# RESTful API适配-基于thinkphp5.1

## 环境适配

整体配置：开发阶段，将app\_debug配置为true，生产环境中要改为false

// 应用调试模式  
'app\_debug' => **true**,

默认输出类型修改为json，这是标准API的返回格式

// 默认输出类型  
'default\_return\_type' => 'json',

配置为强制使用路由，确保访问模式单一

// 是否强制使用路由  
'url\_route\_must' => **true**,

配置异常处理默认入口



## 数据库配置

根据实际数据库情况，配置IP，数据库名，账号，密码，端口

## 路由配置

一条标准的路由，需要按照如下方式配置：

Route::方法(URL,路由);



1. 方法

支持get,post,delete,put四种方法，get主要用于读取信息，post主要用于修改和新增信息，delete用于删除信息，put暂时不用。

1. URL

url部分的格式为：模块名/版本号/API动作

模块名：对于RESTful API的后台程序，统一使用单模块，模块名为api

版本号：为了预留扩展，必须提前考虑版本控制

API目的：为了让客户端能够清楚了解API的目的，在命名的时候需要具备可读性

3、路由

路由部分的格式为：模块名/控制器/接口方法

其中控制器部分需要特别注意，如果控制器有多级目录，需要用目录名.控制器类名，中间用.连接，不能用/连接

接口方法与API动作保持一致，便于两边对接

## 自定义配置

在config目录下新增setting.php，在setting中可以设置自定义属性，比如token失效时间，图片完整路径等。

要在代码中读取配置，可以通过如下方式来读取，配置支持多级配置，用.分割。



在index.php中定义日志存放路径



## API发布格式

API完整命名：

<https://fycathy.com/api/version/action>

其中version，action需要根据实际情况修改

参数传递方式：

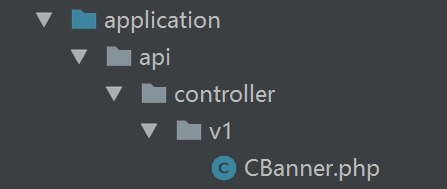
1. 如果只有一个参数，采用<https://fycathy.com/api/version/action/:param>
2. 如果有一组参数(为统一系列，可以采用数组来传递)，则统一在URL后面通过?连接数组，此时配置路由时，不需要设置参数，采用

<https://fycathy.com/api/version/action?param>=x,y,z....

## 控制器开发

1、控制器的作用：后台与API请求的唯一接口，通过路由指定每个控制器下的一个方法进行处理。一个控制器可以对应多条API，视功能而定。

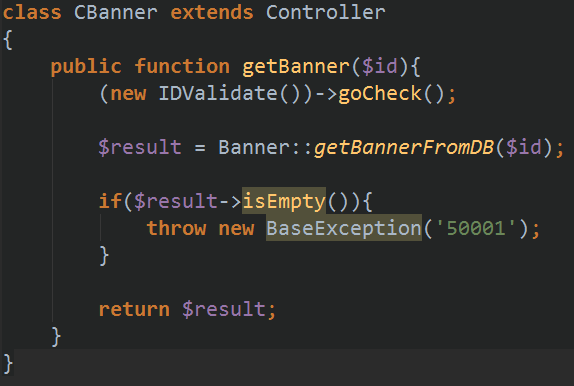
2、在如下目录建立控制器类，控制器的命名用C开头



1. 控制器方法的命名与API动作保持一致

.

1. 控制器的主要动作为：参数校验，通过模型配置数据库，校验结果



1. 控制器的返回

对于RESTful API软件来说，控制器返回的内容，会被封装成一条http响应消息返回给客户端，因此应该返回一条数组。

1. 前置方法

前置方法主要是为了校验权限，如果权限不足，则根本不允许进入控制器入口函数。为了支持前置方法，需要继承TH5的Controller类，而为了把一些权限方法抽象出来，可以定义一个BaseController extends Controller，在BaseController中定义鉴权方法，所有自定义控制器都继承BaseController

