参考：<http://docs.phpcomposer.com/>

中文文档

* [简介](http://docs.phpcomposer.com/00-intro.html)
* [基本用法](http://docs.phpcomposer.com/01-basic-usage.html)
* [库（资源包）](http://docs.phpcomposer.com/02-libraries.html)
* [命令行](http://docs.phpcomposer.com/03-cli.html)
* [架构](http://docs.phpcomposer.com/04-schema.html)
* [资源库](http://docs.phpcomposer.com/05-repositories.html)
* [社区](http://docs.phpcomposer.com/06-community.html)

# 简介

Composer 是 PHP 的一个依赖管理工具。它允许你声明项目所依赖的代码库，它会在你的项目中为你安装他们。

* [简介](http://docs.phpcomposer.com/00-intro.html#Introduction)
  + [依赖管理](http://docs.phpcomposer.com/00-intro.html#Dependency-management)
  + [声明依赖关系](http://docs.phpcomposer.com/00-intro.html#Declaring-dependencies)
  + [系统要求](http://docs.phpcomposer.com/00-intro.html#System-Requirements)
  + [安装 - \*nix](http://docs.phpcomposer.com/00-intro.html#Installation-*nix)
    - [下载 Composer 的可执行文件](http://docs.phpcomposer.com/00-intro.html#Downloading-the-Composer-Executable)
      * [局部安装](http://docs.phpcomposer.com/00-intro.html#Locally)
      * [全局安装](http://docs.phpcomposer.com/00-intro.html#Globally)
      * [全局安装 (on OSX via homebrew)](http://docs.phpcomposer.com/00-intro.html#Globally-on-OSX-via-homebrew)
  + [安装 - Windows](http://docs.phpcomposer.com/00-intro.html#Installation-Windows)
    - [使用安装程序](http://docs.phpcomposer.com/00-intro.html#Using-the-Installer)
    - [手动安装](http://docs.phpcomposer.com/00-intro.html#Manual-Installation)
  + [使用 Composer](http://docs.phpcomposer.com/00-intro.html#Using-Composer)
  + [自动加载](http://docs.phpcomposer.com/00-intro.html#Autoloading)

## 依赖管理

Composer 不是一个包管理器。是的，它涉及 "packages" 和 "libraries"，但它在每个项目的基础上进行管理，在你项目的某个目录中（例如 vendor）进行安装。默认情况下它不会在全局安装任何东西。因此，这仅仅是一个依赖管理。

这种想法并不新鲜，Composer 受到了 node's [npm](http://npmjs.org/" \t "_blank) （me：node将依赖包的下载到本地的node\_modules目录下）和 ruby's [bundler](http://gembundler.com/) 的强烈启发。而当时 PHP 下并没有类似的工具。

Composer 将这样为你解决问题：

a) 你有一个项目依赖于若干个库。

b) 其中一些库依赖于其他库。

c) 你声明你所依赖的东西。

d) Composer 会找出哪个版本的包需要安装，并安装它们（将它们下载到你的项目中）。

## 声明依赖关系

比方说，你正在创建一个项目，你需要一个库来做日志记录。你决定使用 [monolog](https://github.com/Seldaek/monolog)。为了将它添加到你的项目中，你所需要做的就是创建一个 composer.json 文件，其中描述了项目的依赖关系。

{

"require": {

"monolog/monolog": "1.2.\*"

