Chapter

20

오토로드

PHP는 소스 파일을 나누어 작성할 수 있습니다. 분리된 파일은 include/require와 같은 전처리 명령을 통하여 소스를 삽입, 결합할 수 있습니다.

소스의 파일을 분리하여 작성하는 방법은 소스 코드들을 간결화하고, 기능별로 코드들을 분리할 수 있기 때문입니다. 비슷하게 분리된 파일들은 유지보수 및 재사용을 하는 데 있어서 매우 유용합니다.

분리된 파일들은 동작하는 데 있어서 상호 의존관계가 발생됩니다. 예로, aaa.php 파일을 실행하는 데 필요한 함수가 bbb.php에 있다면 2개 파일은 서로 의존하게 됩니다. 만일 aaa.php 파일이 의존관계에 있는 bbb.php 파일을 include하지 않으면 의존성 실패로 스크립트 파일은 오류를 발생합니다.

파일을 분리하는 것은 의존성만 잘 관리한다면 매우 유용한 소스 관리 방법입니다. 하지만 너무 많은 파일로 소스가 분리되어 있을 경우 모든 의존성을 체크하여 처리하는 것은 힘이 듭니다.

```
<?php
        include ('파일...');
        include ('파일...');
        include ('파일...');
        ....
        ....
        include ('파일...');
        include ('파일...');
```

소스 상단에 많은 include문을 포함할 것입니다. 이러한 복잡한 의존관계를 쉽게 해결하기 위해서 PHP는 오토로드라는 기능을 적용했습니다. 오토로드는 PHP에서 의존성이 있는 소스를 실행할 때 먼저 실행되어 소스 결합을 처리하게 됩니다.

20.1 클래스 의존성

클래스는 인스턴스를 생성하거나 정적으로 클래스를 사용하기 전에 반드시 클래스가 정의되어 있어야 합니다. 또한 클래스를 상속받을 때도 상속되는 부모의 클래스는 반드시 사용 전에 정의되어 있어야 합니다.

이처럼 PHP가 클래스를 사용하기 전에 연관된 클래스의 관계를 클래스 의존성이라고 합니다. 프로그램이 크고 클래스의 관계가 복잡할수록 클래스의 의존성을 확인하는 것은 매우 중요합니다.

Bascic.class.php 파일에,

```
<?php
        class basic {
                function show ( ){
                        return "클래스입니다.";
                }
        }
?>
```

클래스를 생성한 후에,

Test.php

```
<?php
        // 클래스 파일을 삽입합니다.
        include “basic.class.php”;

        $basic = new basic;

        echo $basic->show();
?>
```

파일에서 사전에 클래스 파일을 삽입합니다.

20.2 클래스 의존성 체크 함수

기존에는 클래스를 사용하면 모든 클래스 파일을 미리 사전에 inlcude 또는 require하여 파일을 삽입했습니다. 하지만 작성한 모든 클래스들이 하나에 파일에 전부 사용되지는 않습니다.

PHP는 클래스의 인스턴스를 선언하거나 정적 호출 시 클래스의 의존성을 사전에 검사할 수 있는 함수를 제공합니다. 클래스 의존성을 체크할 수 있는 함수를 통하여 필요한 클래스 파일을 실시간으로 inlcude 또는 require함으로써 메모리를 관리하고, 복잡한 의존성 관계를 쉽게 처리할 수 있습니다.

이렇게 클래스 파일을 자동으로 확인해서 처리하는 기능을 오토로드 기능이라고 합니다.

20.3 클래스 파일 삽입

PHP 5로 버전업이 되면서 php가 클래스 또는 인터페이스를 호출할 때 자동으로 실행되는 함수가 있습니다.

```
<?php
        function __autoload($className){
                include $className.".class.php";
        }
        $basic = new Basic;
?>
```

__autoload() 함수는 위의 예처럼 new basic 형태로 클래스를 선언할 때, __autoload 함수가 실행되고 클래스 이름이 __autoload의 $className으로 인자를 전달하게 됩니다. __autoload 함수는 전달된 클래스명 인자를 이용하여 클래스 파일을 include하여 정상적으로 클래스가 선언되도록 합니다.

