第3章

后端技术

Web 应用的前端，包括 Web 页面的结构、Web 的外观视觉表现以及 Web 层面的交互实现，这些都是 Web 应用中用户看得见、碰得着的东西；而 Web 应用的后端，更多的是应用与数据库的交互，从而实现复杂的业务逻辑。所以，后端技术的重点在于，如何实现业务逻辑，如何存储数据，如何让应用具有优越的性能，如何提高应用运行的稳定性等。

本章介绍 PHP 项目开发的后端技术，包括 PHP 基本语法、面向过程程序设计、面向对象程序设计以及 PHP 应用扩展等内容。


视频讲解

3.1 PHP 语言基础

在使用任何一门程序设计语言进行编程之前，都需要详细了解该语言的词法结构和基本语法规则。例如，该语言有哪些数据类型，如何定义变量和常量，怎样控制程序的流程，如何处理程序的错误等。

PHP 的语法规则与大家熟悉的 C、C++ 和 Java 非常相似，所以这里只对 PHP 进行概要性的介绍。

3.1.1 语法基础

PHP 是一种嵌入式的程序设计语言，其代码的编写，除需要特定的语言标记之外，其他语法与 C、C++、Java 等程序设计语言基本相同。

1. PHP 语言标记与注释

PHP 代码常常被嵌入 HTML 页面中，因此，为了区分 HTML 与 PHP 代码，需要使用特定的标记将 PHP 代码进行界定。PHP 提供如下两种形式的语言标记。

1）标准形式

开始标记：<? php ；结束标记：?>。例如：

```
<?php
    echo "HelloWorld";
?>
```

PHP 的所有版本均支持该标记。当一个文件中只有 PHP 代码时，一般省略结束标记，且开始标记最好顶格书写。

2）短风格形式

开始标记：<? ；结束标记：?>。例如：

```
<?
    echo "Hello World";
?>
```

使用 PHP 的短风格标记，需要开启 PHP 的 short_open_tag 指令。若要使用该标记进行快速输出，还可以采用如下形式。

```
<? = "Hello World" ?>
```

PHP 的所有版本均支持该输出形式的短风格标记，与 PHP 的 short_open_tag 指令是否开启无关。

在程序设计过程中，为了便于代码的阅读与维护，在编写某行代码或功能模块时，常常需要添加注释，来对代码进行解释或说明。PHP 的注释分为单行注释和多行注释。

1）单行注释

PHP 的单行注释使用“//”或“#”两种格式。例如：

```
<?php
    $str = 'PHP'; # 定义变量
    echo $str;          //输出变量的值
?>
```

2）多行注释

PHP 的多行注释使用“/* */”或“/** */”两种格式。例如：

```
<?php
    /*
     * 下面的代码用于测试 PHP 中的单行注释
     */
    $str = 'PHP'; # 定义变量
    echo $str;          //输出变量的值
?>
```

或

```
<?php
/**
 * 获取参数的值
 * @param unknown $param
```

```
 * @return unknown
 */
function get( $param) {
    return $param;
}
?>
```

在 PHP 中,多行注释可以嵌套单行注释,但不能再嵌套多行注释。

2. 标识符与关键字

在程序设计过程中,经常需要定义一些符号来标记一些名称,例如变量名、函数名、类名等,这些符号被称为标识符。

PHP 中的标识符必须满足以下规定。

(1) 标识符只能由字母、数字和下画线组成。

(2) 标识符可以由一个或多个字符组成,且必须以字母或下画线开头。

(3) 当标识符用作变量名时,区分大小写。

(4) 标识符可以是任意长度。

(5) 标识符不能与 PHP 预定义关键字相同。

与 C、C++、Java 等语言不同,在 PHP 中,变量名是可以使用其关键字的,但建议最好不要这样做,以避免产生不必要的麻烦。

PHP 的关键字是预先定义好并赋予了特殊含义的单词,也称为保留字,包括关键词、预定义类、预定义常量,以及其他保留字。详情请参见 PHP 手册附录。

3. 变量与常量

变量是可以在不同时刻存储不同数据的符号,它实际上表示了某个内存单元。存放在内存单元中的数据,可以在程序执行期间进行处理。

PHP 的变量由“$”符号和变量名组成,其中的变量名就是前述的标识符。例如:

```
$str = "php";
```

在 PHP 中,变量名大小写敏感。

除普通变量之外,在 PHP 中还可以定义可变变量。所谓可变变量,就是变量名可以变化的变量。例如:

```
$str = "php";
$ $str = "PHP";
```

其中,“$ $str”就是可变变量,它的名字由变量“$str”的值确定。这里,可变变量“$ $str”就是表示变量“$php”。

由于 PHP 是弱类型语言,变量不需要事先声明,可以直接赋值使用。PHP 中的变量赋值分为传值赋值和引用赋值两种形式。

1) 传值赋值

定义两个变量,赋值并输出,如下。

```
$str = "php";
$php = $str;            //传值赋值
$str = 'java';
echo $php;
```

示例输出变量“$php”的值“php”，不会因为变量“$str”的值变化为“java”而改变。

2）引用赋值

定义两个变量，赋值并输出，代码如下。

```
$str = "php";
$php = &$str;           //引用赋值
$str = 'java';
echo $php;
```

示例输出变量“$php”的值“java”，它随变量“$str”的值的变化而变化。

除变量之外，在 PHP 中还可以使用常量来存储数据。常量用于存储在程序运行过程中始终保持不变的数据。

PHP 中的常量通常使用函数 define()或关键字 const 来定义。例如：

```
define('PI',3.1416);
const PAI = 3.1415;
```

常量一旦被定义，就不能再修改它的值，也不能重新定义。

4. 数据类型

数据类型是具有一组相同属性的数据的统称。PHP 支持三大类的数据类型，分别为标量数据类型、复合数据类型和特殊数据类型。

1）标量数据类型

PHP 的标量数据类型包括 4 种，分别是 boolean（布尔型）、integer（整型）、float（浮点型，也称为 double）和 string（字符串型）。例如：

```
$boolean1 = true;                    //布尔型
$boolean2 = false;                   //布尔型
$boolean3 = (bool) - 10;             //布尔型
$boolean4 = (bool)'';                //布尔型
$int1 = 10;                          //十进制整型
$int2 = 0b10;                        //二进制整型
$int3 = 010;                         //八进制整型
$int4 = 0x10;                        //十六进制整型
$int_min = PHP_INT_MIN;              //最小整型值
$int_max = PHP_INT_MAX;              //最大整型值
$int_size = PHP_INT_SIZE;            //整型值的字长
$float1 = 3.0;                       //浮点型
$float2 = 3.1415926;                 //浮点型
$float3 = 3.14e - 2;                 //浮点型
$float4 = 3.14E + 2;                 //浮点型
$string1 = 'php';                    //字符串
$string2 = "java";                   //字符串
```

```
$string3 = <<<HTML
    <h1>PHP 的数据类型</h1>
    <p>浮点型数据</p>
HTML;                                    //字符串
```

注意 PHP 的数据的不同表达方式。

2）复合数据类型

PHP 的复合数据类型包括三种，分别是 array（数组）、object（对象）和 callable（可调用）。例如：

```
$array1 = array('php','java','c++');     //索引数组
$array2 = ['php','java','c++'];          //索引数组
$array3 = array('课程 1'=>'php', '课程 2'=>'java');   //关联数组
$array4 = $_SERVER;                      //预定义数组
$array5 = range(1,10);                   //索引数组
class Student {
    private $name;
    public function setName($name) {
        $this->name = $name;
    }
    public function getName(){
        return $this->name;
    }
}
$obj1 = new Student;                     //对象
$obj2 = (object)10;                      //对象
$double = function ($param) {
    return $param * 2;
};                                       //对象
$num = range(1, 5);
$result = array_map($double, $num);      //函数的第 1 个参数为 callable 类型
```

PHP 的“可调用”数据类型，就是指的“回调函数”，其值表示回调函数的名称。

3）特殊数据类型

PHP 的特殊数据类型包括两种，分别是 resource（资源）和 NULL（空或不存在）。例如：

```
$file = fopen('doc.txt','r');            //资源类型
$name = NULL;                            //NULL 类型
@var_dump($password);                    //此时的变量 $password 为 NULL 类型
$str = 'php';
settype($str,'NULL');                    //函数调用后变量 $str 为 NULL 类型
```

需要注意的是，PHP 中变量的数据类型，通常由该变量使用的上下文在运行时自动设定，程序设计者一般不会手动去设置。

5. 运算符

与其他高级语言一样，PHP 提供了丰富的运算符，用于处理各种类型的数据。PHP 的运算符分为算术运算符、字符串连接运算符、赋值运算符、比较运算符、逻辑运算符、条件运

算符、自增自减运算符、位运算符，以及错误抑制运算符等。

1）算术运算符

PHP的算术运算符有7种，分别是－（取负）、＋（加法）、－（减法）、*（乘法）、/（除法）、%（取模）、**（乘方）。例如：

```
$a = 10;
$b = 20;
$r1 = - $a;                          //取负
$r2 = $a + $b;                       //加法
$r3 = $a - $b;                       //减法
$r4 = $a * $b;                       //乘法
/* 两个数相除，运算结果通常为浮点型数据；只有当两个操作数均为整型且能整除时，运算结果才为整型数据 */
$r5 = $a / $b;                       //除法，结果为浮点数
$r6 = $a / 5;                        //除法，结果为整数
/* 取模运算符的操作数在运算之前都会自动转换成整数，取模运算符的结果和被除数的符号（正负号）相同 */
$r7 = $a % 3;                        //取模，结果为1
$r8 = $a % -3;                       //取模，结果为1
$r9 = $r1 % 3;                       //取模，结果为-1
$r10 = $r1 % -3;                     //取模，结果为-1
$r11 = $a ** 2;                      //乘方，结果为整数
$r12 = $a ** -0.5;                   //乘方，结果为浮点数
```

与C、C++和Java等高级语言不同的是，PHP的变量不需要预先定义，在程序的任何地方给一个新变量赋值，PHP引擎都会自动定义该变量。

2）字符串连接运算符

在PHP中，使用字符串连接运算符（.）将两个字符串拼接成一个字符串。例如：

```
$str1 = 'Hello ';
$str2 = 'PHP';
$str = $str1. $str2;
```

当然，也可以使用字符串连接函数。

3）赋值运算符

PHP中的赋值运算符，包括基本赋值运算符（＝）和复合赋值运算符（如＋＝等）。复合赋值运算符也称为组合运算符，适合所有的二元运算符。例如：

```
$a = 10;                             //赋值
$b = 20;
$a += $b;                            //加等于，相当于$a = ($a+ $b)
$a -= $b;                            //减等于
$a *= $b;                            //乘等于
$a /= $b;                            //除等于
$a %= $b;                            //模等于
$a **= $b;                           //乘方等于
$a .= $b;                            //连接等于
```

PHP实质上是HTML代码生成器，在实际开发过程中，会频繁使用“.＝”来拼接

HTML 代码，希望读者熟练掌握该运算符的使用方法。

4）比较运算符

PHP 的比较运算符有 9 种，分别是==（等于）、!=（不等于）、<>（不等于）、===（恒等于）、!==（不恒等）、>（大于）、<（小于）、>=（大于或等于）、<=（小于或等于）。例如：

```
$a = 10;
$b = '10';
$r1 = ($a == $b);                    //等于,结果为 true
$r2 = ($a != $b);                    //不等于,结果为 false
$r3 = ($a <> $b);                    //不等于,结果为 false
$r4 = ($a === $b);                   //恒等于,结果为 false
$r5 = ($a !== $b);                   //等于,结果为 true
$r6 = ($a == $b);                    //等于,结果为 true
$r7 = ($a > $b);                     //大于,结果为 false
$r8 = ($a < $b);                     //小于,结果为 false
$r9 = ($a >= $b);                    //大于或等于,结果为 true
$r10 = ($a <= $b);                   //小于或等于,结果为 true
```

在 PHP 中，当两个数据类型不相同的数据进行比较时，会自动将其转换成相同类型的数据后再进行运算。在进行恒等与不恒等运算时，除比较操作数的数值是否相同外，还要判断它们的数据类型是否一样。

