0.0

\*@apiSampleRequest /projectName/code/getcode?id=123&page=1&perpage=20

\* @apiErrorExample {json} 错误返回值:

\* {

\* "code": 300,

\* "msg": "ParametersError [Method]:get\_tests参数错误!",

\* "error": {

\* "id": "",

\* "page": "",

\* "perpage": ""

\* },

\* "status": "fail"

\* }

\* @apiSuccessExample {json} 正确返回值:

\* {

\* "code": 0,

\* "msg": "OK ",

\* "data": [

\* {

\* "id": "622051004185471233",

\* "Code": "000050",

\* }

\* ],

\* "status": "ok",

\* "count": "14"

\* }

\*/

public function getcode(){//函数体}

实例3 post请求方式,接收参数为单条

/\*\*

\*@author zhou.long

\*@apiDescription 获取用户信息 请求方式为post传值, 查询用户基本详情

\*@createtime 2019-07-04

\*@api {POST} http://aaa.com/User/GetUser API请求路径 **格式:@api {method} path [title]**

\*@apiVersion 0.1.0 api版本

\*@apiGroup User 模块分组(当前的方法属于那个组)

\*@apiName GetUser 方法名

\*@apiParam {int} id 用户唯一id-非空

\* @apiParam {String} [need\_name] 需求者名称-为空  
 \* @apiParam {String} [e\_mail] 用户邮箱格式-为空  
 \* @apiParam {String} [phone] 用户电话-为空  
 \* @apiParam {String} [company\_name] 需求公司名称-为空  
 \* @apiParam {String} [desc] 个人描述-为空  
 \* @apiparam {String} [password] 用户密码-为空  
 \* @apiSuccess {String} respMsg 响应信息, 如:成功  
 \* @apiSuccess {String[]} data 返回数据对象数组

\*@apiSuccess {String} respCode 返回码。 200成功,其他,失败  
 \*@apiSuccessExample {json} Success-Response:  
 \* HTTP/1.1 200 OK  
 \*{  
 \* "respCode":200,  
 \* "respMsg":"成功",  
 \* "data":[{  
 \* "id":"12",  
 \* "need\_name":"巴菲特",  
 \* "e\_mail":"10.20.17.22@mail.com",  
 \* "phone":"1870018912",  
 \* "company\_name":"上海德拓有限公司",  
 \* "desc":"xxxxxxx",  
 \* "password":"\*\*\*\*\*\*\*\*\*\*"  
 \* }]  
 \* }

\*/

public function GetUser(){//函数体}

实例4 post请求方式,接收参数为多参数

/\*\*

\* @author zhou.long

\* @api {post} /order/selectOrderInfo

\* @apiDescription 查询订单列表信息

\* @apiGroup Order APIs

\* @apiParam {String} userid 用户ID

\* @apiParam {String} [orderdstatus] 订单状态

\* @apiParam {String} pageNo 当前页

\* @apiParam {String} pageSize 每页条数

\* @apiParamExample {json} 请求参数格式:

\*{

\* "userid": "2aa98e233fe34ee59c44825fe33b7990",

\* "orderdstatus":"1",

\* "pageNo": "1",

\* "pageSize":"10"

\*}

\* @apiVersion 1.0.0

\* @apiSuccessExample {json} 正确返回值:

\*{

\* " data": {

\* "pageInfo": {

\* "total": 6,

\* "pageSize": 10,

\* "pageNum": 1

\* },

\* "list": [

\* {

\* "id": "c11e1dbf-36e3-4410-bef2-f83b22f51ba7",

\* "isNewRecord": false,

\* "shopid": "eaa57f8467c94a18a6569b9a7098fbb3",

\* "totalprice": 5140,

\* "buytime": "2019-08-06 21:26:31",

\* "userid": "9aamm98e23dd3fe34ee59c4fe33b8990",

\* "addressid": "3f9e2bd92d756ddcdd499ebuuuuuu9",

\* "orderdstatus": "1",

\* "address": "北京市朝阳区",

\* "name": "张三",

\* "phonenum": "131561131",

\* "isdaflut": "1",

\* "goodsResult": [

\* {

\* "goodid": "9998555f2ced47119f3a688889aaa4476",

\* "goodcount": "1",

\* "goodmoney": "4345",

\* "goodname": "测试商品",

\* "gooddesc": "这是一个测试案例",

\* "photoUrl": "https://localhost/2018-06-30/zhgc1530342624522.jpg",

\* },

\* ...

