composer快速入门

huizong

目　录

概述

>## composer对PHP的作用

现在的世界是要求快速的，如果每个项目都从头开始写，显然不现实，而且质量也得不到保证。PHP有大量开源的项目，如何想自己的项目中使用这些PHP项目，就是composer做的事。

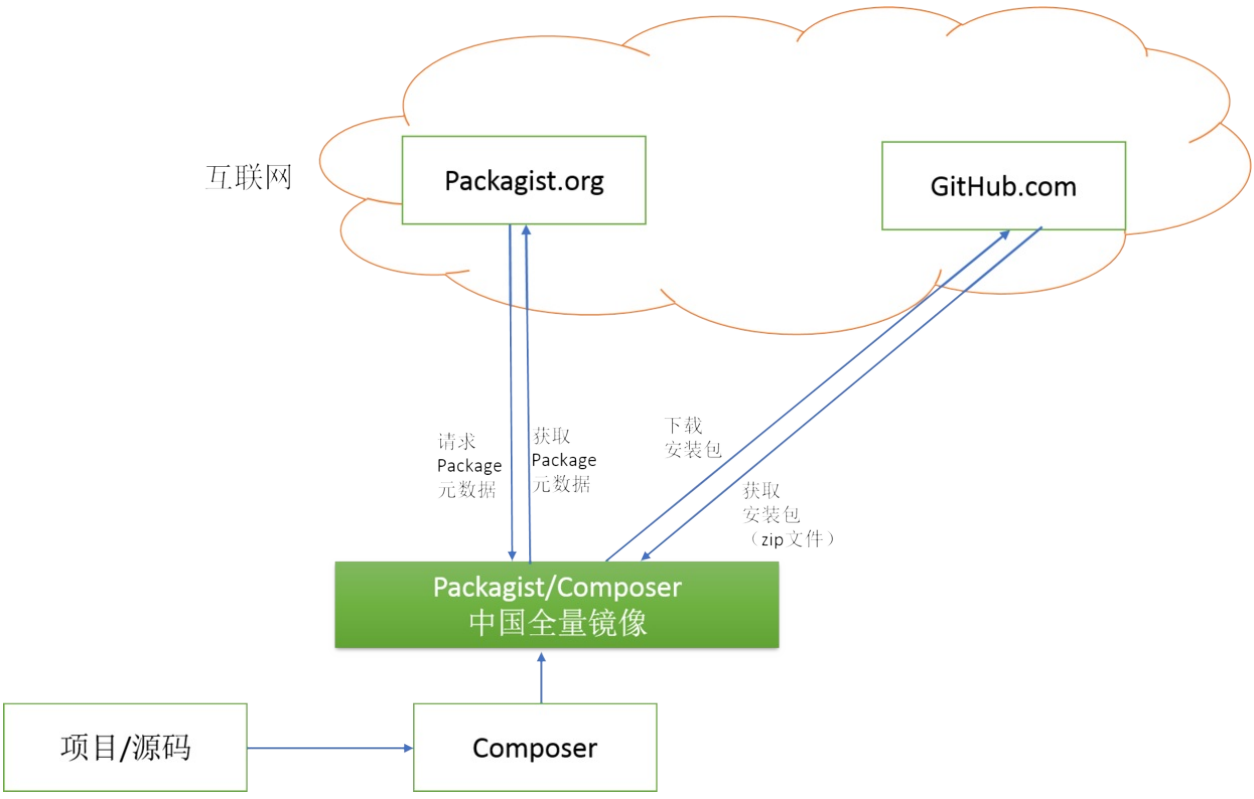
MVC中常用的ORM，如果你要造一个自己的MVC框架，不可能自己写，那去哪里找这些php的ORM库呢，怎么很好地在项目中引用呢？这就是composer做的是。

说白了，composer就是nodeJS的npm，一个php的包管理工具

>## composer与pear、pcel

你可能听说过pear和pcel，这且这两个名字还很像，容易混淆。pear和composer的功能是一样的，只是没有composer好，目前pear已经被淘汰。而pcel和两者不同，它是安装php扩展的，比如GD库、PDO扩展、CURL扩展，这些扩展是使用C编写的，是PHP底层扩展。

>## composer工作原理



这里经过几个步骤：

1.composer读取composer.json，这个json是在当前执行composer目录的

2.composer通过读取到的json数据去Packagist.org获取各个包的包名、作者、下载URL等信息。下载URL经常是GitHub上面的，因为目录的代码大部分都托管在GitHub上面嘛。
3.将从Packagist.org获取到的元数据存放到当前目录的composer.lock中
4.composer读取composer.lock中的元数据，根据元数据一次下载包，并且放到当前目录的vender目录里面

composer有不少常用的命令，比如composer install 、composer update、composer require。这三个命令都是会下载php类库的，composer update 会将步骤1、2、3、4都执行一遍，所以下载的类库是composer.json配置中匹配搭配的最新类库，而composer install，只是执行步骤4。
composer require 会将配置写入composer.json，然后执行步骤1、2、3、4

或许你会问，如何知道不同的包他们的配置是怎么写的，这个你就要去Packagist.org了,每个开源项目都会有安装和使用方法的。而且很简单哦：）

>## composer安装

composer安装脚本也是使用php写的，执行鞋面的命令下载compsoer的安装脚本，并使用php 执行它就可以安装composer的命令行工具composer.phar到php的bin目录下面。

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

或

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

原理：
curl -sS https://getcomposer.org/installer 将会输出 `https://getcomposer.org/installer` 这个文件的内容到界面上，通过 管道 `|` 传递给 `php` ，`https://getcomposer.org/installer` 就会被执行，这是一个php文件，作用是下载 `composer.phar`

