

PHP代码审计

为什么代码会有漏洞

CWE 项目中大约有 1000 种不同的软件缺陷。这些都是软件开发人员使用不安全的方式执行代码逻辑所引发的安全问题。

软件开发没有学习过安全知识，大多数人也没有接受过任何关于软件安全的培训。

这些问题在最近几年变得非常重要，因为我们在以极快的速度增加互联方式，技术和协议。发明技术的能力已经远远超过了保护技术的能力。今天使用的许多技术根本没有受到足够(或任何)的安全审计。

企业没有在安全上花费适当的时间有许多原因。最直观的原因是源于软件市场的一个潜在问题。

因为软件本质上是一个黑盒，客户很难区分安全和不安全代码。

没有这种可见性，就不鼓励供应商花费额外的精力来生产安全的代码。

什么是代码安全审计

代码审计旨在识别应用程序中与其特性和设计相关的安全缺陷，以及产生缺陷的根本原因。

随着应用程序的日益复杂和新技术的出现，传统的测试方法可能无法检测到应用程序中存在的所有安全缺陷。人们必须理解应用程序、外部组件和配置的代码，这样才能更好地发现缺陷。深入地研究应用程序代码也有助于确定可用于避免安全缺陷的缓解技术。

审核应用程序源代码的过程是为了验证适当的安全和逻辑控制是否存在，它们是否按预期工作，以及它们是否在正确的位置被调用。

代码安全审计允许公司确保应用程序开发人员遵循安全开发技术。一般的经验法则是，在应用程序经过适当的代码安全审计后，渗透测试不应发现与开发的代码相关的任何其他应用程序漏洞。或者发现很少的问题。

代码安全审计技术

代码安全审计与被审计的应用程序强相关。它们可能会突出一些新的或特定于应用程序代码实现的缺陷，如执行流的不安全终止、同步错误等。这些缺陷只有当我们理解了应用程序代码流及其逻辑后才能被发现。因此，代码安全审计不仅仅是扫描代码中的一组未知的不安全代码模式，还包括理解应用程序的代码实现和列举它独有的缺陷。

正在审计的应用程序可能已经设计了一些适当的安全控制，例如集中黑名单、输入验证等。必须仔细研究这些安全措施，以确定它们是否可靠。根据控制的实施，必须分析攻击的性质或任何可用于绕过它的特定攻击向量。

列举现有安全控制中的弱点是代码安全审计的另一个重要方面。

应用程序中出现安全缺陷有多种原因，比如缺少输入验证或参数处理不当。在代码审计的过程中，缺陷的根本原因需要被暴露出来，要跟踪完整的数据流。确定应用程序(源)的所有可能的输入，以及它们是如何被应用程序(接收器)处理的。接收器可能是一种不安全的代码模式，如动态 SQL 查询、日志编写器或对客户端设备的响应。

考虑一个源是用户输入的场景。它流经应用程序的不同类/组件，最后落入一个拼接的SQL 查询(一个接收器)中，并且在路径中没有对它进行适当的验证。在这种情况下，应用程序将容易受到 SQL 注入攻击，这是由源到目的分析确定的。这种分析有助于理解哪些易受攻击的输入可能导致应用程序中的漏洞。

一旦发现缺陷，审计者必须列举应用程序中存在的所有可能的实例。这不是由代码变更发起的代码审计，这是由管理部门基于发现的缺陷发起的代码扫描，并且投入资源来查找该缺陷是否存在于产品的其他部分。例如，由于在不安全的显示方法中使用未经验证的输入，应用程序很容易在这些地方受到 XSS 漏洞的攻击。

应用程序功能和业务规则

审计人员应了解应用程序当前提供的所有功能，并获取与这些功能相关的所有业务限制规则。还有一种情况是，要注意潜在的计划中的功能，这些功能可能会出现在应用程序的路线图上，从而在当前的代码审计过程中对安全决策进行提前验证。这个系统失败的后果是什么？如果应用程序不能按预期执行其功能，企业会受到很大影响吗？

上下文

所有的安全都在我们试图保护的范围内。在苹果的应用程序上推荐军事安全标准机制将是矫枉过正不恰当的。什么类型的数据被操纵或处理，如果这些数据被泄露会对公司造成什么损害？上下文是安全代码审计和风险评估的“圣杯”。

敏感数据

审计人员还应记录对应用程序敏感的数据实体，如账号和密码。根据敏感度对数据实体进行分类将有助于审计者确定应用程序中任何类型的数据丢失的影响。

用户角色和访问权限

了解被允许访问应用程序的用户类型很重要。是面向外部还是内部给“信任”的用户？一般来说，只有组织内部用户才能访问的应用程序可能与互联网上任何人都能访问的面临不同的威胁。因此，了解应用程序的用户及其部署的环境将允许审计者正确认识威胁。除此之外，还必须了解应用程序中存在的不同权限级别。这将有助于审计者列举适用于应用程序的不同安全违规/权限提升攻击。

应用类型

这是指了解应用程序是基于浏览器的应用程序、基于桌面的独立应用程序、网络服务、移动应用程序还是混合应用程序。不同类型的应用程序面临不同类型的安全威胁，了解应用程序的类型将有助于审计者查找特定的安全缺陷，确定正确的威胁代理，并突出适合应用程序的必要控制。

代码

使用的语言，从安全角度看该语言的特点和问题。从安全性和性能的角度来看，程序员需要注意的问题和语言最佳实践。

设计

一般来说，如果使用 MVC 设计原则开发，网络应用程序有一个定义良好的代码布局。应用程序可以有自己的定制设计，也可以使用一些著名的设计框架，如 Struts/Spring 等。

应用程序属性/配置参数存储在哪里？

如何为任何功能/URL 识别业务类别？

什么类型的类被执行来处理请求(例如。集中式控制器、命令类、视图页面等)?

对于任何请求，视图是如何呈现给用户的?

确定攻击面

通过分析输入、数据流和事务来确定攻击面。实际执行代码安全审计的主要部分是对攻击面进行分析。应用程序接受输入并产生某种输出。第一步是识别代码的所有输入。

应用程序的输入可能包括以下要点:

| 浏览器输入

| Cookie

| 文件

| 命令行参数

| 环境变量

PHP代码审计思路

- 敏感函数方法回溯(反向审计)
 - 查找项目中的敏感函数方法，查找传入的参数判断用户是否可控
- 用户可控参数正向查找
 - 查找项目中的用户输入 追踪用户输入 判断是否得到有效的过滤/调用敏感函数/存在逻辑问题
- 关键业务功能分析(功能审计)
 - 专门审计易出现漏洞的关键功能点
 - 如 头像上传 系统登陆 文件下载 等功能
- 审计所有代码

用户可控参数

来自用户可控的输入, 安全审计中永远不要相信用户的输入

变量/常量/函数/等	描述
\$_SERVER	包含 服务器信息 环境变量 用户传入的http头和uri路径等信息
\$_GET \$HTTP_GET_VARS	包含 用户传入的URL参数
\$_POST \$HTTP_POST_VARS	包含 用户传入的POST BODY的参数 (当 HTTP头中 Content-Type 值为 application/x-www- form-urlencoded 或 multipart/form-data时才会被传入)
\$_FILES \$HTTP_POST_FILES	包含 用户上传文件信息 文件内容 原文件名 临时文件名 大小 等信息
\$_COOKIE \$HTTP_COOKIE_VARS	包含 用户传入的HTTP头中的Cookies kv值
\$_REQUEST	同时包含 \$GET \$POST \$_COOKIE
php://input \$HTTP_RAW_POST_DATA	包含 用户POST请求中BODY 的完整数据 常见用法 file_get_contents('php://input');
apache_request_headers() getallheaders()	包含 用户传入的http头 (Apache ONLY)

```

$_SERVER[ 'HTTP_ACCEPT_LANGUAGE' ] //浏览器语言
$_SERVER[ 'REMOTE_ADDR' ] //当前用户 IP 。
$_SERVER[ 'REMOTE_HOST' ] //当前用户主机名
$_SERVER[ 'REQUEST_URI' ] //URL
$_SERVER[ 'REMOTE_PORT' ] //端口。
$_SERVER[ 'SERVER_NAME' ] //服务器主机的名称。
$_SERVER[ 'PHP_SELF' ] //正在执行脚本的文件名
$_SERVER[ 'argv' ] //传递给该脚本的参数。
$_SERVER[ 'argc' ] //传递给程序的命令行参数的个数。
$_SERVER[ 'GATEWAY_INTERFACE' ] //CGI 规范的版本。
$_SERVER[ 'SERVER_SOFTWARE' ] //服务器标识的字符串
$_SERVER[ 'SERVER_PROTOCOL' ] //请求页面时通信协议的名称和版本
$_SERVER[ 'REQUEST_METHOD' ] //访问页面时的请求方法
$_SERVER[ 'QUERY_STRING' ] //查询(query)的字符串。
$_SERVER[ 'DOCUMENT_ROOT' ] //当前运行脚本所在的文档根目录
$_SERVER[ 'HTTP_ACCEPT' ] //当前请求的 Accept：头部的内容。
$_SERVER[ 'HTTP_ACCEPT_CHARSET' ] //当前请求的 Accept-Charset：头部的内容。
$_SERVER[ 'HTTP_ACCEPT_ENCODING' ] //当前请求的 Accept-Encoding：头部的内容
$_SERVER[ 'HTTP_CONNECTION' ] //当前请求的 Connection：头部的内容。例如：“Keep-Alive”。
$_SERVER[ 'HTTP_HOST' ] //当前请求的 Host：头部的内容。
$_SERVER[ 'HTTP_REFERER' ] //链接到当前页面的前一页面的 URL 地址。
$_SERVER[ 'HTTP_USER_AGENT' ] //当前请求的 User-Agent：头部的内容。
$_SERVER[ 'HTTPS' ] //如果通过https访问,则被设为一个非空的值(on), 否则返回off

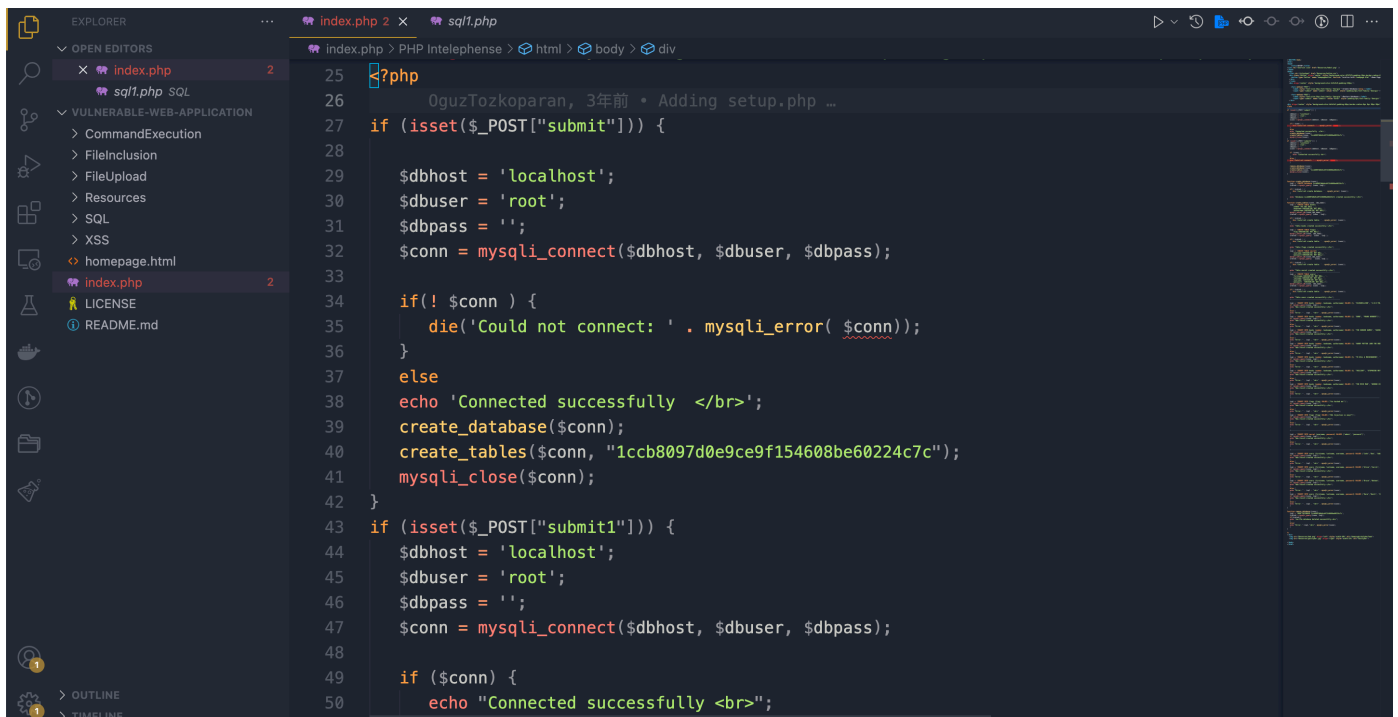
```

```
$_SERVER['SCRIPT_FILENAME'] #当前执行脚本的绝对路径名。
$_SERVER['SERVER_ADMIN'] #管理员信息
$_SERVER['SERVER_PORT'] #服务器所使用的端口
$_SERVER['SERVER_SIGNATURE'] #包含服务器版本和虚拟主机名的字符串。
$_SERVER['PATH_TRANSLATED'] #当前脚本所在文件系统（不是文档根目录）的基本路径。
$_SERVER['SCRIPT_NAME'] #包含当前脚本的路径。这在页面需要指向自己时非常有用。
$_SERVER['PHP_AUTH_USER'] #当 PHP 运行在 Apache 模块方式下，并且正在使用 HTTP 认证功能，这个变量便是用户输入的用户名。
$_SERVER['PHP_AUTH_PW'] #当 PHP 运行在 Apache 模块方式下，并且正在使用 HTTP 认证功能，这个变量便是用户输入的密码。
$_SERVER['AUTH_TYPE'] #当 PHP 运行在 Apache 模块方式下，并且正在使用 HTTP 认证功能，这个变量便是认证的类型
```

