第 1 章

PHP 概述与开发环境搭建

PHP（Hypertext Preprocessor，超文本预处理器）是一种在 Web 开发领域广泛使用的脚本语言。它的起源可以追溯到 1994 年，由 Rasmus Lerdorf（拉斯马斯·勒德尔夫）首次开发，并在 1995 年推出了首个版本。至今，PHP 在 Web 开发技术体系中仍然占据着重要的地位。

本章旨在向读者介绍 PHP 的发展历程以及 PHP 8 带来的新特性。我们假设读者已经具备了 PHP 的基础知识，并且对面向对象编程有所了解。通过本章提供的代码示例及其详细解释，读者将能够快速熟悉 PHP 8 的新功能。同时，我们还将探讨这门语言未来的发展趋势。

掌握一个新的框架需要持续探索和动手实践。我们强烈建议读者亲自动手编写本书提供的代码示例，而不仅仅是复制和粘贴。正如俗话所说的“熟能生巧”，动手实践胜过千遍阅读。

在本章中，我们将介绍以下主要内容：

- PHP的发展历史
- PHP 8的新特性简介
- 安装PHP 8和IDE

1.1 PHP 发展历史

PHP 是一种在 Web 应用开发领域广泛使用的脚本语言。自 20 世纪 90 年代中期诞生至今，PHP 已实现了显著的发展。它从最初的简陋形态，成长为一种卓越的网络编程语言，其发展历程堪称技术爱好者的传奇故事。

以下是 PHP 的一些历史大版本及其特点介绍：

- PHP 1.0（1995年）：作为PHP的起始版本，它具备了基础的CGI功能，主要用于处理表单数据和生成简单的动态网页。
- PHP 2.0（1997年）：此版本增添了新特性，包括数据库连接和正则表达式的支持，使得PHP更为强大和灵活。
- PHP 3.0（1998年）：这是一个重要的升级版本，引入了Zend引擎（Zend Engine），提升了

性能和可扩展性。同时，PHP 3.0增加了对更多数据库的支持，如MySQL和PostgreSQL。

- PHP 4.0（2000年）：该版本带来了众多关键特性，包括面向对象编程（OOP）的支持、异常处理、XML解析等，标志着PHP成为一种更为成熟和强大的编程语言。
- PHP 5.0（2004年）：这一重大升级引入了Zend引擎II（Zend Engine 2.0）、命名空间、改进的异常处理和增强的面向对象编程等特性，进一步提高了PHP的性能和功能性。
- PHP 7.0（2015年）：作为PHP的最新主要版本，它带来了显著的特性和性能改进，如引擎的重写、标量类型声明、返回类型声明、匿名类等，使PHP的性能成倍提升，并优化了开发体验。
- PHP 8.0（2020年）：目前最新的主要版本，也是一个关键的迭代。PHP 8引入了众多新特性和改进，包括Just-In-Time（JIT）编译器、属性初始化的简化语法、联合类型（Union Types）。

那么，为什么没有 PHP 6.0 呢？PHP 6 原本是计划中的一个版本，旨在提供完整的 Unicode 支持。然而，由于种种挑战，PHP 6 项目在 2010 年被取消了。尽管如此，PHP 6 中许多提议的功能已在 PHP 5.x 系列中得到实现，尤其是在面向对象编程方面的重要改进。感兴趣的读者可以访问 PHP 6 的维基页面以获取更多详情。

1.2　PHP 8 新特性概述

PHP 8 作为 PHP 语言的一个重要更新版本，于 2020 年 11 月正式推出，这个版本对于 PHP 社区的开发者而言，无疑是一个激动人心的里程碑。该版本引入了多项新特性和优化，其中包括命名参数、注解、match 表达式、nullsafe 运算符、JIT 编译器，以及对类型系统、错误处理和语法一致性的改进。

1. 命名参数（Named Parameters）

PHP 8 新增了命名参数功能，允许开发者在调用函数或方法时，通过参数名称来指定值，而不是依赖于参数的位置。这一特性增强了代码的可读性和可维护性，使得开发者可以只对感兴趣的参数赋值，而忽略其他可选参数。

2. 注解（Attributes）

PHP 8 引入了注解，也称作属性。注解提供了一种将元数据附加到类、属性、方法或函数上的方式。通过定义特定的类并使用#[Annotation]语法，注解可用于文档生成、代码分析和运行时元编程。

3. match表达式

match 表达式是 PHP 8 的一项新特性，它提供了一种更强大的模式匹配语法，类似于 switch 语句。match 表达式支持严格的类型检查和条件匹配，使得代码更加简洁和易于理解。

4. nullsafe运算符

PHP 8 引入了 nullsafe 运算符（?->），用于简化对可能为 null 的变量进行方法调用或属性访问的操作。在之前的版本中，如果要访问一个可能为 null 的对象的属性或方法，需要使用冗长的 null 检查代码。而 nullsafe 运算符允许你在变量为 null 时直接返回 null，而不会引发错误。

5. JIT编译器

PHP 8 引入了 JIT 编译器，这项技术能够在运行时将 PHP 代码编译为机器码，从而显著提升 PHP 代码的执行性能，尤其是在进行密集计算时。

6. 其他新特性

PHP 8 还包含以下新特性：

- 改进了类型系统，包括对联合类型（union types）的原生支持和属性的类型声明。
- 新的错误处理机制，引入了Throwable接口和统一的异常处理。
- 字符串操作的改进，新增了字符串函数和更好的Unicode支持。
- 新增了WeakMap和WeakReference等标准库类。
- 语言和语法的改进，包括匿名类和闭包语法的简化。
- 性能优化，通过内部改进提高了代码执行效率。

对于想要深入了解 PHP 8 编程语言的读者，推荐阅读清华大学出版社出版的《PHP 8 从入门到精通（视频教学版）》一书。

1.3　安装 PHP 8.0

本节简单介绍在 macOS、Ubuntu/Debian、CentOS 和 Windows 系统中搭建 PHP 8.0 开发环境。建议初学者直接使用 Windows 系统快速学习本书内容，跳过其他系统的内容。

1. macOS系统用户

macOS 系统用户可以使用 Homebrew 安装 PHP，命令如下：

```
    brew install php
```

2. Ubuntu/Debian系统用户

Ubuntu/Debian 系统用户可以使用 apt-get 安装 PHP，命令如下：

```
    apt-get install php php-fpm php-curl php-dev php-mbstring php-mysql php-bcmath
```

3. CentOS系统用户

CentOS 系统用户可以使用 yum 安装 PHP，命令如下：

```
    sudo yum install -y https://dl.fedoraproject.org/pub/epel/epel-
release-latest-7. noarch.rpm
```

```
    sudo yum install https://rpms.remirepo.net/enterprise/remi-release-7.rpm
    sudo yum-config-manager --disable 'remi-php*'
    sudo yum-config-manager --enable remi-php80
    sudo yum update
    sudo yum install php
    sudo yum install php
php-{cli,fpm,mysqlnd,zip,devel,gd,mbstring,curl,xml,pear,bcmath,json}
```

4. Windows系统用户

Windows 系统用户可以前往 https://windows.php.net/downloads/releases/archives/下载 PHP 8.0 版本，比如作者下载的文件名为 php-8.0.29-Win32-vs16-x86.zip，在系统当前用户根目录下解压安装包，解压之后把 PHP 目录加入环境变量 Path 中。打开 Windows 终端管理员，查看一下 PHP 的版本号，命令如下：

```
    PS C:\Users\xiayu> php --version
    PHP 8.0.29 (cli) (built: Jun  7 2023 21:23:12) ( ZTS Visual C++ 2019 x86 )
    Copyright (c) The PHP Group
    Zend Engine v4.0.29, Copyright (c) Zend Technologies
    PS C:\Users\xiayu>
```

1.4　安装 IDE

常用的集成开发环境（IDE）包括 PHPStorm 和 Visual Studio Code，开发者可以根据个人偏好和项目需求选择安装。

PHPStorm 是由 JetBrains 公司开发的，它提供了丰富的特性和工具，专门针对 PHP 开发做了优化。PHPStorm 包括代码自动完成、错误检查、代码重构、版本控制集成等功能，非常适合专业 PHP 开发人员。