\* ]

\* },

\* ...

\* ]

\* }

\*

\* @apiErrorExample {json} 错误返回值:

\* {

\* "state":"0",

\* "message":"查询失败",

\* "data":""

\* }

\*@apiSampleRequest /projectName/order/selectOrderInfo

\* /

public function selectOrderInfo(){//函数体}

## 变量注释：

1. 全局变量**必须**进行注释；  
   2、final、static修饰的变量**必须**进行注释；

3、注释内容包括：简要说明；  
4、示例：

//每页的记录数

public static $EVERY\_PAGE = 15;

## 单个函数/方法的总行数

【推荐】单个方法的总行数不超过 80 行。 说明：包括方法签名、结束右大括号、方法内代码、注释、空行、回车及任何不可见字符的总 行数不超过 80 行。

代码逻辑分清主干功能和次要加工，个性和共性，次要加工逻辑单独出来成为额外方法，使主干功能代码更加清晰；共性逻辑抽取成为共性方法，便于复用和维护。

* 1. **http常见状态码**

200 请求成功 一般用于get

201 创建成功 一般用于post

202 更新成功 一般用于put

400 无效请求

401 地址不存在

403 禁止访问

404 请求资源不存在

500 内部错误

例如公共方法入下面所封装或者封装为其他方法，要按上面定义http 状态码

**function** show($status, $message, $data, $httpCode=200)  
{  
 $data=[  
 **'status'** => $status,  
 **'message'** => $message,  
 **'data'** => $data  
 ];  
 **return** json($data,$httpCode);  
}

**(18) 关于tp5框架里面不可预知的内部异常解决方案**

1.用try catch

{  
 **try**{  
 model(**'hui'**);  
 }**catch** (\Exception $e){  
 **return** show(0,$e->getMessage(),[],400);  
 }  
}

**访问控制器方法返回：**

**{**

**"status": 0,**

**"message": "class not exists:app\\common\\model\\Hui",**

**"data": []**

**}**

1. **新建一个ApihandleException继承tp5里面的handle类重写 render方法**

**然后在config配置文件配置上ApihandleException路径**

**'exception\_handle'** => **'app\common\lib\exception\ApiHandleException'**,

**<?php***/\*\*  
 \* Created by PhpStorm.  
 \* User: apple  
 \* Date: 2019-11-20  
 \* Time: 11:57  
 \*/***namespace** app\common\lib\exception;  
**use** think\exception\Handle;  
  
**class** ApiHandleException **extends** Handle  
{  
 **public $httpCode** = 500;  
 **public function** render(\Exception $e)  
 {  
 **return** show(0, $e->getMessage(), [], $this->**httpCode**);  
 }  
}

**控制器里：**

**public function** index()  
{  
 model(**'hui'**);  
}

**返回：**

**{**

**"status": 0,**

**"message": "class not exists:app\\common\\model\\Hui",**

**"data": []**

**}**

**(19) 主动抛出异常并定义httpCode码和业务code码**

1.新建一个ApiExceptions继承Exception重写构造方法，以下是ApiExceptios代码

**namespace** app\common\lib\exception;  
  
  
**use** think\Exception;  
  
  
**class** ApiExceptions **extends** Exception  
{  
 **public $message** = **''**; *//错误消息* **public $httpCode** = 500; *//http状态码* **public $code** = 0; *//业务码* **public function** \_\_construct($message = **""**, $httpCode = 500, $code = 0)  
 {  
 $this->**httpCode** = $httpCode;  
 $this->**message** = $message;  
 $this->**code** = $code;  
 }  
}