}

}

我们只要指出我们的项目需要一些 monolog/monolog 的包，从 1.2 开始的任何版本。

## 系统要求

运行 Composer 需要 PHP 5.3.2+ 以上版本。一些敏感的 PHP 设置和编译标志也是必须的，但对于任何不兼容项安装程序都会抛出警告。

我们将从包的来源直接安装，而不是简单的下载 zip 文件，你需要 git 、 svn 或者 hg ，这取决于你载入的包所使用的版本管理系统。

Composer 是多平台的，我们努力使它在 Windows 、 Linux 以及 OSX 平台上运行的同样出色。

## 安装 - \*nix

### 下载 Composer 的可执行文件

#### 局部安装

要真正获取 Composer，我们需要做两件事。首先安装 Composer （同样的，这意味着它将下载到你的项目中）：

curl -sS https://getcomposer.org/installer | php

**注意：** 如果上述方法由于某些原因失败了，你还可以通过 php >下载安装器：

php -r "readfile('https://getcomposer.org/installer');" | php

这将检查一些 PHP 的设置，然后下载 composer.phar 到你的工作目录中。这是 Composer 的二进制文件。这是一个 PHAR 包（PHP 的归档），这是 PHP 的归档格式可以帮助用户在命令行中执行一些操作。

你可以通过 --install-dir 选项指定 Composer 的安装目录（它可以是一个绝对或相对路径）：

curl -sS https://getcomposer.org/installer | php -- --install-dir=bin

#### 全局安装

你可以将此文件放在任何地方。如果你把它放在系统的 PATH 目录中，你就能在全局访问它。 在类Unix系统中，你甚至可以在使用时不加 php 前缀。

你可以执行这些命令让 composer 在你的系统中进行全局调用：

curl -sS https://getcomposer.org/installer | php

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

**注意：** 如果上诉命令因为权限执行失败， 请使用 sudo 再次尝试运行 mv 那行命令。

现在只需要运行 composer 命令就可以使用 Composer 而不需要输入 php composer.phar。

#### 全局安装 (on OSX via homebrew)

Composer 是 homebrew-php 项目的一部分。

brew update

brew tap josegonzalez/homebrew-php

brew tap homebrew/versions

brew install php55-intl

brew install josegonzalez/php/composer

## 安装 - Windows

### 使用安装程序

这是将 Composer 安装在你机器上的最简单的方法。

下载并且运行 [Composer-Setup.exe](https://getcomposer.org/Composer-Setup.exe)，它将安装最新版本的 Composer ，并设置好系统的环境变量，因此你可以在任何目录下直接使用 composer 命令。

### 手动安装

设置系统的环境变量 PATH 并运行安装命令下载 composer.phar 文件：

C:\Users\username>cd C:\bin

C:\bin>php -r "readfile('https://getcomposer.org/installer');" | php

**注意：** 如果收到 readfile 错误提示，请使用 http 链接或者在 php.ini 中开启 php\_openssl.dll 。

在 composer.phar 同级目录下新建文件 composer.bat ：

C:\bin>echo @php "%~dp0composer.phar" %\*>composer.bat

关闭当前的命令行窗口，打开新的命令行窗口进行测试：

C:\Users\username>composer -V

Composer version 1.5.1 2017-08-09 16:07:22

## 使用 Composer

现在我们将使用 Composer 来安装项目的依赖。如果在当前目录下没有一个 composer.json 文件，请查看[基本用法](http://docs.phpcomposer.com/01-basic-usage.html)章节。

要解决和下载依赖，请执行 install 命令：

php composer.phar install

如果你进行了全局安装，并且没有 phar 文件在当前目录，请使用下面的命令代替：

composer install

继续 [上面的例子](http://docs.phpcomposer.com/00-intro.html#Declaring-dependencies)，这里将下载 monolog 到 vendor/monolog/monolog 目录（me:默认依赖包都下载到了vendor下）。

## 自动加载

除了库的下载，Composer 还准备了一个自动加载文件，它可以加载 Composer 下载的库中所有的类文件。使用它，你只需要将下面这行代码添加到你项目的引导文件中：

require 'vendor/autoload.php';