프로젝트가 커질수록 클래스를 관리하고 무결성을 유지하면서 클래스 파일을 include하기란 쉽지 않습니다. 이런 자동 로딩 기능을 이용하면 클래스 기반으로 코딩하는 데 매우 편리할 것입니다.

PHP 5.1.2로 업그레이드되면서 기존 __autoload는 spl_autoload_registrer() 함수를 이용하여 보다 더 유연하게 오토로드 처리를 할 수 있게 되었습니다.

최근에는 spl_autoload_register() 함수를 더 많이 쓰는 것 같습니다. spl_autoload_register() 함수는 기존 __autoload 함수 사용법이 같습니다.

```
function spl_autoload_register($className){
        include $className.".class.php";
}
```

위와 같은 형태로 동일하게 사용하면 됩니다.

오토로드 기능을 매번 파일 상단에 선언해서 사용하기란 불편합니다. 또한 오토로드 코드들이 각각의 파일마다 중복될 것입니다. 이런 경우 autoload.php 형태로 별도의 오토로드 처리 파일을 만들어서 사용하면 편리합니다.

autoload.php 파일 생성

```
<?
    // 오토로드 처리를 위한 파일

    // 클래스 파일을 구분하기 위한 사용자 정의 확장자를 사용하면,
    // php 실행 파일과 클래스 파일을 좀 더 쉽게 구분할 수 있습니다.
    $classExt = ".class.php";

    // 클래스를 호출하면 클래스명이 인자로 전달됩니다.
    function spl_autoload_register($className){
        $classFilePath = __DIR__ . "/" . $className . $classExt;

        // 클래스 파일이 존재하는지 검사를 합니다.
        // 클래스 파일이 없는 상태에서 require 등을 통하여 삽입을 한다고 하면,
        // 에러를 발생할 수 있습니다.
        if(is_readable($classFilePath)){

            // 클래스 파일을 불러옵니다.
            require $classFilePath;

        }

    }

?>
```

이렇게 미리 만들어 놓은 클래스 오토로드 처리 파일을 프로그램 작성 시 먼저 한 번 불러오고, 클래스를 선언하여 사용하면 자동으로 php가 선언되지 않은 클래스 파일을 찾아 불러오고 사용할 수 있도록 처리합니다.

```php
<?php
    // 오토로드 처리를 위한 파일을 불러옵니다.
    require autoload.php;

    // 오토로드 파일을 통하여 jiny 클래스가 정의된 클래스 파일을 불러와
    // 클래스를 오류 없이 선언하여 사용할 수 있습니다.
    $jiny = new jiny;
?>
```

20.4 PSR-4 Autoloading

PHP Framework Interop Group(PHP-FIG)에서는 SPR-0 autoloading standard를 제안했습니다. 다음과 같은 규칙을 적용하여 오토로드를 처리하도록 권장하고 있습니다.

- 네임스페이스와 클래스명의 구조는 \<vender Name>\(<namespace>\)*<Clas sname> 형식을 따릅니다.
- 네임스페이스는 서브 네임스페이스를 포함할 수 있습니다.
- 네임스페이스 구분자는 파일을 불러오기 위한 디렉터리 구분자입니다.
- 클래스명에 포함된 _ 글자는 디렉터리 구분자로 사용된다.
- 네임스페이스와 클래스 파일을 불러올 때 .php를 확장자로 불러옵니다.
- 벤더, 네임스페이스, 클래스는 대소문자를 구분한다.

```php
<?php
    function autoload($className) {

        $className = ltrim($className, '\\');
        $fileName = '';
        $namespace = '';

        if ($lastNsPos = strpos($className, '\\')) {
```

```
            $namespace = substr($className, 0, $lastNsPos);
            $className = substr($className, $lastNsPos + 1);
            $fileName = str_replace('\\', DIRECTORY_SEPARATOR, $namespace)
            . DIRECTORY_SEPARATOR;
        }

        $fileName .= str_replace('_', DIRECTORY_SEPARATOR, $className) .
        '.php';
        require $fileName;
    }

?>
```

20.5 컴포저

PHP의 컴포저(composer)는 오토로드와 의존성을 관리할 수 있는 도구입니다. npm이나 apt, pip 같은 것과 유사합니다. 컴포저는 PSR-4 AutoLoader 제안과 PSR를 준수하여 패키지 의존성을 관리합니다.

