# 第3章 PHP函数与数组

## 3.1、函数

目标：掌握函数的概念和作用，掌握模块化编程思想，理解函数的各个部分，能够自定义函数解决问题

- 函数的概念和应用
- 函数的基本结构
- 函数的定义规范

### 3.1.1自定义函数

#### 1、认识函数

目标：掌握函数的概念和作用

提问：如果有一个需求，会在不同的应用场景里都出现，name怎么可以让代码可以多处使用呢?

回答：

1、复制粘贴：灵活性差，维护起来要多处维护

2、使用函数：一处维护，多处使用

概念

函数：某段在一起解决某个问题的代码块，给代码块一个标识符，以后可以使用标识符来访问代码块

函数是由一堆代码块组成

- 函数会有一个函数标识（函数名）
- 函数能够在需要用到的时候通过名字直接让代码块运行
- 函数的作用

- [ ] 模块化编程：业务拆分成小模块，然后使用函数进行代码编写
- [ ] 代码的复用：通过调用函数实现函数代码块的重复利用

- 函数的基本结构：

1. 关键字：function
2. 函数名：自定义的名字（代码块标志）
3. 参数：数据的改变（业务相同，数据可以不同）
4. 函数体：解决问题的代码块
5. return：返回值，函数运行的结果处理

```
function 函数名(参数)
{
    函数体(代码块)
    return 返回值
}
```

示例

```
#买火腿
function getGood(){
    #计算位置
    #计算距离
    #计算价格
    #计算时间
    return '火腿';
}
```

#写好上述代码后，凡是要买火腿的时候，都使用getGoods就可以了

小结

1、函数是一起解决某个问题的代码块，解决的问题是两个

- [ ] 模块化编程：业务拆分成小模块，然后使用函数进行代码编写
- [ ] 代码的复用：通过调用函数实现函数代码块的重复利用

2、函数的组成：

1. 关键字：function
2. 函数名：自定义的名字（代码块标志）
3. 参数：数据的改变（业务相同，数据可以不同）
4. 函数体：解决问题的代码块
5. return：返回值，函数运行的结果处理

**2、函数的应用**

目标：了解函数的定义规范和使用

概念

函数应用：是指根据需求定义函数，然后在需要的位置调用函数

定义函数

调用函数：使用函数名+（）即可实现调用

步骤

1、分析需求功能

2、确定函数名

3、确定函数体

4、调用函数

示例

```
#打印九九乘法表
#1、分析需求：打印九九乘法表，可能有多个位置需要使用（因此使用函数开发）
#2、确定函数名：九九乘法表，确定相关性
function chengfa99(){
    #3、确定函数体：输出（不需要返回值）
    for($i=1;$i<=9;$i++){
        for($j=1;$j<=$i;$j++){
            echo "$j*$i=".$j*$i.' ';
```

```
        }
        echo '<br/>';
    }
}
#调用函数
chengfa99();
```

小结

1、函数应用分为两个部分

- 定义函数：根据需求实现函数
- 调用函数：在需要函数运行的时候调用函数

2、函数一旦定义，可以无限次调用

**3、函数内存分析**

目标：了解函数在内存中的运行原理

概念

- 函数运行（调用）的前提：当前函数已经在内存中存在
- 函数不会自动运行
- 函数运行是在栈区开辟内存运行
- 函数运行结束会自动释放所占用内存

步骤

1、脚本编译：将代码加载到内存中

代码被读取（读到内存中：代码写在文件中）

代码被编译：语法出错会报错（结果）

所有代码被加载到内存中

2、代码执行

代码顺序逐行执行

函数结构不会被执行（跳过）

3、函数执行（函数名+（））

- 在内存中寻找函数名对应的函数是否存在
  - [ ] 不存在：报错
  - [ ] 存在：继续
- 在栈区开辟内存加载函数
  - [ ] 参数处理
  - [ ] 函数体执行
- 函数执行结束
  - [ ] 占用内存释放（栈区内存）
  - [ ] 回到函数调用处