可以通过 `Request` 对象完成全局输入变量的检测、获取和安全过滤，支持包括

`$_GET`、`$_POST`、`$_REQUEST`、`$_SERVER`、`$_SESSION`、`$_COOKIE`、`$_ENV` 等系统变量，以及文件上传信息。

以 `vulnerable-Web-Application` 代码为例



```
?php
0guZTozkoparan, 3年前 • Adding setup.php ...
if (isset($_POST["submit"])) {
    $dbhost = 'localhost';
    $dbuser = 'root';
    $dbpass = '';
    $conn = mysqli_connect($dbhost, $dbuser, $dbpass);

    if(! $conn ) {
        die('Could not connect: ' . mysqli_error( $conn));
    }
    else
    echo 'Connected successfully <br>';
    create_database($conn);
    create_tables($conn, "1ccb8097d0e9ce9f154608be60224c7c");
    mysqli_close($conn);
}

if (isset($_POST["submit1"])) {
    $dbhost = 'localhost';
    $dbuser = 'root';
    $dbpass = '';
    $conn = mysqli_connect($dbhost, $dbuser, $dbpass);

    if ($conn) {
        echo "Connected successfully <br>";
    }
}
```

审计流程：

- 查找用户输入
- 查找敏感函数
- 审计业务功能

Composer安装

Composer 是 PHP 用来管理依赖（dependency）关系的工具。可以在自己的项目中声明所依赖的外部工具库（libraries），Composer 会帮你安装这些依赖的库文件。

安装 Composer，你只需要下载 `composer.phar` 可执行文件。

```
curl -sS https://getcomposer.org/installer | php
mv composer.phar /usr/local/bin/composer
```

在 Windows 中，你需要下载并运行 [Composer-Setup.exe](#)。

为了避免安装过慢，可以使用阿里云的 `composer` 镜像

```
composer config -g repo.packagist composer https://packagist.phpcomposer.com
```

安装依赖时可使用命令

```
composer install
```

THINKPHP框架输入变量

[官方文档](#)

概述

ThinkPHP是一个免费开源的，快速、简单的面向对象的轻量级PHP开发框架，是为了敏捷WEB应用开发和简化企业应用开发而诞生的。ThinkPHP从诞生以来一直秉承简洁实用的设计原则，在保持出色的性能和至简的代码的同时，也注重易用性。遵循 `Apache2` 开源许可协议发布，意味着你可以免费使用ThinkPHP，甚至允许把你基于ThinkPHP开发的应用开源或商业产品发布/销售。

通过 `composer` 启动一个thinkphp示例项目

```
composer create-project tophink/think tp
cd tp
php think run

# 访问本地8000地址
http://localhost:8000
```

```
> ls
LICENSE.txt  app  composer.lock  extend  route  think  view
README.md   composer.json  config  public  runtime  vendor

> php think run
ThinkPHP Development server is started On <http://0.0.0.0:8000/>
You can exit with CTRL-C
Document root is: /opt/test_code/tp/public
[Mon Mar 7 11:15:35 2022] PHP 8.0.13 Development Server (http://0.0.0.0:8000) started
^@^@^@^@[Mon Mar 7 11:16:19 2022] 127.0.0.1:61319 Accepted
[Mon Mar 7 11:16:19 2022] 127.0.0.1:61320 Accepted
[Mon Mar 7 11:16:19 2022] 127.0.0.1:61319 [200]: GET /
[Mon Mar 7 11:16:19 2022] 127.0.0.1:61319 Closing
[Mon Mar 7 11:16:19 2022] 127.0.0.1:61320 [200]: GET /favicon.ico
[Mon Mar 7 11:16:19 2022] 127.0.0.1:61320 Closing
```

thinkphp框架目录如下，可以看到初始的目录结构：

```
project 应用部署目录
├─application          应用目录（可设置）
│   ├─common          公共模块目录（可更改）
│   ├─index           模块目录（可更改）
│   │   ├─config.php  模块配置文件
│   │   ├─common.php  模块函数文件
│   │   ├─controller  控制器目录
│   │   ├─model       模型目录
│   │   ├─view        视图目录
│   │   └─...         更多类库目录
│   ├─command.php     命令行工具配置文件
│   ├─common.php      应用公共（函数）文件
│   ├─config.php      应用（公共）配置文件
│   ├─database.php    数据库配置文件
│   ├─tags.php        应用行为扩展定义文件
│   └─route.php       路由配置文件
├─extend              扩展类库目录（可定义）
├─public              WEB 部署目录（对外访问目录）
│   ├─static          静态资源存放目录(css,js,image)
│   ├─index.php       应用入口文件
│   ├─router.php      快速测试文件
│   └─.htaccess       用于 apache 的重写
├─runtime             应用的运行时目录（可写，可设置）
├─vendor              第三方类库目录（Composer）
├─thinkphp            框架系统目录
│   ├─lang            语言包目录
│   ├─library         框架核心类库目录
│   │   └─think       Think 类库包目录
│   │       └─traits  系统 Traits 目录
│   ├─tpl             系统模板目录
│   ├─.htaccess       用于 apache 的重写
│   ├─.travis.yml     CI 定义文件
│   ├─base.php        基础定义文件
│   ├─composer.json   composer 定义文件
│   ├─console.php     控制台入口文件
│   ├─convention.php  惯例配置文件
│   ├─helper.php      助手函数文件（可选）
│   ├─LICENSE.txt     授权说明文件
│   ├─phpunit.xml     单元测试配置文件
│   ├─README.md       README 文件
│   └─start.php       框架引导文件
├─build.php           自动生成定义文件（参考）
├─composer.json       composer 定义文件
├─LICENSE.txt         授权说明文件
└─README.md           README 文件
```

入口文件

用户请求的PHP文件，负责处理一个请求（注意，不一定是URL请求）的生命周期，最常见的入口文件就是 `index.php`，有时候也会为了某些特殊的需求而增加新的入口文件，例如给后台模块单独设置的一个入口文件 `admin.php` 或者一个控制器程序入口 `think` 都属于入口文件。

MVC模式

MVC的全名是Model View Controller，是模型(Model)–视图(view)–控制器(controller)的缩写，是一种设计模式。它是用一种业务逻辑、数据与界面显示分离的方法来组织代码，将众多的业务逻辑聚集到一个部件里面，在需要改进和个性化定制界面及用户交互的同时，不需要重新编写业务逻辑，达到减少编码的时间，提高代码复用性。

使用的MVC的目的：它将这些对象、显示、控制分离以提高软件的灵活性和复用性，MVC结构可以使程序具有对象化的特征，也更容易维护。

模型层（Model）：指从现实世界中抽象出来的对象模型，是应用逻辑的反应；它封装了数据和对数据的操作，是实际进行数据处理的地方（模型层与数据库才有交互）

视图层（View）：是应用和用户之间的接口，它负责将应用显示给用户 和 显示模型的状态。

控制器（Controller）：控制器负责视图和模型之间的交互，控制对用户输入的响应、响应方式和流程；它主要负责两方面的动作，一是把用户的请求分发到相应的模型，二是把模型的变化及时地反映到视图上。

控制器

每个模块拥有独立的 `mvc` 类库及配置文件，一个模块下面有多个控制器负责响应请求，而每个控制器其实就是一个独立的控制器类。

控制器主要负责请求的接收，并调用相关的模型处理，并最终通过视图输出。严格来说，控制器不应该过多的介入业务逻辑处理。

事实上，5.0中控制器是可以被跳过的，通过路由我们可以直接把请求调度到某个模型或者其他的类进行处理。

5.0的控制器类比较灵活，可以无需继承任何基础类库。

一个典型的 `Index` 控制器类如下：

```
namespace app\index\controller;

class Index
{
    public function index()
    {
        return 'hello,thinkphp!';
    }
}
```


操作

一个控制器包含多个操作（方法），操作方法是一个URL访问的最小单元。

下面是一个典型的 `Index` 控制器的操作方法定义，包含了两个操作方法：

```
namespace app\index\controller;

class Index
{
    public function index()
    {
        return 'index';
    }

    public function hello($name)
    {
        return 'Hello,'.$name;
    }
}
```

操作方法可以不使用任何参数，如果定义了一个非可选参数，则该参数必须通过用户请求传入，如果是URL请求，则通常是 `$_GET` 或者 `$_POST` 方式传入。

检测变量是否设置

可以使用 `has` 方法来检测一个变量参数是否设置，如下：

```
Request::instance()->has('id','get');
Request::instance()->has('name','post');
```

或者使用助手函数

```
input('?get.id');
input('?post.name');
```

变量检测可以支持所有支持的系统变量。

变量获取

变量获取使用 `\think\Request` 类的如下方法及参数：

变量类型方法('变量名/变量修饰符','默认值','过滤方法')

变量类型方法包括：

方法	描述
param	获取当前请求的变量
get	获取 \$_GET 变量
post	获取 \$_POST 变量
put	获取 PUT 变量
delete	获取 DELETE 变量
session	获取 \$_SESSION 变量
cookie	获取 \$_COOKIE 变量
request	获取 \$_REQUEST 变量
server	获取 \$_SERVER 变量
env	获取 \$_ENV 变量
route	获取 路由（包括PATHINFO） 变量
file	获取 \$_FILES 变量

