

运算符

运算符：operator，是一种将数据进行运算的特殊符号，在 PHP 中一共有十种运算符之多。

赋值运算符

赋值运算：符号是“=”，表示将右边的结果（可以是变量、数据、常量和其它运算出来的结果），保存到内存的某个位置，然后将位置的内存地址赋值给左侧的变量（常量）。

算术运算符

算术运算：基本算术操作

+: 执行数据累加

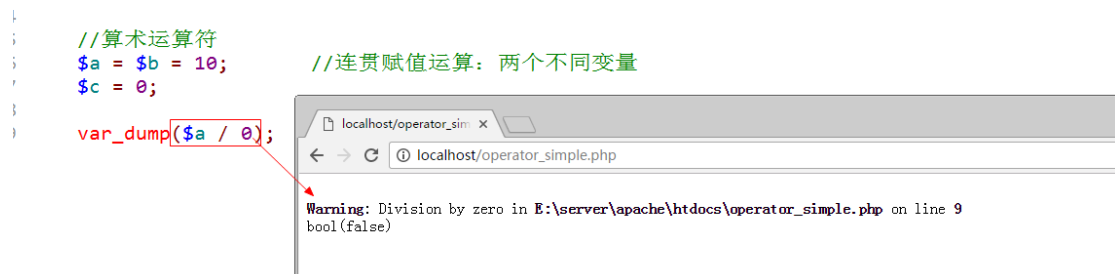
-: 数据相减

*: 键盘上没有乘法符号，使用*代替，两个数相乘

/: 正斜杠代替，表示两个数相除

?: 取余（模）运算，两个数（整数）相除，保留余数

在进行除法运算或者取余运算的时候，对应的除数（第二个数）不能为 0



比较运算符

比较运算：比较两个数据的大小，或者两个内容是否相同，返回的结果都是布尔类型：满足返回 true，不满足返回 false

>: 左边大于右边，返回结果 true

>=: 左边大于等于右边

<: 左边小于右边

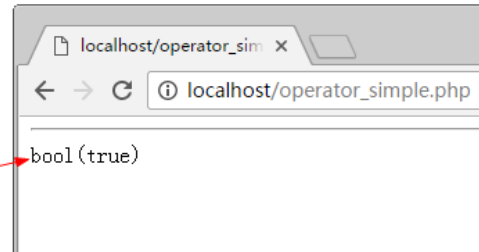
<=: 左边小于或者等于右边

==: 左边的与右边的相同（大小相同）

```

12
13
14 //比较运算符
15 $a = '123'; //字符串
16 $b = 123; //整型
17
18 //判断相等
19 var_dump($a == $b);

```



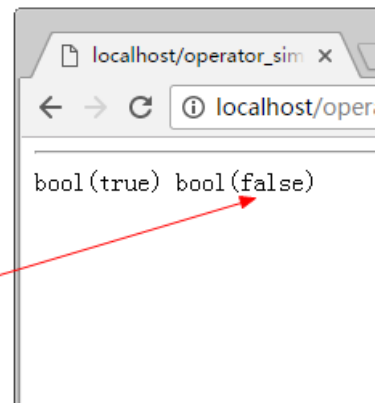
!=: 左边的与右边的不同（大小不同）

===: 全等于，左边与右边相同：大小以及数据的类型都要相同

```

13
14 //比较运算符
15 $a = '123'; //字符串
16 $b = 123; //整型
17
18 //判断相等
19 var_dump($a == $b);
20
21 //全等判断
22 var_dump($a === $b);

```



!==: 不全等于，只有大小或者类型不同

逻辑运算符

逻辑运算：针对不同的结果进行匹配。满足条件返回 **true**，不满足返回 **false**

&&: 逻辑与，左边的条件与右边的条件同时成立（两边结果都为 **true**）

||: 逻辑或，左边的条件或者右边的条件只要有一个满足即可

!: 逻辑非，对已有条件进行取反，本身为 **true**，取反结果就是 **false**

```

echo '<hr/>';

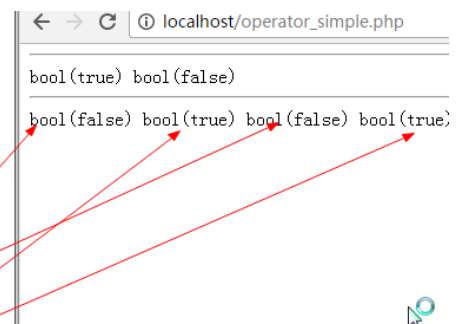
$a = 'weekend';
$b = 'goods';

//逻辑与
var_dump($a == 'weekend' && $b == 'good');

//逻辑或
var_dump($a == 'weekend' || $b == 'good');

//逻辑非
var_dump($b == 'good');
var_dump(!$b == 'good');

