PHP 10권 – 페키지 제작

**10. 패키지 제작**

====================

지금까지 컴포저를 통하여 다른 개발자들이 만들어 놓은 PHP 페키지를 설치하여 사용하는 방법에 대해서 알아보았습니다.

라라벨과 같이 컴포저와 잘 결합된 프레임워크는 수많은 개발자들이 다양한 응용페키지를 만들어 배포를 하고, 생태계가 발전하고 있습니다. 한번 나도 좋은 패키지를 만들어 이 생태계에 기여를 해보고 싶어하는 경우가 많이 있을 것입니다.

이번장에서는 자신이 직접 코드를 작성하고 페키지를 만들어 배포하는 방법에 대해서 학습하도록 하겠습니다.

**10.1 준비작업**

페키지의 소스를 작성하기 전에 먼저 배포를 위한 임시 디렉토리를 하나 만들도록 합니다. 필자는 현재 프로젝트에서 /packages 라는 폴더를 하나 만들도록 하겠습니다.

mkdir packages

cd packages

폴더를 만들고 해당 폴더로 이동을 합니다.

생성한 패키지 폴더에 vender 이름으로 폴더를 하나더 생성해 주도록 합니다. Vendor는 페키지들이 서로 충돌이 되지 않도록 구분하는 용도로 사용됩니다. 이미 다른 사람이 동일한 이름의 벤더명을 사용하고 있다고 한다면, 다른 이름으로 만들어야 합니다.

필자의 vendor 이름은 jiny 입니다.

mkdir jiny

cd jiny

폴더를 만들고 해당 폴더로 이동을 합니다.

이제 배포하고자 하는 페키지명에 대한 폴더를 생성을 합니다.

mkdir hello

cd hello

폴더를 만들고 해당 폴더로 이동을 합니다.

**10.2 json 파일생성**

페키지 제작을 위해서 작업폴더를 만들었습니다. 이곳에 실제적인 패키지들을 만들어 보도록 합니다.

현재의 디렉토리는 /packages/jiny/hello 입니다. 컴포저로 배포가 되는 페키지를 만들기 위해서는 먼저 배포용 composer.json 생성을 해야 합니다.

**10.2.1 초기화**

composer.json 파일을 직접 에디터를 이용하여 만들어 사용해도 됩니다. 하지만처음에컴포저의 json양식을 잘 모르는 사람에게는 아무것도 없이 에디터로 json파일을 만들기는 쉽지 않습니다.

이를 위해서 컴포저는 json 파일을 쉽게 만들수 있도록 init 옵션을 제공합니다.

|  |
| --- |
| **composer init** |

명령은 composer.json 파일을 쉽게 만들어 주는 역할을 합니다.

|  |
| --- |
| D:\htdocs\composer\packages\jiny\hello>**composer init**  Welcome to the Composer config generator  This command will guide you through creating your composer.json config. |

**10.2.2 벤더명/페키지**

먼저 패키지에 대한 벤더명과 이름을 물어봅니다.

|  |
| --- |
| Package name (<vendor>/<name>) [infoh/hello]: **jiny/hello** |

우리가 사전에 정한 이름을 적어 줍니다. 작업 폴더에서 생성한 폴더명과 동일하게 벤더명과 페키지명을 “/”로 구분하여 입력하여 줍니다.

**10.2.3 페키지 설명**

페키지에 대한 설명부분을 입력합니다. 제작하고 있는 페키지의 대하여 간략한 설명을 입력을 합니다.

|  |
| --- |
| Description []: composer package make test |

설명 부분은 별도로 composer.json 파일을 에디터로 수정하여 편집을 할 수도 있습니다.

**10.2.4 작성자 정보**

페키지의 작성자 정보를 입력합니다. 개발자의 이름, 연락처 이메일 주소를 입력을 합니다.

|  |
| --- |
| Author [hojinlee <infohojin@gmail.com>, n to skip]: |

**10.2.5 패키지 상태**

패키지의 현재의 상태에 대해서 표기를 합니다.

|  |
| --- |
| Minimum Stability []: **stable** |

**10.2.6 패키지 타입**

패키지의 타입유형을 작성합니다.

|  |
| --- |
| Package Type (e.g. library, project, metapackage, composer-plugin) []: library |

**10.2.7 라이센스**

페키지 소스에 대한 라이선스 타입을 지정하도록 합니다. 라이선스는 소프트웨어의 사용 권한을 의미합니다.

|  |
| --- |
| License []: **MIT** |

개인적으로 직접 라이선스를 기술할 수도 있지만, 오픈소스에는 여러 유형의 인기 라이선스를 따르기도 합니다.

