|  |  |  |
| --- | --- | --- |
| 1 | $hi="hello";$$hi="world"; | 可变变量，后者的$hi根据前者而变 |
| 2 | $foo=25;$bar=&$foo; | 引用变量，可以用$bar引用$foo |
| 3 | boolean(布尔值) | 标量 |
| 4 | integer(整型) | 标量 |
| 5 | float（浮点型） | 标量 |
| 6 | string(字符串) | 标量 |
| 7 | array(数组) | 复合类型 |
| 8 | object（对象） | 复合类型 |
| 9 | resource(资源) | 特殊类型 |
| 10 | NULL(空) | 特殊类型 |
| 11 | setType() | 把数据类型转换 |
| 12 | gettype() | 返回数据的类型 |
| 13 | is\_bool() | 判断是否是布尔值 |
| 14 | is\_int() | 判断是不是整数 |
| 15 | is\_float() | 判断是不是浮点数 |
| 16 | is\_string() | 判断是不是字符串 |
| 17 | is\_array() | 判断是不是数组 |
| 18 | is\_object() | 判断是不是对象 |
| 19 | is\_resource() | 判断是不是资源 |
| 20 | is\_null() | 判断是不是为空 |
| 21 | is\_scalar() | 判断是不是标量（布尔值，整型，浮点，字符串） |
| 22 | is\_numeric() | 判断是不是数字 |
| 23 | is\_callable | 判断是不是有效函数名 |
| 24 | define(常量名,常量值或表达式,bool) define("NAME","wlc") 使用 echo NAME | 参数一：常量名，通常为大写。参数二：常量值或表达式。参数三可选，布尔值，是否要区分大小写，默认区分 |
| 25 | \_\_FILE\_\_ | 魔术常量：表示当前文件名 |
| 26 | \_\_LINE\_\_ | 魔术常量：表示当前行数 |
| 27 | \_\_FUNCTION\_\_ | 魔术常量：表示当前函数名 |
| 28 | \_\_CLASS\_\_ | 魔术常量：表示当前的类名 |
| 29 | \_\_METHOD\_\_ | 魔术常量：表示当前对象的方法名 |
| 30 | $a % $b | 算余数 |
| 31 | ,+=,-=.\*=,/=,%=,.= | $a+=3等价于$a=$a+3,其他同 |
| 32 | and 或者 && | 逻辑与：左右两边同时成交，才返回true |
| 33 | or 或者 || | 逻辑或：左右两边有一个成立，就返回true |
| 34 | nor 或者！ | 逻辑非：取反 |
| 35 | xor | 逻辑异或：左右两边有且只有一个成立时，才返回true |
| 36 | ？： 使用：$c=($a<$b)?1:0; 使用：$a<$b?$c=1:$c=0; | 三元运算符，同if |
| 37 | @$sum=1 | ,,,,@是忽略任何错误信息 |
| 38 | => | 键=>值 |
| 39 | -> | 对象->成员 |

|  |  |  |
| --- | --- | --- |
| 40 | if(表达式) elseif(表达式) elseif(表达式) … else | if可以有多个判断 |
| 41 | switch(表达式){case 值1：语句break;...default:语句} | 多向条件分支结构 |
| 42 | while(表达式) 语句; | while语句 |
| 43 | do{ }while(表达式) | 会先执行一次，再作条件判断 |
| 44 | for(初始化;表达式;增量）{ 语句块，即循环体} 使用：for(i=0;i<100;i++){} | for循环 |
| 45 | function function\_name(){ …return;} | 函数遇到return立即结束运行，return可以返回一个任何值，但只能被php接收和处理 |
| 46 | global $a; | 访问全局变量 |
| 47 | static $a=0; | 静态变量，在同一次运行中，运行的结果会累加 |
| 48 | function person($name="张三",$age="33"){} | 设计函数变量的默认值，如果调用函数时没额外定义变量，则按默认变量值来进行 |
| 49 | include();include\_once() | 每次使用时都要重新载入其他php文件，这样就能使用其他php的函数，变量，对象，方法等了。建议在载入不同文件时使用 |
| 50 | require();require\_once() | 使用时，会把引用的php代码替换该函数，建议在同一文件，执行多次代码时使用， |
| 51 | foreach($array as $key => $value){} | 遍历数组，$key代表数组的键，$value代表数组的值 |
| 52 | current() | 取得当前指针位置的内容 |
| 53 | key() | 返回当前针对对应的索引指 |
| 54 | next() | 将指针下移一位 |
| 55 | prev() | 将指针上回倒一位 |
| 56 | end() | 指向最后一个元素 |
| 57 | reset() | 返回第一个元素的位置 |
| 58 |  | 以上和jquery的元素选择操作有点像 |
| 59 | $a=1;$GLOBALS["a"] | 访问全局变量，如果$a |
