파이썬 프로그래머를 위한 러스트 입문

윤인도

freedomzero91@gmail.com

CH15. 파이썬 바인딩

이번 챕터에서는 러스트 코드를 파이썬에서 실행하는 방법을 배워보겠습니다. 러스트의 높은 성능 때문에, 많은 파이썬 개발자들이 러스트 코드를 파이썬에서 사용하기 위한 도구를 개발했습니다. 그 중에서 가장 널리 사용되는 PyO3 크레이트의 사용법을 알아보겠습니다.

파이썬 가상환경 만들기

가상환경이란?

- 프로젝트 단위로 의존성이 관리되는 러스트와 달리, 파이썬은 하나의 글로벌 인터프리터 환경에 모든 패키지가 설치됩니다.
- 프로젝트들에서 같은 패키지를 재사용할 수 있다는 장점이 있지만, 프로젝트별로 다른 파이썬 버전과 패키지 버전을 관리하는 일이 어려워지는 문제가 발생합니다.
- 파이썬에서 가상 환경을 생성하는 방법은 여러 가지가 있지만, 여기서는 `pipenv`를 사용하는 방법을 소개합니다.

pipenv

pipenv는 pipenv 명령어 하나로 가상환경의 생성, 삭제, 의존성의 추가, 삭제, 업데이트 등을 모두 할 수 있는 편리한 도구입니다.

pipenv는 프로젝트의 가상 환경을 자동으로 생성 및 관리하고 패키지를 설치/제거할 때 `Pipfile`에서 패키지를 추가/제거합니다. 또한 패키지 유효성을 검사하는 데 사용되는 매우 중요한 `Pipfile.lock`을 생성합니다.

```
pip install pipenv
```

파이썬 버전을 지정해서 가상환경을 생성합니다.

```
pipenv --python 3.11
```

생성된 가상환경 셸로 진입하는 방법은 다음과 같습니다.

```
pipenv shell
```

가상환경에 새로운 패키지를 설치합니다.

```
pipenv install requests
```

만일 개발 단계에서만 사용되는 툴이라면 `--dev` 플래그를 추가합니다. 예를 들어 black과 같은 포매터 패키지는 실제 소스코드에서는 쓰이지 않고 개발 단계에서만 사용되기 때문에 다음과 같시 설치할 수 있습니다.

```
pipenv install --dev black
```

결과적으로 일반 패키지와 개발 패키지가 Pipfile에 구분되어서 추가되는 것을 알 수 있습니다.

```
[[source]]
url = "https://pypi.org/simple"
verify_ssl = true
name = "pypi"

[packages]
requests = "*"

[dev-packages]
black = "*"

[requires]
python_version = "3.11"

[pipenv]
allow_prereleases = true
```

러스트 프로젝트 생성하기

파이썬 바인딩이란?

파이썬 바인딩은 다른 프로그래밍 언어로 작성된 코드를 파이썬에서 사용하는 것을 의미합니다.

PyO3

PyO3는 파이썬에서 러스트 코드를 실행할 수 있고, 반대로 러스트에서 파이썬 코드를 실행할 수 있도록 도와주는 크레이트입니다.

maturin

maturin은 최소한의 구성으로 러스트로 작성한 파이썬 패키지를 빌드할 수 있는 도구입니다.

```
$ pipenv install maturin --dev
$ pipenv shell
```

maturin으로 pyo3 프로젝트를 시작합니다. -b 옵션을 주면 pyo3를 빌드 시스템으로 해서 프로젝트가 생성됩니다.

```
$ maturin init -b pyo3
  ✨ Done! New project created string_sum
```

프로젝트를 생성하면 다음과 같은 폴더 구조가 만들어집니다.

```
.
├── Cargo.lock
├── Cargo.toml
├── Pipfile
├── pyproject.toml
├── src
    └── lib.rs
```

`Cargo.toml` 파일에서 패키지와 라이브러리의 이름을 "fibonacci"로 변경합니다.

```toml
[package]
name = "fibonacci"
version = "0.1.0"
edition = "2021"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html
[lib]
name = "fibonacci"
crate-type = ["cdylib"]

[dependencies]
pyo3 = { version = "0.16.5", features = ["extension-module"] }
```

