**你真的会用 try catch吗？**

 说起PHP异常处理，大家首先会想到try-catch，那好，我们先看一段程序吧：有一个test.php文件，有一段简单的PHP程序，内容如下，然后命令行执行：php test.php

1. 1 <?php
2. 2 $num = 0;
3. 3 try {
4. 4 echo 1/$num;
5. 5
6. 6 } catch (Exception $e){
7. 7 echo $e->getMessage();
8. 8 }
9. 9 ?>

　　我的问题是：这段程序能正确的捕捉到除0的错误信息吗？  
　　如果你回答能，那你就把这篇文章看完吧！应该能学点东西。

本文章分5个部分介绍我的异常处理的理解：

[一、异常与错误的概述](https://www.cnblogs.com/zyf-zhaoyafei/p/6928149.html#excetion1)

[二、ERROR的级别](https://www.cnblogs.com/zyf-zhaoyafei/p/6928149.html#excetion2)

[三、PHP异常处理中的黑科技](https://www.cnblogs.com/zyf-zhaoyafei/p/6928149.html#excetion3)

[四、巧妙的捕获错误和异常](https://www.cnblogs.com/zyf-zhaoyafei/p/6928149.html#excetion4)

[五、自定义异常处理和异常嵌套](https://www.cnblogs.com/zyf-zhaoyafei/p/6928149.html#excetion6)

[六、PHP7中的异常处理](https://www.cnblogs.com/zyf-zhaoyafei/p/6928149.html#excetion5)

一、异常与错误的概述

        PHP中什么是异常：  
　　程序在运行中出现不符合预期的情况，允许发生（你也不想让他出现不正常的情况）但他是一种不正常的情况，按照我们的正常逻辑本不该出的错误，但仍然会出现的错误，属于逻辑和业务流程的错误，而不是编译或者语法上的错误。  
  
　　PHP中什么是错误：  
　　属于php脚本自身的问题，大部分情况是由错误的语法，服务器环境导致，使得编译器无法通过检查，甚至无法运行的情况。warning、notice都是错误，只是他们的级别不同而已，并且错误是不能被try-catch捕获的。

　　上面的说法是有前提条件的：  
　　在PHP中，因为在其他语言中就不能这样下结论了，也就是说异常和错误的说法在不同的语言有不同的说法。在PHP中任何自身的错误或者是非正常的代码都会当做错误对待，并不会以异常的形式抛出，但是也有一些情况会当做异常和错误同时抛出(据说是，我没有找到合适的例子)。也就是说，你想在数据库连接失败的时候自动捕获异常是行不通的，因为这就不是异常，是错误。但是在java中就不一样了，他会把很多和预期不一致的行为当做异常来进行捕获。

　　PHP异常处理很鸡肋？  
　　在上面的分析中我们可以看出，PHP并不能主动的抛出异常，但是你可以手动抛出异常，这就很无语了，如果你知道哪里会出问题，你添加if else解决不就行了吗，为啥还要手动抛出异常，既然能手动抛出就证明这个不是异常，而是意料之中。以我的理解，这就是PHP异常处理鸡肋的地方（不一定对啊）。所以PHP的异常机制不是那么的完美，但是使用过框架的同学都知道有这个情况：你在框架中直接写开头那段php“自动”捕获异常的代码是可以的，这是为什么？看过源码的同学都知道框架中都会涉及三个函数：register\_shutdown\_function，set\_error\_handler，set\_exception\_handler后面我会重点讲解着三个黑科技，通过这几个函数我们可以实现PHP假自动捕获异常和错误。

二、ERROR的级别

　　只有熟悉错误级别才能对错误捕捉有更好的认识。 ERROR有不同的错误级别，我之前的一篇文章中有写到：http://www.cnblogs.com/zyf-zhaoyafei/p/3649434.html  
　　下面我再总结性的给出这几类错误级别：

