Composer 安装与使用

分类 **编程技术**

Composer 是 PHP 的一个依赖管理工具。我们可以在项目中声明所依赖的外部工具库，Composer 会帮你安装这些依赖的库文件，有了它，我们就可以很轻松的使用一个命令将其他人的优秀代码引用到我们的项目中来。

Composer 默认情况下不是全局安装，而是基于指定的项目的某个目录中（例如 vendor）进行安装。

Composer 需要 PHP 5.3.2+ 以上版本，且需要开启 openssl。

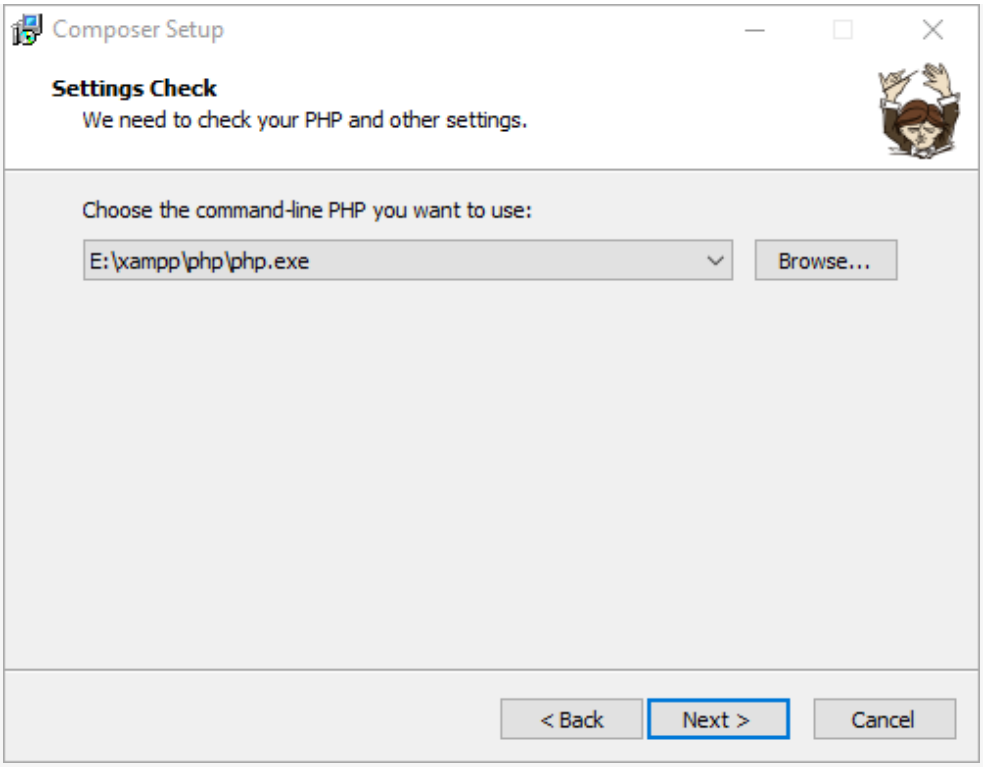
Composer 可运行在 Windows 、 Linux 以及 OSX 平台上。

