tp5.1 API 自定义全局异常处理（上）

我们接着重构 tp5.1 参数校验层的项目进行下面的演示。

现在我们来假设这一种情况，客户端传来了 id 为 50，由于 50 是正整数，所以通过了参数校验，但我们的数据库中没有 id 号为 50 的user，这时候我们就需要进行相应的异常处理。注：在Restful API开发中，查询不到数据也可认为是异常

为了演示，我们手动在User控制器中手动抛出一个异常:

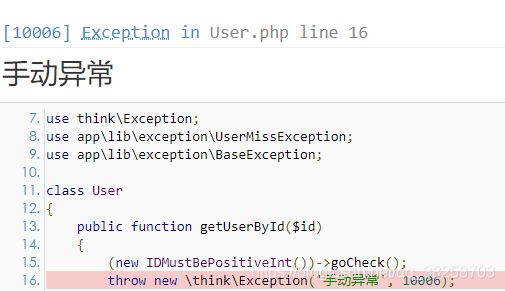
public function getUserById($id)

{

(new IDMustBePositiveInt())->goCheck();

throw new \think\Exception('手动异常', 10006);

}

假如我们控制器中不做处理，那么就会抛给全局异常处理器处理，并返回以下 html 页面：  


这个页面适合后端调试，但是不适合给客户端来看，尤其是作为接口的返回来说。  
那么我们在设计接口的时候该如何向客户端返回错误信息呢？  
基于 RESTFul 规范，我们需要定义一个统一的错误返回消息。  
我们改写一下控制器中的代码：

public function getUserByid($id){

(new IDMustBePositiveInt())->goCheck();

try{

throw new \think\Exception('手动异常', 10006);

}

catch(Exception $e)

{

$err = [

'error\_code' => 10001,

'msg' => $e->getMessage()

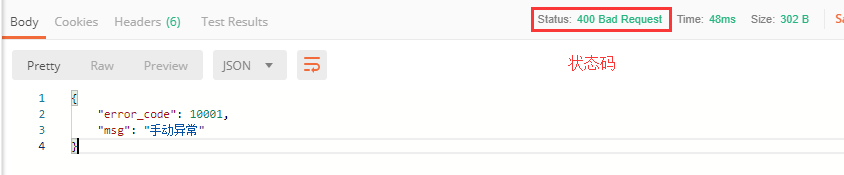
];

return json($err,400); // 注意不能直接返回数组，而应该用json包一下

}

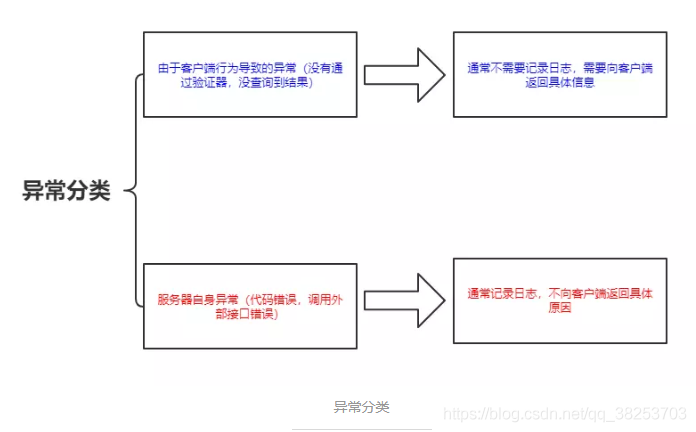
}

返回的页面：



这样我们这种直白的方法就写出来了，但我们反思一下，如果每一个控制器我们都要这样繁琐地处理异常，那么我们今后编写代码的思路一定难以保证十分流畅，而是会在这些异常的处理上耗费大量精力。而且这个仅仅是一个示例，实际上我们很多情况下是不可预知是否会有异常的，可能还会返回 tp5 自己的错误的网页，对于我们 API 来说是不合适的。

