## TP6异常自定义捕捉和消息告警

使用tp6 自定义异常处理，将404、500等错误进行处理  
render渲染方法重写：  
1、如果是ajax请求，则返回对应状态,调试模式下返回堆栈  
2、如果是页面请求，则返回对应错误模板页面渲染。  
  
report报告方法重写：  
1、Undefined index、offset、variable比较常见，屏蔽以不推送消息，采用原生记录即可  
2、推送消息走异步，防止并发出现响应变慢。记录异常的前几个堆栈的参数，方便开发验证问题  
3、推送异常，走默认原生记录，防止死循环

1. <?php
2. /\*
3. \* @Description: 应用异常处理类
4. \* @Author: KingFer
5. \* @Date: 2019-08-13 17:24:45
6. \*/
7. namespace app;
8. use app\common\helper\QueueHelper;
9. use ErrorException;
10. use Exception;
11. use InvalidArgumentException;
12. use ParseError;
13. use PDOException;
14. use think\db\exception\DataNotFoundException;
15. use think\db\exception\ModelNotFoundException;
16. use think\exception\ClassNotFoundException;
17. use think\exception\Handle;
18. use think\exception\HttpException;
19. use think\exception\HttpResponseException;
20. use think\exception\RouteNotFoundException;
21. use think\exception\ValidateException;
22. use think\Response;
23. use Throwable;
24. use TypeError;
25. class ExceptionHandle extends Handle
26. {
27. /\*\*
28. \* 不需要记录信息（日志）的异常类列表
29. \* @var array
30. \*/
31. protected $ignoreReport = [
32. HttpResponseException::class,
33. ValidateException::class,
34. ];
35. protected $error\_num;
36. public function \_\_construct()
37. {
38. $this->error\_num = 'Y' . makeRandCode(13) . date('mdHis');
39. parent::\_\_construct(app());
40. }
41. /\*\*
42. \* Render an exception into an HTTP response.
43. \*
44. \* @access public
45. \* @param \think\Request   $request
46. \* @param Throwable $e
47. \* @return Response
48. \*/
49. public function render($request, Throwable $e): Response
50. {
51. if (!$this->isIgnoreReport($e)) {
52. // 添加自定义异常处理机制
53. // 参数验证错误
54. if ($e instanceof ValidateException) {
55. return json($e->getError(), 422);
56. }
57. // ajax请求404异常 , 不返回错误页面
58. if (($e instanceof ClassNotFoundException || $e instanceof RouteNotFoundException) && request()->isAjax()) {
59. return json(['msg' => env('app\_debug') ? $e->getMessage() : '当前请求资源不存在，请稍后再试', 'code' => 404, 'data' => []]);
60. }
61. //类未找到, 返回错误页面
62. if (($e instanceof ClassNotFoundException || $e instanceof RouteNotFoundException) || ($e instanceof HttpException && $e->getStatusCode() == 404)) {
63. return view(root\_path() . 'public/error/404.html', ['e' => $e], 404);
64. }
65. // ajax请求500异常, 不返回错误页面
66. if (($e instanceof Exception || $e instanceof PDOException || $e instanceof HttpException || $e instanceof InvalidArgumentException || $e instanceof ErrorException || $e instanceof ParseError || $e instanceof TypeError) && request()->isAjax()) {
67. return json(['msg' => env('app\_debug') ? $e->getMessage() : '系统异常，请稍后再试', 'code' => 500, 'data' => env('app\_debug') ? $e->getTrace() : []]);
68. }
69. // 内部异常 , 返回错误页面
70. if ($e instanceof Exception || $e instanceof PDOException || $e instanceof InvalidArgumentException || $e instanceof ErrorException || $e instanceof ParseError || $e instanceof TypeError || ($e instanceof HttpException && $e->getStatusCode() == 500)) {
71. return view(root\_path() . 'public/error/500-2.html', ['e' => $e, 'error\_num' => $this->error\_num], 500);
72. }
73. }
74. // 其他错误交给系统处理
75. return parent::render($request, $e);
76. }
77. /\*\*
78. \* 记录异常信息（包括日志或者其它方式记录）
79. \*
80. \* @access public
81. \* @param  Throwable $exception
82. \* @return void
83. \*/
84. public function report(Throwable $e): void
85. {
86. if (!$this->isIgnoreReport($e)) {
87. if (
88. $e instanceof ClassNotFoundException  || ($e instanceof HttpException && $e->getStatusCode() == 404)
89. || stripos($e->getMessage(), "Undefined index:") !== false
90. || stripos($e->getMessage(), "Undefined offset:") !== false
91. || stripos($e->getMessage(), "Undefined variable:") !== false
92. ) {
93. //这是由Mongo日志记录抛出的异常 , 不能再把异常推进队列中, 否则会造成死循环
94. //这里用原生report方法 , 防止死循环
95. //类不存在的也不记录mongo日志
96. //变量未定义 , 也不记录mongo日志
97. parent::report($e);
98. return;
99. }
100. try {
101. //取出异常的部分堆栈参数，方便排查问题
102. $args = $e->getTrace();
103. $error\_args = array\_slice($args, 0, 5);
104. foreach ($error\_args as $k => $error) {
105. if (!isset($error['file'])) {
106. unset($error\_args[$k]);
107. } else if (isset($error['file']) && (stripos($error['file'], 'thinkphp') || stripos($error['file'], 'index.php'))) {
108. unset($error\_args[$k]);
109. }
110. }
111. //异常信息丢到队列处理
112. QueueHelper::pushQueue("app\job\LogErrorMsg", "log\_error\_msg", [
113. 'msg\_type' => 'exception',
114. 'system' => env('system\_name'),
115. 'error\_num' => $this->error\_num,
116. 'error\_handle' => [
117. 'file' => $e->getFile(),
118. 'line' => $e->getLine(),
119. 'msg' => $e->getMessage(),
120. 'trace' => $e->getTraceAsString(),
121. 'trace\_args' => $error\_args,
122. ],
123. ]);
124. return;
125. } catch (\ErrorException $error) {
126. //入队异常 , 用原生report方法 , 防止死循环
127. parent::report($e);
128. return;
129. } catch (\Exception $error) {
130. //入队异常 , 用原生report方法 , 防止死循环
131. parent::report($e);
132. return;
133. }
134. // 使用内置的方式记录异常日志
135. parent::report($e);
136. return;
137. }
138. }
139. }