第 5 章

PHP 开发技术

学习目标

(1) 理解 PHP 的发展历程、技术演进及安装方法。

(2) 熟练掌握 PHP 的基础语法结构,包括数据类型、运算符、控制结构、数组、函数、面向对象等核心语言要素。

(3) 具备运用 PHP 实现表单验证、会话管理、操作数据库等典型 Web 交互场景的能力。

(4) 了解 PHP 与 Web 安全的相关知识。

5.1 PHP 语言简介

PHP 是开源的服务器端脚本语言,主要用于动态网页开发,由丹麦程序员 Rasmus Lerdorf 于 1993 年首次发布。PHP 从简单的脚本语言发展成功能强大、性能高效的 Web 开发语言,主要经历了以下 7 个关键时间节点和重大事件。

(1) 初期阶段: 1993 年,PHP 最初被创建的目的是用于跟踪网站访问者并生成动态网页内容,最初被称为 PHP Tools(Personal Home Page Tools)。它是一种简单的脚本工具,主要用于在网页中插入动态内容,初期的代码结构相对简单。

(2) PHP 3: 1997 年,Rasmus Lerdorf、Andi Gutmans 和 Zeev Suraski 合作重写 PHP,使其成为更强大、灵活的开发工具,称为 PHP 3。它支持更加复杂的功能,同时加入面向对象编程特性。

(3) PHP 4: 2000 年,PHP 4 发布,该版本引入 Zend 引擎,使得 PHP 的性能大幅提升。同时,进一步强化 PHP 的面向对象编程特性,提供更强大的 HTTP 会话支持,改进数据库扩展。

(4) PHP 5: 2004 年,PHP 5 发布,该版本全面支持面向对象编程,引入类、接口、继承、异常处理等面向对象特性,使得 PHP 能够编写更为复杂和高效的应用程序。另外,PHP 5 增强数据库支持、引入 PDO(PHP Data Objects,PHP 数据对象)扩展,改善 XML 处理,并且实现对 SOAP(Simple Object Access Protocol,简单对象访问协议)的支持。

(5) PHP 6: PHP 6 是计划中的版本,主要目标是引入对 Unicode 的支持,从而解决 PHP 中字符编码的问题,然而,由于技术问题和社区分歧,PHP 6 未正式发布。

(6) PHP 7: 2015 年,PHP 7 发布,通过引入新的 Zend 引擎,PHP 显著提高执行速度,减少内存消耗,在高并发环境下表现优异。PHP 7 还加入许多新特性,如类型声明、联合类

型、匿名类、异常处理等。

(7) PHP 8：2020 年，PHP 8 发布，引入许多新特性和性能改进，包括 JIT 编译器、属性、命名参数、联合类型、增强的类型系统等。

5.2　开发环境的安装

PHP 开发环境包括 PHP 解释器、Web 服务器、数据库及开发工具。常用 Web 服务器包括 Apache 和 Nginx，常用数据库包括 MySQL、MariaDB 和 SQLite，常用开发工具包括集成开发环境(如 PHPStorm、VSCode 等)、代码调试工具(如 Xdebug)和数据库管理工具(如 navcat、phpMyAdmin、DBeaver 等)。搭建开发环境包括以下两种方法。

(1) 手动搭建：手动安装 Apache、PHP 和 MySQL，优点是灵活性高，可根据需求自定义配置，但配置复杂，适合有经验的开发者。

(2) 使用集成开发环境：使用 XAMPP、WAMP、LAMP、PHPStudy 等集成开发环境，可以一键安装所有组件，优点是简单快捷，适合初学者，但不适合生产环境，部分配置较死板。

PHPStudy 是一款专为 PHP 开发设计的集成开发环境工具，通过简单的安装和配置，可以一键搭建 PHP、MySQL、Apache/Nginx 等开发环境，下面介绍 PHPStudy 的基本安装步骤。

(1) 从官网下载相应操作系统的 PHPStudy 安装包。

(2) 运行安装程序，在打开的安装界面中，选择安装路径，然后单击“安装”按钮。在安装过程中，遇到安装提示时，选择“允许”或“继续”，等待安装程序完成安装过程。

(3) 安装完成后，启动 PHPStudy，即可进入 PHPStudy 控制面板，如图 5-1 所示。

图 5-1　PHPStudy 控制面板

(4) 单击“启动”按钮，启动 Apache 和 MySQL 服务，然后在浏览器中访问 http://localhost，PHPStudy 默认主页如图 5-2 所示。

由图 5-2 可知，PHPStudy 安装成功。

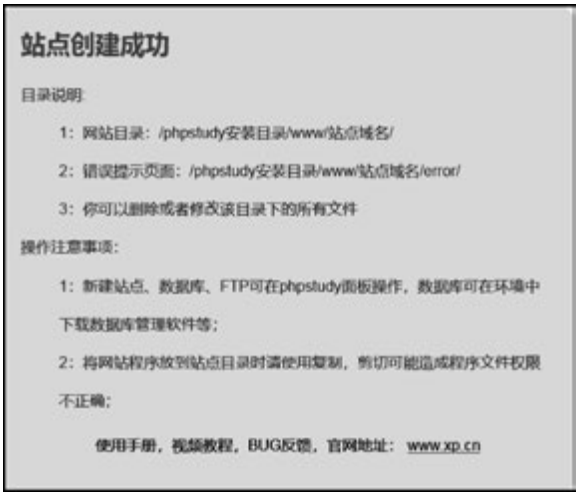

图 5-2　PHPStudy 默认主页

5.3　PHP 的基本语法

PHP 文件包含 HTML 标签和 PHP 脚本代码，代码以<?php 开始，以?>结束，在纯 PHP 文件中，结束标记?>可以省略。在 PHP 中，使用 echo 可以输出一个或多个字符串，使用 print 只能输出一个字符串。

5.3.1　常量和变量

1. 变量

PHP 是一种弱类型语言，定义变量时不需要像在 C 语言中那样显式声明类型，PHP 会根据上下文自动决定变量的数据类型，变量名必须以美元符号 $ 开头，且需符合以下 4 个规则。

(1) 必须以字母或下画线开头。

(2) 后续字符可以是字母、数字或下画线。

(3) 不能包含空格或特殊字符。

(4) 遵循大小写敏感规则。

变量定义的示例代码如下：

```
$name;
$_vul;
$age1;
```

在 PHP 中，变量赋值有简单赋值和引用赋值两种方式，示例代码如下：

```
// 简单赋值
$name = "张三";
$age = 30;
$isStudent = true;
// 引用赋值
```

```
$a = 10;
$b =& $a;                    // $b 是 $a 的引用
$a = 20;
echo $b;                     // 输出:20
```

PHP 提供一系列函数用于检查变量的状态,常用的变量检查函数如表 5-1 所示。

表 5-1　常用的变量检查函数

函　　数	描　　述
isset($var)	检查变量是否已设置且不为 null
empty($var)	检查变量是否为空(0、""、null 等)
is_null($var)	检查变量是否为 null
is_string($var)	检查变量是否为字符串类型
is_int($var)	检查变量是否为整数类型
is_array($var)	检查变量是否为数组类型
is_bool($var)	检查变量是否为布尔类型

2. 变量作用域

变量作用域指的是变量在脚本中的可见性和生命周期,在 PHP 中,根据作用域的不同,变量可以分为局部、全局、静态和超全局 4 种类型。

(1) 局部变量:定义在函数内部的变量,其作用域仅限于函数内,示例代码如下:

```
function test() {
    $localVar = "局部变量";
    echo $localVar;
}
echo $localVar;  // 报错,变量不可用
```