Visual Studio Code（简称 VS Code）是由微软开发的轻量级 IDE，它免费且可扩展。VS Code 支持多种编程语言，包括 PHP，通过安装相应的 PHP 扩展插件，可以实现类似于 PHPStorm 的编程体验。

读者可以根据自己的开发习惯和需求，选择适合自己的 IDE 进行安装和使用。

1. Visual Studio Code的安装

安装 Visual Studio Code 可以从官方网站 https://code.visualstudio.com/下载 Visual Studio Code，如图 1-1 所示。选择与当前操作系统相对应的版本进行下载。笔者选择下载的文件名是 VSCodeUserSetup-x64-1.91.0.exe。下载完成后，双击安装文件，按照安装向导的提示进行安装即可。

安装完 Visual Studio Code 后，需要再浏览器中打开 https://www.devsense.com/en 链接，在页面上单击“Install”按钮，可自动在 Visual Studio Code 中安装 PHP 扩展，如图 1-2 和图 1-3 所示。PHP 扩展安装完成之后，Visual Studio Code 才能支持 PHP 的代码补全等操作，读者可以自行测试一下。

此处建议读者选用 VS Code 这个免费的集成开发环境来学习和运行本书的示例代码。

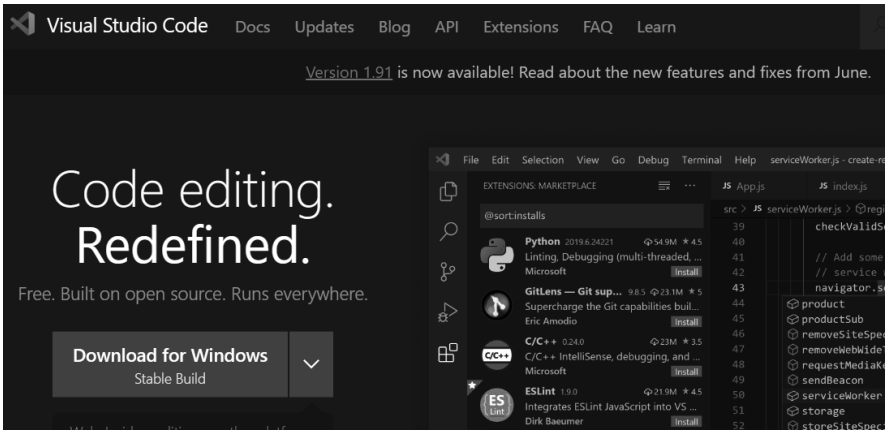
Visual Studio Code
Docs
Updates
Blog
API
Extensions
FAQ
Learn
Version 1.91 is now available! Read about the new features and fixes from June.
Code editing.
Redefined.
Free. Built on open source. Runs everywhere.
Download for Windows
Stable Build

图 1-1

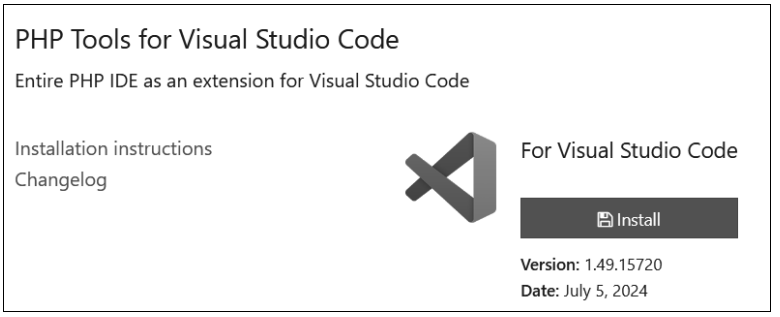
PHP Tools for Visual Studio Code
Entire PHP IDE as an extension for Visual Studio Code
Installation instructions
Changelog
For Visual Studio Code
Install
Version: 1.49.15720
Date: July 5, 2024

图 1-2

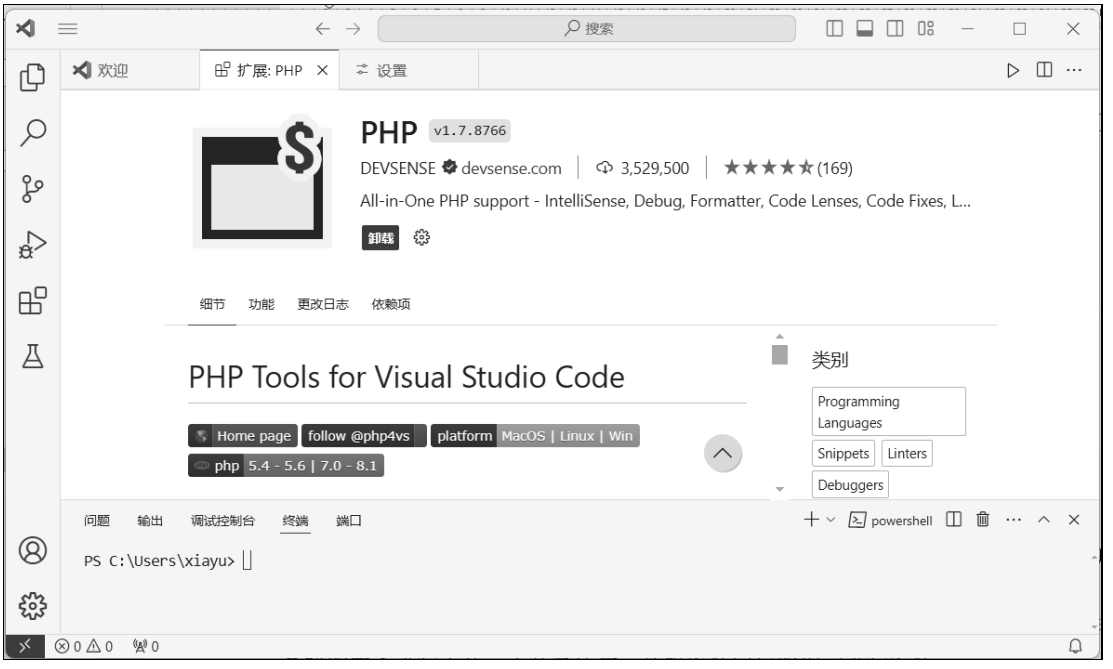
搜索
欢迎
扩展: PHP
设置
PHP
v1.7.8766
DEVSENSE
devsense.com
3,529,500
(169)
All-in-One PHP support - IntelliSense, Debug, Formatter, Code Lenses, Code Fixes, L...
卸载
细节
功能
更改日志
依赖项
PHP Tools for Visual Studio Code
Home page
follow @php4vs
platform
MacOS | Linux | Win
php
5.4 - 5.6 | 7.0 - 8.1
类别
Programming Languages
Snippets
Linters
Debuggers
问题
输出
调试控制台
终端
端口
powershell
PS C:\Users\xiayu>

图 1-3

2. PHPStorm的安装

PHPStorm 是一个专为 PHP 开发者设计的集成开发环境，提供了许多针对 PHP 开发的高级功能和工具，如代码自动完成、调试器、版本控制等（注意，PHPStorm 需要收费使用）。

PHPStorm 可以从 JetBrains 官方网站 https://www.jetbrains.com/phpstorm/download 中选择适合读者当前操作系统的版本进行下载，下载完成后，按照安装向导的指示进行安装即可。

PHPStorm 是开箱即用的，无须安装其他插件即可开始开发。

1.5　验证 PHP 开发环境

本节将使用 PHP 内置的 Web 服务器验证 PHP 是否安装成功。那么，为什么我们不需要安装 Nginx 呢？这是因为从 PHP 5.4 版本开始，PHP 引入了一个非常有用的特性——内置的 Web 服务器。这个特性允许开发者在开发或测试阶段，快速地运行和调试 PHP 应用程序，而无须配置和启动外部 Web 服务器，如 Apache 或 Nginx。

PHP 内置的 Web 服务器是一个简洁、易用的服务器，它可以通过命令行轻松启动。它基于命令行脚本运作，使得在开发环境中模拟 HTTP 请求和响应变得非常方便。该服务器能够处理静态文件和动态 PHP 脚本。当接收到 HTTP 请求时，它会分析请求并将其转发给相应的 PHP 脚本进行进一步处理。此外，它还支持 URL 重写和路由功能，可以根据不同的 URL 路径来分配请求。