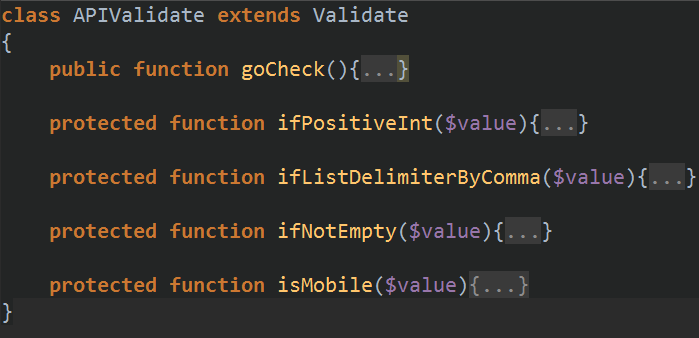
## 验证器

1. 验证器作用

验证器的主要目的是对参数进行校验，防止无效参数和恶意参数对系统的破坏

1. 验证器的基本方案

模块的验证器需要有一个APIValidate类继承TP5的系统类Validate，APIValidate类主要实现goCheck方法和一系列自定义校验规则（通用部分）



1. goCheck方法

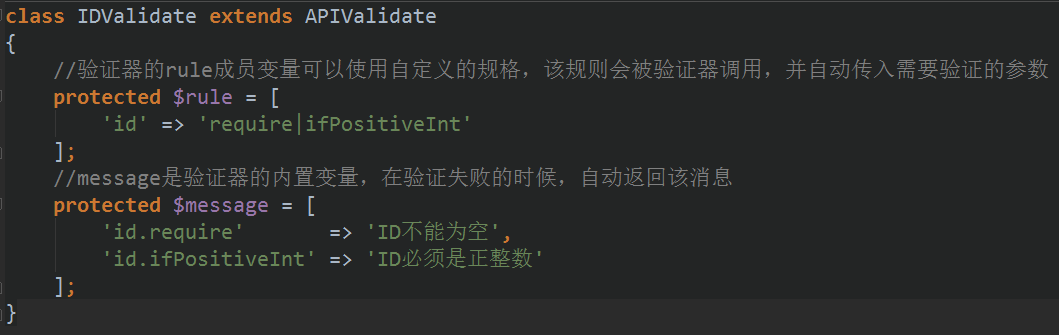


goCheck主要目的是通过Rquest方法获取所有参数，然后调用Validate基类的check方法进行参数的批量校验，

1. 其他校验规则

用if开头，体现校验的内容。

1. 除了APIValidate外，每个参数校验要定义自己专门的校验类，统一以XXValidate命名



专用校验类中定义$rule：用于设定校验规则；$message：用于设定每种校验规则如果不通过返回的消息。另外还可以定义专门的校验规则函数。

## 模型开发

1. 模型的作用是对数据库进行增删查改，模型的定义需要和数据表同名，采用驼峰命名，则在内部操作数据表的时候，TP5会自动定位到与模型同名的数据表操作。
2. 定义BaseModel，继承Model，在BaseModel中，定义公共方法，如对读取的URL地址进行完整化处理。
3. 每个模型主要有3部分内容：