4、脚本执行结束：释放所有内存（包括存放代码的代码段）

示例

```
#先调用函数
hello();
world();
#后定义函数
function hello(){
    echo 'hello';
}
```

结果分析：

1、代码编译：函数hello被加载在内存

2、执行代码

hello()执行：发现内存中有：执行，输出hello

world()执行：发现内存中没有：报错

小结

1、函数运行的前提是内存中已经存在该函数

2、函数运行与定义的顺序没有先后顺序（建议先定义后运行）

3、函数的执行是栈区运行（效率高）

4、函数运行完就会释放自己所用的的内存

5、PHP脚本运行结束会释放所有内存

**4、函数形参**

目标：了解形参的概念和意义，灵活运用形参增加函数的灵活性

概念

- 形参：形式参数，指在函数定义结构时所使用到的占位符
- 形参在函数定义时设置的变量
- 形参设定后实在函数内部使用
- 形参名字与外部任何变量无关
- 形参数量根据具体的需要，理论不限（不要太多）
- 形参作用

- [ ] 在函数内部先使用对应的形参假设运算（允许外部调整运算数据）

步骤

1、确定函数需求

2、在定义函数时：确定是否需要形参参与内部运算以及需求数量

3、在函数内部利用形参实现运算

```
#打印九九乘法表
#1、确定需求：打印乘法表，但是不确定层级
```

```
#2、不确定的层级无法完成，所以需要一个变量来控制，通过形参实现$n
function chengfa99($n){
    #3、内部利用$n来代替层级
    for($i=1;$i<=$n;$i++){
        for($j=1;$j<=$i;$j++){
            echo "$j*$i=".$j*$i.' ';
        }
        echo '<br/>';
    }
}
#调用函数
chengfa99(9);
```

2、求两个数的平方和

```
#1、确定需求：两个数的平方和
#2、需要两个数不确定，利用形参代替
function mysum($n1,$n2){
    echo $n1**2 +$n2 **2;
}
```

小结

1、形参是在函数定义时指定的变量

2、形参的作用是在函数内部代替数据进行运行（占位符）

- 形参数量可以根据需求来确定

3、形参作用

- 允许内部数据产生变化（从而运算出不同结果）

**5、函数实参**

目标：了解实参的工作原理，以及实参与形参的关系

概念

实参：实际参数，指在调用函数时传递进去的实际数据

- 实参是调用函数时传递的数据
- 实参数据可以是数据也可以是保存数据的其他表达式

- [ ] 数据常量
- [ ] 变量
- [ ] 常量
- [ ] 运算表达式

实参的本质是将数据赋值给形参

实参需要对准形参（数量、顺序）

步骤

1、确定函数时形参数量和顺序

2、在调用函数时传入相应的实参

数量对号

顺序对齐

示例：

```
function mySum($n1,$n2){
    echo $n1**2+$n2**2;
}
#1、确定形参数量和顺序：数量2个，顺序第一个是整数，第二个也是整数
#2、调用函数：传入实际参数（数据常量）
mySum(2,3);
#调用函数：传入变量和常量
$var1=2;
const V=3;
mySum($var1,V);
#调用函数：传入运算结果
mySum(1+1,1+2);
```