现在我们就可以使用 monolog 了！想要学习更多关于 Composer 的知识，请查看“基本用法”章节。

[基本用法](http://docs.phpcomposer.com/01-basic-usage.html) →

# 基本用法

* [基本用法](http://docs.phpcomposer.com/01-basic-usage.html#Basic-usage)
  + [安装](http://docs.phpcomposer.com/01-basic-usage.html#Installation)
  + [composer.json：项目安装](http://docs.phpcomposer.com/01-basic-usage.html#composer.json-Project-Setup)
    - [关于 require Key](http://docs.phpcomposer.com/01-basic-usage.html#The-require-Key)
    - [包名称](http://docs.phpcomposer.com/01-basic-usage.html#Package-Names)
    - [包版本](http://docs.phpcomposer.com/01-basic-usage.html#Package-Versions)
    - [下一个重要版本（波浪号运算符）](http://docs.phpcomposer.com/01-basic-usage.html#Next-Significant-Release)
    - [稳定性](http://docs.phpcomposer.com/01-basic-usage.html#Stability)
  + [安装依赖包](http://docs.phpcomposer.com/01-basic-usage.html#Installing-Dependencies)
  + [composer.lock - 锁文件](http://docs.phpcomposer.com/01-basic-usage.html#composer.lock-The-Lock-File)
  + [Packagist](http://docs.phpcomposer.com/01-basic-usage.html#Packagist)
  + [自动加载](http://docs.phpcomposer.com/01-basic-usage.html#Autoloading)

## 安装

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

curl -sS https://getcomposer.org/installer | php

详细请查看 [简介](http://docs.phpcomposer.com/00-intro.html) 章节。

要检查 Composer 是否正常工作，只需要通过 php 来执行 PHAR：

php composer.phar（me：命令查看帮助，当前目录下需要先下载composer.phar文件，或者安装composer到全局，这样就不用每次都要在工作空间下下载composer.phar；composer命令和php composer.phar的命令是一样的）

这将返回给你一个可执行的命令列表。

**注意：** 你也可以仅执行 --check 选项而无需下载 Composer。 要获取更多的信息请使用 --help。

curl -sS https://getcomposer.org/installer | php -- --help

## composer.json：项目安装

要开始在你的项目中使用 Composer，你只需要一个 composer.json 文件。该文件包含了项目的依赖和其它的一些元数据。

这个 [JSON format](http://json.org/) 是很容易编写的。它允许你定义嵌套结构。

### 关于 require Key

第一件事情（并且往往只需要做这一件事），你需要在 composer.json 文件中指定 require key 的值。你只需要简单的告诉 Composer 你的项目需要依赖哪些包。

{

"require": {

"monolog/monolog": "1.0.\*"

}

}

你可以看到， require 需要一个 **包名称** （例如 monolog/monolog） 映射到 **包版本** （例如 1.0.\*） 的对象。

### 包名称

包名称由供应商名称和其项目名称构成。通常容易产生相同的项目名称，而供应商名称的存在则很好的解决了命名冲突的问题。它允许两个不同的人创建同样名为 json 的库，而之后它们将被命名为 igorw/json 和 seldaek/json。

这里我们需要引入 monolog/monolog，供应商名称与项目的名称相同，对于一个具有唯一名称的项目，我们推荐这么做。它还允许以后在同一个命名空间添加更多的相关项目。如果你维护着一个库，这将使你可以很容易的把它分离成更小的部分。

### 包版本

在前面的例子中，我们引入的 monolog 版本指定为 1.0.\*。这表示任何从 1.0 开始的开发分支，它将会匹配 1.0.0、1.0.2 或者 1.0.20。

版本约束可以用几个不同的方法来指定。

| 名称 | 实例 | 描述 |
| --- | --- | --- |
| 确切的版本号 | 1.0.2 | 你可以指定包的确切版本。 |
| 范围 | >=1.0 >=1.0,<2.0>=1.0,<1.1|>=1.2 | 通过使用比较操作符可以指定有效的版本范围。  有效的运算符：>、>=、<、<=、!=。  你可以定义多个范围，用逗号隔开，这将被视为一个**逻辑AND**处理。一个管道符号|将作为**逻辑OR**处理。  AND 的优先级高于 OR。 |
| 通配符 | 1.0.\* | 你可以使用通配符\*来指定一种模式。1.0.\*与>=1.0,<1.1是等效的。 |
| 赋值运算符 | ~1.2 | 这对于遵循语义化版本号的项目非常有用。~1.2相当于>=1.2,<2.0。想要了解更多，请阅读下一小节。 |

