**PHP的错误处理与异常处理**

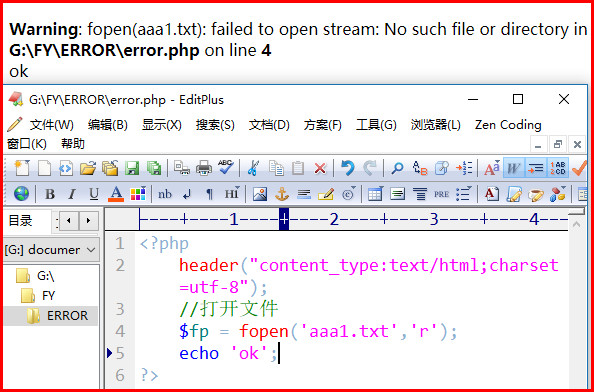
**第一章 错误处理**

**1.基本介绍**

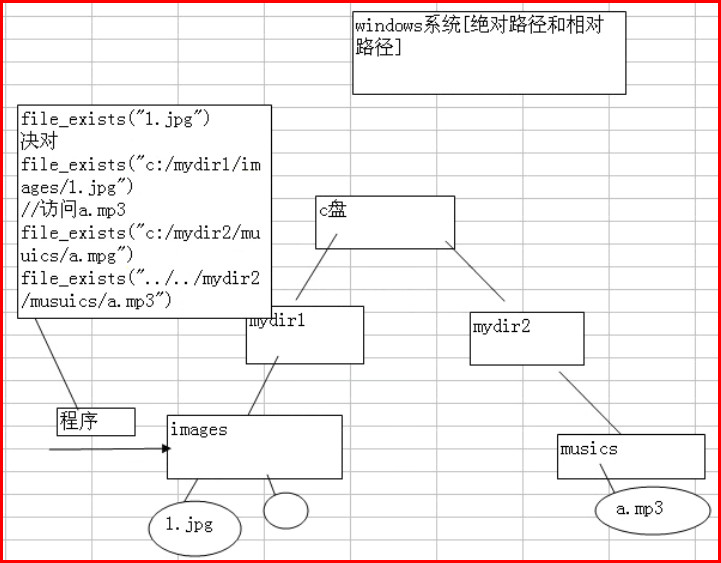
问题导出:没有错误处理机制会怎样?

简单演示:

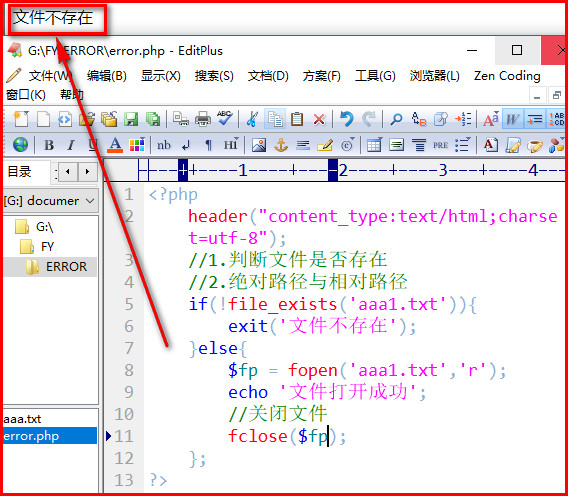
下面的程序没有对可能出现的错误进行处理



引入介绍绝对路径与相对路径



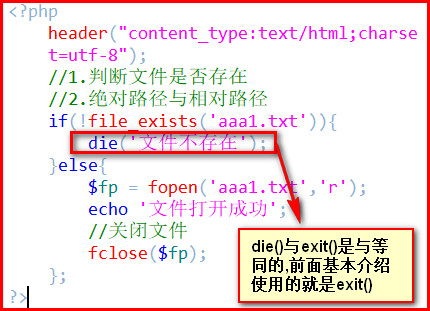
下面的代码引入了简单的错误处理方式,代码更有健壮性



**PHP一般的错误处理有三种方案:**

* **简单的”die()或exit()”语句**
* **自定义错误与错误触发器**
* **错误日志**

**2.简单错误处理”die()或exit()”**

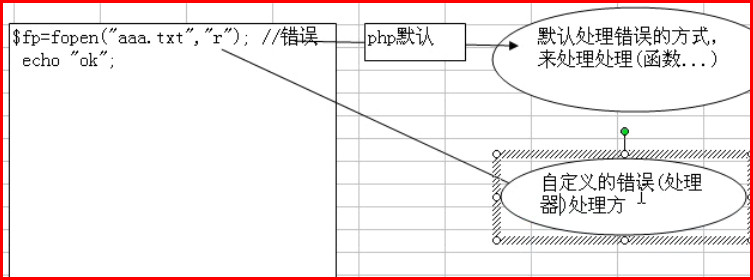




**总结:die()或exit()就是一个简单的错误处理机制,在发生错误后就终止了脚本,仅仅采用这样的方式是无法满足我们的需求的,下面介绍更好一点的处理错误的方式-------创建自定义错误处理器**

**3.自定义错误函数(处理器)**

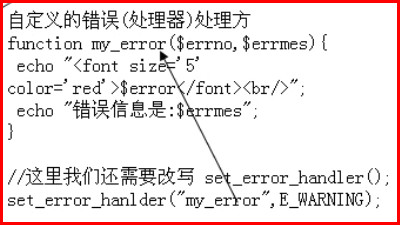
画图说明自定义错误函数(处理器):





创建自定义错误函数(处理器)基本介绍:

表示有我们自己简单创建一个专用函数来处理PHP中发生的错误,该函数必须有能力处理至少两个参数(error level与error message),最多五个参数,其它三个参数可选(file, line-number, error context)



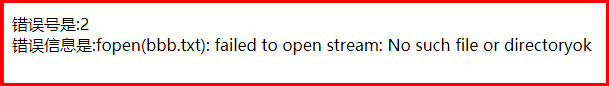
基本语法:

**error\_function\_name(error\_level,error\_message,[ error\_file, error\_line, error\_context])**

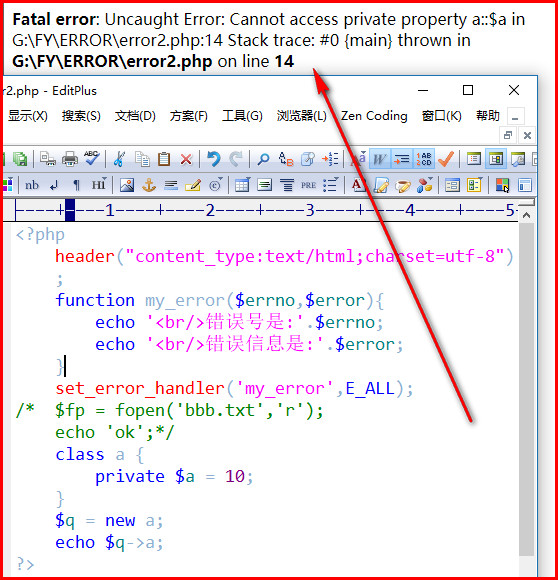
**set\_error\_handler(“error\_function\_name”,错误级别);**

(下面以E\_WARNING进行举例):





在PHP7中貌似E\_ALL没啥用



**4.错误触发器**

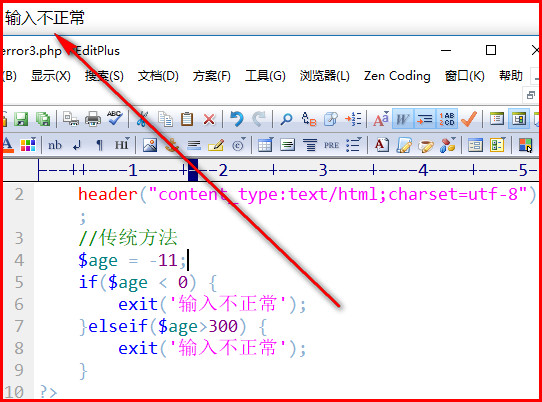
1.基本概念:

**前面的错误是程序本身的错误(语法错误或打开不存在文件),不是逻辑性的错误**,如果对逻辑性的错误进行处理时,就需要用到错误触发器--------trigger\_error()来完成.

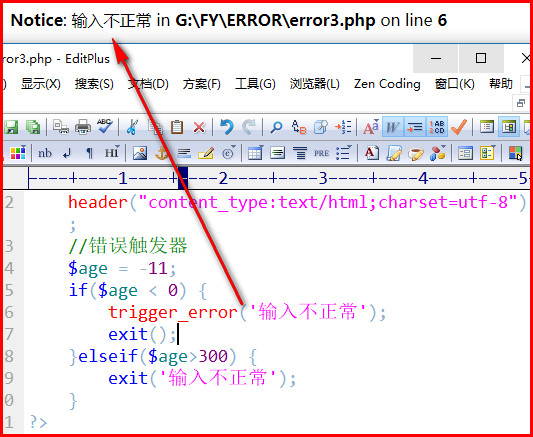
2.案例演示

* 有一段代码,输入年龄,如果输入的年龄不符合逻辑就认为是一个错误(比如年龄为负数,或年龄过大)

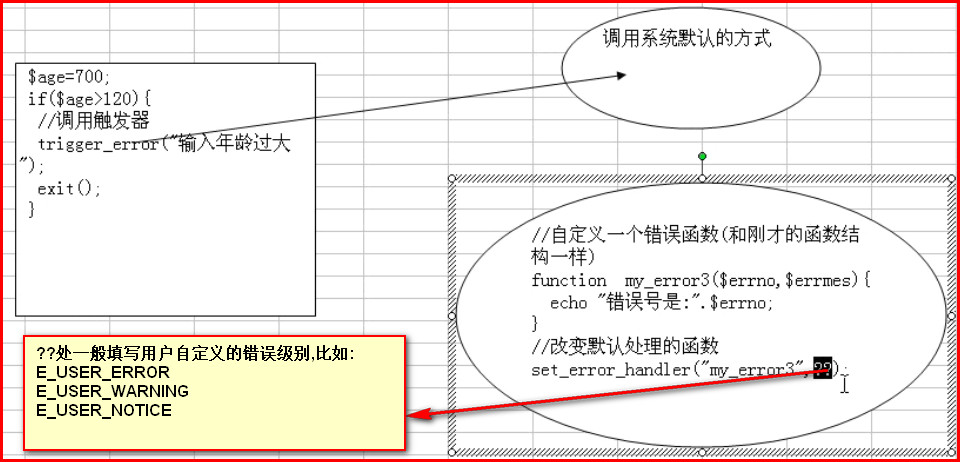
传统的方法:

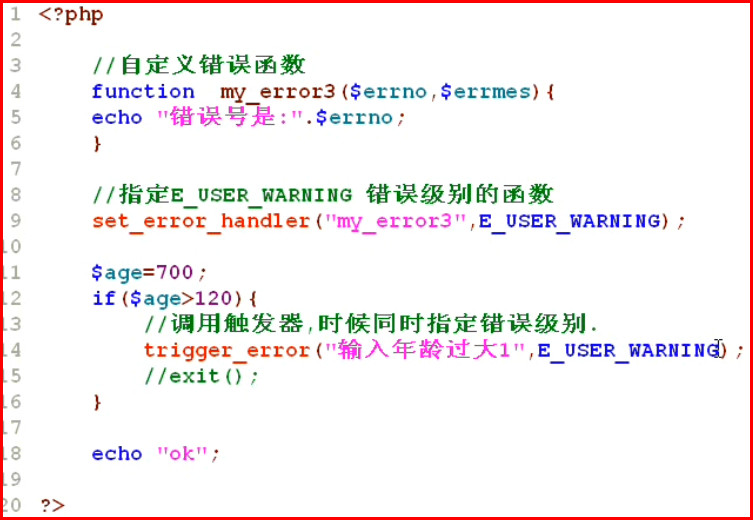


错误触发器:

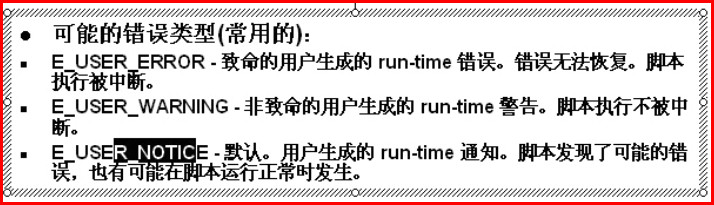


配合自定义错误处理函数就能满足一般的逻辑错误判断需求了





说明:在用**set\_error\_handler**(‘自定义错误函数名’,错误级别)设置过后,**trigger\_error**(‘错误说明语句’,用户自定义错误级别);中错误级别应该与**set\_error\_handler()**中错误级别一致!!!!!



