Abara One More

May.16.2023 Ash-Hun

Pray2U

Index

- Dependency Manager
- Class Dependency
- Package Dependecy
- Now…?

Dependency Manager

ⅰ. 의존성 관리란?

👉 프로젝트에서 외부의 어떤 라이브러리를 사용하고 있는지를 별도로 관리하는 것

ⅱ. 왜 의존성 관리를 해야하나?

프로젝트에서 사용하고 있는 외부 라이브러리들을 남들이 알 수 있도록 하기 위해 철저한 관리를 해야한다. 개인프로젝트면 모를까, 협업 프로젝트의 경우 해당 프로젝트에서 사용하고 있는 외부 라이브러리를 매번 받아야 한다. 그마저도 진행기간동안 라이브러리의 버전업이 되어 에러나는 경우도 있고 버전을 일일이 맞춰줘야하는 상황이 발생하기도 한다. 다만, 의존성 관리를 하게 되면 이를 좀 더 쉽게 자동화하여 관리할 수 있다.

Class Dependency

> "클래스 A는 클래스 B와 의존관계에 있다."
>
> 말이 어려워 보이지만 다르게 표현하면 "A 클래스는 B 클래스에 의존한다."는 것이다.
>
> 즉, 클래스 A를 설계할 때 클래스 B를 사용하는 경우 클래스 A가 없으면 클래스 B를 구현할 수 없고 이를 A는 B에 의존성을 가지고 있다고 표현한다.

위 예시를 보면 class A는 class B의 인스턴스를 생성자의 인자로 받아서 사용하고 있다.

이러한 클래스간 의존관계를 이상적으로 구현하는 방법이 몇가지 있는데 이는 이전 SOLID에 대한 얘기를 할때 나온것이기도 하니 키워드만 알아보고 넘어가자.

- **DI** (Dependency Injection)
- **IoC** (Inverse of Control)