(2) 全局变量:定义在函数外的变量,在函数内默认不可访问,使用 global 关键字可以在函数内部访问全局变量,示例代码如下:

```
$globalVar = "全局变量";
function showGlobal() {
    global $globalVar;          // 使用 global 声明
    echo $globalVar;
}
showGlobal();                   // 输出:全局变量
```

(3) 静态变量:用 static 关键字定义在函数内的变量,在函数调用结束后仍保留其值,示例代码如下:

```
function counter() {
    static $count = 0;          // 静态变量
    $count++;
    echo $count;
}
counter();                      // 输出:1
counter();                      // 输出:2
counter();                      // 输出:3
```

(4) 超全局变量：PHP 提供一些内置的超全局变量，能在脚本的任何地方使用，常用的超全局变量包括以下 8 个。

① $_GET：存储 GET 请求中的数据。

② $_POST：存储 POST 请求中的数据。

③ $_SERVER：存储服务器和执行环境的信息。

④ $_SESSION 和 $_COOKIE：用于会话管理。

⑤ $_FILES：存储上传文件信息。

⑥ $_ENV：存储环境变量。

⑦ $_REQUEST：$_GET、$_POST 和 $_COOKIE 的组合。

⑧ $_GLOBALS：存储所有全局作用域中的所有变量。

3. 常量

在 PHP 中，使用 define()或 const 关键字定义常量。

(1) define()函数：函数定义的常量不需要 $ 符号，函数的语法格式如下：

```
define(string $name, mixed $value, bool $case_insensitive = false)
```

函数参数的意义如下。

① $name：常量的名称。

② $value：常量的值。

③ $case_insensitive：是否不区分大小写，默认为 false。

define()函数的示例代码如下：

```
define("VUL_NAME", "SQL 注入漏洞");
define("VERSION", 1.0);
echo VUL_NAME;              // 输出:SQL 注入漏洞
echo VERSION;               // 输出:1.0
```

(2) const 关键字：const 不能用于动态定义，必须在编译时确定值，示例代码如下：

```
const PI = 3.14159;
const GREETING = "欢迎来到 PHP!";
echo PI;                  // 输出:3.14159
echo GREETING;            // 输出:欢迎来到 PHP!
```

(3) 魔术常量：PHP 提供一些预定义的常量，称为魔术常量，值根据上下文动态改变，常用的魔术常量如表 5-2 所示。

表 5-2 常用的魔术常量

常　量	描　述
__LINE__	文件中的当前行号
__FILE__	当前文件的完整路径和文件名
__DIR__	当前文件所在的目录
__FUNCTION__	当前函数的名称
__CLASS__	当前类的名称

续表

常　　量	描　　述
__METHOD__	当前类的方法名称
__NAMESPACE__	当前命名空间的名称

魔术常量的示例代码如下：

```
echo "当前文件路径:" . __FILE__ . PHP_EOL;          // 输出当前文件路径
echo "当前行号:" . __LINE__ . PHP_EOL;              // 输出当前行号
```

5.3.2　数据类型

在 PHP 中，数据主要分为整型、浮点型、字符串、布尔型、NULL、资源型等类型。

1. 整型

整型是 PHP 中的基本数据类型之一，用于表示不带小数部分的数值。自 PHP 7.0 版本起，整型统一采用 64 位有符号整数格式存储，其取值范围为 -2^{63} 到 $2^{63}-1$。示例代码如下：

```
$num = 100;
echo $num;       // 输出:100
```

2. 浮点型

浮点型用于表示带有小数部分的数字，可以通过标准小数表示法或科学记数法定义。由于计算机内部无法精确表示某些小数，浮点数存在精度误差。例如，表达式 0.1+0.2，结果是 0.30000000000000004，而非 0.3。

在进行浮点数运算时，round()函数用于四舍五入，floor()函数用于向下取整，ceil()函数用于向上取整。由于精度问题，直接使用==判断两个浮点数是否相等会导致误差，可以使用 abs()函数与一个允许的误差范围判断浮点数是否相等。

使用 is_float()或 is_double()函数可以检查一个变量是否为浮点数类型，PHP_FLOAT_MAX 和 PHP_FLOAT_MIN 表示最大浮点数和最小浮点数。浮点数的基本使用方法的示例代码如下：

```
// 浮点数的定义
$num1 = 3.14;                                    // 标准小数表示法
$num2 = 1.23e4;                                  // 科学记数法,表示 12300
// 浮点数的精度问题
$val = 0.1 + 0.2;
echo "0.1 + 0.2 = $val";                         // 输出:0.30000000000000004
// 使用 round()函数四舍五入
$pi = 3.14159;
$roundedPi = round($pi, 2);                      // 保留两位小数
echo "四舍五入的结果:$roundedPi";                 // 输出:3.14
// 使用 floor()和 ceil()函数取整
$number = 5.6;
echo "Floor()函数的取整结果:" . floor($number); // 输出:5
echo "Ceil()函数的取整结果:" . ceil($number);   // 输出:6
// 浮点数的比较
$a = 0.1 + 0.2;
$b = 0.3;
$epsilon = 0.00001;                              // 允许的误差范围
if (abs($a - $b) < $epsilon) {
```

```
    echo "a 和 b 相等。";                          // 输出:a 和 b 相等。
} else {
    echo "a 和 b 不相等。";
}
```

3. 字符串

在 PHP 中,可以用双引号或单引号定义字符串,双引号支持变量插值,示例代码如下:

```
$name = "张三";
echo "你好, $name";                                // 输出:Hello,张三!
```

PHP 提供一系列常用字符串处理函数,实现字符串的创建、连接、操作等功能。

(1) 字符串连接:使用点(.)和赋值(.=)连接符可以连接字符串,也可以使用 implode()函数将数组元素拼接为字符串。示例代码如下:

```
// 使用.连接字符串
$firstName = "张";
$lastName = "三";
echo $firstName . $lastName;                      // 输出: 张三
// 使用.=追加字符串
$greeting = "你好,";
$name = "李四";
$greeting .= $name;
echo $greeting;                                   // 输出: 你好,李四
// 使用 implode()函数拼接字符串
$colors = ["红", "绿", "蓝"];
$colorString = implode("、", $colors);            // 用顿号连接,拼接结果为:"红、绿、蓝"
```

(2) 字符串长度:使用 strlen()函数可以获取字符串的长度,示例代码如下:

```
$str = "Hello, World!";
echo strlen($str);                                // 输出:13
```

(3) 获取子字符串:使用 substr()函数可以获取字符串的一部分,示例代码如下:

```
$str = "Hello, World!";
echo substr($str, 7, 5);                          // 输出:World
```

其中,第一个参数是原字符串,第二个参数是起始位置,第三个参数是子字符串的长度。

(4) 查找字符串:使用 strpos()函数查找子字符串在字符串中的位置,如果找不到子字符串,函数返回 false,示例代码如下:

```
$str = "Hello, World!";
$position = strpos($str, "World");
echo $position;                                   // 输出:7
```

(5) 替换字符串:使用 str_replace()函数可以替换字符串中的元素,示例代码如下:

```
$str = "Hello, World!";
$new_str = str_replace("World", "PHP", $str);
echo $new_str;                                    // 输出:Hello,PHP!
```