小结

1、实参是在调用时传入的实际数据

- 数据常量
- 变量
- 常量
- 有结果的表达式

2、实参是在调用函数时将值赋值给形参，本质是形参拿到值之后在参与函数运算

3、实参的数量和顺序必须对应形参

- 数量少于形参，运行时形参得不到赋值，会报错
- 数据多于形参，不影响函数运行
- 顺序必须在调用时对应好，系统不能自动匹配

**6. 形参默认值(3.1.2)**

目标：掌握默认值的意义和应用场景，了解默认值的工作原理

概念

形参默认值：是指在定义形参的时候就给定一个大概率值，可以在调用时不传递

默认值在定义时赋值

默认值是最常见出现的值

默认值是调用时才给形参赋值

有默认值后

- 调用时不传实参，系统自动给形参赋值默认值
- 调用时传递实参，系统会使用实参而不是默认值

步骤

1、定义函数：给形参设定默认值

2、调用函数

传递实参，形参使用实参数据赋值

不传递实参，形参使用默认值赋值

```
#打印乘法表
#定义函数：乘法表打印最多就是九九乘法表，所以给定默认值
#2、
function chengfa99($n=9){
    #3、内部利用$n来代替层级
    for($i=1;$i<=$n;$i++){
        for($j=1;$j<=$i;$j++){
            echo "$j*$i=".$j*$i.' ';
        }
        echo '<br/>';
    }
}
#调用函数
chengfa99();
chengfa99(5);
```

小结

1、默认值是在定义函数时给形参设定的预计值

2、默认值设定的通常是参数最可能出现的值

3、默认值的设定目的

让函数能够在不传递实参也能正常工作（便捷性）

允许调用者通过数据改变函数计算结果（灵活性）

4、默认值实在函数调用的时候赋值

调用时没有传递实参：使用默认值赋值给形参

调用时有传递实参：使用实参赋值给形参

**7、参数传值方式**

目标：了解参数传值方式的区别和应用

概念

参数传值方式：在函数调用时，实参给形参传值时所采用的的传值方式

- 参数传值方式在函数定义时规定
- 参数传值方式有两种：

- [ ] 值传递：默认，即外部数据，function a($b)
- [ ] 引用传递：使用$符号，即传递外部变量存储数据的内存地址，如function a(&$b)

步骤

1、确定函数参数的目的

- 外部传入数据内部运算，不影响外部：值传递

- 外部传入数据内部运算，同时外部也受影响，引用传递

2、传递实参时要确定形参的传值方式

- 值传递：数据常量、变量、常量、运算表达式都可以（能有数据都可以）
- 引用传递：变量

示例

```
#值传递
function display1($n){
    while($n>1){
    echo $n--;
    }
}

display1(4);
$n=9
display($n);
echo $n;
echo '<hr/>';

#引用传递
function display(&$n){
    while($n>1){
    echo $n--,'<br/>';
    }
}
#display(2);#报错
$n=9;
display($n);
echo $n;
```

**8、返回值**

目标：掌握返回值的操作，了解return关键字的作用

概念

返回值：即函数运行过程中对调用处返回的处理结果

关键字：return

函数都有返回值

默认返回null

可以明确使用return返回任意数据类型

return会强制结束函数

步骤

1、确定函数功能：是否需要运算结果

- 不需要：不用管返回值
- 需要：在得到结果后将结果返回

2、函数是否需要终止运行，需要的话：直接return

3、在函数调用处对结果进行操作

- 输出结果
- 使用变量保存结果

示例

1、默认返回值

```
function display1(){
    echo __FUNCTION__;
}
$res=display1();   #display1
var_dump($res);   #null
```

2、使用返回值

```
function display2(){
    return __FUNCTION__;
}
$res=display2();
var_dump($res);   #display2
```

3、

```
function mySum($a,$b){
    if(!is_numeric($a)||!is_numeric($b)){
        return;
    }
    return $a+$b;
}
var_dump(mySum('a',1));
var_dump(mySum(1,2));
```

小结

1、函数使用return返回内部结果

- 默认函数最后会自动返回null
- return可以返回任意类型的数据

2、return返回值是返回给函数调用处

3、return一旦运行，之后的其他函数体就不会执行了

**9、函数规则**

概念

函数规则：函数本身没有太多规则，通常是我们开发者对其有一些约定俗称的规范

1、函数命名规则

- 函数名字是由数字、字母和下划线组成，数字不能开头
- PHP中函数名不区分大小写
- PHP中函数名不能重复：即一个脚本运行周期内不能有两个同名函数
- 函数命名应当见面知意
- 当函数有多个单词组成的时候，通常使用以下两种方式

