PHP 实践之路 - 面向对象篇课程

主讲：孙胜利

新浪微博：@私房库

微博主页：http://weibo.com/sifangku

第二章 异常与错误处理

一、异常处理

二、错误处理

本章导语：

无论是异常还是错误都反应了我们的代码存在一定的问题，我们这一章就是和大家一起学习如何妥善的处理好这些问题！

一、异常处理

异常是指我们的程序表面是没有问题的，但是在运行过程中还是出现了问题。我们应该对程序中可能出现的异常**手动**做出提前预判和预备解决措施，通过 PHP 的异常处理机制我们可以将程序的主体逻辑与解决各种异常的代码进行分离！

1、PHP 为我们提供了现成的异常处理类 Exception，这个类怎么用？

这个类需要配合一些特殊的语句结构，我们一起来学习一下！

```
try{
    当执行可能出现异常的代码请放在这里面执行！
    注：在可能出现异常的地方需要使用 throw new Exception($error);将异常对象抛出！
    如果抛出异常那么这边之后的代码都不会执行，从而跳到 catch 那边执行。
    如果没有抛出异常那么 catch 中的代码永远不会执行！
}catch(Exception $e){//抛出的异常会在这边被捕获到！为什么前面加上 Exception？
    $e 这个对象里面有很多成员供我们使用！
    getCode()           返回接收到的异常代号
    getMessage()        返回接收到的异常信息
    getPrevious()       返回异常链中的前一个异常
    getFile()           返回发生异常的文件名
    getLine()           返回发生异常的代码行数
    getTrace()          获取异常追踪信息（保存了文件名、行号等数据的数组）
    getTraceAsString()  获取字符串类型的异常追踪信息
    __toString()        将异常对象转换为字符串
}
程序继续向下执行...
```

注意：

1>如果一个异常没有被捕获，PHP 会产生一个致命错误并且报出“未捕获的异常...”信息，除非设

置一个用户定义的异常处理函数

2>当一个异常被抛出，try 里面之后的语句代码将不会继续执行，PHP 会尝试查找能与之匹配的 catch。

3>抛出的异常会一层层的传到最初执行的代码那边（如果在中间没有被捕获的话）。

4>每个 try 都必须至少有一个相应的 catch 或 finally 块。

5>当 Exception 类不够用的时候，我们完全可以扩展它（继承）

6>抛出对象必须是异常类(Exception)的一个实例或异常的子类的实例

2、捕获其他类型的异常

假如我在可能出问题的代码那边抛出了其他类型的异常对象（比如 Exception 的某个派生类的实例）那么应该如果捕获到这个异常呢？

```
try{

}catch(MyException $e){


}catch(Exception $e){//Exception 放在最后面，处理上面都处理不了的异常


}如果还有其他类型的异常对象请继续...
```

即：当 try 里面的代码抛出异常对象时，会根据对象的类型跳转到对应的 catch 那边执行！

3、用户定义的异常处理函数

```
set_exception_handler (callable $exception_handler)
```

二、错误处理

错误主要由那些不符合 PHP 语法或者访问了一些不可访问的内容造成的，错误都是**自动**产生的（确切的说是不得不产生的），它代表我们的程序有很大的问题，这些问题可能造成我们的程序强制终止所以要认真对待！

1、错误级别

级别常量	描述	值
E_ERROR	致命的运行时错误。这类错误一般是不可恢复的情况，例如内存分配导致的问题。后果是导致脚本终止不再继续运行。	1
E_WARNING	运行时警告 (非致命错误)。仅给出提示信息，但是脚本不会终止运行。	2
E_PARSE	编译时语法解析错误。	4
E_NOTICE	运行时注意通知，表示脚本遇到可能会表现为错误的情况，但是在可以正常运行的脚本里面也可能会有类似的通知。(可能是也有可能不是问题)	8
E_CORE_ERROR	在 PHP 初始化启动过程中发生的致命错误。该错误类似 E_ERROR，但是是由 PHP 引擎核心产生的。	16
E_CORE_WARNING	PHP 初始化启动过程中发生的警告 (非致命错误) 。类似 E_WARNING，但是是由 PHP 引擎核心产生的。	32
E_COMPILE_ERROR	致命编译时错误。类似 E_ERROR, 但是是由 Zend 脚本引擎产生的。	64
E_COMPILE_WARNING	编译时警告 (非致命错误)。类似 E_WARNING，但是是由 Zend 脚本引擎产生的。	128
E_USER_ERROR	用户产生的错误信息。类似 E_ERROR, 但是是由用户自己在代码中使用 PHP 函数 trigger_error()来产生的。	256
E_USER_WARNING	用户产生的警告信息。类似 E_WARNING, 但是是由用户自己在代码中使用 PHP 函数 trigger_error()来产生的。	512
E_USER_NOTICE	用户产生的通知信息。类似 E_NOTICE, 但是是由用户自己在代码中使用 PHP 函数 trigger_error()来产生的。	1024
E_STRICT	启用 PHP 对代码的修改建议，以确保代码具有最佳的互操作性和向前兼容性。	2048
E_RECOVERABLE_ERROR	可被捕捉的致命错误。 它表示发生了一个可能非常危险的错误，但是还没有导致 PHP 引擎处于不稳定的状态。 如果该错误没有被用户自定义句柄捕获 (参见 set_error_handler())，将成为一个 E_ERROR 从而脚本会终止运行。	4096
E_DEPRECATED	运行时通知。启用后将会对在未来版本中可能无法正常工作的代码给出警告。	8192
E_USER_DEPRECATED	用户产少的警告信息。 类似 E_DEPRECATED, 但是是由用户自己在代码中使用 PHP 函数 trigger_error()来产生的。	16384
E_ALL	E_STRICT 除外的所有错误和警告信息。 PHP 5.4.x：32767；PHP 5.3.x：30719；PHP5.2.x：6143；更早：2047	32767 不同的 PHP 版本可能不同

