一、异常处理

1、生活与程序

生活中的应急议案的制定，应对突发情况，尽可能减少损失。

程序中出现非语法的错误能让程序继续执行，及时处理异常。

2、定义异常处理

异常（Exception）处理用于在指定的错误发生时改变脚本的正常流程。是 PHP5 中的一个新的重要特性。异常处理是一种可扩展、易维护的错误处理统一机制，并提供了一种新的面向对象的错误处理方式。

3、异常处理格式：

try{

使用 try 去包含可能会发生异常的代码

一旦出现异常 try 进行捕获异常，交给 catch 处理。

抛出异常语句：throw 异常对象。

}catch（异常对象参数）{

在这里做异常处理。

}[catch（。，，）{

.. .. ..

}]

4、类名不区分大小

5、深析异常处理

```php
<?php
    header("Content-type:text/html;charset=utf-8");
    //异常处理的使用
    /*
    //第一种方法处理错误，终止程序
    function demo($a,$b){
        //处理错误信息
        if($b==0){
            //输出提示信息
            die("除数不能为0");
        }
        return $a/$b;
    }

    //调用
    echo demo(100,0);
    */
    /*
    //第二种方法处理错误，不终止程序脚本,让后续的代码继续执行
    function demo1($a,$b){
        //处理错误信息
        if($b==0){
            //输出提示信息,同时让后面的代码继续执行
            return("除数不能为0");
        }
        return $a/$b;
    }

    //调用
    echo demo1(100,0);

    echo '<br>后续的代码能继续执行';
```

```php
//第三种方式处理错误
function demo1($a,$b){
    //处理错误信息,语句执行此处有问题
    if($b==0){
        //立即抛出异常处理,程序继续往下执行时会出错的,自定义异常对象(人为)
        $error = '除数不能为0<br>';

        //当函数调用执行时指定了$b为0,人为定义抛出异常的条件,只要满足立即抛出
        throw new Exception($error,110);
    }
    return $a/$b;
}

echo '<br>start.........<br>';

//异常处理
try{ //捕获异常
    echo '11111111<br>';
    //调用时代码在执行时发现异常的现象立即捕获
    echo demo1(100,0);
    /*try会看着这个函数的调用的执行的代码过程,有错误就
    直接捕获送到catch部分处理,没有错误就忽略*/

    echo '22222222<br>';//捕获后的语句不执行

}catch(Exception $e){//处理异常

    echo '33333333<br>';

    /*异常抛出的实例化的对象传给参数$e恰好与系统的异常对象(类型约束)
    一致,接下交给系统内置异常对象处理(获取信息,行号...)*/
    echo '错误信息: '.$e -> getMessage().'<br>';
    echo '错误号码: '.$e -> getCode().'<br>';
    echo '错误文件: '.$e -> getFile().'<br>';
    echo '错误行号: '.$e -> getLine().'<br>';

}

echo '<br>end......';
/*意义: 捕获异常后处理异常,能程序继续运行不影
响整体的程序,错误任自身把控并且错误形成日志保存*/
```

结果：

```
start.........
11111111
33333333
错误信息：除数不能为0

错误号码：110
错误文件：D:\wamp\www\OOP_94\oop6\demo\3.php
错误行号：46

end......
```

6、异常处理实例：

<?php

//异常处理实例:思路先按照正常定义类再指定异常抛出条件

```
class Stu{
	private $name;
	private $age=20;

	public function __set($name,$value){
		//判断是否是给年龄赋值,自定义异常条件
		if($name=="age" && $value<0){
			throw new Exception("年龄不可以小于 0！");
		}
		$this->$name = $value;
	}

	public function __get($name){
		return $this->$name;
	}
}

$s = new Stu();
//怀疑设置参数有异常就直接使用 try
try{//捕获异常
	$s->name="张三";
	$s->age=-30;
}catch(Exception $e){ //异常处理
	echo "错误原因：".$e->getMessage();
}
echo $s->name.":".$s->age;
```