- 驼峰法：第一个单词首字母小写，其他单词首字母大写
- 下划线法：都是小写，但是单词间使用下划线连接（较多使用）

2、函数运用规则

- 函数的目的是为了模块化开发，实现代码的重复利用
- 函数解决问题的颗粒度较小，即函数应该实现小功能（不贪大，越小复用性越高，大问题可以由多个小函数组成，调用即可）

3、函数体规则

函数体主要是用来解决某个具体问题

函数体可以进行定义变量、运算数据、数据判定（分支）、数据重复（循环），也可以通过调用其他函数解决问题

函数体内基本不进行输出操作，如果有数据的话通常是通过返回值返回给调用处

函数操作通常只负责运算，不对结果负责（即结果交给调用处分析判定）

### 3.1.3 作用域

目标：了解PHP作用域的概念，掌握作用域的限制和破解关系

作用域：即作用范围，指变量能够被访问的范围

- 局部作用域
  - 局部变量
- 全局作用域
  - 全局变量
- 超全局作用域
  - 超全局变量
- 跨域访问

**1、局部作用域（局部变量）**

概念

局部作用域：某个函数内部（函数体）的作用域

局部变量：在函数内部定义的变量（形参）

局部变量只能在当前函数内部被访问

形参只能在局部作用域使用

示例：

1、局部作用域定义局部变量

```
#局部作用域
function display(){
    #函数的{}内部属于局部作用域，这里定义的变量属于局部变量
    $a=100;
    echo $a;
}
display(); #输出100
var_dump($a);
```

2、形参属于局部作用域

```
function display(a=1){
    echo $a;
}
display(); #输出1
echo $a;   #报错：  未定义变量
```

小结

1、局部作用域就是函数内部

2、局部作用域里定义的变量是局部变量

3、局部变量只能在对应的局部作用域访问，其他地方不可访问

- 函数外部
- 其他函数内部

4、形参只能在局部作用域中使用

5、扩展：形参引用传值，本质是外部变量与局部变量指向同一个内存地址，在外部访问的是外部变量

**2、全局作用域（全局变量）**

概念

全局作用域：即未在其他结构（函数）内部定义的作用域

- 全局变量：在全局作用域定义的变量
- 全局变量只能在全局作用域访问
- 全局变量可以当做实参使用

示例：

1、全局变量

```
#默认属于全局作用域
$a=100;  #全局变量
```

2、全局变量只能在全局作用域访问

```
$a=100;
function display(){
    echo $a; #报错：未定义的变量
}
display();
```

3、全局变量可以当做实参传递给函数内部使用

```
$a=100;
function display($a){#$a是函数内部的局部变量
    echo $a;
}
display($a); #100,当前是在全局作用域使用
```

**3、超全局作用域（超全局变量）**

概念

超全局作用域：即脚本中任意位置

超全局变量：系统定义的预定义变量

超全局变量不受作用域限制，都可以访问

全局变量都会被收纳在$GLOBALS中

可以利用超全局变量来保存数据，让数据不受作用域限制

示例：

1、超全局变量$GLOABLS

```
#定义全局变量
$a=100;
function display(){
    var_dump($GLOBALS);
    echo $GLOBALS['a']; #输出100
}
display();
```

2、将局部变量保存到$GLOBALS中，也可以任意访问

```
function display(){
    $GLOBALS['a']=100;
}
display();
echo $GLOBALS['a'];
```

小结

1、超全局作用域是PHP脚本任意位置

2、超全局变量是系统定义，开发者一般只用来访问数据（不建议通过超全局变量来让变量跨域）

3、$GLOBALS会自动包含全局变量，通识还有以下超全局变量：

- $_GET:自动接收GET提交的数据
- $_POST:自动接收POST提交的数据
- $_COOKIE:自动接收cookie数据
- $_FILES:自动接收文件上传的数据
- $GLOBALS(自己包含自己)