### 下一个重要版本（波浪号运算符）

~ 最好用例子来解释： ~1.2 相当于 >=1.2,<2.0，而 ~1.2.3 相当于 >=1.2.3,<1.3。正如你所看到的这对于遵循 [语义化版本号](http://semver.org/) 的项目最有用。一个常见的用法是标记你所依赖的最低版本，像 ~1.2 （允许1.2以上的任何版本，但不包括2.0）。由于理论上直到2.0应该都没有向后兼容性问题，所以效果很好。你还会看到它的另一种用法，使用 ~ 指定最低版本，但允许版本号的最后一位数字上升。

**注意：** 虽然 2.0-beta.1 严格地说是早于 2.0，但是，根据版本约束条件， 例如 ~1.2 却不会安装这个版本。就像前面所讲的 ~1.2 只意味着 .2 部分可以改变，但是 1. 部分是固定的。

### 稳定性

默认情况下只有稳定的发行版才会被考虑在内。如果你也想获得 RC、beta、alpha 或 dev 版本，你可以使用 [稳定标志](http://docs.phpcomposer.com/04-schema.html#Package-links)。你可以对所有的包做 [最小稳定性](http://docs.phpcomposer.com/04-schema.html#minimum-stability) 设置，而不是每个依赖逐一设置。

## 安装依赖包

获取定义的依赖到你的本地项目，只需要调用 composer.phar 运行 install 命令。

php composer.phar install（me：如果在全局安装了composer,也可以使用composer install命令，并且不需要在每个工作空间下载composer.phar）

接着前面的例子，这将会找到 monolog/monolog 的最新版本，并将它下载到 vendor 目录。 这是一个惯例把第三方的代码到一个指定的目录 vendor。如果是 monolog 将会创建 vendor/monolog/monolog 目录。

**小技巧：** 如果你正在使用Git来管理你的项目， 你可能要添加 vendor 到你的 .gitignore 文件中。 你不会希望将所有的代码都添加到你的版本库中。

另一件事是 install 命令将创建一个 composer.lock 文件到你项目的根目录中。

## composer.lock - 锁文件

在安装依赖后，Composer 将把安装时确切的版本号列表写入 composer.lock 文件。这将锁定改项目的特定版本。

**请提交你应用程序的 composer.lock （包括 composer.json）到你的版本库中**

这是非常重要的，因为 install 命令将会检查锁文件是否存在，如果存在，它将下载指定的版本（忽略 composer.json 文件中的定义）。

me：composer install 命令会查看当前目录下的composer.lock，按照指定地版本下载依赖，并据此创建composer.lock文件，之后依赖的下载只会参照composer.lock，忽略composer.json

这意味着，任何人建立项目都将下载与指定版本完全相同的依赖。你的持续集成服务器、生产环境、你团队中的其他开发人员、每件事、每个人都使用相同的依赖，从而减轻潜在的错误对部署的影响。即使你独自开发项目，在六个月内重新安装项目时，你也可以放心的继续工作，即使从那时起你的依赖已经发布了许多新的版本。

如果不存在 composer.lock 文件，Composer 将读取 composer.json 并创建锁文件。

这意味着如果你的依赖更新了新的版本，你将不会获得任何更新。此时要更新你的依赖版本请使用 update 命令。这将获取最新匹配的版本（根据你的 composer.json 文件）并将新版本更新进锁文件。

php composer.phar update

或

Composer update命令（如果安装了composer的话）

(me:如果你修改了composer.json文件的话。当在工作目录下执行composer Install命令会检查composer.lock是否是composer.json生成而来，不是则会提示update；

更新命令会重新读取composer.json文件以更新依赖或者下载新的依赖包，然后将更新的依赖的版本更新到composer.lock文件中)

如果只想安装或更新一个依赖，你可以白名单（名单的值为依赖的包名-提供商的名+项目名）它们：

php composer.phar update monolog/monolog [...]