获取 PARAM 变量

PARAM变量是框架提供的用于自动识别 GET、POST 或者 PUT 请求的一种变量获取方式，是系统推荐的获取请求参数的方法，用法如下：

```
// 获取当前请求的name变量
Request::instance()->param( 'name' );
// 获取当前请求的所有变量（经过过滤）
Request::instance()->param();
// 获取当前请求的所有变量（原始数据）
Request::instance()->param( false );
// 获取当前请求的所有变量（包含上传文件）
Request::instance()->param( true );
```

param方法会把当前请求类型的参数和PATH_INFO变量以及GET请求合并。

使用助手函数实现：

```
input( 'param.name' );
input( 'param.' );
或者
input( 'name' );
input( '' );
```

因为 input 函数默认就采用PARAM变量读取方式。

获取 GET 变量

```
Request::instance()->get('id'); // 获取某个get变量
Request::instance()->get('name'); // 获取get变量
Request::instance()->get(); // 获取所有的get变量（经过过滤的数组）
Request::instance()->get(false); // 获取所有的get变量（原始数组）
```

或者使用内置的助手函数 `input` 方法实现相同的功能：

```
input('get.id');
input('get.name');
input('get.');
```

注：pathinfo地址参数不能通过get方法获取，查看“获取PARAM变量”

获取 POST 变量

```
Request::instance()->post('name'); // 获取某个post变量
Request::instance()->post(); // 获取经过过滤的全部post变量
Request::instance()->post(false); // 获取全部的post原始变量
```

使用助手函数实现：

```
input('post.name');
input('post.');
```

获取 PUT 变量

```
Request::instance()->put('name'); // 获取某个put变量
Request::instance()->put(); // 获取全部的put变量（经过过滤）
Request::instance()->put(false); // 获取全部的put原始变量
```

使用助手函数实现：

```
input('put.name');
input('put.');
```

获取 REQUEST 变量

```
Request::instance()->request('id'); // 获取某个request变量
Request::instance()->request(); // 获取全部的request变量（经过过滤）
Request::instance()->request(false); // 获取全部的request原始变量数据
```

使用助手函数实现：

```
input('request.id');  
input('request.');
```

获取 **SERVER** 变量

```
Request::instance()->server('PHP_SELF'); // 获取某个server变量  
Request::instance()->server(); // 获取全部的server变量
```

使用助手函数实现：

```
input('server.PHP_SELF');  
input('server.');
```

获取 **SESSION** 变量

```
Request::instance()->session('user_id'); // 获取某个session变量  
Request::instance()->session(); // 获取全部的session变量
```

使用助手函数实现：

```
input('session.user_id');  
input('session.');
```

获取 **Cookie** 变量

```
Request::instance()->cookie('user_id'); // 获取某个cookie变量  
Request::instance()->cookie(); // 获取全部的cookie变量
```

使用助手函数实现：

```
input('cookie.user_id');  
input('cookie.');
```

变量过滤

框架默认没有设置任何过滤规则，你可以是配置文件中设置全局的过滤规则：

```
// 默认全局过滤方法 用逗号分隔多个  
'default_filter' => 'htmlspecialchars',
```

也支持使用 `Request` 对象进行全局变量的获取过滤，过滤方式包括函数、方法过滤，以及PHP内置的Types of filters，我们可以设置全局变量过滤方法，例如：

```
Request::instance()->filter('htmlspecialchars');
```

支持设置多个过滤方法，例如：

```
Request::instance()->filter(['strip_tags','htmlspecialchars'],
```

也可以在获取变量的时候添加过滤方法，例如：

```
Request::instance()->get('name','', 'htmlspecialchars'); // 获取get变量 并用
htmlspecialchars函数过滤
Request::instance()->param('username','', 'strip_tags'); // 获取param变量 并用strip_tags函
数过滤
Request::instance()->post('name','', 'org\Filter::safeHtml'); // 获取post变量 并用
org\Filter类的safeHtml方法过滤
```

可以支持传入多个过滤规则，例如：

```
Request::instance()->param('username','', 'strip_tags,strtoupper'); // 获取param变量 并依次
调用strip_tags、strtoupper函数过滤
```

Request对象还支持PHP内置提供的Filter ID过滤，例如：

```
Request::instance()->post('email','', FILTER_VALIDATE_EMAIL);
```

框架对FilterID做了转换支持，因此也可以使用字符串的方式，例如：

```
Request::instance()->post('email','', 'email');
```

采用字符串方式定义 FilterID 的时候，系统会自动进行一次 filter_id 调用转换成 Filter 常量。

具体的字符串根据 filter_list 函数的返回值来定义。

需要注意的是，采用Filter ID 进行过滤的话，如果不符合过滤要求的话 会返回false，因此你需要配合默认值来确保最终的值符合你的规范。

例如，

```
Request::instance()->post('email','', FILTER_VALIDATE_EMAIL);
```

就表示，如果不是规范的email地址的话 返回空字符串。

如果当前不需要进行任何过滤的话，可以使用（v5.0.3+ 版本）

```
// 获取get变量 并且不进行任何过滤 即使设置了全局过滤
Request::instance()->get('name','', null);
```

获取部分变量

如果你只需要获取当前请求的部分参数，可以使用：

```
// 只获取当前请求的id和name变量
Request::instance()->only('id,name');
```

或者使用数组方式

```
// 只获取当前请求的id和name变量
Request::instance()->only(['id','name']);
```

默认获取的是当前请求参数，如果需要获取其它类型的参数，可以使用第二个参数，例如：

```
// 只获取GET请求的id和name变量
Request::instance()->only(['id','name'],'get');
// 只获取POST请求的id和name变量
Request::instance()->only(['id','name'],'post');
```

排除部分变量

也支持排除某些变量获取，例如

```
// 排除id和name变量
Request::instance()->except('id,name');
```

或者使用数组方式

```
// 排除id和name变量
Request::instance()->except(['id','name']);
```

同样支持指定变量类型获取：

```
// 排除GET请求的id和name变量
Request::instance()->except(['id','name'],'get');
// 排除POST请求的id和name变量
Request::instance()->except(['id','name'],'post');
```

变量修饰符

`input` 函数支持对变量使用修饰符功能，可以更好的过滤变量。

用法如下：

input('变量类型.变量名/修饰符');

或者

Request::instance()->变量类型('变量名/修饰符');

例如：

```
input('get.id/d');
input('post.name/s');
input('post.ids/a');
Request::instance()->get('id/d');
```

ThinkPHP5.0版本默认的变量修饰符是 /s，如果需要传入字符串之外的变量可以使用下面的修饰符，包括：

修饰符	作用
s	强制转换为字符串类型
d	强制转换为整型类型
b	强制转换为布尔类型
a	强制转换为数组类型
f	强制转换为浮点类型

原生查询

Db 类支持原生 SQL 查询操作，主要包括下面两个方法：

query 方法

query 方法用于执行 SQL 查询操作，如果数据非法或者查询错误则返回false，否则返回查询结果数据集（同 select 方法）。

使用示例：

```
Db::query("select * from think_user where status=1");
```

execute 方法

execute用于更新和写入数据的sql操作，如果数据非法或者查询错误则返回false，否则返回影响的记录数。

使用示例：

```
Db::execute("update think_user set name='thinkphp' where status=1");
```

参数绑定

支持在原生查询的时候使用参数绑定，包括问号占位符或者命名占位符，例如：

```
Db::query('select * from think_user where id=?',[8]);
Db::execute('insert into think_user (id, name) values (?, ?)',[8,'thinkphp']);
```

也支持命名占位符绑定，例如：

```
Db::query('select * from think_user where id=:id',['id'=>8]);
Db::execute('insert into think_user (id, name) values (:id, :name)',
['id'=>8,'name'=>'thinkphp']);
```

基本查询

查询一个数据使用：

```
// table方法必须指定完整的数据表名
Db::table('think_user')->where('id',1)->find();
```

find 方法查询结果不存在，返回 null

查询数据集使用：

```
Db::table('think_user')->where('status',1)->select();
```

select 方法查询结果不存在，返回空数组

如果设置了数据表前缀参数的话，可以使用

```
Db::name('user')->where('id',1)->find();
Db::name('user')->where('status',1)->select();
```

PHP敏感函数

命令执行函数

函数/语法	描述	例子
system	执行命令并输出结果	system('id');
exec	执行命令 只可获取最后一行结果	exec('id',\$a); print_r(\$a);
passthru	同 system	passthru('id');
shell_exec (反引号) 执行命令并返回结果 \$a=shell_exec('id');print_r(\$a);\$a=id`;print_r(\$a);		
popen	执行命令并建立管道 返回一个指针 使用fread等函数操作指针进行读写	\$a=popen("id", "r"); echo fread(\$a, 2096);
proc_open	同 popen (进程控制功能更强大)	见PHP手册
pcntl_exec	执行命令 只返回是否发生错误	pcntl_exec('id');

```
<?php
    exec('ping 127.0.0.1',$output,$return_var);

    system('ping -c 127.0.0.1',$return_var);

    passthru('ping 12.0.0.1',$return_var);
    passthru('id');

    shell_exec("ping 127.0.0.1");
    shell_exec(`ping 127.0.0.1`);

    popen("id", "r");

    pcntl_exec('id');
```

mail 函数

php的mail函数声明如下：

其参数含义分别表示如下：

- to，指定邮件接收者，即接收人
- subject，邮件的标题
- message，邮件的正文内容
- additional_headers，指定邮件发送时其他的额外头部，如发送者From，抄送CC，隐藏抄送BCC
- additional_parameters，指定传递给发送程序sendmail的额外参数。

在Linux系统上，mail函数在底层实现中，默认调用Linux的[sendmail](#)程序发送邮件。在sendmail程序的参数中，有一个 `-x` 选项，用于记录所有的邮件进出流量至log文件中。

通过 `-x` 指定log文件记录邮件流量，实际可以达到写文件的效果。

例如，如下php代码

```
$to = 'Alice@example.com';
$subject = 'Hello Alice!';
$message='<?php phpinfo(); ?>';
$headers = "CC: somebodyelse@example.com";
$options = '-OQueueDirectory=/tmp -X/var/www/html/rce.php';
mail($to, $subject, $message, $headers, $options);
```

执行后，查看log文件 `/var/www/html/rce.php`

```
17220 <<< To: Alice@example.com
17220 <<< Subject: Hello Alice!
17220 <<< X-PHP-Originating-Script: 0:test.php
17220 <<< CC: somebodyelse@example.com
17220 <<<
17220 <<< <?php phpinfo(); ?>
17220 <<< [EOF]
```

发现被写入了包含在邮件标题或正文中的php代码，通过访问此log文件可以执行预先可控的php代码。

因此，对php mail函数使用时，应该特别注意第5个参数 `additional_parameters` 的使用，防止被攻击者可控，注入 `-x` 参数，执行命令。

修复方案

方案一：使用过滤函数内置函数 `escapeshellcmd()`, `escapeshellarg()` ,过滤和转义输入中的特殊字符

escapeshellcmd()：会在以下字符之前插入反斜线 (\)：'|*?~<>^()[]{}\$\\, \x0A 和 \xFF。' 和 " 仅在不配对儿的时候被转义。

在 Windows 平台上，所有这些字符以及 % 和 ! 字符都会被空格代替。

escapeshellarg()：将给字符串增加一个单引号并且能引用或者转码任何已经存在的单引号，这样以确保能够直接将一个字符串传入 shell 函数，并且还是确保安全的。对于用户输入的部分参数就应该使用这个函数。