现在我们花了大量的篇幅展示了一种错误的、复用性差的直白写法，比起直接展示最终的结果，演示这些错误的写法我认为也是很有必要的，因为这是我们一步一步思路的体现。重构代码不是一蹴而就的，期间代码的写法也会越来越抽象，所以我们需要静下心来，不断地完善。

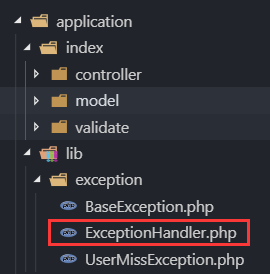
我们先来梳理一下异常的分类：  


tp5.1 有一个全局异常处理类，如果我们想自己实现上面的分类，需要覆盖和重写默认的全局异常处理类。

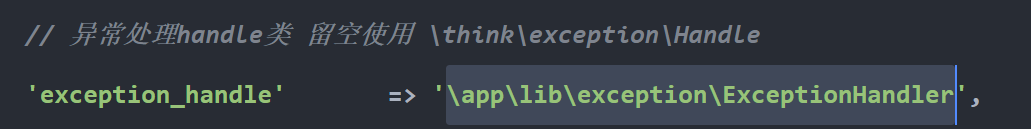
我们现在在 api 模块的同级下新建一个 lib 文件夹，再新建一个 exception 文件夹。

（我们想让这个 exception 里的类是一个通用的，可以供很多模块使用的一个类库。）

新建 ExceptionHandler 的 php class，并继承 Handle 类。



并且重新指定 tp5.1 的全局异常处理类：  
在 config.php 文件中的 'exception\_handle’字段输入我们自定义的处理器的命名空间：  
app\lib\exception\ExceptionHandler



代码如下：

<?php

namespace app\lib\exception;

use think\exception\Handle;

class ExceptionHandler extends Handle {

public function render(Exception $e){ // 重写render方法

return json('是否重写成功');

}

}

User控制器代码：

public function getUserById($id)

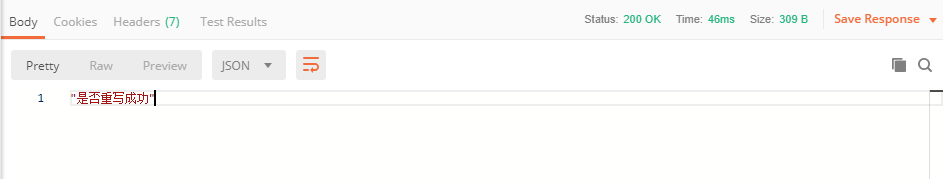
{

(new IDMustBePositiveInt())->goCheck();

throw new \think\Exception('手动异常', 10006);

｝

结果如下：



现在我们继续来写 render 方法来区分前面提到的两种异常：

其中有一种异常需要向客户端返回具体信息，我们在初始化对象的成员变量赋值最好的方式是通过构造函数来初始化赋值操作，更加符合面向对象的基本特性。则实例化的时候即可传入参数，相应的构造函数（父类中创建即可）会进行处理。需要新建一个 BaseException 类：

<?php

namespace app\lib\exception;

use Exception;

class BaseException extends Exception

{

public $code = 400; // HTTP 状态码 404,200...

public $msg = '参数错误'; // 错误信息具体

public $error\_code = 10000; // 自定义错误码

public function \_\_construct($params = [])

{

if (!is\_array($params)) {

return; // 如果没有传入数组，那么就是使用默认的 code、msg 和 errorCode

}

if (array\_key\_exists('code', $params)) {

$this->code = $params['code'];

}

if (array\_key\_exists('msg', $params)) {

$this->msg = $params['msg'];

}

if (array\_key\_exists('error\_code', $params)) {

$this->error\_code = $params['error\_code'];

}

}

}

只要是继承于 BaseException 的异常类都是我们自定义的类，且需要返回给客户端信息。  
我们这样修改 render 方法：

<?php

namespace app\lib\exception;

use Exception;

use think\exception\Handle;

class ExceptionHandler extends Handle

{

private $code;

private $msg;

private $error\_code;

public function render(Exception $e)