a.隐藏或显示字段：通过定义$hidden或$visible来决定隐藏或者显示某些字段

b.关联查询：以query开头，后面接关联的数据库名字，如



c.提供给控制器的查询接口

d.获取器

1. 关联查询

一对多关联

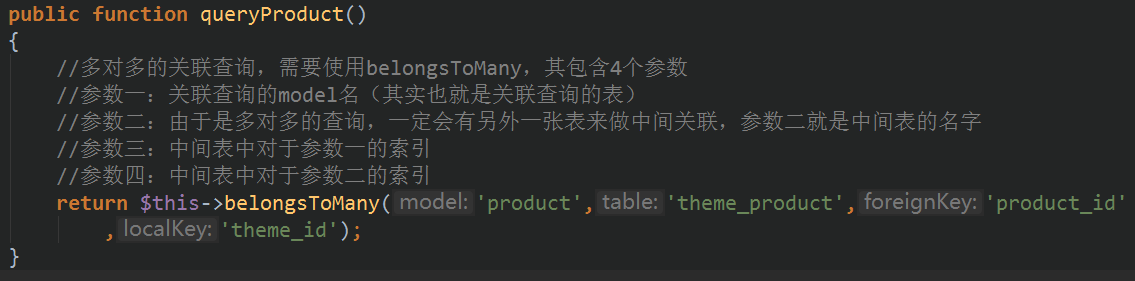


一对一关联





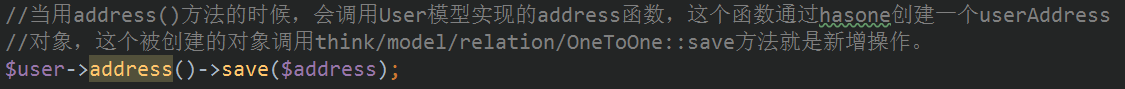
多对多关联



1. 关联查询函数

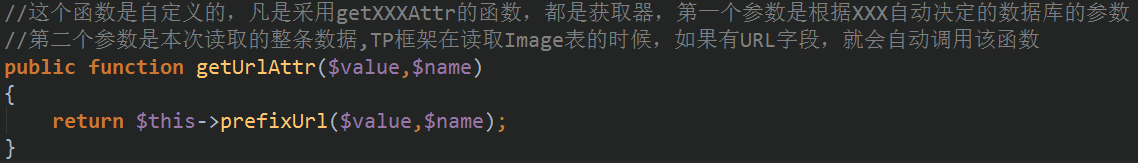
关联查询函数有两种用法：

1. 把函数名作为字符串，放到with的数组参数里，然后进行链式查询，这种方式主要是为了查询关联表的一些具体表项；
2. 通过model直接调用，这种方式主要是为了获得整个关联表，从而对其进行新增操作，如



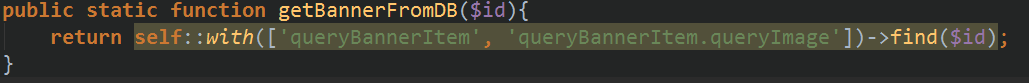
1. 获取器

获取器的目的是自动把获取器定义的字段进行某种修改，比如从数据库读取的图片url地址是相对地址，那么可以定义获取器把读取出来的url完整化处理



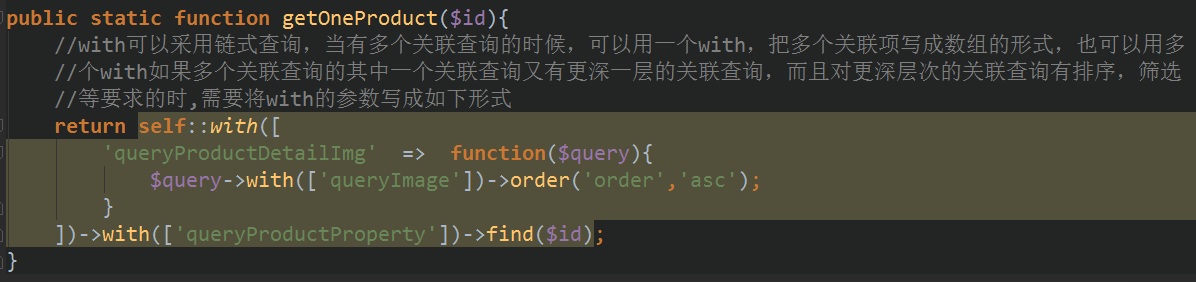
1. 数据库查询

为了实现更加灵活的查询，统一采用构造数据库查询器进行查询，而不用模型自带的查询函数，单条查询使用find，由于只查询一条，必然是通过主键进行查询，所以需要传入主键。



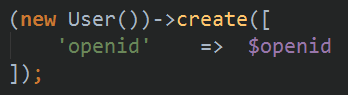
通过with可以指定关联查询项，with接收一个数组，数组的每项为一个关联项，还可以设置二级关联，通过.连接。

如果要对二级关联进行排序，则需要采用下面的方式

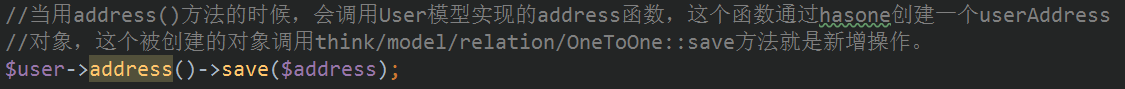


1. 数据库新增

对一个数据库表新增一条记录，必须使用这个数据库表的对象来操作，比如先new一个User对象，然后通过create方法，把需要新增的参数写上，也可以传入数组。

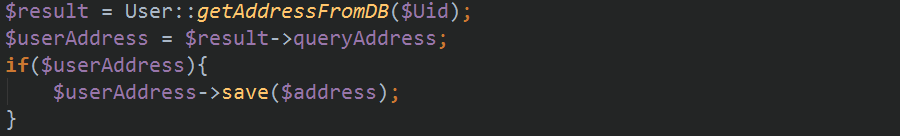


如果是通过关联查询找到一个数据库表，也可以通过create或save方法来新增



1. 数据库修改

对某一条数据进行修改，一定要通过查询的方式获取到对象，然后用对象调用save方法，如果是二级查询的结果，可以用一级查询结果->关联查询方法名字，来获取二级查询数据表的对象，如：