然而，需要注意的是，PHP 内置的 Web 服务器仅适用于开发和测试环境。它并不适合用于生产环境，因为它缺乏专业、成熟的 Web 服务器所提供的全面功能和优秀性能。因此，在将应用程序部署到生产环境时，我们仍然建议使用成熟的 Web 服务器，如 Apache 或 Nginx，来提供更强大的功能和性能。

【示例 1-1】在当前用户的根目录下新建 phpinfo.php 文件，命令如下：

```
<?php
phpinfo();
```

打开终端，在该目录执行以下命令开启 PHP 内置 Web 服务器，命令如下：

```
php -S localhost:8080 -t .
```

- -S：表示Web服务器监听地址，localhost表示本地服务器，8080表示服务器端口。
- -t：表示Web应用根目录，“.”表示当前目录。

PHP 内置 Web 服务器启动的效果如图 1-4 所示，注意在图下方的终端窗口中，显示了服务器启动的提示信息。

如果 Web 服务器监听失败，一般情况下是因为端口被占用导致，此时使用其他端口重新执行命令即可。

使用浏览器访问 http://localhost:8080/phpinfo.php 可以看到 phpinfo 相关信息，如图 1-5 所示。

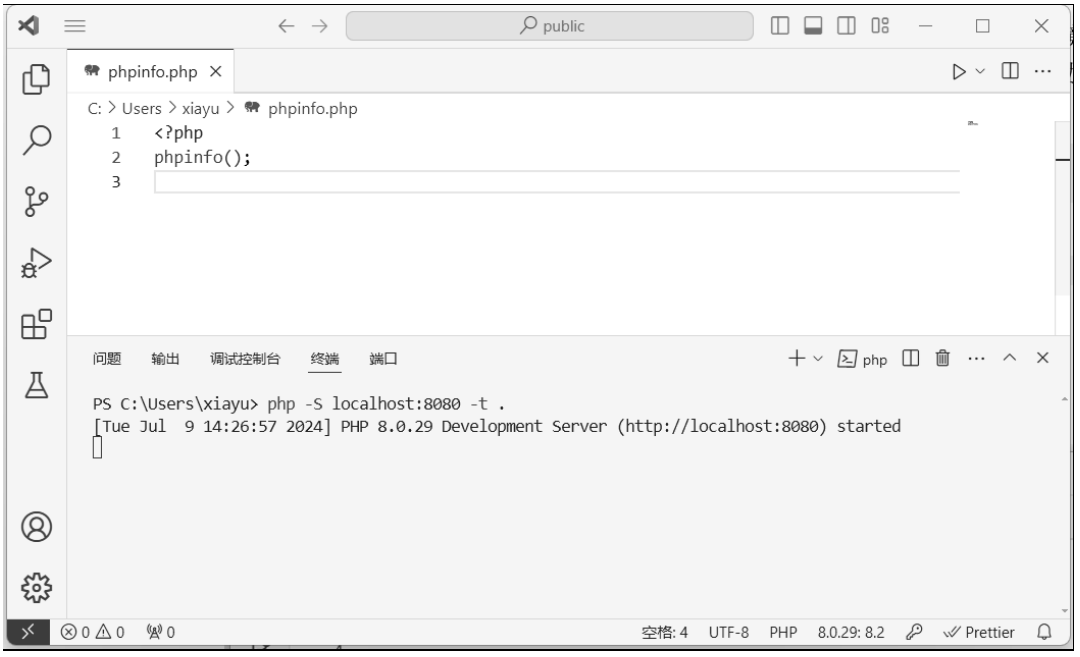

图 1-4

PHP Version 8.0.29

System	Windows NT WORKSTATION 10.0 build 22631 (Windows 10) i586
Build Date	Jun 7 2023 21:18:50
Build System	Microsoft Windows Server 2019 Datacenter [10.0.17763]
Compiler	Visual C++ 2019
Architecture	x86
Configure Command	cscript /nologo /e:jscript configure.js "--enable-snapshot-build" "--enable-debug-pack" "--with-pdo-oci=..\..\..\..\instantclient\sdk,shared" "--with-oci8-19=..\..\..\..\instantclient\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--without-analyzer" "--with-pgo"
Server API	Built-in HTTP server
Virtual Directory Support	enabled
Configuration File (php.ini) Path	*no value*
Loaded Configuration File	(none)
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)

图 1-5

恭喜你！已经成功搭建 PHP 8 开发环境，接下来我们将正式进入 ThinkPHP 8 的学习！

1.6　安装 ThinkPHP 开发环境

ThinkPHP 官方网站给出了 ThinkPHP 的安装方法，网址为 https://doc.thinkphp.cn/v8_0/setup.html。本节将结合官方文档讲解相关安装步骤。

1. 安装composer

在 Windows 系统中安装 composer，打开下载网址 https://getcomposer.org/Composer-Setup.exe，选择需要下载的文件 Composer-Setup.exe，下载完成后，在当前用户根目录下执行，之后会打开安装向导，读者可按照向导提示一步一步进行操作即可，最后将生成 3 个文件，结果如图 1-6 所示。

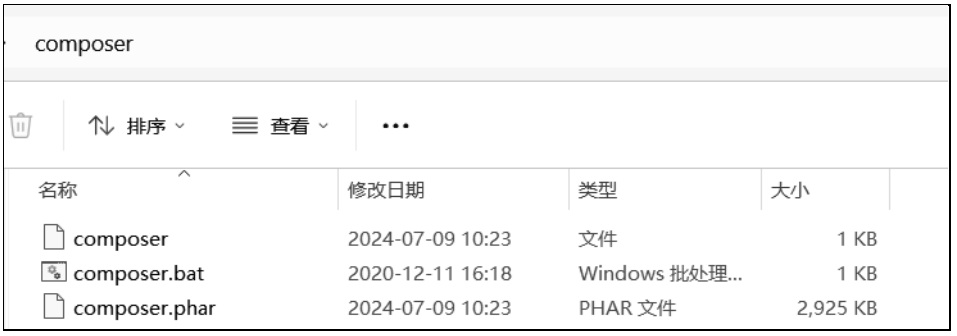

图 1-6

现在将 composer 目录加入环境变量 Path 中，以方便在任何目录下执行 composer.bat 命令。

Linux 和 macOS 系统可以执行以下命令安装 composer：

```
# 下载安装脚本
wget https://getcomposer.org/installer
# 执行安装器
php installer
# 将 composer 移动到 PATH 中包含的路径
sudo mv composer.phar /usr/local/bin
# 查询 composer 版本
composer -v
```

下面是笔者的输出：

```
Composer version 2.7.7 2024-06-10 22:11:12
PHP version 8.3.7 (/opt/homebrew/Cellar/php/8.3.7/bin/php)
Run the "diagnose" command to get more detailed diagnostics output.
```

2. 安装稳定版ThinkPHP

如果读者是第一次安装，需要打开命令行窗口，切换到当前用户的根目录（比如，笔者的当前用户根目录为 C:\Users\xiayu）下，并执行下面的命令：

```
composer config -g repo.packagist composer https://mirrors.aliyun.com/composer
composer create-project topthink/think tp
```

上述命令中第 1 个命令把安装源修改为国内的阿里云源。第 2 个命令中的 tp 是当前目录下自定义的目录名，也是项目名。执行此命令，将下载 ThinkPHP 框架代码，局部截图如图 1-7 所示。最终在当前用户根目录下，成功下载 ThinkPHP，如图 1-8 所示，读者可以到目录中查看具体信息。

```
PS C:\Users\xiayu> composer create-project topthink/think tp
Creating a "topthink/think" project at "./tp"
Installing topthink/think (v8.0.0)
  - Installing topthink/think (v8.0.0): Extracting archive
Created project in C:\Users\xiayu\tp
Loading composer repositories with package information
Updating dependencies
Lock file operations: 13 installs, 0 updates, 0 removals
  - Locking league/flysystem (2.5.0)
  - Locking league/mime-type-detection (1.15.0)
  - Locking psr/container (2.0.2)
  - Locking psr/http-message (1.1)
  - Locking psr/log (3.0.0)
  - Locking psr/simple-cache (3.0.0)
  - Locking symfony/polyfill-mbstring (v1.29.0)
  - Locking symfony/var-dumper (v6.0.19)
  - Locking topthink/framework (v8.0.3)
  - Locking topthink/think-filesystem (v2.0.2)
  - Locking topthink/think-helper (v3.1.6)
  - Locking topthink/think-orm (v3.0.14)
  - Locking topthink/think-trace (v1.6)
Writing lock file
Installing dependencies from lock file (including require-dev)
```