| 60 | array\_values(数组) | 返回数组的所有的值，并按数字重新索引组成新数组。 |
| 61 | array\_keys(数组) | 返回数组的所有键，并按数字重新索引组成新数组。 |
| 62 | in\_array(搜索内容，数组，bool是否区分大小分） | 判断内容是否存在于数组，返回搜索结果的布尔值 |
| 63 | array\_key\_exists() | 判断键是否存在 |
| 64 | array\_flip(数组) | 把数组内的键和值位置互换，返回一个新数组 |
| 65 | array\_reverse(数组) | 把数组内的排序倒过来，返回一个新数组 |
| 66 | count(数组，1或0),sizef() | 计算数组内元素个数，1或0表示是否计算二维数组内的元素个数 |

|  |  |  |
| --- | --- | --- |
| 67 | array\_count\_values(数组) | 计算每个值出现的个数，返回值对应个数的新数组 |
| 68 | array\_unique(数组) | 删除重复值，返回一个新数组 |
| 69 | array\_filter(数组，函数名) | 根据函数筛选数组元素，返回一个新数组 |
| 70 | array\_map(回调函数，数组，更多数组） |  |
| 71 | sort(数组)，rsort(数组) | 对数组的值进行从小到大排序（另一个从大到小），并返回给数组，sort(数组)整个这个是一个布尔值 |
| 72 | ksort(数组)，krsort(数组) | 对数组的值进行从小到大排序（另一个从大到小），并返回给数组，ksort(数组)整个这个是一个布尔值 |
| 73 | natsort(数组) | 自然排序，从1-9，从a-z，从小到大，从短到长，非常适合模糊排序，推荐排序 |
| 74 | array\_slice(数组,1,2) | 返回数组序号1开始的2个元素，保留键名 |
| 75 | array\_splice(数组,1);array\_splice(数组,-1) | 删除数组序号1之前的元素;只保留数组序号1之前的元素 |
| 76 | array\_combine(数组1,数组2) | 组合数组，数组1是键名，数组2是键值，若不对应，就会返回false |
| 77 | array\_merge(数组1,数组2);array\_merge(数组) | $a和$b合并，并剔除重复元素，保留重复的最后一个元素,merge是融入的意思，#合并数组最佳方式#;键名从0开始排序 |
| 78 | array\_intersect(数组1,数组2) | 数组1和数组2取交集，相交的值按数字排序，生成新的数组 |
| 79 | array\_diff(数组1,数组2) | 数组1和数组2取差集，相差的值按数字排序，生成新的数组 |
| 80 | array\_rand(数组);array\_rand(数组,2) | 随机返回一个键值，随机返回两个键值，并数字排序 |
| 81 | shuffle(数组)) | 将数组随机排序，并返回布尔值，以数字排序 |
| 82 | array\_sum(数组) | 只计算键值总和 |
| 83 | range(0,5);range(0,50,10);range("a","e") | 生成从1-5的数组;生成从1-50的数组，步长是10;生成从a到e的数组 |
| 84 | 数组1+数组2 | $a和$b合并，但和merge不同，此处是前者覆盖后者 |
| 85 | unset($a[2]) | 删除下标为2的元素，并且排序不变 |
| 86 | array\_values(数组) | 重新排序 |
| 87 | $this | $this在class中是指本对象 |
| 88 | $this->name | 使用已经声明的本对象内的成员属性 |
| 89 | $this->function() | 使用本对象内的方法（函数） |
| 90 | function \_\_construct($name="wlc",$age="23"){ $this->name=$name; $this->age=$age; } | 构造函数，用来初始化对象，调用类之前，会先执行该函数 如果外界调用这个类，未重新定义name，age，则使用此处的默认设置 |
| 91 | function \_\_destruct()  {echo "执行结束后，统一加上这一句".$this->name;} | 析构函数，全部对象引用执行完之后，最后的时候调用1次，常用来清理！ |
| 92 | private class person(){};private $age;private function say(){} | 私有化对象，属性，方法，用来封装函数，禁止外界访问和设置 |

|  |  |  |
| --- | --- | --- |
| 93 | function \_\_set($privateName,$privateValue){ $this->$privateName=$privateValue;} | set魔术方法，通过这个，可以集中去访问私有变量，并设置私有变量，只写，不能读 |
| 94 | function \_\_get($privateName){  if ($privateName=="name") {  return "保密";}  if ($privateName=="age") {  return $this->age;}} | get魔术方法，通过这个，可以集中去访问私有变量，只读，不写 |