```



逻辑与和逻辑或又称之为短路运算：如果第一个表达式结果已经满足条件了，那么就不会运行逻辑运算符后面的表达式：在书写代码的时候，尽量将出现概率最高的（能够直接判断出

结果) 的表达式放到第一位

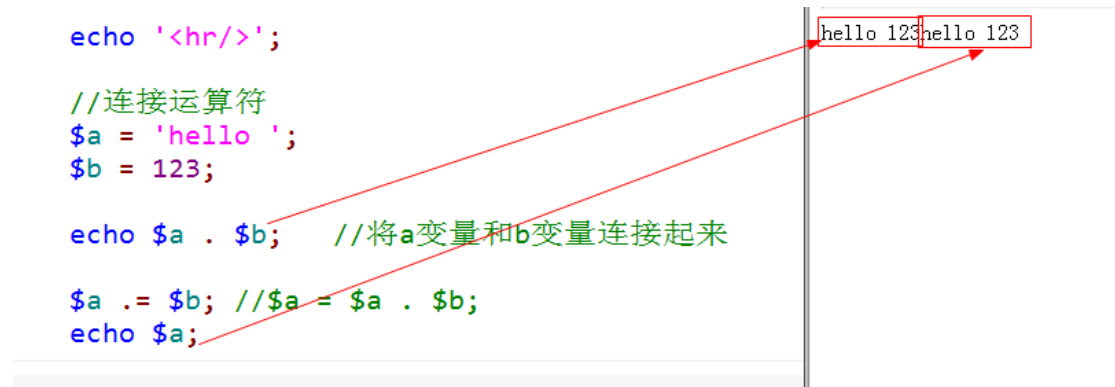
连接运算符

连接运算：是 PHP 中将多个字符串拼接的一种符号

. : 将两个字符串连接到一起

.= : 复合运算，将左边的内容与右边的内容连接起来，然后重新赋值给左边变量

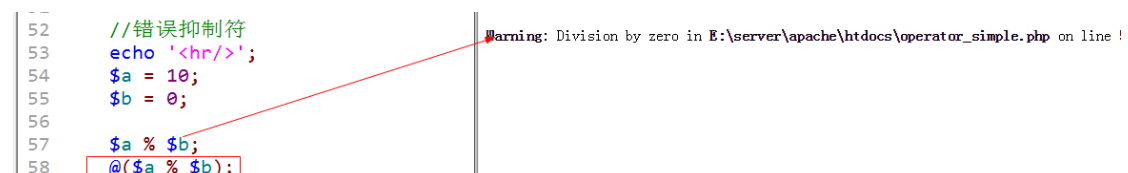
A.= b ===== A = A.b



错误抑制符

在 PHP 中有一些错误可以提前预知，但是这些错误可能无法避免，但是又比希望报错给用户看，可以使用错误抑制符处理。

@: 在可能出错的表达式前面使用@符号即可



错误抑制符通常在生产环境（上线）会用到，在开发的时候不会用：系统本身最好没有任何错误。

三目运算符

三目运算：有三个表达式参与的运算（简单的分支结构缩写）

语法格式：

表达式 1 ? 表达式 2 : 表达式 3;

运算：如果表达式 1 成立，那么执行表达式 2，否则执行表达式 3;

注意：如果表达式本身比较复杂，建议使用括号包起来。

```
//三木运算符
echo '<hr/>';
$a = 10;

$b = $a > 10 ? 100 : 0;
echo $b;
```

三目运算可以进行复合三目运算:三目运算中的表达式2和3都是可以是另外一个三目运算。
表达式1 ? (表达式2 ? 表达式4: 表达式5):(表达式3 ? 表达式5: 表达式6);

自操作运算符

自操作: 自己操作自己的运算符

++: 在原来的值上+1

--: 在原来的值上-1

```
$a = 1;
$a++; // $a = $a + 1;
```

在 PHP 中自操作符是可以放到变量前或者后: 前置自操作和后置自操作

```
$a = 1;
$a++;
++$a; //前置或者后置如果本身只有自操作, 不与其他运算(自操作同时), 那么效果是一样的。但是如果自操作同时还参与别的运算, 那么效果就不一样
```

```
9 //自操作符
0 echo '<hr/>';
1
2 $a = $b = 1;
3
4 $a++;
5 ++$b; //独立操作, 不参与其
6 echo $a, $b;
```

```
$a = 1;
$b = $a++; // $a++ 会导致 $a = $a + 1; $a = 2; , 上面的 $b = 1
$c = ++$a; // ++$a 会导致 $a = $a + 1; $a = 2; , $c = 2;
```