1. 模型的返回值

find返回的就是模型对象本身，因此控制器在拿到返回值后，需要直接判断是否为空，而不是用isempty方法。

Select返回的是数据集，需要使用isempty来验证是否为空，并可以用数据集的方法进行格式化

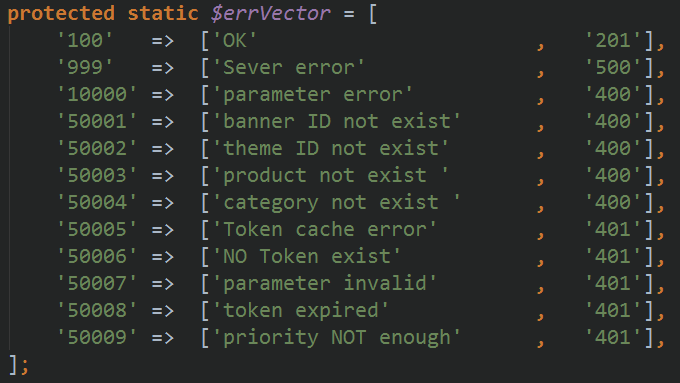
## 异常处理

1. 异常的目的

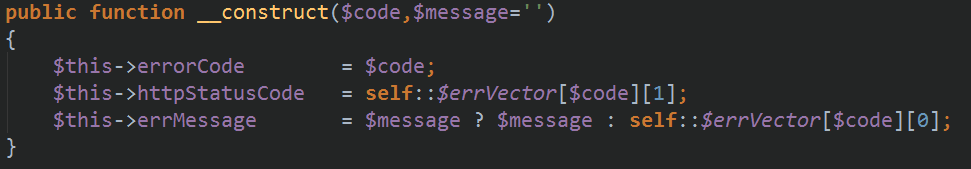
通过定义异常，可以把软件运行中的一些错误，特别是和客户端有关的错误，通过异常的方式返回给客户端，从而帮助客户端更加正确地调用API

因此自定义的异常归根接底是一种定义消息的机制，通过定义http码，错误码，错误消息等实现错误消息的语义性。

1. 异常处理的生成由一个BaseException类来处理，其继承自Exception类，在BaseExcetion中定义所有异常消息：特别注意，成功的消息也可以定义成异常，但是在抛出一个成功异常的时候，只能在控制器方法的return语句中抛出，否则后面的语句就不会执行。



通过构造函数生成消息体：



构造函数的第二个参数是消息，如果不传入消息，就使用错误码对应的消息，如果传入了消息，就显示外部传入的消息

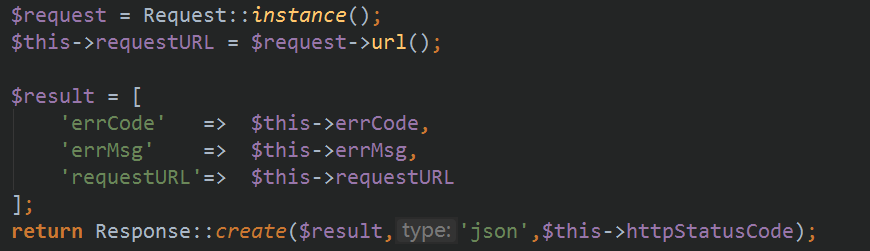
1. 异常的获取也必须继承Handle类，并重写render函数，接收一个全局异常



在render中判断异常是否是自定义的异常，如果是，就按照自定义处理，如果不是，就记录日志，提交到TP5的render处理（调试模式下）

对于自定义异常，需要返回错误码，错误消息，并回送客户端发送的url

Render在return后，软件本身的代码不会再执行，所以return的格式要使用Response::create函数来格式化成一个标准的json格式。

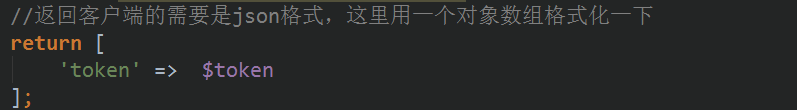


## 公共方法

统一写到common.php中。

## 数据格式

1. 从外部获取的数据，统一用json\_decode转换为数组，方便处理
2. 发送给客户端的数据，必须是JSON格式。如果return的是一个对象或数组，则TP5会自动转换为JSON，如果return是一个字符串，则修成数组格式。如



## 附录：数据库操作方法

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| Get | 查找单条记录 |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |