파이썬 프로그래머를 위한 러스트 입문

윤인도

freedomzero91@gmail.com

CH8. 모듈과 크레이트

Rust의 모듈 시스템

러스트의 모듈 시스템은 아래 4가지를 말합니다.

- 패키지(Packages) : cargo에서 제공하는 기능으로, crate를 빌드하고 생성할 수 있습니다.
- 크레이트(Crates) : 라이브러리 또는 바이너리를 생성하는 모듈 트리(a tree of modules)입니다.
- `mod` 와 `use` : 코드 안에서 다른 모듈들을 구성하고, 불러오거나 다른 모듈에 노출할 지 여부(private or public)를 결정합니다.
- 경로: 모듈에서 특정 요소(함수, 구조체, 변수 등)를 찾기 위한 방법

패키지

`cargo.toml` 파일

하나의 패키지에는 단 하나의 라이브러리 크레이트만 포함할 수 있습니다. 하지만 바이너리 크레이트는 여러 개를 넣을 수 있습니다.

크레이트

바이너리 크레이트

`main.rs`

```
cargo new
```

컴파일되어 바이너리 파일을 생성하는 크레이트입니다.

라이브러리 크레이트

`lib.rs`

```
cargo new --lib
```

컴파일되지 않기 때문에 바이너리를 생성하지 않습니다. 다른 크레이트나 패키지에서 코드를 참조할 수 있도록 제공되는 크레이트입니다.

크레이트 루트

크레이트 루트란 컴파일 엔트리포인트를 의미합니다. 바이너리 크레이트는 `src/main.rs` 파일이, 라이브러리 크레이트는 `src/lib.rs` 파일이 크레이트 루트가 됩니다.

private vs public

러스트의 모든 모듈과 객체는 기본적으로 private입니다. 외부에서 모듈에 접근하거나 모듈 내부의 객체에 접근을 허용하려면 `pub` 키워드를 사용해야 합니다.

```
pub mod {

}

pub fn {

}

pub struct {

}

pub static
```

`use` 와 `mod`

`use` 키워드는 특정 경로를 현재 스코프로 가져오는 역할을 합니다. 주의해야 하는 점은 경로는 항상 크레이트 루트로부터 시작된다는 점입니다.

`mod` 키워드는 해당 모듈을 사용하겠다고 선언하는 역할입니다. 예를 들어 `mod new_module`이 사용되면, 컴파일러는 아래 위치에서 해당 모듈을 찾아봅니다.

1. `mod new_module` 다음에 해당 모듈의 정의가 나와야 합니다.

```
mod new_module {
  fn new_func() {
    ...
  }
  ...
}
```

2. `src/new_module.rs` 파일을 찾아봅니다.
3. `src/new_module` 폴더에서 `mod.rs` 파일을 찾아봅니다.

```
pub mod new_module;
```

특정 모듈에 대한 접근은 크레이트 루트를 기준으로 절대경로를 사용하면 됩니다. 예를 들어 코드 어디에서라도 다음과 같이 모듈에 접근이 가능합니다.

```
// src/new_module.rs -> MyType
use crate::new_module::MyType
```

상대 경로도 사용 가능합니다.

`self` 는 struct 자기 자신

```
mod mod2 {
    fn func() {
        println!("mod2::func()");
    }
    mod mod1 {
        pub fn func() {
            println!("mod2::mod1::func()");
        }
    }
    pub fn dummy() {
        func();
        self::func();
        mod1::func();
        self::mod1::func();
    }
}
fn main() {
    mod2::dummy();
}
```

`super`는 현재 모듈의 상위 모듈을 의미합니다.