(6) 转换大小写：PHP 提供函数来转换字符串的大小写，主要包括以下 3 种。

① strtoupper()：将字符串转换为大写。

② strtolower()：将字符串转换为小写。

③ ucwords()：将字符串的每个单词的首字母转换为大写。

示例代码如下：

```
$str = "hello, world!";
echo strtoupper($str);          // 输出:HELLO, WORLD!
echo strtolower($str);          // 输出:hello, world!
echo ucwords($str);             // 输出:Hello, World!
```

(7) 去除空格：使用 trim()函数可以去除字符串两端的空白字符，示例代码如下：

```
$str = "   Hello, World!   ";
echo trim($str);              // 输出:Hello, World!
```

(8) 字符串拆分：使用 explode()函数可以将字符串按指定的分隔符拆分成数组，示例代码如下：

```
$str = "apple, banana, orange";
$array = explode(",", $str);
print_r($array);          // 输出:Array ( [0] => apple [1] => banana [2] => orange )
```

其中，第一个参数是拆分依据字符串，第二个参数是被拆分的字符串。

4. 布尔型

布尔型用于表示逻辑真(true)或假(false)，用于控制程序流程，如条件判断、循环控制等，示例代码如下：

```
$isActive = true;
echo $isActive;              // 输出:1(true)
```

5. NULL

NULL 是 PHP 中的一个特殊值，表示变量没有值，常用于处理变量初始化、函数返回值和数据验证。NULL 不区分大小写，也可以写成 null 或 Null。使用 is_null()函数或===运算符检测变量是否为 NULL，示例代码如下：

```
$var1 = NULL;
// 检查变量 $var1 是否为 NULL
if (is_null($var1)) {
    echo "\$var1 为 NULL!";
}
// 使用 === 比较
if ($var1 === NULL) {
    echo "\$var1 为 NULL!";
}
```

NULL 与空字符串、0、false 等值在松散比较(==)时相等，严格比较(===)时不相等。

6. 资源

资源是一种特殊的数据类型，用于表示外部资源，如文件、数据库连接、图像画布等。资源类型的特点包括以下 3 种。

(1) 外部引用：资源变量存储的是对外部资源的引用。

(2) 无法直接操作：必须通过专门的函数操作资源。

(3) 有限生命周期：资源在脚本执行结束时自动释放。

常用的资源类型的示例代码如下：

```
// 打开文件资源
$file = fopen('example.txt', 'r');
var_dump($file);            // 输出:resource(5) of type (stream)
// 使用文件资源
$content = fread($file, filesize('example.txt'));
// 关闭资源
fclose($file);
// MySQL 连接(旧版 mysql_* 函数)
$conn = mysql_connect('localhost', 'user', 'pass');
var_dump($conn);            // 输出: resource(6) of type (mysql link)
```

许多传统资源在新版 PHP 中已被对象替代，例如，MySQL 扩展被 PDO 对象替代，文件操作被 SplFileObject 对象替代。

5.3.3 运算符

运算符是用于执行各种操作的符号，用于变量和数据的处理，PHP 的运算符分为以下几类。

1. 算术运算符

算术运算符用于数学运算，包括加、减、乘、除、取模和幂运算，算术运算符如表 5-3 所示。

表 5-3 算术运算符

运 算 符	名 称	用 法	意 义
+	加法	$a+$b	$a 和 $b 的和
-	减法	$a-$b	$a 减 $b
*	乘法	$a* $b	$a 和 $b 的乘积
/	除法	$a/$b	$a 除以 $b
%	取模(余数)	$a%$b	$a 除以 $b 的余数
**	幂运算	$a** $b	$a 的 $b 次幂

算术运算的示例代码如下：

```
$a = 10;
$b = 3;
echo "加法 (\$a + \$b): " . ($a + $b) . "\n";          // 输出:13
echo "减法 (\$a - \$b): " . ($a - $b) . "\n";          // 输出:7
echo "乘法 (\$a * \$b): " . ($a * $b) . "\n";          // 输出:30
echo "除法 (\$a / \$b): " . ($a / $b) . "\n";          // 输出:3.3333333333333
```

```
echo "取模 (\$a % \$b): " . ($a % $b) . "\n";                    // 输出:1
echo "幂运算 (\$a ** \$b): " . ($a ** $b) . "\n";                // 输出:1000
```

2. 赋值运算符

赋值运算符用于将值赋给变量,包括简单赋值和组合赋值两类,赋值运算符如表 5-4 所示。

表 5-4　赋值运算符

运　算　符	名　　称	用　　法	意　　义
=	赋值	$a=$b	赋值
+=	加等于	$a+=$b	$a=$a+$b
-=	减等于	$a-=$b	$a=$a-$b
*=	乘等于	$a*=$b	$a=$a*$b
/=	除等于	$a/=$b	$a=$a/$b
%=	模等于	$a%=$b	$a=$a%$b

赋值运算的示例代码如下:

```
$a = 10;
$b = 3;
echo "赋值 (\$a = \$b): " . ($a = $b) . "\n";              // 输出:3
$a = 10;                                                  // 重置 $a
echo "加法赋值 (\$a += \$b): " . ($a += $b) . "\n";        // 输出:13
$a = 10;                                                  // 重置 $a
echo "减法赋值 (\$a -= \$b): " . ($a -= $b) . "\n";        // 输出:7
$a = 10;                                                  // 重置 $a
echo "乘法赋值 (\$a *= \$b): " . ($a *= $b) . "\n";        // 输出:30
$a = 10;                                                  // 重置 $a
echo "除法赋值 (\$a /= \$b): " . ($a /= $b) . "\n";        // 输出:3.3333333333333
$a = 10;                                                  // 重置 $a
echo "取模赋值 (\$a %= \$b): " . ($a %= $b) . "\n";        // 输出:1
```

3. 三元运算符

?:为三元运算符,用于简化条件判断,语法格式如下所示:

```
$a ? $b : $c
```

如果表达式 $a 为 true,返回 $b,否则返回 $c,示例代码如下:

```
$a = true;
$b = "值为真";
$c = "值为假";
echo "三元运算符 (\$a ? \$b : \$c): " . ($a ? $b : $c) . "\n";  // 输出:值为真
```

4. 比较运算符

比较运算符用于比较两个值,包括等于、全等、不等于、大于、小于等,运算结果返回布尔值,比较运算符如表 5-5 所示。

表 5-5 比较运算符

运 算 符	名 称	用 法	意 义
==	等于	$a == $b	如果 $a 等于 $b 返回 true
===	全等	$a === $b	如果 $a 等于 $b 且类型相同返回 true
!=和<>	不等于	$a != $b	如果 $a 不等于 $b 返回 true
>	大于	$a > $b	如果 $a 大于 $b 返回 true
<	小于	$a < $b	如果 $a 小于 $b 返回 true
>=	大于或等于	$a >= $b	如果 $a 大于或等于 $b 返回 true
<=	小于或等于	$a <= $b	如果 $a 小于或等于 $b 返回 true

比较运算的示例代码如下：

```
$a = 10;
$b = 5;
$c = "10";
echo "等于 (\$a == \$c): " . ($a == $c ? "true" : "false") . "\n";       // 输出:true
echo "全等 (\$a === \$c): " . ($a === $c ? "true" : "false") . "\n";     // 输出:false
echo "不等于 (\$a != \$b): " . ($a != $b ? "true" : "false") . "\n";     // 输出:true
echo "大于 (\$a > \$b): " . ($a > $b ? "true" : "false") . "\n";         // 输出:true
echo "小于 (\$a < \$b): " . ($a < $b ? "true" : "false") . "\n";         // 输出:false
echo "大于或等于 (\$a >= \$b): " . ($a >= $b ? "true" : "false") . "\n"; // 输出:true
echo "小于或等于 (\$a <= \$b): " . ($a <= $b ? "true" : "false") . "\n"; // 输出:false
```

