链滴

开发自己的 composer 包

作者：xiaoxiezaijia

原文链接：https://ld246.com/article/1568879631783

来源网站：链滴

许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

composer 是 PHP 用来管理依赖（dependency）关系的工具, 使用 composer 在业务中是非常常的，比如使用 阿里 oss 的 sdk ， 短信 sdk ， 非常好用的微信sdk EasyWeChat 都是使用composer 装， 一个业务的形成就如同搭建积木一样，通过composer 引入各种组件完成，但很多 phper 依然会开发自己的 composer 包(我也只是会用一下)，下面是记录自己开发 composer 的过程。

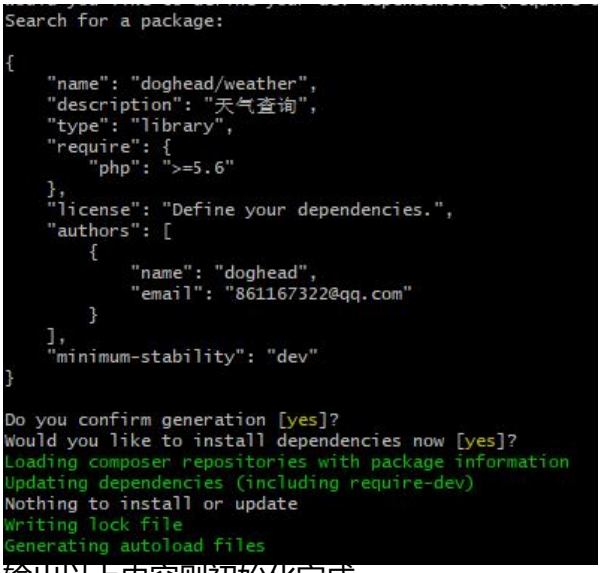
1. 创建一个开发目录

```
mkdir weather
cd weather
```

2. 利用composer生成一个composer.json

```
composer init
> Welcome to the Composer config generator
> This command will guide you through creating your composer.json config.

// 1. 输入项目命名空间
// 注意<vendor>/<name> 必须要符合 [a-z0-9_.-]+/[a-z0-9_.-]+
Package name (<vendor>/<name>) [dell/htdocs]: yourname/projectname

// 2. 项目描述
Description []: 这是一个测试

// 3. 输入作者信息，可以直接回车
Author [maopanfeng <861167322@qq.comm>, n to skip]:

// 4. 输入最低稳定版本，stable, RC, beta, alpha, dev
Minimum Stability []: dev

// 5. 输入项目类型，
Package Type (e.g. library, project, metapackage, composer-plugin) []: library

// 6. 输入授权类型
License []:
> Define your dependencies.

// 7. 输入依赖信息
Would you like to define your dependencies (require) interactively [yes]?

// 如果需要依赖，则输入要安装的依赖
Search for a package: php

// 输入版本号
Enter the version constraint to require (or leave blank to use the latest version): >=5.4.0

// 如需多个，则重复以上两个步骤

// 8. 是否需要require-dev，
Would you like to define your dev dependencies (require-dev) interactively [yes]?

// 操作同上
```

```
/*
{
"name": "mpf/test",
"description": "这是一个测试",
"type": "library",
"require": {
"php": ">=5.4.0"
},
"license": "MIT",
"authors": [
{
"name": "maopanfeng",
"email": "1052661052@qq.comm"
}
],
"minimum-stability": "dev"
}
*/

// 9. 是否生成composer.json
Do you confirm generation [yes]? yes
```



输出以上内容则初始化完成

3. 创建开发目录

```
mkdir src
mkdir tests
```

4. 声明自动加载

接下来我们需要在 composer.json 中声明包自动加载的命名空间

```
"autoload": {
      "psr-4": {
         "Doghead\\Weather\\": "./src/"
      }
```