2、设置错误报告级别

error_reporting(参数)

参数：

错误级别（数值或者符号）用于建立一个二进制位掩码，来制定要报告的错误信息。

可以使用按位运算符（&、|、~）来**组合**这些值或者**屏蔽**某些类型的错误。

常用：

```
//开发阶段
ini_set('display_errors','On');
error_reporting(E_ALL);
//上线阶段
ini_set('display_errors','Off');
error_reporting(E_ALL);//可以适当屏蔽一些程度较轻的错误
```

最重要的是 在 PHP 配置文件里设置上面两个选项以及错误日志的相关选项：

```
display_errors = Off
error_reporting = E_ALL
log_errors = On
log_errors_max_len = 1024
error_log = "位置"
```

注意：

在 PHP 脚本里面设置了不报错不代表就真的不会报错，因为脚本里面的这句话生效的前提是 PHP 脚本能够运行起来，而如果错误出现在编译阶段（PHP 其实从 PHP4 开始就不是直接解释执行了，而是由 Zend 引擎**自动编译**后才执行的这也是 PHP 执行速度如此高效的原因，PHP7 中 Zend 引擎再次优化升级，PHP 的执行速度会更快！）会按照配置文件里的相关配置来报错！

PHP 配置文件里面有错误报告的相关配置和错误日志的相关配置，这些我们放在学完 Linux 之后再说。

3、接管 PHP 错误处理机制（设置用户自定义的错误处理函数）

set_error_handler (callable $error_handler[,int $error_types = E_ALL | E_STRICT])

参数：

1）error_handler

该回调函数接受的参数

errno

第一个参数 errno，包含了错误的级别，是一个 integer。

errstr

第二个参数 errstr，包含了错误的信息，是一个 string。

errfile

第三个参数是可选的，errfile， 包含了发生错误的文件名，是一个 string。

errline

第四个参数是一个可选项， errline， 包含了错误发生的行号，是一个 integer。

errcontext

第五个可选参数， errcontext 会包含错误触发处作用域内所有变量的数组。

注意：

1>如果有必要该函数内部应该使用 exit()函数终止程序的执行

2>如果该回调函数函数返回 FALSE，标准错误处理处理程序将会继续调用

3>以下级别的错误不能由用户定义的函数来处理： E_ERROR、 E_PARSE、 E_CORE_ERROR、 E_CORE_WARNING、 E_COMPILE_ERROR、 E_COMPILE_WARNING

2）error_types

规定哪些级别的错误会执行该函数

4、手动产生一个用户级别错误

trigger_error (string $error_msg [, int $error_type = E_USER_NOTICE])

产生一个用户级别的 error/warning/notice 信息

该函数在你运行出现异常时，需要产生一个特定的响应时非常有用。

参数：

1）error_msg

该 error 的特定错误信息，长度限制在了 1024 个字符。超过 1024 长度的字符都会被截断。

2）error_type

该 error 所特定的错误类型。仅 E_USER 系列常量对其有效，默认是 E_USER_NOTICE。

5、PHP7 中的错误处理

PHP7 与之前的 PHP 版本有很大的不同，PHP7 中的很多错误也会抛出异常（Error 异常），Error 异常是自动抛出的，而我们 PHP 里面常规讲的异常是需要手动抛出的！

1）PHP7 中与之前的 PHP 版本有很大的不同，PHP7 中的很多错误是作为异常的一种（Error 异常），不过 Error 异常是自动抛出的，而我们常规讲的异常是需要手动抛出的！

2）这种 Error 异常可以像普通异常一样被 try / catch 块所捕获。如果没有匹配的 try / catch 块，则调用异常处理函数（由 set_exception_handler() 注册）进行处理。 如果尚未注册异常处理函数，则按照传统方式处理：被报告为一个致命错误（Fatal Error）

3）Error 类并不是从 Exception 类 扩展出来的，所以用 catch (Exception $e) { ... } 这样的代码是捕获不 到 Error 的。你可以用 catch (Error $e) { ... } 这样的代码，或者通过注册异常处理函数（ set_exception_handler()）来捕获 Error