Composer 的安装

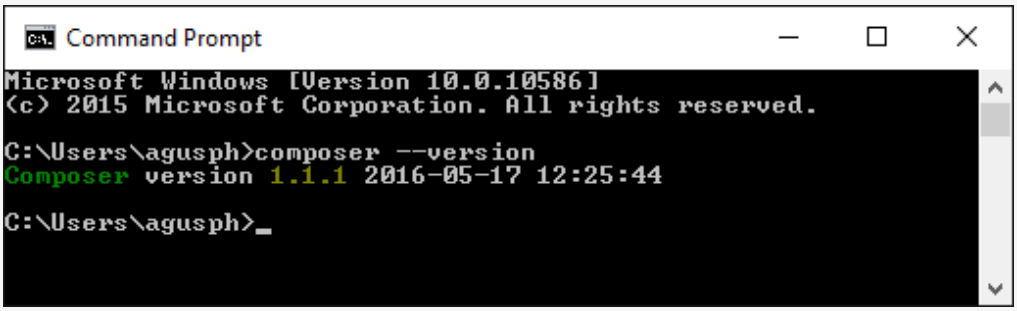
Wondows 平台

Wondows 平台上，我们只需要下载 Composer-Setup.exe 后，一步步安装即可。

需要注意的是你需要开启 openssl 配置，我们打开 php 目录下的 php.ini，将 `extension=php_openssl.dll` 前面的分号去掉就可以了。



安装成功后，我们可以通过命令窗口(cmd) 输入 `composer --version` 命令来查看是否安装成功：



接下来我们可以更改 Packagist 为国内镜像：

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

Linux 平台

Linux 平台可以使用以下命令来安装：

```
# php -r "copy('https://install.phpcomposer.com/installer', 'composer-setup.php');"
# php composer-setup.php

All settings correct for using Composer
Downloading...

Composer (version 1.6.5) successfully installed to: /root/composer.phar
Use it: php composer.phar
```

移动 composer.phar，这样 composer 就可以进行全局调用：

```
# mv composer.phar /usr/local/bin/composer
```

切换为国内镜像：

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

更新 composer：

```
# composer selfupdate
```

Mac OS 系统

Mac OS 系统可以使用以下命令来安装：

```
$ curl -sS https://getcomposer.org/installer | php
$ sudo mv composer.phar /usr/local/bin/composer
$ composer --version
Composer version 1.7.2 2018-08-16 16:57:12
```

切换为国内镜像：

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

更新 composer：

```
$ composer selfupdate
```

Composer 的使用

要使用 Composer，我们需要先在项目的目录下创建一个 composer.json 文件，文件描述了项目的依赖关系。

文件格式如下：

```
{
    "require": {
        "monolog/monolog": "1.2.*"
    }
}
```

以上文件说明我们需要下载从 1.2 开始的任何版本的 monolog。

接下来只要运行以下命令即可安装依赖包：

```
composer install
```

require 命令

除了使用 install 命令外，我们也可以使用 require 命令快速的安装一个依赖而不需要手动在 composer.json 里添加依赖信息：

```
$ composer require monolog/monolog
```

Composer 会先找到合适的版本，然后更新composer.json文件，在 require 那添加 monolog/monolog 包的相关信息，再把相关的依赖下载下来进行安装，最后更新 composer.lock 文件并生成 php 的自动加载文件。

update 命令

update 命令用于更新项目里所有的包，或者指定的某些包：

```
# 更新所有依赖
$ composer update

# 更新指定的包
$ composer update monolog/monolog

# 更新指定的多个包
$ composer update monolog/monolog symfony/dependency-injection

# 还可以通过通配符匹配包
$ composer update monolog/monolog symfony/*
```

需要注意的时，包能升级的版本会受到版本约束的约束，包不会升级到超出约束的版本的范围。例如如果 composer.json 里包的版本约束为 ^1.10，而最新版本为 2.0。那么 update 命令是不能把包升级到 2.0 版本的，只能最高升级到 1.x 版本。关于版本约束请看后面的介绍。

remove 命令

remove 命令用于移除一个包及其依赖（在依赖没有被其他包使用的情况下），如果依赖被其他包使用，则无法移除：

```
$ composer remove monolog/monolog
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 0 installs, 0 updates, 2 removals
  - Removing psr/log (1.0.2)
  - Removing monolog/monolog (1.23.0)
Generating autoload files
```

search 命令

search 命令可以搜索包：

```
$ composer search monolog
```

该命令会输出包及其描述信息，如果只想输出包名可以使用 `--only-name` 参数：

```
$ composer search --only-name monolog
```

show 命令

show 命令可以列出当前项目使用到包的信息：

```
# 列出所有已经安装的包
$ composer show

# 可以通过通配符进行筛选
$ composer show monolog/*

# 显示具体某个包的信息
$ composer show monolog/monolog
```

