

面向对象编程——单元 3：静态成员与类常量

前言：

static 放在 PHP 普通函数内部时，在函数执行结束后，static 变量的值不丢失。



```
1 <?php
2 header("Content-type:text/html;charset=utf8");
3
4 function fun(){
5     //静态局部变量
6     static $var = 0;
7     $var++;
8     echo "函数第{$var}次被调用<br>";
9 }
10 fun();
11 fun();
12 ?>
```

localhost:81/2018/oop/3/

函数第1次被调用
函数第2次被调用

一、类的静态成员

类中定义的普通的成员变量和方法是属于对象的属性和方法。必须先实例化产生对象，再调用普通的属性和方法。

还有一些特殊的成员是**属于类**的，不需要实例化就可以使用。这些成员称为**静态成员**。语法上**使用类名进行调用**。在声明静态成员时，使用 **static** 关键字修饰。

访问时使用::范围解析操作符来引用静态成员。

```
1 <?php
2 class A
3 {
4     private $name='张三';
5     public static $age=10; //静态属性
6     public function getName(){
7         return $this->name;
8     }
9     public static function getAge(){ //静态方法
10        //echo $this->name; //静态方法不能访问非静态成员
11        echo A::$age; //方式1: 类名::
12        echo self::$age; //方式2(推荐): self::
13        echo static::$age; //方式3(推荐): static::
14    }
15 }
16
17 echo A::$age; //在类外部访问公有的静态属性。类名::访问静态属性age,注意有$符号
18 echo A::getAge();
19
20 ?>
```

在类内部引用静态成员推荐使用 `self::`和 `static::`。


两者的区别在于继承时:

`self::`访问的永远是其定义所在的类的静态成员。

`static::`访问的是调用的类的静态成员。

经常使用的是 `static::`,因为它更符合常规逻辑。

```
1 <?php
2 class A
3 {
4     public static $age=10; //静态属性
5     public static function getAgeSelf(){
6         return self::$age;
7     }
8     public static function getAgeStatic(){
9
10        return static::$age;
11    }
12 }
13
14 class B extends A{
15     public static $age=20; //重写静态属性
16 }
17 echo A::getAgeSelf();
18 echo A::getAgeStatic();
19 echo "<br>";
20 echo B::getAgeSelf();
21 echo B::getAgeStatic();
22
23 ?>
```



localhost:81/objectD
← → ↻ ① localh
1010
1020

二、为什么要使用静态成员？

静态成员用于在各个对象上共享数据和操作。

例如：每当创建一个学生对象（Student 类）时，学生的总数量增加 1，每当销毁一个学生对象时，减少学生的总数量。

```
1 <?php
2 class Student
3 {
4     private static $stu_Num=0;
5     function __construct() {
6         static::$stu_Num++;
7     }
8     function destruct(){
9         static::$stu_Num--;
10    }
11    public static function getStuNum(){
12        echo "当前学生数量是".static::$stu_Num."<br/>";
13    }
14 }
15 $stu1 = new Student();
16 $stu2 = new Student();
17 $stu3 = new Student();
18 Student::getStuNum();
19 $stu1 = null;
20 $stu2 = null;
21 Student::getStuNum();
22 ?>
```

类内使用static::引用静态变量

产生对象时自增，销毁对象时自减

类外使用类名引用静态成员

引用变量赋值为null时，对象变成垃圾被销毁

localhost:81/objectDemo/

当前学生数量是3
当前学生数量是1

三、静态方法和非静态方法的调用区别。

静态方法属于类，应该直接使用类名在类的外部调用；非静态方法属于对象，所以应该使用对象调用非静态方法。推荐的写法如下：



四、::范围解析操作符

范围解析操作符 (::) 是一对冒号，可以用于访问静态成员和常量，以及被覆盖的父类中的属性和方法。

当在类的外部使用 :: 符号访问这些静态属性、方法和常量时，必须使用类的名字。

关键字	绑定的类	用途
parent::	代表父类	子类里访问被重写的父类的同名方法
self::	代表当前定义的类	在类内访问定义自己的类的静态成员
static::	代表当前调用的类	在类内访问当前调用的类内的静态成员


```
1 <?php
2 class Person{
3     public function say(){
4         echo "我是人，我可以说话<br/>";
5     }
6 }
7 class Student extends Person
8 {
9     public function say(){
10        parent::say();
11        echo "我还是一个学生<br/>";
12    }
13 }
14 $stu1 = new Student();
15 $stu1->say();
16
17 ?>
```

访问父类被重写的方法



localhost:81/objectDen x
localhost:81/
我是人，我可以说话
我还是一个学生

```
1 <?php
2 class Person{
3     protected static $country = "中国人";
4     public function say(){
5         echo "我是" . self::$country . "，我可以说话<br/>";
6         echo "我是" . static::$country . "，我可以说话<br/>";
7     }
8 }
9 class Student extends Person
10 {
11     public static $country = "中国学生";
12 }
13 $stu1 = new Student();
14 $stu1->say();
15
16 ?>
```

代表Person类

student类对象调用say()方法，所以此时static代表Student类，Student::\$country='中国学生'



localhost:81/objectDen x
localhost:81/objectDe
我是中国人，我可以说话
我是中国学生，我可以说话

五、类常量

使用常量的好处：（1）数据不能被修改（2）将特定的数据语义化。

1. 普通的常量：一旦定义，不能被修改。在语法层面上能够保证数据不会被修改，并且了解了常量的含义。

定义语法：**define(常量名,常量值)**

```
define(PI, 3.14);  
var_dump(PI);  
  
define(PI, 3.1415926);  
var_dump(PI);
```

```
float 3.14  
float 3.14
```

2. 类常量：如果在类内需要保存一个永远不需要改变的数据，可以使用类常量来实现。

定义语法：**const** 常量名。

注意事项：类常量名前不要有\$,且为大写，定义时赋初值；在类内部使用 `self::`访问，在类外部使用类名::`访问。`

```
1  <?php  
2  class Person{  
3      const SEX_MALE = 1;    //1代表男  
4      const SEX_FEMALE = 2; //2代表女  
5      const SEX_SECRET = 3; //3代表保密  
6      private $sex;  
7      public function setSex($sex){  
8          $this->sex = $sex;  
9      }  
10 }  
11 $p1 = new Person();  
12  
13 //1:字符串形式好理解  
14 $p1->setSex('male');  
15 //2:数字形式计算速度快  
16 $p1->setSex(1);  
17 //3:类常量保存的是数字，速度快；同时解决了语义性差的问题  
18 $p1->setSex(Person::SEX_MALE);|  
19 ?>
```

一般代表状态的数值型数据可以使用类常量来代表。