5. 逻辑运算符

逻辑运算符用于逻辑判断，包括与(&& 或 and)、或(|| 或 or)、非(!)等。逻辑运算返回布尔型结果，常用于条件语句中，逻辑运算符如表 5-6 所示。

表 5-6 逻辑运算符

运 算 符	名 称	用 法	意 义
&&	逻辑与	$a&&$b	如果 $a 和 $b 都为 true，返回 true
||	逻辑或	$a\|\|$b	如果 $a 或 $b 为 true，返回 true
!	逻辑非	!$a	如果 $a 为 false，返回 true
and	逻辑与	$aand$b	同 &&，但优先级更低
or	逻辑或	aorb	同||，但优先级更低
xor	逻辑异或	$axor$b	如果 $a 和 $b 不相同，返回 true

逻辑运算的示例代码如下：

```
$a = true;
$b = false;
echo "逻辑与 (\$a && \$b): " . (($a && $b) ? "true" : "false") . "\n";       // 输出:false
echo "逻辑或 (\$a || \$b): " . (($a || $b) ? "true" : "false") . "\n";       // 输出:true
echo "逻辑非 (!\$a): " . ((!$a) ? "true" : "false") . "\n";                  // 输出:false
echo "逻辑与 (优先级低) (\$a and \$b): " . (($a and $b) ? "true" : "false") . "\n";
                                                                             // 输出:false
echo "逻辑或 (优先级低) (\$a or \$b): " . (($a or $b) ? "true" : "false") . "\n";  // 输出:true
echo "逻辑异或 (\$a xor \$b): " . (($a xor $b) ? "true" : "false") . "\n";  // 输出:true
```

6. 位运算符

位运算符用于对整数值的二进制位进行操作，包括按位与、或、异或、取反、左移和右移

运算，位运算符如表 5-7 所示。

表 5-7　位运算符

运　算　符	名　　称	用　　法	意　　义
&	按位与	$a&$b	$a 和 $b 对应二进制位都为 1，则为 1
|	按位或	$a\|$b	$a 和 $b 对应二进制位有一位为 1，则为 1
^	按位异或	$a^$b	$a 和 $b 对应二进制位不同则为 1
~	按位取反	~$a	$a 取反所有二进制位(1 变 0，0 变 1)
<<	左移	$a<<$b	$a 的二进制位左移 $b 位
>>	右移	$a>>$b	$a 的二进制位右移 $b 位

位运算的示例代码如下：

```
$a = 6;  // 二进制:110
$b = 3;  // 二进制:011
echo "按位与 (\$a & \$b): " . ($a & $b) . "\n";          // 输出:2(二进制: 010)
echo "按位或 (\$a | \$b): " . ($a | $b) . "\n";          // 输出:7(二进制: 111)
echo "按位异或 (\$a ^ \$b): " . ($a ^ $b) . "\n";        // 输出:5(二进制: 101)
echo "按位取反 (~\$a): " . (~$a) . "\n";  // 输出: -7(二进制取反，具体值依赖系统字长)
echo "左移位 (\$a << 1): " . ($a << 1) . "\n";           // 输出:12(二进制: 1100)
echo "右移位 (\$a >> 1): " . ($a >> 1) . "\n";           // 输出:3(二进制: 011)
```

7. 错误控制运算符

错误控制运算符@是一个特殊的符号，用于隐藏系统默认的错误信息，示例代码如下：

```
// 文件打开错误
$file = @fopen("non_existent_file.txt", "r");       // 如果文件不存在，不显示错误信息
if (!$file) {
    echo "无法打开文件!";
}
// 除法运算错误
$a = 5;
$b = 0;
$result = @$a / $b;                                 // 如果除数为零，不显示错误信息
if (!$result) {
    echo "除法运算错误!";
}
```

5.3.4　流程控制语句

PHP 提供多种控制结构来控制程序的执行流程，包括条件语句、循环语句和跳转语句。

1. 条件语句

在 PHP 中，常用的条件语句包括 if 语句、if…else 语句、if…elseif…else 语句和 switch 语句。

(1) if 语句：用于根据条件执行代码块，如果条件为 true，则执行代码块，否则跳过代码块，继续执行后续代码，示例代码如下：

```
$a = 5;
if ($a > 0) {
    echo "a 是正数。";
}
```

(2) if…else 语句：在条件为 true 时执行 if 代码块，否则执行 else 代码块，示例代码如下：

```
$a = -3;
if ($a > 0) {
    echo "a是正数。";
} else {
    echo "a是负数。";
}
```

(3) if…elseif…else 语句：用于多个条件的判断，若条件为 false，则继续判断后面的条件，否则执行当前代码块，执行完成后，继续执行整个语句块后的代码，示例代码如下：

```
$a = 0;
if ($a > 0) {
    echo "a是正数。";
} elseif ($a < 0) {
    echo "a是负数。";
} else {
    echo "a是零。";
}
```

(4) switch 语句：根据变量的不同值执行不同的代码块，适用于多重条件判断，示例代码如下：

```
$day = 3;
switch ($day) {
    case 1:
        echo "星期一";
        break;
    case 2:
        echo "星期二";
        break;
    case 3:
        echo "星期三";
        break;
    default:
        echo "其他日期";
}
```

2. 循环语句

循环语句允许代码块重复执行，直至指定条件不满足为止。PHP 提供 for 循环、while 循环、do…while 循环和 foreach 循环 4 种循环结构。

(1) for 循环：用于已知次数的重复执行代码块，示例代码如下：

```
for ($i = 1; $i <= 5; $i++) {
    echo "第$i次循环。";
}
```

（2）while 循环：用于在条件为真时重复执行代码块，示例代码如下：

```
$i = 1;
while ( $i <= 5) {
    echo "第 $i 次循环。";
    $i++;
}
```

（3）do…while 循环：至少执行一次代码块，然后根据条件判断是否继续执行，示例代码如下：

```
$i = 1;
do {
    echo "第 $i 次循环。";
    $i++;
} while ( $i <= 5);
```

（4）foreach 循环：用于遍历数组或对象，示例代码如下：

```
$array = [1, 2, 3, 4, 5];
foreach ( $array as $value) {
    echo "数组值：$value。";
}
```

3. 跳转语句

PHP 提供 3 种跳转语句，break 跳出当前的循环或 switch 语句，continue 跳过当前循环的剩余部分，直接进行下一次循环，goto 跳转到脚本中的指定标签，示例代码如下：

```
// break 语句
for ( $i = 1; $i <= 5; $i++) {
    if ( $i == 3) {
        break;              // 跳出循环
    }
    echo "第 $i 次循环。";
}
// continue 语句
for ( $i = 1; $i <= 5; $i++) {
    if ( $i == 3) {
        continue;           // 跳过第 3 次循环
    }
    echo "第 $i 次循环。";
}
// goto 语句
$i = 1;
start:
if ( $i <= 5) {
    echo "第 $i 次循环。";
    $i++;
    goto start;             // 跳转到 start 标签
}
```

5.3.5 数组

1. 数组类型

数组是用于存储多个值的变量，在PHP中，数组分为索引数组（数字索引）和关联数组（键值对）两种类型。

（1）索引数组是用数字索引访问的数组，默认从0开始，也可以指定索引，示例代码如下：

```
// 自动索引
$vuls = ["SQL注入漏洞", "RCE漏洞", "文件上传漏洞"];
echo $vuls[0];                    // 输出:SQL注入漏洞
// 指定索引
$vuls[3] = "XSS漏洞";
echo $vuls[3];                    // 输出:XSS漏洞
```

