baesy0 echo 경로 수정 c7fec0d 23 days ago

1 contributor

137 lines (104 sloc) 8.45 KB

의존성

제가 최초 프로그래밍 언어만 배울 때는 의존성을 모르고 프로그래밍을 하던 시절이었습니다. 그렇게 시간이 흐르고 제가 생각했던 형태로 프로그램들을 제작할 수 있는 단계까지 왔습니다. 몇가지 테스트를 끝내고 친구들 또는 그룹에게 제가 만든 코드를 배포했습니다. 당연히 잘 실행되는 곳도 있었지만, 실행이 되지 않거나 또는 에러가 발생했습니다. 저는 무엇을 생각하지 못하고 프로그래밍 한걸까요? 이 문제를 스스로 알기까지는 많은 시간이 필요했습니다. 이 문제와 엮여있는 문제점은 많겠지만 이 책은 그중에서도 의존성에 대해서 다룹니다. 그리고 의존성을 해결하기 위해서 제가 선택했던 Go 언어 이야기도 잠깐 다루게 됩니다.

의존성이란?

프로그램이 작동되기 위해 필요한 조건입니다.

- 연결된 소프트웨어 : DB, 기타 응용프로그램
- 필요한 라이브러리 또는 패키지
- 환경변수
- 프로그램을 사용하는 사용자의 특성
- 사내 소프트웨어라면 조직의 특성 및 셋팅 특이사항

위 사항이 소프트웨어에 얼마나 반영되고 연결되어 있는가? 에 대한 특성입니다. 특히나 리눅스, 유닉스계열의 OS에 사용되는 많은 패키지는 각 라이브러리에 대해 의존성을 많이 가지고 있습니다.

현대 프로그래밍에서 각 개발 분야별로 업무의 특성상 의존성 완벽하게 피하기는 힘듭니다. 프로그래머 스스로가 자신의 코드에 얼마나 많은 의존성을 가지고 있는지 분석하고 이 의존성을 줄이려고 노력한다면 보다 좋은 소프트웨어를 만들 수 있을것입니다.

의존성을 줄이면 얻는 이익

- 소프트웨어의 수명을 늘릴 수 있습니다.
- OS 또는 주변환경이 업그레이드되어도 자신의 프로그램에 미치는 영향이 적습니다.
- 버그의 범위가 줄어들고, 예측하기 쉽습니다.
- 다른 개발자의 사용률이 올라갑니다.

의존성 자가진단

여러분이 파이썬을 이용해서 중형규모의 소프트웨어를 제작했다고 가정하겠습니다. 몇가지 질문을 던지겠습니다. 아래 질문중에 일부는 의존성과도 연관이 있습니다.

- 여러분의 코드는 Windows, Linux(종류에 상관없이), macOS에서 잘 작동합니까?
- 여러분의 코드는 회사내부의 테스트 환경에서는 잘 작동합니다. 외부 환경에서도 잘 작동됩니까?
- 사용자의 컴퓨터에 파이썬 또는 필요한 라이브러리가 설치되어 있지 않을때 작동을 위해서 어떻게 배포합니까?
- 배포의 규모를 알 수 없을 때 계속 파이썬 또는 필요한 라이브러리를 설치해달라고 해야할까요?
- 여러분이 개발한 파이썬 버전과 사용자가 사용하는 파이썬 버전이 다르다면 어떤 오류가 생길지 예측할 수 있습니까?
- 파이썬은 다이나믹 렝귀지 입니다. 언어의 특징으로 인해서 버그를 가진 코드가 실행되기 직전까지는 실제 코드에 문법 에러가 존재한다는 것을 알아차리기 힘듭니다. 이런경우 어떻게 여러분의 코드를 테스트하겠습니까?
- 제 자리에서 코드를 테스트할 때는 에러가 없었습니다. 하지만 다른 사용자 측에서는 환경 및 다른 셋팅이 꼬여있어서 에러가 발생합니다.

이 문제는 누구의 문제일까요? 어떻게 이 문제를 개선하겠습니까?

의존성과 클라우드

많은 기업들이 클라우드를 준비합니다. 수많은 컴퓨터가 필요시 생성 및 셋팅됩니다. 설치된 소프트웨어들이 재빨리 설치되고 잘 작동해야하며, 비용의 이슈로 사용이 끝난 머신은 제거되어야 합니다. 이런 환경은 의존성을 테스트하기에 사실은 굉장히 좋은 환경입니다.

클라우드에서는 매번 아래 절차에 의해서 소프트웨어가 실행되기 때문입니다. (배포를 위한 CI툴에서 비슷한 절차를 가집니다.)