后置自操作: 先把自己所保存的值留下来, 然后改变自己, 自己给别人的值是原来的值;

前置自操作: 先把自己改变, 然后把改变后的值给别人。

```

72     $a = $b = 1;
73
74     $a++;
75     ++$b;           //独立操作，不参与其他运算
76     echo $a,$b;
77
78     echo '<br/>';
79     echo $a++, ++$b; // $a和$b不只是独立运算，还参与了输出
80
81     echo $a,$b;

```

衍生符号：类似自操作

$+=$ ：左边的结果与右边结果相加，然后赋值给左边

$-=$ ：左边的减去右边的结果，然后复制给左边

$*=$ ：乘法操作

$/=$ ：除法操作

$\% =$ ：模操作

注意：右边是一个整体 $\$a += \$b; \Rightarrow \$a = \$a + (\$b);$

```

84
83
84     $a = 10;
85     $b = 5;
86
87     $a += $b;           // $a = $a + $b = 15;
88     $a -= $b - 1;       // $a = $a - ($b - 1); // 15 - 5 + 1 = 11
89     echo '<br/>',$a,$b;

```

如果进行除法或者取余运算，那么要考虑右边表达式的结果是否为 0（为 0 出错）

计算机码

计算机码：计算机在实际存储数据的时候，采用的编码规则（二进制规则）

计算机码：原码、反码和补码，数值本身最左边一位是用来充当符号位：正数为 0，负数为 1

原码：数据本身从十进制转换成二进制得到的结果

正数：左边符号位为 0（正数的原码、反码和补码就是原码本身）

负数：左边符号位为 1

反码：针对负数，符号位不变，其他位取反

补码：针对负数，反码+1

系统中存在两个 0：+0 和 -0

+0: 00000000

-0: 10000000 原码

取反 11111111

补码 00000000

```
4
5 //计算机码
6 $a = 5;
7 $b = -5;
8
9 /*
10 5原码: 00000101
11
12 -5原码: 10000101
13 取反: 11111010 //反码: 符号位不变, 其他位取反
14 求补: 11111011 //补码: 反码+1
15
16 */
```

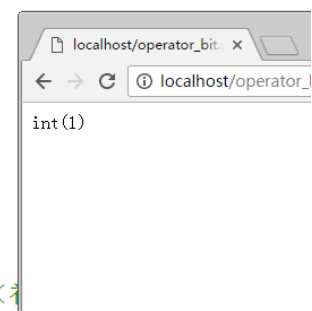
位运算符

位运算: 取出计算机中最小的单位 (位 bit) 进行运算

&: 按位与, 两个位都为 1, 结果为 1, 否则为 0

|: 按位或, 两个有一个为 1, 结果为 1

```
5 //计算机码
6 $a = 5;
7 $b = -5;
8
9 /*
10 5原码: 00000101
11
12 -5原码: 10000101
13 取反: 11111010 //反码: 符号位不变, 其他位取反
14 求补: 11111011 //补码: 反码+1
15
16 */
17
18 //按位与
19 var_dump($a & $b);
20 /*
21 //取出系统存储的结果进行与操作
22 5 00000101
23 -5 11111011
24 & 00000001 //最终结果
25 转换: 判断符号位, 0表示正数 (原码), 1表示负数 (补码)
--
```



注意:

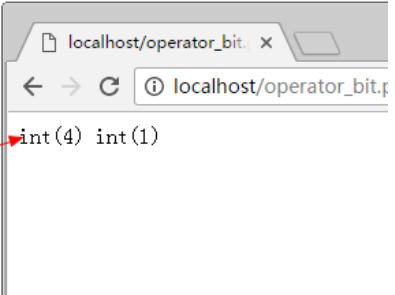
- 1、 系统进行任何位运算的时候都是使用的补码
- 2、 运算结束之后都必须转换成原码才是最终要显示的数据

~: 按位非, 一个位如果为 1 则变成 0, 否则反之

```

6  $a = 5;
7  $b = -5;
8
9  /*
0  5原码: 00000101
1
2  -5原码: 10000101
3  取反: 11111010 //反码: 符号位不变, 其他位取反
4  求补: 11111011 //补码: 反码+1
5
6  */
7  //按位非
8  var_dump(~$b);
9  /*
0      -5  11111011  补码
1  取反   00000100
2  原码   00000100
3  */