`pyproject.toml` 파일에서도 프로젝트 이름을 "fibonacci"로 수정합니다.

```
[build-system]
requires = ["maturin>=0.13,<0.14"]
build-backend = "maturin"

[project]
name = "fibonacci"
requires-python = ">=3.7"
classifiers = [
    "Programming Language :: Rust",
    "Programming Language :: Python :: Implementation :: CPython",
    "Programming Language :: Python :: Implementation :: PyPy",
]
```

라이브러리 크레이트 만들기

- 러스트 함수에 `#[pyfunction]` 어트리뷰트를 추가하면 PyO3는 해당 함수가 일반 파이썬 함수인 것처럼 파이썬에서 호출할 수 있는 코드를 생성합니다.
- 러스트 함수에 `#[pymodule]` 어트리뷰트를 추가하면 PyO3는 해당 함수를 파이썬 모듈의 초기화 함수로 사용할 수 있는 코드를 생성합니다.
- 모듈에 함수를 추가하려면 `add_function` 메서드를 사용합니다.

```rust
use pyo3::prelude::*;

fn _run(n: u64) -> u64 {
    match n {
        0 => 0,
        1 => 1,
        _ => _run(n - 1) + _run(n - 2),
    }
}

#[pyfunction]
fn run(n: u64) -> PyResult<u64> {
    Ok(_run(n))
}

/// A Python module implemented in Rust.
#[pymodule]
fn fibonacci(_py: Python, m: &PyModule) -> PyResult<()> {
    m.add_function(wrap_pyfunction!(run, m)?)?;
    Ok(())
}
```

파이썬에서 러스트 코드 실행해 보기

개발 모드로 빌드해보기

`maturin develop` 명령어를 사용하면, 러스트 패키지를 빌드한 다음 파이썬 가상환경에 패키지를 자동으로 설치해줍니다. 이때 러스트 컴파일 타겟이 `[unoptimized + debuginfo]`가 되는데, 빠른 개발을 위해 코드 성능보다는 컴파일 속도를 중요하게 생각한 옵션입니다.

```
$ maturin develop
🔗 Found pyo3 bindings
🐍 Found CPython 3.8 at /Users/.local/share/virtualenvs/ch14-4UzrGkRt/bin/python
   Compiling pyo3-build-config v0.16.5
   Compiling pyo3-ffi v0.16.5
   Compiling pyo3 v0.16.5
   Compiling fibonacci v0.1.0 (/Users/code/Tutorials/sap_rust_tutorial/ch14)
    Finished dev [unoptimized + debuginfo] target(s) in 12.64s
📦 Built wheel for CPython 3.8 to /var/folders/74/l6jhlmk114g8kx1pzz2s9fm80000gn
/T/.tmpBh1Xiw/fibonacci-0.1.0-cp38-cp38-macosx_10_7_x86_64.whl
🛠️  Installed fibonacci-0.1.0
```

> 만일 가상환경에서 실행하지 않을 경우 에러가 발생하므로 주의하세요!

```
$ maturin develop
💥 maturin failed
  Caused by: You need to be inside a virtualenv or conda environment to use develop
(neither VIRTUAL_ENV nor CONDA_PREFIX are set).
See https://virtualenv.pypa.io/en/latest/index.html
on how to use virtualenv or use `maturin build` and
`pip install <path/to/wheel>` instead.
```

`main.py` 파일을 만들고 다음 코드를 추가합니다. 파이썬으로 피보나치 수열을 구하는 함수 `pyrun` 을 추가해 러스트 구현체와 성능을 비교해봅니다.

```
import time

from fibonacci import run


def pyrun(n: int):
    if n < 2:
        return n

    return pyrun(n - 1) + pyrun(n - 2)


N = 35

start = time.time()
result = pyrun(N)
print(f"python: {time.time()-start:.2f}, result: {result}")
start = time.time()
result = run(N)
print(f"rust: {time.time()-start:.2f}, result: {result}")
```