- 가상 인스턴스 생성
- 소프트웨어 작동을 위한 프로그램설치
- 환경변수 셋팅
- 프로그램 실행
- 인스턴스 제거

의존성을 생각하며 소프트웨어를 제작했다면 클라우드환경에서도 프로그램은 잘 작동할 수 밖에 없습니다.

링커, 공유라이브러리

컴파일러는 코드를 기계어로 변환하는 역할을 합니다. 링커는 컴파일러가 만든 결과물을 서로 연결시키는 역할을 합니다. 구체적으로 이야기하면 링커는 우리가 만든 코드와 라이브러리를 연결합니다.

옛날에는 컴파일러와 링커가 서로 역할을 나누어서 컴파일 작업진행을 했지만, 현대 컴파일러는 컴파일러와 링커의 역할을 같이 하고 있습니다.

라이브러리중에서 의존성과 관련되어 우리가 소프트웨어를 설치하고 배포시 우리를 괴롭히는것은 동적 라이브러러리 라고 불리는 파일입니다.

윈도우즈에서는 dll, 리눅스에서는 so, 맥에서는 dylib 확장자를 가지고 있습니다. 자주 사용하는 라이브러리를 매 소프트웨어마다 넣으면 소프트웨어의 용량도 커지고 관리상 비효율적이기 때문에 이런형태를 띄게 되었습니다. 분명 이구조는 효율적이지만 이 특징때문에 많은 의존성문제를 발생시킵니다. 동적라이브러리 또는 동적라이브러리를 사용하는 소프트웨어를 우리가 제작하는 코드에 넣게 되면 코드 그 자체로는 실행되지 않습니다. OS에 존재하는 많은 동적라이브러리가 물려서 실행되게 됩니다. OS에서 해당 공유라이브러리가 없거나 버전이 너무 크게 다르다면 우리가 만든 실행파일은 실행될 수 없습니다.

우리가 리눅스에서 자주 사용하는 파일중 하나인 `echo` 명령어는 터미널에서 입력받은 인수를 출력하는 명령어 입니다. 이 명령어가 사용하는 공유라이브러리를 체크해보겠습니다.

리눅스에는 공유라이브러리를 체크하기 위해서 ldd 라는 명령어가 존재합니다. /bin/echo 명령어를 한번 ldd로 체크해보겠습니다.

```
$ ldd /usr/bin/echo
```

출력은 아래과 같습니다.

```
        linux-vdso.so.1 =>  (0x00007ffd7b3c8000)
        libc.so.6 => /lib64/libc.so.6 (0x00007fadbdb9a000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fadbdf67000)
```

두번째에 보이는 libc.so.6 파일을 다시 ldd 체크하겠습니다.

```
$ ldd /lib64/libc.so.6
```

다시 출력은 아래와 같습니다.

```
        /lib64/ld-linux-x86-64.so.2 (0x00007f0b0573e000)
        linux-vdso.so.1 =>  (0x00007ffe356a9000)
```

다시 ld-linux-x86-64.so.2를 ldd 체크합니다.

```
        statically linked
```

echo명령어를 실행하기 위해서 ldd를 체크해보면 수많은 .so파일이 필요합니다. 이 파일이 없다면 echo는 실행될 수 없습니다.

대부분 주요한 라이브러리는 OS가 설치될 때 대부분 자동설치됩니다만, 링크된 특징을 알 필요는 있습니다. 아마 계속 타이핑하면 언젠가는 사용되는 모든 동적라이브러리를 알아낼 수 있습니다. 우리가 기능상 단순하게 생각했던 echo 명령어가 이제는 단순해보이지 않습니다.

다른 OS에서 의존성 체크하기

macOS

otool이 리눅스의 ldd와 같은 역할을 합니다.

```
$ otool -L /bin/echo
```

Windows

- 만약 윈도우즈에서도 공유라이브러리를 체크하고 싶다면, Cygwin을 설치하면 ldd를 사용할 수 있습니다.
- DependencyWalker라는 소프트웨어를 이용하여 의존성을 체크할 수 있습니다.

상용 프로그램 vs 성숙된 오픈소스의 사용

프로그램을 작성할 때 자신이 원한는 기술이 상용프로그램과 오픈소스 솔루션중 둘다 존재한다면 두개의 프로그램을 전부 테스트해보세요. 결과의 차이가 크게 나지 않고, 상용툴와 오픈소스 두개중 선택해서 의존성을 가져가야 한다면 개인적으로 성숙한 오픈소스에 의존성을 가지는 것을 추천합니다.

가치가 발휘하는 순간

의존성을 생각하며 프로그래밍하는것은 결코 쉽지 않은일 입니다. 하지만 이런 특성들을 고민하고 프로그래밍에 적용한다면 분명 미래에 큰 가치로 여러분에게 돌아올 것 입니다.