1. 1 Fatal Error:致命错误（脚本终止运行）
2. 2 E\_ERROR // 致命的运行错误，错误无法恢复，暂停执行脚本
3. 3 E\_CORE\_ERROR // PHP启动时初始化过程中的致命错误
4. 4 E\_COMPILE\_ERROR // 编译时致命性错，就像由Zend脚本引擎生成了一个E\_ERROR
5. 5 E\_USER\_ERROR // 自定义错误消息。像用PHP函数trigger\_error（错误类型设置为：E\_USER\_ERROR）
6. 6
7. 7 Parse Error：编译时解析错误，语法错误（脚本终止运行）
8. 8 E\_PARSE //编译时的语法解析错误
9. 9
10. 10 Warning Error：警告错误（仅给出提示信息，脚本不终止运行）
11. 11 E\_WARNING // 运行时警告 (非致命错误)。
12. 12 E\_CORE\_WARNING // PHP初始化启动过程中发生的警告 (非致命错误) 。
13. 13 E\_COMPILE\_WARNING // 编译警告
14. 14 E\_USER\_WARNING // 用户产生的警告信息
15. 15
16. 16 Notice Error：通知错误（仅给出通知信息，脚本不终止运行）
17. 17 E\_NOTICE // 运行时通知。表示脚本遇到可能会表现为错误的情况.
18. 18 E\_USER\_NOTICE // 用户产生的通知信息。

　　由此可知有5类是产生ERROR级别的错误，这种错误直接导致PHP程序退出。  
　　可以定义成：

1 ERROR = E\_ERROR | E\_CORE\_ERROR | E\_COMPILE\_ERROR | E\_USER\_ERROR | E\_PARSE

三、PHP异常处理中的黑科技

　　前面提到框架中是可以捕获所有的错误和异常的，之所以能实现应该是使用了黑科技，哈哈！其实也不是什么黑科技，主要是三个重要的函数：

　　1：set\_error\_handler()  
　　看到这个名字估计就知道什么意思了，这个函数用于捕获错误，设置一个用户自定义的错误处理函数。

1. 1 <?php
2. 2 set\_error\_handler('zyferror');
3. 3 function zyferror($type, $message, $file, $line)
4. 4 {
5. 5 　　var\_dump('<b>set\_error\_handler: ' . $type . ':' . $message . ' in ' . $file . ' on ' . $line . ' line .</b><br />');
6. 6 }
7. 7 ?>

　　当程序出现错误的时候自动调用此方法，不过需要注意一下两点：第一，如果存在该方法，相应的error\_reporting()就不能在使用了。所有的错误都会交给自定义的函数处理。第二，此方法不能处理以下级别的错误：E\_ERROR、 E\_PARSE、 E\_CORE\_ERROR、 E\_CORE\_WARNING、 E\_COMPILE\_ERROR、 E\_COMPILE\_WARNING，set\_error\_handler() 函数所在文件中产生的E\_STRICT，该函数只能捕获系统产生的一些Warning、Notice级别的错误。  
　　并且他有多种调用的方法：

1. 1 <?php
2. 2 // 直接传函数名 NonClassFunction
3. 3 set\_error\_handler('function\_name');
4. 4
5. 5 // 传 class\_name && function\_name
6. 6 set\_error\_handler(array('class\_name', 'function\_name'));
7. 7 ?>

 　　2：register\_shutdown\_function()  
　　捕获PHP的错误：Fatal Error、Parse Error等，这个方法是PHP脚本执行结束前最后一个调用的函数，比如脚本错误、die()、exit、异常、正常结束都会调用，多么牛逼的一个函数啊！通过这个函数就可以在脚本结束前判断这次执行是否有错误产生，这时就要借助于一个函数：error\_get\_last()；这个函数可以拿到本次执行产生的所有错误。error\_get\_last();返回的信息：  
　　[type]           - 错误类型  
　　[message] - 错误消息  
　　[file]              - 发生错误所在的文件  
　　[line]             - 发生错误所在的行