컴포저를 PHP에서 사용하기 위해서는 PHP 5.3.2 이상의 버전이 필요로 합니다. 또한 컴포저는 멀티 플랫폼을 지원함으로 윈도우, 리눅스, 맥OS 등에서도 사용이 가능합니다.

20.5.1 컴포저 설치

컴포저는 https://getcomposer.org에 접속하면 설치 파일을 다운로드할 수 있습니다. 컴포저는 무료로 다운로드해서 설치할 수 있습니다.

20.5.2 구성 설정

컴포저를 통하여 패키지들을 설치하면 자동적으로 의존성과 관련된 설정 파일들이 생성됩니다. 의존성 설정 파일은 json 타입으로 작성되어 있습니다.

composer.json 파일은 컴포저를 이용하여 의존성 및 설치된 패키지를 관리할 수 있는 설정 파일입니다. 컴포저를 통하여 패키지의 의존성이 설치되면 기존 conposer.json 파일과 별개로 composer.lock 파일이 추가로 생성됩니다. composer.lock 파일은 컴포넌트의 목록, 버전 등 세부적인 정보들을 담고 있습니다.

또한 composer.lock 파일은 컴포저가 패키지들의 최신 버전과 별개로 composer.lock에 명시된 버전으로 다운로드 및 사용하게 됩니다. 이는 컴포저가 패키지를 최신 버전으로 업데이트하여 패키지의 버전 차이로 인해 발생할 수 있는 오류를 줄일 수 있습니다.

20.5.3 컴포넌트

기존의 PHP는 다양한 해결 문제점들을 개발자들이 중복으로 개발하느라 많은 시간을 보냈습니다. 그리고 이렇게 개발, 공개된 소스들은 자신의 홈페이지나 블로그에 공개하는 방식으로 다른 개발자들과 공유했습니다.

개발자들의 서로 다른 공개 방식 및 사이트들을 검색하고 찾아서 자신의 프로젝트에 설치하는 것 또한 개발자로서는 힘든 과정 중 하나였습니다.

PHP는 이러한 분산되어 있는 PHP 소스들을 컴포저 의존성 설치 툴과 더불어 패키지스트라는 통합 관리 사이트를 통하여 체계적으로 공유된 소스들을 관리하고 설치할 수 있는 서비스가 출현했습니다.

PHP 개발자들은 공동의 인터페이스와 코드 스타일로 소스를 코딩하고, 패키지스트 사이트를 통하여 배포, 컴포저를 이용한 설치 과정을 한 번에 할 수 있도록 체계화되고 있습니다. 이러한 공통된 규약과 작업 등으로 지금까지 PHP는 다양한 컴포넌트를 찾아보고, 빠른 시간 안에 문제점을 해결할 수 있게 되었습니다.

컴포넌트들은 작은 단위로 정확한 문제점 해결과 작은 코드로 되어 있기 때문에 빠르고 안정적으로 테스트하면서 적용할 수 있습니다. 또한 프레임워크를 적용하여 사용하는 데 확장성과 향후 유지보수를 편리하게 도와줍니다.

20.5.4 패키지스트

패키지스트(https://packagist.org) 사이트는 PHP 컨포넌트를 등록 관리하는 사이트입니다. 패키지스트에는 다양한 PHP 패키지들이 등록되어 있고 검색하여 필요한 패키지를 컴포저를 통하여 설치, 사용할 수 있습니다.

컴포넌트는 대부분 배포하는 사람의 벤더명과 패키지명으로 구성합니다.