**注意：** 对于库，并不一定建议提交锁文件 请参考：[库的锁文件](http://docs.phpcomposer.com/02-libraries.html#Lock-file).

## Packagist

[packagist](https://packagist.org/)（https://packagist.org/） 是 Composer 的主要资源库。 一个 Composer 的库基本上是一个包的源：记录了可以得到包的地方。Packagist 的目标是成为大家使用库资源的中央存储平台（me：就像java的maven）。这意味着你可以 require 那里的任何包。

当你访问 [packagist website](https://packagist.org/" \t "_blank) (packagist.org)，你可以浏览和搜索资源包。

任何支持 Composer 的开源项目应该发布自己的包在 packagist 上。虽然并不一定要发布在 packagist 上来使用 Composer，但它使我们的编程生活更加轻松。

## 自动加载

对于库的自动加载信息，Composer 生成了一个 vendor/autoload.php 文件。你可以简单的引入这个文件，你会得到一个免费的自动加载支持。

require 'vendor/autoload.php';

这使得你可以很容易的使用第三方代码。例如：如果你的项目依赖 monolog，你就可以像这样开始使用这个类库，并且他们将被自动加载。

$log = new Monolog\Logger('name');

$log->pushHandler(new Monolog\Handler\StreamHandler('app.log', Monolog\Logger::WARNING));

$log->addWarning('Foo');

你可以在 composer.json 的 autoload 字段中增加自己的 autoloader。

{

"autoload": {

"psr-4": {"Acme\\": "src/"}

}

}

Composer 将注册一个 [PSR-4](http://www.php-fig.org/psr/psr-4/) autoloader 到 Acme 命名空间。

你可以定义一个从命名空间到目录的映射。此时 src 会在你项目的根目录，与 vendor 文件夹同级。例如 src/Foo.php 文件应该包含 Acme\Foo 类。

添加 autoload 字段后，你应该再次运行 install 命令来生成 vendor/autoload.php 文件。

引用这个文件也将返回 autoloader 的实例，你可以将包含调用的返回值存储在变量中，并添加更多的命名空间。这对于在一个测试套件中自动加载类文件是非常有用的，例如。

$loader = require 'vendor/autoload.php';

$loader->add('Acme\\Test\\', \_\_DIR\_\_);

除了 PSR-4 自动加载，classmap 也是支持的。这允许类被自动加载，即使不符合 PSR-0 规范。详细请查看 [自动加载-参考](http://docs.phpcomposer.com/04-schema.html#autoload)。

**注意：** Composer 提供了自己的 autoloader。如果你不想使用它，你可以仅仅引入 vendor/composer/autoload\_\*.php 文件，它返回一个关联数组，你可以通过这个关联数组配置自己的 autoloader。

← [简介](http://docs.phpcomposer.com/00-intro.html) | [库（资源包）](http://docs.phpcomposer.com/02-libraries.html) →

# 库（资源包）

本章将告诉你如何通过 Composer 来安装你的库。

* [库（资源包）](http://docs.phpcomposer.com/02-libraries.html#Libraries)
  + [每一个项目都是一个包](http://docs.phpcomposer.com/02-libraries.html#Every-project-is-a-package)
  + [平台软件包](http://docs.phpcomposer.com/02-libraries.html#Platform-packages)
  + [指明版本](http://docs.phpcomposer.com/02-libraries.html#Specifying-the-version)
    - [标签](http://docs.phpcomposer.com/02-libraries.html#Tags)
    - [分支](http://docs.phpcomposer.com/02-libraries.html#Branches)
    - [别名](http://docs.phpcomposer.com/02-libraries.html#Aliases)
  + [锁文件](http://docs.phpcomposer.com/02-libraries.html#Lock-file)
  + [发布到 VCS（线上版本控制系统）](http://docs.phpcomposer.com/02-libraries.html#Publishing-to-a-VCS)
  + [发布到 packagist](http://docs.phpcomposer.com/02-libraries.html#Publishing-to-packagist)