```



^: 按位异或, 两个相同则为 0, 不同则为 1

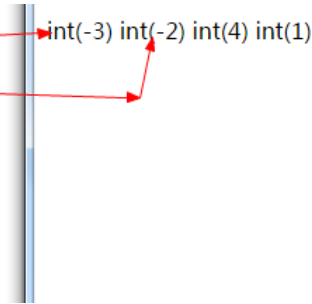
<<: 按位左移, 整个位 (32 位), 向左移动一位, 右边补 0

>>: 按位右移, 整个位向右移动一位, 左边补符号位对应内容 (正数补 0, 负数补 1)

```

18  //按位右移
19  var_dump($b>>1);
20  var_dump($b>>2);
21  /*
22      -5  11111011
23      >>2 11111110 //运算结果: 补码
24      -1  11111101 //反码
25      取反 10000010 //原码: -2
26  */
27

```



按位左移: 乘以 2 的操作

按位右移: 除以 2 的操作 (不完全正确): 整数除 2 会出现小数

运算符优先级

运算符优先级: 在多种运算符同时存在的时候, 如何结合运算

左	[array()
非结合	++ --	递增/递减运算符
非结合	~ - (int) (float) (string) (array) (object) (bool) @	类型
非结合	instanceof	类型
右结合	!	逻辑操作符
左	* / %	算术运算符
左	+ - .	算术运算符 和 字符串运算符
左	<< >>	位运算符
非结合	< <= > >= <>	比较运算符
非结合	== != === !==	比较运算符
左	&	位运算符 和 引用
左	^	位运算符
左		位运算符
左	&&	逻辑运算符
左		逻辑运算符
左	? :	三元运算符
右	= += -= *= /= .= %= &= = ^= <<= >>=	赋值运算符
左	and	逻辑运算符
左	xor	逻辑运算符
左	or	逻辑运算符
左	,	多处用到