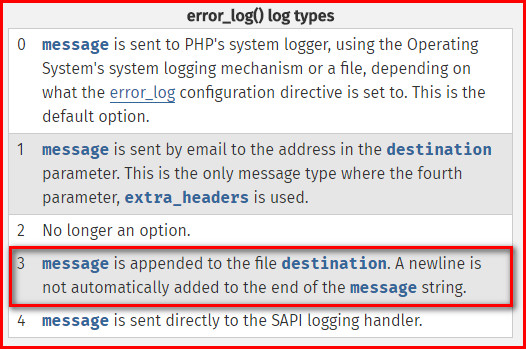
**5.PHP错误日志**

1.基本介绍

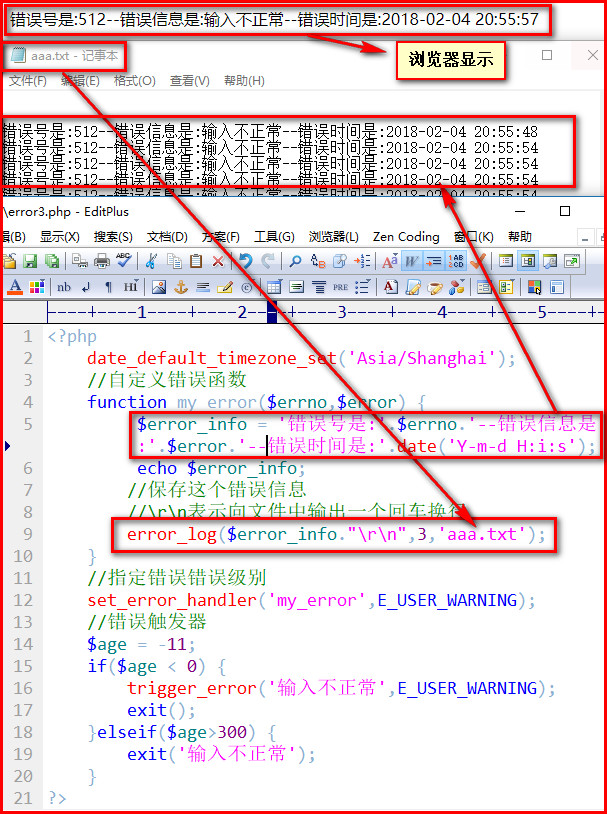
默认地,根据在**php.ini中的error\_log配置,PHP向服务器的错误记录系统或文件发送发送错误记录**.**通过使用error\_log()函数也可以指定向文件或远程目的地发送错误记录**

函数error\_log函数

**bool error\_log ( string $message [, int $message\_type = 0 [, string $destination [, string $extra\_headers ]]] )**



一般error\_log函数的**$message\_type**参数是采用3



**第二章 异常处理**

**1.基本介绍**

异常处理用于指定的错误(异常)情况发生时改变脚本的正常流程

异常处理的基本语法是:

try{

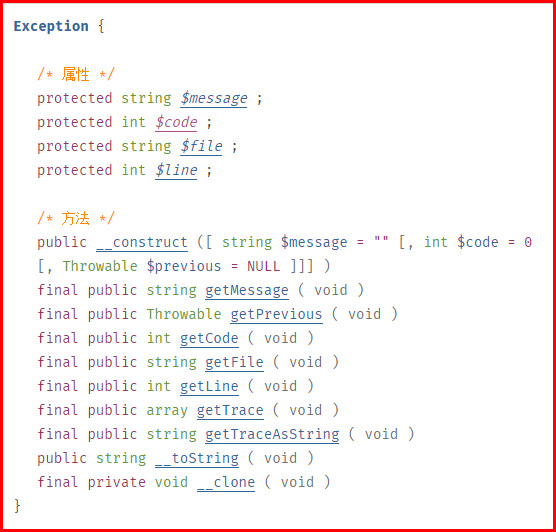
//可能出现的错误或异常的代码

}catch(Exception $e){

//捕获异常并处理,通常是抛出一个异常

//处理方式:1.自己处理 2.不处理,抛出throw $e;

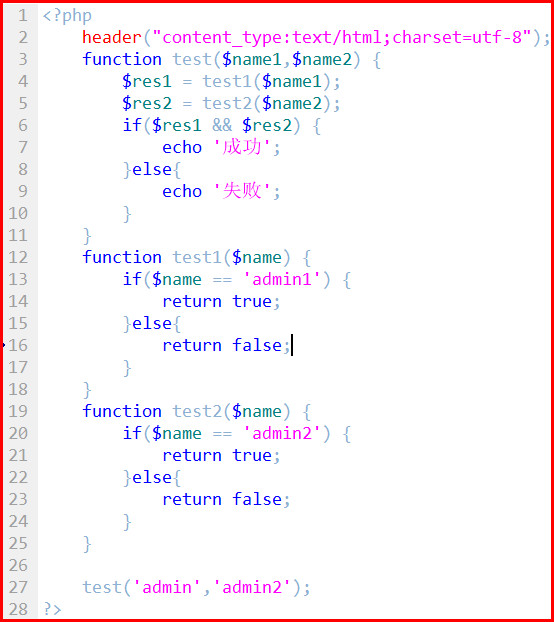
}



**2.案例说明**

问题引出:调用两个函数,必须两个函数同时成功才能满足需求,如何判别两个函数是否成功(类似MySQL的事务)