| 95 | function \_\_isset($privateName) {  if ($privateName=="name") {//禁止判断name是否存在  return;}  return isset($privateName);//允许判断其他私有属性是否存在} | isset魔术方法，通过这个，可以集中去访问私有变量，判断私有变量是否存在，只读，不写 |
| 96 | class student extends person { public $student;  public function study(){  echo $this->student."正在学习";  }} | 继承父类，学生这个类，继承人这个类，拥有人的属性和方法 |
| 97 | final class person{} | 因为person类加了 final，将不能被继承，但我感觉final不能限制方法 |
| 98 | const URL='www.baidu.com'; | 声明常量，不能用$,通常用大写表示，访问的话，同静态变量的方法 |
| 99 | function \_\_call($functionName,$args)  { echo "你所调用的函数".$functionName."不存在"} | call魔术方法，外部调用的函数不存在里，会调用此函数，输出错误提示 |
| 100 | serialize(实例化的类) | 对象串行化，就是把对象里的一些定义的变量，数值，记录下来，并不是把对象全部都记下来。 |
| 101 | file\_put\_contents("文件名","字符串"); | 把字符串写入例如txt的文件中 |
| 102 | substr("字符串",2,4) | 输出从序号2开始的，后4位字符串 |
| 103 | $str = "lamp";$str[1], | 访问序号为1的字符串，php4以前的方式 |
| 104 | $str = "lamp";$str{1}, | 访问序号为1的字符串，php4以后的方式 |
| 105 | strlen($str) | 计算字符串长度 |
| 106 | header("Content-type:text/html;charset=utf-8"); | php的header头，让浏览器正确显示中文字符串 |
| 107 | echo 'this','is','echo','可以用逗号隔开，但字符串输出后依然会连接在一起'; | 用逗号隔开，但输出时，仍然是一整句话 |
| 108 | $str=" 123456 "; echo strlen($str); echo strlen(ltrim($str)); echo strlen(rtrim($str)); echo strlen(trim($str));  $str="123 This is a test ..."; echo ltrim($str,"0..9"); echo ltrim($str,"."); echo ltrim($str,"0..9 A..Z ."); | 前面4个空格，后面3个空格 输出13 去掉左侧空格，输出9 去掉左侧空格，输出10 去掉所有空格，输出6   去掉所有数字 去掉所有点 去掉所有数字，大写字母，点 |

|  |  |  |
| --- | --- | --- |
| 109 | $str="123456"; str\_pad($str,10);  str\_pad($str,10,"-+",STR\_PAD\_LEFT) ;  str\_pad($str,10,"-+",STR\_PAD\_RIGHT) ;  str\_pad($str,10,"-+",STR\_PAD\_BOTH) ;  str\_pad($str,10,"-+") | 默认在右侧用空格填补，在左边，右边，两边用指定符号填充字符串 |
| 110 | $str="<h1>hello</h1><div>haha</div>" ; echo htmlspecialchars($str); echo strip\_tags($str,"<h1>"); | 会把html的标签字符串化 用于删除所有标签，div的标签会被删除，h1有标签会被保留 |
| 111 | str="this is test you know" ; echo strstr($str,"test"); | 返回test出现后的，包括test的字符串 |
| 112 | $str="thisistestyouknow" ; echo strrpos($str,"test"); | 返回test第1次出现的位置，从0开始计数，返回6 |
| 113 | $str="this is test you know test test test test" ; str\_replace("test","success",$str,$count); | 在$str中用success替换test，返回替换后的字符串，并计算替换的次数 |
| 114 | error\_reporting(0); error\_reporting(E\_ALL); error\_reporting(E\_ALL & ~ENOTICE); | 完全关闭错误报告 抛出所有错误报告 抛出除了注意错误之外的错误报告 |
| 115 | try{$error = 'i am error';//创建一个异常对象  throw new Exception($error);//用该语句抛出 } catch (Exception $e){  echo 'caught exception:', $e->getMessage(),"<br>";//捕获异常，并输出 }echo "hello";//程序没有崩溃，则向下执行 | |