优先级降低

流程控制

流程控制：代码执行的方向

控制分类

顺序结构：代码从上往下，顺序执行。（代码执行的最基本结构）

分支结构：给定一个条件，同时有多种可执行代码（块），然后会根据条件执行某一段代码

循环结构：在某个条件控制范围内，指定的代码（块）可以重复执行

顺序结构

顺序结构：最基本结构，所有代码默认都是从上往下依次执行

分支结构

在 PHP 中，分支结构主要有两种：if 分支和 switch 分支

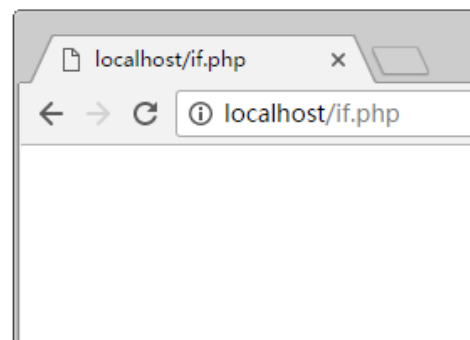
If 分支

if：如果的意思，给定一个条件，同时为该条件设置多种（两种）情况，然后通过条件判断来实现具体的执行段

基本语法：if 分支 PHP 也提供多种方式来实现

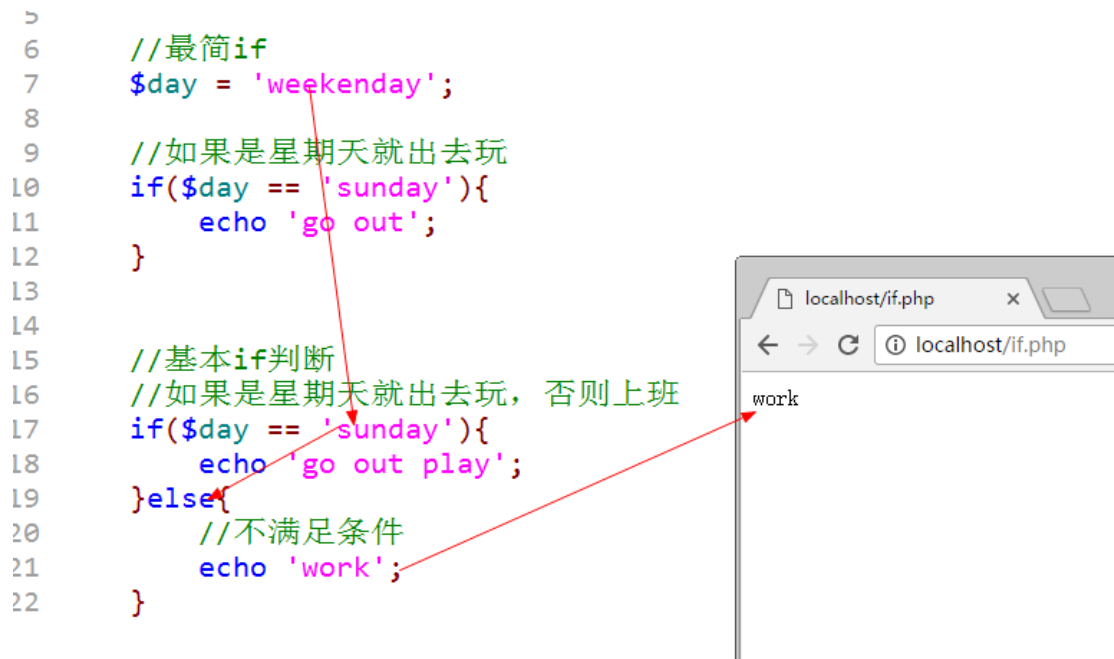
最简 if：只有一段代码，但是可以选择是否执行

```
if(条件表达式){  
    //满足条件所要执行的内容; //顺序结构  
}  
  
//最简if  
$day = 'weekend';  
  
//如果是星期天就出去玩  
if($day == 'sunday'){  
    echo 'go out';  
}
```



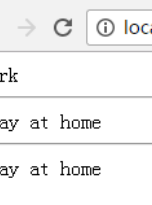
基础 if：有两面性，满足条件或者不满足条件都有对应的执行代码

```
if(条件表达式){  
    //满足条件后执行的代码段;  
}  
else{  
    //不满足条件执行的代码段;  
}
```



复杂 if 结构：在判断条件之后，通常就有两种结果：满足或者不满足，在不满足之后还可以再次进行条件判断

```
if(条件表达式 1){  
    //满足条件表达式 1 的代码段;  
}elseif(条件表达式 2){  
    //不满足表达式 1 条件，但是满足表达式 2 的代码;  
}... //可以使用多个 elseif 来进行再次条件筛选  
Else{  
    //全部不满足要执行的代码;  
}
```



A screenshot of a web browser window. The address bar shows 'localhost/iframe.php'. The page content displays the output of a PHP script: 'work' followed by a horizontal line, 'play at home' followed by a horizontal line, and 'play at home' at the bottom. A red arrow points to the bottom of the page.

- 1、 如果条件比较单一（同一个条件），会采用 **elseif** 复合方式
- 2、 如果判断条件不一致，建议使用嵌套语法（不宜有太多层嵌套：影响代码美观）

Switch 分支

Switch 判断方式： 是将条件放到分支结构内部判断

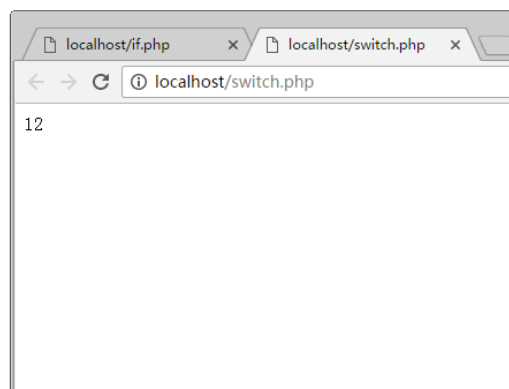
//可以使用类似 **else** 的语法：都不匹配

Default:

//匹配失败的代码;

Break;

```
}
4
5 //根据日期做不同的事情
6 $day = 1;
7
8 //从1到5做不同的事情
9 switch($day){
10     // $day条件一定是个具体的值
11     case 1: // $day == 1
12         echo '1';
13         //break;
14     case 2:
15         echo '2';
16         break;
17     case 3:
18         echo '3';
19         break;
20     // ... 4 ...
21 }
```



