PHP 10권 – 컴포저 소개

**01. 컴포저 소개**

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

컴포저는 PHP 페키지들의 의존성을 관리하는 도구 입니다. 컴포저의 등장으로 PHP 생태계가 획기적으로 발전하게 되는 단계가 되었습니다.

또한 컴포저는 요즘같이 글로벌화된 생태계 환경에서 프로그래밍을 하는데 필요한 중요한 도구입니다.

그럼 컴포저가 도입되기 전과, 도입후의 장단점을 알아보도록 하겠습니다.

**01.1 라이브러리**

아무리 뛰어난 개발자라 하더라도 모든 소스코드를 혼자만의 힘으로 개발을 할 수는 없습니다. 이전에 많은 개발자들이 기여한 소스코드를 기반으로 진화하듯이 발전을 하였습니다.

외부에서 제공된 소스코드를 자신의 프로젝트에 삽입하여 코드를 이용할 수 있는 생태계 있습니다.

라이브러리를 사용한다는 것은 남의 소스코드를 자신의 프로그램에 통합한다는 의미입니다.

**01.1.1 유용성**

우리는 매번 수많은 유사한 프로젝트들을 진행합니다. 또한 대형 프로젝트들은 코드의 양이 방대하기 때문에 관리하는데 어려움이 있습니다. 그중 일부 코드들은 다른 프로젝트에도 사용이 되기도 합니다.

이처럼 큰 코드를 좀더 쉽게 관리하고, 다른 프로젝트에서도 코드를 사용할 수 있도록 코드들을 분리하여 라이브러리화 하여 관리하면 여러모로 이점이 있습니다.

또한 인기있는 외부 라이브러리의 경우 수많은 개발자들로부터 검증받은 제품으로 좀더 신뢰를 하고 사용을 할 수 있습니다.

**01.1.2 라이브러리의 한계**

이전에는 라이브러리를 직접 설치하는 방법이 주 였습니다. 여러곳에 분포되어 있는 라이브러리 코드를 다운로드 받아 압축을 풀고, 서버에 올려서 코드를 맞추기도 합니다.

또한 운영되는 서버가 여러대일 경우 각각의 서버에 복잡한 라이브러리 설치과정을 반복하여 작업을 해야 합니다.

이처럼 라이브러리를 설치하는 과정은 귀찬고, 실수가 많이 발생될 수 있는 일련의 과정들 입니다. 하지만 앞으로 소개할 컴포저를 이용하면 이러한 과정들을 손쉽게 처리할 수 있습니다.

**01.2 의존성**

라이브러리는 자체의 고유한 것만 사용하지 않습니다. 라이브러리가 공유되고, 다른 라이브러리를 참조할 경우에는 단순히 복사형태의 라이브러리를 설치할 수 없습니다.

예를들어 라이브러리는 다른 유형의 라이브러리를 사용하는 경우가 있습니다. 예로 A라이브러리는 B를 사용하고, B는 또다른 D와 E라이브러리를 사용할 수도 있습니다.

하나의 라이브러리는 다른 라이브러리를 읽어와서 사용하기도 합니다. 이런경우 라이브러리들은 상호 의존성을 가지게 됩니다.

이렇게 라이브러리들 끼리 서로 사용하는 관계를 의존성이라고 합니다.

**01.2.1 의존성 지옥**

이렇게 라이브러리들관에 의존성을 유지하면서 설치와 배포를 하는 과정은 그다지 쉽지 않습니다. 얼마나 이러한 과정들이 힘들었으면 “의존성 지옥(dependency hell)”이라는 표현까지 써가면서 예기를 하고 있습니다.

시스템이 커지고, 다양화 될수록 더 많은 라이브러리를 참조하게 됩니다. 그리고 각각의 라이브러리는 의존성이 더 심해 집니다.

윈도우에서는 “DLL지옥”, 자바에서는 “jar 지옥”이라는 말도 흔하게 사용을 하곤 하였습니다. 의존성의 문제는 항상 개발을 하는데 있어서 고려해야 되는 골치아픈 문제 입니다.

**01.2.2 문제의 일반화**

운영체제, 프레임워크, 대규모 프로젝트등 라이브러리 설계기법을 응용한 작업들은 이러한 의존성 문제하 항상 봉착되어 있습니다.