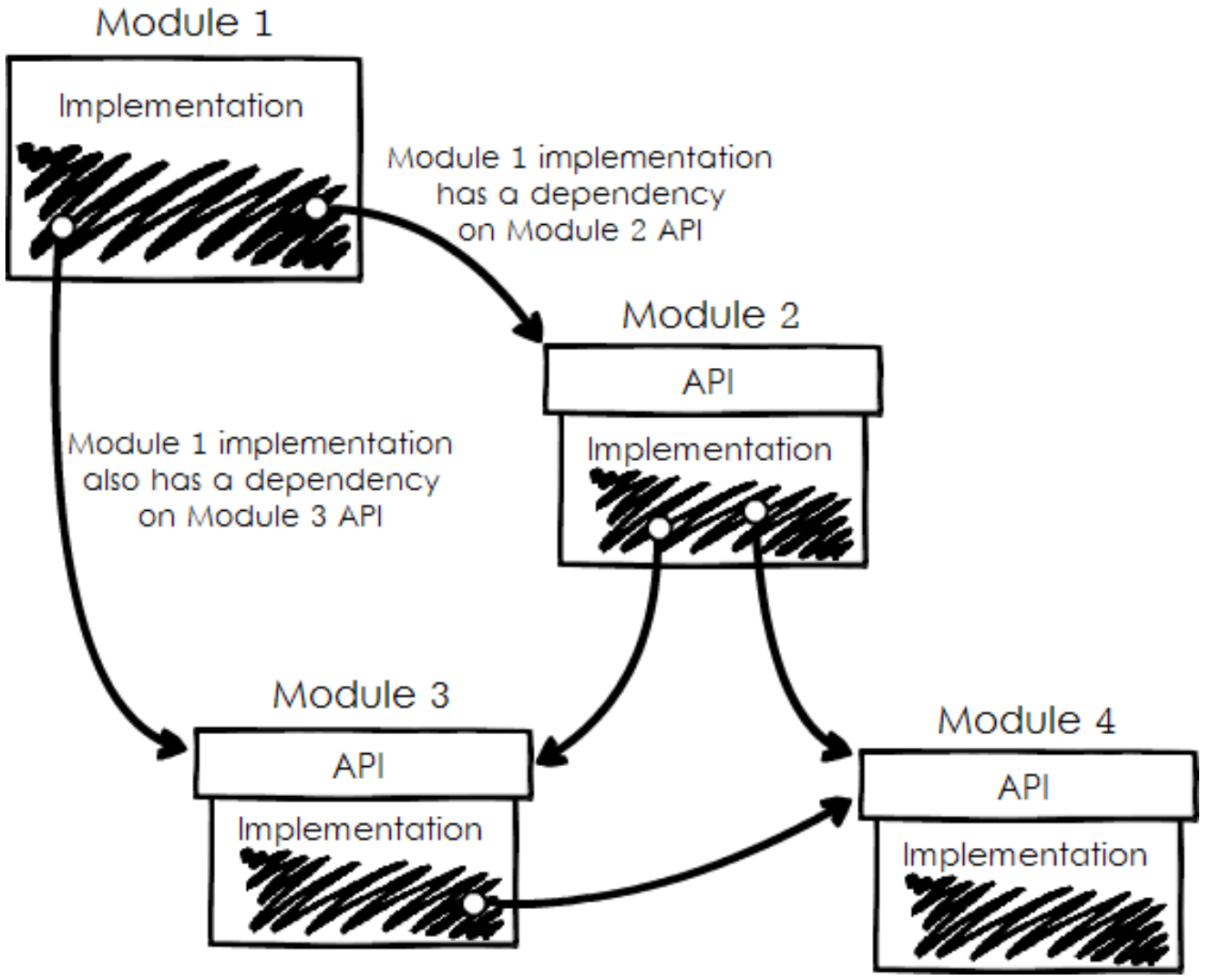
여담으로, **Spring Framework**에서는 제어의 역행(=IoC)와 의존성 주입(=DI)이 굉장히 잘 관리되는듯 하다. 역시 많은 사람이 선택한 것에는 이유가 다 있다~ ~~(tlqkf 딥러닝..ㅎ)~~

Java

```java
class A {
  private final B thisIsB;
  A (B thisIsB) {
    this.thisIsB = thisIsB;
  }
  String nameOfB() {
    return thisIsB.getName();
  }
}

class B {
  String name;
  B (String name) {
    this.name = name;
  }
  String getName() {
    return name;
  }
}
```

Package Dependency



Package Dependency

세상에 존재하는 다양한 기술 서비스에는 각기다른 프로그래밍 언어 혹은 프레임워크를 사용한다. 이는 자연스레 사용하는 기술스택에 따라 의존성 관리를 다르게 해야한다는 의미이면서 동시에 각각의 특징이 다르기 때문에 자연스럽게 이해해야하는 부분이기도 하다.

우리가 알고있는 의존성 관리 도구는 대개 `pip` , `npm` , `yarn` , `gem` , `maven` , `gradle` 등과 같은 종류가 있다.

Package Dependency

ⅰ. 의존성 관리 도구

소프트웨어 어플리케이션 개발에는 대부분 라이브러리가 필요하다. 바닥에서부터 구현하는 일이 잘 없다는 말과 같다. 즉, 대다수가 만들어져있는 도구들을 가져다 사용하게 되는데 이는 언어 차원에서 지원되는 것(built-in)일 수도 있고, 외부 라이브러리 저장소에 의존하는 것일 수도 있다.

그래도 일단은 프로젝트가 사용하고 있는 외부 라이브러리들을 남들이 알도록 하는 것이 기본적인 목표이다.