代码注入/文件包含函数

函数/语法结构	描述	例子
eval	将传入的参数内容作为PHP代码执行 eval 不是函数 是一种语法结构 不能当做函数动态调用	eval('phpinfo();');
assert	将传入的参数内容作为PHP代码执行 版本在PHP7以下是函数 PHP7及以上为语法结构	assert('phpinfo();');
preg_replace	当preg_replace使用/e修饰符且原字符串可控时时 有可能执行 php代码	echo preg_replace("/e","\${PHPINFO()}}","123");
call_user_func	把第一个参数作为回调函数调用 需要两个参数都完全可控才可利用 只能传入一个参数调用	call_user_func('assert', 'phpinfo();');
call_user_func_array	同call_user_func 可传入一个数组带入多个参数调用函数	call_user_func_array('file_put_contents', ['1.txt','6666']);
create_function	根据传递的参数创建匿名函 数，并为其返回唯一名称 利用需要第二个参数可控 且创建的函数被执行	\$f = create_function("", 'system(\$_GET[123]);'); \$f();
include	包含并运行指定文件 执行出错会抛出错误	include 'vars.php'; (括号可有可无)
require	同include 执行出错会抛出警告	require('somefile.php'); (括号可有可无)
require_once	同require 但会检查之前是否已经包含该文件 确保不重复包含	
include_once	同include 但会检查之前是否已经包含该文件 确保不重复包含	

```
<?php
eval('phpinfo();');
assert('phpinfo();');
echo preg_replace("/e","${PHPINFO()}}","123");
call_user_func('assert', 'phpinfo();');
call_user_func_array('file_put_contents', ['1.txt','6666']);
$f = create_function('', 'system($_GET[123]);'); $f();
include 'vars.php';
require('somefile.php');
```

文件读取/SSRF函数

函数	描述	例子
file_get_contents	读入文件返回字符串	echo file_get_contents("flag.txt"); echo file_get_contents("https://www.w.bilibili.com/");
curl_setopt curl_exec	Curl访问url获取信息	function curl(\$url){ \$ch = curl_init(); curl_setopt(\$ch, CURLOPT_URL, \$url); curl_exec(\$ch); curl_close(\$ch); } \$url=\$_GET['url']; curl(\$url); https://www.php.net/manual/zh/function.curl-exec.php
fsockopen	打开一个套接字连接(远程tcp/udp raw)	https://www.php.net/manual/zh/function.fsockopen.php
readfile	读取一个文件，并写入到输出缓冲	同file_get_contents
fopen/fread/fgets/fgetss/fgetc/fgetcsv/fpassthru/fscanf	打开文件或者 URL 读取文件流	\$file = fopen("test.txt","r"); echo fread(\$file,"1234"); fclose(\$file);
file	把整个文件读入一个数组中	echo implode("", file("https://www.bilibili.com/ "));
highlight_file/show_source	语法高亮一个文件	highlight_file("1.php");
parse_ini_file	读取并解析一个ini配置文件	print_r(parse_ini_file('1.ini'));
simplexml_load_file	读取文件作为XML文档解析	

```
<?php
echo file_get_contents("flag.txt");
echo file_get_contents("https://www.bilibili.com/");

$file = fopen("test.txt","r");
echo fread($file,"1234");
fclose($file);

echo implode(' ', file('https://www.bilibili.com/ '));

highlight_file("1.php");

print_r(parse_ini_file('1.ini'));
```

```
$fp = fsockopen("www.example.com", 80, $errno, $errstr, 30);
if (!$fp) {
    echo " $errstr ($errno)<br />\n";
} else {
    $out = "GET / HTTP/1.1\r\n";
    $out .= "Host: www.example.com\r\n";
    $out .= "Connection: Close\r\n\r\n";
    fwrite($fp, $out);
    while (!feof($fp)) {
        echo fgets($fp, 128);
    }
    fclose($fp);
}
```

文件上传/写入/其他函数

函数	描述	例子
file_put_contents	将一个字符串写入文件	file_put_contents("test.txt","magedu");
move_uploaded_file	将上传的临时文件移动到新的位置	move_uploaded_file(\$_FILES["pictures"] ["tmp_name"],"magedu.php")
rename	重命名文件/目录	rename(\$oldname,\$newname);
rmdir	删除目录	
mkdir	创建目录	
unlink	删除文件	
copy	复制文件	copy(\$file, \$newfile);
fopen/fputs/fwrite	打开文件或者 URL	https://www.php.net/manual/zh/function.fwrite.php
link	创建文件硬链接	link(\$target, \$link);
symlink	创建符号链接(软链接)	symlink(\$target, \$link);
tmpfile	创建一个临时文件 (在临时目录存放 随机文件名 返回句柄)	\$temp = tmpfile(); fwrite(\$temp, "123456"); fclose(\$temp);
request()->file()->move() request()->file()->file()	Thinkphp 文件上传	\$file = request()->file(\$name); \$file->move(\$filepath);
extractTo	解压ZIP到目录	
DOMDocument loadXML simplexml_import_dom	加载解析XML 有可能存在XXEE 漏洞 file_get_contents 获取客户端输入内容 new DOMDocument()初始化XML 解析器 loadXML(\$xmlfile)加载客户端输入的XML内容simplexml_import_dom(\$dom)获取XML文档节点 如果成功则返回SimpleXMLElement对象，如果失败则返回FALSE。	<?php \$xmlfile = file_get_contents('php://input'); \$dom = new DOMDocument(); \$dom->loadXML(\$xmlfile); \$xml = simplexml_import_dom(\$dom); \$xxe = \$xml->xxe; \$str = "\$xxe \n"; echo \$str;
simplexml_load_string	加载解析XML字符串 有可能存在XXE 漏洞	\$xml=simplexml_load_string(\$_REQUEST['xml']); print_r(\$xml);
simplexml_load_file	读取文件作为XML文档解析 有可能存在XXE 漏洞	
unserialize	反序列化	

PHP原生过滤方法

过滤函数

1. `escapeshellarg` 传入参数添加单引号并转义原有单引号 用于防止命令注入。

例：传入 `' id #` 处理后 `'\ ' id #'` 处理后的字符串可安全的添加到命令执行参数`escapeshellcmd` 转义字符串中的特殊符号 用于防止命令注入

反斜线 (\) 会在以下字符之前插入： `&#;`、```、`|`、`*`、`?`、`~`、`<`、`>`、`^`、`()`、`[]`、`{}`、`$`、`\`，`\x0A` 和 `\xFF`。 `'` 和 `"` 仅在不配对儿的时候被转义

2. `addslashes` 在单引号 (`'`)、双引号 (`"`)、反斜线 (`\`) 与 `NUL`前加上反斜线 可用于防止SQL注入

`PDO::quote` 转义特殊字符 并添加引号

`PDO::prepare` 预处理SQL语句 有效防止SQL注入 (推荐)

`htmlspecialchars` 和 `htmlentities` 将特殊字符转义成html实体 可用于防止XSS

`intval($input)` `floatval()` `floatval()` `floor()` `(int)$input` `num+****0****` 将输入强制转换为整数/浮点 常见于防止SQL注入

命令注入

将用户输入拼接到命令行中执行 导致的任意命令执行问题

例子

```
<?php

$command = 'ping -c 4 ' . $_GET['ip'];

system($command); //system函数特性 执行结果会自动打印
```

执行ping 命令并输出内容正常输入: `/magedu.php?ip=8.8.8.8`

执行命令 `ping -c 4 8.8.8.8`，由于ip参数没有任何过滤限制，所以攻击者可以这样输入: `/magedu.php?ip=8.8.8.8;id`

执行命令 `ping -c 1 8.8.8.8;id`，这样就可以执行攻击者定义命令 `id`

通常在实际审计时输入常常不会非常简单都有复杂的处理，需要慢慢追踪参数来源。

在审计时遇到输入可控时 要检查是否存在`escapeshellarg` `escapeshellcmd` 函数转义 或者是其他的处理方法(如强制类型转换 替换字符 等)

常见bash shell

符号	描述	示例
<、>	输入输出重定向	echo abc >1.txt
分号	按照从左到右顺序执行命令	id;whoami;ls
管道符	将左侧命令的输出作为右侧命令的输入	ps -aux grep root
&&	按照从左到右顺序执行命令 只有执行成功才执行后面的语句	
	按照从左到右顺序执行命令 只有执行失败才执行后面的语句	

代码注入

将用户输入拼接到PHP代码中执行 导致的任意代码执行问题

例如使用 `eval` 等代码执行函数

```
<?php eval($_GET['magedu']);
```

输入: magedu=phpinfo(); 执行 phpinfo() 函数，执行系统命令使用 magedu=system("cmd");

函数/语法结构	描述	例子
eval	将传入的参数内容作为PHP代码执行 eval 不是函数 是一种语法结构 不能当做函数动态调用	eval('phpinfo();');
assert	将传入的参数内容作为PHP代码执行 版本在PHP7以下是函数 PHP7及以上为语法结构	assert('phpinfo();');
preg_replace	当preg_replace使用/e修饰符且原字符串可控时时 有可能执行 php代码	echo preg_replace("/e","\${PHPINFO()}", "123");
call_user_func	把第一个参数作为回调函数调用 需要两个参数都完全可控才可利用 只能传入一个参数调用	call_user_func('assert', 'phpinfo();');
call_user_func_array	同call_user_func 可传入一个数组带入多个参数调用函数	call_user_func_array('file_put_contents', ['1.txt','6666']);
create_function	根据传递的参数创建匿名函数，并为其返回唯一名称 利用需要第二个参数可控 且创建的函数被执行	\$f = create_function("", 'system(\$_GET[123]);'); \$f();

文件包含

将远程/本地文件 包含入当前页面的PHP代码并执行 详细加载执行原理见[PHP7内核剖析](#)

例子

开发人员希望自己写的页面实现更加灵活的加载

```
1 <?php
2 $file = $_GET['page'];
3 include("pages/$file");
4 ?>
```

正常输入 ?page=login.php

```
1 <?php
2 $file = $_GET['page'];
3 /*
4 pages/login.php 文件代码被包含执行
5 */
6 ?>
```

服务器包含并执行pages目录下的login.php 攻击者输入 ?page=../image/123.jpg

服务器包含并执行pages的上层目录image目录下的123.jpg

该漏洞通常需要参数后半部分可控或者参数完全可控才存在

文件包含利用方法

包含上传文件 (上传头像图片等)

包含 data:// php://filter 或 php://input 伪协议 (php.ini allow_url_include 设置为 on) 包含日志 Apache nginx 等 web服务器访问日志 SSH FTP 等登陆错误日志 PHP框架日志

包含 session文件 (通常在临时目录下 (linux /tmp/) sess_会话ID文件) PHP间接或直接创建的其他文件 比如数据库文件 缓存文件 应用日志等

函数/语法结构	描述	例子
include	包含并运行指定文件 执行出错会抛出错误	include 'vars.php'; (括号可有可无)
require	同include 执行出错会抛出警告	require('somefile.php'); (括号可有可无)
require_once	同require 但会检查之前是否已经包含该文件 确保不重复包含	
include_once	同include 但会检查之前是否已经包含该文件 确保不重复包含	

SQL注入

将用户输入拼接到数据库将要执行的SQL语句中 导致攻击者可以修改原有执行的SQL语句

例子

```
<?php

include('conn.php');//数据库连接省略

$sql = "SELECT id, name FROM users WHERE id=$_GET['id']";

$result = $mysqli->query($sql);

if ($result->num_rows > 0) {

    while($row = $result->fetch_assoc()) {

        echo "id: " . $row["id"]. " - Name: " . $row["name"]; 9 }

    } else {

        echo "没有查询到结果"; 12 }

?>
```

正常请求 ?id=123 执行SQL

```
1 SELECT id, name FROM users WHERE id=123;
```

攻击者构造请求:

```
?id=123 UNION SELECT name,password FROM users; 执行SQL
```

```
SELECT id, name FROM users WHERE id=123 UNION SELECT name,password FROM users;
```