5）逻辑运算符

PHP 的逻辑运算符有 4 种，分别是 &&（与）或 and（与）、||（或）或 or（或）、!（非）、xor（异或）。例如：

```
$a = true;
$b = false;
$r1 = $a && $b;                      //逻辑与,结果为 false
$r2 = $a and $b;                     //逻辑与
$r3 = $a || $b;                      //逻辑或,结果为 true
$r4 = $a or $b;                      //逻辑或
$r5 = !$a;                           //逻辑非,结果为 false
$r6 = $a xor $b;                     //逻辑异或,结果为 true
```

在 PHP 中，尽管 &&、||与 and、or 功能相同，但前者的优先级别高于后者；对于“与”运算和“或”运算，在使用时需要注意以下两点。

（1）进行逻辑“与”运算时，如果两个操作数均为表达式，当左边表达式的值为 false 时，右边表达式不会执行，逻辑运算结果为 false。例如：

```
$a = 1;
$r = (1 > 2) && ($a += 1);
```

由于表达式 1 > 2 的值为 false，则表达式 $a+=1 不会被执行，也就是变量 $a 的值仍为 1，此时变量 $r 的值为 false。

（2）进行逻辑“或”运算时，如果两个操作数均为表达式，当左边表达式的值为 true 时，右边表达式不会执行，逻辑运算结果为 true。例如：

```
$a = 1;
```

```
$r = (1 < 2) || ($a += 1);
```

由于表达式 1＜2 的值为 true，则表达式 $a＋＝1 不会被执行，也就是变量 $a 的值仍为 1，此时变量 $r 的值为 true。

6）条件运算符

条件运算符（？：）是 PHP 中唯一的一个三元运算符，其运算结果由第 1 个操作数（或表达式）的值确定。当第 1 个操作数的值为 true 时，运算结果为第 2 个操作数（或表达式）的值；否则，运算结果为第 3 个操作数（或表达式）的值。例如：

```
$a = 10;
$b = 20;
$r = ($a > $b) ? $a : $b;
```

上述代码中，由于表达式“$a＞$b”的值为 false，则变量 $r 的值等于变量 $b 的值。

7）自增自减运算符

PHP 的自增（＋＋）、自减（－－）运算符，也称为递增、递减运算符，是一种特定形式的复合赋值运算符。例如：

```
$a = 10;
$r1 =  $a++;                    //后缀自增，执行后变量 $r1 的值为 10，变量 $a 的值为 11
$r2 = ++ $a;                    //前缀自增，执行后变量 $r1 的值为 11，变量 $a 的值为 11
$r3 =  $a--;                    //后缀自减，执行后变量 $r1 的值为 10，变量 $a 的值为 9
$r4 =  -- $a;                   //前缀自减，执行后变量 $r1 的值为 9，变量 $a 的值为 9
```

在实际开发过程中，一般仅在循环结构中使用这些运算符，强烈不建议让它们参与混合运算。

8）位运算符

位运算符，就是对整型数据的各个位进行操作。这些操作分别为 &（按位与）、|（按位或）、～（取反）、<<（左移）、>>（右移）。例如：

```
$a = 0b1010;
$b = 0b1111;
$r1 =  $a & $b;                 //按位与，执行后变量 $r1 的值为 1010（二进制）或 10（十进制）
$r2 =  $a | $b;                 //按位或，执行后变量 $r2 的值为 1111（二进制）或 15（十进制）
$r3 =  $a ^ $b;                 //按位异或，执行后变量 $r3 的值为 0101（二进制）或 5（十进制）
$r4 =  ~ $a;                    //按位取反
$r5 =  $a << 1;                 //左移，将 $a 的每个位都向左移动一位，即变量 $a 乘以 2
$r6 =  $a >> 1;                 //右移，将 $a 的每个位都向右移动一位，即变量 $a 除以 2
```

PHP 的位运算与其算术运算相比较，其执行效率高，运行速度快。

9）错误抑制运算符

PHP 支持一个错误控制运算符@。当将其放置在一个 PHP 表达式之前，该表达式可能产生的任何错误信息都被忽略掉。例如：

```
$a = 10;
$r = @($a +  $b); //由于变量 $b 没有定义，执行该语句时会出现错误，使用@将其抑制
```

当然，也可以使用 PHP 的错误显示函数来控制页面中的错误显示级别。

10）类型运算符

instanceof 用于确定一个 PHP 变量是否属于某一个类的实例。例如：

```
$a = 10;
$r1 = $a instanceof stdClass; //变量 $r1 的值为 false
$a = (object) $a;
$r2 = $a instanceof stdClass; //变量 $r2 的值为 true
```

这里的 stdClass 为 PHP 预定义的标准类。

6. 数据类型转换

在 PHP 中，对两个变量进行操作时，若其数据类型不相同，则需要对其进行数据类型转换。通常情况下，数据类型转换分为自动类型转换和强制类型转换。

1）自动类型转换

自动类型转换也称为隐式转换，是指当运算需要或与期望的结果类型不匹配时，PHP 会将数据类型进行自动转换。例如：

```
$a = 10;
$b = "20";
$c = "3.14e2";
$r1 = $a + $b;              //变量 $r1 的值为 30，变量 $b 被自动转换为整型数据
$r2 = $c * 2;               //变量 $r2 的值为 624，变量 $c 被自动转换为浮点型数据
$r3 = $a ? $b : null;       //变量 $r3 的值为"20"，变量 $a 被自动转换为布尔型数据
$r4 = $a . $b;              //变量 $r4 的值为"1020"，变量 $a 被自动转换为字符串型数据
```

PHP 的自动类型转换规则与 C、C++ 和 Java 基本相同。

2）强制类型转换

强制类型转换也称为显式类型转换，是指在编写程序时，将一个变量强制转换为与原类型不同的另一种类型。其实现方法有两种，一是通过强制类型转换运算符，另一种就是通过调用 settype()、intval()、floatval()等函数。

PHP 提供了 8 种强制类型转换运算符，它们是 bool 或 boolean、int 或 integer、float 或 double 或 real、string、array、object、binary 和 unset。例如：

```
$a = 10;
$b = "20PHP";
$c = "0";
$r1 = (bool) $a;            //强制转换为布尔型，变量 $r1 的值为 true
$r2 = (boolean) $c;         //强制转换为布尔型，变量 $r2 的值为 false
$r3 = (float) $b;           //强制转换为布尔型，变量 $r3 的值为 true
$r4 = (string) $a;          //强制转换为字符串型，变量 $r4 的值为"10"
$r5 = (array) $a;           //强制转换为数组，变量 $r5 的值为 array(0 => 10)
$r6 = (object) $a;          //强制转换为 stdClass 对象，变量 $r5 的 scalar 属性值为 10
$r7 = (binary) $a;          //强制转换为二进制字符串，变量 $r7 的值为"0b10"
echo bindec( $r7);          //输出十进制整数 2
$r8 = (unset) $a;           //强制转换为 NULL 类型
```

需要注意的是，使用强制类型转换符将变量的类型转换为新类型，只是改变了其参与运算时的类型，变量本身的数据类型并没有发生变化。

使用函数实现数据类型的强制转换。例如：

```
$a = 10;
$b = 3.14;
$c = true;
settype($a,'float');            //将变量$a由整型强制转换为浮点型
var_dump($a);                   //输出变量$a的数据类型为float
$r1 = intval($b);               //取整,变量$r1为整型数据3,变量$b仍为浮点数3.14
$r2 = strval($c);               //转换为字符串,变量$r2为字符串"1",变量$c仍为整数10
$r3 = floatval(20);             //转换为浮点型数据
```

注意区分变量本身的类型与其参与运算时的临时类型之间的差别。在PHP的高版本中,会更加强调变量的数据类型,尤其是在调用自定义函数的时候。在PHP的高版本中,在定义函数时,要求指定形参的数据类型以及返回值的数据类型。

3.1.2 流程控制

流程控制就是确定应用程序中的代码执行流程。例如,程序中的某代码块是否需要执行,是否需要执行多次;是否需要从主流程进入辅流程(函数流程等);是否需要执行其他文件中的代码等。

1. 分支结构

PHP的分支结构分为单分支和多分支。

1) 单分支

使用if条件判断语句,来实现流程的单分支。例如：

```
$a = 10;
$b = 20;
if($a >= $b){
    echo $a;                    //当条件为真时,执行该语句
}
```

2) 多分支

在PHP中,使用if…else…语句、if…elseif…else…语句和switch语句,实现流程的多分支。例如：

```
$a = 10;
$b = 20;
if($a > $b){
    echo $a;                    //当条件为真时,执行该语句
}else{
    echo $b;                    //当条件为假时,执行该语句
}
if($a%2 == 0){
    echo $a.'是偶数';
}elseif($a%3 == 0){
    echo $a.'是奇数,也是3的倍数';
```

```
}else{
echo $a.'是奇数,但不是3的倍数';
}
switch($a%3){
case 1:
    echo '变量$a除以3余1';
    break;                          //跳出switch结构
case 2:
    echo '变量$a除以3余2';
    break;
default:
    echo '变量$a是3的倍数';
    break;
}
```

这些结构与C、C++和Java语言几乎一样。

2. 循环结构

PHP有4种形式的循环结构,分别是while、do…while、for和foreach。例如:

```
$data = range(1, 100);
$i = 0;
$sum = 0;
while ($i<100){
    $sum += $data[$i];
    $i++;
}
$i = 0;
$sum = 0;
do{
    $sum += $data[$i];
    $i++;
}while ($i<100);
for ($i = 0, $sum = 0; $i<100; $i++) {
    $sum += $data[$i];
}
$sum = 0;
foreach ($data as $value) {
    $sum += $value;
}
$sum = 0;
foreach ($data as $key=>$value) {
    $sum += $value;
}
echo $sum;
```

在PHP的循环结构中,常常使用continue语句终止本次循环,而使用break语句终止循环。例如:

```
$data = range(1, 100);
```

```
$i = 1;
$sum = 0;
while ( $i++){
    if( $i % 2) continue;          //若为奇数,终止本次循环
    if( $i > 100) break;           //若变量 $i 的值大于 100,终止循环
    $sum += $data[ $i - 1];
}
echo $sum;                         //输出 1～100 中所有偶数之和
```

循环结构一般用于对集合数据的遍历。

3. 函数

PHP 的函数分为自定义函数和内置函数。

1）自定义函数

自定义 PHP 函数,可以采用如下示例中的两种格式。

```
//PHP 7 以下版本格式,PHP 7 兼容这种格式
function sum( $data) {
    $sum = 0;
    foreach ( $data as $value) {
        $sum += $value;
    }
    return $sum;
}
$data = range(1, 100);             //测试数据
echo sum( $data);                  //输出 1～100 所有自然数之和

//PHP 7 版本新增格式
function dataSum(array $data):float {
    $sum = 0;
    foreach ( $data as $value) {
        $sum += $value;
    }
    return $sum;
}
$data = array(1.2, 2.3, 10, 20.5);   //测试数据
// $data = 2.5;                    //若将该变量作为实参传入 dataSum()函数,会出现数据类型错误
var_dump(dataSum( $data));         //输出的值为 float 数据类型
```

PHP 的高版本会逐渐加强对数据类型的检查,所以建议使用上述新格式来定义 PHP 函数。

2）内置函数

内置函数是预先在 PHP 中定义的函数,可以直接使用。在 PHP 的核心库及扩展库中,预定义了丰富的函数,正是这些函数成就了 PHP 的强大功能。

在 PHP 的核心库中,常用的内置函数主要有以下几种。

(1) 字符串函数：strlen()、strrpos()、str_replace()、substr()等。

(2) 数组函数：is_array()、count()、range()、sort()、rsort()、ksort()、krsort()、array_

search()、array_unique()、array_column()、array_keys()、array_values()、array_unshift()、array_push()、array_shift()、array_pop()、in_array()、array_key_exists()等。

(3) 数学函数：abs()、ceil()、floor()、fmod()、is_nan()、max()、min()、pi()、pow()、sqrt()、round()、rand()等。

(4) 日期时间函数：checkdate()、date()、gettimeofday()、getdate()、time()、mktime()、setlocale()、strftime()、strtotime()、getlastmod()等。

PHP的内置函数的使用方法，请参考PHP手册或其他技术文档。

4. 文件包含

为了提高效率，在项目开发过程中需要特别注重代码的重用性以及功能的模块化。PHP提供了4种在应用程序中包含文件的语句，用于组装被隔离的单独的功能模块。

1) include与include_once

include语句将在其被调用的位置处判断并包含一个文件；include_once与include功能相同，不同的是，它会首先判断是否已经包含该文件。例如：

```
if(file_exists('./inc.php') {
    include './inc.php';
})
```