**10.2.8 의존성 정의**

페키지의 의존성을 정의 합니다. 이부분은 별도로 에디터를 통하여 추가하거나 수정하도록 합니다.

|  |
| --- |
| Define your dependencies.  Would you like to define your dependencies (require) interactively [yes]? no  Would you like to define your dev dependencies (require-dev) interactively [yes]? no |

**10.2.9 완료**

앞에서 입력한 정보를 이용하여 간단하게 composer.json 파일을 생성한 결과 하면을 보여줍니다.

|  |
| --- |
| {  "name": "jiny/hello",  "description": "composer package make test",  "type": "library",  "license": "MIT",  "authors": [  {  "name": "hojinlee",  "email": "infohojin@gmail.com"  }  ],  "minimum-stability": "stable",  "require": {}  }  Do you confirm generation [yes]? **yes** |

Yes를 선택하시면 생성한 json정보를 composer.json 파일로 저장을 하게 됩니다. 이제 페키지를 배포하기 위해서 기본적인 정보를 만드는데 성공을 하였습니다.

**10.3 코드작성**

이제 배포를 위한 예제소스를 작성해 보도록 하겠습니다.

컴포저를 이용하여 패키지를 제작 배포하게 되면 vendor/밴더명/패키지명 형태의 디렉토리 구조를 생성하게 됩니다. 우리가 순수하게 PHP코드만 작성을 한다고 할 때 바로 페키지명 디렉토리에 코드를 작성을 해도 됩니다.

하지만, 페키지를 배포하고, 이를 사용하기 위해서 테스트 코드나, 예제코드, 문서등을 같이 배포하고자 하는 경우에는 서브 폴더들을 더 만들어 사용하는 것도 좋습니다.

밴더명/패키지명 안에 다음과 같은 폴더를 추가로 더 만들어 보도록 합니다.

* src
* doc
* example
* test

그리고 소스코드를 담을 src폴더로 이동하여 이곳에 코드를 작성해 보도록 합니다.

**10.3.1 예제코드 추가**

페키지명을 통해서 알 수 있듯이 간단하게 hello 인사말을 출력하는 페키지를 만들어 보도록 합니다.

HelloMessage.php

<?php

namespace Jiny\Hello;

class HelloMessage

{

/\*\*

\* @return void

\*/

public function \_\_construct()

{

echo \_\_CLASS\_\_." 객체가 생성이 되었습니다.<br>";

}

public function sayHello($name='')

{

echo "Hi! ". $name. "nice meet you.<br>";

}

}

컴포저를 통하여 페키지 코드를 작성을 할때는 네임스페이스 같이 지정을 해주어야 합니다. 코드 시작시 namespace 키워드와 **“밴더명\페키지”** 형태로 지정을 해주면 됩니다.

**10.4 오토로드 등록**

테스트를 위해서 위의 작성한 패키지를 임시로 오도로드 PSR4 등록을 해보도록 하겠습니다.

메인 root 디렉토리에 있는 composer.json 파일을 에디터를 통해서 편집해 보도록 합니다. 설정 내용중에는 “autoload”필드 항목과 서브 “psr-4”항목을 찾아 볼 수 있습니다.

"autoload": {

"psr-4": {

"App\\": "app",

"Jiny\\Hello\\": "packages/jiny/hello/src"

},

"files": [

"app/function.php"

],

"classmap": [

"app/classes"

]

}

이곳에 지금 작성한 패키지의 코드를 등록해 봅니다.

수정한 컴포저의 오토로드를 갱신하기 위해서 dumpautoload 옵션을 실행하도록 합니다. 루트 디렉토리에서 명령을 입력합니다.

|  |
| --- |
| D:\htdocs\composer\packages\jiny\hello>cd ..  D:\htdocs\composer\packages\jiny>cd ..  D:\htdocs\composer\packages>cd ..  D:\htdocs\composer>**composer dumpautoload**  Generating autoload files |

오토로드 파일이 재생성 되는 것을 확인할 수 있습니다.

**10.5 테스트**

이제 실제 코드에서 앞에서 작성한 페키지를 삽입하여 코드를 작성해 보도록 하겠습니다.

<?php

echo "Composer Study<br>";

require "vendor/autoload.php";

$pack = new \Jiny\Hello\HelloMessage;

$pack->sayHello();

결과

Composer Study  
Jiny\Hello\HelloMessage 객체가 생성이 되었습니다.  
Hi! nice meet you.

먼저 컴포저의 autoload 파일을 PHP코드에 삽입을 합니다. 그러면, 오토로드는 페키지의 클래스를 사용하고자 할 때 자동으로 해당 클래스 파일을 인크루드 하여 처리해 합니다.

이후 객체를 생성합니다. 페키지의 객체를 생성할때는 밴더명과 페키지명이 결합된 형태의 네임스페이스 경로를 입력해 주어야 합니다.

**10.6 추가정보 설정**

페키지가 정상적으로 동작하는 것을 확인후에 배포를 위한 추가 정보를 패키지의composer.json 파일에 저장을 합니다.