攻击者改变了原有的SQL语句逻辑

以下为使用PDO连接MySQL数据库的实例:

```
<?php
$type = 'mysql';
$hostname = '127.0.0.1';
$dbname = 'magedu';
$username = 'root';
$password = 'root';
try {
    $dsn = sprintf('%s:dbname=%s;host=%s;charset=utf8', $type, $dbname, $hostname);
    //初始化一个PDO对象
    $pdo = new PDO($dsn, $username, $password, [
        PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION //开启异常模式
    ]);
} catch (PDOException $e) {
    die ("Database error: " . $e->getMessage());
}
$input_data=1; // 假设input_dta 为用户输入数据
$smt = $pdo->query('SELECT * FROM t_user WHERE id='.$input_dta);
$data = $smt->fetchAll(PDO::FETCH_ASSOC);
var_dump($data);
?>
```

PDO 类:

- [PDO::beginTransaction](#) — 启动一个事务
- [PDO::commit](#) — 提交一个事务
- [PDO::__construct](#) — 创建一个表示数据库连接的 PDO 实例
- [PDO::errorCode](#) — 获取跟数据库句柄上一次操作相关的 SQLSTATE
- [PDO::errorInfo](#) — 返回最后一次操作数据库的错误信息
- [PDO::exec](#) — 执行一条 SQL 语句, 并返回受影响的行数
- [PDO::getAttribute](#) — 取回一个数据库连接的属性
- [PDO::getAvailableDrivers](#) — 返回一个可用驱动的数组
- [PDO::inTransaction](#) — 检查是否在一个事务内
- [PDO::lastInsertId](#) — 返回最后插入行的ID或序列值
- [PDO::prepare](#) — 备要执行的SQL语句并返回一个 PDOStatement 对象

- [PDO::query](#) — 执行 SQL 语句，返回PDOStatement对象,可以理解为结果集
- [PDO::quote](#) — 为SQL语句中的字符串添加引号。
- [PDO::rollBack](#) — 回滚一个事务
- [PDO::setAttribute](#) — 设置属性

PDOStatement 类：

- [PDOStatement::bindColumn](#) — 绑定一列到一个 PHP 变量
- [PDOStatement::bindParam](#) — 绑定一个参数到指定的变量名
- [PDOStatement::bindValue](#) — 把一个值绑定到一个参数
- [PDOStatement::closeCursor](#) — 关闭游标，使语句能再次被执行。
- [PDOStatement::columnCount](#) — 返回结果集中的列数
- [PDOStatement::debugDumpParams](#) — 打印一条 SQL 预处理命令
- [PDOStatement::errorCode](#) — 获取跟上一次语句句柄操作相关的 SQLSTATE
- [PDOStatement::errorInfo](#) — 获取跟上一次语句句柄操作相关的扩展错误信息
- [PDOStatement::execute](#) — 执行一条预处理语句
- [PDOStatement::fetch](#) — 从结果集中获取下一行
- [PDOStatement::fetchAll](#) — 返回一个包含结果集中所有行的数组
- [PDOStatement::fetchColumn](#) — 从结果集中的下一行返回单独的一列。
- [PDOStatement::fetchObject](#) — 获取下一行并作为一个对象返回。
- [PDOStatement::getAttribute](#) — 检索一个语句属性
- [PDOStatement::getColumnMeta](#) — 返回结果集中一列的元数据
- [PDOStatement::nextRowset](#) — 在一个多行集语句句柄中推进到下一个行集
- [PDOStatement::rowCount](#) — 返回受上一个 SQL 语句影响的行数
- [PDOStatement::setAttribute](#) — 设置一个语句属性
- [PDOStatement::setFetchMode](#) — 为语句设置默认的获取模式。

PHP PDO 预处理语句与存储过程

```
<?php
// insert
$stmt = $dbh->prepare("INSERT INTO REGISTRY (name, value) VALUES (:name, :value)");
$stmt->bindParam(':name', $name);
$stmt->bindParam(':value', $value);

// 插入一行
$name = 'one';
$value = 1;
$stmt->execute();

// select
$stmt = $dbh->prepare("SELECT * FROM REGISTRY where name = ?");
if ($stmt->execute(array($_GET['name']))) {
    while ($row = $stmt->fetch()) {
        print_r($row);
    }
}
?>
```

使用命名 (:name) 参数来准备SQL语句

实例

```
<?php
/* 通过数组值向预处理语句传递值 */
$sql = 'SELECT name, colour, calories
      FROM fruit
      WHERE calories < :calories AND colour = :colour';
$stmt = $dbh->prepare($sql, array(PDO::ATTR_CURSOR => PDO::CURSOR_FWDONLY));
$stmt->execute(array(':calories' => 150, ':colour' => 'red'));
$red = $stmt->fetchAll();
$stmt->execute(array(':calories' => 175, ':colour' => 'yellow'));
$yellow = $stmt->fetchAll();
?>
```

使用问号 (?) 参数来准备SQL语句

实例

```
<?php
/* 通过数组值向预处理语句传递值 */
$stmt = $dbh->prepare('SELECT name, colour, calories
      FROM fruit
      WHERE calories < ? AND colour = ?');
$stmt->execute(array(150, 'red'));
$red = $stmt->fetchAll();
$stmt->execute(array(175, 'yellow'));
$yellow = $stmt->fetchAll();
?>
```

Order by涉及到动态表名和列名时，不能用参数绑定方式处理，如果强制绑定，会没有排序效果，

如Order by age。建议使用白名单参数形式：

```
orders=array("name","age","magedu");
$key=array_search($_GET['data'],$orders); // 查询是否存在数组中
$order=$orders[$key];
$query="SELECT * from table WHERE is_live = :is_live ORDER BY $order";
```

MySQLi 函数

函数	描述
mysqli_affected_rows()	返回前一次 MySQL 操作所影响的记录行数。
mysqli_autocommit()	打开或关闭自动提交数据库修改。
mysqli_change_user()	更改指定数据库连接的用户。
mysqli_character_set_name()	返回数据库连接的默认字符集。
mysqli_close()	关闭先前打开的数据库连接。
mysqli_commit()	提交当前事务。
mysqli_connect_errno()	返回上一次连接错误的错误代码。
mysqli_connect_error()	返回上一次连接错误的错误描述。
mysqli_connect()	打开一个到 MySQL 服务器的新的连接。
mysqli_data_seek()	调整结果指针到结果集中的一个任意行。
mysqli_debug()	执行调试操作。
mysqli_dump_debug_info()	转储调试信息到日志中。
mysqli_errno()	返回最近调用函数的最后一个错误代码。
mysqli_error_list()	返回最近调用函数的错误列表。
mysqli_error()	返回最近调用函数的最后一个错误描述。
mysqli_fetch_all()	从结果集中取得所有行作为关联数组，或数字数组，或二者兼有。
mysqli_fetch_array()	从结果集中取得一行作为关联数组，或数字数组，或二者兼有。
mysqli_fetch_assoc()	从结果集中取得一行作为关联数组。
mysqli_fetch_field_direct()	从结果集中取得某个单一字段的 meta-data，并作为对象返回。
mysqli_fetch_field()	从结果集中取得下一字段，并作为对象返回。
mysqli_fetch_fields()	返回结果中代表字段的对象的数组。
mysqli_fetch_lengths()	返回结果集中当前行的每个列的长度。
mysqli_fetch_object()	从结果集中取得当前行，并作为对象返回。
mysqli_fetch_row()	从结果集中取得一行，并作为枚举数组返回。
mysqli_field_count()	返回最近查询的列数。
mysqli_field_seek()	把结果集中的指针设置为指定字段的偏移量。

mysqli_field_tell()	返回结果集中的指针的位置。
mysqli_free_result()	释放结果内存。
mysqli_get_charset()	返回字符集对象。
mysqli_get_client_info()	返回 MySQL 客户端库版本。
mysqli_get_client_stats()	返回有关客户端每个进程的统计。
mysqli_get_client_version()	将 MySQL 客户端库版本作为整数返回。
mysqli_get_connection_stats()	返回有关客户端连接的统计。
mysqli_get_host_info()	返回 MySQL 服务器主机名和连接类型。
mysqli_get_proto_info()	返回 MySQL 协议版本。
mysqli_get_server_info()	返回 MySQL 服务器版本。
mysqli_get_server_version()	将 MySQL 服务器版本作为整数返回。
mysqli_info()	返回有关最近执行查询的信息。
mysqli_init()	初始化 MySQLi 并返回 mysqli_real_connect() 使用的资源。
mysqli_insert_id()	返回最后一个查询中自动生成的 ID。
mysql_kill()	请求服务器杀死一个 MySQL 线程。
mysqli_more_results()	检查一个多查询是否有更多的结果。
mysqli_multi_query()	执行一个或多个针对数据库的查询。
mysqli_next_result()	为 mysqli_multi_query() 准备下一个结果集。
mysqli_num_fields()	返回结果集中字段的数量。
mysqli_num_rows()	返回结果集中行的数量。
mysqli_options()	设置额外的连接选项，用于影响连接行为。
mysqli_ping()	进行一个服务器连接，如果连接已断开则尝试重新连接。
mysqli_prepare()	准备执行一个 SQL 语句。
mysqli_query()	执行某个针对数据库的查询。
mysqli_real_connect()	打开一个到 MySQL 服务器的新的链接。
mysqli_real_escape_string()	转义在 SQL 语句中使用的字符串中的特殊字符。
mysqli_real_query()	执行 SQL 查询
mysqli_reap_async_query()	返回异步查询的结果。
mysqli_refresh()	刷新表或缓存，或者重置复制服务器信息。

mysqli_rollback()	回滚数据库中的当前事务。
mysqli_select_db()	更改连接的默认数据库。
mysqli_set_charset()	设置默认客户端字符集。
mysqli_set_local_infile_default()	撤销用于 load local infile 命令的用户自定义句柄。
mysqli_set_local_infile_handler()	设置用于 LOAD DATA LOCAL INFILE 命令的回滚函数。
mysqli_sqlstate()	返回最后一个 MySQL 操作的 SQLSTATE 错误代码。
mysqli_ssl_set()	用于创建 SSL 安全连接。
mysqli_stat()	返回当前系统状态。
mysqli_stmt_init()	初始化声明并返回 mysqli_stmt_prepare() 使用的对象。
mysqli_store_result()	返回的当前的结果集。
mysqli_thread_id()	返回当前连接的线程 ID。
mysqli_thread_safe()	返回是否将客户端库编译成 thread-safe。
mysqli_use_result()	从上次使用 mysqli_real_query() 执行的查询中初始化结果集的检索。
mysqli_warning_count()	返回连接中的最后一个查询的警告数量。

执行原生sql代码示例

```
<?php
// 假定数据库用户名: root, 密码: 123456, 数据库: RUNOOB
$con=mysqli_connect("localhost","root","123456","RUNOOB");
if (mysqli_connect_errno($con))
{
    echo "连接 MySQL 失败: " . mysqli_connect_error();
}

// 执行查询
mysqli_query($con,"SELECT * FROM websites");
mysqli_query($con,"INSERT INTO websites (name, url, alexa, country)
VALUES ('百度','https://www.baidu.com/','4','CN')");

mysqli_close($con);
?>
```

预编译sql代码示例

预编译三个步骤为: 1.\$stmt -> prepare. 2.\$stmt -> bind_param. 3.\$stmt -> execute

```

<?php

mysqli_report(MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT);
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

$city = "Amersfoort";

/* create a prepared statement */
$stmt = $mysqli->prepare("SELECT District FROM City WHERE Name=?");

/* bind parameters for markers */
$stmt->bind_param("s", $city);

/* execute query */
$stmt->execute();

/* bind result variables */
$stmt->bind_result($district);

/* fetch value */
$stmt->fetch();

printf("%s is in district %s\n", $city, $district);

```

mysql 扩展

mysql扩展没有预编译机制，容易存在sql注入

```

<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
{
    die('Could not connect: ' . mysql_error());
}

mysql_select_db("my_db", $con);

$result = mysql_query("SELECT * FROM Persons");

while($row = mysql_fetch_array($result))
{
    echo $row['FirstName'] . " " . $row['LastName'];
    echo "<br />";
}

mysql_close($con);
?>