或

```
if(file_exists('./inc.php') {
    $data = include_once './inc.php';      //被包含的文件中有return语句
})
```

若被包含的文件不存在，使用上述包含语句时，PHP会给出错误信息，程序会继续运行。

2) require与require_once

require和require_once语句的用法与include、include_once相同，区别在于，若被包含的文件不存在，使用require和require_once语句时，PHP会给出错误信息，并终止程序运行。

3.1.3 字符串

字符串是PHP的重要数据类型，也是PHP应用程序中使用最多的一种数据。

1. 字符串的定义

PHP提供了4种定义字符串的方法，分别是单引号('')、双引号("")和定界符(<<<TAG)与(<<<'TAG')。例如：

```
$str1 = 'PHP';                  //单引号
$str2 = "学习{$str1}课程";      //双引号
$str3 = <<<HTML
    <h4>课程名称：$str1</h4>
```

```
    <p>课程简介：$str2</p>
HTML;                                    //定界符:heredoc 结构,定界符可以为任意的 PHP 标识符
$str4 = <<<'HTML'
    <h4>课程名称：$str1</h4>
    <p>课程简介：$str2</p>
HTML;                                    //定界符:nowdoc 结构,开始标识符要用单引号引起来
echo $str1.'<br>';
echo $str2;
echo $str3;
echo $str4;
```

上述例程输出结果：

```
PHP
学习 PHP 课程
课程名称：PHP

课程简介：学习 PHP 课程

课程名称：$str1

课程简介：$str2
```

注意字符串的不同定义方式中,对 PHP 变量及转义字符的解析方法的差异。

2. 字符串操作

对字符串的操作一般使用 PHP 的内置函数来完成。

1）判断变量是否为字符串

使用 PHP 内置函数 is_string()来判断数据是否是字符串。例如：

```
$str = "<a href='https://www.baidu.com'>百度搜索</a>";      //测试字符串
$r1 = is_string($str1);        //变量 $r1 的值为 true
```

2）确定字符串长度

使用 PHP 内置函数 strlen()确定字符串的长度。例如：

```
$r2 = strlen($str);            //$str 为上述测试字符串,变量 $r2 的值为 51
```

3）访问字符串中的字符

直接使用数组下标的方式来访问字符串中的字符,注意其下标值是从 0 开始的。例如：

```
$r3 = $str[0];                 //变量 $r3 的值为'',即空格
```

4）去除字符串两端的空格及特殊字符

使用 PHP 的 trim()等内置函数来去除字符串首尾的空格及特殊字符。例如：

```
$r4 = trim($str);              //去除两端空格
Echo strlen($r4);              //输出 48
$r41 = ltrim($str);            //变量 $r41 的值为 50
$r42 = rtrim($str);            //变量 $r41 的值为 49
```

5）转义字符串中的特殊字符

使用 PHP 的 htmlentities()、nl2br()等内置函数来处理字符串中的一些特殊字符。例如：

```
$r5 = htmlentities($str);
Echo $r5;                          //输出<a href='https://www.baidu.com'>百度搜索</a>
$r51 = htmlspecialchars($str);     //变量$r51的值与$r5相同
$str1 = <<<'HTML'
if($a>$b)
    echo $a;
else
    echo $b;
HTML;
$r52 = nl2br($str1);               //将字符串的\n换行转换为<br>换行
$table = array('<'=>'{','>'=>'}');
$r53 = strtr($str,$table);         //替换字符串中的"<"与">"字符
$r54 = strip_tags($str);           //将HTML转换为纯文本,$r54的值为"百度搜索"
```

为了保证项目数据的安全，必须对用户输入的数据进行无害化处理，上述这些函数是最基本的，必须熟练掌握它们的使用方法。

6）处理字符串大小写

使用 PHP 的 strtoupper()、strtolower()等内置函数将字符串中部分或全部字符进行大小写转换，它们只适用于英文字符。例如：

```
$r6 = strtoupper($str);            //将字符串中的字母全部转换成大写
$r61 = strtolower($str);           //将字符串中的字母全部转换成小写
$r62 = ucfirst('chinese deam');    //将字符串的首字母转换成大写
$r63 = ucwords('chinese deam');    //将字符串的每个单词的首字母转换成大写
```

7）确定字符串中子串的位置

使用 PHP 的 stripos()、strrpos()等内置函数确定子串在字符串中的位置。例如：

```
$r7 = stripos($str,'h');           //查找"h"在字符串中首次出现的位置,不区分大小写
$r71 = strpos($str,'h');           //查找"h"在字符串中首次出现的位置,区分大小写
$r72 = strrpos($str,'h');          //查找"h"在字符串中最后出现的位置,区分大小写
$r72 = strripos($str,'h');         //查找"h"在字符串中最后出现的位置,不区分大小写
```

8）查找并替换字符串中的子串

使用 PHP 的内置函数 str_replace()查找并替换字符串中的子串。例如：

```
$r8 = str_replace(array('<','>'),array('{','}'),$str);
```

如果只对单个的子串进行操作，函数的前面两个参数可以直接使用字符串。

9）将字符串转换成数组

使用 PHP 的 str_split()等内置函数可以将字符串转换为数组。例如：

```
$r9 = str_split('武汉欢迎你',3);       //将字符串转换成数组,每个汉字为一个元素
$r91 = explode(' ','武 汉 欢 迎 你');  //用' '字符串拆分字符串
```

注意函数返回的数组的维数。

10）将字符串解析成多个变量

使用 PHP 的 parse_str()等内置函数可以将字符串解析为多个变量，它们一般用于解析 URL 字符串。例如：

```
$str10 = 'page=1&id=10';
parse_str($str10);
echo $page;                          //输出 1
echo $id;                            //输出 10
$str11 = 'https://www.baidu.com?wd=php';
$r10 = parse_url($str11);            //解析 URL,返回其组成部分
```

注意这些函数返回的关联数组中的键和值。

3. 正则表达式

正则表达式是一个从左到右匹配目标字符串的模式，它表示了具有某些特征的一类字符串。正则表达式的结构与一般的数学表达式相似，由各个元素（操作符）或直接量（字面量）组合而成。下面简单介绍 Perl 风格的正则表达式语法。

1）分隔符

正则表达式需要由分隔符闭合包裹。分隔符可以是任意的非字母数字、非反斜线（\）、非空白 ASCII 字符。经常使用的分隔符有右斜线（/）、hash 符号（#）等。例如：

```
/wuhan China/、#^[^0-9]$ #、+php+、%[a-zA-Z0-9_-]%
```

2）元字符

正则表达式中具有特殊含义的字符，称为元字符。例如：

```
\A、\b、\B、\d、\D、\s、\S、[]、()、{}、$、^、.、\、|、?、*、+、-、\w、\W
```

这些元字符的含义，请查询相关技术文献。

3）修饰符

修饰符是对正则表达式解释的调整。例如，要求匹配时不区分大小写，匹配成功一次后停止等。

常用的修饰符主要有：

```
i、m、s、x、e、A、D、S、U、X、u
```

修饰符直接放在正则表达式的后面。例如：

```
/china/i
```

该正则表达式匹配字符串“china”的任何大小写形式，如 China、CHINA、chiNA 等。

4）转义符

转义符（\）用来取消字符所代表的特殊含义。例如：

```
\?、\-、\.、\+
```

5）汉字的正则表达式

匹配一个汉字：

```
/[\x{4e00}-\x{9fa5}]/u
```

匹配多个汉字：

```
/[\x{4e00}-\x{9fa5}]+/u
```

4. 正则表达式函数

PHP 为使用 Perl 兼容的正则表达式搜索字符串，提供了一些功能函数，主要包括 preg_filter()、preg_match()、preg_grep()、preg_match_all()、preg_quote()、preg_replace()、preg_split()等。

1) preg_filter()

执行一个正则表达式搜索和替换。例如：

```
$pattern = '/\x{6b66}\x{6c49}/u'; //正则表达式为"/武汉/"
$subject = "湖北 武汉 中国";
echo preg_filter( $pattern, 'Wuhan', $subject);
```

上述例程输出：

```
湖北 Wuhan 中国
```

2) preg_grep()

搜索字符串或数组，返回与模式匹配的所有元素组成的数组。例如：

```
$pattern = '/^\x{738b}/u';          //正则表达式为"/^王/u"
$subject = ['王一','李二','张三','王五'];
print_r(preg_grep( $pattern, $subject));
```

上述例程输出：

```
Array ([0] => 王一 [3] => 王五 )
```

3) preg_match()

在字符串中搜索模式，如果存在则返回整数 1，否则返回 0；如果出现错误，则返回 false。例如：

```
$pattern = '/\x{738b}/u';          //正则表达式为"/王/u"
$subject = '王一,李二,张三,王五'; //目标字符串
print_r(preg_match( $pattern, $subject));
```

上述例程输出：

```
1
```

3.1.4 数组

PHP 的数组分为索引数组和关联数组，索引数组的键为数字，关联数组的键为字符串。例如：

```
array('PHP','C++','java');              //索引数组
array('admin'=>['王一','李四'],'edit'=>['王二','赵五']);      //关联数组
```

1. 创建、检查数组

在 PHP 中,数组除了可以用赋值的方式创建外,还可以使用 array()语言结构,以及 range()等函数来创建。例如:

```
$course = array('PHP','C++','java');
$int_num = range(1,100,2);          //用预定义的值范围创建数组,第 3 个参数为步长
```

使用函数 is_array()判断变量是否为数组。例如:

```
if(is_array($course)){
    echo '变量 $course 是数组'
}else{
    echo '变量 $course 不是数组';
}
```

使用函数 in_array()检查数组中是否存在某个值。例如:

```
if(in_array('C',$course)){
    echo '数组中有值为 C 的元素'
}else{
    echo '数组中没有值为 C 的元素';
}
```

使用函数 array_unique()将数组调整为没有重复值元素的数组。例如:

```
$course = array('PHP','C++','java','PHP');
$course = array_unique($course);
print_r($course);                       //输出 Array([0]=>PHP[1]=>C++[2]=>java)
```

使用函数 count()或其别名 sizeof()确定数组元素个数。例如:

```
$users = array('admin'=>['王一','李四'],'edit'=>['王二','赵五']);
print_r(count($users));                 //输出 2
print_r(count($users,COUNT_RECURSIVE)); //输出 6
print_r(Sizeof($users,1));              //输出 6
```

2. 添加和删除数组元素

使用 array_unshift()和 array_push()函数分别在数组头和尾添加一个或多个元素。例如:

```
$array = array(1,2,3);
$r1 = array_unshift($array, 0);   //变量 $r1 的值为 4,新数组长度
array_push($array, 4);            //变量 $r2 的值为 5,新数组长度
print_r($array);                  //输出 Array([0]=>0[1]=>1[2]=>2[3]=>3[4]=>4)
$r3 = array_shift($array);        //变量 $r3 的值为 0,被删除的数组的第一个元素
$r4 = array_pop($array);          //变量 $r4 的值为 4,被删除的数组的最后一个元素
print_r($array);                  //输出 Array([0]=>1[1]=>2[2]=>3)
```

3. 定位数组元素

使用 in_array()函数在数组中搜索一个特定的元素是否存在。例如：

```
$array1 = array('武汉','北京','上海');
if (in_array('武汉', $array1)) {
    echo '武汉位于其中';
}else{
    echo '武汉不在其中';
}
```

使用 array_search()函数在数组中搜索一个指定的元素。例如：

```
$array2 = array('a'=>'武汉','b'=>'北京','c'=>'上海');
$r = array_search('上海', $array2);    //搜索特定元素
print_r($r);                           //输出 c,搜索成功后返回元素的键
```

使用 array_key_exists()函数在数组中搜索一个指定的键是否存在。例如：

```
$r = array_key_exists('b', $array2);   //搜索数组中是否存在键值为 b 的元素
var_dump($r);                          //输出 true
```

使用 array_keys()函数获取数组的所有键。例如：

```
$r = array_keys($array2);              //$array2 为上述测试数组
print_r($r);                           //输出 Array([0]=>a[1]=>b[2]=>c)
```

使用 array_values()函数获取数组的所有值。例如：

```
$r = array_values($array2);
print_r($r);                           //输出 Array([0]=>武汉[1]=>北京[2]=>上海)
```

4. 遍历数组

使用函数 next()、prev()、end()、reset()分别将数组指针移动到后一个元素、前一个元素、最后一个元素和第一个元素，并返回当前数组元素的值。例如：

```
$r1 = next($array2);                   //$array2 为上述 3 种测试数组
echo $r1;                              //输出"北京"
$r2 = prev($array2);
echo $r1;                              //输出"武汉"
$r3 = end($array2);
echo $r1;                              //输出"上海"
$r4 = reset($array2);
echo $r4;                              //输出"武汉"
```

使用 key()和 current()函数分别获取数组当前指针所指元素的键和值。例如：

```
while($key = key($array)){
    echo $key;                         //当前元素的键
    next($array);
}
```

```
echo '<br>';
reset( $array);
while ( $value = current( $array)) {
    echo $value;                          //当前元素的值
    next( $array);
}
```

使用函数 array_walk()将数组中的各个元素传递到用户自定义的函数进行处理。例如：

```
$array = array('username'=>'<script>alert("武汉")</script>');
print_r( $array);                         //页面中会出现一个弹窗,非正常输出

function input(&$value, $key) {
    $value = htmlentities( $value);
}
array_walk( $array, 'input');             //对数组中的每个元素的值进行处理
//下面的语句输出 Array([username]=><script>alert("武汉")</script>)
print_r( $array);
```

5. 数组排序

在 PHP 中,默认情况下按英语指定的规则进行排序。如果需要按另一种语言的约定进行排序,需要使用 setlocale()函数设置本地化环境。

常用的排序函数主要有 array_reverse()、array_flip()、sort()、asort()、rsort()、arsort()、natsort()、natcasesort()、ksort()、krsort()、usort()等。

这些函数的具体用法,请参考相关的技术文档。

6. 合并数组、拆分、连接和分解数组

1) 合并数组

使用函数 array_merge()、array_merge_recursive(),将数组合并到一起,并返回一个联合的数组。例如：

```
$array1 = array("color" => "red", 2, 4);
$array2 = array("a","b","color"=>"green","shape"=>"trapezoid",4);
$result = array_merge( $array1, $array2);
print_r( $result);
```

上述例程输出：

```
Array ( [color] => green [0] => 2 [1] => 4 [2] => a [3] => b [shape] => trapezoid [4] => 4 )
```

联合数组以第 1 个数组 $array1 为基础,在其后面追加第 2 个数组 $array2。具有相同的字符串键名的元素会被覆盖。

```
$ar1 = array("color" => array("favorite" => "red"), 5);
$ar2 = array(10, "color" => array("favorite" => "green", "blue"));
$result = array_merge_recursive( $ar1, $ar2);
echo '<pre>';
```

```
print_r( $ result);
echo '</pre >';
```

上述例程输出：

```
Array
(
    [color] = > Array
        (
            [favorite] = > Array
                (
                    [0] = > red
                    [1] = > green
                )
            [0] = > blue
        )
    [0] = > 5
    [1] = > 10
)
```

当某个输入数组中的某个键已经存在于结果数组中时，函数 array_merge_recursive() 把两个值合并在一起，形成一个新的数组，并以原有的键作为键名。

2）连接数组

使用函数 array_combine()创建一个新数组，以一个数组的值作为其键名，另一个数组的值作为其值。例如：

```
$ keys = array('username','password');
$ values = array('王一','123456');
$ user = array_combine( $ keys, $ values);
print_r( $ user);                          //输出 Array([username] = >王一[password] = > 123456)
```

3）拆分数组

使用函数 array_chunk()将一个数组分割成多个。例如：

```
$ array = range(1, 5);
$ r = array_chunk( $ array, 2);          //将数组进行拆分，每个子数组包含两个元素
echo '< pre >';
print_r( $ r);
echo '</pre >';
```

上述例程输出：

```
Array
(
    [0] = > Array
        (
            [0] = > 1
            [1] = > 2
        )
    [1] = > Array
        (
```

```
            [0] => 3
            [1] => 4
        )
    [2] => Array
        (
            [0] => 5
        )
)
```

若使用如下函数调用形式：

```
$r = array_chunk($array, 2, true);
```

则会保留原数组中的键/值顺序，上述输出结果变成如下形式。

```
Array
(
    [0] => Array
        (
            [0] => 1
            [1] => 2
        )
    [1] => Array
        (
            [2] => 3
            [3] => 4
        )
    [2] => Array
        (
            [4] => 5
        )
)
```

使用函数 array_slice()取出数组中的一部分。例如：

```
$array = range(1, 5);
$r = array_slice($array, 2, 2);
print_r($r);              //输出 Array([0] => 3[1] => 4)
```

使用函数 array_splice()去掉数组中的某一部分，并用其他值取代。例如：

```
$array = range(1, 5);
$r = array_splice($array, 2, 2, array(10,20));
print_r($array);          //输出 Array([0] => 1[1] => 2[2] => 10[3] => 20[4] => 5)
print_r($r);              //输出 Array([0] => 3[1] => 4)
```

4）获取数组的交集与差集

使用函数 array_intersect()、array_diff()、array_intersect_assoc()、array_diff_assoc()获取数组的交集或差集。例如：

```
$array1 = array("a" => "green", "red", "blue");
$array2 = array("b" => "green", "yellow", "red");
$r1 = array_intersect($array1, $array2);
```

```
$r2 = array_diff( $array1, $array2);
$r3 = array_intersect_assoc( $array1, $array2);
$r4 = array_diff_assoc( $array1, $array2);
print_r( $r1);              //输出 Array ([a] => green[0] => red )
print_r( $r2);              //输出 Array ([1] => blue )
print_r( $r3);              //输出 Array ( )
print_r( $r4);              //输出 Array ([a] => green[0] => red[1] => blue )
```

5) 其他数组函数

除了上述介绍的数组函数之外,还有很多不同功能的数组处理函数。例如,从数组中随机取出一个或多个元素的函数 array_rand();对数组中所有值求和的函数 array_sum();随机以数组进行重排的函数 shuffle();获取数组中的指定列的函数 array_column()等,这些函数的使用方法请参考相关的技术文档。

3.1.5 错误处理

在运行 PHP 程序时,难免会出现各种各样的错误,需要对这些错误的显示进行控制。PHP 中的错误级别主要包括 E_ALL、E_COMPILE_ERROR、E_COMPILE_WARNING、E_CORE_ERROR、E_CORE_WARNING、E_ERROR、E_NOTICE、E_WARNING、E_PARSE、E_RECOVERABLE_ERROR、E_STRICT、E_USER_ERROR、E_USER_WARNING、E_USER_NOTICE、E_DEPRECATED、E_USER_DEPRECATED 等。

使用函数 error_reporting()设置应该报告哪一种级别的 PHP 错误。例如:

```
error_reporting(E_NOTICE); //显示提示信息
echo $a;                   //这里会有运行提示信息
include 'doc.txt';         //假设 doc.txt 文件不存在,这里会有 E_WARNING 信息
```

当给函数 error_reporting()传递参数 E_ALL 时,会显示所有的错误信息;当给其传递参数 0 时,则不显示任何级别的错误。

PHP 中错误的显示除了使用上述函数外,还可以通过设置 php.ini 中的相关配置项来进行控制。这些配置项主要有 error_reporting、display_errors、display_startup_errors、log_errors、log_errors_max_len、ignore_repeated_errors、ignore_repeated_source 等。它们的含义请参考相关的技术文档。

3.2 面向过程编程


视频讲解

PHP 程序设计可以采用面向过程方法,也可以采用面向对象方法。使用面向过程的方法进行程序设计,主要是通过 PHP 的内置函数和用户自定义函数来实现程序功能,它涉及数据的输入与检验、会话处理、文件和数据库操作等相关技术。

3.2.1 数据输入

PHP 程序的数据输入方式有多种,包括 URL 查询字符串、Web 表单、文件以及数据库

等。为了保证输入数据的安全,需要对接收到的用户数据进行检验、过滤等相关处理。

1. 查询字符串

查询字符串是附加在 URL 后面的字符串,它一般以“?”符号开头,并以“参数名/值”对的形式传递参数。例如:

```
https://www.example.com?id=1&page=1
```

当有多个参数需要传递时,使用“&”符号进行连接。

通过查询字符串传递的参数,采用 GET 数据传送方式,它存储在 PHP 的预定义数组 $_GET 中,可以直接通过参数名来获取。例如:

```
$id = $_GET['id'];
$page = $_GET['page'];
```

接收到的 GET 数据一般要进行安全处理。例如:

```
$id = isset($_GET['id'])?trim($_GET['id']):false;
if (!filter_var($id,FILTER_VALIDATE_INT)) {
    echo '数据类型错误';
}
```

这里采用 PHP 内置函数对用户输入数据进行处理。

2. Web 表单

Web 表单是 PHP 与用户交互的主要方式,通过表单提交的数据,可以采用 GET 方式或 POST 方式来传递数据,但一般都使用 POST 方式。例如:

```
<form action="" method="post">
<label for="username">用户名:</label>
<input type="text" name="username">
<input type="submit" value="提交">
</form>
```

通过 POST 方式传递的数据,存储在 PHP 的预定义数组 $_POST 中,可以通过参数名来获取。例如:

```
$username = $_POST['username'];
```

为了保证数据的安全,需要对用户输入的数据进行检查与处理。例如:

```
$username = isset($_POST['username'])?trim($_POST['username']):'';
$username = htmlentities($username);
```

这里采用 PHP 内置函数对用户输入数据进行处理。也可以使用自定义函数来检验用户输入的数据是否符合格式、类型等要求。例如:

```
function validateUsername($username) {
    $pattern = '/^[\w\x{4E00}-\x{9FA5}]{2,10}$/u';
    if (!preg_match($pattern, $username)) {
```

```
            return '用户名只允许使用英文字母、数字、下画线和汉字,长度为 2～10';
        }
        return true;
    }
```

采用正则表达式对用户输入数据进行验证。

3. 文件

文件是数据的常用存储方式,可以从文件中批量导入数据作为 PHP 程序的输入。

1) 配置文件

从配置文件中导入数据。例如:

```
//config.ini 配置文件
# 应用设置
appName = example

# 数据库配置
host = localhost
user = root
password = 123456
database = exampleDB
//导入 config.ini 中的数据
$filename = 'inc.ini';
$config = parse_ini_file($filename);
echo '<pre>';
print_r($config);
echo '</pre>';
```

示例代码输出:

```
Array
(
    [appName] => example
    [host] => localhost
    [user] => root
    [password] => 123456
    [database] => exampleDB
)
```

2) 其他文件

将数据以特定的格式存储在文件中,通过读取文件内容,将数据导入 PHP 程序中。具体操作方法见后续的 3.2.3 节。

4. 数据库

数据的持久化,一般都是通过文件和数据库来实现的。因此,将数据库中的数据读取到 PHP 程序中,即可作为程序的输入数据来使用。具体操作方法见后续的 3.2.4 节。

3.2.2 会话管理

HTTP(超文本传输协议)定义了通过万维网(WWW)传输文本、图形、视频和所有其他数据所用的规则。它是一种无状态的协议,也就是说,服务器对每次的处理都与之前或之后的请求无关。

在 PHP 中,解决 HTTP 的无状态问题可以使用两种技术,一种是 Cookie 技术,另一种是 Session 技术。Session 也被称为会话。

1. 配置

在 PHP 的配置文件 php.ini 中,有许多与 Session 或 Cookie 有关的配置项,它们负责确定 PHP 会话处理的功能行为。主要有 session.save_handler、session.save_path、session.use_strict_mode、session.use_cookies、session.use_only_cookies、session.name、session.auto_start、session.cookie_lifetime、session.cookie_path、session.cookie_domain、session.cookie_httponly、session.referer_check 等。

2. Cookie 技术

Cookie 是 Web 应用为了辨别用户而存储在客户端的数据。通过 Cookie,可以跟踪用户与服务器之间的会话状态,通常应用于保存浏览历史、保存用户登录状态等场景。

1) 创建 Cookie

使用函数 setcookie()创建或修改 Cookie。例如:

```
setcookie('username','李木子',time() + 60);          //1 分钟后过期
setcookie('username','李木子',time() + 24 * 60 * 60); //1 天后过期
setcookie('username','李木子',time() - 1);           //立即过期,删除 Cookie
```

2) 使用 Cookie

当浏览器向服务器发送请求时,会携带 GET、POST 和 Cookie 等数据,因此,可以通过 PHP 的预定义数组 $_COOKIE 来获取 Cookie 数据。例如:

```
If(isset( $ _COOKIE['username'])){
 $ username =  $ _COOKIE['username'];                //从 Cookie 中获取 username 数据
}
```