2.修改ApiHandleException类，拿到传入的httpCode和code 代码如下

**namespace** app\common\lib\exception;  
**use** think\exception\Handle;  
  
**class** ApiHandleException **extends** Handle  
{  
 **public $httpCode** = 500;  
 **public $code** = 0;  
 **public function** render(\Exception $e)  
 {  
 *//如果debug开启是后端调试模式用tp5自带的异常处理机制* **if** (config(**'app\_debug'**)) {  
 **return parent**::*render*($e);  
 }  
 *//如果异常属于ApiExceptions 拿到传入的code和httpCode* **if**($e **instanceof** ApiExceptions){  
 $this->**code** = $e->**code**;  
 $this->**httpCode** = $e->**httpCode**;  
 }  
 **return** show($this->**code**, $e->getMessage(), [], $this->**httpCode**);  
 }  
}

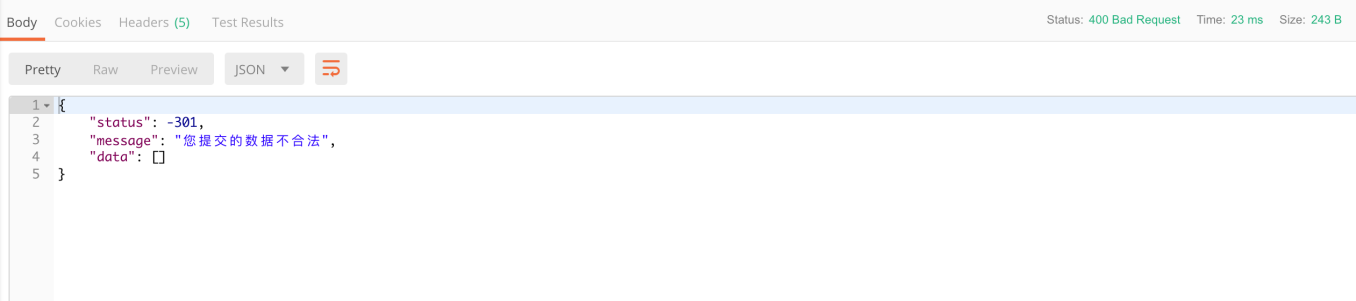
在控制器里引入

**use** app\common\lib\exception\ApiExceptions;

控制器方法代码：

**public function** index()  
 {  
*// model('hui');* **throw new** ApiExceptions(**'您提交的数据不合法'**,400,-301);  
 }

结果：



可以看到http状态码和code都已经改变

以上ApiExceptions和ApihandleException都在application\common\lib\exception下面，可以自己定义路径，自己修改正确

# 前端规范

## 编码

1. html、css、js统一采用**UTF-8**编码；  
    2、在HTML中指定编码 ；

## 框架：

1. 对于已有项目，不建议跟换框架；
2. 对于新开，采用交付统一的VUE框架，按照如下文档搭建开发环境，和打包发布；



## 缩进

1、统一采用**「Tab 键」**进行缩放，Tab是**4个空格，禁止**Tab与空格混用；

2、应保持代码层次明确，架构清晰，特别是**html**文件；

## 文件分离

1、原则上html，css，js需分离在不同文件中；

2、**禁止**在html文件中写css，css需写入单独的文件中，通过link导入；

3、较简单的js可写在html中，但必须在html**尾部**，但较复杂或较长的js需写入单独的文件中(包括但不仅限于地图操作、Echarts图表、3D绘制等)，使用script导入；

4、css，js的导入原则上应该在html文档**头部**，其中公共库（包括但不仅限于jquery.js、echarts.js等）必须在头部且先于其他js、css文件导入；

5、css样式原则上应**避免**写在html标签的style中；

## 公共css

1、原则上每个项目需要有且仅有一个公共css（bass.css、public.css等命名），公共css中用于定义项目通用的样式，且每个html**必须优先导入**公共css；

2、非公共css中，**禁止**直接对标签定义样式，如下代码仅能在公共css中：

body {

width: 98vw;

height: 60vw;

background: url(../images/background.png) no-repeat;

background-size: 100% 100%;