（2）关联数组使用命名键来访问数组元素，示例代码如下：

```
$person = [
    "name" => "张三",
    "age" => 25,
    "city" => "河南"
];
echo $person["name"];             // 输出:张三
echo $person["age"];              // 输出:25
```

2. 多维数组

多维数组是数组中的数组，用于表示更复杂的数据结构，示例代码如下：

```
$contacts = [
    [
        "name" => "张三",
        "email" => "zs@163.com"
    ],
    [
        "name" => "李四",
        "email" => "ls@163.com"
    ]
];
echo $contacts[0]["name"];        // 输出:张三
echo $contacts[1]["email"];       // 输出:zs@163.com
```

3. 数组操作函数

PHP提供操作数组的内置函数，用于数组创建和初始化、添加和删除数组元素、数组排序等，常用数组函数包括以下13个。

（1）array()函数：用于创建数组，示例代码如下：

```
$arr = array(1, 2, 3);            // 创建数组[1, 2, 3]
```

(2) range()函数：用于创建一个包含指定范围的数组，示例代码如下：

```
$arr = range(1, 5);                          // 生成数组[1, 2, 3, 4, 5]
```

(3) array_push()函数：用于向数组末尾添加一个或多个元素，示例代码如下：

```
$arr = [1, 2];
array_push($arr, 3, 4);                      // 添加 3 和 4
```

(4) array_pop()函数：删除数组末尾的元素，并返回该元素，示例代码如下：

```
$arr = [1, 2, 3];
$last = array_pop($arr);                     // 删除 3
```

(5) sort()函数：用于对数组进行升序排序，示例代码如下：

```
$arr = [3, 1, 2];
sort($arr);                                  // 排序结果为[1, 2, 3]
```

(6) rsort()函数：用于对数组进行降序排序，示例代码如下：

```
$arr = [3, 1, 2];
rsort($arr);                                 // 排序结果为[3, 2, 1]
```

(7) asort()函数：用于根据数组的值对数组进行升序排序，保持键值关系，示例代码如下：

```
$arr = ["b" => 2, "a" => 1];
asort($arr);                                 // 排序结果为["a" => 1, "b" => 2]
```

(8) ksort()函数：用于根据数组的键对数组进行升序排序，示例代码如下：

```
$arr = ["b" => 2, "a" => 1];
ksort($arr);                                 // 排序结果为["a" => 1, "b" => 2]
```

(9) in_array()函数：用于检查数组中是否存在某个值，示例代码如下：

```
$arr = [1, 2, 3];
$exists = in_array(2, $arr);                 // 返回 true
```

(10) array_search()函数：用于查找数组中某个值的键，示例代码如下：

```
$arr = ["a" => "SQL 注入漏洞", "b" => "RCE 漏洞"];
$key = array_search("SQL 注入漏洞", $arr);  // 返回"a"
```

(11) count()函数：用于计算数组中元素的数量，示例代码如下：

```
$arr = [1, 2, 3];
$count = count($arr);                        // 结果为 3
```

(12) array_keys()函数：用于获取数组中的所有键，示例代码如下：

```
$arr = ["a" => "SQL 注入漏洞", "b" => "RCE 漏洞"];
$keys = array_keys( $arr);      // 结果为["a", "b"]
```

(13) array_values()函数：用于获取数组中的所有值，示例代码如下：

```
$arr = ["a" => "SQL 注入漏洞", "b" => "RCE 漏洞"];
$values = array_values( $arr);  // 结果为["SQL 注入漏洞", "RCE 漏洞"]
```

5.3.6 函数

函数用于封装特定代码逻辑的模块，方便复用和提高代码的可读性，PHP 函数分为用户自定义函数和内置函数两种类型。

1. 用户自定义函数

(1) 函数定义：在 PHP 中，使用 function 关键字定义函数，语法格式如下：

```
function functionName( $param1, $param2, ...) { 函数体 }。
```

示例代码如下：

```
function add(num1, num2) {
    return num1 + num2;
}
```

(2) 函数调用：调用函数时需要使用函数名，并传入相应的参数值，示例代码如下：

```
add(1, 2);
```

2. 内置函数

PHP 提供大量内置函数，用于字符串处理、文件操作、日期时间处理等，常用的内置函数包括以下 4 种。

(1) 字符串函数：用于操作和处理字符串，包括获取长度、查找、替换等操作，常用的字符串处理函数包括以下 4 个。

① strlen($string)：获取字符串的长度。

② strpos($haystack, $needle)：查找子字符串首次出现的位置。

③ str_replace($search, $replace, $subject)：替换字符串。

④ trim($string)：去除字符串的首尾空格。

示例代码如下：

```
$string = "  Hello PHP";
echo strlen( $string);                      // 输出:12
echo strpos("Hello World", "World");        // 输出:6
echo str_replace("PHP", "World", $string); // 输出:"  Hello World"
echo trim( $string);                        // 输出:"Hello PHP"
```

(2) 日期与时间函数：获取、格式化当前时间或将时间字符串转换为时间戳，用于日志

记录、时间计算和格式化输出，常用日期与时间函数包括以下 3 个。

① date($format)：格式化日期和时间。

② time()：获取当前时间戳。

③ strtotime($time， $now)：将时间字符串转换为时间戳。

示例代码如下：

```
echo date("Y-m-d H:i:s");          // 输出当前日期和时间，如 2025-01-23 12:34:56
echo time();                       // 输出当前时间戳，如 1674450896
echo strtotime("2025-01-01");      // 输出时间戳，如 1735660800
```

(3) 文件操作函数：读取、写入、删除文件或打开流资源，用于文件内容的管理，常用的文件操作函数包括如下 4 个。

① file_get_contents($filename)：读取文件内容。

② file_put_contents($filename， $data)：写入文件内容。

③ fopen($filename， $mode)：打开文件。

④ unlink($filename)：删除文件。

示例代码如下：

```
file_put_contents("test.txt", "Hello World");          // 写入内容到文件
echo file_get_contents("test.txt");                    // 输出:Hello World
$handle = fopen("test.txt", "r");                      // 以只读模式打开文件
unlink("test.txt");                                    // 删除文件
```

(4) JSON 函数：用于处理 JSON 格式的数据，包括将数据编码为 JSON 字符串或从 JSON 字符串解析为 PHP 数组或对象，常用的 JSON 处理函数包括如下两个。

① json_encode()：数据编码为 JSON 字符串。

② json_decode()：JSON 字符串解析为 PHP 数组或对象。

示例代码如下：

```
$data = ["name" => "张三", "age" => 25];
$json = json_encode( $data);              // 输出:{"name":"张三", "age":25}
$decoded = json_decode( $json, true);     // 将 JSON 对象转换为数组
echo $decoded['name'];                    // 输出:张三
```

