PHP 语法

- ✓ 出处：站长百科
- ✓ 原文地址：http://www.zzbaike.com/wiki/PHP流程控制
- ✓ 本电子书整理自站长百科PHP流程控制词条，查看内容请访问网站。

✓ 出处：站长百科
✓ 原文地址：http://www.zzbaike.com/wiki/PHP 流程控制
✓ 本电子书整理自站长百科 PHP 流程控制词条，查看内容请访问网站。

PHP嵌入方法

要将 Homepage 中放入 PHP，有以下数种做法

1.<? echo ("这是一个 PHP 语言的嵌入范例\n"); ?>

2.<?php echo("这是第两个 PHP 语言的嵌入范例\n"); ?>

3.<script language="php"> echo ("这是类似 JavaScript 及 VBScript 语法的 PHP 语言嵌入范例");</script>

4.<% echo ("这是类似 ASP 嵌入语法的 PHP 范例"); %>

其中第一种及第两种是最常用的两个方法，在小于符号加上问号后，可以加也可以不加 php 三个字，之后就是 PHP 的程序码。在程序码结束后，加入问号大于两个符号就可以了。第三种方法对熟悉 Netscape 服务器产品的 Webmaster 人员而言，有相当的亲切感，它是类似 JavaScript 的写作方式。而对于从 Windows NT 平台的 ASP 投向 PHP 的用户来说，第四种方法似曾相似，只要用 PHP 3.0.4 版本以后的服务器都可以用小于百分比的符号开始，以百分比大于结束 PHP 的部分，但想用第四种方法的用户别忘了在 php.ini 加入 asp_tags 或是在编译 PHP 时加入 --enable-asp-tags 的选项，才能使第四种方法有效。建议少用第四种方法，当 PHP 与 ASP 源代码混在一起时就麻烦了。

其实，在写作 PHP 的程序最好的方法，就是先处理好纯 HTML 格式的页面文件之后，再将需要变量或其它处理的地方改成 PHP 程序。这种方法，可以让您在开发上达到事半功倍的效果。

PHP常量

PHP 常量是一个简单值的标识符（名字）。如同其名称所暗示的，在脚本执行期间该值不能改变（除了所谓的魔术常量，它们其实不是常量）。常量默认为大小写敏感。按照惯例常量标识符总是大写的。

常量名和其它任何 PHP 标签遵循同样的命名规则。合法的常量名以字母或下划线开始，后面跟着任何字母，数字或下划线。用正则表达式是这样表达的：[a-zA-Z_\x7f- \xff][a-zA-Z0-9_\x7f-\xff]注：在这里，字母是 a-z，A-Z，以及从 127 到 255（0x7f-0xff）的 ASCII 字符。

和 superglobals 一样，常量的范围是全局的。不用管作用域就可以在脚本的任何地方访问常量。有关作用得更多信息请阅读PHP变量作用域。

语法

可以用 define() 函数来定义常量。一个常量一旦被定义，就不能再改变或者取消定义。

常量只能包含标量数据（boolean，integer，float 和 string）。

可以简单的通过指定其名字来取得常量的值，不要在常量前面加上 $ 符号。如果常量名是动态的，也可以用函数 constant() 来读取常量的值。用 get_defined_constants() 可以获得所有已定义的常量列表。注：常量和（全局）变量在不同的名字空间中。这意味着例如 TRUE 和 $TRUE 是不同的。

如果使用了一个未定义的常量，PHP 假定你想要的是该常量本身的名字，如同你用字符串调用它一样（CONSTANT 对应 "CONSTANT"）。此时将发出一个 E_NOTICE 级的错误。参见手册中为什么 $foo[bar] 是错误的（除非你事先用

define() 将 bar 定义为一个常量）。如果你只想检查是否定义了某常量，用 defined() 函数。

常量和变量不同：

- 常量前面没有美元符号（$）；
- 常量只能用 define() 函数定义，而不能通过赋值语句；
- 常量可以不用理会变量范围的规则而在任何地方定义和访问；
- 常量一旦定义就不能被重新定义或者取消定义；
- 常量的值只能是标量。

常量类型

PHP 向它运行的任何脚本提供了大量的预定义常量。不过很多常量都是由不同的扩展库定义的，只有在加载了这些扩展库时才会出现，或者动态加载后，或者在编译时已经包括进去了。

PHP 在常量中定义了以下一些常量。

__FILE__ 这个默认常量是 PHP 程序文件名。若引用文件（include 或 require)则在引用文件内的该常量为引用文件名，而不是引用它的文件名。

__LINE__ 这个默认常量是 PHP 程序行数。若引用文件（include 或 require)则在引用文件内的该常量为引用文件的行，而不是引用它的文件行。

PHP_VERSION 这个内建常量是 PHP 程序的版本，如 '3.0.8-dev'。

PHP_OS 这个内建常量指执行 PHP 解析器的操作系统名称，如 'Linux'。

TRUE 这个常量就是真值（true)。

FALSE 这个常量就是伪值（false)。

E_ERROR 这个常量指到最近的错误处。

E_WARNING 这个常量指到最近的警告处。

E_PARSE 本常式为解析语法有潜在问题处。

E_NOTICE 这个常式为发生不寻常但不一定是错误处。例如存取一个不存在的变量。

这些 E_ 开头形式的常量，可以参考 error_reporting() 函数，有更多的相关说明。

PHP变量

PHP 变量用于存储值，比如数字、字符串或函数的结果，这样我们就可以在脚本中多次使用它们了。在 PHP 中，不需要在设置变量之前声明该变量，不必向 PHP 声明变量的数据类型，根据变量被设置的方式，PHP 会自动地把变量转换为正确的数据类型。

变量的命名规则

在PHP中，变量之前用一个美元符号$引导，例如$a、$temp_358、$test_result_str。

PHP 的变量名区分大小写，例如 $Name 与 $name 是两个不同的变量。

变量名（$之后的部分）与 PHP 中其它标签遵循相同的命名规则：

- 以字母或下划线开头，或者 0x7F-FF 的扩展 ASCII 字符开头
- 后面跟任意数量的字母/数字/下划线，或者 0x7F-FF 的扩展 ASCII 字符

以上命名规则，用正则表达式描述就是：[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*。

PHP 变量的命名习惯

- PHP 中，变量一般以小写命名；大写则一般用于 PHP 常量
- 除少量临时变量之外，一般不使用$a、$t 等简单命名，而采用意义明确的$student_name 等形式。这有助于以后的程序维护。
- 尽管不限制变量名长度，一般使用时控制在 32 字节之内为好。
- 尽管可以使用汉字变量名（例如“$my 汉字变量 1”），但一般不建议这样使用。

PHP变量的使用

PHP 不需要事先申明变量

PHP 给变量赋值： $var_name = value;

PHP变量作用域

在PHP中变量主要有：内置超级全局变量，一般的变量，常量，全局变量，静态变量等.

- 内置超级全局变量可以在脚本的任何地方使用和可见。即如果我们在一个 PHP 页面中改变了其中的一个值，那么在其他 PHP 页面中使用时，它的值也会发生改变。

- 常量一旦被声明将可以在全局可见，也就是说，它们可以函数内外使用，但是这仅仅限于一个页面之中（包含我们通过 include 和 include_once）包含进来的 PHP 脚本，但是在其他的页面中就不能使用了。

- 在一个脚本中声明的全局变量在整个脚本中是可见的，但不是在函数内部，在函数内部的变量如果与全局变量名称相同，以函数内部的变量为准。

- 函数内部使用的变量声明为全局变量时，其名称要与全局变量的名称一致，在这样的情况下，我们就可以在函数中使用函数外部的全局变量了，这样就可以避免上一种因为函数内部的变量与外部的全局变量名称相同而覆盖了外部变量这样的情况。

- 在函数内部创建并声明为静态的变量无法在函数外部可见，但是可以在函数的多次执行过程中保持该值，最常见的情况就是在函数的递归执行的过程之中。

- 在函数内部创建的变量对函数来说是本地的，而当函数终止时，该变量也就不存在了。

超级全局变量的完整列表如下：

- .$GOBALS 所有全局变量数组
- .$_SERVER 服务器环境变量数组
- .$_POST 通过 POST 方法传递给该脚本的变量数组
- .$_GET 通过 GET 方法传递给该脚本的变量数组
- .$_COOKIE cookie 变量数组
- .$_FILES 与文件上传相关的变量数组
- .$ENV 环境变量数组
- .$_REQUEST 所有用户输入的变量数组包括$_GET $_POST $_COOKIE 所包含的输入内容
- .$_SESSION 会话变量数组