벤더명/패키지명

이러한 구조들은 서로 패키지들이 중첩되지 않고 각각의 관리를 하기 위함입니다. 동일한 패키지스트 사이트는 수많은 PHP 개발자들이 만들어 놓은 패키지 컴포넌트를 검색할 수 있는 것과 더불어 자기 자신도 새로운 패키지 컴포넌트를 생성하여 다른 개발자들과 공유할 수 있습니다.

20.5.5 패키지 다운로드

패키지 및 의존성 관리 도구인 컴포저를 이용하면 패키지스트에 등록된 다양한 PHP 컴포넌트를 다운로드 및 설치할 수 있습니다.

패키지스트 사이트에서 필요한 패키지를 검색해 봅니다. 검색 창에서 키워드를 입력하면 관련된 다양한 패키지들을 찾을 수 있습니다.

패키지를 검색하면 간략한 패키지 설명과 함께 다운로드 수와 별점이 표기됩니다. 다운로드 수와 별점은 해당 패키지 컴포넌트가 얼마나 인기가 있고, 많은 사람들이 사용하는지 가늠해 볼 수 있습니다.

하지만 이러한 평점들은 단순한 수치고, 많은 테스트와 자신에게 맞는 패키지를 설치하여 사용하면 됩니다.

필요한 패키지를 찾았으면 콘솔 창에서 composer 명령을 통하여 패키지를 설치할 수 있습니다.

```
composer require jiny/routes
```

패키지를 다운로드하면 자동으로 최상의 디렉터리에 /vender 폴더가 하나 생성됩니다. /vender 폴더는 다운로드한 컴포넌트 패키지들을 저장, 관리하는 폴더입니다.

/vander 폴더 아래에 각각의 컴포넌트의 벤더명으로 폴더를 생성하고, 패키지명으로 폴더를 생성합니다.

/vender / 패키지벤더명 / 패키지명

또한 composer.json과 composer.lock 파일을 생성하고 관리해 줍니다.

20.5.6 사설 저장소

모든 PHP의 컴포넌트를 위의 패키지리스트와 같은 서비스 사이트를 이용해야만 하는 것은 아닙니다. 자체적인 컴포넌트 저장소를 만들어서 개인적으로 제작한 사설 컴포넌트를 관리하고 컴포저를 통하여 설치할 수 있습니다.

개인적인 내부 서버를 통하여 컴포넌트를 관리하기 위해서는 컴포저에게 내부 서버명과 접속 정보를 함께 제공해야 합니다. 다음과 같은 형태로 접속 정보를 auth.json 파일을 생성하여 관리할 수 있습니다. auth.json 파일은 composer.json과 같은 파일 위치에 있습니다.

```
{
    "패키지명":{
        "서버도메인": {
            "username":"접속아이디",
            "password":"접속패스워드"
        }
    }
}
```

또는 콘솔상에서 composer 명령을 입력할 때 직접 입력하거나 --global 명령을 통하여 글로벌 정보로 저장할 수 있습니다.

composer config 패키지명.도메인 아이디 패스워드

또는

composer config —global 패키지명.도메인 아이디 패스워드

--global로 접속 정보를 저장하면 ~/.composer/auth.json 파일에 저장됩니다.

20.5.7 패키지 사용

컴포저를 통하여 컴포넌트 패키지를 다운로드하면 /vender 폴더에 자동적으로 저장됩니다. 컴포저는 모든 패키지들의 의존성과 호환이 되는 PSR-4 기준의 오토로더를 생성, 제공합니다.

자동적으로 제공되는 오토로더를 프로그램 상단에 require 명령을 통하여 삽입하여 패키지들을 사용하면 됩니다.

```
<php
        require “vender/autoload.php”;
```

설치된 패키지를 통하여 클래스의 인스턴스를 생성할 때는 패키지의 벤더명과 네임스페이스를 함께 사용해야 합니다.

만일 네임스페이스명이 길면 use 명령을 통하여 별칭 또는 현재의 네임스페이스 경로를 변경할 수도 있습니다.

```
$obj = new \jiny\routes( );
```