5.3.7　面向对象

面向对象编程是一种强大的编程范式，是将数据和操作数据的方法组合成对象，通过对象之间的交互来构建程序的编程方法。

1. 类和对象

类是面向对象编程的基础构建块，定义一类对象的属性和行为，可以看作是对象的模板。对象是类的实例化结果，拥有类定义的属性和方法，是程序运行时实际存在的实体。

在 PHP 中，使用 class 关键字定义类，通过 new 关键字实例化对象。类包含成员变量(属性)和成员函数(方法)，通过->运算符访问对象的属性和方法，示例代码如下：

```
class Person {
    private $name;
    private $age;
    // Setter 方法,用于获取对象的属性值,以 get 开头,后跟属性名
    public function setName($name) {
        $this->name = $name;
    }
    public function setAge($age) {
            $this->age = $age;
    }
    // Getter 方法,用于设置对象的属性值,以 set 开头,后跟属性名
    public function getName() {
        return $this->name;
    }
    public function getAge() {
        return $this->age;
    }
}
// 使用成员方法赋值
$person = new Person();
$person->setName("张三");
$person->setAge(25);
echo "姓名:" . $person->getName() . ",年龄:" . $person->getAge() . "!";
?>
```

2. 构造函数和析构函数

构造函数是一种特殊方法,当创建对象时自动调用。在 PHP 中,构造函数命名为__construct(),用于初始化对象属性或执行必要的设置操作。析构函数是另一个特殊方法,当对象被销毁时自动调用,命名为__destruct(),用于执行清理操作。示例代码如下:

```
class Person {
    private $name;
    private $age;
    // 构造方法,用于初始化赋值
    public function __construct($name, $age) {
        $this->name = $name;
        $this->age = $age;
    }
    // Getter 方法
    public function getName() {
        return $this->name;
    }
    public function getAge() {
        return $this->age;
    }
    // 析构方法,用于对象销毁时调用
    public function __destruct() {
        echo "Person 对象已销毁:{$this->name}\n";
    }
}
// 使用构造方法直接赋值
$person = new Person("张三", 25);
```

```
echo "姓名:" . $person->getName() . ",年龄:" . $person->getAge() . "!";
// 手动销毁
unset($person);
```

3. 对象序列化

序列化和反序列化是将对象转换为可存储或传输的数据格式,以及从这种格式重建对象的过程,示例代码如下:

```
$serialized = serialize($person);                    // 序列化对象
$unserialized = unserialize($serialized);            // 反序列化
```

PHP 的对象系统提供完整的面向对象编程功能,包括封装、继承、多态等核心特性。

5.4　Web 交互

PHP 与 HTML、CSS、JavaScript 相结合,通过处理表单数据、SESSION 会话管理、文件上传、文件包含等功能,实现高效的 Web 交互。

1. 处理表单数据

前端通过 HTML 表单提交数据,后端通过 PHP 动态脚本处理,示例代码如下:

```
// HTML 表单
<form action="process.php" method="post">
    <input type="text" name="username" placeholder="请输入姓名">
    <input type="submit" value="Submit">
</form>
// PHP 处理前端提交数据的脚本
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // 通过超全局变量 $_POST 获取数据
    $username = htmlspecialchars($_POST['username']);
    echo "Hello, $username!";
}
```

2. SESSION 会话管理

使用超全局变量 $_SESSION 保存用户的状态和信息,用于登录状态保持、购物车等场景,示例代码如下:

```
session_start();              // 开启 SESSION 会话
$_SESSION['username'] = "张三";
echo "欢迎, " . $_SESSION['username'];
session_destroy();            //销毁 SESSION 会话
```

3. Cookie 的使用

使用 setcookie()函数保存 Cookie,超全局变量 $_COOKIE 读取 Cookie,示例代码如下:

```
// 设置 Cookie
setcookie("username", "张三", time() + (86400 * 7), "/");  // 设置的 cookie 有效期为 7 天
```

```
// 读取 Cookie
if (isset( $ _COOKIE['username'])) {
    echo "欢迎回来," . $ _COOKIE['username'];
} else {
    echo "欢迎,游客!";
}
```

4. 处理文件上传

使用 $ _FILES 可处理前端上传的文件,将文件保存到服务器。当文件上传时,$ _FILES 将存储一个关联数组,包含上传文件的详细信息,其中 name 存储上传文件的原始名称,type 存储文件的 MIME 类型,tmp_name 存储文件在服务器的临时存储路径,size 存储文件大小。使用 move_uploaded_file()函数可将文件复制到服务器指定的位置。示例代码如下:

```
// HTML 表单
< form action = "upload. php" method = "post" enctype = "multipart/form - data">
    < input type = "file" name = "fileToUpload">
    < input type = "submit" value = "Upload">
</form >
// PHP 处理前端上传文件的脚本
 $ target_dir  =  "uploads/";
 $ target_file  =  $ target_dir . basename( $ _FILES["fileToUpload"]["name"]);
if (move_uploaded_file( $ _FILES["fileToUpload"]["tmp_name"], $ target_file)) {
    echo "文件" . basename( $ _FILES["fileToUpload"]["name"]) . "上传成功!";
} else {
    echo "文件上传错误!";
}
```

5. 文件包含

文本包含是在 PHP 脚本中动态地包含并执行其他文件中的代码,用于模块化代码、加载配置文件等。PHP 通过 include、include_once、require 和 require_once 语言结构实现文件包含功能。

(1) include:包含并执行指定的文件,如果包含的文件不存在,include 将产生一个警告,但脚本会继续执行,适用于包含非必需的文件。

(2) include_once:与 include 类似,但会检查文件是否已经被包含。如果已经包含过,则不会再次包含,避免重复包含同一个文件,防止函数或类重复定义的错误。

(3) require:包含并执行指定的文件,如果包含的文件不存在,require 将产生一个致命错误,并停止脚本执行,适合用于包含必需的文件,如核心库文件。

(4) require_once:与 require 类似,但会检查文件是否已经被包含过。如果已经包含过,则不会再次包含。

下面通过案例,展示如何实现文件包含功能。案例模拟了一个简单的网站,通过文件包含技术将头部、内容和尾部分离到不同的文件中,从而实现代码复用和模块化开发。

1. 头部文件 header. php

文件包含网站的头部内容,例如,HTML 的< head >部分和导航栏,代码如下:

```
<! DOCTYPE html >
< html lang = "en">
```

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>文件包含示例</title>
    <style>
        body {
            font-family: Arial, sans-serif;
        }
        header {
            background: #333;
            color: #fff;
            padding: 10px;
            text-align: center;
        }
        nav a {
            color: #fff;
            margin: 0 10px;
            text-decoration: none;
        }
    </style>
</head>
<body>
    <header>
        <h1>欢迎光临!</h1>
        <nav>
            <a href="index.php">Home</a>
            <a href="#">About</a>
            <a href="#">Contact</a>
        </nav>
    </header>
```

2. 尾部文件 footer.php

文件包含网站的尾部内容,如版权信息,代码如下:

```
    <footer style="text-align: center; margin-top: 20px; padding: 10px; background: #333;
color: #fff;">
        <p>&copy; 2023 My Website. All rights reserved.</p>
    </footer>
</body>
</html>
```

3. 主文件 index.php

文件是网站的主页面,通过 include 将头部和尾部文件包含进来,代码如下:

```
<?php
// 包含头部文件
include 'header.php';
?>
    <main style="padding: 20px;">
        <h2>Home Page</h2>
        <p>网页核心内容.</p>
    </main>
```

