一、下载源地址：

1、中国镜像：<https://packagist.phpcomposer.com>

2、镜像配置

a）全局配置：打开命令行窗口（windows用户）或控制台（Linux、Mac 用户）并执行如下命令：composer config -g repo.packagist composer <https://packagist.phpcomposer.com>

b）单项目配置：打开命令行窗口（windows用户）或控制台（Linux、Mac 用户），进入你的项目的根目录（也就是 composer.json 文件所在目录），执行如下命令：

"repositories": {

"packagist": {

"type": "composer",

"url": "https://packagist.phpcomposer.com"

}

}

3、解除镜像：composer config -g --unset repos.packagist

二、内容详解

1、引用地址：

<https://blog.csdn.net/qq_23864697/article/details/80116576>

<https://docs.phpcomposer.com/04-schema.html>

介绍：

1、name：包的名字，由供应方（vendor）名和项目名组成，用 / 分隔，发布包的时候用，包名称不能和当前存在所有的包重名

2、description：对包的一个简短描述，通常是一行的长度。在发布包的时候需要填

3、version：包的版本，格式必须是 X.Y.Z，选择性后缀：-dev、-alphaN、-betaN、-RCN。

4、type：包的类型，默认为 library，如果用于定制安装逻辑，可以配置如下类型symfony-bundle、wordpress-plugin、typo3-module，可以被特定项目用，特定项目提供安装器安装

提供3种类型

library：默认值。它将复制文件到 vendor 目录

project：它表示这是个项目，而不是库。比如像 Symfony 标准版这种应用

metapackage：一个含有依赖的空包，能触发安装，但不包含文件，不会向文件系统写任何东西

composer-install：为其他的定制类型的包提供安装器的包。

5、keywords：一个与包相关的关键词数组。用于包的搜索和过滤 ----》可选

6、homepage：项目的网站 URL ----》可选

7、time：版本发布时间。必须是 YYYY-MM-DD 或 YYYY-MM-DD HH:MM:SS 格式 ----》可选

8、license：包的许可证。可以是字符串或字符串数组 ----》可选

开源软件license

GPL：General Public License，只要一个软件使用了GPL协议的产品，则该软件也必须采用GPL协议，即衍生或修改后的代码，不可用于闭源的商业软件销售和发布——传染性

BSD：Berkeley Software Distribution，使用者可以自由的使用、修改源码，甚至在源码基础上二次开发后进行商用发布和销售

MIT：Massachusetts Institution of Technology，X条款，MIT与BSD较为类似，差异较小。即在二次开发后的发行版中，需要包含原许可证声明。

MPL：Mozilla Public License，为了更好的平衡“开发者对源码的需求和他们利用源代码获得的收益

Apache License 2.0：该许可协议与BSD/MIT协议类似，属于比较宽松、商业友好的开源协议。只需要使用者在使用了该协议下的源代码后，再发布后，依然带有对源代码的协议、商标、及其他作者规定的说明，即可。

LGPL：Lesser General Public License，GPL V2，LGPL 允许商业软件通过类库引用(link)方式使用LGPL类库而不需要开源商业软件的代码。但如果是修改LGPL的代码或者衍生的代码，则所有修改或衍生的代码，均需要遵循LGPL协议。

最常见的：Apache-2.0、BSD-2-Clause、BSD-3-Clause、BSD-4-Clause、GPL-2.0、GPL-2.0+、GPL-3.0、GPL-3.0+、LGPL-2.1、LGPL-2.1+、LGPL-3.0、LGPL-3.0+、MIT

**{**

**"license": "MIT"**

**}**

**{**

**"license": [**

**"LGPL-2.1",**

**"GPL-3.0+"**

**]**

**}**

**{**

**"license": "(LGPL-2.1 or GPL-3.0+)"**

**}**

9、authors：包的作者，对象数组

name：作者名字  
email：作者邮箱  
homepage：作者网站 URL  
role：作者在项目中的角色（如：developer 或 translator）

**{**

**"authors": [**

**{**

**"name": "Nils Adermann",**

**"email": "naderman@naderman.de",**

**"homepage": "http://www.naderman.de",**

**"role": "Developer"**

**},**

**{**

**"name": "Jordi Boggiano",**

**"email": "j.boggiano@seld.be",**

**"homepage": "http://seld.be",**

**"role": "Developer"**

**}**

**]**

**}**

10、support：各种关于该项目如何获取支持的信息

email：获取支持的邮箱  
issues：问题跟踪的 URL  
forum：论坛的 URL  
wiki：Wiki 的 URL  
irc：IRC 的频道，类似于 irc://server/channel  
source：查看或下载源码的 URL