实例讲解：

<?php

```
    $a = 4;
    function sendValue($x)
        {
            echo $x;
        }
    sendValue($a);
?>
```

讲解： $a 定义在函数外，函数定义了参数，当函数被调用时，$a 将以参数的形式被传递。因此上面代码能够正常运行。

```
<?php
    $a = 4;
    function sendValue()
        {
            echo $a;
        }
    sendValue();
?>
```

讲解：当函数被调用时，$a 不能以参数的形式被传递。所以上面代码不能够正常运行。

Php5 魔术函数、魔术常量

魔术函数

- __construct() 实例化对象时被调用.

当__construct 和以类名为函数名的函数同时存在时，__construct 将被调用，另一个不被调用。

- __destruct()

当删除一个对象或对象操作终止时被调用。

- __call() 对象调用某个方法，

若方法存在，则直接调用；若不存在，则会去调用__call 函数。

- __get() 读取一个对象的属性时.

若属性存在，则直接返回属性值；若不存在，则会调用__get 函数。

- __set() 设置一个对象的属性时.

若属性存在，则直接赋值；若不存在，则会调用__set 函数。

- __toString()

打印一个对象的时被调用。如 echo $obj;或 print $obj;

- __clone()

克隆对象时被调用。如：$t=new Test();$t1=clone $t;

- __sleep()

serialize 之前被调用。若对象比较大，想删减一点东东再序列化，可考虑一下此函数。

- __wakeup()

unserialize 时被调用，做些对象的初始化工作。

- __isset()

检测一个对象的属性是否存在时被调用。如：isset($c->name)。

- __unset()

unset 一个对象的属性时被调用。如：unset($c->name)。

- __set_state()

调用 var_export 时，被调用。用__set_state 的返回值做为 var_export 的返回值。

- __autoload()

实例化一个对象时，如果对应的类不存在，则该方法被调用。

魔术常量

- __LINE__

返回文件中的当前行号。

- __FILE__

返回文件的完整路径和文件名。如果用在包含文件中，则返回包含文件名。自PHP 4.0.2 起，__FILE__ 总是包含一个绝对路径，而在此之前的版本有时会包含一个相对路径。

- __FUNCTION__

返回函数名称（PHP4.3.0 新加）。自 PHP5 起本常量返回该函数被定义时的名字（区分大小写）。在 PHP4 中该值总是小写字母的。

- __CLASS__

返回类的名称（PHP4.3.0 新加）。自 PHP5 起本常量返回该类被定义时的名字（区分大小写）。在 PHP4 中该值总是小写字母的。

- __METHOD__

返回类的方法名（PHP5.0.0 新加）。返回该方法被定义时的名字（区分大小写）。

PHP运算符

PHP具有C,C 和Java中的通常见到的运算符。这些运算符的优先权也是一致的。赋值同样使用"="。

算术和字符

以下只有一种运算符是有关字符的：

$a $b ： 加

$a - $b ： 减

$a * $b ： 乘

$a / $b ：除

$a % $b ：取模（余数）

$a . $b ：字符串连接

逻辑和比较

逻辑运算符有：

$a || $b ：或

$a or $b ：或

$a && $b ：与

$a and $b ：与

$a xor $b ：异或 （当$a 或$b 为 true 时为 true，两者一样时为 false）

! $a ：非

比较运算符有：

$a == $b ：相等

$a != $b ：不等

$a < $b ：小于

$a <= $b ：小于等于

$a > $b ：大于

$a >= $b ：大于等于

与 C 一样 PHP 也有三重运算符（?:）。位操作符在 PHP 同样存在。

优先权

就和 C 以及 Java 一样！

PHP流程控制

PHP控制语句

- ✓ 出处：站长百科
- ✓ 原文地址：http://www.zzbaike.com/wiki/PHP 流程控制
- ✓ 本电子书整理自站长百科 PHP 流程控制词条，查看内容请访问网站。