传统方法:-->效率差,扩展性差,



如何有效的控制多条可能出现错误或异常的代码

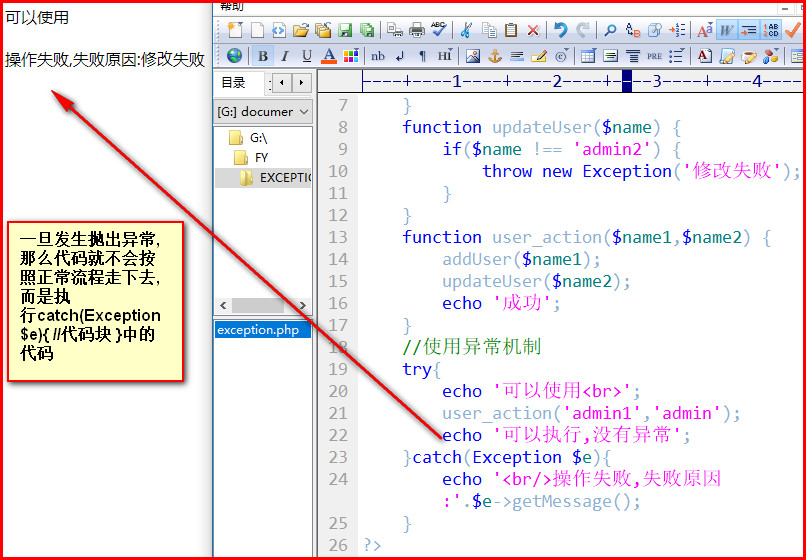
异常处理方法



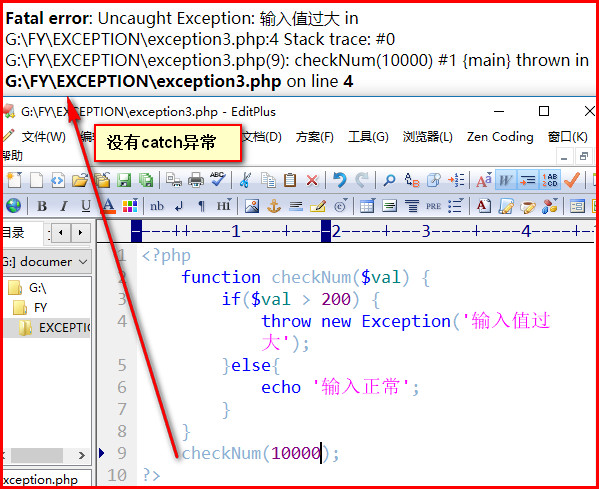
**3.异常处理的注意事项**

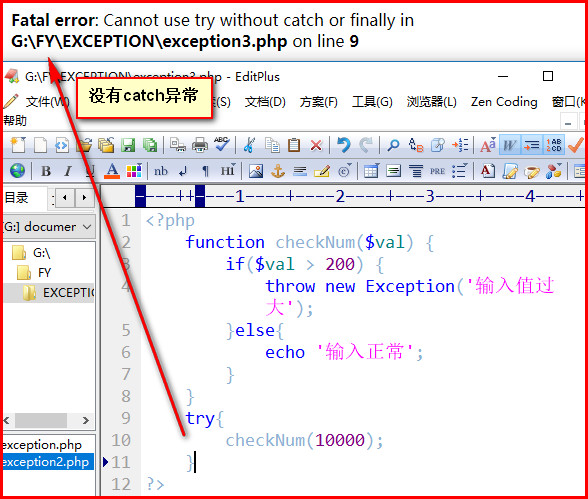
上面的案例可以看出使用异常机制可以更有效的控制错误,在开发中使用较多.

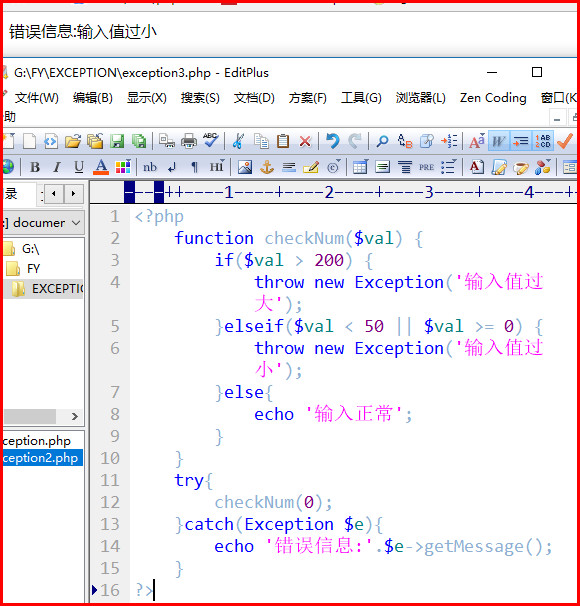
* 1.当异常被抛出后,其后的代码不会继续执行,PHP会尝试查找匹配的catch{ }代码块.