}

3、css中**禁止**使用!important

## 文件命名

1、文件命名不做强制规定，可采用驼峰命名、或者小写加中划线、小写加下划线等方式；

2、同一项目同一类型的文件（如css、js、html、php）等**必须**统一格式命名，导入的第三方库除外；

## html标签命名

1、标签id/class命名均采用单词字母小写，多个单词组成时，采用中划线-分隔，例如：input-name；

2、层级嵌套较深的情况下，标签id/class命名应体现出其层级，例如：login-input-name；

3、**同一标签的同一id/class，不得即用于css样式，又绑定js事件**；

4、绑定js事件的id/class，采用js-开头，例如：js-login-submit-button；

## JS变量以及函数命名

1、JS变量及函数采用驼峰命名，首字母小写，例如：submitNickname()；

2、内部带有ajax请求的函数，需以ajax开头，例如：ajaxGetUserInfo()；

## 注释

1、html文件中，每个大模块的div必须注释，注释内容为该div的简要说明；

2、js文件中，全局变量**必须**进行注释；

3、js文件中，函数、DOM时间等**必须**进行注释，注释内容包括简要说明、参数（名称、数据类型、简要说明）、返回值（名称、数据类型、简要说明），作者，创建时间等，例如：

/\*\*

\* 点击提交按钮提交数据

\*

\* @author zhou.long

\* @version 1.0

\* @createtime 2018-07-04

\*/

# 数据库规范

## 命名规范

1、库名、对象名（表名、列名、函数名、视图名、序列名、等对象名称）均采用小写字母，下划线，数字命名；禁止出现数字开头，禁止两个下划线中间只 出现数字。数据库字段名的修改代价很大，因为无法进行预发布，所以字段名称需要慎重考虑。

说明：MySQL 在 Windows 下不区分大小写，但在 Linux 下默认是区分大小写。因此，数据库名、 表名、字段名，都不允许出现任何大写字母，避免节外生枝。

2、同一项目的所有表名，应使用项目英文名称或者英文名称缩写加下划线开头，例如：追逃项目以ec\_开头，但避免用pg开头。

## 设计规范

1、多表中的相同列，必须保证列名一致，数据类型一致，如:user\_id；

2、表结构中字段定义的数据类型与应用程序中的定义应保持一致，表之间字段校对规则一致，避免报错或无法使用索引的情况发生；

3、原则上所有的表需要有id字段，与业务相关的表需要有updatetime（更新时间）、updateuser（更新人）字段；

4、业务上具有唯一特性的字段，即使是多个字段的组合，也必须建成唯一索引。即使在应用层做了非常完善的校验控制，只要没有唯一索引，根据墨菲定律，必然有脏数据产生。

## 注释规范

1、所有字段，**必须**进行注释；

2、对于有id字段的表，表的注释放在id字段中；

3、对于type、status等标识类型的字段，**必须注释每个类型**，例如：type 在逃人员类型：1-重点在逃人员，2-已注销在逃人员，3-已抓获在逃人员；

## 数据安全规范

1、数据库的连接信息不要随意存放，不在群内公布连接信息，如有需要私聊发送；

2、在数据库中严禁中drop进行表结构删除（自己建的表除外）；

3、在使用delete关键语句时，切记后面应该加条件并需要确认；

4、在使用update更新数据时，切记后面应该加条件并需要确认；

5、在做update、delete、insert最好先建立备份表，防止出现意外情况；

6、开发存储过程（函数）、触发器等等要先做好测试，不能直接在生产服务器上开发。

## 调优建议

1.不要使用count(列名)或count(常量)来替代count()，count()就是SQL92定义的标准统计行数的语法，跟数据库无关，跟NULL和非NULL无关；

**说明：count( )会统计NULL值（真实行数），而count(列名)不会统计。**

2、除了ETL程序，应该尽量避免向客户端返回大数据量；

3、尽量避免使用 select \* from table，使用具体字段列表代替\*，不要返回不相关的任何字段，同时避免表结构发生变化时出现问题；

4、在代码写分页查询逻辑时，若count为0应直接返回，避免执行后面的分页语句；

5、尽量避免使用delete 全表，请使用truncate代替；

6、需要判断是否有数据的情况，避免使用count(\*)进行判断，建议使用limit 1，速度更快;

7、尽量避免频繁创建和删除临时表；

# 服务器管理规范

## 口令规范

1. 服务器口令不得使用建议数字和字母，至少8位，且包含数字、大小写字母和特殊符号；
2. 不同权限的账号应使用不同的口令；
3. 口令应定期修改，最长不得超过三个月；

## 权限规范

1. 服务器应该有完善的用户权限，除管理员必要的操作外，不应直接使用root账户；