## 每一个项目都是一个包

只要你有一个 composer.json 文件在目录中，那么整个目录就是一个包(me:整个目录都可以作为包发布到仓库中供其他人使用,从仓库中下载的依赖包也是这样的格式，他门的包内目录页含有composer.json文件定义着它的依赖)。当你添加一个 require 到项目中，你就是在创建一个依赖于其它库的包。你的项目和库之间唯一的区别是，你的项目是一个没有名字的包。

为了使它成为一个可安装的包，你需要给它一个名称(me：包名是提供商的名字+项目的名字)。你可以通过 composer.json 中的 name 来定义：

{

"name": "acme/hello-world",

"require": {

"monolog/monolog": "1.0.\*"

}

}

在这种情况下项目的名称为 acme/hello-world，其中 acme 是供应商的名称。供应商的名称是必须填写的。

**注意：** 如果你不知道拿什么作为供应商的名称， 那么使用你 github 上的用户名通常是不错的选择。 虽然包名不区分大小写，但惯例是使用小写字母，并用连字符作为单词的分隔。

## 平台软件包

Composer 将那些已经安装在系统上，但并不是由 Composer 安装的包视为一个虚拟的平台软件包。这包括PHP本身，PHP扩展和一些系统库。

* php 表示用户的 PHP 版本要求，你可以对其做出限制。例如 >=5.4.0。如果需要64位版本的 PHP，你可以使用 php-64bit 进行限制。
* hhvm 代表的是 HHVM（也就是 HipHop Virtual Machine） 运行环境的版本，并且允许你设置一个版本限制，例如，'>=2.3.3'。
* ext-<name> 可以帮你指定需要的 PHP 扩展（包括核心扩展）。通常 PHP 拓展的版本可以是不一致的，将它们的版本约束为 \* 是一个不错的主意。一个 PHP 扩展包的例子：包名可以写成 ext-gd。
* lib-<name> 允许对 PHP 库的版本进行限制。  
  以下是可供使用的名称：curl、iconv、icu、libxml、openssl、pcre、uuid、xsl。

你可以使用 composer show --platform 命令来获取可用的平台软件包的列表。

## 指明版本

你需要一些方法来指明自己开发的包的版本，当你在 Packagist 上发布自己的包，它能够从 VCS (git, svn, hg) 的信息推断出包的版本（me：但是需要vcs上指明开发包的版本，如git的tag等），因此你不必手动指明版本号，并且也不建议这样做。请查看 [标签](http://docs.phpcomposer.com/02-libraries.html#Tags) 和 [分支](http://docs.phpcomposer.com/02-libraries.html#Branches) 来了解版本号是如何被提取的。

如果你想要手动创建并且真的要明确指定它，你只需要添加一个 version 字段：

{

"version": "1.0.0"

}

**注意：** 你应该尽量避免手动设置版本号（me：最好使用vcs提供的版本功能，这样也便于管理），因为标签的值必须与标签名相匹配。

### 标签

对于每一个看起来像版本号的标签，都会相应的创建一个包的版本。它应该符合 'X.Y.Z' 或者 'vX.Y.Z' 的形式，-patch、-alpha、-beta 或 -RC 这些后缀是可选的。在后缀之后也可以再跟上一个数字。

下面是有效的标签名称的几个例子：

* 1.0.0
* v1.0.0
* 1.10.5-RC1
* v4.4.4beta2
* v2.0.0-alpha
* v2.0.4-p1

**注意：** 即使你的标签带有前缀 v， 由于在需要 require 一个[版本的约束](http://docs.phpcomposer.com/01-basic-usage.html#Package-Versions)时是不允许这种前缀的， 因此 v 将被省略（例如标签 V1.0.0 将创建 1.0.0 版本）。