图 1-7

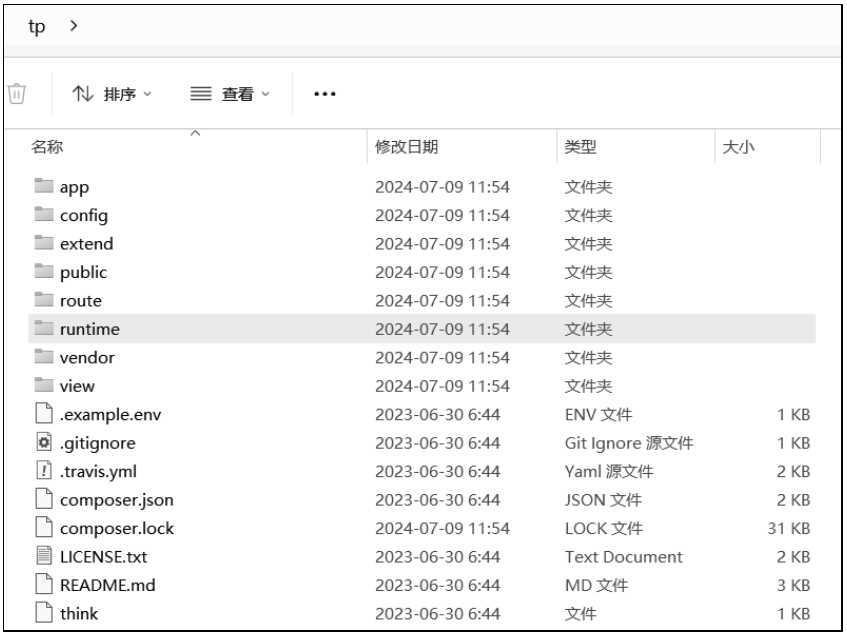

图 1-8

3. 验证ThinkPHP能否正确运行

上一步搭建好了 tp 项目，需要记住这个项目名。接下来，验证 ThinkPHP 能否正确运行。进入命令行窗口，在 tp 项目目录下执行下面命令，运行 Web 服务器：

```
php think run
```

命令执行结果如图 1-9 所示。

```
PS C:\Users\xiayu> cd tp
PS C:\Users\xiayu\tp> php think run
ThinkPHP Development server is started On <http://0.0.0.0:8000/>
You can exit with `CTRL-C`
Document root is: C:\Users\xiayu\tp\public
[Tue Jul  9 12:02:30 2024] PHP 8.0.29 Development Server (http://0.0.0.0:8000) started
```

图 1-9

我们在浏览器中输入地址 http://localhost:8000/，将会看到欢迎页面，如图 1-10 所示，说明 ThinkPHP 可以正确运行了。

图 1-10

4. 运行本书配套的示例源码

本书大部分章节都提供配套的示例源码，按照其中的文件名编写代码后，执行 php think run 启动服务器，再使用浏览器访问相应的 URL，就能看到结果（第 2 章是纯 PHP 的特性介绍，不包含 ThinkPHP 框架的功能，也可以在命令行窗口直接使用 php -S localhost:8000 命令启动服务器）。

接下来展示一下本书示例代码的运行方法。比如，下面将给出一个路由示例，其代码文件放在上面搭建的 tp 项目下的相应目录中，读者暂时不管代码起什么作用，按提示编辑好代码文件即可。

【示例 1-2】

编辑 route/app.php，代码如下：

```
<?php
use think\facade\Route;

Route::get('news/:id','News/show');
```

新建 app/controller/News.php，代码如下：

```
<?php
namespace app\controller;
```

```
class News
{
    public function show($id)
    {
        return '显示'.$id.'新闻详情';
    }
}
```

执行 php think run 命令运行服务器后，用浏览器访问 http://localhost:8000/news/10，结果如下：

```
显示 10 新闻详情
```

这样就成功执行了示例代码。读者在后续学习每一章时，都可以重新创建一个新项目，项目名可以自定义，比如学习第 4 章时，我们可以定义一个新项目，名为 ch04，再执行命令 composer create-project topthink/think ch04 创建这个项目，这样第 4 章的所有示例代码都可以在 ch04 这个项目目录下运行了。记住这种自定义的目录（项目），也是我们后面会经常提到的应用根目录。

使用 Visual Studio Code 编辑项目文件，只要通过菜单项“文件”→“打开文件夹…”，打开项目文件夹（比如上面介绍的 ch04），即可建立本项目编辑开发环境。

第 2 章

PHP 8 新特性及其示例

第 1 章中介绍了 PHP 8 的新特性，本章对这些新特性进行具体的讲解（本章示例代码的运行环境，参见 1.6 节最后的讲解）。

在本章中，我们将介绍以下主要内容：

- 命名参数
- 注解
- match表达式
- nullsafe运算符
- JIT编译器

2.1 命名参数

命名参数是 PHP 8 带来的一个革命性特性，它极大地提升了函数和方法的参数传递灵活性和代码的可读性。在以往的 PHP 版本中，函数调用时参数的传递必须按照预定的位置顺序进行，而命名参数允许开发者根据参数的名称来指定值，从而摆脱了对参数顺序的依赖。

使用命名参数，我们可以在调用函数时明确指出每个参数的名称，并赋予相应的值。这种方式的优点在于，我们可以轻松地跳过那些非必需的参数，只为我们关心的参数提供值，无须为了满足参数顺序而填充不必要的占位符。这对于那些拥有众多可选参数的函数来说特别有用，它使得代码更加清晰和易于理解。

命名参数的另一个显著优势是提高了函数调用的可读性。通过显式地指定参数名称，代码的意图变得更加明确，这使得其他开发者（或未来的你）更容易阅读和维护代码。此外，由于不再需要关注参数的顺序，因此也降低了因参数位置错误而导致 Bug 的风险。

总的来说，命名参数是 PHP 8 中一个强大的语言特性，它不仅让代码更加整洁，也提高了开发效率和代码质量。

2.1.1　语法

使用命名参数非常简单。只需在调用一个函数或方法时，使用“参数名:值”的方式进行传参。例如下面计算两数之和的示例。

【示例 2-1】

```
<?php
function sum($a, $b): int
{
    return $a + $b;
}

echo sum(a: 1, b: 2);
```

将以上代码保存在任意目录中，比如笔者保存在 C:\Users\xialei\PhpProjects\ch02\2-1.php 中，在该目录中执行 php -S localhost:8000 启动 Web 服务器，输出代码如下：

```
[Wed Jul 10 09:45:53 2024] PHP 8.3.7 Development Server (http://localhost:8000)
started
```

接下来打开浏览器访问 http://localhost:8000/2-1.php，结果如下：

```
3
```

本节的示例代码都可以使用上面的方法运行，需要说明的是，前面章节中执行的 php think run 是启动 ThinkPHP 项目的 Web 服务器，需要 ThinkPHP 框架才能运行，本示例中执行的 php -S localhost:8000 是启动普通的 PHP 项目 Web 服务器，只能直接运行普通 PHP 文件。

下面来看一下命名参数语法的优点和缺点。

2.1.2　命名参数的优点

命名参数的优点包括允许跳过默认值和参数顺序无关性。

1. 允许跳过默认值

在基于参数位置的传参方式时，即使只需要传递某个位置的值，也必须为前面的默认值传递参数，比如下面的示例。

【示例 2-2】

```
<?php
echo htmlspecialchars("<p>Hello World!</p>", ENT_COMPAT | ENT_HTML401,
ini_get('default_charset'), false);
```

运行结果如下（注意，本书后面所有示例代码的运行，均需要启动 PHP 内置的 Web 服务器，结果均通过浏览器显示出来）：

```
<p>Hello World!</p>
```