1. 1 <?php
2. 2 register\_shutdown\_function('zyfshutdownfunc');
3. 3 function zyfshutdownfunc()
4. 4 {
5. 5 if ($error = error\_get\_last()) {
6. 6 var\_dump('<b>register\_shutdown\_function: Type:' . $error['type'] . ' Msg: ' . $error['message'] . ' in ' . $error['file'] . ' on line ' . $error['line'] . '</b>');
7. 7 }
8. 8 }
9. 9 ?>

 　　通过这种方法就可以巧妙的打印出程序结束前所有的错误信息。但是我在测试的时候我发现并不是所有的错误终止后都会调用这个函数，可以看下面的一个测试文件，内容是：

1. 1 <?php
2. 2 register\_shutdown\_function('zyfshutdownfunc');
3. 3 function zyfshutdownfunc()
4. 4 {
5. 5 if ($error = error\_get\_last()) {
6. 6 var\_dump('<b>register\_shutdown\_function: Type:' . $error['type'] . ' Msg: ' . $error['message'] . ' in ' . $error['file'] . ' on line ' . $error['line'] . '</b>');
7. 7 }
8. 8 }
9. 9 var\_dump(23+-+); //此处语法错误
10. 10 ?>

 　　自己可以试一下，你可以看到根本就不会触发zyfshutdownfunc()函数，其实这是一个语法错误，直接报了一个：

1. 1 <?php
2. 2 Parse error: syntax error, unexpected ')' in /www/mytest/exception/try-catch.php on line 71
3. 3 ?>

　　由此引出一个奇葩的问题：问什么不能触发，为什么框架中是可以的？其实原因很简单，只在parse-time出错时是不会调用本函数的。只有在run-time出错的时候，才会调用本函数，我的理解是语法检查器前没有执行register\_shutdown\_function()去把需要注册的函数放到调用的堆栈中，所以就根本不会运行。那框架中为什么任何错误都能进入到register\_shutdown\_function()中呢，其实在框架中一般会有统一的入口index.php，然后每个类库文件都会通过include \*\* 的方式加载到index.php中，相当与所有的程序都会在index.php中聚集，同样，你写的具有语法错误的文件也会被引入到入口文件中，这样的话，调用框架，执行index.php，index.php本身并没有语法错误，也就不会产生parse-time错误，而是 include 文件出错了，是run-time的时候出错了，所以框架执行完之后就会触发register\_shutdown\_function();  
　　所以现在可是试一下这个写法，这样就会触发zyfshutdownfunc()回调了：

1. 1 a.php文件
2. 2 <?php
3. 3 　　// 模拟语法错误
4. 4 　　var\_dump(23+-+);
5. 5 ?>
6. 6
7. 7 b.php文件
8. 8 <?php
9. 9 register\_shutdown\_function('zyfshutdownfunc');
10. 10 function zyfshutdownfunc()
11. 11 {
12. 12 if ($error = error\_get\_last()) {
13. 13 var\_dump('<b>register\_shutdown\_function: Type:' . $error['type'] . ' Msg: ' . $error['message'] . ' in ' . $error['file'] . ' on line ' . $error['line'] . '</b>');
14. 14 }
15. 15 }
16. 16 require 'a.php';
17. 17 ?>

 　　3：set\_exception\_handler()  
　　设置默认的异常处理程序，用在没有用try/catch块来捕获的异常，也就是说不管你抛出的异常有没有人捕获，如果没有人捕获就会进入到该方法中，并且在回调函数调用后异常会中止。看一下用法：

1. 1 <?php
2. 2 set\_exception\_handler('zyfexception');
3. 3 function zyfexception($exception)
4. 4 {
5. 5 var\_dump("<b>set\_exception\_handler: Exception: " . $exception->getMessage() . '</b>');
6. 6 }
7. 7 throw new Exception("zyf exception");
8. 8 ?>

四、巧妙的捕获错误和异常  
　　1：把错误以异常的形式抛出(不能完全抛出)  
　　　　由上面的讲解我们知道，php中的错误是不能以异常的像是捕获的，但是我们需要让他们抛出，已达到扩展 try-catch的影响范围，我们前面讲到过set\_error\_handler() 方法，他是干嘛用的，他是捕获错误的，所以我们就可以借助他来吧错误捕获，然后再以异常的形式抛出，ok，试试下面的写法：