IF 语句

IF 语句是多数语言中的一个重要特点，它根据条件执行程序段。PHP 的 IF 语句类似于 C:

```
if (expr)
statement
```

正如在表达式中所论述，expr 被计算为它的真值。如果 expr 为 TRUE，PHP 执行相应语句，如果为 FALSE 则忽略它。

如果$a 大于 $b，下例将显示 ’ a is bigger than b’ :

```
if ($a >$b)
print "a is bigger than b";
```

通常，你希望根据条件执行多于一条语句。当然，不需要给每条语句都加上 IF 判断。取而代之，可以把多条语句组成一个语句组。

If 语句可以嵌套于其他 IF 语句中，使你能够灵活地有条件的执行程序的各个部分。

ELSE 语句

通常你希望满足特定条件时执行一条语句，不满足条件是执行另一条语句。ELSE 就是用来做这个的。ELSE 扩展 IF 语句，在 IF 语句表达式为 FALSE 时执行另一条语句。例如，下面程序执行如果 $a 大于 $b 则显示 ’ a is bigger than b’，否则显示 ’ a is NOT bigger than b’ :

```
if ($a>$b) {
print "a is bigger than b";
          }
else {
    print "a is NOT bigger than b";
      }
```

ELSEIF 语句

ELSEIF,就象名字所示,是 IF 和 ELSE 的组合，类似于 ELSE，它扩展 IF 语句在 IF 表达式为 FALSE 时执行其他的语句。但与 ELSE 不同,它只在 ELSEIF 表达式也为 TRUE 时执行其他语句。

可以在一条 IF 语句中使用多条 ELSEIF 语句。第一个 ELSEIF 表达式为 TRUE 的语句将被执行。在 PHP 3 中,你也可以写成 ’else if’ (写成两个单词)和 ’elseif’ (写成一个单词)效果一样。这只是写法上的细小差别(如果你熟悉 C,它也是)，结果是完全一样的。

ELSEIF 语句仅在 IF 表达式和任何前面的 ELSEIF 表达式都为 FALSE,且当前 ELSEIF 表达式为 TRUE 时执行。

下面是一个含有 ELSEIF 和 ELSE 的嵌套格式的 IF 语句：

```
if ($a==5):
  print "a equals 5";
  print "...";
elseif ($a==6):
        print "a equals 6";
        print "!!!";
      else:
        print "a is neither 5 nor 6";
endif;
```

WHILE 语句

WHILE 循环是 PHP 3 的一种简单的循环。象在 C 中一样。WHILE 语句的基本格式是: WHILE(expr) statement

WHILE 语句的意思非常简单。它告诉 PHP 只要 WHILE 表达式为 TRUE 就重复执行嵌套的语句。每次循环开始时检查 WHILE 表达式的值,所以即使在 嵌套语句内改变了它的值,本次执行也不会终止,而直到循环结束(每次 PHP 运行嵌套的语句称为一次循环)。类似于 IF 语句,你可以用大括号把一组语句括起 来，在同一个 WHILE 循环中执行多条语句:

```
WHILE(expr): statement ... ENDWHILE;
```

下面例子完全相同，都打出数字 1 到 10:

```
/* example 1 */
  $i=1;
  while ($i0);
```

上面循环只执行一次，因为第一次循环后,当检查真值表达式时，它算出来是 FALSE ($i 不大于 0)循环执行终止。

FOR 循环语句

FOR 循环是 PHP 中最复杂的循环。象在 C 中一样。 FOR 循环的语法是:

FOR (expr1; expr2; expr3) statement 第一个表达式(expr1)在循环开始时无条件的计算（执行）。

每一次循环，表达式 expr2 都被计算。如果结果为 TRUE，则循环和嵌套的语句继续执行。如果结果为 FALSE,则整个循环结束。

每次循环结束时，expr3 被计算(执行)．每一个表达式都可为空。expr2 为空则循环的次数不定(PHP 默认它为 TRUE,象 C 一样)。除非你要通过一个条件的 BREAK 语句代替 FOR 的真值表达式来结束循环，否则不要这样。

考虑下面例子。它们都显示数字 1 到 10:

```
/* example 1 */
   for ($i=1; $i10) {
            break;
            }
      print $i;
   }
/* example 3 */
   $i = 1;
```

```
for (;;) {
    if ($i >10) {
        break;
      }
    print $i;
    $i++;
  }
```