在这个示例中，为了设置最后的 double_encode 参数，我们必须把 flag 和 encoding 参数设置为默认值，这样会造成代码冗余，还有带来潜在 bug 的风险（如果 PHP 在后续版本更改了 flag 和 encoding 的默认值，上述代码可能无法正常工作，当然 PHP 内置函数一般不会更改默认行为。但是如果我们使用的是第三方库的话就可能会存在这个问题）。

而有了命名参数之后，我们只需要传递关注的参数，提高了代码可读性，也不会影响 PHP 默认值，使用命名参数的示例如下所示。

【示例 2-3】

```
<?php
echo htmlspecialchars("<p>Hello World!</p>", double_encode: false);
```

结果如下：

```
<p>Hello World!</p>
```

2. 参数顺序无关性

在函数或方法调用中，我们可以根据参数的名称来传递值，而不必担心参数的顺序。这使得函数调用更加灵活，可以跳过可选参数或按照自己的需求传递参数，而不需要依赖于参数在定义时的顺序。

【示例 2-4】

```
<?php
function addNumbers($a, $b) {
    return $a + $b;
}

// 使用命名参数调用 addNumbers()函数
$result = addNumbers(b: 2, a: 3);
echo $result; // 输出: 5
```

输出结果如下：

```
5
```

在上面的示例中，我们使用命名参数将值传递给 addNumbers()函数。参数 a 和 b 的顺序并不重要，因为我们明确指定了它们的名称。因此，addNumbers(b: 2, a: 3)和 addNumbers(a: 3, b: 2)将得到相同的结果，即 5。

2.1.3　命名参数的缺点

命名参数的缺点是，当我们修改参数名时会破坏兼容性，将导致代码无法正常工作。

假设有一个 calculateArea()函数用于计算矩形的面积，接受两个参数 width 和 height，并返回它们的乘积。

【示例 2-5】

```
<?php
```

```
function calculateArea($width, $height) {
    return $width * $height;
}

// 调用 calculateArea()函数
$result = calculateArea(width: 5, height: 10);
echo $result; // 输出: 50
```

输出结果如下：

```
50
```

现在，假设我们决定重构参数名 width 为 w，以更符合代码规范或更具描述性。

【示例 2-6】

```
<?php
function calculateArea($w, $height) {
    return $w * $height;
}

// 调用 calculateArea()函数（修改后的版本）
$result = calculateArea(w: 5, height: 10);
echo $result; // 输出: 50（仍然正常工作）
```

输出结果如下：

```
50
```

在上述示例中，我们修改了 calculateArea()函数的参数名 width 为 w。由于我们使用了命名参数，在调用函数时明确指定了参数名称，所以即使参数名发生了变化，代码仍然可以正常工作。

然而，这也意味着如果其他地方的代码也在调用 calculateArea()函数并使用了旧的参数名 width，那么这些代码将会因为参数名的变化而无法正常工作。

【示例 2-7】

```
<?php
function calculateArea($w, $height) {
    return $w * $height;
}

// 调用 calculateArea()函数（使用旧的参数名）
$result = calculateArea(width: 5, height: 10);
// 由于参数名已经修改，上述代码将会引发错误或产生意外结果
```

输出结果如下：

```
    Fatal error: Uncaught Error: Unknown named parameter $width in
/Users/xialei/PhpstormProjects/php8-tutorial/2-7.php:7 Stack trace: #0 {main}
thrown in /Users/xialei/PhpstormProjects/php8-tutorial/2-7.php on line 7
```

为了避免这种问题，当在修改函数的参数名时，需要仔细考虑并确保及时更新所有调用该函数的地方，以保持代码的一致性。

2.1.4　小结

通过使用命名参数，我们可以根据参数的名称来传递值，提升代码可读性以及编程体验。然而，在进行代码重构时，需要特别关注命名参数的重构，避免修改参数名时导致一些潜在问题，虽然该问题可以通过单元测试提供一定程度的保障，但更为重要的是需要仔细思考适合用命名参数的业务场景。

2.2　注解

注解是在 PHP 8 中引入的另一个强大特性，它为我们提供了一种在函数、方法和类中添加元数据的方式。通过注解，我们可以使用特定的语法来标记代码的某些部分，以提供额外的信息和上下文。这为我们的代码带来了更高的可读性、可维护性和扩展性。

注解的使用方式非常灵活，我们可以在函数、方法和类的声明之前使用#[]格式来定义注解。这些注解可以包含各种信息，比如参数类型、返回值类型、异常处理、作者信息等。注解可以帮助我们更好地理解代码的意图，以及如何正确地使用这些代码。

通过使用注解，我们可以在代码中添加更多的上下文和文档信息，使得代码更加自解释和易于理解。这对于团队合作和代码维护非常有帮助，因为它们可以提供一致的规范和约定，帮助开发人员更好地理解和使用代码库。

除了提供更多的上下文信息，注解还可以与其他工具和框架集成，进一步增强代码的功能和灵活性。例如，一些测试框架可以使用注解来标记测试用例，自动化文档生成工具可以使用注解来生成 API 文档，依赖注入容器可以使用注解来自动解析和注入依赖等。注解的使用可以大大简化我们的开发过程，提高代码的质量和效率。

2.2.1　模拟“注解”

在 PHP 8 之前，虽然没有原生的注解功能，但我们可以通过使用文档注释（docComment）来模拟一些注解的功能。文档注释是一种特殊的注释格式，可以在函数、方法和类的声明之前使用，以提供关于代码的额外信息和上下文。

文档注释通常以“/**”开始，以“*/”结束，位于被注释元素的前面。在文档注释中，我们可以使用特定的标签来描述代码的不同方面，比如参数类型、返回值类型、异常处理、作者信息等。这些标签可以提供一些元数据，帮助开发人员理解和使用代码。

虽然文档注释并不直接影响代码的执行，但它们可以被一些特定的工具和框架解析所利用。例如，一些文档生成工具可以读取文档注释，并生成代码的 API 文档。一些静态分析工具可以分析文档注释，提供代码的静态检查和类型提示。一些测试框架可以使用文档注释来标记测试用例。

下面是一个使用文档注释模拟注解功能的示例。

【示例 2-8】

```
<?php
// 定义函数
```

```
/**
 * @Author Xia Lei
 * @Description 计算两个数字之和
 */
function add($num1, $num2) {
    return $num1 + $num2;
}

// 使用反射获取函数的信息
$reflectionFunction = new ReflectionFunction('add');

// 获取文档注释
$docComment = $reflectionFunction->getDocComment();

// 解析文档注释中的作者信息和描述信息
$author = '';
$description = '';
$lines = explode("\n", $docComment);
foreach ($lines as $line) {
    if (strpos($line, '@Author') !== false) {
        $author = trim(str_replace(['*', '@Author'], '', $line));
    } elseif (strpos($line, '@Description') !== false) {
        $description = trim(str_replace(['*', '@Description'], '', $line));
    }
}

// 输出作者信息和描述信息
echo 'Author: ' . $author . "\n";
echo 'Description: ' . $description . "\n";
```

输出结果如下：

```
Author: Xia Lei
Description: 计算两个数字之和
```

在上面的示例中，我们使用@Author 标签指定了函数的作者信息，使用@Description 标签提供了函数的描述信息。通过反射读取注释得到了注解值。虽然文档注释可以模拟一些注解的功能，但它们也有一些限制。首先，文档注释只是作为注释存在，不会直接影响代码的行为；其次，文档注释的解析和利用需要依赖额外的工具和框架支持。因此，它们的功能和灵活性相对有限。

2.2.2　语法

注解语法包含几个关键部分。首先，注解声明总是以“#[”开头，以“]”结尾。注解内容包含一个或多个以逗号分隔的注解。其次，注解的名称可以是非限定、限定、完全限定的名称。然后，注解的参数是可选的，如果存在，则必须使用圆括号“()”包围，其参数可以是字面量或常量表达式。它同时接受位置参数和命名参数两种语法。

通过反射 API 请求注解实例时，注解的名称会被解析到一个类，注解的参数则传入该类的构造器中。因此每个注解都需要引入一个注解类。注解类本身需要使用 Attribute 进行标识。

1. 基本使用

下面介绍使用 PHP 8 注解的示例代码，其中定义了 Author 和 Description 注解类，以获取函数的@Author 和@Description 注解信息：