　　1：把错误以异常的形式抛出(不能完全抛出)  
　　　　由上面的讲解我们知道，php中的错误是不能以异常的像是捕获的，但是我们需要让他们抛出，已达到扩展 try-catch的影响范围，我们前面讲到过set\_error\_handler() 方法，他是干嘛用的，他是捕获错误的，所以我们就可以借助他来吧错误捕获，然后再以异常的形式抛出，ok，试试下面的写法：　　1：把错误以异常的形式抛出(不能完全抛出)  
　　　　由上面的讲解我们知道，php中的错误是不能以异常的像是捕获的，但是我们需要让他们抛出，已达到扩展 try-catch的影响范围，我们前面讲到过set\_error\_handler() 方法，他是干嘛用的，他是捕获错误的，所以我们就可以借助他来吧错误捕获，然后再以异常的形式抛出，ok，试试下面的写法：

1. 1 <?php
2. 2 set\_error\_handler('zyferror');
3. 3 function zyferror($type, $message, $file, $line)
4. 4 {
5. 5 throw new \Exception($message . 'zyf错误当做异常');
6. 6 }
7. 7
8. 8 $num = 0;
9. 9 try {
10. 10 echo 1/$num;
11. 11
12. 12 } catch (Exception $e){
13. 13 echo $e->getMessage();
14. 14 }
15. 15 ?>

　　好了，试一下，会打印出：

1 Division by zero zyf123

　　流程：本来是除0错误，然后触发set\_error\_handler()，在set\_error\_handler()中相当与杀了个回马枪，再把错误信息以异常的形式抛出来，这样就可以实现错误以异常的形式抛出。大家要注意：这样做是有缺点的，会受到set\_error\_handler()函数捕获级别的限制。  
  
     2：捕获所有的错误  
        由set\_error\_handler()可知，他能够捕获一部分错误，不能捕获系统级E\_ERROR、E\_PARSE等错误，但是这部分可以由register\_shutdown\_function()捕获。所以两者结合能出现很好的功能。  
        看下面的程序：

1. 1 a.php内容：
2. 2 <?
3. 3 // 模拟Fatal error错误
4. 4 //test();
5. 5
6. 6 // 模拟用户产生ERROR错误
7. 7 //trigger\_error('zyf-error', E\_USER\_ERROR);
8. 8
9. 9 // 模拟语法错误
10. 10 var\_dump(23+-+);
11. 11
12. 12 // 模拟Notice错误
13. 13 //echo $f;
14. 14
15. 15 // 模拟Warning错误
16. 16 //echo '123';
17. 17 //ob\_flush();
18. 18 //flush();
19. 19 //header("Content-type:text/html;charset=gb2312");
20. 20 ?>
21. 21 b.php内容：
22. 22 <?
23. 23 error\_reporting(0);
24. 24 register\_shutdown\_function('zyfshutdownfunc');
25. 25 function zyfshutdownfunc()
26. 26 {
27. 27 if ($error = error\_get\_last()) {
28. 28 var\_dump('<b>register\_shutdown\_function: Type:' . $error['type'] . ' Msg: ' . $error['message'] . ' in ' . $error['file'] . ' on line ' . $error['line'] . '</b>');
29. 29 }
30. 30 }
31. 31
32. 32 set\_error\_handler('zyferror');
33. 33 function zyferror($type, $message, $file, $line)
34. 34 {
35. 35 var\_dump('<b>set\_error\_handler: ' . $type . ':' . $message . ' in ' . $file . ' on ' . $line . ' line .</b><br />');
36. 36 }
37. 37
38. 38 require 'a.php';
39. 39 ?>