* 2.当捕获一个异常时,如果没有catch,则会报Fatal Error

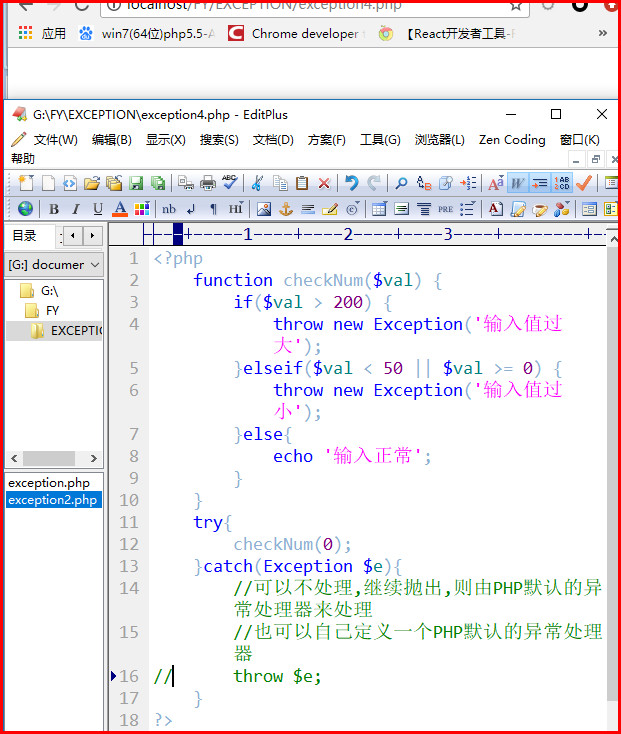


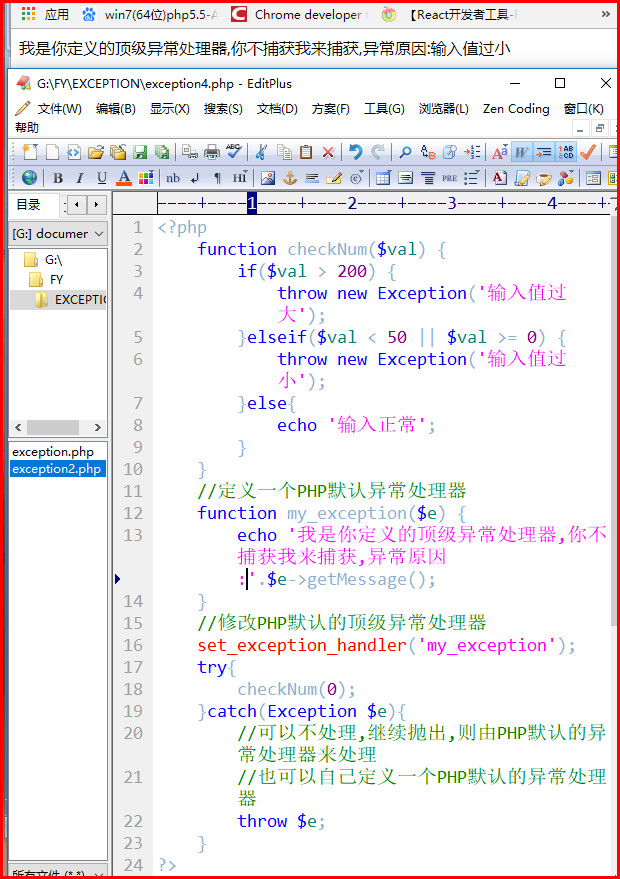




只有满足上方的catch才能正常使用异常处理

* 3.当catch一个异常后可以处理也可以不处理,不处理就可以抛出,语法是:throw new Exception(‘信息’);





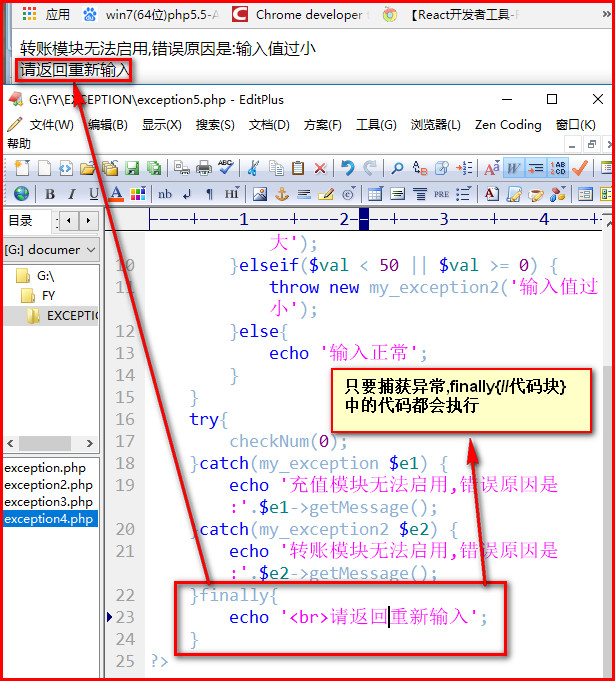
* 4.可以自己定义异常类(Exception说到底就是一个类,用OOP那一套都可以)

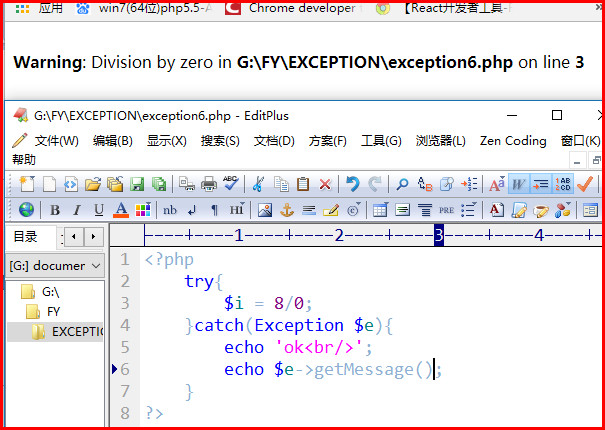
class my\_exception extends Exception {

}

**PHP异常处理--小结**

* **需要进行异常处理的代码应该放入try代码块内,以便于捕获潜在异常**
* **每个try或throw代码必须至少拥有一个对应的catch代码块**
* **使用等多个catch代码块可以捕获不同种类的异常**
* **可以在try代码块内的catch代码块中再次抛出异常**
* **如果抛出异常就必须捕获或者使用最顶级异常处理器**





**只有throw(抛出)才能catch,不然都不会catch!!!!!!!!!**