基本约束

精确版本

我们可以告诉 Composer 安装的具体版本，例如：1.0.2，指定 1.0.2 版本。

范围

通过使用比较操作符来指定包的范围。这些操作符包括：`>, >=, <, <=, !=`。

你可以定义多个范围，使用空格或者逗号，表示逻辑上的与，使用双竖线 || 表示逻辑上的或。其中与的优先级会大于或。 实例：

- >=1.0
- >=1.0 <2.0
- >=1.0 <1.1 || >=1.2

我们也可以通过使用连字符 `-` 来指定版本范围。

连字符的左边表明了 >= 的版本，如果右边的版本不是完整的版本号，则会被使用通配符进行补全。例如`1.0 - 2.0`等同于`>=1.0.0 <2.1`（`2.0`相当于`2.0.*`），而`1.0.0 - 2.1.0`则等同于`>=1.0.0 <=2.1.0`。

通配符

可以使用通配符来设置版本。1.0.*相当于>=1.0 <1.1。

例子：1.0.*

波浪号 ~

我们先通过后面这个例子去解释~操作符的用法：~1.2相当于>=1.2 <2.0.0，而~1.2.3相当于>=1.2.3 <1.3.0。对于使用Semantic Versioning作为版本号标准的项目来说，这种版本约束方式很实用。例如~1.2定义了最小的小版本号，然后你可以升级2.0以下的任何版本而不会出问题，因为按照Semantic Versioning的版本定义，小版本的升级不应该有兼容性的问题。简单来说，~定义了最小的版本，并且允许版本的最后一位版本号进行升级（没懂得话，请再看一边前面的例子）。

例子：~1.2

> *需要注意的是，如果~作用在主版本号上，例如~1，按照上面的说法，Composer可以安装版本1以后的主版本，但是事实上是~1会被当作~1.0对待，只能增加小版本，不能增加主版本。*

折音号 ^

^操作符的行为跟Semantic Versioning有比较大的关联，它允许升级版本到安全的版本。例如，^1.2.3相当于>=1.2.3 <2.0.0，因为在2.0版本前的版本应该都没有兼容性的问题。而对于1.0之前的版本，这种约束方式也考虑到了安全问题，例如^0.3会被当作>=0.3.0 <0.4.0对待。

例子：^1.2.3

版本稳定性

如果你没有显式的指定版本的稳定性，Composer会根据使用的操作符，默认在内部指定为-dev或者-stable。例如：

约束	内部约束
1.2.3	=1.2.3.0-stable
>1.2	>1.2.0.0-stable
>=1.2	>=1.2.0.0-dev
>=1.2-stable	>=1.2.0.0-stable
<1.3	<1.3.0.0-dev
<=1.3	<=1.3.0.0-stable
1 - 2	>=1.0.0.0-dev <3.0.0.0-dev
~1.3	>=1.3.0.0-dev <2.0.0.0-dev
1.4.*	>=1.4.0.0-dev <1.5.0.0-dev

例子：`1.0 - 2.0`
如果你想指定版本只要稳定版本，你可以在版本后面添加后缀`-stable`。
`minimum-stability` 配置项定义了包在选择版本时对稳定性的选择的默认行为。默认是`stable`。它的值如下（按照稳定性排序）：`dev`，`alpha`，`beta`，`RC`和`stable`。除了修改这个配置去修改这个默认行为，我们还可以通过稳定性标识（例如`@stable`和`@dev`）来安装一个相比于默认配置不同稳定性的版本。例如：

```
{
    "require": {
        "monolog/monolog": "1.0.*@beta",
        "acme/foo": "@dev"
    }
}
```

← const char*, char const*, char*const 的区别 Matplotlib 教程 →