**01.2.3 업데이트의 문제**

라이브러리 또한 완성된 소프트웨어가 아닙니다. 각각의 라이브러리는 계속해서 코드를 개선해 나가게 됩니다. 그리고 이렇게 개선된 라이브러리 코드를 업그레이드 하게 되는데, 이 경우에도 의존성의 문제가 발생할 수 있습니다.

**01.2.4 버전**

사용하는 라이브러리가 많아지고 업데이트 되면서 상호 의존성을 유지하기 위해서 사용하는 방법이 버전의 명칭입니다.

버전은 소프트웨어, 라이브러리의 특정한 시점에 대해서 이름을 지정합니다. 그리고 이러한 버전이름을 통하여 각각의 라이브러리의 의존관계를 지정합니다.

이렇게 하면 무작정 업데이트로 인하여 발생될 수 있는 의존성의 문제를 다소 해결하는데 도움이 됩니다.

**01.3 컴포저**

라이브러리들의 의존성을 좀더 쉽게 해결하기 위해서 관리를 해주는 전문 소프트웨어가 등장합니다. 컴포저는 PHP언어 세계에서 의존성을 관리하는 도구 입니다.

과거 PEAR 비해서 매우 설치와 사용이 간단합니다.

노드JS의 NPM, 파이썬 PIP, 루비 gem, 자바 Maven, 펄 cpan과 유사합니다. 컴포저를 이용하면 여러 개의 PHP 라이브러리 패키지를 한번에 불러와 적용하고, 관리가 가능합니다.

전용 의존성 관리 소프트웨어를 사용하면, 좀 더 쉽게 라이브러리를 배포하고 적용을 할 수 있습니다.

**01.3.1 패키지**

컴포저로 배포되는 라이브러리를 패키지(package)라고 부릅니다. 컴포저는 PHP 패키지 관리 시스템입니다. PHP 컴포저 페키지를 실행하고 사용하는지에 대해서 알아보도록 하겠습니다.

컴포터는 PHP를 통하여 개발을 할 때 라이브러리를 쉽게 사용하고 설치할 수 있도록 도와줍니다.

**01.3.1 라이브러리 설치와 업데이트**

컴포저는 install 명령을 통하여 패키지를 설치하고, update 명령을 통하여 기존 패키지들의 버전을 일괄적으로 업데이트가 가능합니다.

컴포저를 통하여 다양한 사람들이 생성한 라이브러리를 쉽게 사용을 할 수 있습니다.

**01.3.2 의존성 문제해결**

컴포저는 라이브러리의 버전을 이용하여 이러한 의존성 문제를 해결할려 노력하고 있습니다.

이전에는 라이브러리를 공통으로 설치하여 여러 개의 프로젝트에 사용을 하는 경우가 많이 있었습니다. 이로 인하여 라이브러리의 버전의 불일치로 인하여 여러 문제를 발생하였습니다.

컴포저는 각각의 프로젝트마다 라이브러리를 설치합니다. 이런 방식은 의존성 문제를 해결하고 관리 하는데 매우 유용합니다. 설치되는 라이브러리는 vendor라는 폴더를 만들어 관련 라이브러리 패키지를 설치합니다.

이렇게 독립적인 라이브러리 관리는 여러 프로젝트를 진행하는데 있어서 상호 간섭을 주지 않고 개별적으로 개발을 진행할 수 있다는 장점이 있습니다.

Modern PHP 개발환경이라면 composer는 필수적인 관리 도구라 해도 과언이 아닙니다.

**01.4 PEAR**

컴포저 이전에 PHP용으로 많이 사용하던 다른 패키지 도구들도 있습니다. 그중 대표적으로 PEAR가 있습니다.

PEAR도 패키지 단위로 라이브러리를 다운로드하여 설치를 관리해 줍니다.

$pear install 패키지이름

의존성 문제로 페키지를 한꺼번에 설치하는 옵션을 별도로 제공합니다. 모든 패키지를 설치할때는 -a옵션을 이용합니다. 선택적으로 할때는 -o옵션을 사용합니다.

PEAR는 서버 단위로 라이브러리를 설치합니다. 한번에 설치된 서버에서 패키지들을 공유하여 사용됩니다.