| 116 | date("Y-m-d",mktime(0,0,0,12,36,2008)); date("Y-m-d",mktime(0,0,0,14,28,2008)); date("Y-m-d",mktime(0,0,0,12,28,88); | 自动纠正日期，超出5天，放到下个月的5天  年份自动加19 |
| 117 | date("Y-m-d",strtotime("now")) date("Y-m-d",strtotime("yesterday") date("Y-m-d",strtotime("last monday")) | strtotime能自动识别一些固定格式英文的日期 |
| 118 | getdate(time()) | 把当前的时间戳变成一个格式化的时间数组 |
| 119 | date("Y年m月d日 H:i:s") | 输出当前格式化的时间 |
| 120 | microtime() | 输出毫秒级时间 |
| 121 | filetype('01-error.php') | 输出文件类型 |
| 122 | var\_dump(file\_exists('01-error.php'));  echo filesize('01-error.php'); var\_dump(is\_readable('01-error.php')); var\_dump(is\_writeable('01-error.php')); var\_dump(is\_executable('01-error.php')); echo filectime('01-error.php'); echo filemtime('01-error.php'); echo fileatime('01-error.php'); var\_dump(stat('01-error.php')); | 判断文件是否存在，大小，是否可读，可写，可操作，创建时间，修改时间，访问时间，stat返回一个以上信息的数组 |

|  |  |  |
| --- | --- | --- |
| 123 | $path="01-error.php"; echo basename($path)."<br>"; echo basename($path,".php")."<br>"; echo dirname($path)."<br>"; var\_dump(pathinfo($path)); | 返回目录下的文件名，返回去掉后缀的文件名，返回目录名，返回一个关于目录信息的数组 |
| 124 | $dirname='dirname'; $dir\_handle = opendir($dirname); rmdir("php"); | 在当前目录下创建php文件夹，文件夹不能已经存在 在当前目录下删除php文件夹，文件夹得为空才能删除 |
| 125 | $handle=fopen("test.txt","a"); fwrite($handle,"hello wlc "); | 注意fopen的第二个参数，有很多，记得看说明，现在这个是在文件的最后插入新内容 |
| 126 | $handle=fopen("test.txt","r"); echo fread($handle,20); | 注意fopen的第二个参数，有很多，记得看说明，现在这个是只读；读取从头开始的20个字节 |
| 127 | print\_r(file('test.txt')); | 会把文本以每行输出【以数组的形式】，行对应从0开始的下标 |
| 128 | copy('test.txt','dirname/test.txt'); | unlink('test.txt')删除文件，rename()重命名文件 |
| 129 | $filename="dirname/3.jpg"; header('Content-Type: image/jpg'); header('Content-Disposition: attachment; filename="'.$filename.'"'); header('Content-Length:'.filesize($filename)); readfile($filename); | //这里设置文件的地址 //声明文件类型 //下载文件的描述 //下载文件的大小 //将文件内容读取出来 |
| 130 | <<<EOT  这里面的字符串，函数，变量都会被解析  EOT; | 在Heredoc的格式，适合引用大段字符串文本 |
| 131 | $array = [  "foo" => "bar",  "bar" => "foo", ]; | 自php5.4起可以这样定义数组 |
| 132 | key 可以是 integer 或者 string。value 可以是任意类型。 此外 key 会有如下的强制转换：  - 包含有合法整型值的字符串会被转换为整型。例如键名 "8" 实际会被储存为 8。但是 "08" 则不会强制转换，因为其不是一个合法的十进制数值。  - 浮点数也会被转换为整型，意味着其小数部分会被舍去。例如键名 8.7 实际会被储存为 8。  - 布尔值也会被转换成整型。即键名 true 实际会被储存为 1 而键名 false 会被储存为 0。  - Null 会被转换为空字符串，即键名 null 实际会被储存为 ""。  - 数组和对象不能被用为键名。坚持这么做会导致警告：Illegal offset type。 | |
| 133 | 如果在数组定义中多个单元都使用了同一个键名，则只使用了最后一个，之前的都被覆盖了。 | |

|  |  |  |
| --- | --- | --- |
