PHP

Composer 使用指南

Composer 是 PHP 的依赖管理工具，类似于 Node.js 的 npm 或 Python 的 pip。它可以帮助你声明、安装和管理 PHP 项目所依赖的库。

安装 Composer

Windows 系统

1. 下载 Composer-Setup.exe
2. 运行安装程序，按照向导完成安装
3. 安装完成后，在命令行输入 `composer --version` 验证安装

Linux/macOS 系统

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
php composer-setup.php
php -r "unlink('composer-setup.php');"
sudo mv composer.phar /usr/local/bin/composer
```

基本使用

1. 初始化项目

```
composer init
```

按照提示填写项目信息，会生成 `composer.json` 文件。

2. 安装依赖

```
composer install
```

这会读取 `composer.json` 文件并安装所有指定的依赖包到 `vendor` 目录。

3. 添加新依赖

```
composer require vendor/package
```

例如:

```
composer require monolog/monolog
```

这会自动更新 `composer.json` 和 `composer.lock` 文件。

4. 更新依赖

```
composer update
```

更新所有依赖到最新版本（在 `composer.json` 约束范围内）。

5. 更新特定包

```
composer update vendor/package
```

6. 移除依赖

```
composer remove vendor/package
```

composer.json 文件结构

典型示例:

```
{
    "name": "your-project/name",
    "description": "Your project description",
```

```
    "type": "project",
    "require": {
        "php": "^7.4 || ^8.0",
        "monolog/monolog": "^2.0"
    },
    "require-dev": {
        "phpunit/phpunit": "^9.0"
    },
    "autoload": {
        "psr-4": {
            "YourNamespace\\": "src/"
        }
    }
}
```

常用命令

- `composer list` - 显示所有可用命令
- `composer show` - 显示已安装的包
- `composer search package` - 搜索包
- `composer dump-autoload` - 重新生成自动加载文件
- `composer self-update` - 更新 Composer 自身
- `composer check-platform-reqs` - 检查平台要求

自动加载

Composer 提供了自动加载功能，你只需要在代码中添加：

```
require 'vendor/autoload.php';
```

开发和生产环境

- `require` 部分包含生产环境所需的包
- `require-dev` 部分包含开发环境所需的包

安装时排除开发依赖：

```
composer install --no-dev
```

全局包

安装全局包（如 Laravel 安装器）：

```
composer global require laravel/installer
```

镜像加速（中国用户）

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

恢复官方源：

```
composer config -g --unset repos.packagist
```

常见问题解决

1. 内存不足错误：

   ```
   COMPOSER_MEMORY_LIMIT=-1 composer update
   ```

2. 版本冲突：仔细检查包版本要求，或使用 `composer why-not vendor/package version` 查看冲突原因
3. 清除缓存：

   ```
   composer clear-cache
   ```

Composer 是现代化 PHP 开发不可或缺的工具，熟练掌握它可以大大提高开发效率。

Rocky Linux 安装 phpbrew 及使用指南

phpbrew 是一个 PHP 多版本管理工具，允许你在同一系统上安装和管理多个 PHP 版本。

在 Rocky Linux 上安装 phpbrew

1. 安装必要依赖

```
sudo dnf install -y git curl wget make automake gcc gcc-c++ \
    kernel-devel bison re2c libxml2-devel openssl-devel \
    libcurl-devel libjpeg-turbo-devel libpng-devel libicu-devel \
    libxslt-devel libzip-devel bzip2-devel readline-devel \
    sqlite-devel oniguruma-devel
```

2. 安装 phpbrew

```
curl -L -O https://github.com/phpbrew/phpbrew/releases/latest/download/phpbrew.phar
chmod +x phpbrew.phar
sudo mv phpbrew.phar /usr/local/bin/phpbrew
```

3. 初始化 phpbrew

```
phpbrew init
```

将以下行添加到你的 shell 配置文件（如 `~/.bashrc` 或 `~/.zshrc`）：

```
[[ -e ~/.phpbrew/bashrc ]] && source ~/.phpbrew/bashrc
```

然后重新加载 shell：

```
source ~/.bashrc  # 或 source ~/.zshrc
```

phpbrew 基本使用

1. 查看可用 PHP 版本

```
phpbrew known
```

2. 安装 PHP 版本

```
# 安装指定版本
phpbrew install 8.1 +default +openssl +fpm


# 安装带有额外扩展的版本
phpbrew install 8.0 +default +openssl +fpm +mysql +pgsql +gd
```

常用变体:

- `+default`: 包含常用扩展
- `+openssl`: 启用 OpenSSL 支持
- `+fpm`: 启用 PHP-FPM
- `+mysql`: MySQL 支持
- `+pgsql`: PostgreSQL 支持
- `+gd`: GD 图像处理库

3. 列出已安装的 PHP 版本