If 和 switch 的选择

- 1、 if 能做所有的分支结构事情
- 2、 switch 处理的是条件比较多，同时比较单一，而且是固定值匹配的分支结构

循环结构

循环结构：代码段在一定的控制下，可以多次执行

在 PHP 中循环结构有以下几种：

For 循环：通过条件、起始和终止判断执行

While 循环：通过判断条件终止

Do-while 循环：跟 while 差不多

Foreach 循环：专门针对数组

For 循环

For 循环基本语法

For(条件表达式 1;条件表达式 2;条件表达式 3){

//条件表达式 1：定义初始化条件，可以有多种赋值语句存在，使用逗号分隔即可

//条件表达式 2：边界判定，限定循环执行的次数

//条件表达式 3：用来执行条件变化（自操作）

//循环体

}

```

1 <?php
2
3     //循环结构: for循环
4
5
6     //从1到10输出: 初始为1, 截止为10
7     for($i = 1; $i <= 10; $i++){
8         //输出
9         echo $i, '<br/>';
10    }
11
12    //最后: $i == 11
13    echo $i;

```

For 循环执行原理:

- 1、 执行条件表达式 1: 定义初始化条件 (执行一次)
- 2、 执行条件表达式 2: 判断条件 (N 次)
 - 2.1 满足条件: 执行循环体
 - 2.2 不满足条件: 循环结束
- 3、 执行循环体: (N 次)
- 4、 执行条件表达式 3: 循环变量变化 (N 次)
- 5、 执行条件表达式 2: 判断条件 (N 次)
- 6、 重复执行 3-4-2 步骤: 直到第 2 步不满足条件结束循环

For 循环中条件表达式 1 的多变量定义

```

//最后: $i == 11
//echo $i;

//从1到10输出 (10通常是动态得到)
for($i = 1, $end = 10; $i <= $end; $i++){
    echo $i, '<br/>';
}

```

For 循环特殊使用: for 循环中对应的括号 (条件) 可以一个都没有 (死循环): 一定要避免出现

```

For(;;){
    //循环体
}

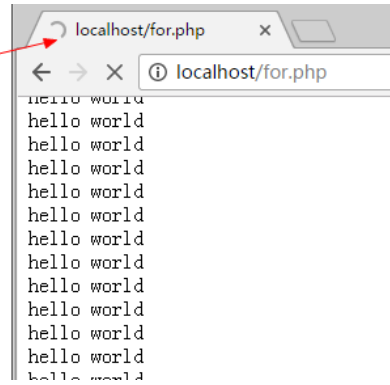
```

```

        //echo $i, '<br/>';
    }

    //无条件for循环
    for(;;){
        echo 'hello world<br/>';
    }

```



循环结构

while 循环

while 循环基本语法:

条件初始化;

while(条件表达式){

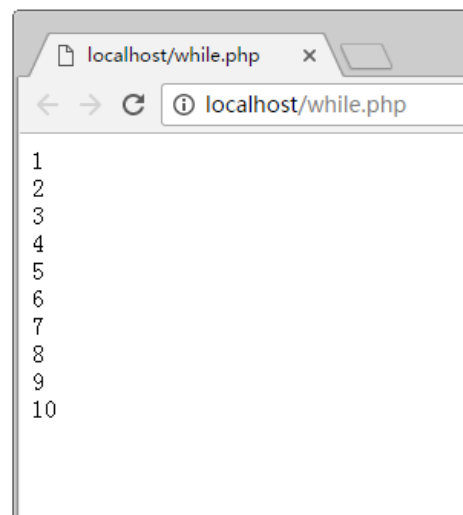
 //条件表达式就是判断边界条件
 循环体; //循环条件的变化

}

```

1
2
3    //while循环
4
5
6    //定义条件
7    $i = 1;
8
9    //循环判定执行
10   while($i <= 10){
11       //循环体
12       echo $i++, '<br/>';
13
14       //循环条件变更
15       //$i++;
16   }

```



For 与 while 的选择

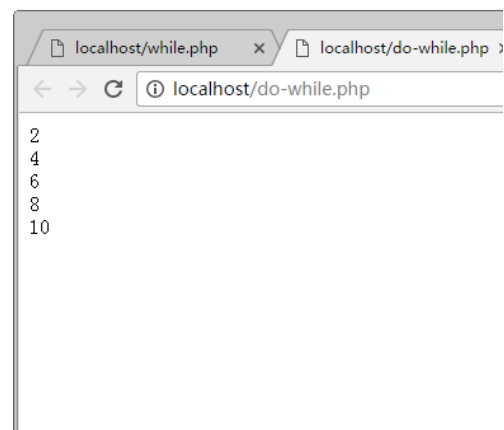
- 1、如果是基于固定已知条件（数值而且是有规律的变化），使用 for 循环
- 2、while 可以做灵活的条件判定（while 使用的比较多）