原文链接：开发自己的 composer 包

```
    },
```

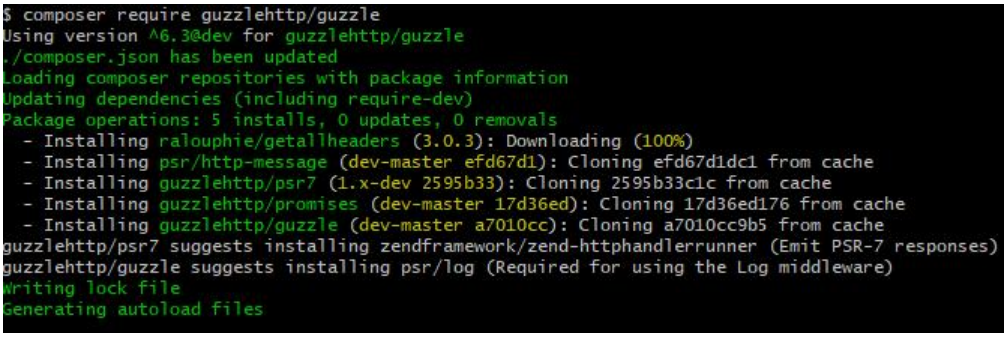
然后执行命令：composer dump-autoload 或者 composer du 「可选步骤」

5. 安装依赖

我们的项目需要请求接口，所以我们选择 guzzle/guzzle 来做为 http client， 其它暂时用不到，后用到的时候再安装即可：

```
$ cd weather/
$ composer require guzzlehttp/guzzle
```

输出



最终 composer.json 内容如下：

composer.json:

```
{
    "name": "doghead/weather",
    "description": "天气查询",
    "type": "library",
    "require": {
        "php": ">=5.6",
        "guzzlehttp/guzzle": "^6.3@dev"
    },
    "license": "Define your dependencies.",
    "authors": [
        {
            "name": "doghead",
            "email": "861167322@qq.com"
        }
    ],
    "autoload": {
        "psr-4": {
            "doghead\\Weather\\": "./src/"
        }
    },
    "minimum-stability": "dev"
}
```

基本结构搞定，，，终于开始写代码了！

代码编写

1. 从接口获取天气数据

创建文件

touch src/Weather.php

src/Weather.php

```
<?php

namespace Doghead\Weather;

class Weather
{

}
```

方法设计

根据之前设计的功能，结合 天气查询接口文档 的参数说明，我们添加几个方法：

注意：方法名通常是 **动名词** 形式，比如：getUsers，updateProfile，deleteOrder，revertAction。

构造函数 __construct($key)

参数说明：

- $key 为高德开放平台创建的应用 API Key；

我们调用天气 API 需要用到 API Key，所以把它设计在构造函数中。

```
<?php

namespace Doghead\Weather;

class Weather
{
    protected $key;

    public function __construct(string $key)
    {
        $this->key = $key;
    }
}
```

HTTP 客户端 getHttpClient()

获取天气需要用到 http 请求，所以需要先创建一个方法用于返回 guzzle 实例：

```
<?php
```

```php
namespace Doghead\Weather;

use GuzzleHttp\Client;

class Weather
{
    protected $key;
    protected $guzzleOptions = [];
    .
    .
    .
    public function getHttpClient()
    {
        return new Client($this->guzzleOptions);
    }

    public function setGuzzleOptions(array $options)
    {
        $this->guzzleOptions = $options;
    }
    .
    .
    .
```

其中我们设计了一个 $guzzleOptions 参数与方法 setGuzzleOptions，旨在用户可以自定义 guzzle 实例的参数，比如超时时间等。

获取天气 getWeather($city, $type = 'base', $format = 'json')

参数说明：

- $city - 城市名 / 高德地址位置 adcode，比如：“深圳” 或者（adcode：440300）；
- $type - 返回内容类型：base: 返回实况天气 / all: 返回预报天气；
- $format - 输出的数据格式，默认为 json 格式，当 output 设置为 “xml” 时，输出的为 XML 格式的数据。

获取天气的写法非常简单，获取 http client，组装参数，返回请求结果：

```php
<?php

namespace Doghead\Weather;

use GuzzleHttp\Client;

class Weather
{
    protected $key;
    protected $guzzleOptions = [];
    .
    .
    .
    public function getWeather($city, string $type = 'base', string $format = 'json')
    {
```

```
        $url = 'https://restapi.amap.com/v3/weather/weatherInfo';

        $query = array_filter([
            'key' => $this->key,
            'city' => $city,
            'output' => $format,
            'extensions' =>  $type,
        ]);

        $response = $this->getHttpClient()->get($url, [
            'query' => $query,
        ])->getBody()->getContents();

        return 'json' === $format ? \json_decode($response, true) : $response;
    }
```