`super` 는 현재 모듈의 상위 모듈을 의미합니다.

```
mod mod1 {
    pub fn dummy() {
        println!("Hello, world!");
    }
}

mod mod2 {
    // use crate::mod1;
    use super::mod1;

    pub fn dummy() {
        mod1::dummy();
    }
}

fn main() {
    mod2::dummy();
}
```

모듈과 크레이트 사용해보기

파이썬 폴더에 `my_modle.py` 를 생성합니다.

```
def greet():
    print(f"Hi! I am hello_bot")


class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def get_older(self, year):
        self.age += year
```

이제 이 함수와 클래스를 `main.py` 에서 참조합니다.

```
from my_module import greet, Person

if __name__ == '__main__':
    greet()

    john = Person("john", 20)
    john.get_older(3)
    print(john.age)
```

이번에는 `bots` 폴더를 만들고 `hello_bot.py` 파일을 추가합니다.

```
.
├── bots
│   └── hello_bot.py
├── main.py
└── my_module.py
```

`hello_bot.py` 는 다음과 같습니다.

```
BOT_NAME = "hello_bot"


def hello():
    print("Hello, humans!")
```

`my_module.py`에서 `greet` 함수가 `BOT_NAME` 을 이용하도록 합니다.

```
from bots.hello_bot import BOT_NAME


def greet():
    print(f"Hi! I am {BOT_NAME}")
```

그 다음 `main.py` 에서 `bots` 모듈을 사용해 보겠습니다.

```
from bots.hello_bot import hello
from my_module import greet, Person

if __name__ == '__main__':
    hello()

    greet()

    john = Person("john", 20)
    john.get_older(3)
    print(john.age)
```

이번에는 동일한 구조를 러스트에서 구현해 보겠습니다. src 폴더에 `my_module.rs` 를 생성합니다.

```
src
├── main.rs
└── my_module.rs
```

이때 public으로 만드려면 `pub` 키워드를 사용해야 합니다.

```rust
pub fn greet() {
    println!("Hi! I am hello_bot");
}

pub struct Person {
    pub name: String,
    age: i32,
}

impl Person {
    pub fn new(name: &str, age: i32) -> Self {
        Person {
            name: String::from(name),
            age: age,
        }
    }
    pub fn get_older(&mut self, year: i32) {
        self.age += year;
    }
}
```

```rust
mod my_module; // will look for a file src/my_module.rs

// actually import the function and struct from my_module.rs
use my_module::{greet, Person};

fn main() {
    greet();

    let mut john = Person::new("john", 20);
    john.get_older(3);
    println!("{}", john.name);
    // println!("Am I alive? {}", john.alive); // won't compile!
}
```

다음으로는 하위 폴더 `bots`를 만들어 보겠습니다. `bots` 폴더에는 `hello_bot.rs`와 `mod.rs` 두 파일을 생성합니다.

```
src
├── bots
│   ├── hello_bot.rs
│   └── mod.rs
├── main.rs
└── my_module.rs
```

항상 하위 폴더를 모듈로 만드는 경우에는 `mod.rs` 가 있어야 합니다. 이 파일은 해당 모듈의 엔트리 포인트가 되어 이 모듈 안에 있는 다른 하위 모듈들을 찾을 수 있게 합니다. 따라서 `mod.rs` 에는 `hello_bot` 모듈의 정보가 있어야 합니다.

```
pub mod hello_bot; // will look for hello_bot.rs
```

이제 `hello_bot.rs` 파일을 작성합니다.

```
pub static BOT_NAME: &str = "hello_bot";

pub fn hello() {
    println!("Hello, humans!");
}
```

static 변수와 함수 하나가 생성되어 있고, 둘 다 public으로 선언되었습니다. 먼저, `BOT_NAME` 스태틱을 `src/my_module.rs` 에서 참조해 보겠습니다. `my_module.rs` 는 크레이트 루트가 아니기 때문에 `use crate::` 문법으로 참조해야 합니다. 여기서 `greet` 함수가 이 `BOT_NAME` 스태틱을 참조해 실행되도록 수정해 봅시다.

```
use crate::bots::hello_bot::BOT_NAME;

pub fn greet() {
    println!("Hi! I am {}", BOT_NAME);
}
```

이제 `main.rs` 에서 `bots` 모듈을 사용해 보겠습니다. `main.rs` 는 크레이트 루트기 때문에 `use bots::hello_bot::hello;` 로 모듈을 불러올 수 있습니다.

```
mod my_module; // will look for a file src/my_module.rs
mod bots; // will look for a file src/hello/mod.rs

use my_module::{greet, Person};
use bots::hello_bot::hello;


fn main() {
    hello();

    greet();

    let mut john = Person::new("john", 20);
    john.get_older(3);
    println!("{}", john.name);
    // println!("Am I alive? {}", john.alive); // won't compile!
}
```