```
phpbrew list
```

4. 切换 PHP 版本

```
# 临时切换（仅当前会话）
phpbrew use 8.1


# 设置为默认版本
```

```
phpbrew switch 8.1
```

5. 运行指定版本的 PHP

```
phpbrew php 8.1 -v
```

6. 安装扩展

```
# 查看可用扩展
phpbrew ext

# 安装扩展
phpbrew ext install xdebug 3.1.2

# 或从源码安装
phpbrew ext install https://pecl.php.net/get/xdebug-3.1.2.tgz
```

7. 清理构建文件

```
phpbrew clean
```

常见问题解决

1. 安装失败

如果安装失败，可以尝试：

```
phpbrew -d install 8.1  # 启用调试模式查看详细错误
```

2. 缺少依赖

根据错误信息安装缺少的依赖，例如：

```
sudo dnf install libffi-devel
```

3. 更新 phpbrew

```
phpbrew self-update
```

高级用法

1. 自定义编译选项

```
phpbrew install 8.1 \
    -- --with-libdir=lib64 \
    --with-openssl-dir=/usr/include/openssl
```

2. 并行编译加速

```
phpbrew -j $(nproc) install 8.1  # 使用所有 CPU核心编译
```

3. 创建 fpm 服务

```
sudo cp ~/.phpbrew/php/php-8.1.0/sbin/php-fpm /usr/sbin/php-fpm8.1
sudo cp ~/.phpbrew/php/php-8.1.0/etc/php-fpm.conf.default /etc/php-fpm8.1.conf
sudo cp ~/.phpbrew/php/php-8.1.0/etc/php-fpm.d/www.conf.default /etc/php-fpm8.1.d/www.conf
```

然后可以创建 systemd 服务文件来管理 PHP-FPM。

卸载 phpbrew

1. 删除 phpbrew 目录:

```
rm -rf ~/.phpbrew
```

2. 从 shell 配置文件中移除 phpbrew 相关行

3. 删除 phpbrew 可执行文件:

```
sudo rm /usr/local/bin/phpbrew
```

phpbrew 是管理多 PHP 环境的强大工具，特别适合需要在不同项目中使用不同 PHP 版本的开发者。

PHP Composer 最佳实践指南

Composer 是 PHP 生态系统的依赖管理工具，正确使用可以显著提升项目质量和开发效率。以下是 Composer 的最佳实践：

1. 基础配置

项目初始化

```
# 创建新项目时使用
composer init --require="php:^8.1" --require-dev="phpunit/phpunit:^9.0" -n
```

合理的 `composer.json` 结构

```
{
    "name": "vendor/project",
    "description": "项目描述 ",
    "type": "project",
    "license": "proprietary",
    "require": {
        "php": "^8.1",
        "ext-json": "*",
        "laravel/framework": "^10.0"
    },
    "require-dev": {
        "phpunit/phpunit": "^10.0",
        "mockery/mockery": "^1.5"
    },
    "autoload": {
        "psr-4": {
            "App\\": "src/"
        },
        "files": [
```

```
            "src/helpers.php"
        ]
    },
    "autoload-dev": {
        "psr-4": {
            "Tests\\": "tests/"
        }
    },
    "scripts": {
        "post-autoload-dump": [
            "Illuminate\\Foundation\\ComposerScripts::postAutoloadDump",
            "@php artisan package:discover --ansi"
        ],
        "test": [
            "@php vendor/bin/phpunit"
        ]
    },
    "config": {
        "optimize-autoloader": true,
        "preferred-install": "dist",
        "sort-packages": true,
        "allow-plugins": {
            "php-http/discovery": true
        }
    },
    "minimum-stability": "dev",
    "prefer-stable": true
}
```

2. 依赖管理

版本约束策略

约束方式	示例	说明
精确版本	`1.2.3`	锁定特定版本
范围约束	`>=1.0 <2.0`	指定版本范围

约束方式	示例	说明
通配符	`1.0.*`	匹配 1.0 系列最新
波浪号	`~1.2`	允许最后一位升级 (1.2.0-1.2.9)
脱字号	`^1.2.3`	允许非破坏性升级 (1.2.3-1.9.9)

依赖安装最佳实践