```
<?php
// 包含尾部文件
include 'footer.php';
?>
```

5.5 数据库操作

操作数据库用于实现动态数据的存储、读取、更新和删除，PHP 支持多种数据库管理系统，如 MySQL、PostgreSQL、SQLite 等。

数据库操作主要包括连接数据库、执行 SQL 查询（如 SELECT、INSERT、UPDATE、DELETE 等）及处理查询结果，通过这些操作，可以实现如用户认证、数据展示、后端管理等功能。PHP 提供 MySQLi 和 PDO 两种数据库操作方法。

1. 使用 MySQLi 操作 MySQL 数据库

MySQLi 是 MySQL 的专用扩展，支持面向过程和面向对象的编写方法，使用 MySQLi 操作数据库的基本步骤如下。

（1）连接数据库：首先设置连接参数，包括主机名、用户名、密码和数据库名称等，然后使用 new mysqli()（面向对象）或 mysqli_connect()（面向过程）创建连接，最后，检查连接是否成功，若失败，使用 connect_error()输出错误信息。以面向对象为例，示例代码如下：

```
$servername = "localhost";
$username = "root";
$password = "root";
$database = "my_database";
// 创建数据库连接
$conn = new mysqli($servername, $username, $password, $database);
// 检查连接
if ($conn->connect_error) {
    die("连接失败:" . $conn->connect_error);
}
echo "连接成功!";
```

（2）执行 SQL 语句：首先，编写 SQL 语句，功能包括查询数据、插入数据、修改数据和删除数据，然后，使用 $conn->query($sql)执行查询，对于 SELECT 查询，使用 fetch_assoc()、fetch_array()、fetch_row()、fetch_all()等方法遍历结果集，最后，检查查询是否成功，失败时可通过 $conn-> error 输出错误信息。

① 查询数据的示例代码如下：

```
$sql = "SELECT id, name FROM users";
$result = $conn->query($sql);
if ($result->num_rows > 0) {
    // 输出每行数据
    while($row = $result->fetch_assoc()) {
        echo "序号:" . $row["id"] . " - 姓名:" . $row["name"] . "<br>";
    }
```

```
} else {
    echo "数据为空!";
}
```

② 插入数据的示例代码如下：

```
$sql = "INSERT INTO users (name, email) VALUES ('张三', zs@163.com')";
if ($conn->query($sql) === TRUE) {
    echo "新记录插入成功!";
} else {
    echo "错误:" . $sql . "<br>" . $conn->error;
}
```

③ 修改数据的示例代码如下：

```
$sql = "UPDATE users SET email = 'zhangsan@163.com' WHERE name = '张三'";
if ($conn->query($sql) === TRUE) {
    echo "记录更新成功!";
} else {
    echo "错误:" . $sql . "<br>" . $conn->error;
}
```

④ 删除数据的示例代码如下：

```
$sql = "DELETE FROM users WHERE name = '张三'";
if ($conn->query($sql) === TRUE) {
    echo "记录删除成功!";
} else {
    echo "错误:" . $sql . "<br>" . $conn->error;
}
```

(3) 关闭连接：完成操作后，通过 $conn->close() 关闭数据库连接，释放资源。

2. 使用 PDO 操作数据库

PDO 是一个更通用的数据库操作方式，支持多种数据库系统，具备更高的灵活性和可移植性，使用 PDO 操作数据库的基本步骤如下。

(1) 连接数据库：PDO 通过配置数据源名称、用户名和密码即可建立连接，并提供异常模式来捕获错误，示例代码如下：

```
$dsn = "mysql:host=localhost; dbname=my_database";
$username = "root";
$password = "root";
try {
    $pdo = new PDO($dsn, $username, $password);
    // 设置 PDO 错误模式为异常
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "连接成功!";
} catch (PDOException $e) {
    echo "连接失败:" . $e->getMessage();
}
```

(2) 执行 SQL 语句：PDO 支持多种数据库操作，包括查询、插入、更新和删除数据。可

以使用简单的 query()方法执行语句,也可以通过预处理函数 prepare()和 execute()提高安全性,防止 SQL 注入。

① 查询数据的示例代码如下:

```
// 使用 query 查询
$sql = "SELECT id, name FROM users";
$stmt = $pdo->query($sql);
while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
    echo "序号:" . $row['id'] . " - 姓名:" . $row['name'] . "<br>";
}
// 使用预处理语句(防止 SQL 注入)
$sql = "SELECT id, name FROM users WHERE name = :name";
$stmt = $pdo->prepare($sql);
$stmt->execute(['name' => '张三']);
while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
    echo "序号:" . $row['id'] . " - 姓名:" . $row['name'] . "<br>";
}
```

② 插入数据的示例代码如下:

```
$sql = "INSERT INTO users (name, email) VALUES (:name, :email)";
$stmt = $pdo->prepare($sql);
$stmt->execute(['name' => '张三', 'email' => 'zs@163.com']);
echo "新记录插入成功!";
```

③ 修改数据的示例代码如下:

```
$sql = "UPDATE users SET email = :email WHERE name = :name";
$stmt = $pdo->prepare($sql);
$stmt->execute(['email' => 'zhangsan@163.com', 'name' => '张三']);
echo "记录更新成功!";
```

④ 删除数据的示例代码如下:

```
$sql = "DELETE FROM users WHERE name = :name";
$stmt = $pdo->prepare($sql);
$stmt->execute(['name' => '张三']);
echo "记录删除成功!";
```

(3) 关闭连接:PDO 无关闭连接的方法,当对象销毁时将自动关闭连接。

提示:PHP 被广泛用于构建动态网站和 Web 应用,常见的安全问题如下。

(1) SQL 注入:最常见的安全漏洞之一,:通过在用户输入中嵌入恶意 SQL 代码,从而访问或篡改数据库。

(2) XSS 攻击:将恶意脚本注入页面中,当其他用户浏览该页面时,恶意脚本将在其浏览器上执行,窃取信息或进行其他恶意操作。

(3) CSRF 攻击:诱使已登录的用户在不知情的情况下执行不必要的操作,通过伪造请求来修改用户的账户信息或执行其他恶意行为。

(4) 文件上传漏洞:如果没有做充分的检查,可能会上传恶意文件,从而执行服务器端代码。

(5) 会话劫持：通过窃取或操控用户的会话 ID 来获取访问权限。

(6) 不安全的配置和错误信息泄露：PHP 在开发环境中显示详细的错误信息，但这些错误信息可能暴露数据库结构、文件路径或其他敏感信息。

(7) 不安全的依赖和过时的库：PHP 项目中使用的第三方库或框架可能存在已知的安全漏洞，可能通过这些漏洞发起攻击。

(8) 目录遍历攻击：通过提供恶意的文件路径，尝试访问应用程序未授权的目录或文件。

(9) 未加密的敏感数据：如果敏感数据以明文形式存储或传输，可能被攻击者窃取。

5.6　案　　例

使用 PHP 为第 3 章案例的页面添加真实的后端逻辑功能，实现用户登录、用户注册及 Web 漏洞列表的前后端交互功能。

5.6.1　用户登录

1. 前端

将第 3 章案例中从 user.json 加载数据修改为从后端 user.php 获取数据，并根据后端返回的 JSON 格式响应数据 result.status 来判断登录是否成功，如果返回结果是 status: 'success'，则跳转至 Web 漏洞列表页面，否则显示错误信息，核心代码如下：