`php -r "readfile('https://getcomposer.org/installer');" | php` 的原理也是一样的。

> 因为下载和执行php需要时间，所以请耐心等待几分钟

安装到命令行

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。

```
sudo mv composer.phar /usr/bin/composer
```

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。

下载七牛云SDK：
>composer require qiniu/php-sdk

- 中国化composer
  将composer的pakagist库地址修改为中国全量镜像地址：

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

>## composer卸载包
虽然前面讲解了原理，并且一步步做了安装示例，但是卸载包还是要讲一下。

正如前面说的，`composer update`会将步骤1、2、3、4都执行一遍，所以，只要我们将`composer.json`修改了，然后执行`composer update`就重新安装整个库了，自然那些不再`composer.json`的包也就不见了，以七牛云为例：

在composer.json中删除`"qiniu/php-sdk": "^7.1"`：
删除前：

```
"require": {
  "php": ">=5.4.0",
  "topthink/framework": "^5.0",
  "qiniu/php-sdk": "^7.1"
},
```

删除后：

```
"require": {
  "php": ">=5.4.0",
  "topthink/framework": "^5.0"
},
```

```
#重新安装整个库，OK
composer update
```

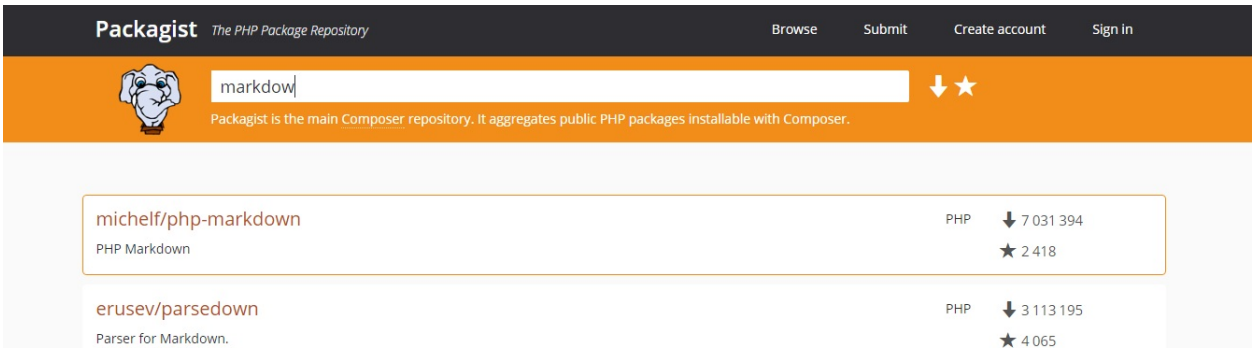
>注意，`"topthink/framework": "^5.0",`中的都好`，`一定要删除，否则不符合json格式，会报错。

composer安装开源项目到类库

先贴一个地址：Packagist.org

Packagist.org是PHP包管理平台，composer就是根据composer.json去上面拿到各种包的元数据，才可以下载的。

>##Packagist.org的真容



选择一个你需要的类库，进去后就有对应的文档，包括类库的使用环境、安装、使用文档等。

Usage

This library package is meant to be used with class autoloading. For autoloading to work, your project ne
Readme.php file for a minimal autoloader setup. (If you cannot use autoloading, see below.)

With class autoloading in place, putting the 'Michelf' folder in your include path should be enough for thi

```
use \Michelf\Markdown;
$my_html = Markdown::defaultTransform($my_text);
```

Markdown Extra syntax is also available the same way:

```
use \Michelf\MarkdownExtra;
$my_html = MarkdownExtra::defaultTransform($my_text);
```

If you wish to use PHP Markdown with another text filter function built to parse HTML, you should filter t
SmartyPants][psp]:

```
use \Michelf\Markdown, \Michelf\SmartyPants;
$my_html = Markdown::defaultTransform($my_text);
$my_html = SmartyPants::defaultTransform($my_html);
```

自动加载

1.自动加载

引入composer的自动加载文件，那么使用composer下载的项目就会被自动加载，不能更爽了。

2.使用composer自动加载来加载自己的php代码

项目开发的时候，免不了有一些自己写的类库、全局函数等，那如何将他们也添加到composer的自动加载里面呢？

composer一个支持四种方式自动加载自己的php代码，每种方式适合不同的场景。

2.1 PSR-4

php5.3+以上的，而且php类库必须班组PSR-4的规范

composer.json同级目录src下的所有命名空间为Monolog的PSR-4类库，即src下的Monolog\XXX.php和任意目录下的Vendor\Namespace\XXX.php类库：

```
{
    "autoload": {
        "psr-4": {
            "Monolog\\": "src/",
            "Vendor\\Namespace\\": ""
        }
    }
}
```

目录是多个的时候，可以采用数组：

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

src目录下的任何命令空间都取：

```
{
    "autoload": {
        "psr-4": { "": "src/" }
    }
}
```

2.2 PSR-0

对于使用PSR-0规范创建的php类库，可以使用PSR-0标准进行进行字段加载

```
{
    "autoload": {
        "psr-0": {
            "Monolog\\": "src/",
            "Vendor\\Namespace\\": "src/",
            "Vendor_Namespace_": "src/"
        }
    }
}
```

2.3 classmap

你可以用 classmap 生成支持支持自定义加载的不遵循 PSR-0/4 规范的类库。要配置它指向需要的目录，以便能够准确搜索到类文件。

```
{
    "autoload": {
        "classmap": ["src/", "lib/", "Something.php"]
    }
}
```

2.4 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"]
    }
}
```

关于PHP的这四种自动加载规范，请移步到：https://github.com/PizzaLiu/PHP-FIG