结果：

错误原因：年龄不可以小于 0！张三:20

7、自定义异常

```
<?php
header("Content-type:text/html;charset=utf-8");
// 自定义异常

//自定义异常类，必须直接或间接继承 Exception(核心)
class MyException extends Exception{
	public function getinfo(){
		//Exception 基类受保护属性子类能获取到
		return "自己定义的异常类，错误原因：".$this->message;
	}
}

class Stu{
	private $name;
```

```
    private $age=20;

    public function __set($name,$value){
        //判断是否是给年龄赋值,定义抛出异常处理条件
        if($name=="age" && $value<0){
            throw new MyException("年龄不可以小于 0！");//message 信息
        }
        $this->$name = $value;
    }

    public function __get($name){
        return $this->$name;
    }

    //除法函数,基类
    public function demo($a,$b){
        if($b==0){
            //当除数为 0 时，自定义的抛出一个异常错误对象。
            throw new Exception("除数不可以为零！",110);
        }
        return $a/$b;
    }
}

$s = new Stu();
//多层异常处理
try{ //捕获异常，根据异常对象去选择下面对应的处理
    $s->age=-10;
    echo $s->demo(4,0);
}catch(MyException $me){
    //有错误要进行反应
    echo $me->getinfo();
}catch(Exception $e){//永远基类放在最后面
    echo $e->getMessage();
}
```

结果：

自己定义的异常类，错误原因：年龄不可以小于 0！

==========

Classes/Object 函数

==========

8、参考手册中--与变量和类型有关的扩展

1. class_alias — 为类创建一个别名

格式：bool class_alias ([string $original [, string $alias]])

示例：

```
class foo { }
class_alias('foo', 'bar');

$a = new foo;
$b = new bar;
// the objects are the same
var_dump($a == $b, $a === $b);  //bool(true)
var_dump($a instanceof $b);     //bool(false)

// the classes are the same
var_dump($a instanceof foo);    //bool(true)
var_dump($a instanceof bar);    //bool(true)

var_dump($b instanceof foo);    //bool(true)
var_dump($b instanceof bar);    //bool(true)
```

*2. class_exists — 检查类是否已定义

格式： bool class_exists (string $class_name [, bool $autoload])

--如果由 class_name 所指的类已经定义，此函数返回 TRUE，否则返回 FALSE。

默认将会尝试调用 __autoload，如果不想让 class_exists() 调用 __autoload，

可以将 autoload 参数设为 FALSE。

```
<?php
 // 使用前检查类是否存在
 if ( class_exists ( 'MyClass' )) {
     $myclass  = new  MyClass ();
}

?>
```

3. get_called_class — the "Late Static Binding" class name

(PHP 5 >= 5.3.0) 获取调用者的类名

*4. get_class_methods — 返回由类的方法名组成的数组

格式：array get_class_methods (mixed $class_name)

返回由 class_name 指定的类中定义的方法名所组成的数组。如果出错，则返回 NULL。

从 PHP 4.0.6 开始，可以指定对象本身来代替 class_name

5. get_class_vars — 返回由类的默认公有属性组成的数组

格式： array get_class_vars (string $class_name)

返回由类的默认公有属性组成的关联数组，此数组的元素以 varname => value 的形式存在。

*6. get_class — 返回对象的类名

格式： string get_class ([object $obj])

返回对象实例 obj 所属类的名字。如果 obj 不是一个对象则返回 FALSE。

7. get_declared_classes — 返回由已定义类的名字所组成的数组

格式：array get_declared_classes (void)

返回由当前脚本中已定义类的名字组成的数组。

8. get_declared_interfaces — 返回一个数组包含所有已声明的接口

格式：array get_declared_interfaces (void)