【示例 2-9】

```
<?php

#[Attribute]
class Author
{
    public string $name;

    public function __construct(string $name)
    {
        $this->name = $name;
    }
}

#[Attribute]
class Description
{
    public string $content;

    public function __construct(string $content)
    {
        $this->content = $content;
    }

}

#[Author("Xia Lei")]
#[Description("计算两数之和")]
function add(int $num1, int $num2): int
{
    return $num1 + $num2;
}

// 使用反射获取函数的信息
$reflection = new ReflectionFunction('add');
$author = $reflection->getAttributes(Author::class)[0]->newInstance()->name;
$description =
$reflection->getAttributes(Description::class)[0]->newInstance()->content;

// 输出作者名和描述信息
echo 'Author: ' . $author . "\n";
echo 'Description: ' . $description . "\n";
```

输出结果如下：

```
Author: Xia Lei
Description: 计算两个数字之和
```

2. 多个注解

可以通过给 Attribute 注解传递 Attribute::IS_REPEATABLE 标志来允许重复运用注解。下面介绍使用多个 Author 注解的示例。

【示例 2-10】

```
<?php
#[Attribute(Attribute::IS_REPEATABLE | Attribute::TARGET_ALL)]
class Author
{
    public string $name;

    public function __construct(string $name)
    {
        $this->name = $name;
    }
}

#[Author("Xia Lei")]
#[Author("Zhang San")]
function add(int $num1, int $num2): int
{
    return $num1 + $num2;
}

// 使用反射获取函数的信息
$reflection = new ReflectionFunction('add');
$authors = $reflection->getAttributes(Author::class);

foreach ($authors as $author) {
    echo $author->newInstance()->name . PHP_EOL;
}
```

输出结果如下：

```
Xia Lei
Zhang San
```

2.2.3　高级应用

基于 PHP 8 的注解特性，我们可以高效地实现一些此前用注释模拟的注解功能。

1. RBAC（基于角色的访问控制）

RBAC 是一种常见的访问控制模型，其中权限与角色相关联，当用户被授予角色时，即可获得相应的权限。在 PHP 8 中，可以使用注解来标记需要进行权限验证的方法或类，并通过自定义的注解处理器来检查当前用户是否具有执行该操作的权限。

下面介绍的是在函数中使用注解拦截调用的示例。

【示例 2-11】

```
<?php

#[Attribute(Attribute::TARGET_METHOD | Attribute::TARGET_FUNCTION)]
class RequirePermission
{
    public string $name;

    public function __construct(string $name)
    {
        $this->name = $name;
    }
}

/**
 * 添加用户
 */
#[RequirePermission('create')]
function createUser(string $name): void
{
    echo "Creating user $name\n";
}

/**
 * 查看用户
 */
function viewUser(string $name): void
{
    echo "Viewing user $name\n";
}

/**
 * 执行函数
 * @throws Exception
 */
function execute(array $permissions, string $functionName, array $args): void
{
    $reflection = new ReflectionFunction($functionName);
    // 方法需要的权限列表
    $attributes = $reflection->getAttributes(RequirePermission::class);

    foreach ($attributes as $attribute) {
        $permission = $attribute->newInstance();
        if (!in_array($permission->name, $permissions)) {
            throw new Exception("Permission $permission->name required");
        }
    }
```

```
        call_user_func_array($functionName, $args);
    }

    // case1: 允许创建和查看
    $permissions = ['create'];
    try {
        execute($permissions, 'viewUser', ['Xia Lei']);
        execute($permissions, 'createUser', ['Xia Lei']);
    } catch (Exception $e) {
        echo $e->getMessage();
    }

    // case2: 只允许查看
    $permissions = [];
    try {
        execute($permissions, 'viewUser', ['Xia Lei']);
        execute($permissions, 'createUser', ['Xia Lei']);
    } catch (Exception $e) {
        echo $e->getMessage();
    }
```

输出结果如下：

```
Viewing user Xia Lei
Creating user Xia Lei
Viewing user Xia Lei
Permission create required
```

2. API路由和文档生成

注解可以用于定义 API 路由和生成 API 文档。可以使用注解来标记控制器方法或类，并通过自定义的注解解析器来提取路由信息，以便构建动态路由表。同时，注解可以包含其他元数据，如请求方法、路径参数等，以及用于生成 API 文档的描述信息，示例如下：

【示例 2-12】

```
<?php
#[Route('/users/{id}', methods: ['GET'])]
#[Description('Get user by ID')]
function getUser($id) {
    // 获取用户的逻辑
}

// 在注解解析器中提取路由信息并建立路由表
// ...
```

3. 表单验证

注解可以用于定义表单验证规则，简化表单数据的验证过程。可以使用注解来标记表单实体类的属性，并定义各种验证规则，如必填字段、最大长度、数字范围等。下面介绍使用自定义的表单验证器来解析注解并执行相应的验证逻辑，示例如下：

【示例 2-13】

```
<?php
class UserForm {
    #[Required]
    #[MaxLength(50)]
    public string $name;

    #[Required]
    #[Email]
    public string $email;

    #[Range(18, 99)]
    public int $age;
}

// 在表单验证器中解析注解并执行验证逻辑
// ...
```

通过自定义的注解处理器、注解解析器和验证器，可以根据具体需求扩展和定制注解功能，用于满足不同的应用场景。

2.2.4　小结

PHP 8 的注解功能提供了一种在代码中添加元数据和附加信息的方法。通过使用注解功能，可以实现各种高级应用，如权限控制、路由定义、表单验证等。注解可以通过反射 API 获取，并通过自定义的注解处理器或注解解析器进行处理。这种方式可以提高代码的可读性、可维护性和灵活性，为开发人员带来更好的开发体验。

2.3　match 表达式

match 表达式作为一种强大的模式匹配工具，为开发者提供了一种简洁、灵活的方式用来处理各种条件分支。传统的 switch 语句在处理多个条件分支时可能显得笨重烦琐，而 match 表达式则以其简洁而强大的语法帮助我们摆脱这一束缚。通过使用 match 表达式，我们可以轻松地对一个值进行多重模式匹配，并根据匹配结果执行相应的代码块。无论是简单的相等比较，还是复杂的类型检查，match 表达式都能胜任。

需要注意的是，match 表达式和 switch 语句类似，都有一个表达式主体，可以和多个可选项进行比较。与 switch 不同的是，它会像三元表达式一样求值。与 switch 的另一个不同点是，它的比较是严格比较（===）而不是宽松比较（==）。

- ===：严格比较符，表示类型和值都要相等，比如0===null的值是false。
- ==：宽松比较符，表示只要求值相等，比如0==null的值是true。

2.3.1　语法

match 表达式的语法如下：

```
$return_value = match (求值对象) {
    单个条件表达式 => 返回值,
    表达式 1, 表达式 2 => 返回值,
};
```

下面是一个使用 match 表达式的示例。

【示例 2-14】

```
<?php
$food = 'cake';

$return_value = match ($food) {
    'apple' => 'This food is an apple',
    'bar' => 'This food is a bar',
    'cake' => 'This food is a cake',
};
echo $return_value;
```

输出结果如下：

```
This food is a cake
```

match 表达式与 switch 语法的区别如下：

- match表达式使用严格比较（===），switch语句使用宽松比较（==）。
- match是表达式，返回一个值，而switch是语句，语句不能返回值。
- match是表达式，因此必须以分号结尾，而switch语句不要求以分号结尾。
- match表达式不会自动执行下一个case，而switch语句默认执行下一个case，需要使用break关键字来避免该行为。
- match表达式必须列举所有情况，switch语句则无此要求。

2.3.2　示例

1. 不匹配的case

下面示例由于存在不匹配的 case，因此 match 表达式会抛出 Error 错误。

【示例 2-15】

```
<?php
$food = 'cake1';

$return_value = match ($food) {
    'cake' => 'This food is a cake',
};
echo $return_value;
```

输出结果如下：

```
    Fatal error: Uncaught UnhandledMatchError: Unhandled match case 'cake1' in
index.php:5
    Stack trace: #0 {main} thrown in index.php on line 5
```

针对错误结果，可以使用 default 分支来进行修复。

【示例 2-16】

```
    <?php
    $food = 'cake1';

    $return_value = match ($food) {
        'cake' => 'This food is a cake',
        default => 'not match'
    };
    echo $return_value;
```