当然，第一个例子显然是最好的，但借此你可以发现在 FOR 循环中很多场合可以使用空的表达式。

其他的语言有一条 foreach 语句用来遍历一个数组或哈希（hash）表。PHP 使用 while 语句和 list()、each() 函数来达到这个功能。

SWITCH 选择语句

SWITCH 语句就象是对同一个表达式的一系列 IF 语句。在很多时侯,你想把同一个变量(或者表达式)和许多不同的值去比较 ,并根据不同的比较结果执行不同的程序段。这就是 SWITCH 语句的用处了。

下面两个例子通过不同的方法做同一件事，一个用一组 IF 语句，另外一个用 SWITCH 语句:

```
/* example 1 */
  if ($i == 0) {
    print "i equals 0";
    }
  if ($i == 1) {
    print "i equals 1";
    }
  if ($i == 2) {
```

```
    print "i equals 2";
  }
/* example 2 */
  switch ($i) {
  case 0:
    print "i equals 0";
    break;
  case 1:
    print "i equals 1";
    break;
  case 2:
    print "i equals 2";
    break;
  }
```

REQUIRE 语句

REQUIRE 语句用指定的文件代替自己，很象 C 中的预处理 #include 。这意味着你不能为了每次调用该函数来包含不同文件的内容，而把 require()语句放在一个循环结构，。要这么做，使用 INCLUDE 语句。

```
require(' header.inc' );
```

INCLUDE 语句

INCLUDE 语句包含指定的文件。

每次遇到 INCLUDE 是 INCLUDE 语句就包含指定的文件。所以你可以在一个循环结构中使用 INCLUDE 语句以包含一系列不同的文件。

```
$files = array(' first.inc' , ' second.inc' , ' third.inc' );
for ($i = 0; $i items[$artnr] += $num;
}
```

```
// Take $num articles of $artnr out of the cart
function remove_item($artnr, $num) {
        if ($this->items[$artnr] >$num) {
            $this->items[$artnr] -= $num;
            return true;
          } else {
                return false;
              }
          }
      }
    ?>
```

上面定义了一个叫 Cart 的类，其中包括一个关联数组和两个用来从 cart 中增加和删除项目的函数。

类是实际变量的原始模型。你要通过 new 操作符来建立一个所需类型的变量。

```
$cart = new Cart;
$cart->add_item("10", 1);
```

这建立起一个 Cart 类的对象$cart。该对象的函数 add_item()被调用来给第 10 项加 1。

类可以从其他的类扩充得到。扩充或派生出来的类拥有基类的所有变量和函数及你在扩充定义中所定义的东西。这要使用 extends 关键字。

```
class Named_Cart extends Cart {
    var $owner;
    function set_owner($name) {
        $this->owner = $name;
```

```
        }
    }
```

这里定义了一个名为 Named_Cart 的类它继承了 Cart 类所有变量和函数并增加了一个变量 $owner 和一个函数 set_owner()。 你建立的 named_cart 类的变量现在就能设置 carts 的 owner 了。在 named_cart 变量中你仍然可以使用一般的 cart 函数:

```
$ncart = new Named_Cart; // Create a named cart
$ncart->set_owner("kris"); // Name that cart
print $ncart->owner; // print the cart owners name
$ncart->add_item("10", 1); // (inherited functionality from cart)
```

函数中的变量 $this 意思是当前的对象。你需要使用 $this->something 的形式来存取所有当前对象的变量或函数。

类中的构造器是你建立某种类的新变量时自动被调用的函数。类中和类名一样的函数就是构造器。

```
class Auto_Cart extends Cart {
    function Auto_Cart() {
        $this->add_item("10", 1);
    }
}
```

这里定义一个类 Auto_Cart ，它给 Cart 类加了一个每次 new 操作时设置项目 10 进行变量初始化的构造器。构造器也可以有参数，这些参数是可选的，这种特点也使得其十分有用。

```
class Constructor_Cart {
    function Constructor_Cart($item = "10", $num = 1) {
        $this->add_item($item, $num);
    }
```

```
    }
        // Shop the same old boring stuff.
$default_cart = new Constructor_Cart;
    // Shop for real...
$different_cart = new Constructor_Cart("20", 17);
```