注意:当 PHP 第 1 次通过 setcookie()函数创建 Cookie 时,$_COOKIE 数组中没有这个数据,只有当浏览器再次请求并携带 Cookie 时,才能通过 $_COOKIE 数组获取到该数据。

3. Session 技术

Session 在 Web 应用中称为“会话”,是指用户在访问某个 Web 应用时,从进入应用到离开应用所经过的时间。Session 技术是一种服务器端的技术,它的生命周期从用户访问应用开始,直到断开与应用的连接时结束。当 PHP 启动 Session 时,服务器为每个用户的浏

览器创建一个供其独享的 Session 文件，用于保存用户登录状态、验证码等。

当服务器创建 Session 时，每个 Session 文件都具有一个唯一的会话 ID，用于标识不同的用户。会话 ID 分别保存在客户端和服务器端两个位置。在客户端上通过浏览器 Cookie 来保存，在服务器端以文件的形式保存在指定的 Session 目录中。

1）启动 Session

在使用 Session 之前必须先启动。使用 session_start()函数启动 Session。

2）使用 Session

通过预定义数组 $_SESSION，来添加、读取或修改 Session 中的数据。例如：

```
Session_start();
$_SESSION['username'] = '李木子';          //添加 Session 数据
if(isset($_SESSION['password'])){
$password = $_SESSION['password'];         //获取 Session 数据
}
```

3）删除 Session 数据

可以删除 Session 中的单个数据，也可以销毁全部的 Session 数据。例如：

```
unset($_SESSION['username']);              //删除单个数据
$_SESSION = array();                       //删除所有数据
Session_unset();                           //删除所有数据
Session_destroy();                         //结束会话
```

4）编码和解码 Session 数据

无论采用什么样的存储方式，PHP 都会以标准化格式存储 Session 数据。例如：

```
username|s:9:"李木子";password|s:32:"e10adc3949ba59abbe56e057f20f883e";
```

从示例可以看出，各个会话变量用分号隔开，每个会话变量由 3 部分组成，即名称、长度和值。

PHP 自动处理会话的编码和解码。若需要手工执行这些操作，可以使用 session_encode()和 session_decode()函数。例如：

```
session_start();
$_SESSION['username'] = '李木子';
$_SESSION['password'] = md5('123456');
$session = session_encode();               //会话编码
session_unset();                           //清空所有数据
echo $session;                             //输出格式化的所有会话数据
echo '<br>';
session_decode($session);                  //解码会话数据
echo $_SESSION['username'];                //输出会话数据，如果没有前面的解码，这里会出错
```

Session 数据的编码与解码操作，常常用于需要在数据库中保存会话数据的情形。

3.2.3 文件操作

PHP 只能对位于 Web 服务器上的文件进行相关操作。

1. 获取文件信息

使用 PHP 的内置函数，来获取文件的一些基本信息。这些信息包括文件名、文件大小、文件的最后访问时间、文件的最后改变时间等。

常用的文件信息查询函数，主要有 file_exists()、basename()、dirname()、pathinfo()、realpath()、filesize()、fileatime()、filectime()、filemtime()、fileowner()、fileperms()、stat()等。这些函数的使用方法，请参考相关技术文献。

2. 打开和关闭文件

使用 PHP 的内置函数 fopen()和 fclose()来打开和关闭文件。例如：

```
$filename = __FILE__;
$file = fopen(filename,'r');                    //以只读方式打开文件,变量 $file 为资源类型
fclose( $file);                                 //关闭由 fopen()打开的文件
```

示例代码中的参数“r”为文件模式，表示只读。文件模式还有 r+(读写)、w(只写)、w+(读写)、a(只写)、a+(读写)、b(以二进制模式打开)、t(以文本模式打开)。

3. 读取文件

在 PHP 中，可以用不同的方式读取文件内容。

1) 将文件读入数组

使用 file()函数将文件读入数组。例如：

```
//假设存在如下 config.ini 文件
# 应用设置
appName = example

# 数据库配置
host = localhost
user = root
password = 123456
database = exampleDB

//读取文件
$filename = 'inc.ini';
$file = file( $filename,FILE_IGNORE_NEW_LINES|FILE_SKIP_EMPTY_LINES);
echo '<pre>';
print_r( $file);
echo '</pre>';
```

示例代码输出：

```
Array
(
    [0] => # 应用设置
    [1] => appName = example
    [2] => # 数据库配置
```

```
    [3] => host = localhost
    [4] => user = root
    [5] => password = 123456
    [6] => database = exampleDB
)
```

2）将文件内容读入字符串变量

使用 file_get_contents()函数将文件中的内容读到字符串中。例如：

```
$filename = 'inc.ini';                    //上述 1)中的测试文件
$file = file_get_contents($filename);
$file = nl2br($file);
echo $file;
```

示例代码输出：

```
# 应用设置
appName = example

# 数据库配置
host = localhost
user = root
password = 123456
database = exampleDB
```

还可以使用 fread()和 readfile()函数，将文件中的内容读到字符串中。例如：

```
$filename = 'inc.ini';
$file = fopen($filename, 'r');
$fcontent = fread($file,filesize($filename));   //变量 $fcontent 为字符串
fclose($file);

$size = readfile($filename);              //将文件内容输出到缓存中,返回文件大小
$content = ob_get_contents();             //获取缓存数据
ob_clean();                               //清除缓存
echo '文件大小:'.$size.'<br>';            //输出文件大小
echo $content;                            //输出文件内容
```

3）读取一定数目的字符

使用函数 fgetc()读取文件中的一个字符。例如：

```
$filename = 'inc.ini';
$file = fopen($filename, 'r');
$content = '';
while(!feof($file)){
    $content .= fgetc($file);             //获取文件中的所有字符,拼接成字符串
}
fclose($file);
echo nl2br($content);                     //输出
```

使用函数 fgets()读取文件中的一行(默认大小为 1KB)或指定大小的字符。例如：

```
while(!feof($file)){                      //变量 $file 与上述示例代码中的相同
```

```
    $line = fgets($file,filesize($filename)); //获取一行
    $content .= $line;                        //拼接文件内容
    echo $line.'<br>';                        //输出一行
}
```

还可以使用函数 fgetss()获取文件中的一行,并过滤掉其中的 HTML 和 PHP 标记;使用函数 fscanf()获取文件中的一行,并按照预定义格式解析资源。

4) 读取 CSV 文件内容

CSV 是 Comma-Separated Values 的缩写,即逗号分隔值(有时也称为字符分隔值,因为分隔字符也可以不是逗号),CSV 文件以纯文本形式存储表格数据。Microsoft Excel 和 Access、MySQL 等都能导入和导出 CSV 数据。

在 PHP 中,使用 fgetcsv()函数可以很方便地读取 CSV 文件内容。例如:

```
//文件 users.csv 中用户数据
1514290335,金平,男,环境工程学院,1426_建筑环境与能源应用工程,2015,建环 11501
1514290306,张静,女,环境工程学院,1426_建筑环境与能源应用工程,2015,建环 11501
1514290302,燕子,女,环境工程学院,1426_建筑环境与能源应用工程,2015,建环 11501
1514290207,韩月,女,环境工程学院,1426_建筑环境与能源应用工程,2015,建环 11501
1514290134,李杰,男,环境工程学院,1426_建筑环境与能源应用工程,2015,建环 11501
1514290133,杨宇,男,环境工程学院,1426_建筑环境与能源应用工程,2015,建环 11501
//读取文件中的数据
$filename = "user.csv";
$file = fopen($filename, 'r');
while (!feof($file)) {
    $users[] = fgetcsv($file);          //读取文件中的数据,并以数组形式存储
}
```

4. 写入文件

使用函数 fwrite()将字符串写入指定的文件中。例如:

```
$filename = "user.txt";
$file = fopen($filename, 'a+');         //读写方式打开文件
$string = "李木子,女,计科 11901\n";
fwrite($file, $string);                 //在文件中追加内容,若文件不存在,则新建
fclose($file);
```

为了保持 PHP 变量的结构,在使用函数 fwrite()保存数据时,常常使用 serialize()和 unserialize()函数来对数据进行预处理。例如:

```
$users = array(
    ['name'=>'王一','gender'=>'男','birth'=>'1990-01-01'],
    ['name'=>'李二','gender'=>'男','birth'=>'1991-01-01'],
    ['name'=>'胡莹','gender'=>'女','birth'=>'1992-01-01'],
);                                      //测试数据
$filename = "data.txt";
$file = fopen($filename, 'w+');
$string = serialize($users);            //将 PHP 变量序列化
fwrite($file, $string);                 //将数据写入文件中
```

```
fclose( $ file);
 $ content = file_get_contents( $ filename); //字符串数据
echo '< pre >';
print_r(unserialize( $ content));              //恢复到 PHP 的变量形式,这里为数组
echo '</pre >';
```

除 fwrite()函数之外,还可以使用 file_put_contents()函数将数据写入文件中。

5. 文件上传

在 PHP 中,通过适当的配置,利用上传表单可以将客户端的文件上传到服务器上。

1) 文件上传配置

若通过 PHP 上传文件,首先需要在 PHP 的文件中进行适当的配置。相关的配置项主要有 file_uploads、upload_tmp_dir、upload_max_filesize、max_file_uploads、post_max_size 等。

2) 文件上传表单

上传的文件数据通过 POST 方法传送,表单的 enctype 必须设置为 multipart/form-data。例如:

```
< form action = "<? = $ _SERVER['PHP_SELF']?>" method = "post" enctype = "multipart/form - data">
< input type = "file" name = "upload">
< input type = "submit" value = "上传">
</form >
```

3) 上传文件的接收

在 PHP 中,与上传文件有关的信息存储在预定义数组 $ _FILES 中。例如:

```
<?php
    echo '< pre >';
      print_r( $ _FILES);                    //输出上述 2)中的表单上传的文件信息
      echo '</pre >';
?>
//示例代码输出下面的数组
Array
(
    [upload] = > Array
        (
            [name] = > pic_htmltree.gif
            [type] = > image/gif
            [tmp_name] = > E:\wampserver3.0.6\tmp\php1B66.tmp
            [error] = > 0
            [size] = > 3051
        )
)
```

4) 上传文件的移动

PHP 上传文件存储在服务器上的临时目录中,需要将其移动到指定的目录下。使用函数 copy()或 move_upload_file()实现上传文件的移动。例如:

```
if (is_uploaded_file( $ _FILES['upload']['tmp_name'])) {
copy( $ _FILES['upload']['tmp_name'],
'./upload/'. $ _FILES['upload']['name']);
}
```

将上传文件复制到当前目录下的 upload 子目录中,文件名不变。

```
$ext = explode('.', $ _FILES['upload']['name']);
      move_uploaded_file( $ _FILES['upload']['tmp_name'], './upload/'.time().". $ ext[1]");
```

将上传文件移动到当前目录下的 upload 子目录中,并更改文件名。

3.2.4 数据库操作

PHP 支持多种数据库的操作,下面以 MySQL 为例简单介绍。

1. 使用 mysqli 扩展

PHP 对数据的支持,是通过扩展来实现的。打开 PHP 的配置文件 php.ini,开启相应的数据库扩展。例如:

```
extension = php_mysqli.dll
```

启用该配置,开启 mysqli 扩展。

2. 连接数据库

使用 mysqli_connect()函数连接 MySQL 数据库。例如:

```
$link = @mysqli_connect('localhost','root','123456','exampleDB');
```

该函数其实是函数 mysqli::__construct()的别名。

若连接时出现错误,可以使用 mysqli_connect_errno()和 mysqli_connect_error()函数判断并显示错误信息。

3. 执行查询

可以使用 mysqli_query()函数执行查询。例如:

```
$query = 'show databases';                  //查询语句
$result = mysqli_query( $link, $query);  //执行查询
```

该函数为 mysqli::query()函数别名。

4. 处理结果集

执行查询后得到的结果集的类型,与查询的种类有关。例如,成功执行 select 查询后返回的是 mysqli_result 对象;而执行 insert 后,返回的是受影响的行数。所以,要根据不同的结果类型,进行不同的处理。

下面是执行 select 查询后的示例。

```
$query = 'select * from users';
$result = mysqli_query($link, $query);
mysqli_close($link);                          //断开连接
$data = mysqli_fetch_all($result);            //从结果集中取出所有记录(数组形式)
mysqli_free_result($result);                  //释放资源
```