```
// 使用 fetch()方法发送 POST 请求，提交用户名和密码，并根据返回结果做出不同的响应
try {
    const response = await fetch('login.php', {     // fetch()方法发送请求到 login.php 脚本
        method: 'POST',
        headers: {
            'Content-Type': 'application/x-www-form-urlencoded',
        },
        body: `username=${encodeURIComponent(username)}&password=${encodeURIComponent
(password)}`,
    });
    const result = await response.json();           // 解析服务器端响应数据
    if (result.status === 'success') {
        // 登录成功，跳转到 vuls.html
        window.location.href = "vuls.html";
    } else {
        // 登录失败，显示错误信息
        alert(result.message);
    }
} catch (error) {
    alert('请求出错:', error);
}
```

2. 后端

后端的主要功能是接收前端 POST 请求中的用户名和密码，与 MySQL 数据库中的用户信息进行比对，根据比对结果，返回 JSON 格式的响应信息，核心步骤如下。

(1) 连接数据库：首先，设置 Content-Type 为 application/json，即返回 JSON 格式数

据。然后，定义数据库连接参数并使用 mysqli 创建数据库连接，如果数据库连接失败，程序将返回错误信息并终止执行，核心代码如下：

```
header('Content-Type: application/json');
$servername = "localhost";          // 数据库服务器地址
$username = "root";                 // 数据库用户名
$password = "root";                 // 数据库密码
$dbname = "vuls";                   // 数据库名称
$conn = new mysqli($servername, $username, $password, $dbname);
// 检查数据库连接是否成功
if ($conn->connect_error) {
    // 连接失败时返回错误信息
    echo json_encode([
        'status' => 'error',
        'message' => '数据库连接失败:'. $conn->connect_error
    ]);
    exit();                         // 终止脚本执行
}
```

（2）获取 POST 请求中的用户名和密码：使用超全局变量 $_POST 获取 username 和 password，并使用 trim 去除多余的空格，核心代码如下：

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

（3）验证用户输入：通过预处理语句查询数据库中的 user 表，如果查询结果中存在用户，则验证密码是否正确，若密码正确，则返回登录成功的信息，若用户名不存在或密码错误，则返回错误信息，核心代码如下：

```
$stmt = $conn->prepare("SELECT * FROM user WHERE username = ?");
$stmt->bind_param("s", $username);                  // 绑定参数
$stmt->execute();                                   // 执行查询
$result = $stmt->get_result();                      // 获取查询结果
if ($result->num_rows > 0) {                        // 如果用户名存在
    $user = $result->fetch_assoc();
    if ($password === $user['password']) {          // 如果密码正确
        $response = [
            'status' => 'success',
            'message' => '登录成功!'
        ];
    } else {
        $response['message'] = '密码错误!';
    }
} else {
    $response['message'] = '用户名不存在!';
}
```

（4）处理空输入和关闭数据库连接：如果用户名或密码为空，则返回提示信息，处理完查询后，关闭预处理对象和数据库连接，核心代码如下：

```
if (empty($username) || empty($password)) {
    $response['message'] = '用户名和密码不能为空!';
```

```
}
$stmt->close();          // 关闭预处理对象
$conn->close();          // 关闭连接
```

(5) 返回结果：根据验证结果，返回 JSON 格式的结果。前端可以根据 status 字段的值进行相应处理，显示登录成功或失败的消息，后端代码如下：

```
echo json_encode($response);
```

5.6.2　用户注册

1. 前端

第 3 章案例中用户注册页面已经实现前后端交互的逻辑，本例修改注册结果的处理方法：检查返回的 JSON 数据中的 status 字段是否等于 success，如果 status 不等于 success，则弹出 result.message 作为错误提示，否则跳转到登录页面，核心代码如下：

```
if (result.status === 'success') {
    window.location.href = "login.html";
} else {
    alert(result.message);
}
```

2. 后端

后端主要处理前端提交的数据并与 MySQL 数据库交互。首先，获取前端提交的用户名、邮箱和密码。然后，检查用户名或邮箱是否已注册，若未被注册，则将用户数据插入数据库，数据插入成功返回 success 状态，失败则返回错误信息。最后关闭数据库连接，确保资源释放，核心步骤如下。

(1) 获取前端请求数据：从前端请求体中读取数据，提取 username、email 和 password，核心代码如下：

```
$requestData = json_decode(file_get_contents("php://input"), true);
$username = isset($requestData['username']) ? trim($requestData['username']) : '';
$email = isset($requestData['email']) ? trim($requestData['email']) : '';
$password = isset($requestData['password']) ? trim($requestData['password']) : '';
```

(2) 验证用户输入：通过预处理语句查询数据库表 user 中的数据，如果用户名或邮箱已经存在，则返回错误信息，否则，将用户名、密码和邮箱信息插入 user 表，若数据插入成功则返回注册成功信息，核心代码如下：

```
// 检查用户名或邮箱是否已存在
$stmt = $conn->prepare("SELECT id FROM user WHERE username = ? OR email = ?");
$stmt->bind_param("ss", $username, $email);
$stmt->execute();
$result = $stmt->get_result();
if ($result->num_rows > 0) {
    $response['message'] = '用户名或邮箱已被注册!';
```

```
    echo json_encode( $ response);
    exit;
}
// 插入新用户数据到数据库
$ stmt =  $ conn -> prepare("INSERT INTO user (username, password, email) VALUES (?, ?, ?)");
$ stmt -> bind_param("sss",  $ username,  $ password,  $ email);
if ( $ stmt -> execute()) {
    $ response = [
        'status' => 'success',
        'message' => '注册成功!'
    ];
} else {
    $ response['message'] = '注册失败,请稍后再试!';
}
```

5.6.3 Web 漏洞列表

1. 前端

将第 3 章案例中从 vulnerabilities.json 加载数据修改为从后端 vuls.php 获取数据,其余代码不变。

2. 后端

后端主要实现从 MySQL 数据库中查询漏洞信息。从 vulnerabilities 表中获取漏洞的 ID、CVE 编号、名称、类型和描述,并按 id 升序排序,如果查询成功,将查询结果保存到一个数组中,并以 JSON 格式输出查询到的漏洞数据,核心代码如下:

```
$ sql = "SELECT id, cve AS cveId, name, type, description FROM vulnerabilities ORDER BY id ASC";
$ result =  $ conn -> query( $ sql);
$ vulnerabilities = [];
if ( $ result -> num_rows > 0) {
    while ( $ row =  $ result -> fetch_assoc()) {
        $ vulnerabilities[] =  $ row;
    }
}
// 返回漏洞列表数据
echo json_encode( $ vulnerabilities, JSON_UNESCAPED_UNICODE);
```

5.7 本章小结

本章介绍了 PHP 的基础知识和关键技术,主要内容包括 PHP 的发展历史与特点,以及基本使用方法;PHPStudy 集成开发环境的安装方法。此外,还通过具体案例展示了如何实现登录、用户注册和 Web 漏洞列表的前后端交互功能,深入探讨使用 PHP 开发动态网页的方法。通过本章的学习,读者能够掌握 PHP 的基本使用方法,深入了解动态网页前后端交互的原理及常见功能的开发方法。

课 后 实 验

一、实验目的和要求

(1) 掌握 PHPStudy 的安装方法。

(2) 掌握 PHP 的基本用法。

二、实验环境

(1) 实验设备：微型计算机。

(2) 软件系统：Windows、VS Code 和 PHPStudy。

三、实验内容

基于第 3 章的课后实验，结合 PHP 语言，在下列两类题目中，分别任选一个题目完成，实现网页的前后端交互功能。

1. 登录验证类

(1) 用户注册页面的设计与实现。

(2) 用户密码重置页面的设计与实现。

(3) 用户信息修改页面的设计与实现。

(4) 用户登录记录页面的设计与实现。

(5) 用户权限管理页面的设计与实现。

2. 列表显示类

(1) 用户信息管理列表页面。

(2) 商品列表页面。

(3) 订单管理列表页面。

(4) 新闻公告列表页面。

(5) 文件管理列表页面。