输出结果如下：

```
    not match
```

多个 default 分支也会导致错误。

2. case执行顺序

match 表达式和 switch 语句类似，可以逐个检测匹配分支。在一开始时不会执行代码，只有在所有之前的条件不匹配主体表达式时，才会执行剩下的条件表达式。

```
    <?php
    $result = match ($x) {
        method1() => ...,
        method2() => ..., // 如果 method1() === $x，不会执行 method2()
        $value => hello(), // 只有 $x === $value 时，才会执行 hello()
        // 其他 case
    };
```

3. 多个分支表达式

match 表达式分支可以通过使用逗号分隔，分隔后可以包含多个表达式，其中只要有 1 个表达式匹配，就会执行右侧代码：

```
    <?php
    $result = match ($x) {
        // 匹配分支
        $a, $b, $c => 5,
        // 等同于以下三个分支
        $a => 5,
        $b => 5,
        $c => 5,
    };
```

4. 优雅的条件判断

由于 match 是表达式，可以返回值，因此可以优雅地实现一些之前由 if 实现的功能。

【示例 2-17】

```
<?php
$score = 75;
$status = match (true) {
    $score >= 90 => '优秀',
    $score >= 70 => '良好',
    $score >= 60 => '及格',
    default => '不及格',
};

echo $status;
```

输出结果如下：

```
良好
```

2.3.3　小结

相对于传统的 switch 语句，match 表达式在处理条件分支时具有明显的优势。首先，它的语法更加简洁明了，使用 match 关键字并将待匹配的值作为主项表达式，然后根据模式匹配执行相应的代码块。相比之下，switch 语句需要使用 case 和 break 语句，语法上较为烦琐。

除了简洁性，match 表达式还提供了更灵活的模式匹配功能。它可以处理更复杂的条件，包括类型匹配、比较运算和自定义模式等。这使得开发者能够更精确地匹配和处理不同的情况，使代码更加清晰和可读。

2.4　nullsafe 运算符

在当今的软件开发中，处理变量为空的情况一直是一个普遍存在的挑战。在过去的 PHP 版本中，我们通常需要通过一系列的条件语句和空值判断来确保代码的健壮性和安全性。然而，随着 PHP 8 的到来，nullsafe 运算符为该问题提供了解决方案。

nullsafe 运算符是 PHP 8 引入的一项重要功能，它的目的是简化代码中对空值的处理。使用 nullsafe 运算符，我们可以优雅地处理可能为空的对象，而无须烦琐的条件判断和链式调用。

2.4.1　语法

nullsafe 运算符的使用非常简单。当我们想要调用一个对象的方法或访问其属性时，只需在对象名之前加上“?->”即可。如果对象为空，nullsafe 运算符会立即返回 null，而不会引发错误或异常。

下面是一个 PHP 8 之前进行链式取值的示例。

【示例 2-18】

```
<?php
$country =  null;

if ($session !== null) {
    $user = $session->user;

    if ($user !== null) {
        $address = $user->getAddress();

        if ($address !== null) {
            $country = $address->country;
        }
    }
}
```

为了获取用户所属的国家，我们需要使用 3 个 if 语句才能够实现，而使用 PHP 8 的 nullsafe 运算符实现相同的功能仅需一行代码：

```
$country = $session?->user?->getAddress()?->country;
```

下面是一个直接使用“->”运算符导致异常的示例。

【示例 2-19】

```
<?php
$user = null;
var_dump($user->address->city);
```

输出结果如下：

```
Warning: Attempt to read property "address" on null in index.php on line 4
Warning: Attempt to read property "city" on null in index.php on line 4
NULL
```

需要注意的是“?->”只能运用于对象，应用于数组则会报错：

【示例 2-20】

```
<?php
$user = [];

var_dump($user?->zhangsan?->address?->city);
```

输出结果如下：

```
Warning: Attempt to read property "zhangsan" on array in index.php on line 4
NULL
```

将示例 2-20 修改为示例 2-21，则会输出 NULL，不会报错。

【示例 2-21】

```
<?php
```

```
$user = [
    'zhangsan' => null
];

var_dump($user['zhangsan']?->address?->city);
```

输出结果如下：

```
NULL
```

2.4.2　null 合并运算符

PHP 中的 null 合并运算符（null coalescing operator）是一项在 PHP 7 中引入的特性，它提供了一种简洁的方式来处理变量为空的情况。

在过去，我们需要使用条件判断来检查变量是否为 null，并在变量为空时提供一个默认值。例如：

```
$value = isset($variable) ? $variable : $default;
```

然而，这种写法显得冗长且不够直观。为了简化这种情况下的代码，PHP 引入了 null 合并运算符“??”。

使用 null 合并运算符可以通过一行简洁的代码实现相同的功能：

```
$value = $variable ?? $default;
```

null 合并运算符的作用是检查变量是否为 null。如果变量不为 null，则返回变量的值；如果变量为 null，则返回指定的默认值。

此外，null 合并运算符还支持链式操作。我们可以在连续的变量中使用多个 null 合并运算符，以便逐个检查它们是否为 null，并返回第一个非 null 的值。

```
$result = $value1 ?? $value2 ?? $value3 ?? $default;
```

2.4.3　nullsafe 运算符和 null 合并运算符区别

nullsafe 运算符和 null 合并运算符是两种在 PHP 中处理空值的不同特性，它们的功能和适用场景不同。

1. 功能不同

- nullsafe运算符（?->）用于处理可能为空的对象的方法调用和属性访问。它允许我们通过在对象名之前添加“?->”来安全地调用方法或访问属性，如果对象为空，将立即返回null，因此不会引发错误或异常。
- null合并运算符（??）用于处理变量为空的情况。如果变量不为null，则返回变量的值；如果变量为null，则返回指定的默认值。

2. 适用场景不同

- nullsafe运算符：主要用于处理对象。它用于确保在对象连续调用或属性访问的过程中，每一个对象都存在且不为空。

- null合并运算符：适用于任何类型的变量，包括对象、数组和其他数据类型。

2.4.4　小结

使用 nullsafe 运算符可以编写更加简洁、可读和易维护的代码。它提供了一种优雅而高效的方式来处理复杂的对象关系和嵌套的属性访问。无论是在大型应用程序中还是在小型脚本中，nullsafe 运算符都能够显著提升代码的可靠性和开发效率。

2.5　JIT 编译器

PHP 是一种广泛使用的脚本语言，被用于构建各种规模的 Web 应用程序。然而，由于其解释执行的本质，PHP 在处理大量计算密集型任务时可能会遇到性能瓶颈。为了解决这个问题，PHP 8 引入了一个令人激动的新特性——JIT（即时编译）编译器。

JIT（Just-In-Time）编译器是一种在运行时将解释的代码转换为机器码的技术。这种转换可以显著提高代码的执行速度，使得 PHP 在处理复杂算法和大数据集时表现更出色。本节将深入探索 PHP 8 的 JIT 编译器，了解其工作原理、优势和使用技巧。

2.5.1　PHP 中 JIT 编译器的特性

PHP 中 JIT 编译器的特性包括解释执行、直接执行、解释执行过程和 Opcache。

1. 解释执行

在解释执行中，代码是逐行解释并执行的。解释器读取源代码的任意行，同时分析该行的含义，并立即执行相应的操作。解释器将代码逐行翻译为机器指令并执行，不进行额外的编译步骤。每次执行代码时，都需要进行解释和执行的过程。

解释执行的优点是灵活性高，因为它可以根据具体环境和条件进行动态调整。它还支持动态特性，如动态类型和运行时代码修改。然而，解释执行的效率通常较低，因为它需要在执行过程中对代码进行解释和翻译，导致执行速度相对较慢。

2. 直接执行

在直接执行中，代码在执行之前会被编译成机器码（或类似的低级形式）。编译过程将源代码转换为一系列机器指令，这些指令可以直接在硬件上执行。直接执行通常是通过编译器完成的，这样可以将代码转换为与特定硬件架构相关的机器指令。

直接执行的优点是执行速度快，因为代码已经被预先编译成机器码了，因此不需要在执行时进行解释和翻译。它还可以进行各种优化，如静态类型检查和编译器优化。然而，直接执行通常缺乏灵活性，因为编译过程发生在执行之前，无法根据具体环境和条件进行动态调整。