do-while 循环

do-while: 看着很像 **while**，**while** 首先进行条件判定然后执行循环体，有可能出现第一次就条件不满足，那么就会直接失败（循环体一次都不执行）。**Do-while** 就是先干了再说（执行循环体），后判断条件。（至少会执行一次循环体）

do-while 基本语法:

```
do{  
    //循环体  
}while(条件表达式);  
  
4  
5    //输出1-10之间的偶数（条件判定加入）  
6  
7    //定义基础条件  
8    $i = 1;  
9  
10   //循环判定  
11   do{  
12       //执行输出  
13       if($i % 2 != 1){  
14           //是偶数  
15           echo $i, '<br/>';  
16       }  
17  
18       //条件变更  
19       $i++;  
20   }while($i <= 10);
```



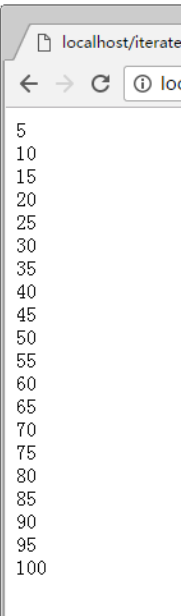
循环结构

循环控制

循环控制: 在循环内部对循环本身进行控制

中断控制: 重新开始循环，循环体中还有其他内容，也再执行
Continue 层级; //默认是 1（循环可以多层嵌套）

```
7
8     $i = 1;
9
10    while($i <= 100){
11
12        //判断：是否是5的倍数
13        if($i % 5 != 0) {
14            //说明当前$i不是5的倍数
15
16            //重新循环
17            $i++;
18
19            //重新循环
20            continue;           //系统重新跳到循环开始处
21        }
22
23        //输出数值
24        echo $i++, '<br/>';
25    }
26 }
```



终止控制：循环直接结束

Break 层级; //默认是 1

```
while($i <= 100){

    //判断：是否是5的倍数
    if($i % 5 != 0) {
        //说明当前$i不是5的倍数

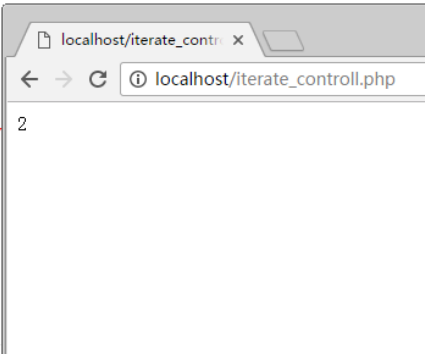
        //重新循环
        $i++;

        //重新循环
        //continue;           //系统重新跳到循环开始处

        //终止循环
        break;
    }

    //输出数值
    echo $i++, '<br/>';
}

echo $i;
```



因为循环经常性地会碰到嵌套（循环中间包含循环），如果在循环内部有些条件下，明确可以知道当前循环（或者说外部循环）不需要继续执行了，那么就是可以使用循环控制来实现：其中内部循环也可以控制到外部，就是通过使用层级参数。

Continue 2; //当前自己循环后面内容不再执行，同时外部循环如果还有循环体也不再执行，重新来过；

Break 2; //当前自己循环结束，同时外部也结束（如果还有外部不受影响，继续执行）

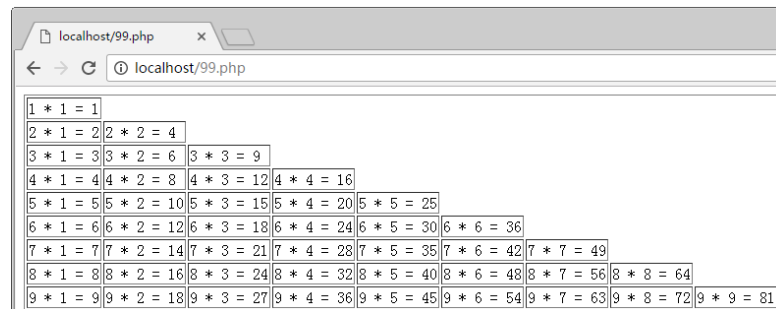
流程控制替代语法

流程控制替代语法：分支和循环结构的替代语法

PHP 本身是嵌入到 HTML 中的脚本语言，需要在 HTML 中书写一些关于判断或者循环的结构语法，必须符合 PHP 标签规范，需要 HTML 与 PHP 进行混搭，如果使用原始的 PHP 代码那么会非常不美观。