### 分支

对于每一个分支，都会相应的创建一个包的开发版本。如果分支名看起来像一个版本号，那么将创建一个如同 {分支名}-dev 的包版本号。例如一个分支 2.0 将产生一个 2.0.x-dev 包版本（加入了 .x 是出于技术的原因，以确保它被识别为一个分支，而 2.0.x 的分支名称也是允许的，它同样会被转换为 2.0.x-dev）。如果分支名看起来不像一个版本号，它将会创建 dev-{分支名} 形式的版本号。例如 master 将产生一个 dev-master的版本号。

下面是版本分支名称的一些示例：

* 1.x
* 1.0 (equals 1.0.x)
* 1.1.x

**注意：** 当你安装一个新的版本时，将会自动从它 source 中拉取。 详细请查看 [install](http://docs.phpcomposer.com/03-cli.html#install) 命令。

### 别名

它表示一个包版本的别名。例如，你可以为 dev-master 设置别名 1.0.x-dev，这样就可以通过 require 1.0.x-dev 来得到 dev-master 版本的包。

详细请查看[“别名”](http://docs.phpcomposer.com/articles/aliases.html)。

## 锁文件

如果你愿意，可以在你的项目中提交 composer.lock 文件。他将帮助你的团队始终针对同一个依赖版本进行测试（me：除非执行了update命令才能重新读取composer.json，并据此来更新composer.lock文件）。任何时候，这个锁文件都只对于你的项目产生影响。

如果你不想提交锁文件，并且你正在使用 Git，那么请将它添加到 .gitignore 文件中。

## 发布到 VCS（线上版本控制系统）

一旦你有一个包含 composer.json 文件的库存储在线上版本控制系统（例如：Git），你的库就可以被 Composer 所安装。在这个例子中，我们将 acme/hello-world 库发布在 GitHub 上的 github.com/username/hello-world 中。

现在测试这个 acme/hello-world 包，我们在本地创建一个新的项目。我们将它命名为 acme/blog。此博客将依赖 acme/hello-world，而后者又依赖 monolog/monolog。我们可以在某处创建一个新的 blog 文件夹来完成它，并且需要包含 composer.json 文件：

{

"name": "acme/blog",

"require": {

"acme/hello-world": "dev-master"

}

}

在这个例子中 name 不是必须的，因为我们并不想将它发布为一个库。在这里为 composer.json 文件添加描述。

现在我们需要告诉我们的应用，在哪里可以找到 hello-world 的依赖。为此我们需要在 composer.json 中添加 repositories 来源申明：

{

"name": "acme/blog",

"repositories": [

{

"type": "vcs",

"url": "https://github.com/username/hello-world"

}

],

"require": {

"acme/hello-world": "dev-master"

}

}

更多关于包的来源是如何工作的，以及还有什么其他的类型可供选择，请查看[资源库](http://docs.phpcomposer.com/05-repositories.html)。

这就是全部了。你现在可以使用 Composer 的 install 命令来安装你的依赖包了！

**小结：** 任何含有 composer.json 的 GIT、SVN、HG 存储库，都可以通过 require 字段指定“包来源”和“声明依赖”来添加到你的项目中。

## 发布到 packagist

好的，你现在可以发布你的包了，但你不会希望你的用户每次都这样繁琐的指定包的来源。

你可能注意到了另一件事，我们并没有指定 monolog/monolog 的来源。它是怎么工作的？答案是 packagist。

[Packagist](https://packagist.org/) 是 Composer 主要的一个包信息存储库，它默认是启用的。任何在 packagist 上发布的包都可以直接被 Composer 使用。就像 monolog 它被 [发布在 packagist 上](https://packagist.org/packages/monolog/monolog)，我们可以直接使用它，而不必指定任何额外的来源信息。

如果我们想与世界分享我们的 hello-world，我们最好将它发布到 packagist 上。这样做是很容易的。

你只需要点击那个大大的 "Submit Package" 按钮并注册。接着提交你库的来源地址，此时 packagist 就开始了抓取。一旦完成，你的包将可以提供给任何人使用。

← [基本用法](http://docs.phpcomposer.com/01-basic-usage.html) | [命令行](http://docs.phpcomposer.com/03-cli.html) →