Package Dependency

(1) 왜 필요하징

- 프로그래밍 언어가 설치되면 자연스레 사용할 수 있는 Built-in tools이다.
  즉, 따로 관리할 필요가 없다!
- 의존성관리가 되지않으면 협업이 힘들어진다. 같은 프로젝트를 서로 다른 환경의 개개인이 접근하기 때문!
- 외부 라이브러리의 경우 문서화만 해둬도 큰 영향이 없지만 **의존성의 의존성**과 같은 **재귀 의존현상**과 **라이브러리 각각의 버전 업데이트 대응**을 별도의 도구 없이 해결하기엔 너무 낭비다.
- **배포 과정**에서 필요하다!
  예를 들어, **테스트와 같은 배포 전처리**를 위해 어플리케이션의 의존성들을 모두 다운로드해야 하고, 빌드/패키징 등과 같이 **컴파일이 들어가는 과정에서 항상 필요**하다.

Package Dependency

대부분 어떤 라이브러리의 무슨 버전을 쓸 지 목록화시킨 파일로서 의존성을 관리한다.
예시 1) Python의 requirements.txt

Plain Text

복사 캡션 ...

```
aiohttp==3.4.4
async-timeout==3.0.1
beautifulsoup4==4.6.3
Flask==1.0.2
Flask-Cors==3.0.7
Flask-JWT-Extended==3.13.1
Flask-RESTful==0.3.6
influxdb==5.2.0
jsonschema==2.6.0
mongoengine==0.16.1
multidict==4.5.0
nose==1.3.7
openpyxl==2.5.10
pymongo==3.7.2
redis==3.0.1
Werkzeug==0.14.1
```

pendency

예시 2) JavaScript의 package-lock.json

```
body-parser@1.18.2:
  version "1.18.2"
  resolved "https://registry.yarnpkg.com/body-parser/-/body-parser-1.18.2.tgz#87678a19c
  dependencies:
    bytes "3.0.0"
    content-type "~1.0.4"
    debug "2.6.9"

express@^4.15.3:
  version "4.16.2"
  resolved "https://registry.yarnpkg.com/express/-/express-4.16.2.tgz#e35c6dfe2d64b7dca
  dependencies:
    accepts "~1.3.4"
    array-flatten "1.1.1"
    body-parser "1.18.2"
    content-disposition "0.5.2"
    content-type "~1.0.4"
    cookie "0.3.1"
    cookie-signature "1.0.6"

fresh@0.5.2:
  version "0.5.2"
  resolved "https://registry.yarnpkg.com/fresh/-/fresh-0.5.2.tgz#3d8cadd90d976569fa835a

type-is@~1.6.15:
  version "1.6.15"
  resolved "https://registry.yarnpkg.com/type-is/-/type-is-1.6.15.tgz#cab10fb4909e441c8
  dependencies:
    media-typer "0.3.0"
    mime-types "~2.1.15"
```

예제 3) Ruby의 Gemfile

```
source 'https://rubygems.org'

gem 'sinatra'
gem 'httparty'
gem 'shopify_api'
gem 'dotenv'
```

Now...?

(4) 요즘은 뭘 쓰나?

최근 트렌드는 대부분 3가지 형태로 나타난다!

1. 프레임워크단의 패키지 관리

   → PIP, Conda, pipenv, etc...