```


常见过滤/防护

addslashes 在单引号 (')、双引号 (")、反斜线 (\) 与 NULL 前加上反斜线 可用于防止SQL注入

防护方法

```
<?php

/*强制类型转换*/

$id=intval($_GET['id']); //因查询ID为整数 所以可以强制转换为整数

/*转义特殊字符 加上引号 (字符串类型)*/

$id=$pdo->quote($_GET['name']);

/*预处理语句*/

$stmt = $pdo->prepare("SELECT id, name FROM users WHERE id=?");

$stmt->execute([$_GET['id']]); //简单的预处理 完整使用方法见PHP手册

?>
```

PDO::quote 转义特殊字符 并添加引号

PDO::prepare 预处理SQL语句 有效防止SQL注入 (推荐) intval(\$input) floatval() floatval() floor() (int)\$input num

将输入强制转换为整数/浮点 用于整数/浮点类型的输入参数处理 可防止SQL注入

一些执行SQL语句的函数

函数/方法	备注
mysql_query	
odbc_exec	
mysqli_query	
mysql_db_query	
mysql_unbuffered_query	
mysqli::query用法 <code>\$mysqli = new mysqli("localhost", "my_user", "my_password", "world"); \$mysqli->query();</code>	
pg_query	
pg_query_params	
pg_send_query	
pg_send_query_params	
sqldrvc_query	
pdo::query <code>\$pdo=new PDO("mysql:host=localhost;dbname=phpdemo","root","1234"); \$pdo->query(\$sql);</code>	PDO
SQLite3::query SQLite3::exec <code>\$db = new SQLite3('mysqlitedb.db'); \$db->query('SELECT bar FROM foo'); \$db->exec('CREATE TABLE bar (bar STRING);</code>	

文件操作

文件操作相关关键参数用户可控 导致文件/目录 删除/移动/写入(上传)/读取等

文件/目录删除

- 1 unlink(文件路径)//删除文件
- 2 rmdir(文件夹路径)//删除目录

攻击者常见用法

- 删除lock文件(解除重复程序安装保护等安全限制) 删除网站关键文件(导致网站拒绝服务 数据丢失)

文件写入/上传

文件写入

```
file_put_contents ( 路径, 写入字符串 ); // 直接将字符串写入文件 ( 不存在会自动创建 ) 3

$fp = fopen ( 文件路径, "w" ); // 以写入模式打开一个文件 返回文件指针 ( 不存在会自动创建 )
```

```
fwrite($fp,写入字符串);//写入数据
```

```
fclose($fp);//关闭文件
```

文件上传

```
move_uploaded_file(临时上传文件路径,目标文件路径);//移动临时上传文件
```

php的原生文件上传

- 收到POST表单->随机文件名写入临时目录->(执行PHP文件处理逻辑->移动临时文件到保存位置)->删除临时文件(如果临时文件没有被移动)
- 临时文件路径必须是php上传表单自动处理产生的 例如 `$_FILES["pictures"]["tmp_name"]`
- `"pictures"` 为表单中的 `name` ; `"tmp_name"` 为固定变量名(临时文件名)

注:只要PHP收到POST上传文件表单 哪怕php页面一行代码没有 都会将上传文件保存到临时目录 在请求结束后如果临时文件没有被移走就会被自动删除 从写入文件到删除文件有个短暂的窗口时间 可用于文件包含

文件解压

```
$zip = new \ZipArchive;
```

```
$zip->open('test_new.zip', \ZipArchive::CREATE) //打开一个zip文件
```

```
$zip->addFile('test.txt');//添加压缩文件
```

```
$zip->addEmptyDir('newdir');//添加空目录
```

```
$zip->addFromString('new.txt', '文本');//从字符串添加文件到压缩包
```

```
$zip->extractTo('upload');//将压缩包文件解压到upload目录下
```

```
$zip->close();//关闭zip
```

注:ZipArchive扩展在windows平台 php版本>5.6时默认安装。 linux及windows其他版本需要手动编译安装。

审计时重点查找 extractTo方法

判断解压目录是否在web目录下 是否检查压缩包内文件类型 如果不在web目录下也可以使用 `..` 进行目录穿越控制 上传目录 到web目录下 或者在权限足够的情况下写入文件到系统关键目录

文件写入/上传

函数	描述	例子
file_put_contents	将一个字符串写入文件	file_put_contents("1.txt","666 6");
move_uploaded_file	将上传的临时文件移动到新的位置	move_uploaded_file(\$_FILES["pictures"] ["tmp_name"],"1.php")
rename	重命名文件/目录	rename(\$oldname,\$newname);
rmdir	删除目录	
mkdir	创建目录	
unlink	删除文件	
copy	复制文件	copy(\$file, \$newfile);
fopen/fputs/fwrite	打开文件或者 URL	https://www.php.net/manual/zh/function.fwrite.php
link	创建文件硬链接	link(\$target, \$link);
symlink	创建符号链接(软链接)	symlink(\$target, \$link);
tmpfile	创建一个临时文件 (在临时目录存放 随机文件名 返回句柄)	\$temp = tmpfile(); fwrite(\$temp, "123456"); fclose(\$temp);
request()->file()->move()request()->file()->file()	Thinkphp 文件上传	\$file = request()->file(\$name);\$file->move(\$filepath);
extractTo	解压zip 到目录	

文件读取

函数	描述	例子
file_get_contents	读入文件返回字符串	echo file_get_contents("flag.txt"); echo file_get_contents("https://www.bilibili.com/");
readfile	读取一个文件，并写入到输出缓冲	同file_get_contents
fopen/fread/fgets/fgetss/fgetc/fgetcsv/fpassthru/fscanf	打开文件或者 URL 读取文件流	\$file = fopen("test.txt","r"); echo fread(\$file,"1234"); fclose(\$file);
file	把整个文件读入一个数组中	echo implode(" ", file(https://www.bilibili.com/));
highlight_file/show_source	语法高亮一个文件	highlight_file("1.php");
parse_ini_file	读取并解析一个ini配置文件	print_r(parse_ini_file('1.ini'));
simplexml_load_file	读取文件作为XML文档解析	

全部文件操作函数可参考PHP官方手册 <https://www.php.net/manual/zh/ref.filesystem.php>

XSS

跨站脚本(攻击) 让用户浏览器执行到攻击者指定的JS脚本代码

XSS

反射型XSS

反射型XSS是服务器后端处理时把处理不当的用户输入输出到网页 导致用户浏览器执行恶意代码

```
<?php

echo 'Hello ' . $_GET['name'] . '!' ;
```

在控制器活模板中直接echo

正常输入: ?name=magedu 服务器返回 Hello magedu!

浏览器渲染纯文本 Hello magedu!

```
攻击者输入：
?name=<script type="text/javascript">alert('XSS!');</script>

服务器返回
Hello <script type="text/javascript">alert('XSS!');</script>!

浏览器渲染 Hello ! script被作为html标签解析 执行其中的代码 弹出警告框xss!
```

此种漏洞比较明显 很容易分析问题的存在审计时注意 PHP常使用 htmlspecialchars 和 htmlentities函数 转义用户的输入作为防护

非前后端分离框架审计时，可以关注templet、view等含有视图关键字的目录

可以搜索所有php 输出功能的函数

php输出函数

函数名	功能描述
echo()	输出字符串
print()	输出一个或多个字符串
print_r()	打印关于变量的易于理解的信息
printf()	输出格式化字符串
sprintf()	把格式化的字符串写入一个变量中
var_dump()	输出变量的内容、类型或字符串的内容、类型、长度
die()	输出一条消息，并退出当前脚本

查看输出函数是否将用户输入直接输出到前端页面当中

储存型XSS

将服务器储存的处理不当的用户输入输出到网页 导致用户浏览器执行恶意代码

- 攻击者输入->服务器储存 攻击者得到一个返回储存值的页面
- 被害者请求页面->服务器调用储存并输出->XSS 常见于评论 留言 文章 等

比如:

- 发布评论 攻击者发布带有恶意代码的评论 被害者访问评论展示页面 触发XSS 后台管理员审核评论时 触发XSS
- 用户系统设置昵称时 攻击者将昵称写入数据库 在评论显示时读取数据库值输出用户昵称

前端外部文件引用

攻击者修改前端引用的文件链接 引用外部网站文件常见于**用户头像 文章/评论图片**等

被害者访问到攻击者个人页面**文章 评论 聊天内容**时会访问远程图片文件

- 这可能会使攻击者获取到访问者的ip 浏览器 系统等信息
- 也可以绕过内容审查 在审查通过后动态替换文件内容(hsbc广告等信息)
 - 解决方法:正则匹配限制url域名

防护

后端过滤

- 服务端返回HTTP头 添加内容安全策略 [content-security-policy](#) 头
- 正确设置安全策略可以有效减少未知XSS/html外部文件引用漏洞产生的危害
- COOKIES添加Httponly属性 防止使用js读取用户cookies (js发起表单仍可携带cookies)
- `htmlspecialchars()`将特殊字符转换为 **HTML 实体**,注意默认不转义单引号,需设置`ENT_QUOTES`常量

PHP设置csp

可以使用PHP `header()` 方法设置

```
<?php
    header("Content-Security-Policy: default-src 'self'");
?>
```

[CSP安全策略参考手册](#)

CSRF

跨站请求伪造

CSRF

表单请求

攻击者使被害者的浏览器在用户不知情的情况下发起目标网站表单请求 这些表单常常带有目标网站的用户cookies 可以以用户在目标网站的身份进行操作 (攻击者不能获取cookies)

检查鉴权后的操作是否添加token/Referrer校验 拒绝空Referrer

JSONP请求

除了表单常见的还有jsonp请求

服务器返回一段带有函数调用的json 浏览器把jsonp页面当做js加载执行调用回调函数将数据传入函数

用于浏览器从服务器动态获取信息 由于js等静态资源调用浏览器默认放行,造成了风险

可获取用户登陆后才能获取的信息, 比如登陆用户个人资料、账户余额 等

防护方法

添加随机token 在表单/jsonp请求时附加token(非常有效)

服务端检测 Referrer (一定要拒绝空Referrer html表单可以发起空referrer)(表单/静态资源引用/jsonp 请求)

如果使用正则匹配一定要检查正则是否可以被绕过

服务器返回Access-Control-Allow-Origin 头 (表单/静态资源引用/jsonp请求无效 仅AJAX请求)(一定不要设置成 *)

1.启动会话并生成一个随机令牌。

```
session_start();
$_SESSION["token"] = bin2hex(random_bytes(32));
```

2.将 CSRF 令牌嵌入到 HTML 表单中。

```
<input type="hidden" name="token" value="<?=$_SESSION["token"]?>" />
```

3.提交表单后, 将提交的令牌与会话进行交叉检查。

```
if (!isset($_POST["token"]) || !isset($_SESSION["token"])) {
    exit();
}
if ($_POST["token"] == $_SESSION["token"]) {
    DO PROCESSING
}
```

XXE

XML外部实体(注入) 攻击者利用xml的性质可以获取本地/远程文件内容 (不同于其他语言 PHP 中xml实体可以使用PHP伪协议)

XXE

XML外部实体是XML的一个特性 XML可以使用外部实体引用来包含和解析其他文档当然XML还有其他实体 详细内容可以参考[这个DTD教程](#)

这里就不详细将利用技巧了

审计时如果发现使用了文末列表的函数 就要检查是否禁用了外部实体

```
libxml_disable_entity_loader(true); //禁用外部实体使用到的函数 参数为true时禁用
```

注意: php环境中libxml 版本>=2.9.0时外部实体默认禁用

漏洞常见处:

函数	描述	
DOMDocument::loadXML	加载解析XML	<pre><?php\$xml=file_get_contents('php://input'); \$dom=new DOMDocument(); \$dom->loadXML(\$xml); \$xml=simplexml_import_dom(\$dom); \$xxe=\$xml->xxe; echo \$xxe; ?></pre>
simplexml_load_string	加载解析XML字符串	<pre>\$xml=simplexml_load_string(\$_REQUEST['xml']); print_r(\$xml);</pre>
simplexml_load_file	读取文件作为XML文档解析	<pre>simplexml_load_file("1.xml")</pre>

Xml parser

```
<?php
    $simple = "simple note";
    $p = xml_parser_create();
    xml_parse_into_struct($p, $simple, $vals, $index);
    xml_parser_free($p);
    echo "Index array\n";
    print_r($index);
    echo "\nVals array\n";
    print_r($vals);
```

如果要防止php中的xxe漏洞，可以在simplexml_load_string前加上一句

```
libxml_disable_entity_loader(true);禁止从外部加载XML实体
```

libxml_disable_entity_loader - 禁用加载外部实体的功能

描述

```
bool libxml_disable_entity_loader ([ bool $disable = true ] )
```

禁用/启用加载外部实体的功能。

参数

`disable`

禁用（**TRUE**）或启用（**FALSE**）libxml扩展（如[DOM](#)，[XMLWriter](#)和[XMLReader](#)）来加载外部实体。

反序列化

序列化

函数: `serialize`

将PHP中的值转化为一个字符串 有利于存储或传递 PHP 的值 同时不丢失其类型和结构如果被序列化的值是一个对象

反序列化

函数: `unserialize`

- `__construct()` 当一个对象创建时被调用
- `__destruct()` 当一个对象销毁前被调用
- `__sleep()` 在对象被序列化前被调用
- `__wakeup` 将在反序列化之后立即被调用

反序列化在序列化操作后产生的字符串 还原序列化前的值

如果被反序列化的结果包含对象 则会调用对应对象的`__wakeup` 魔法函数

如果反序列化的字符串被用户可控 攻击者则有可能利用PHP中现有的对象调用对应魔法函数进行攻击(主要看对象定义的魔法函数所拥有的功能)

```
<?php

class magedu2
{
    var $id = '1';

    function __wakeup()
    {
        eval($this->id);
    }
}

$magedu1 = new magedu2();
print_r(serialize($magedu1));
```

```
unserialize('O:7:"magedu2":1:{s:2:"id";s:10:"phpinfo()";}');
```

其他漏洞

逻辑漏洞

业务相关漏洞是非常灵活的一类漏洞

多挖一些互联网公司SRC的业务漏洞，在挖掘达到一定数量后，会对业务漏洞会有更深的理解

越权漏洞

越权是指用户可以进行超出业务设计上的权限限制的访问/操作
平行越权: 访问/操作与当前用户同等权限的其他用户的数据

例:

普通用户A可以删除普通用户B投稿的文章 从业务逻辑设计上用户A只能删除自己创建的稿件 但是在后端未做相应的限制

垂直越权: 访问/操作与当前用户未拥有权限的数据

例: 日志审计管理员 设计上可以审计服务器日志 但不能进行其他操作 系统管理员可以进行 系统配置 管理账号等操作

如果使用日志审计管理员账号可以进行系统管理员的系统配置操作 这就是一个垂直越权漏洞非常灵活 具体细节要结合业务功能进行判断是否存在问题

未授权访问/无鉴权/鉴权绕过

访问/操作业务前未进行权限检查 或权限检查不严易被绕过

比如上文的越权漏洞

如果用户未进行登陆操作仍然可以访问后台页面获取数据/进行后台操作 这就是未授权访问(未授权访问>垂直越权)

有些后台操作为了方便添加后门 采用了弱校验(硬编码token 从head获取的UA和IP地址 弱JWT秘钥等) 这就可能造成鉴权绕过

频率限制

在一些关键业务点应做频率限制

例如 用户登陆 使用ip/用户名限制单位时间内登陆次数 必要时增加多次失败锁定限制 (可以用例如memcache redis等缓存机制 记录次数 限制频率)配合验证码进行限制 防止账号密码爆破

用户回复 发表文章 订阅关注功能 发起订单 等，也应增加频率和次数限制，防止大量填充垃圾信息，频率过高时增加验证码校验

邮箱/短信验证码 加上单日次数及频率限制 必要时增加验证码校验

拒绝服务

通过特殊的用户输入 消耗的服务器资源 比如生成图片不限宽高(二维码 验证码 图片处理)

URL跳转

跳转到网站外第三方链接 可用于钓鱼 如果存在head注入则可以构成xss

PHP 框架

Laravel

介绍

默认的 Laravel 应用程序结构旨在为大型和小型应用程序提供一个很好的起点。但是您可以随意组织您的应用程序。Laravel 对任何给定类的位置几乎没有任何限制——只要 Composer 可以自动加载该类。

根目录

App 目录

`app` 目录包含应用程序的核心代码。我们将很快更详细地探索这个目录；但是，您应用程序中的几乎所有类都将在此目录中。

Bootstrap 目录

`bootstrap` 目录包含启动框架的 `app.php` 文件。该目录还包含一个“缓存”目录，其中包含用于性能优化的框架生成文件，例如路由和服务缓存文件。您通常不需要修改此目录中的任何文件。

Config 目录

顾名思义，`config` 目录包含应用程序的所有配置文件。最好把这些文件都浏览一遍，并熟悉所有可用的选项。

Database 目录

`database` 目录包含数据库迁移，模型工厂和种子生成器文件。如果你愿意，你还可以把它作为 SQLite 数据库存放目录。

Lang 目录

`lang` 目录包含应用程序的所有语言文件。

Public 目录

`public` 目录包含 `index.php` 文件，该文件是进入你应用程序的所有请求的入口，并配置自动加载。该目录还包含你的资源，如图像、JavaScript 脚本和 CSS 样式。

Resources 目录

`resources` 目录包含了 [views](#) 以及未编译的资源文件（如 CSS 或 JavaScript）。此目录还包含所有的语言文件。

Routes 目录

`routes` 目录包含应用程序的所有路由定义。默认情况下，Laravel 包含几个路由文件：

`web.php`，`api.php`，`console.php` 以及 `channels.php`。

`web.php` 文件包含 `RouteServiceProvider` 放置在 `web` 中间件组中的路由，该中间件组提供会话状态，CSRF 保护和 cookie 加密，如果你的应用程序不提供无状态的 RESTful API，那么你的所有路由都很可能在 `web.php` 文件。

`api.php` 文件包含 `RouteServiceProvider` 放置在 `api` 中间件组中的路由。这些路由旨在是无状态的，因此通过这些路由进入应用程序的请求旨在[通过令牌]（/cndocs/9.x/sanctum）进行身份验证，并且无法访问会话状态。

`console.php` 文件是您可以定义所有基于闭包的控制台命令的地方。每个闭包都绑定到一个命令实例，允许一种简单的方法与每个命令的 IO 方法进行交互。即使此文件没有定义 HTTP 路由，它也会定义应用程序中基于控制台的入口点（路由）。

`channels.php` 文件是您可以注册应用程序支持的所有 [事件广播](#) 频道的地方。

Storage 目录

`storage` 目录包含你的日志、编译的 Blade 模板、基于文件的会话、文件缓存和框架生成的其他文件。该目录分为“app”、“framework”和“logs”目录。`app` 目录可用于存储应用程序生成的任何文件。`framework` 目录用于存储框架生成的文件和缓存。最后，`logs` 目录包含应用程序的日志文件。

`storage/app/public` 目录可用于存储用户生成的文件，例如个人资料头像，这些文件应该可以公开访问。你应该在 `public/storage` 创建一个指向这个目录的符号链接。您可以使用 `php artisan storage:link` Artisan 命令创建链接。

Tests 目录

`tests` 目录包含您的自动化测试。开箱即用的示例 [PHPUnit](#) 单元测试和功能测试。每个测试类都应以单词“Test”作为后缀。您可以使用 `phpunit` 或 `php vendor/bin/phpunit` 命令运行测试。或者，如果您想要更详细和更漂亮的测试结果表示，您可以使用 `php artisan test` Artisan 命令运行测试。

Vendor 目录

`vendor` 目录包含您的 [Composer](#) 依赖项。

注意

请确保，Web 服务器将所有请求定向到项目目录的 `public/index.php` 文件。不应该将 `index.php` 文件移动到项目的根目录，因为从项目根目录提供应用程序会将许多敏感配置文件暴露到公网

入口

Laravel 应用程序的所有请求的入口点都是 `public/index.php` 文件。所有请求都由你的 `web` 服务器（Apache/Nginx）配置定向到此文件。那个 `index.php` 文件不包含太多代码。相反，它是加载框架其余部分的起点。

该 `index.php` 文件将加载 `Composer` 生成的自动加载器定义，然后从 `bootstrap/app.php` 中检索 `Laravel` 应用程序的实例。`Laravel` 本身采取的第一个操作是创建应用 / [服务容器](#) 的实例。

基本路由

最基本的Laravel路由接受一个 URI 和一个闭包，提供了一个简单优雅的方法来定义路由和行为，而不需要复杂的路由配置文件：

```
use Illuminate\Support\Facades\Route;

Route::get('/greeting', function () {
    return 'Hello World';
});
```

默认路由文件

所有Laravel路由都定义在你的路由文件中，它位于 `routes` 目录。这些文件会被你的应用程序中的 `App\Providers\RouteServiceProvider` 自动加载。`routes/web.php` 文件用于定义 `web` 界面的路由。这些路由被分配给 `web` 中间件组，它提供了 SESSION 状态和 CSRF 保护等功能。定义在 `routes/api.php` 中的路由都是无状态的，并且被分配了 `api` 中间件组。

对于大多数应用程序，都是以在 `routes/web.php` 文件定义路由开始的。可以通过在浏览器中输入定义的路由 URL 来访问 `routes/web.php` 中定义的路由。例如，你可以在浏览器中输入 `http://example.com/user` 来访问以下路由：

```
use App\Http\Controllers\UserController;

Route::get('/user', [UserController::class, 'index']);
```

定义在 `routes/api.php` 文件中的路由是被 `RouteServiceProvider` 嵌套在一个路由组内。在这个路由组内，将自动应用 `/api` URI 前缀，所以你无需手动将其应用于文件中的每个路由。你可以通过修改 `RouteServiceProvider` 类来修改前缀和其他路由组选项。

可用的路由方法

路由器允许你注册能响应任何 HTTP 请求的路由：

```
Route::get($uri, $callback);
Route::post($uri, $callback);
Route::put($uri, $callback);
Route::patch($uri, $callback);
Route::delete($uri, $callback);
Route::options($uri, $callback);
```

有的时候你可能需要注册一个可响应多个 HTTP 请求的路由，这时你可以使用 `match` 方法，也可以使用 `any` 方法注册一个实现响应所有 HTTP 请求的路由：

```
Route::match(['get', 'post'], '/', function () {
    //
});

Route::any('/', function () {
    //
});
```

技巧：当定义多个相同路由时，使用 `get`，`post`，`put`，`patch`，`delete`，和 `options` 方法的路由应该在使用 `any`，`match`，和 `redirect` 方法的路由之定义，这样可以确保请求与正确的路由匹配。

依赖注入

你可以在路由的回调方法中，以形参的方式声明路由所需要的任何依赖项。这些依赖会被 Laravel 的 [容器](#) 自动解析并注入。例如，你可以在闭包中声明 `Illuminate\Http\Request` 类，让当前的 HTTP 请求自动注入依赖到您的路由回调中：

```
use Illuminate\Http\Request;

Route::get('/users', function (Request $request) {
    // ...
});
```

CSRF 保护

请记住，指向 `POST`、`PUT`、`PATCH`、或 `DELETE` 路由的任何 HTML 表单都应该包含一个 CSRF 令牌字段，否则，这个请求将会被拒绝。可以在 [CSRF 文档](#) 中阅读有关 CSRF 更多的信息：

```
<form method="POST" action="/profile">
    @csrf
    ...
</form>
```