3. 解释执行过程

下面介绍一下传统的 PHP 代码解释执行的过程：

（1）词法分析（Lexical Analysis）：PHP 解释器首先会将源代码分解为一个一个的词法单元（tokens）。词法单元可以是关键字、标识符、运算符、常量等。

（2）语法分析（Syntax Analysis）：在这个阶段，PHP 解释器将词法单元组合成语法结构，形成抽象语法树（Abstract Syntax Tree，AST）。抽象语法树表示了源代码的结构和语义。

（3）语义分析（Semantic Analysis）：在这一阶段，PHP 解释器会检查代码的语义正确性，包括变量和函数的声明、类型检查、作用域等。它还会解析命名空间、类、接口和其他语言特性。

（4）字节码生成（Bytecode Generation）：在这个阶段，PHP 解释器将抽象语法树转换为中间表示形式，即字节码。字节码是一种类似于机器码但不直接在硬件上执行的低级代码。

（5）执行：PHP 解释器按照顺序逐条执行字节码指令。这是解释器的核心阶段，其中包括变量赋值、函数调用、条件判断、循环等操作。解释器会根据指令的操作码和操作数执行相应的操作。

需要注意的是，PHP 在解释执行过程中是逐行解释的，即每次只执行一行代码。这与编译型语言不同，编译型语言在程序执行前会将源代码编译成机器码，然后直接在硬件上执行。

4. Opcache

Opcache 是 PHP 的一个扩展，旨在提高 PHP 脚本的性能。它通过缓存编译后的字节码，避免在每次脚本执行时重复进行词法分析、语法分析和字节码生成的过程。

启用 Opacache 扩展后 PHP 的执行流程如下：

（1）检查是否有可用的字节码缓存，如果没有，则需要先生成字节码，步骤为：词法分析→语法分析→语义分析→字节码生成。

（2）PHP 解释器执行缓存好的字节码。

2.5.2　PHP 中的 JIT 编译器

虽然启用 Opcache 能够避免每次请求时都重新生成字节码，但字节码仍然需要通过 PHP 解释器来执行，这本质上仍然是解释执行的方式，因此性能上存在一定的限制。然而，引入 JIT（即时编译器）后，频繁执行的热点代码可以被编译成原生机器码，这样 CPU 可以直接执行这些代码，显著提升了执行效率。

PHP 的 JIT 编译器是对 Opcache 的一个补充而非替代品。JIT 编译器在 Opcache 已经缓存好的字节码基础上，结合运行时的信息进行进一步优化，直接将字节码编译为机器码。

PHP 的 JIT 工作原理概述如下：

- 热点代码识别：JIT通过监控代码的执行情况来识别热点代码。这些热点代码是指在程序运行时频繁被执行的代码片段，比如循环和计算密集型的函数。JIT技术会根据代码的执行频率和重要性来决定哪些代码应该被编译。
- 即时编译：一旦识别出热点代码，JIT便会将这些代码片段提取出来，并动态编译成机器码。这些编译后的机器码可以直接在硬件上执行，跳过了逐行解释的过程。
- 缓存与重用：编译后的机器码通常会存储在缓存中，以便后续执行时直接调用，避免了重复编译的过程，从而提升了执行效率。

- 动态优化：JIT还能应用多种优化技术来增强代码的执行性能。例如，它可以通过内联优化减少函数调用的开销，或者根据变量类型信息进行基于类型的优化，生成更高效的机器码。

在执行脚本时，如果已经有可用的机器码，那么将直接执行这些机器码，无须从 Opcache 获取字节码，也省去了生成字节码的步骤，从而实现了性能的大幅提升。

2.5.3　使用 JIT 编译器

前面的内容提到过，JIT 编译器是构建在 Opcache 基础之上的，因此只需要安装 Opcache 扩展即可通过配置文件启用 JIT。

下面是笔者在 Mac 系统下的 Opcache 配置文件，路径是/opt/homebrew/etc/php/8.3/conf.d/ext-opcache.ini：

```
zend_extension=opcache.so
opcache.enable=1 # 开启 Opacache
opcache.enable_cli=1 # 通过命令行执行 PHP 脚本时也启用 Opcache
opcache.jit=1255
opcache.jit_buffer_size=64
```

而在 Windows 系统下使用 JIT 也是需要先开启 Opcache，此扩展默认已经包含到 PHP Windows 版本中，只要在 php.ini 中启用这个扩展即可。使用 Opcache 可以自动编译和优化 PHP 脚本，并将它们缓存在内存中，这样就不会在每次加载页面时动态编译 PHP 脚本。在 Windows 系统中，Opcache 需要配置在 php.ini 中，部分配置如下所示：

```
zend extension=opcache
opcache.enable=1
opcache.enable cli=0
opcache.jit=1255
opcache.jit_buffer_size=64
```

Windows 系统中有关 Opcache 每个配置项的含义，读者可以参考 PHP 官网的帮助文档。

1. JIT配置

Opcache 的 JIT 配置相对复杂，它接受以下几种类型的值：

- disable：在启动时完全禁用JIT功能，并且在运行时无法启用。
- off：禁用，但是可以在运行时启用JIT。
- on：启用tracing模式。
- tracing：启用跟踪模式，并允许进行细化配置，使用4位数字1254。
- function：启用函数模式，并允许进行细化配置，使用4位数字1205。
- 4位数字模式：启用细化配置，可以精确控制JIT的行为。

JIT 的 4 位标志是 4 个配置选项组合而成，按照 CRTO 的格式进行配置。

2. C-CPU特定的优化标志

C-CPU 特定的优化标志的不同配置如下：

- 0：不使用。
- 1：使用。

3. R-寄存器分配

R-寄存器分配的不同配置如下：

- 0：不执行寄存器分配。
- 1：使用本地线性扫描寄存器分配器。
- 2：使用全局线性扫描寄存器分配器。

4. T-JIT触发器

T-JIT 触发器级别的不同配置如下：

- 0：在PHP脚本加载时进行JIT编译。
- 1：在函数首次被调用时启动JIT编译。
- 2：在脚本运行后，对调用次数最多的函数进行JIT编译，这些函数的调用次数占所有函数调用次数的百分之（opcache.prof_threshold * 100）。
- 3：当函数或方法的执行次数超过特定阈值N时（N由opcache.jit_hot_func配置决定）启动JIT编译。
- 4：当函数或方法的注释中包含@jit标签时，对该函数或方法进行JIT编译。
- 5：当一个Trace（代码执行路径）被执行超过特定次数（次数由opcache.jit_hot_loop、opcache.jit_hot_return等配置决定）后，启动JIT编译。

5. O-优化级别

O-优化级别的不同配置如下：

- 0：最小化JIT编译（调用标准虚拟机处理程序）。
- 1：执行opline之间的跳转部分的JIT编译。
- 2：基于单个函数的静态类型推断进行优化JIT编译。
- 3：基于类型推断和过程调用图，进行函数级别的JIT编译。
- 4：基于类型推断和过程调用图，进行脚本级别的JIT编译。

从上述配置可以看出，PHP 的 JIT 编译器配置较为复杂。以下是根据不同使用场景的建议配置：

- 对于Web服务器，建议使用1235或1255配置。
- 对于CLI（命令行接口）脚本，建议使用1205配置。

表 2-1 展示了当 JIT 配置为 1255 时，PHPINFO 输出的部分 JIT 信息，这可以用来确认 JIT 编译器是否成功启用。

表 2-1　当 JIT 配置为 1255 时，PHPINFO 输出的部分 JIT 信息

Directive	Local Value	Master Value
opcache.enable	On	On
opcache.enable_cli	On	On
opcache.jit	1255	1255
opcache.jit_buffer_size	64	64
opcache.jit_hot_func	127	127
opcache.jit_hot_loop	64	64
opcache.jit_hot_return	8	8
opcache.jit_hot_side_exit	8	8

2.5.4　小结

JIT 编译器是 PHP 8 中引入的一个关键特性，其主要目的是提升 PHP 脚本的执行效率。与传统逐行解释执行的方式不同，JIT 编译器能够将频繁执行的热点代码动态转换为机器码。这一转换过程消除了逐行解释的需要，显著加快了代码的执行速度。