실행 결과

```
$ python main.py
python: 3.13, result: 9227465
rust: 0.10, result: 9227465
```

릴리즈 모드로 빌드해보기

빌드 옵션을 `--release` 로 주면, 러스트 코드를 최대한 최적화해서 컴파일한 바이너리가 패키지로 만들어지게 됩니다. 컴파일 타겟이 `[optimized]` 인 걸 알 수 있습니다.

```
$ maturin build --release
🔗 Found pyo3 bindings
🐍 Found CPython 3.8 at /Users/.local/share/virtualenvs/temp-nO4s4P8m/bin/python3
   Compiling target-lexicon v0.12.4
    ...
   Compiling pyo3-macros v0.16.5
   Compiling fibonacci v0.1.0 (/Users/code/temp)
    Finished release [optimized] target(s) in 20.61s
📦 Built wheel for CPython 3.8 to /Users/code/temp/target/wheels/fibonacci-0.1.0-cp38-cp38-macosx_10_7_x86_64.whl
```

이제 파이썬 코드를 그대로 실행하면 최적화된 패키지로 실행이 가능합니다.

실행 결과

```
$ python main.py
python: 3.03, result: 9227465
rust: 0.03, result: 9227465
```

PyO3와 GIL

앞에서는 단일 스레드 환경에서 러스트 코드를 실행하는 예제를 살펴봤습니다. 하지만 이 강의의 가장 큰 목표인 파이썬 GIL을 우회하는 러스트 패키지를 만들기 위해서는 멀티스레드 환경에서 러스트 코드를 실행해보아야 합니다. 이를 살펴보기 위해서 새로운 프로젝트 `gil`을 생성합니다.

```
maturin init -b pyo3
```

GIL 획득과 해제

- `py.allow_threads`는 PyO3에서 제공하는 메서드로, 클로저를 실행하는 동안 GIL을 일시적으로 해제
- PyO3에서 GIL을 해제하는 다른 방법으로는 `Python::with_gil` 메서드를 사용하여 명시적으로 GIL을 획득하고 해제

```rust
// lib.rs
use std::thread;
use std::time::Duration;

use pyo3::prelude::*;
use pyo3::types::PyList;

#[pyfunction]
fn double_list(list: &PyList, result: &PyList, idx: usize) -> PyResult<()> {
    println!("Rust: Enter double_list...");
    thread::sleep(Duration::from_secs(1));
    let doubled: Vec<i32> = list.extract::<Vec<i32>>()?.iter().map(|x| x * 2).collect();
    Python::with_gil(|py| {
        println!("Rust: Acquire GIL...");
        thread::sleep(Duration::from_secs(1));
        let py_list = PyList::new(py, &doubled);
        println!("Rust: Exit...");
        result.set_item(idx, py_list)
    })
}

/// A Python module implemented in Rust.
#[pymodule]
fn gil(_py: Python, m: &PyModule) -> PyResult<()> {
    m.add_function(wrap_pyfunction!(double_list, m)?)?;
    Ok(())
}
```

파이썬에서 `sleep` 함수를 사용해 GIL을 해제한 다음 러스트 코드와 번갈아가면서 실행되는 예제입니다.

```
import time
import threading

from gil import double_list


def double_list_py(list, result, idx):
    print("Py: Enter double_list_py...")
    time.sleep(0.1)
    result[idx] = [x * 2 for x in list]
    print("Py: Exit...")


result = [[], []]
nums = [1, 2, 3]

t1 = threading.Thread(target=double_list_py, args=(nums, result, 0))
t2 = threading.Thread(target=double_list, args=(nums, result, 1))

t1.start()
t2.start()

t1.join()
t2.join()

print(f"Py: {result[0]}")
print(f"Rust: {result[1]}")
```

실행 결과

```
Py: Enter double_list_py...
Rust: Enter double_list...
Rust: Acquire GIL...
Rust: Exit...
Py: Exit...
Py: [2, 4, 6]
Rust: [2, 4, 6]
```