**{**

**"support": {**

**"email": "support@example.org",**

**"irc": "irc://irc.freenode.org/composer"**

**}**

**}**

11、Package links：依赖包的映射表，由包名映射版本约束（对象操作）

require：列出包所依赖的包，除非这些依赖已经存在，否则这个包不会被安装。

"require": {  
 "php-ffmpeg/php-ffmpeg": "^0.14.0",  
 "jaeger/querylist": "^4.1"  
},

require-dev（root-only）：列出开发这个包（或跑测试等等）所依赖的包。在使用 install 命令时，只有带上 “–dev” 参数才能安装 dev 包。在使用 update 命令时，带上 “–no-dev” 则不更新。------》 主本地开发

"require-dev": {  
 "php-ffmpeg/php-ffmpeg": "^0.14.0",  
 "jaeger/querylist": "^4.1"  
},

conflict：列出包会和哪些包发生冲突。它们将不被允许和你的包一起安装。如果约束了版本，则只会针对特定的版本

replace：列出哪些包要被这个包替代。

provide：这个包所推荐的包列表。这个对公共接口最有用，一个包可以依赖一个虚拟的 logger 包，而实现 logger 接口的库可以放到 provide 字段中。

12、suggest：建议一些能让这个包工作的更好或得到增强的包列表。这些信息只在包安装完成时给出，暗示用户可以添加更多包，虽然不是必须要安装的。

13、autoload：提供给 PHP autoloader 的自动加载映射。psr-0、psr-4、classmap、files

psr-0：定义一个命名空间到路径的映射表，相对于包的根目录，命名空间的声明得以 \\ 结尾，确保自动加载器正确响应。安装或更新时生成的文件中查看：vendor/composer/autoload\_namespaces.php

**{**

**"autoload": {**

**"psr-0": {**

**"Monolog\\": "src/",**

**"Vendor\\Namespace\\": "src/",**

**"Vendor\_Namespace\_": "src/"**

**}**

**}**

**}**

如果你需要在多个目录里查找同一个前缀的命名空间，你可以用数组，如：

**{**

**"autoload": {**

**"psr-0": { "Monolog\\": ["src/", "lib/"] }**

**}**

**}**

PSR-0 风格并不局限于加载命名空间的声明的东西，也可以用于类这个层级。当库中只有一个在全局命名空间中的类时，这种方式就能用上。比如你有个 PHP 源文件放在项目的根目录，

**{**

**"autoload": {**

**"psr-0": { "UniqueGlobalClass": "" }**

**}**

**}**

如果你想设置一个目录作为任何命名空间的备用目录，你可以使用空的前缀，像这样

**{**

**"autoload": {**

**"psr-0": { "": "src/" }**

**}**

**}**

psr-4：定义从命名空间到路径的映射，相对于包的路径，当自动载入一个类像Foo\\Bar\\Baz，命名空间Foo\\指向文件夹目录src/，意思自动加载将会查找文件名为src/Bar/Baz.php的文件，如果存在将会引用。注意这个和psr-0的风格不一样，Foo\\不会出现在路径中。命名空间前缀必须以\\结束，避免与相似的冲突，例如Foo将会在FooBar命名空间匹配类。安装或更新完成文件中查看：vendor/composer/autoload\_psr4.php

**{**

**"autoload": {**

**"psr-4": {**

**"Monolog\\": "src/",**

**"Vendor\\Namespace\\": ""**

**}**

**}**

**}**

如果你需要在多个目录里查找同一个前缀的命名空间，你可以用数组

**{**

**"autoload": {**

**"psr-4": { "Monolog\\": ["src/", "lib/"] }**

**}**

**}**

如果你想设置一个目录作为任何命名空间的备用目录，你可以使用空的前缀

**{**

**"autoload": {**

**"psr-4": { "": "src/" }**

**}**

**}**

Classmap：安装或更新时生成的文件中：vendor/composer/autoload\_classmap.php。类映射表是通过扫描指定的目录或文件下的所有的 .php 和 .inc 文件生成的。