2. Docker를 이용한 컨테이너 화

   → 개발환경의 컨테이너 화

3. 1과 2의 혼종

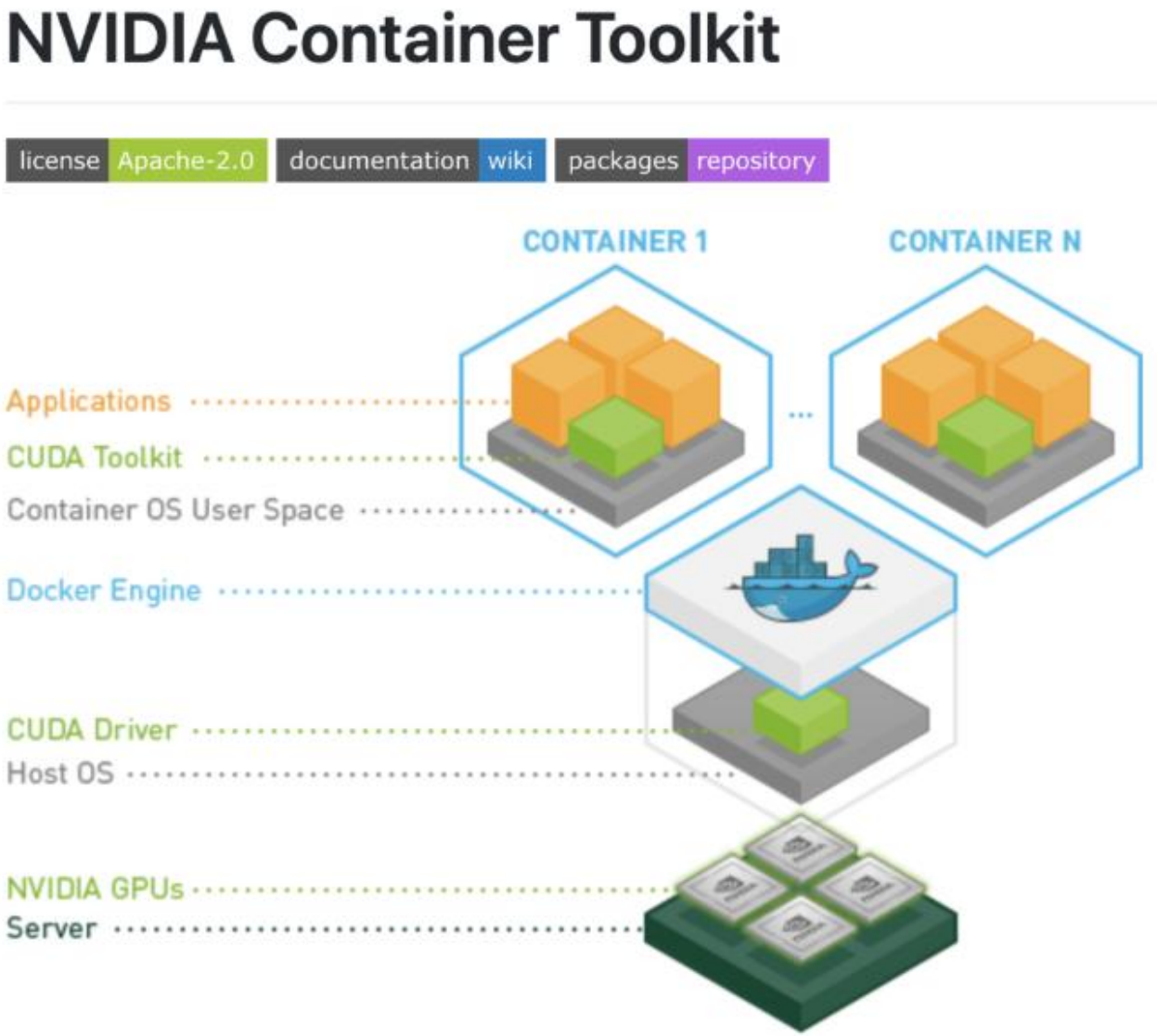
   → (프레임워크 패키지 관리 + 개발환경을 세팅) 컨테이너화

이중에서 단연 3번의 경우가 가장 깔끔하지만 대부분 1번과 2번의 비율이 높은 듯 하다.

왜냐하면 프로젝트 사이즈 및 현실적인 코스트 문제가 관여되기 때문!

하지만, 우리는 공부를 하는 입장이니 1,2,3 모두 알아놓을 필요가 있다.

Plus +



현재 Window 11 Home + Docker Engine + CUDA 11.8 + Geforce RTX 3070 + DeepLearning Framework 컨테이너화를 전부터 꾸준히 실험하고 있다. 조만간 더 실험할텐데 과정과 기록을 나중에 공개하도록 하겠다...!

Q & A

May.16.2023 Ash-Hun

Pray2U