PHP函数

一个函数就是执行特定任务的事物。当要重复做某件事时，写一个函数将会节省大量空间和时间。php 含有大量的内置函数，但为了需要，我们通常自己会自定义很多函数。

函数的定义

- 所有的函数都使用关键词 "function()" 来开始
- 函数名和 PHP 中的其它标识符命名规则相同。有效的函数名以字母或下划线打头，后面跟字母，数字或下划线。
- 添加 "{" - 开口的花括号之后的部分是函数的代码。
- 添加一个 "}" - 函数通过关闭花括号来结束。

例子： 下面这个函数就是自己定义的。

```
<?php

function myCompanyMotto(){

    echo "We deliver quantity, not quality!<br />";

}

?>
```

函数的使用

例子：

```
<?php
function myCompanyMotto(){
    echo "We deliver quantity, not quality!<br />";
}
echo "Welcome to down.zzbaike.com <br />";
myCompanyMotto();
echo "and remember... <br />";
myCompanyMotto();
?>
```

说明：先定义，后引用。这个函数是一个非常简单的函数。它只能输出一个静态的字符串。为了给函数添加更多的功能。我们必须添加参数，参数类似一个变量。

例子：

```
<?php
function writeMyName($fname)
  {
```

```
  echo $fname . " Yang.<br />";  }

echo "My name is ";

writeMyName("David");

echo "My name is ";

writeMyName("Mike");

echo "My name is ";

writeMyName("John");

?>
```

带返回值的函数

例子：

```
<?php

function add($x,$y)

  {

  $total = $x + $y;

  return $total;

  }

echo "1 + 16 = " . add(1,16);

?>
```

PHP常用函数

PHP Date/Time函数 | include,require函数 | list函数

PHP类

类是一个独立的程序单位，是具有相同属性和方法的一组对象的集合。

类的定义

```
<?php
Class abc
{
//成员函数和变量
}
?>
```

PHP函数库

PHP 系统功能

phpinfo

PHP中GET和POST

在php有两种常用的数据获取方法一种是$_GET形式的一种是$_POST形式的,前者一般用于小量数据如地址栏a.php?id=1 而后者post用于表单数据处理.

- get 是从服务器上获取数据，post 是向服务器传送数据。

- get 是把参数数据队列加到提交表单的 ACTION 属性所指的 URL 中，值和表单内各个字段一一对应，在 URL 中可以看到。post 是通 过 HTTP post 机制，将表单内各个字段与其内容放置在 HTML HEADER 内一起传送到 ACTION 属性所指的 URL 地址。用户看不到这个过程。

- 对于 get 方式，服务器端用 Request.QueryString 获取变量的值，对于 post 方式，服务器端用 Request.Form 获取提交的数据。

- get 传送的数据量较小，不能大于 2KB。post 传送的数据量较大，一般被默认为不受限制。但理论上，IIS4 中最大量为 80KB，IIS5 中为 100KB。

- get 安全性非常低，post 安全性较高。但是执行效率却比 Post 方法好。

建议：

- get 方式的安全性较 Post 方式要差些，包含机密信息的话，建议用 Post 数据提交方式;

- 在做数据查询时，建议用 Get 方式;而在做数据添加、修改或删除时，建议用 Post 方式;

PHP中Cookie和Session

Cookie机制采用的是在客户端保持状态的方案，而Session机制采用的是在服务器端保持状态的方案.

Cookie

cookie是一种在浏览器端储存数据并以此来跟踪和识别用户的机制。

PHP在http协议的头信息里发送cookie,因此setcookie()函数必须在其它信息被输出到浏览器前调用，这和对header()函数的限制类似。

- **设置 cookie:**

可以用 setcookie()或 setrawcookie()函数来设置 cookie。也可以通过向客户端直接发送 http 头来设置。

- **使用 setcookie()函数设置 cookie:**

格式： bool setcookie (string name [, string value [, int expire [, string path [, string domain [, bool secure [, bool httponly]]]]]])

name: cookie 变量名 value: cookie 变量的值 expire: 有效期结束的时间, path: 有效目录, domain:有效域名,顶级域唯一 secure: 如果值为 1,则 cookie 只能在 https 连接上有效,如果为默认值 0,则 http 和 https 都可以.