那基本上这个类就已经完成了，最终的样子：

src/Weather.php

```
<?php
namespace Doghead\Weather;

use GuzzleHttp\Client;
use Doghead\Weather\Exceptions\HttpException;
use Doghead\Weather\Exceptions\InvalidArgumentException;

class Weather
{
    protected $key;
    protected $guzzleOptions = [];

    public function __construct(string $key)
    {
        $this->key = $key;
    }

    public function getHttpClient()
    {
        return new Client($this->guzzleOptions);
    }

    public function setGuzzleOptions(array $options)
    {
        $this->guzzleOptions = $options;
    }

    public function getWeather($city, string $type = 'base', string $format = 'json')
    {
        $url = 'https://restapi.amap.com/v3/weather/weatherInfo';

        $query = array_filter([
            'key' => $this->key,
            'city' => $city,
```

```
            'output' => $format,
            'extensions' =>  $type,
        ]);

        $response = $this->getHttpClient()->get($url, [
            'query' => $query,
        ])->getBody()->getContents();

        return 'json' === $format ? \json_decode($response, true) : $response;
    }
}
```

由于 getWeather 中 $type 参数支持多样性的参数格式，所以满足了我们前面需求分析时的三点要：

- 按地名查询实时天气；
- 获取最近的天气预报。

2. 异常处理

创建目录&文件

```
mkdir src/Exceptions
touch src/Exceptions/Exception.php
```

src/Exceptions/Exception.php

```
<?php

namespace Overtrue\Weather\Exceptions;

class Exception extends \Exception
{

}
```

自定义的异常需要继承 PHP 内置的异常类 Exception，且不需要包含任何方法，后面我会告诉你为么。

另外，当调用方传递的 $format 不是 xml 也不是 json 时需要抛出参数异常，所以还需要创建一个类：

创建文件

```
touch src/Exceptions/InvalidArgumentException.php
```

src/Exceptions/InvalidArgumentException.php

```
<?php

namespace Doghead\Weather\Exceptions;

class InvalidArgumentException extends Exception
{

}
```

当请求接口失败的时候，需要抛出异常类：

创建文件

touch src/Exceptions/HttpException.php

src/Exceptions/HttpException.php

```
<?php

namespace Doghead\Weather\Exceptions;

class HttpException extends Exception
{

}
```

接下来将天气获取方法加上异常处理：

src/Weather/Weather.php

```
<?php

namespace Doghead\Weather;

use GuzzleHttp\Client;
use Doghead\Weather\Exceptions\HttpException;
use Doghead\Weather\Exceptions\InvalidArgumentException;

class Weather
{
    .
    .
    .
    public function getWeather($city, string $type = 'base', string $format = 'json')
    {
        $url = 'https://restapi.amap.com/v3/weather/weatherInfo';

        if (!\in_array(\strtolower($format), ['xml', 'json'])) {
            throw new InvalidArgumentException('Invalid response format: '.$format);
        }

        if (!\in_array(\strtolower($type), ['base', 'all'])) {
            throw new InvalidArgumentException('Invalid type value(base/all): '.$type);
        }

        $query = array_filter([
            'key' => $this->key,
            'city' => $city,
            'output' => \strtolower($format),
            'extensions' =>  \strtolower($type),
        ]);

        try {
```

```
            $response = $this->getHttpClient()->get($url, [
                'query' => $query,
            ])->getBody()->getContents();

            return 'json' === $format ? \json_decode($response, true) : $response;
        } catch (\Exception $e) {
            throw new HttpException($e->getMessage(), $e->getCode(), $e);
        }
    }
```

当传递错误的 $format 或 $type 参数时将会抛出 \Overtrue\Weather\Exceptions\InvalidArgumen Exception 异常，当请求接口失败时将会抛出 \Overtrue\Weather\Exceptions\HttpException 异 ，注意不要忘记引入类名哦。

6. 测试扩展包

```
mkdir weather-test
cd weather-test
```

然后在这个测试项目根目录使用 composer 引入我们的包：

```
# 需要先初始化 composer.json, 一路回车即可
$ composer init

# 配置包路径，注意，这里 `../weather` 为相对路径，不要弄错了
$ composer config repositories.weather path ../weather

# 安装扩展包 这里 `dev-master` 中的 dev 指该分支下最新的提交，master 是指定的包中的分支名
$ composer require doghead/weather:dev-master
```