除了以数组的形式保存数据之外,也可以使用对象形式;数组可以使用索引数组,也可以使用关联数组;可以从结果集中取出所有的数据,也可以只取出一条数据。不同的情况使用不同的函数进行操作,请参考相关的技术文档。


视频讲解

3.3 面向对象编程

目前,Web 应用开发普遍采用面向对象的程序设计方法,该方法允许开发人员把相似的任务组织在类中,这有助于编写出遵守“不重复自己”(Don't Repeat Yourself,DRY)原则,并且易于维护的程序代码。

3.3.1 类与对象

类是用户自定义的数据类型,是用于生成对象的代码模板;对象是根据类中定义的模板所构造的数据,也称为类的实例,它是由类定义的数据类型。

1. 类的定义

在 PHP 中,类由关键字 class 来声明。例如:

```
class Employee {
    private $name;
    public function setName($name){
        $this->name = $name;
    }
    public function printInfo() {
        echo $this->name;
    }
}
```

声明了一个名为 Employee 的类。其中,$name 为属性,setName()和 printInfo()为方法。

PHP 7 开始支持匿名类,使用匿名类可以创建一次性的简单对象。例如:

```
var_dump(new class{
    public function __construct(){
        echo '****';
}});
```

这里的 class{…}为匿名类。

2. 对象的创建

对象是类的实例。通过 new 关键字创建类的对象。例如：

```
$staff = new Employee;                    //Employee 为 1 中定义的类
$staff = new Employee();
```

类的方法中还可以使用一个名为 $this 的内部变量，它表示对象本身。该对象默认存在，不需要创建。

3. 对象的使用

对象创建完成后，就可以用它访问对象的属性和方法了。例如：

```
$staff->setName('李木子');                //$staff 为 2 中定义的对象
$staff->printInfo();                      //输出字符串"李木子"
```

注意对象的属性和方法具有访问权限，外部对象只能访问 public 权限的属性和方法；内部对象可以访问对象本身的所有权限的属性和方法，以及父类的 public、protected 权限的属性和方法。例如：

```
$staff->name;                             //会出现错误
```

4. 对象方法

对象方法，也就是对象中的函数，用于定义对象的行为。在 PHP 的类中定义方法，除了必须使用关键字 function 外，还可以用 public、private、protected、abstract、final 和 static 关键字进行修饰。

1）构造方法

对象的构造方法用于初始化对象，例如给属性赋值、调用方法等。构造方法由 PHP 自动调用，并且具有特定的名称“__construct”。例如：

```
class Employee {
    private $name;
    public function __construct($name = ''){
        $this->name = $name;
    }
    public function printInfo() {
        echo $this->name;
    }
}
$staff = new Employee('李木子');          //自动调用构造方法，初始化 name 属性
$staff->printInfo();                      //输出字符串"李木子"
```

2）析构方法

PHP 的程序执行完毕后，对象会被自动销毁。也可以在类中定义析构方法，来完成对象的清除工作。例如：

```
class Employee {
```

```
    private $name;
    public function __construct($name = ''){
        $this->name = $name;
    }
    public function __destruct(){
        echo get_class($this).'的对象被销毁!';
    }
}
$staff = new Employee('李木子');
```

运行上述程序,会输出字符串“Employee 的对象被销毁!”。

3) 魔术方法

在 PHP 中,将所有以__(双下画线)开头的类方法,称为魔术方法。主要有__construct()、__destruct()、__call()、__callStatic()、__get()、__set()、__isset()、__unset()、__sleep()、__wakeup()、__toString()、__invoke()、__set_state()、__clone()和__debugInfo()等。例如:

```
class Employee {
    private $name;
    public function __set($name, $value){
        $this->$name = $value;
    }
    public function __get($name) {
        return $this->$name;
    }
}
$staff = new Employee;
$staff ->name = '李木子';                    //给不可访问属性赋值,自动调用__set()方法
echo $staff ->name;                          //获取不可访问属性值,自动调用__get()方法
```

若没有在类中定义__set()和__get()魔术方法,用对象 $staff 访问 name 属性会出现错误。使用__set()魔术方法,还可以给对象增加属性。例如:

```
$staff = new Employee;
$staff ->age = 20;                           //添加新属性
echo $staff ->age;                           //输出 20
```

4) 静态方法

在 PHP 中,用 static 关键字定义的方法称为静态方法。静态方法既可以通过类的实例访问,也可以通过类名直接访问。例如:

```
class Employee{
    private static $name = '';
    public static function init($name) {
        self::$name = $name;
    }
    public static function printInfo(){
        echo self::$name;
    }
}
```

```
//通过类名直接调用
Employee::init('李木子');
Employee::printInfo();
//通过类的对象调用
$staff = new Employee;
$staff->init('aa');
$staff->printInfo();
```

5. 对象属性

对象属性,也就是对象中的变量或常量,用于定义对象的静态特性。属性的定义与普通的 PHP 变量或常量基本相同,唯一不同的是,它的前面会有 public、private、protected、var 或 static、const 关键字修饰。例如:

```
class Employee{
    var $name = 'a';              //var 相当于 public,是对低版本 PHP 的兼容,最好不使用
    private $age;
    protected $gender;
    public static $depart = 'b'; //静态属性
    const COMPANY = 'WUHAN';      //常量
}
```

1) 普通属性

普通属性在定义时可以初始化,也可以只声明。如上述示例代码中的 $name、$age 和 $gender。它只能用对象访问。

2) 静态属性

用 static 关键字修饰的属性,称为静态属性。在类的内部,使用 self 来对其进行访问;在类的外部,只能使用类名对其进行访问。例如:

```
echo Employee::$depart;           //输出 b
```

需要注意的是,类的静态属性的任何改变,都会反映到该类的所有实例化对象中。例如:

```
class Visitor {
    private static $visitors = 0;
    public function __construct() {
        self::$visitors++;
    }
    public static function getVisitors() {
        return self::$visitors;
    }
}
$visits1 = new Visitor();
echo Visitor::getVisitors();      //输出 1
echo '<br>';
$visits2 = new Visitor();
echo Visitor::getVisitors();      //输出 2
```

3）常量属性

类中的常量既可以用 const 定义，也可以用 define()函数定义。前者定义的常量需要通过类名来访问；而后者定义的常量可以直接访问。例如：

```
class MathConstant {
    const PI = 3.1415926;
    const E = 2.7182818284;
    public function __construct(){
        define('M_C', 'math_constant');
    }
}
$C = new MathConstant();
echo M_C;                          //可以直接使用常量 M_C
//echo PI;                         //不能直接使用常量 PI
echo MathConstant::PI;             //通过类名访问常量 PI
```

4）属性初始化

在 PHP 中，类属性的初始化可以通过声明时直接赋值来实现，也可以通过类的构造方法来实现。在项目开发过程中更多的是使用构造方法来初始化类的属性。

直接初始化属性示例：

```
class Employee{
    private $name = '';
    private $hello = <<<'EOD'
        Hello PHP
EOD;
}
```

在构造方法中初始化属性示例：

```
class Employee{
    private $name;
    public function __construct( $name = ''){
        $this->name = $name;
    }
}
```

3.3.2 继承与多态

面向对象程序设计有三大特征，即封装性、继承性和多态性。封装性通过访问控制修饰符 public、protected 和 private 来实现，它们可以控制类中成员的可见性；继承是一种在现有类的基础上构建新类的机制；多态则是类中的成员方法在不同的情形下会呈现出不同的特性，也就是同一操作作用于不同的对象，会产生不同的执行结果。

1. 继承

在 PHP 中，可以在一个现有的类的基础上创建一个新的类，新类拥有原有类的全部或部分属性及方法，这种创建新类的机制就是继承。其中，原有的类被称为基类或父类；新类

被称为子类或派生类。

1）继承的实现

在 PHP 中，类的继承通过关键字 extends 来实现。例如：

```
class Employee{
    private $name;
    public function setName($name) {
        if (empty($name)) {
            echo '姓名不能为空!';
        }else{
            $this->name = $name;
        }
    }
    public function getName() {
        return $this->name;
    }
}
class Executive extends Employee{
    public function Notice() {
        echo "通知:明天上午 8:00 例会".'【'.$this->getName().'】';
    }
}
```

代码中的 Executive 类继承于类 Employee，它除了自己特有的 Notice()方法外，还拥有 Employee 类的两个公有成员方法 setName()和 getName()。Executive 类也被称为派生类或子类，Employee 被称为基类或父类。

PHP 不支持多继承，类似 C++中多继承的功能，由接口来实现。PHP 可以多重继承，也就是说，可以用子类再派生出新的类。例如：

```
class CEO extends Executive{          //基类 Executive 声明见上述示例代码
    public function verify() {
        echo '同意'.'【'.$this->getName().'】';
    }
}
$exc = new Executive();
$exc->setName('李木');
echo $exc->notice();
echo '<br>';
$ceo = new CEO;
$ceo->setName('李木子');
echo $ceo->verify();
```

示例代码输出：

```
通知:明天上午 8:00 例会【李木】
同意【李木子】
```

2）继承中的构造方法

在类的继承中，父类和子类中的构造方法的调用，与它们是否存在有关。若父类有构造方法而子类没有，在子类实例化时，自动调用父类的构造方法。例如：

```
class Employee{
    private $name;
    public function __construct($name){ //在 Employee 类中增加构造方法
        $this->setName($name);
    }
    …
}

$exec = new Executive('李木');          //给父类的构造方法传递参数
//$exec = new Executive();              //这里会出错,子类实例化时自动调用父类的构造方法
echo $exec ->getName();                 //输出字符串"李木"
```

若子类定义了构造方法,则自动调用子类自己的构造方法,而不管父类的构造方法是否存在。例如:

```
class Executive extends Employee{
    public function __construct($name){
        echo get_class().'类的构造方法';
    }
    …
}
$exec = new Executive('李木');          //输出字符串"Executive 类的构造方法"
var_dump($exec->getName());             //输出 null
```

若要在子类的构造方法中调用父类的构造方法,可以使用 parent 关键字实现。例如:

```
class Executive extends Employee{
    public function __construct($name){
        parent::__construct($name);     //调用父类的构造方法
        //echo get_class().'类的构造方法';
    }
    …
}
$exc = new Executive('李木');
echo $exc->getName();                   //输出字符串"李木"
```

3) 方法覆盖

方法覆盖也称为方法重写,是指子类和父类中存在同名的方法,子类方法是对父类方法的重新定义。无论是静态方法还是非静态方法都可以被覆盖。在进行方法覆盖操作时,应该注意以下两点。

(1) 方法的参数数量必须一致。

(2) 子类方法的访问级别应该等于或弱于父类中被覆盖的方法的访问级别。

例如:

```
class Employee {
    public function show() {
        self::introduce();              //优先访问父类方法
        static::introduce();            //优先访问子类方法
    }
    public static function introduce() {
```

```
        echo '职员';
    }
}
class Executive extends Employee{
    public function show() {
        parent::show();                    //调用父类的方法
    }
    public static function introduce() {
        echo '经理';
    }
}
$executive = new Executive();
$executive->show();                        //输出字符串"职员经理"
```

在上述示例代码中,子类 Executive 重写了父类 Employee 的 show()和 introduce()方法,当调用 Executive 对象的 show()方法时,调用 Employee 类的 show()方法。从输出结果可以看出,父类 Employee 的 show()方法,分别调用了父类和子类的 introduce()方法。

2. 多态

与 C++不同,PHP 不支持通过函数重载实现多态,它只能通过函数的覆盖来实现。即通过在继承关系中的不同层次的类中,定义同名的方法来实现。例如:

```
class Animal {
   public function shout() {}
}
class Cat extends Animal{
    public function shout() {
        echo '喵喵';
    }
}
class Dog extends Animal{
    public function shout() {
        echo '汪汪';
    }
}
function animalShout(Animal $obj) {
    $obj->shout();                         //同一个方法的调用,会出现不同的输出结果
}
animalShout(new Cat);                      //输出字符串"喵喵"
animalShout(new Dog);                      //输出字符串"汪汪"
```

从示例代码的输出结果可以看出,当传入的对象不同时,调用同一个函数 shout()输出的结果是不一样的。注意函数的形参类型是父类,调用时的实参对象为子类。

3.3.3 辅助函数

在 PHP 中,内置了一些函数用于帮助开发人员管理和使用类库,主要有 class_exits()、get_class()、get_class_methods()、get_class_vars()、get_declared_classes()、get_object_

vars()、get_parent_class()、interface_exists()、is_a()、is_subclass_of()、method_exists()等。