| 134 | 双冒号：：和->这两种方法引用类里面的属性（即变量）和方法（即函数）的区别  class Test {  public static $var1 = “static variable 1";  public $var2 = "object property 2";   public function a() { echo "method a"; }  public static function b() { echo "static method b"; } } 类里的静态属性和function调用时是不需要先declare（定义，附值）的，直接用双冒号 echo Test::var1; //表示引用Text类下面的var1属性（即变量） Test::b();//表示引用Text类下面的b方法（即函数） 非静态的必须要在类被declare（定义，附值）的情况下才能被引用，是用 -> $o\_test = new Test; echo $o\_test->var2; $o\_test->a(); | |
| 135 | parent::类名 | 来访问被覆盖的方法或属性 |
| 136 | 定义一个函数 这个函数里的变量都是局部变量 它们只能在这个函数内部使用 return是这个函数的返回值 调用这个函数就会输出这个值 这个值可以是函数内的变量 也就是局部变量 | |
| 137 | 全局变量只要是在这个页面里就能使用 |  |
| 138 | 超全局变量 在所有页面都能使用 |  |
| 139 | mixed | 说明一个参数可以接受多种不同的（但不一定是所有的）类型 |
| 140 | number | 说明一个参数可以是 integer 或者 float |
| 141 | callback | 说明参数是一个函数 |
| 142 | void | void 作为返回类型意味着函数的返回值是无用的。void 作为参数列表意味着函数不接受任何参数。 |
| 143 | $foo = ( int ) $bar;$fst = (string) $foo | 强制转化为整数，字符串 |
| 144 | class Base {  public function sayHello() {  echo 'Hello ';}} trait SayWorld {  public function sayHello() {  parent::sayHello();  echo 'World!';}} class MyHelloWorld extends Base {  use SayWorld;} $o = new MyHelloWorld(); $o->sayHello(); ?> | 插入：从基类继承的成员被插入的 SayWorld Trait 中的 MyHelloWorld 方法所覆盖。其行为 MyHelloWorld 类中定义的方法一致。优先顺序是当前类中的方法会覆盖 trait 方法，而 trait 方法又覆盖了基类中的方法。 |
| 145 | public string \_\_toString ( void ) | toString魔术方法，当类被当作是一个字符串来用时，原本是要报错的，但使用了这个魔术方法，则会由他来输出 |
| 146 | \_\_invoke() | toString魔术方法，当类被当作是一个函数来用时，原本是要报错的，但使用了这个魔术方法，则会由他来输出 |

|  |  |
| --- | --- |
| 147 | session\_abort — Discard session array changes and finish session session\_cache\_expire — 返回当前缓存的到期时间 session\_cache\_limiter — 读取/设置缓存限制器 session\_commit — session\_write\_close 的别名 session\_decode — 解码会话数据 session\_destroy — 销毁一个会话中的全部数据 session\_encode — 将当前会话数据编码为一个字符串 session\_get\_cookie\_params — 获取会话 cookie 参数 session\_id — 获取/设置当前会话 ID session\_is\_registered — 检查变量是否在会话中已经注册 session\_module\_name — 获取/设置会话模块名称 session\_name — 读取/设置会话名称 session\_regenerate\_id — 使用新生成的会话 ID 更新现有会话 ID session\_register\_shutdown — 关闭会话 session\_register — Register one or more global variables with the current session session\_reset — Re-initialize session array with original values session\_save\_path — 读取/设置当前会话的保存路径 session\_set\_cookie\_params — 设置会话 cookie 参数 session\_set\_save\_handler — 设置用户自定义会话存储函数 session\_start — 启动新会话或者重用现有会话 session\_status — 返回当前会话状态 session\_unregister — Unregister a global variable from the current session session\_unset — 释放所有的会话变量 session\_write\_close — Write session data and end session |