**4、跨作用域访问**

目标：了解PHP中有实现跨越访问的机制，在必要时指导如何实现跨作用域访问

概念

跨作用域访问：即在局部作用域访问全局变量，或者再全局作用域访问局部变量

- 在函数内部使用global关键字声明变量
- 在函数内部声明一个局部变量
- 在函数外部声明一个全局变量
- 两个变量指向同一个内存
- global声明变量时，不能赋值

步骤

1、定义函数在函数内部使用global声明变量

- 函数外存在同名全局变量：内部创建一个同名局部变量
- 函数外不存在同名全局变量：内部创建一个局部变量，外部创建一个同名全局变量

2、访问修改

- 外部有：可以直接访问
- 外部没有：内部可以直接给值

3、函数运行结束

- 内部局部变量消失
- 外部局部变量存在

示例

1、局部访问全局变量

```
#定义全局变量
$a=100;
function display(){
    #声明变量：引入全局
    global $a;
    echo $a++;
}
display();  #输出100
echo $a;    #输出101
```

2、局部声明全局变量，全局作用域访问

```
#定义全局变量
$a=100;
function display1(){
    #声明变量：引入全局
    global $b;
    $b='hello world';
}
display1();
echo $b;
```

小结

1、global是局部作用域声明全局变量的关键字

- global只能声明，不能赋值
- global声明逻辑
    - 全局变量存在：内部创建一个同名局部变量，并执行外部全部变量
    - 全局变量不存在：内部创建一个局部变量，外部创建一个同名全局变量，并指向同一个内存地址

2、跨作用域访问比较少用，一般要跨作用域访问的都是访问预定义变量

3、作用域划分的目的

- 方便开发者更好的管理数据
- 为了保证数据的安全（同名变量在不同作用域互不干扰）

**5、静态变量**

目标：了解静态变量的作用，掌握静态变量的应用

概念

静态变量：在函数内部定义的，可以在函数多次调用时共用的局部变量

静态变量使用static关键字修饰

静态变量是在函数内部定义的局部变量

静态变量在函数加载（编译）时初始化（只初始化一次）

静态变量可以在同一函数多次调用时共享数据的变化

步骤

1、确定函数内的局部变量是否需要在函数多次调用时共享

不需要：普通局部变量即可

需要：使用static修饰成静态变量

2、在函数内部对静态变量进行运算

示例

当函数被调用时输出函数是第几次调用

```
function display(){
    $count1=0;
    static $count2=0;
    echo ++$count1,++$count2,'<br/>';
}
display();
display();
```

小结

1、静态变量是在函数内部定义的使用static修饰的局部变量

- static修饰的变量是在函数编译（加载）时初始化
- static修饰的变量行在函数调用时跳过执行
- 静态变量是保存在函数里面:所以每次用时修改的值都可以被下次调用使用

2、静态变量的作用

- 让函数在多次调用时能够共享内部数据变化
- 普通局部变量每次调用都会初始化

3、静态变量的使用场景

- 数据需要函数多次调用共享数据
  在函数使用递归存储数据的时候

## 3.1.4、可变函数

概念

可变函数：并不函数可以变化，而是函数的名字具有不确定性，是通过变量保存函数名字，然后也是通过变量+（）的方式来实现函数的访问

1.基本语法

```
<?php
//定义函教
function display(){
    echo " he1lo wor1d' ;
}

//定义一个变里:保存一个字符串与函数名同名
$var = 'display';
//使用可变函数访问函教
$var();  //display: $var代表的结果就是display，然后与()结合就是display()，所以运行display函数
?>
```