安装完成后，在 weather-test 根目录创建一个 index.php 来测试：

index.php

```
<?php

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

use Doghead\Weather\Weather;

// 高德开放平台应用 API Key
$key = 'bb5e3bd493d1f29f52f9d8ee4bf47049';
$w = new Weather($key);

echo "获取实时天气：\n";

$response = $w->getWeather('深圳');
echo json_encode($response, JSON_UNESCAPED_UNICODE | JSON_PRETTY_PRINT);

echo "\n\n获取天气预报：\n";

$response = $w->getWeather('深圳', 'all');
echo json_encode($response, JSON_UNESCAPED_UNICODE | JSON_PRETTY_PRINT);
```

```
echo "\n\n获取实时天气(XML)：\n";

echo $w->getWeather('深圳', 'base', 'XML');
```

然后测试一下：

```
$ php index.php
```

结果如下：

```
获取实时天气:
{
    "status": "1",
    "count": "1",
    "info": "OK",
    "infocode": "10000",
    "lives": [
        {
            "province": "广东",
            "city": "深圳市",
            "adcode": "440300",
            "weather": "中雨",
            "temperature": "27",
            "winddirection": "南",
            "windpower": "6",
            "humidity": "94",
            "reporttime": "2018-08-21 16:00:00"
        }
    ]
}

获取天气预报:
{
    "status": "1",
    "count": "1",
    "info": "OK",
    "infocode": "10000",
    "forecasts": [
        {
            "city": "深圳市",
            "adcode": "440300",
            "province": "广东",
            "reporttime": "2018-08-21 11:00:00",
            "casts": [
                {
                    "date": "2018-08-21",
                    "week": "2",
                    "dayweather": "雷阵雨",
                    "nightweather": "雷阵雨",
                    "daytemp": "31",
                    "nighttemp": "26",
                    "daywind": "无风向",
                    "nightwind": "无风向",
```

```
                "daypower": "≤3",
                "nightpower": "≤3"
            },
            {
                "date": "2018-08-22",
                "week": "3",
                "dayweather": "雷阵雨",
                "nightweather": "雷阵雨",
                "daytemp": "32",
                "nighttemp": "27",
                "daywind": "无风向",
                "nightwind": "无风向",
                "daypower": "≤3",
                "nightpower": "≤3"
            },
            {
                "date": "2018-08-23",
                "week": "4",
                "dayweather": "雷阵雨",
                "nightweather": "雷阵雨",
                "daytemp": "32",
                "nighttemp": "26",
                "daywind": "无风向",
                "nightwind": "无风向",
                "daypower": "≤3",
                "nightpower": "≤3"
            },
            {
                "date": "2018-08-24",
                "week": "5",
                "dayweather": "雷阵雨",
                "nightweather": "雷阵雨",
                "daytemp": "31",
                "nighttemp": "26",
                "daywind": "无风向",
                "nightwind": "无风向",
                "daypower": "≤3",
                "nightpower": "≤3"
            }
        ]
    }
  ]
}
```


获取实时天气(XML):


```
<?xml version="1.0" encoding="UTF-8"?>
<response>
    <status>1</status>
    <count>1</count>
    <info>OK</info>
    <infocode>10000</infocode>
    <lives type="list">
        <live>
            <province>广东</province>
```

```
        <city>深圳市</city>
        <adcode>440300</adcode>
        <weather>中雨</weather>
        <temperature>27</temperature>
        <winddirection>南</winddirection>
        <windpower>6</windpower>
        <humidity>94</humidity>
        <reporttime>2018-08-21 16:00:00</reporttime>
      </live>
    </lives>
</response>
```

就可以正常拿到返回的内容了。

原理说明

你可以打开 composer.json 看一下，你会发现上面我们执行的：

```
$ composer config repositories.weather path ../weather
```

它在 composer.json 中添加了如下部分：

composer.json

```
    .
    .
    .
     "repositories": {
       "weather": {
          "type": "path",
          "url": "../weather"
       }
     }
    .
    .
    .
```

这样我们在安装的时候 composer 会创建一个软链接 vendor/overtrue/weather 到包所在目录 ../weather，这样一来，你可以直接在测试项目的 vendor/overtrue/weather 下修改文件，包里的文件也跟着变了，是不是对于开发过程中来讲非常的方便?