```
# 生产依赖
composer require package/name

# 开发依赖
composer require --dev package/name

# 交互式选择版本
composer require package/name -W

# 全局安装 (谨慎使用)
composer global require friendsofphp/php-cs-fixer
```

3. 自动加载优化

优化策略

```
# 生成优化后的自动加载器
composer dump-autoload -o

# 生产环境使用权威类映射
composer dump-autoload --classmap-authoritative

# 开发时禁用类映射
composer dump-autoload --no-dev
```

PSR-4 自动加载示例

```
"autoload": {
    "psr-4": {
        "MyLibrary\\": "src/",
        "Vendor\\Module\\": "module/src/"
    },
    "files": ["src/helpers.php"],
    "exclude-from-classmap": ["tests/"]
}
```

4. □□□□

□□□□□

```
"scripts": {
    "post-install-cmd": [
        "@php artisan cache:clear"
    ],
    "post-update-cmd": [
        "@php artisan migrate"
    ],
    "test": [
        "@php vendor/bin/phpunit",
        "@php vendor/bin/phpstan analyse"
    ],
    "cs-check": "php-cs-fixer fix --dry-run --diff",
    "cs-fix": "php-cs-fixer fix",
    "deploy": [
        "@composer install --no-dev",
        "@php artisan optimize:clear",
        "@php artisan migrate --force"
    ]
}
```

□□□□□

```
namespace App\Scripts;

class DeploymentScript
{
    public static function run($event)
    {
        // 部署逻辑
    }
}
```

```
"scripts": {
    "post-deploy-cmd": ["App\\Scripts\\DeploymentScript::run"]
}
```

5. 性能优化

提升安装速度

```
# 并行安装 (Composer 2+)
composer install -j4

# 禁用 Xdebug
COMPOSER_ALLOW_XDEBUG=0 composer install

# 使用中国镜像
composer config -g repo.packagist composer https://mirrors.aliyun.com/composer/
```

缓存优化

```
# 清除缓存
composer clear-cache

# 预下载依赖 (CI/CD 环境)
composer install --prefer-dist --no-dev --no-scripts --no-progress
```

6. 安全实践

安全审计

```
# 检查已知漏洞
composer audit


# 更新有漏洞的包
composer update vendor/package --dry-run
```

锁定文件安全

```
# 验证 composer.lock 与 composer.json 同步
composer validate --no-check-all --strict


# 生产环境使用 --no-dev
composer install --no-dev
```

7. 多环境管理

环境区分

```
"require": {
    "php": "^8.1",
    "production-package": "^1.0"
},
"require-dev": {
    "phpunit/phpunit": "^10.0",
    "mockery/mockery": "^1.5"
},
"suggest": {
    "ext-redis": "Required for Redis support",
    "ext-gd": "Required for image processing"
```

```
}
```

平台配置

```
"config": {
    "platform": {
        "php": "8.1.12",
        "ext-redis": "5.3.7"
    }
}
```

8. 私有包管理

私有仓库配置

```
"repositories": [
    {
        "type": "composer",
        "url": "https://repo.pkg.example.com",
        "options": {
            "ssl": {
                "verify_peer": "true"
            }
        }
    },
    {
        "type": "vcs",
        "url": "git@github.com:mycompany/private-package.git"
    }
]
```

认证配置

```
# 添加认证信息
composer config http-basic.repo.pkg.example.com username password

# 或使用 auth.json (不提交到版本控制)
{
    "http-basic": {
        "repo.pkg.example.com": {
            "username": "token",
            "password": "your-api-token"
        }
    }
}
```

9. 工作流集成

CI/CD 集成示例

```
# .gitlab-ci.yml 示例
stages:
  - test
  - deploy

composer_install:
  stage: test
  script:
    - composer install --prefer-dist --no-progress --no-suggest
    - composer validate --strict
    - composer audit

run_tests:
  stage: test
  script:
    - composer test

deploy_production:
  stage: deploy
```

```
  script:
    - composer install --no-dev --optimize-autoloader
    - composer dump-env prod
  only:
    - main
```

10. 常见问题解决

版本冲突解决

```
# 查看依赖树
composer why vendor/package

# 查看哪些包依赖特定版本
composer depends --tree vendor/package

# 尝试更新单个包
composer update vendor/package --with-dependencies
```

调试技巧

```
# 详细输出
composer -vvv install

# 显示内存使用
COMPOSER_MEMORY_LIMIT=-1 composer update

# 忽略平台要求
composer install --ignore-platform-reqs
```

遵循这些最佳实践可以确保您的 PHP 项目依赖管理更加健壮、安全和高效。根据项目规模和环境需求适当调整这些策略。