2.可变函数通常不是看到一个函数，然后刻意用变量去访问，而是在不确定函数名叫做什么的时候才用这种方式

总结:可变函数一般不会用，但是在一些大的函数（实现功能比较复杂，需要多个函数执行）或者系统函数或者对外提供接口时，会用到这个知识。

```
<?php
//定义一个函教:获取N的3次方
function cube($n){
    return $n**3;
}

//定义函数:函教内部需要通过调用一个外部的函数来辅助操作
function getCube($n,$func_name){
//系统认定$func_name一定要是一个函数《回调函数》
    echo $func_name($n); //调用外部函数，同时传入参数
}
getCube(10, 'cube');//1000，将cube这个外部函数名传入给一个函数，函数在内部调用cube(回调函数的原理就是如此)
?>
```

总结:可变函数一般不会用，但是在一些大的函数（实现功能比较复杂，需要多个函数执行）或者系统函数或者对外提供接口时，会用到这个知识。

## 3.1.5、匿名函数

目标:了解匿名函数，掌握匿名函数的使用

- 简单匿名函数
- 回调匿名函数
- 匿名函数闭包

### 1、简单匿名函数

概念
匿名函数:即没有名字的函数

- 定义函数时函数没有名字
- 使用变量保存函数的内存地址
- 通过变量访问函数
- 匿名函数语法

```
变量= function(){
    函数体
    return 返回值
};  #注意:本质是给变量赋值,所以需要语句结束符
```

示例

```
#定义匿名函数
$func=function($str){
    echo $str;
};
#调用
$func('hello world');#输出hello world
```

小结
1、匿名函数就是函数定义时没有名字
2、简单匿名函数就是通过变量指向无名函数
3、匿名函数是通过变量来进行访问
4、一般情况下匿名函数比较少这么用

**2、回调匿名函数**

概念
回调匿名函数:指直接将匿名函数定义在函数的实参中，当做一个实参传递给形参

示例
1、自定义求一个数组所有元素的立方

```
$num = [1, 2, 3];
#求数组立方的函数:需要一个回调函数专门求一个数的立方
function my_cube($arr,$cube)
{
    $list = [];
    foreach($arr as $value){
        $list[]=$cube($value);
    }
    return $list;
}
#调用
$res=my_cube($num,function($n){return $n**3;});
print_r($res);   #array (0=>1,1=>8,2=>27)
```

小结
1、回调匿名函数就是在当函数调用时需要回调函数时，定义一个匿名函数作为实参
2、回调匿名函数的本质也是在定义一个函数，让变量(形参）进行保存并调用

3、回调匿名函数的作用:匿名函数用后即焚，可以释放掉占用的内存

### 3、匿名函数闭包

概念

闭包:闭包(closure)是一个定义在函数， 能够读取其他函数内部变量的函数（闭包函数）

- 函数定义在某个函数内部（内部函数）
- 内部函数是匿名函数
- 内部函数访问外部函数的局部变量

- [ ] 不能直接访问
- [ ] 访问使用use

示例

1、简单闭包函数

```
function outer(){
    #定义内部函数（闭包函数）
    $inner=function(){
        echo 'hello world';
    };
    $inner();
}
outer();
```

2、闭包函数访问内部变量

```
#错误示例
function outer($str){
    #定义内部函数（闭包函数）
    $inner=function(){
        echo $str;
    };
    $inner();
}
outer('hello world');#报错：提示未定义变量$str
```

原理：

outer是一个函数，$str是outer的一个形参，也就是局部变量，只能在函数内部访问

$inner也是一个函数，函数内部只能访问局部变量，而$str对于匿名函数$inner来说是外部的，所以不可访问

```
#正确示例
function outer($str){
    #定义内部函数（闭包函数）
    $inner=function() use ($str){
        echo $str;
    };
    $inner();
}
outer('hello world');#报错：提示未定义变量$str
```

3、闭包函数可以返回给外部函数调用，这样可以使得闭包所占用的外部函数资源不会被释放

```
#正确示例
function outer($str){
    #定义内部函数（闭包函数）
    $inner=function($s) use ($str){
        echo $str . $s;
    };
    return $inner;
}
$res=outer('hello');
$res('world');
```