可以给任何不支持 PSR-0/4 的库用 classmap 生成器实现自动加载，配置上只要指定类所在的目录或文件即可

**{**

**"autoload": {**

**"classmap": ["src/", "lib/", "Something.php"]**

**}**

**}**

files：如果你想要明确的指定，在每次请求时都要载入某些文件，那么你可以使用 'files' autoloading。通常作为函数库的载入方式（而非类库）。

**{**

**"autoload": {**

**"files": ["src/MyLibrary/functions.php"]**

**}**

**}**

autoload-dev (root-only)：测试用到的路径

**{**

**"autoload": {**

**"psr-4": { "MyLibrary\\": "src/" }**

**},**

**"autoload-dev": {**

**"psr-4": { "MyLibrary\\Tests\\": "tests/" }**

**}**

**}**

14、include-path：将被弃用，它的功能由 autoload 代替。其实就是设置 include\_path ---》 可选

**{**

**"include-path": ["lib/"]**

**}**

15、target-dir：指定安装目标路径，如果包的根目录是在命名空间下，自动加载就不正确了，所以才有 target-dir 来解决这个问题。校正目录。 ----》可选

**{**

**"autoload": {**

**"psr-0": { "Symfony\\Component\\Yaml\\": "" }**

**},**

**"target-dir": "Symfony/Component/Yaml"**

**}**

16、minimum-stability（root-only）：定义根据稳定性如何过滤包。默认是 stable，如果你信赖一个 dev 包，你需要指明。可用的稳定性标识（按字母排序）：dev、alpha、beta、RC、stable

17、prefer-stable（root-only）：如果开启，Composer 会在稳定包和不稳定包中选择前者。

18、repositories（root-only）：定制包的仓库地址。默认为Packagist 仓库。通过指定仓库地址，你可以从任何地方获取包。仓库不能递归。你只能将它们添加到主的 composer.json 中。所依赖包中 composer.json 文件中的仓库定义是被忽略的

composer：通过网络提供 packages.json 文件，它包含一个 composer.json 对象的列表，还有额外的 dist 或 source 信息。packages.json 文件通过 PHP 流加载

vcs：版本控制系统仓库，如：git、svn、hg。

pear：你可以导入任何 pear 仓库到你的项目中

package：如果你依赖一个不支持 composer 的项目，你可以定义一个 package 类型的仓库，然后将 composer.json 对象直接写入。

**{**

**"repositories": [**

**{**

**"type": "composer",**

**"url": "http://packages.example.com"**

**},**

**{**

**"type": "composer",**

**"url": "https://packages.example.com",**

**"options": {**

**"ssl": {**

**"verify\_peer": "true"**

**}**

**}**

**},**

**{**

**"type": "vcs",**

**"url": "https://github.com/Seldaek/monolog"**

**},**

**{**

**"type": "pear",**

**"url": "http://pear2.php.net"**

**},**

**{**

**"type": "package",**

**"package": {**

**"name": "smarty/smarty",**

**"version": "3.1.7",**

**"dist": {**

**"url": "http://www.smarty.net/files/Smarty-3.1.7.zip",**

**"type": "zip"**

**},**

**"source": {**

**"url": "http://smarty-php.googlecode.com/svn/",**

**"type": "svn",**

**"reference": "tags/Smarty\_3\_1\_7/distribution/"**

**}**

**}**

**}**

**]**

**}**

19、config（root-only）：针对项目的一些配置。

process-timeout：默认 300 秒，Composer 进程执行超时时间；  
use-include-path：默认 false，如果是 true，Composer 自动加载器也会到 PHP 的 include\_path 中查找；  
preferred-install：默认 auto，设置 Composer 安装方式；  
github-protocols：默认 [“git”, “https”]，设置与 github 通信协议；  
github-oauth：设置 oauth；  
vendor-dir：默认 vendor，你可以换成别的；  
bin-dir：默认 vendor/bin，如果项目有二进制文件，会链接到这；  
cache-dir：默认 $home/cache，存放 Composer 运行时产生的缓存；  
cache-files-dir：默认 $cache-dir/files，存放包的 zip 文件；  
cache-repo-dir：默认 $cache-dir/repo，存放仓库元数据；  
cache-vcs-dir：默认 $cache-dir/vcs，存放 vcs 克隆；  
cache-files-ttl：默认六个月，缓存的过期时间；  
cache-files-maxsize：默认 300M；  
notify-no-install：默认 true，从仓库安装包会有个通知，可以关掉；  
discard-changes：默认false，如何处理脏的更新；

20、scripts（root-only）：Composer 允许你在安装进程中安装钩子脚本，钩子是基于事件的；

21、extra：供 scripts 消费的额外数据；这可以是几乎任何东西。若要从脚本事件访问处理程序，你可以这样做，$extra = $event->getComposer()->getPackage()->getExtra();

22、bin：指定哪些文件必须被当做二进制文件处理的；

23、archive：设置创建包时的选项，exclude 属性可以设置排除哪些目录

**{**

**"archive": {**

**"exclude": ["/foo/bar", "baz", "/\*.test", "!/foo/bar/baz"]**

**}**

**}**