 　　到此就可以解释开头的那个程序了吧，test.php 如果是单文件执行是不能捕获到错误的，如果你在框架中执行就是可以的，当然你按照我上面介绍的来扩展也是可以的。

　　1：自定义异常处理

　　在复杂的系统中，我们往往需要自己捕获我们需要特殊处理的异常，这些异常可能是特殊情况下抛出的。所以我们就自己定义一个异常捕获类，该类必须是 exception 类的一个扩展，该类继承了 PHP 的 exception 类的所有属性，并且我们可以添加自定义的函数，使用的时候其实和之前的一样，大致写法如下：

1. 1 <?php
2. 2 class zyfException extends Exception
3. 3 {
4. 4 public function errorzyfMessage()
5. 5 {
6. 6 return 'Error line ' . $this->getLine().' in ' . $this->getFile()
7. 7 .': <b>' . $this->getMessage() . '</b> Must in (0 - 60)';
8. 8 }
9. 9 }
10. 10
11. 11 $age = 10;
12. 12 try {
13. 13 $age = intval($age);
14. 14 if($age > 60) {
15. 15 throw new zyfException($age);
16. 16 }
17. 17
18. 18 } catch (zyfException $e) {
19. 19 echo $e->errorzyfMessage();
20. 20
21. 21 }
22. 22 ?>

　　2：异常嵌套

　　异常嵌套是比较常见的写法，在自定义的异常处理中，try 块中可以定义多个异常捕获，然后分层传递异常，理解和冒泡差不多，看下面的实现：

1. 1 <?php
2. 2 $age = 10;
3. 3 try {
4. 4 $age = intval($age);
5. 5 if($age > 60) {
6. 6 throw new zyfException($age);
7. 7 }
8. 8
9. 9 if ($age <= 0) {
10. 10 throw new Exception($age . ' must > 0');
11. 11 }
12. 12
13. 13 } catch (zyfException $e) {
14. 14 echo $e->errorzyfMessage();
15. 15
16. 16 } catch(Exception $e) {
17. 17 echo $e->getMessage();
18. 18 }
19. 19 ?>

　　当然也可以在catch中再抛出异常给上层：

1. 1 <?php
2. 2 $age = 100;
3. 3 try {
4. 4 try {
5. 5 $age = intval($age);
6. 6 if($age > 60) {
7. 7 throw new Exception($age);
8. 8 }
9. 9
10. 10 } catch (Exception $e) {
11. 11 throw new zyfException($age);
12. 12
13. 13 }
14. 14
15. 15 } catch (zyfException $e) {
16. 16 echo $e->errorzyfMessage();
17. 17 }
18. 18 ?>

六、PHP7中的异常处理

　　现在写PHP必须考虑版本情况，上面的写法在PHP7中大部分都能实现，但是也会有不同点，在PHP7更新中有一条：更多的Error变为可捕获的Exception，现在的PHP7实现了一个全局的throwable接口，原来老的Exception和其中一部分Error实现了这个接口(interface)，PHP7中更多的Error变为可捕获的Exception返回给捕捉器，这样其实和前面提到的扩展try-catch影响范围一样，但是如果不捕获则还是按照Error对待，看下面两个：

1. 1 <?php
2. 2 try {
3. 3 test();
4. 4
5. 5 } catch(Throwable $e) {
6. 6 echo $e->getMessage() . ' zyf';
7. 7 }
8. 8
9. 9 try {
10. 10 test();
11. 11
12. 12 } catch(Error $e) {
13. 13 echo $e->getMessage() . ' zyf';
14. 14 }
15. 15 ?>

 因为PHP7实现了throwable接口，那么就可以使用第一个这种方式来捕获异常。又因为部分Error实现了接口，并且更多的Error变为可捕获的Exception，那么就可以使用第二种方式来捕获异常。下面是在网上找的PHP7的异常层次树：  
Throwable  
　　Exception 异常  
　　　　...  
　　Error 错误  
　　　　ArithmeticError 算数错误  
　　　　　　DivisionByZeroError 除数为0的错误  
　　　　AssertionError 声明错误  
　　　　ParseError 解析错误  
　　　　TypeError 类型错误