例子:

```
<?php

$value='something from somewhere';
```

```
setcookie("TestCookie",$value);/* 简单 cookie 设置 */

setcookie("TestCookie",$value,time()+3600);/* 有效期 1 个小时 */

setcookie("TestCookie",$value,time()+3600,"/~rasmus/",".example.com",1);/* 有效目录 /~rasmus,有效域名 example.com 及其所有子域名 */

?>
```

设置多个 cookie 变量: setcookie('var[a]','value');用数组来表示变量,但他的下标不用引号.这样就可以用$_COOKIE[‘var’][‘a’]来读取该 COOKIE 变量.

- 使用 header()设置 cookie

header(“Set-Cookie: name=$value[;path=$path[;domain=xxx.com[;]] “);

后面的参数和上面列出 setcookie 函数的参数一样.

比如:$value= ‘something from somewhere ‘;
header(“Set-Cookie:name=$value “);

- Cookie 的读取:

直接用php内置超级全局变量 $_COOKIE就可以读取浏览器端的cookie.上面例子中设置了cookie “TestCookie “,现在我们来读取:

```
print $_COOKIE['TestCookie'];
```

- 删除 cookie

只需把有效时间设为小于当前时间, 和把值设置为空.例如:setcookie(“name“, “”,time()-1);用 header()类似.

- 常见问题解决:
  - 用 setcookie()时有错误提示,可能是因为调用 setcookie()前面有输出或空格.也可能你的文档使从其他字符集转换过 来,文档后面可能带有 BOM 签名(就是在文件内容添加一些隐藏的 BOM 字符).解决的办法就是使你的文档不出现这种情况.还有通过使用 ob_start()函数有也能处理一点.
  - $_COOKIE 受 magic_quotes_gpc 影响,可能自动转义
  - 使用的时候,有必要测试用户是否支持 cookie

- cookie 工作机理:
  - 服务器通过随着响应发送一个 http 的 Set-Cookie 头,在客户机中设置一个 cookie(多个 cookie 要多个头).
  - 客户端自动向服务器端发送一个 http 的 cookie 头,服务器接收读取.

HTTP/1.x 200 OK

X-Powered-By: PHP/5.2.1

Set-Cookie: TestCookie=something from somewhere; path=/

Expires: Thu, 19 Nov 2007 18:52:00 GMT

Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0

Pragma: no-cache

Content-type: text/html

这一行实现了 cookie 功能,收到这行后 Set-Cookie: TestCookie=something from somewhere; path=/浏览器将在客户端的磁盘上创建一个 cookie 文件,并在里面写入: TestCookie=something from somewhere;/

这一行就是我们用 setcookie(‘TestCookie ‘, ‘something from somewhere ‘, ‘/ ‘);的结果.也就是用 header(‘Set-Cookie: TestCookie=something from somewhere; path=/ ‘);的结果.

Session

session使用过期时间是设为 0 的cookie,并且将一个称为session ID的唯一标识符(一长串字符串),在服务器端同步生成一些session文件(可以自己定义session的保存类型),与用户机关联起来.web应用程序存贮与这些session相关的数据,并且让数据随着用户在页面之间传递.

访问网站的来客会被分配一个唯一的标识符，即所谓的会话ID。它要么存放在客户端的cookie，要么经由URL传递。

会话支持允许用户注册任意数目的变量并保留给各个请求使用。当来客访问网站时，PHP 会自动（如果 session.auto_start 被设为 1）或在用户请求时（由 session_start() 明确调用或 session_register()暗中调用）检查请求中是否发送了特定的会话 ID。如果是，则之前保存的环境就被重建。

sessionID 的传送

- 通过 cookie 传送 sessin ID

使用 session_start()调用 session,服务器端在生成 session 文件的同时,生成 session ID 哈希值和默认值为 PHPSESSID 的 session name,并向客户端发送变量为(默认的是)

PHPSESSID(session name),值为一个 128 位的哈希值.服务器端将通过该 cookie 与客户端进行交互.session 变量的值经 php 内部系列化后保存在服务器机器上的文本文件中,和客户

端的变量名默认情况下为 PHPSESSID 的 coolie 进行对应交互.即服务器自动发送了 http 头:header(‘Set-Cookie: session_name()=session_id(); path=/ ‘);即 setcookie

(session_name(),session_id());当从该页跳转到的新页面并调用 session_start()后,PHP 将检查与给定 ID 相关联的服务器端存贮的 session 数据,如果没找到,则新建一个数据集.