原理:
函数outer调用结束，本要释放所有内存
outer返回了一个内部闭包函数的引用，所以函数不会释放到该函数
$inner内部闭包函数引用了一个外部函数的局部变量，所以outer也不能释放$str
$res('world")找到函数执行，并且应用了之前函数的局部变量$str，所以组合输出hello world

小结
1、匿名闭包函数是在函数内部定义的匿名函数
2、虽然闭包函数在函数内部，但是也不能直接访问外部函数的局部变量，需要使用use来引用外部函数变量

3、闭包函数一旦引用了外部函数的局部变量，那么外部函数执行完之后，被引用的局部变量也不会被释放

4、闭包函数可以被返回给外部函数调用，然后在其他时候直接调用内部闭包函数

### 3.1.6、字符串函数

掌握字符串函数的使用，能够对字符串进行长度获取、查找内容、截取子串、字符替换等操作

字符串函数：是PHP的内置函数，用于操作字符串。

| 函数名称 | 功能描述 |
| --- | --- |
| strlen() | 获取字符串的长度 |
| strpos() | 在指定字符串中查找目标字符串首次出现的位置 |
| strrpos() | 获取指定字符串在目标字符串中最后一次出现的位置 |
| str_replace() | 用于对字符串中的某些字符进行替换操作 |
| substr() | 用于获取字符串中的子串 |
| explode() | 将指定字符串拆分成数组 |
| implode() | 用指定的分隔符将数组中的键值拼接成字符串 |
| trim() | 去除字符串首尾处的空白字符（或指定的字符串） |
| str_repeat() | 重复字符串 |
| strcmp() | 比较两个字符串的大小 |

**案例：字符串函数使用示例**

```
echo strlen('abc');                                  // 输出结果：3
echo strrpos('itcast', 'a');                         // 输出结果：3
echo substr('welcome', 3);                           // 输出结果：come
echo str_replace('e', 'E', 'welcome', $count);       // 输出结果：wElcomE
echo $count;                                         // 输出结果：2
```

### 3.1.7、数学函数

数学函数：PHP的内置函数，方便处理程序中的数学运算。

| 函数名 | 功能描述 | 函数名 | 功能描述 |
| --- | --- | --- | --- |
| abs() | 绝对值 | min() | 返回最小值 |
| ceil() | 向上取最接近的整数 | pi() | 返回圆周率的值 |
| floor() | 向下取最接近的整数 | pow() | 返回x的y次方 |
| fmod() | 返回除法的浮点数余数 | sqrt() | 返回一个数的平方根 |
| is_nan() | 判断是否为合法数值 | round() | 对浮点数进行四舍五入 |
| max() | 返回最大值 | rand() | 返回随机整数 |

**案例：数学函数使用示例**

```
echo ceil(5.2);        // 输出结果：6
echo floor(7.8);       // 输出结果：7
echo rand(1, 20);      // 随机输出1到20间的整数
```

### 3.1.8、时间和日期函数

时间和日期函数：PHP的内置函数，用于处理日期和时间。

| 函数名 | 功能描述 |
| --- | --- |
| time() | 获取当前的UNIX时间戳 |
| date() | 格式化一个本地时间/日期 |
| mktime() | 获取指定日期的UNIX时间戳 |
| strtotime() | 将字符串转化成UNIX时间戳 |
| microtime() | 获取当前UNIX时间戳和微秒数 |

**案例：时间和日期函数使用示例**

```
echo time();                    // 输出结果：1582793102
echo date('Y-m-d');             // 输出结果：2020-02-27
echo microtime();               // 输出结果：0.46448400 1582793102
echo microtime(true);           // 输出结果：1582793102.4645
```

**【案例】获取文件扩展名**

在开发文件上传功能时，经常需要判断用户上传文件的类型，看其是否符合要求。例如，网站只允许用户上传JPG格式的商品图片，需要使用PHP来获取上传文件的扩展名。

通过自定义函数和字符串函数实现获取文件扩展名的功能。