重定向路由

如果要定义重定向到另一个 URI 的路由，可以使用 `Route::redirect` 方法。这个方法可以快速的实现重定向，而不再需要去定义完整的路由或者控制器：

```
Route::redirect('/here', '/there');
```

默认情况下，`Route::redirect` 返回 `302` 状态码。你可以使用可选的第三个参数来定制状态码：

```
Route::redirect('/here', '/there', 301);
```

或者，你可以使用 `Route::permanentRedirect` 方法返回 `301` 状态码：

```
Route::permanentRedirect('/here', '/there');
```

注意：在重定向路由中使用路由参数时，以下参数由 Laravel 保留，不能使用：`destination` 和 `status`

视图路由

如果你的路由只需要返回一个 [视图](#)，可以使用 `Route::view` 方法。它和 `redirect` 一样方便，不需要定义完整的路由或控制器。`view` 方法有三个参数，其中前两个是必填参数，分别是 URI 和视图名称。第三个参数选填，可以传入一个数组，数组中的数据会被传递给视图：

```
Route::view('/welcome', 'welcome');

Route::view('/welcome', 'welcome', ['name' => 'Taylor']);
```

注意：当在图路由使用路线参数，下面的参数是由 Laravel 保留，不能使用：`view`，`data`，`status`，和 `headers`

路由参数

必填参数

有时您将需要捕获路由内的 URI 段。例如，您可能需要从 URL 中捕获用户的 ID。您可以通过定义路由参数来做到这一点：

```
Route::get('/user/{id}', function ($id) {
    return 'User '.$id;
});
```

也可以根据您的需要在路由中定义多个参数：

```
Route::get('/posts/{post}/comments/{comment}', function ($postId, $commentId) {
    //
});
```

路由的参数通常都会被放在 `{ }`，并且参数名只能为字母。下划线 (`_`) 也可以用于路由参数名中。路由参数会按路由定义的顺序依次注入到路由回调或者控制器中，而不受回调或者控制器的参数名称的影响。

参数和依赖注入

如果您的路由具有依赖关系，而您希望 Laravel 服务容器自动注入到路由的回调中，则应在依赖关系之后列出路由参数：


```
use Illuminate\Http\Request;

Route::get('/user/{id}', function (Request $request, $id) {
    return 'User '.$id;
});
```

可选参数

有时，您可能需要指定一个路由参数，但您希望这个参数是可选的。您可以在参数后面加上 `?` 标记来实现，但前提是要确保路由的相应变量有默认值：

```
Route::get('/user/{name?}', function ($name = null) {
    return $name;
});

Route::get('/user/{name?}', function ($name = 'John') {
    return $name;
});
```

正则表达式约束

您可以使用路由实例上的 `where` 方法来限制路由参数的格式。 `where` 方法接受参数的名称和定义如何约束参数的正则表达式：

```
Route::get('/user/{name}', function ($name) {
    //
})->where('name', '[A-Za-z]+');

Route::get('/user/{id}', function ($id) {
    //
})->where('id', '[0-9]+');

Route::get('/user/{id}/{name}', function ($id, $name) {
    //
})->where(['id' => '[0-9]+', 'name' => '[a-z]+']);
```

为方便起见，一些常用的正则表达式模式具有帮助方法，可让您快速将模式约束添加到路由：

```
Route::get('/user/{id}/{name}', function ($id, $name) {
    //
})->whereNumber('id')->whereAlpha('name');

Route::get('/user/{name}', function ($name) {
    //
})->whereAlphaNumeric('name');

Route::get('/user/{id}', function ($id) {
    //
})->whereUuid('id');
```

如果传入的请求与路由模式约束不匹配，将返回 404 HTTP 响应。

全局约束

如果您希望路由参数始终受给定正则表达式的约束，您可以使用 `pattern` 方法。您应该在 `App\Providers\RouteServiceProvider` 类的 `boot` 方法中定义这些模式：

```
/**
 * 定义路由模型绑定、模式过滤器等。
 *
 * @return void
 */
public function boot()
{
    Route::pattern('id', '[0-9]+');
```

一旦定义了模式，它就会自动应用到使用该参数名称的所有路由：

```
Route::get('/user/{id}', function ($id) {
    // 仅当{id}是数字时执行。。。
});
```

命名路由

命名路由允许为特定路由方便地生成URL或重定向。通过将 `name` 方法链接到路由定义上，可以指定路由的名称：

```
Route::get('/user/profile', function () {
    //
})->name('profile');
```

您还可以为控制器操作指定路由名称：

```
Route::get(
    '/user/profile',
    [UserProfileController::class, 'show']
)->name('profile');
```

注意：路由名称应始终是唯一的。

生成命名路由的 URL

一旦你为给定的路由分配了一个名字，你可以在通过 Laravel 的 `route` 和 `redirect` 辅助函数生成 URL 或重定向时使用该路由的名称：

```
// 正在生成 URL...
$url = route('profile');

// 正在生成重定向...
return redirect()->route('profile');
```

如果命名路由定义了参数，您可以将参数作为第二个参数传递给 `route` 函数。给定的参数将自动插入到生成的 URL 的正确位置：

```
Route::get('/user/{id}/profile', function ($id) {
    //
})->name('profile');

$url = route('profile', ['id' => 1]);
```

如果您在数组中传递其他参数，这些键/值对将自动添加到生成的 URL 的查询字符串中：

```
Route::get('/user/{id}/profile', function ($id) {
    //
})->name('profile');

$url = route('profile', ['id' => 1, 'photos' => 'yes']);

// /user/1/profile?photos=yes
```

技巧：有时，您可能希望为 URL 参数指定请求范围的默认值，例如当前语言环境。为此，您可以使用 `URL::defaults` 方法。

路由组

路由组允许您共享路由属性，例如中间件，而无需在每个单独的路由上定义这些属性。

嵌套组尝试智能地将属性与其父组“合并”。中间件和 `where` 条件合并，同时附加名称和前缀。URI 前缀中的命名空间分隔符和斜杠会在适当的地方自动添加。

路由中间件

要将 [middleware](#) 分配给组内的所有路由，您可以在定义组之前使用 `middleware` 方法。中间件按照它们在数组中列出的顺序执行：

```
Route::middleware(['first', 'second'])->group(function () {
    Route::get('/', function () {
        // 使用第一个和第二个中间件...
    });

    Route::get('/user/profile', function () {
        // 使用第一个和第二个中间件...
    });
});
```

控制器

如果一组路由都使用相同的 [控制器](#)，您可以使用 `controller` 方法为组内的所有路由定义公共控制器。然后，在定义路由时，您只需要提供它们调用的控制器方法：

```
use App\Http\Controllers\OrderController;

Route::controller(OrderController::class)->group(function () {
    Route::get('/orders/{id}', 'show');
    Route::post('/orders', 'store');
});
```

子域路由

路由组也可以用来处理子域路由。子域可以像路由 uri 一样被分配路由参数，允许您捕获子域的一部分以便在路由或控制器中使用。子域可以在定义组之前调用 `domain` 方法来指定：

```
Route::domain('{account}.example.com')->group(function () {
    Route::get('user/{id}', function ($account, $id) {
        //
    });
});
```

注意：为了确保子域路由是可以访问的，你应该在注册根域路由之前注册子域路由。这将防止根域路由覆盖具有相同 URI 路径的子域路由。

路由前缀

`prefix` 方法可以用给定的 URI 为组中的每个路由做前缀。例如，你可能想要在组内的所有路由 uri 前面加上 `admin` 前缀：

```
Route::prefix('admin')->group(function () {
    Route::get('/users', function () {
        // Matches The "/admin/users" URL
    });
});
```

隐式绑定

Laravel 自动解析定义在路由或控制器操作中的 Eloquent 模型，其类型提示的变量名称与路由段名称匹配。例如：

```
use App\Models\User;

Route::get('/users/{user}', function (User $user) {
    return $user->email;
});
```

由于 `$user` 变量被类型提示为 `App\Models\User` Eloquent 模型，并且变量名称与 `{user}` URI 段匹配，Laravel 将自动注入 ID 匹配相应的模型实例 来自请求 URI 的值。如果在数据库中没有找到匹配的模型实例，将自动生成 404 HTTP 响应。

当然，使用控制器方法时也可以使用隐式绑定。同样，请注意 `{user}` URI 段与控制器中的 `$user` 变量匹配，该变量包含 `App\Models\User` 类型提示：

```
use App\Http\Controllers\UserController;
use App\Models\User;

// Route definition...
Route::get('/users/{user}', [UserController::class, 'show']);

// Controller method definition...
public function show(User $user)
{
    return view('user.profile', ['user' => $user]);
}
```

定义一个包含 `{user}` 参数的路由：

```
use App\Models\User;

Route::get('/users/{user}', function (User $user) {
    //
});
```

由于我们已将所有 `{user}` 参数绑定到 `App\Models\User` 模型，该类的一个实例将被注入到路由中。因此，例如，对 `users/1` 的请求将从 ID 为 `1` 的数据库中注入 `User` 实例。

如果在数据库中没有找到匹配的模型实例，则会自动生成 404 HTTP 响应。

获取参数

Laravel 通过 `Illuminate\Http\Request` 类来处理用户的请求

传入的请求实例将由 Laravel [服务容器](#) 自动注入

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class UserController extends Controller
{
    /**
     * 存储一个新用户。
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {
        $name = $request->input('name');

        //
    }
}
```

你可以使用 `header` 方法从 `Illuminate\Http\Request` 实例中检索一个请求头。如果请求中不存在该头，将返回 `null`。然而，`header` 方法接受一个可选的第二个参数，如果请求中不存在该头，将返回该参数：

```
$value = $request->header('X-Header-Name');

$value = $request->header('X-Header-Name', 'default');
```

使用一些简单的方法，可以从 `Illuminate\Http\Request` 实例获取所有的用户输入数据，而不用在意用户使用的是哪种 HTTP 动词。不管是什么 HTTP 动词，`input` 方法都可以用来获取用户的输入数据：

```
$name = $request->input('name');
```

可以将默认值作为第二个参数传递给 `input` 方法。如果请求中不存在第一个参数指定的字段的输入值，则将返回此值：

```
$name = $request->input('name', 'Sally');
```

sql注入

原生表达式

要创建原始字符串表达式，可以使用 `DB` facade提供的 `raw` 方法：

```
$users = DB::table('users')
    ->select(DB::raw('count(*) as user_count, status'))
    ->where('status', '<>', 1)
    ->groupBy('status')
    ->get();
```

`selectRaw` 方法可以代替 `addSelect(DB::raw(...))`。该方法的第二个参数是可选项，值是一个绑定参数的数组：

```
$orders = DB::table('orders')
    ->selectRaw('price * ? as price_with_tax', [1.0825])
    ->get();
```

使用命名绑定

除了使用 `?` 表示参数绑定外，你还可以使用命名绑定的形式来执行一个查询：

```
$results = DB::select('select * from users where id = :id', ['id' => 1]);
```

执行 Insert 语句

你可以使用 `DB` Facade 的 `insert` 方法来执行 `insert` 语句。跟 `select` 方法一样，该方法的第一个和第二个参数分别是原生 SQL 语句和绑定的数据：

```
use Illuminate\Support\Facades\DB;

DB::insert('insert into users (id, name) values (?, ?)', [1, 'Marc']);
```

执行 Update 语句

`update` 方法用于更新数据库中现有的记录。该方法将会返回受到本次操作影响的记录行数：

```
use Illuminate\Support\Facades\DB;

$affected = DB::update(
    'update users set votes = 100 where name = ?',
    ['Anita']
);
```

Laravel测试项目

```
git clone https://github.com/crétueusebiu/laravel-web-push-demo.git
cd laravel-web-push-demo
cp .env.example .env
composer install
```