{

if ($e instanceof BaseException) {

$this->code = $e->code;

$this->msg = $e->msg;

$this->error\_code = $e->error\_code;

} else {

$this->code = 500;

$this->msg = '服务器内部异常';

$this->error\_code = 999;

}

$request = request();

$result = [

'msg' => $this->msg,

'error\_code' => $this->error\_code,

'request\_url' => $request->url()

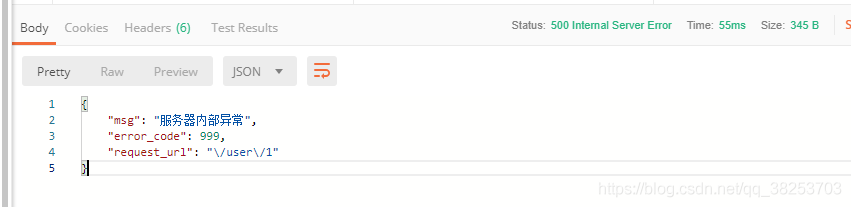
];

return json($result, $this->code);

}

}

运行后发现：



我们来测试一下BaseException，User控制器代码为：

public function getUserById($id)

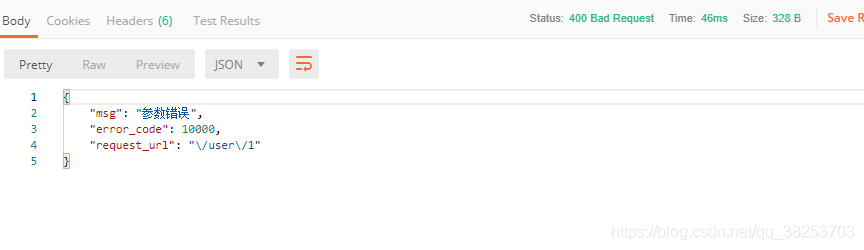
{

(new IDMustBePositiveInt())->goCheck();

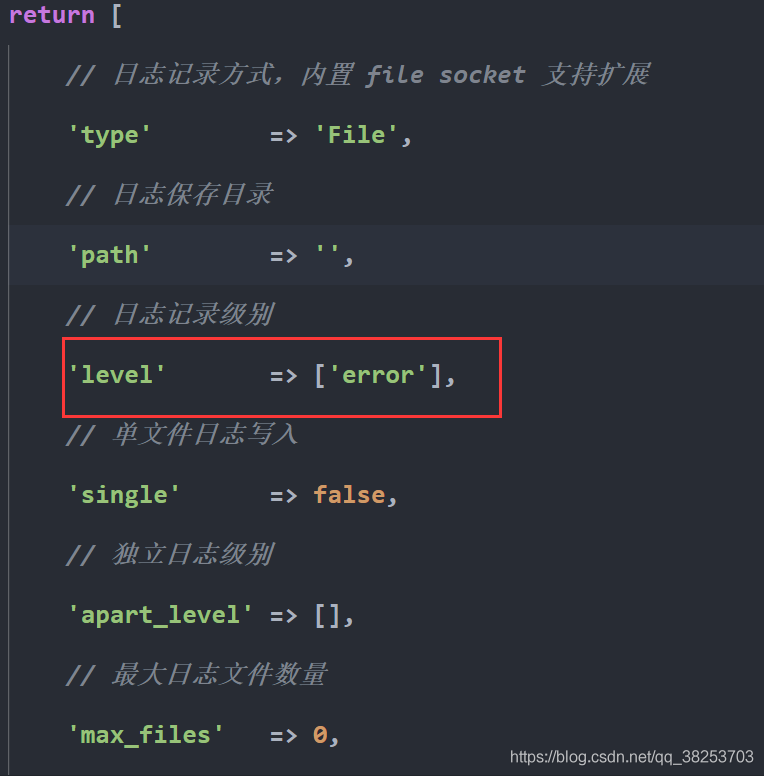
throw new BaseException(); //抛出一个BaseException的异常

｝

结果:



实现记录日志错误信息：  
在config目录下，找到Log.php，开启一下配置信息：



在ExceptionHandler类render的方法下，写入记录日志代码：

Log::record($e->getMessage(),'error');

这里要注意引入

use think\facade\Log;

即可看到服务器内部异常时会记录日志

tp5.1自带的错误页面是非常清晰的，而客户端的开发者需要一个简化的 json 信息，前面的上中两篇已经具体介绍了如何返回 json 信息，但是作为客户端开发我们还是希望使用 tp5.1 自带的错误页面来定位错误，这样如何调和呢？我们可以通过一个开关来进行操作，开关开的时候返回具体的包含错误信息的 html 页面，如果开关是关闭的我们就返回 json。

现在我们来分析一下思路，我们通过自己写的 render 方法覆盖了父类的 render 方法，所以当我们需要重新调用父类 render 方法的时候，我们重新将 else 里面的代码补充一下：

<?php

namespace app\lib\exception;

use Exception;

use think\exception\Handle;

use think\facade\Log;

class ExceptionHandler extends Handle

{

private $code;

private $msg;

private $error\_code;

public function render(Exception $e)

{

if ($e instanceof BaseException) {

$this->code = $e->code;

$this->msg = $e->msg;

$this->error\_code = $e->error\_code;

} else {

if(config('app\_debug')){

return parent::render($e);

}

$this->code = 500;

$this->msg = '服务器内部异常';

$this->error\_code = 999;

}

$request = request();

$result = [

'msg' => $this->msg,

'error\_code' => $this->error\_code,

'request\_url' => $request->url()

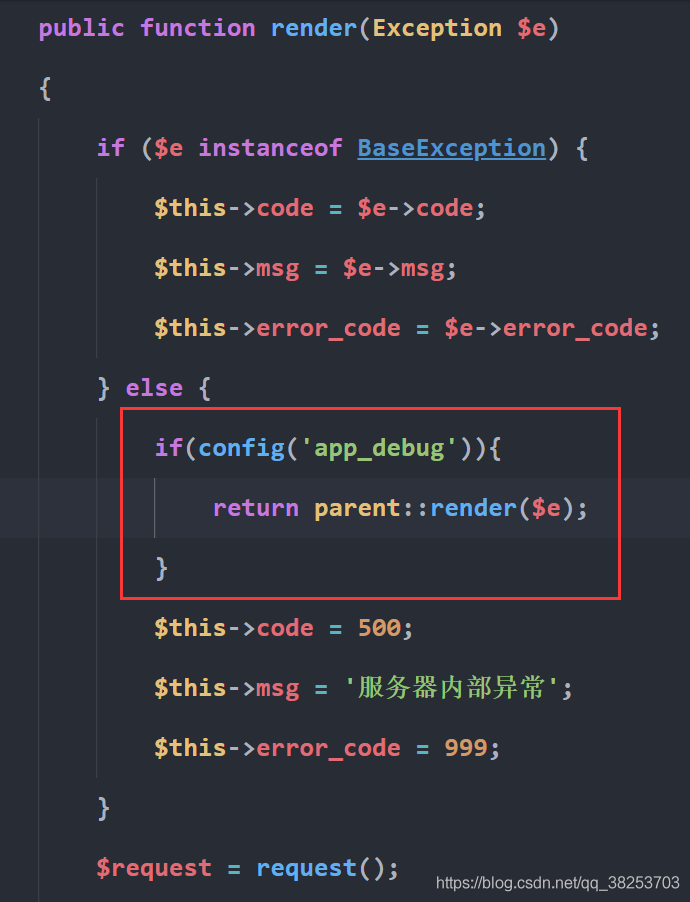
];

Log::record($e->getMessage(),'error');

return json($result, $this->code);

}

}



即，当在开发模式下的时候，我们可以通过配置文件中的"****app\_debug****"控制返回 Handle 类的 render 方法的结果，这样我们就可以看到一个页面了，而生产模式的时候，可以关闭该模式，就会返回 json 了。