需求：打印一个九九乘法表，使用表格来展示

```
1 <table border=1>
2     <?php for($i = 1;$i < 10;$i++){?>
3         <tr>
4             <?php for($j = 1;$j <= $i;$j++){?>
5                 <td>
6                     <?php echo $i . ' * ' . $j . ' = ' . $i * $j;?>
7                 </td>
8             <?php }?>
9         </tr>
10    <?php }?>
11 </table>
```

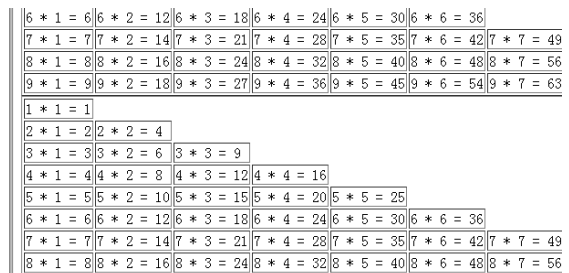


1 * 1 = 1									
2 * 1 = 2	2 * 2 = 4								
3 * 1 = 3	3 * 2 = 6	3 * 3 = 9							
4 * 1 = 4	4 * 2 = 8	4 * 3 = 12	4 * 4 = 16						
5 * 1 = 5	5 * 2 = 10	5 * 3 = 15	5 * 4 = 20	5 * 5 = 25					
6 * 1 = 6	6 * 2 = 12	6 * 3 = 18	6 * 4 = 24	6 * 5 = 30	6 * 6 = 36				
7 * 1 = 7	7 * 2 = 14	7 * 3 = 21	7 * 4 = 28	7 * 5 = 35	7 * 6 = 42	7 * 7 = 49			
8 * 1 = 8	8 * 2 = 16	8 * 3 = 24	8 * 4 = 32	8 * 5 = 40	8 * 6 = 48	8 * 7 = 56	8 * 8 = 64		
9 * 1 = 9	9 * 2 = 18	9 * 3 = 27	9 * 4 = 36	9 * 5 = 45	9 * 6 = 54	9 * 7 = 63	9 * 8 = 72	9 * 9 = 81	

在 PHP 书写到 HTML 中的这些大括号{}非常不美观，所以 PHP 提供了一种替代机制，让其可以不用书写大括号：

For(;;){ ==> for(;;):
} ==> endfor;

```
13
14 <table border=1>
15     <?php for($i = 1;$i < 10;$i++):?>
16         <tr>
17             <?php for($j = 1;$j <= $i;$j++):?>
18                 <td>
19                     <?php echo $i . ' * ' . $j
20                 </td>
21             <?php endfor?>
22         </tr>
23     <?php endfor;?>
24 </table>
```



6 * 1 = 6	6 * 2 = 12	6 * 3 = 18	6 * 4 = 24	6 * 5 = 30	6 * 6 = 36				
7 * 1 = 7	7 * 2 = 14	7 * 3 = 21	7 * 4 = 28	7 * 5 = 35	7 * 6 = 42	7 * 7 = 49			
8 * 1 = 8	8 * 2 = 16	8 * 3 = 24	8 * 4 = 32	8 * 5 = 40	8 * 6 = 48	8 * 7 = 56			
9 * 1 = 9	9 * 2 = 18	9 * 3 = 27	9 * 4 = 36	9 * 5 = 45	9 * 6 = 54	9 * 7 = 63			
1 * 1 = 1									
2 * 1 = 2	2 * 2 = 4								
3 * 1 = 3	3 * 2 = 6	3 * 3 = 9							
4 * 1 = 4	4 * 2 = 8	4 * 3 = 12	4 * 4 = 16						
5 * 1 = 5	5 * 2 = 10	5 * 3 = 15	5 * 4 = 20	5 * 5 = 25					
6 * 1 = 6	6 * 2 = 12	6 * 3 = 18	6 * 4 = 24	6 * 5 = 30	6 * 6 = 36				
7 * 1 = 7	7 * 2 = 14	7 * 3 = 21	7 * 4 = 28	7 * 5 = 35	7 * 6 = 42	7 * 7 = 49			
8 * 1 = 8	8 * 2 = 16	8 * 3 = 24	8 * 4 = 32	8 * 5 = 40	8 * 6 = 48	8 * 7 = 56			

PHP 中具体有哪些替代语法呢？PHP 应该在 HTML 中只做数据输出，输出通常伴有条件判断和循环操作，因此 PHP 提供了对应分支结构和循环结构的替代语法：全部都是对应的一个模式：

左大括号{使用冒号替代：

右大括号}使用 end+对应的起始标记替代

If: if(): endif;

Switch: switch(): endswitch;

For
While
foreach