러스트의 Option 타입

러스트에서 Option은 값이 있을 수도 있고 없을 수도 있는 상황을 안전하게 표현하는 핵심 타입입니다. null 참조 오류를 컴파일 타임에 방지하여 더 안정적인 코드를 작성할 수 있게 해줍니다.

Option<T>의 정의

Option은 두 가지 상태를 가진 열거형입니다. Some(T)는 값이 존재하는 경우를 나타내고, None은 값이 없는 경우를 표현합니다. 이를 통해 null 대신 타입 안전한 방식으로 "값의 부재"를 다룰 수 있습니다.

왜 Option을 사용하나요?

Option을 사용하면 컴파일러가 값의 존재 여부를 확인하도록 강제합니다. null 포인터 참조 오류를 런타임이 아닌 컴파일 타임에 잡아낼 수 있어, 프로그램의 안정성이 크게 향상됩니다.

실무 활용 시나리오

검색 결과가 없을 수 있는 경우, 설정값이 선택적인 경우, 파일 읽기가 실패할 수 있는 경우 등 다양한 상황에서 Option을 활용합니다. 명시적인 처리를 통해 예상치 못한 버그를 사전에 방지할 수 있습니다.

```
enum Option<T> {
    Some(T),
    None,
}
```

러스트의 Option은 함수형 프로그래밍의 Maybe 타입과 유사한 개념으로, 값의 존재 여부를 타입 시스템에 명시적으로 표현합니다. 이는 러스트가 메모리 안전성을 보장하는 핵심 메커니즘 중 하나입니다.

Result 타입으로 에러 처리하기

Result는 러스트에서 작업의 성공 또는 실패를 명시적으로 다루는 타입입니다. 예외 처리 대신 Result를 사용하여 에러를 값으로 취급하고, 컴파일러의 도움을 받아 모든 에러 케이스를 처리할 수 있습니다.

Result<T, E>의 구조

Result는 두 가지 변형을 가진 열거형입니다:

- **Ok(T)**: 성공한 경우, 결과값 T를 포함
- **Err(E)**: 실패한 경우, 에러 정보 E를 포함

이 명시적인 구조 덕분에 모든 에러 경로를 추적하고 처리할 수 있습니다.

실무에서의 활용

파일 입출력, 네트워크 통신, 데이터 파싱 등 실패할 수 있는 모든 작업에서 Result를 사용합니다. ? 연산자를 통해 에러 전파를 간결하게 표현할 수 있으며, match나 if let으로 세밀한 에러 처리가 가능합니다.

1

에러 발생

작업 실행 중 문제가 발생하면 Err 변형으로 에러 정보를 반환합니다.

2

에러 전파

? 연산자로 에러를 호출자에게 자동으로 전달하여 보일러플레이트를 줄입니다.

3

에러 처리

match나 unwrap_or 등의 메서드로 에러를 적절히 처리하거나 기본값을 제공합니다.

```
fn read_file(path: &str) -> Result<String, std::io::Error> {
    std::fs::read_to_string(path)
}

// ? 연산자로 에러 전파
fn process_file(path: &str) -> Result<usize, std::io::Error> {
    let content = read_file(path)?;
    Ok(content.len())
}
```

Result를 사용하면 에러가 "특별한 상황"이 아니라 정상적인 프로그램 흐름의 일부가 됩니다. 이는 더 견고하고 예측 가능한 코드를 작성하는 데 도움이 됩니다.

Unit Type: 러스트의 빈 값

Unit 타입 ()은 러스트에서 "의미 있는 값이 없음"을 나타내는 특수한 타입입니다. 단순히 값이 없는 것이 아니라, 작업이 완료되었지만 반환할 값이 없다는 의미를 명확히 전달합니다.

Unit 타입이란?

()는 정확히 하나의 값만 가지는 타입으로, 그 값 역시 ()입니다. 이는 C/Java의 void와 비슷하지만, 러스트에서는 실제 타입으로 취급되어 다른 타입들과 동일하게 사용할 수 있습니다.

언제 사용하나요?

부수 효과만 수행하는 함수(예: 파일 쓰기, 로그 출력), 에러 처리에서 성공만 표시하는 경우(Result<(), Error>), 반복문이나 패턴 매칭에서 특정 분기 등에서 Unit 타입을 사용합니다.

실용적 예시

함수가 명시적인 return을 사용하지 않으면 마지막 표현식이 세미콜론으로 끝나는 경우 자동으로 ()를 반환합니다. 이는 러스트의 표현식 기반 문법에서 매우 자연스럽게 동작합니다.

```
// 명시적 Unit 반환
fn print_message(msg: &str) -> () {
    println!("{}", msg);
}

// 암묵적 Unit 반환 (더 일반적)
fn update_cache(key: String, value: String) {
    // 캐시 업데이트 로직
    cache.insert(key, value);
    // 세미콜론으로 끝나므로 () 반환
}

// Result와 함께 사용
fn save_config(path: &str) -> Result<(), Error> {
    // 저장 성공 시 Ok(())
    // 실패 시 Err(error)
    Ok(())
}
```

핵심 포인트

- Unit은 실제 타입이며 값입니다
- 반환값이 필요 없는 함수에 사용
- Result<(), E>로 성공/실패만 표현
- 표현식 vs 문장을 구분하는 기준

팁: 세미콜론의 유무가 Unit 타입 반환 여부를 결정합니다. 마지막 표현식에 세미콜론을 붙이면 ()가 반환됩니다.

Unit 타입은 러스트의 타입 시스템이 얼마나 일관성 있게 설계되었는지를 보여주는 좋은 예입니다. "값이 없음"조차도 명시적인 타입으로 표현하여, 프로그램의 의도를 더욱 명확하게 만듭니다.