例如：

```
…     //这里是第 3.3.2 节(2)中类声明代码
$cat = new Cat();
echo 'Cat 的父类是:'.get_parent_class($cat).'<br>';
echo 'Animal 类的方法有:'.implode(',',get_class_methods('Animal'));
echo is_a($cat, 'Dog')?'<br>$cat 是 Dog 对象':'<br>$cat 不是 Dog 对象';
echo is_subclass_of($cat, 'Animal')?'<br>该对象是 Animal 子类':'<br>该对象不是 Animal 子类
';
```

上述示例代码输出：

```
Cat 的父类是:Animal
Animal 类的方法有:shout
$cat 不是 Dog 对象
该对象是 Animal 子类
```

除上述辅助函数之外，PHP 还提供了一个重要的魔术方法__autoload()，使用这个方法可以非常轻松地加载类的声明文件，而不需要使用文件包含语句。例如：

```
//假设类的声明文件存放在子目录 class 中,文件名格式为"类名.class.php"
Animal.class.php
Cat.class.php
Dog.class.php
//假设自定义函数库文件 function.php 存放在子目录 lib 中,文件内容如下
function __autoload($class) {
    require_once './class/'.$class.'.class.php';
}
//使用类创建对象并调用方法
require_once './lib/function.php';
$cat = new Cat();                    //使用类时会自动加载 Cat 类的声明文件
$cat->shout();                       //输出"喵喵"
```

3.3.4 高级特性

随着版本的不断更新，PHP 的面向对象功能越来越强大。下面简单介绍几个 PHP 的面向对象高级特性。

1. 对象复制

对象复制，也称为对象克隆，就是创建已有对象的一个副本。在程序设计过程中，一般情况下并不需要完全复制一个对象来获得其中的属性，但有些情况下，采用对象复制会大大提高设计效率。例如，对于同一公司的员工，他们的很多属性是相同的，这时就可以采用复制的方式来得到不同的员工对象。

对象的复制使用关键字 clone 来实现。示例如下。

```
<?php
class Employee {
    private $name;
    private $department;
    private $company;
    public function getDepartment()
    {
        return $this->department;
    }
    public function getCompany()
    {
        return $this->company;
    }
    public function setDepartment($department)
    {
        $this->department = $department;
    }
    public function setCompany($company)
    {
        $this->company = $company;
    }
    public function getName()
    {
        return $this->name;
    }
    public function setName($name)
    {
        $this->name = $name;
    }
}
$e1 = new Employee();                    //员工对象 1
$e1->setCompany('捷晨科技');
$e1->setDepartment('设计部');
$e1->setName('张三');
$e2 = clone $e1;                         //员工对象 2
$e2->setName('李四');
print_r($e1);
echo '<hr>';
print_r($e2);
```

上述代码输出：

```
Employee Object ([name:Employee:private] => 张三 [department:Employee:private] => 设计部
[company:Employee:private] => 捷晨科技 )
Employee Object ([name:Employee:private] => 李四 [department:Employee:private] => 设计部
[company:Employee:private] => 捷晨科技 )
```

这里，员工“张三”和“李四”属于同一公司的同一部门，他们具有很多共同的属性，所以可以通过复制的方式，将这些共同属性从一个员工对象复制到另外一个员工对象。

还可以在对象类中定义一个魔术方法(__clone())来调整对象的克隆行为。例如：

```
class Employee {
    …
    public function __clone(){
        $this->department = '产品部';//克隆后新对象的 department 属性
}
…
}
```

增加__clone()方法后，对于上述示例代码，克隆到的员工“李四”对象的部门属性为“产品部”。注意，该方法在进行对象的克隆操作时自动调用。

2. 抽象类

用 abstract 关键字修饰的类，称为抽象类，它不能被实例化，只能被继承。抽象类可以确保子类的一致性。例如：

```
abstract class Animal {                    //抽象类
    public abstract function shout();      //抽象方法
}
class Cat extends Animal{
    public function shout() {              //必须实现基类中的抽象方法
        echo '喵喵';
    }
}
class Dog extends Animal{
    public function shout() {
        echo '汪汪';
    }
}
// $animal = new Animal;                   //实例化抽象类错误
(new Cat())->shout();                      //输出"喵喵"
```

3. 接口

接口(interface)定义了实现某种服务的一般规范，声明了所需要的函数和常量，但不指定如何实现。之所以不给出实现的细节，是因为不同的实体可以需要用不同的方式来实现公共的方法定义。关键是要建立必须实现的一组原则，只有满足了这些原则才能说实现了这个接口。

接口通过 interface 关键字来定义，就像定义一个标准的类一样，但其中定义的所有方法都是没有函数体的。接口中定义的所有方法都必须是公有访问权限。例如：

```
interface Animal{
    CONST TYPE = '动物';
    public function shout();
}
class Cat implements Animal{
    private $name = 'Cat';
    public function shout()
    {
```

```
        return '"喵喵"';
    }
    public function getName() {
        return $this->name;
    }
}
$cat = new Cat;
echo '这是一种'.CAT::TYPE;
echo '它的叫声为'.$cat->shout();
```

上述代码输出：

```
这是一种动物它的叫声为"喵喵"
```

4. trait 结构

从 PHP 5.4.0 开始，PHP 实现了一种代码重用的方法，这种方法称为 trait。它是为类似 PHP 的单继承语言而准备的一种代码重用机制。通过 trait，开发人员能够在不同结构的类中重用某些方法（函数）。

trait 和 class 相似，但它仅仅是用某种一致的方式来组合功能函数，trait 不能被实例化。trait 结构为传统的继承增加了水平特性的组合，也就是说，应用的几个 class 之间不需要继承，就可以使用某些相同的方法。trait 的功能与 C++中的友元函数有点儿类似。

例如：

```
trait log{                                    //trait 结构
    function print_log($data) {
        $log_file = fopen('log.txt', 'a+');
        $log_txt = 'log_time:'.date('Y-m-d H:i:s')."\t";
        $log_txt .= 'user:'.$data['user']."\t";
        $log_txt .= 'operate:'.$data['operate']."\n";
        fwrite($log_file, $log_txt);
        fclose($log_file);
    }
}
class AdminModel{
    use log;                                  //在类中使用 trait
    public function insert() {
        $log_data = array('user'=>'admin','operate'=>'insert');
        $this->print_log($log_data);//直接使用 trait 中的方法
        //echo '增加新用户操作';
    }
    public function delete() {
        $log_data = array('user'=>'admin','operate'=>'delete');
        $this->print_log($log_data);
        //echo '删除用户操作';
    }
    public function update() {
        $log_data = array('user'=>'admin','operate'=>'update');
        $this->print_log($log_data);
```

```
        //echo '修改用户信息操作';
    }
}
class UserModel{
    use log;                                    //在类中使用 trait
    public function update() {
        $log_data = array('user'=>'wwp','operate'=>'insert');
        $this->print_log($log_data);//直接使用 trait 中的方法
        //echo '修改用户信息操作';
    }
}
$admin = new AdminModel();
$admin->delete();
$user = new UserModel();
$user->update();
```

上述代码输出：

```
log_time:2019-10-26 09:36:45  user:admin  operate:delete
log_time:2019-10-26 09:36:45  user:wwp  operate:insert
```

在 Web 应用开发中，对数据库的操作往往关系到数据的安全，因此，需要对每次操作进行记录。使用 PHP 的 trait，能够非常方便地实现操作日志的生成。

在类中使用 trait 结构，实际上就是将其中的方法导入类中当成类的成员方法。若当前类中存在与 trait 同名的方法，类中的方法会覆盖 trait 中的同名方法。在继承结构中，若基类中存在与 trait 同名的方法，trait 中的方法会覆盖基类中的同名方法。

5. 反射 API

PHP 中的反射 API 就像 Java 中的 java.lang.reflect 包一样。它由一系列可以分析属性、方法和类的内置类组成。它在某些方面和前述的辅助函数功能相似，但更加灵活。

PHP 的反射 API 类主要有 Reflection、ReflectionClass、ReflectionMethod、ReflectionParameter、ReflectionProperty、ReflectionFunction、ReflectionExtension 和 ReflectionException 等。

1）查询类信息

使用 Reflection 类的静态方法 export()查询类的相关信息。例如：

```
class Employee {
    private $name;
    public function __construct($name = ''){
        $this->name = $name;
    }
    public function getName()
    {
        return $this->name;
    }
    public function setName($name)
    {
        $this->name = $name;
    }
```

```
}
$reflector = new ReflectionClass('Employee');
echo '<pre>';
Reflection::export($reflector);          //输出类的详细信息
```

代码输出：

```
Class [ class Employee ] {
  @@ F:\wtu_php_course\test\index.php 2 - 15
  - Constants [0] {
  }
  - Static properties [0] {
  }
  - Static methods [0] {
  }
  - Properties [1] {
    Property [ private $name ]
  }
  - Methods [3] {
    Method [ public method __construct ] {
      @@ F:\wtu_php_course\test\index.php 4 - 6
      - Parameters [1] {
        Parameter #0 [ $name = '' ]
      }
    }
    Method [ public method getName ] {
      @@ F:\wtu_php_course\test\index.php 7 - 10
    }
    Method [ public method setName ] {
      @@ F:\wtu_php_course\test\index.php 11 - 14
      - Parameters [1] {
        Parameter #0 [ $name ]
      }
    }
  }
}
```

从输出结果可以看出，Reflection::export()可以提供类的大量信息，包括属性和方法的访问控制方式、每个方法需要的参数以及每个方法在代码文件中的位置等。

2）检查类

除了使用 Reflection 类的静态方法 export()查询类信息外，还可以使用反射类中的方法，获取特定的类信息。例如：

```
$class_name = $reflector->getName();            //类名
$isUserDefined = $reflector->isUserDefined();   //是否是用户自定义类
$isInternal = $reflector->isInternal();         //是否是内置类
$isInterface = $reflector->isInterface();       //是否是接口
$isAbstract = $reflector->isAbstract();         //是否是抽象类
$isFinal = $reflector->isFinal();               //是否是最终类
$isInstantiable = $reflector->isInstantiable();//是否可以实例化
```

除了以上代码中的方法外，还有很多其他的方法，请参考相关的技术文档。

3）检查方法

使用 ReflectionMethod 类的对象检查类中的方法。例如：

```
$reflector = new ReflectionClass('Employee');        //上述 1)中 Employee 类
$methods = $reflector->getMethods();
foreach ($methods as $m) {
    $method_name[] = $m->name;                       //变量 $m 为 ReflectionMethod 对象
}
echo '<pre>';
print_r($method_name);                               //输出 Employee 类的方法名数组
```

获取到类的某个方法的 ReflectionMethod 对象后，就可以使用 ReflectionMethod 类的成员函数对该方法进行检查了。

ReflectionMethod 类的成员函数主要有 getDeclaringClass()、isAbstract()、isConstructor()、isPublic()、isStatic()等。这些成员函数的含义及使用，请参考相关的技术文档。

4）检查属性

使用 ReflectionProperty 类的成员方法，查询类属性的相关信息。例如：

```
$e = new Employee('李木子');                          //上述 1)中 Employee 类
$reflector = new ReflectionClass($e);
$properties = $reflector->getProperties();
echo $p[0]->getName();                               //输出 name
var_dump($p[0]->isPrivate());                        //输出 true
$p[0]->setAccessible(true);                          //设置可返回属性
echo $p[0]->getValue($e);                            //输出"李木子"
```

5）反射 API 的使用

使用 PHP 的反射 API 可以动态地调用对象中的方法。

例如，假设需要根据用户的请求来调用控制器的不同方法，可以使用如下的示例代码。

```
class Controller{                                    //测试用控制器
    public function index() {
        echo 'index';
    }
    public function login() {
        echo 'login';
    }
    public function register() {
        echo 'register';
    }
}
$action = 'login';                                   //用户请求的控制器方法
$class = new ReflectionClass('Controller');
$controller = $class->newInstance();
if ($class->hasMethod($action)) {
```

```
        //若请求的方法存在,则执行该方法
        $class->getMethod($action)->invoke($controller);
    }else{
        //若请求的方法不存在,则执行控制器的 index 方法
        $class->getMethod('index')->invoke($controller);
    }
```