本函数返回一个数组，其内容是当前脚本中所有已声明的接口的名字。

9. get_object_vars — 返回由对象属性组成的关联数组

格式：array get_object_vars (object $obj)

返回由 obj 指定的对象中定义的属性组成的关联数组。

10. get_parent_class — 返回对象或类的父类名

格式：string get_parent_class ([mixed $obj])

如果 obj 是对象，则返回对象实例 obj 所属类的父类名。

11. interface_exists — 检查接口是否已被定义

格式：bool interface_exists (string $interface_name [, bool

$autoload])

本函数在由 interface_name 给出的接口已定义时返回 TRUE，否则返回 FALSE。

*12. is_a — 如果对象属于该类或该类是此对象的父类则返回 TRUE

我们可以使用运算符： instanceof 代替上面的 is_a 操作

格式：bool is_a (object $object , string $class_name)

如果对象是该类或该类是此对象的父类则返回 TRUE，否则返回 FALSE。

13. is_subclass_of — 如果此对象是该类的子类，则返回 TRUE

格式：bool is_subclass_of (object $object , string $class_name)

如果对象 object 所属类是类 class_name 的子类，则返回 TRUE，否则返回 FALSE。

*14. method_exists — 检查类的方法是否存在

格式：bool method_exists (object $object , string $method_name)

如果 method_name 所指的方法在 object 所指的对象类中已定义，则返回 TRUE，否则返回 FALSE。

*15. property_exists — 检查对象或类是否具有该属性

格式：bool property_exists (mixed $class , string $property)

本函数检查给出的 property 是否存在于指定的类中（以及是否能在当前范围内访问）。

测试代码：

```
<?php
//类和对象的相关函数
class A{
    public $x=10;
    protected $y=20;
    private $z=30;

    public function fun1(){}
    protected function fun2(){}
    private function fun3(){}
}
```

```php
//class_exists — 检查类是否已定义
echo "<h3>1. class_exists — 检查类是否已定义</h3>";
if(class_exists("A")){
    echo "存在！";
}else{
    echo "不存在！";
}

echo "<hr/>";

//get_class_methods — 返回由类的方法名组成的数组
echo "<h3>2. get_class_methods — 返回由类的方法名组成的数组</h3>";
$res = get_class_methods("A");
print_r($res); //只能获取公有的方法


echo "<hr/>";
//get_class_vars — 返回由类的默认公有属性组成的数组
echo "<h3>3. get_class_vars — 返回由类的默认公有属性组成的数组</h3>";
$res = get_class_vars("A");
print_r($res); //只能获取公有的属性


echo "<hr/>";
//get_class — 返回对象的类名
echo "<h3>4. get_class — 返回对象的类名</h3>";
$e = new A();
$res = get_class($e);
print_r($res); //获取对象的类名


echo "<hr/>";
//get_parent_class — 返回对象或类的父类名
echo "<h3>5. get_parent_class — 返回对象或类的父类名</h3>";
class B extends A{

}
class C extends B{

}
$b = new C();
$res = get_parent_class($b);
print_r($res); //获取对象的类父类名
```

结果：

1. class_exists — 检查类是否已定义

存在!

---

2. get_class_methods — 返回由类的方法名组成的数组

```
Array ( [0] => fun1 )
```

---

3. get_class_vars — 返回由类的默认公有属性组成的数组

```
Array ( [x] => 10 )
```

---

4. get_class — 返回对象的类名

```
A
```

---

5. get_parent_class — 返回对象或类的父类名

```
B
```

9、框架常用函数

```
<?php
//method_exists — 检查对象的方法是否存在
class A{
	public function fun(){
		echo "class A....<br/>";
		//判断自己是否拥有 test 方法，若有则调用(对子类的预留)
		if(method_exists($this,"test")){
			$this->test();
		}
	}
}
class B extends A{
	public function test(){
		echo "class B....<br/>";
	}
}
$a =new B();
$a->fun();
```

结果：

```
class A....
class B....
```