- 通过 URL 传送 session ID

只有在用户禁止使用cookie的时候才用这种方法,因为浏览器cookie已 经通用,为安全起见,可不用该方法.<a href=“p.php?<?php print session_name() ?>=<?php print session_id() ?>“>xxx,也可以通过POST来传递session值.

session 基本用法实例

```
<?php

// page1.php

session_start();

echo ‘Welcome to page #1 ‘;

/* 创建 session 变量并给 session 变量赋值 */

$_SESSION['favcolor'] = ‘green ‘;
```

```
$_SESSION['animal'] = 'cat';

$_SESSION['time'] = time();


// 如果客户端使用 cookie,可直接传递 session 到 page2.php

echo '<br /><a href="page2.php">page 2</a>';


// 如果客户端禁用 cookie

echo '<br /><a href="page2.php?'. SID . '">page 2</a>';

/*

默认 php5.2.1 下,SID 只有在 cookie 被写入的同时才会有值,如果该 session 对应的 cookie 已经存在,那么 SID 将为(未定义)空

*/

?>

<?php

// page2.php

session_start();

print $_SESSION['animal']; // 打印出单个 session

var_dump($_SESSION); // 打印出 page1.php 传过来的 session 值
```

?>

'''使用 session 函数控制页面缓存.'''

很多情况下,我们要确定我们的网页是否在客户端缓存,或要设置缓存的有效时间,比如我们的网页上有些敏感内容并且要登录才能查看,如果缓存到本地了,可以直接打开本地的缓存就可以不登录而浏览到网页了.

使用 session_cache_limiter(‘private ‘);可以控制页面客户端缓存,必须在 session_start()之前调用.

控制客户端缓存时间用 session_cache_expire(int);单位(s).也要在 session_start()前调用.

这只是使用 session 的情况下控制缓存的方法,我们还可以在 header()中控制控制页面的缓存.

'''删除 session'''

要三步实现.

<pre>

<?php

session_destroy(); // 第一步: 删除服务器端 session 文件,这使用

setcookie(session_name(),”,time()-3600); // 第二步: 删除实际的 session:

$_SESSION = array(); // 第三步: 删除$_SESSION 全局变量数组

?>

session 使用实例

```
<?php
/**
* 效验 session 的合法性
*
*/
function sessionVerify() {
if(!isset($_SESSION['user_agent'])){
$_SESSION['user_agent'] = MD5($_SERVER['REMOTE_ADDR']
.$_SERVER['HTTP_USER_AGENT']);
}
/* 如果用户 session ID 是伪造,则重新分配 session ID */
elseif ($_SESSION['user_agent'] != MD5($_SERVER['REMOTE_ADDR']
. $_SERVER['HTTP_USER_AGENT'])) {
session_regenerate_id();
}
}
```

```
/**
* 销毁 session
* 三步完美实现,不可漏
*
*/
function sessionDestroy() {
session_destroy();
setcookie(session_name(),”,time()-3600);
$_SESSION = array();
}
?>
```

附录

PHP 编程起步自学教程：

http://down.zzbaike.com/ebook/phpbcqbzx-750.html

PHP 经典实例(第二版)：

http://down.zzbaike.com/ebook/phpjdshl-996.html

PHP 与 MYSQL 彻底研究：

http://down.zzbaike.com/ebook/phpmysqlcdyj-828.html

更多软件、教程下载：http://down.zzbaike.com/

更多PHP信息：http://www.zzbaike.com/wiki/PHP

Apache2.2 中文文档电子书：http://bbs.zzbaike.com/thread-9955-1-1.html

IXWeHosting 控制面板使用手册(在线版+PDF 电子书)：

http://bbs.zzbaike.com/thread-9953-1-1.html

美国主机 IXWebHosting 使用视频教程（在线观看及下载）

http://bbs.zzbaike.com/thread-47008-1-1.html

Godaddy 主机及域名使用视频教程（在线观看及下载）

http://bbs.zzbaike.com/thread-50005-1-1.html

如果您有站长类电子书，请到这里与我们分享：

http://bbs.zzbaike.com/forum-69-1.html

详情见： http://bbs.zzbaike.com/thread-23156-1-1.html

站长百科感谢您下载阅读，多谢支持！