3.3.5 数据库操作

在 PHP 中,使用面向对象的方法与 MySQL 数据库进行交互,既可以使用 mysqli 扩展,也可以使用 PDO 扩展。

1. 使用 mysqli 扩展

打开 PHP 的配置文件 php.ini,开启 mysqli 数据库扩展。

```
extension = php_mysqli.dll
```

1) 连接数据库

使用面向对象方法连接 MySQL,需要实例化 mysqli 扩展中的 mysqli 类,该类的对象表示了 PHP 和 MySQL 数据库之间的一个连接。

使用 mysqli 类的对象连接 MySQL,可以使用以下两种形式。

(1) 构造方法。其语法格式为:

```
$link = new mysqli($host = null, $username = null, $passwd = null, $dbname = null,
$port = null, $socket = null));
```

(2) 成员方法。其语法格式为:

```
$objLink = new mysqli();
$link = $objLink->connect($host = null, $user = null, $password = null, $database
= null, $port = null, $socket = null);
```

从上述面向对象的两种格式可以看出,不管是使用构造方法,还是使用 connect()成员方法,需要的参数都是一样的,这些参数也都与 mysqli_connect()函数参数相同。其实,mysqli_connect()只是 mysqli 类的 connect 对象方法的别名而已。

例如,假设需要连接本地 MySQL 的 mydatabase 数据库,登录用户名为 root,密码为 123456,则连接代码为:

```
$link = new mysqli('localhost', 'root', '123456', 'mydatabase');
```

或

```
$link = new mysqli();
$link->connect('localhost', 'root', '123456', 'mydatabase');
```

2) 执行查询

使用 mysqli 类成员函数 query()来执行一个 SQL 查询。该函数原型如下。

```
mixed mysqli::query ( string $query [, int $resultmode = MYSQLI_STORE_RESULT ] )
```

其中，参数 query 为必选项，指定查询字符串，也就是 SQL 语句；参数 resultmode 为可选常量，可以是 MYSQLI_USE_RESULT 与 MYSQLI_STORE_RESULT 中的任意一个。前者在需要检索大量数据时使用，后者一般情况下使用，为默认值。

该函数的返回结果分为两种情况，针对成功的 SELECT、SHOW、DESCRIBE 或 EXPLAIN 查询，将返回一个 mysqli_result 类的对象，如果查询执行不正确则返回 FALSE；针对其他成功的查询，则返回 TRUE，如果失败，返回 FALSE。非 FALSE 的返回值意味着查询是合法的，并能够被服务器执行。

例如，需要查询 mydatabase 数据库的数据表信息，可以使用如下的代码。

```
$link = new mysqli('localhost', 'root', '123456', 'mydatabase');
$query = 'show tables';
$result = $link->query($query);
```

3）处理结果集

PHP 查询语句执行成功以后，若返回的是 mysqli_result 结果集对象，则需要使用函数对结果集进行处理。例如，获取结果集中记录的总条数，获取一条或多条记录等。

例如，若需要获取上述 2）代码结果集中的全部记录，并以数组的形式表示，可以使用下面的代码。

```
$data = $result->fetch_all(MYSQLI_ASSOC);
```

或者

```
$data = mysqli_fetch_all($result, MYSQLI_ASSOC);
```

若需要从结果集中获取一行，并以对象的形式表示，则代码为：

```
$row = $result->fetch_object();
```

或者

```
$row = mysqli_fetch_object($result);
```

在 PHP 中，处理结果集的方法还有很多，限于教材篇幅，这里只简单介绍这些，其他方法请参考相关的技术文档。

4）使用准备语句查询

在 PHP 与 MySQL 的交互过程中，通常会重复执行一个查询，但每次使用的参数会有所不同。此时，若采用上述 query()方法及传统的循环机制来实现操作，不仅系统开销大，而且编写代码也不方便。解决重复执行查询带来的问题，可以使用 PHP 对 MySQL 数据库准备(Prepared)语句的扩展支持。

MySQL 数据库准备语句的基本思想是，可以向 MySQL 发送一个需要执行的查询模板，然后再单独发送数据。因此，可以向一个相同的准备语句发送大量的数据，大大提高了查询执行速度。这个特性对批处理的插入操作来说是非常方便的。

下面是使用准备语句实现多条数据插入，并进行数据查询的示例代码。

```
$link = new mysqli('localhost', 'root', '123456', 'demo');
$stmt = $link->stmt_init();
$query = 'insert into tb_user (username, password) values (?, ?)';
$stmt->prepare($query);
$data = array(
    ['username'=>'aa','password'=>'11'],
    ['username'=>'bb','password'=>'22'],
    ['username'=>'cc','password'=>'33']
);
//插入数据
foreach ($data as $v){
    $stmt->bind_param('ss', $v['username'], $v['password']);
    $stmt->execute();
}
//查询数据
$query = 'select * from tb_user';
$stmt->prepare($query);
$stmt->execute();
$stmt->bind_result($id, $username, $password);
//输出查询数据
while ($stmt->fetch()) {
    printf('%s - %s - %s<br>', $id, $username, $password);
}
//释放资源
$stmt->close();
```

关于准备语句应用的其他知识,请参考相关的技术文档。

2. 使用 PDO 扩展

PHP 的 PDO 扩展,就是为 PHP 访问数据库定义的一个轻量级的一致接口。通过这个接口,PHP 可以与各种不同的数据库进行交互。

打开 PHP 的配置文件 php.ini,开启 PDO 扩展。

```
extension = php_pdo_mysql.dll
```

1) 创建 PDO 对象

使用 PDO 与不同数据库之间交互时,使用的操作函数都是相同的,都是 PDO 对象中的成员方法,所以在使用 PDO 与数据库交互之前,首先要创建一个 PDO 对象。

PDO 类位于 PDO.php 文件中,与其相关的类还有 PDOException、PDOStatement 以及 PDORow。

示例代码如下。

```
<?php
    $dsn = "mysql:dbname=mydatabase;host=localhost";
    $user = 'root';
    $pwd = '123456';
    try {
        $pdo = new PDO($dsn, $user, $pwd);
    } catch (PDOException $e) {
        echo '数据库连接失败: '.$e->getMessage();
```

```
        exit();
    }
?>
```

这里创建了一个名为 pdo 的 PDO 对象，同时也创建了一个与 MySQL 数据库 mydatabase 的连接。

2）执行查询

使用 PDO 对象的 query()和 exec()成员函数执行 SQL 查询。示例如下。

```
//数据插入
$statement = "insert into tb_user (username,password) values ('wp','888888')";
$pdo->exec($statement);
//数据查询
$statement = 'select * from tb_user';
$result = $pdo->query($statement);
```

3）获取数据

PDO 的数据获取方法与前面介绍的 mysqli 数据库扩展中使用的方法非常相似，只是获取数据的函数都来自 PDOStatement 类的成员，如 fetch()方法、fetchAll()方法等。

例如，对于上述 2)中的 select 查询结果，可以通过下面的语句获取到全部数据。

```
//获取所有数据
$data = $result->fetchAll();
```

4）使用准备语句

PDO 扩展中对数据库准备语句的支持，是通过 PDOStatement 类的对象来实现的。所以，首先必须创建 PDOStatement 类的对象，然后通过对象调用其成员方法，实现查询模板的导入、参数的绑定、查询的执行以及对结果集的处理等。

示例代码如下。

```
$statement = "insert into tb_user (username,password) values (?,?)";
$stmt = $pdo->prepare($statement);
$data = array('wp','666666');
$result = $stmt->execute($data);
…
$statement = "insert into tb_user (username,password) values (:username, :password)";
$stmt = $pdo->prepare($statement);
$stmt->bindParam(':username', $username);
$stmt->bindParam(':password', $password);
$username = 'wp';
$password = '888888';
$result = $stmt->execute();
```

上述代码用两种方法，完成了单行数据的插入，注意它们数据绑定方式的区别。

3.4 PHP 扩展与应用

视频讲解

在 PHP 项目开发过程中，除了使用 PHP 的内置函数及类外，还经常使用由第三方开发的 PHP 扩展及应用。

3.4.1 PEAR 扩展库

PEAR(PHP Extension and Application Repository)是一个 PHP 扩展及应用的代码仓库,在 PHP 4 和 PHP 5 的应用项目开发中被广泛使用。

若要在项目开发中使用 PEAR,需要先安装它。在 Windows 环境下,PHP 默认情况是没有 PEAR 的,需要运行 go-pear.bat 或 go-pear.phar 文件进行安装。检查自己的 PHP 安装目录中是否存在上述文件,若不存在,到 PEAR 官网(https://pear.php.net/)下载,并将其存放在 PHP 的安装目录下。

安装了 PEAR 包以后,就可以在程序中使用包中的文件了。下面是使用 PEAR DB 包中的类,实现数据库查询的示例代码。

```
<?php
//加载 PEAR 的 DB 包
require 'db.php';
//数据库连接数据
$userName = 'root';
$password = '123456';
$hostName = 'localhost';
$dbName = 'demo';
//连接数据库
$dsn = "mysqli://$userName:$password@$hostName/$dbName";
$dbCon = DB::connect($dsn);
//检测是否有连接错误
if (DB::isError($dbCon)) {
    die ($dbCon->getMessage());
}
//设置查询结果以关联数组形式表示
$dbCon->setFetchMode(DB_FETCHMODE_ASSOC);
//执行查询
$sql = "select * from tb_user";
$result = $dbCon->query($sql);
if (DB::isError($result)) {
    die ($result->getMessage());
}
//输出查询结果
for($i=0;$i<$result->numRows();$i++)
{
    $info = &$result->fetchRow();
    echo "username:".$info['username'].'/ ';
    echo "password:".$info['password']."<br>";
}
//释放资源,断开连接
$result->free();
$dbCon->disconnect();
```

3.4.2 PDF 扩展

在 PHP 中,有很多用于支持 PDF 文档的库,比如 FPDF、TPDF、TCPDF 等。这里使用

较为流行且相对简单的 FPDF 库，该库是一个 PHP 的代码集，可以用包含的方式直接将其导入 PHP 代码文件中，不需要进行任何服务器端配置或支持。FPDF 库的下载地址为 http://www.fpdf.org。

下面是使用 FPDF 1.81 库进行 PDF 文档输出的示例代码。代码中的第 1 列为行号，是为了方便程序中语句的功能说明而设置的。

```
1  <?php
2    define('FPDF_FONTPATH','fpdf/font');
3    require_once 'fpdf/fpdf.php';
4     $pdf = new FPDF('p','mm','A4');
5     $pdf->AddPage();
6     $pdf->Image('images/fpdf_logo.jpg',5,5,30,10);
7     $pdf->SetFont('helvetica','',10);
8     $pdf->SetXY(160, 15);
9     $pdf->Cell(50,20,'by Mashian, Weiwping');
10     $pdf->SetFont('courier','B',16);
11     $pdf->SetXY(70, 10);
12     $pdf->Cell(100,20,'FPDF Example 001');
13     $pdf->Line(5, 30, 200, 30);
14     $pdf->SetFont('times','',12);
15     $pdf->SetXY(10, 35);
16     $content = <<<EOD
17  FPDF 1.81 Reference Manual
18    __construct - constructor
19  AcceptPageBreak - accept or not automatic page break
20  AddFont - add a new font
21  EOD;
22     $pdf->Write(8, $content);
23     $width = $pdf->GetPageWidth();
24     $height = $pdf->GetPageHeight();
25     $pdf->Line(5, $height-20, $width-10, $height-20);
26     $pdf->Output();
27  ?>
```

在上述代码中：第 2 行，定义字体文件路径；第 3 行，包含 FPDF 库文件；第 4 行，创建一个 PDF 文档对象，该 PDF 文档为纵向页面方向、A4 幅面，页面度量单位为 mm；第 5 行，在 PDF 文档中增加一个页面；第 6 行，在页面中添加图像；第 7 行，设置字体为 helvetica 体、普通字体、大小为 10；第 8 行，设置后续文本起始位置；第 9 行，在页面中添加单元格，显示文本；第 10～12 行，在页面中添加另一个单元格，显示文本；第 13 行，绘制一条水平线；第 14～22 行，添加文档正文到页面中；第 23～25 行，为页面添加页脚分隔线，其中获取了文档页面尺寸；第 26 行，将 PDF 文档输出到浏览器中。

使用第三方开发的库，是 PHP 项目开发中经常使用的方法。大家平时可以多收集一些优秀的资源，以提高项目打开的效率，提升项目性能及运行的稳定性。

3.5 本章小结

本章只是简单地总结了一下 PHP 程序设计语言的基本用法，更多详情请参见作者的另外一部教材《PHP Web 程序设计与项目案例开发(微课版)》或其他的技术文档。