HSPACE Rust 특강

Rust Basic #1 - 기본 타입

Chris Ohk

utilForever@gmail.com

기본 타입

- 고정된 크기를 갖는 수치 타입
- `bool` 타입
- 문자
- 튜플 (Tuple)
- 포인터 타입
- 배열, 벡터 (Vector), 슬라이스 (Slice)
- 문자열 타입
- 타입 별칭

고정된 크기를 갖는 수치 타입

- 정수 타입
  - 부호 없음 : `u8`, `u16`, `u32`, `u64`, `u128`, `usize`
  - 부호 있음 : `i8`, `i16`, `i32`, `i64`, `i128`, `isize`
  - `usize`와 `isize`는 C/C++의 `size_t`와 `ptrdiff_t`에 해당하는 타입이다.
  - 타입을 나타내는 접미사를 가질 수 있다. (예 : `42u8` → `u8` 값, `1729isize` → `isize` 값)
  - 접두사 `0x`, `0o`, `0b` → 16진수, 8진수, 2진수 리터럴
  - 길이가 긴 수는 `_`를 넣어서 좀 더 읽기 쉽게 만들 수 있다.
  - Rust는 수치 타입과 `char` 타입을 서로 별개로 보지만,
    그 대신 `u8` 값으로 쓸 수 있는 문자열 리터럴인 바이트 리터럴(Byte Literal)을 제공한다. (예 : `b'A' = 65u8`)
  - `as` 연산자를 사용해 한 정수 타입을 다른 정수 타입으로 변환할 수 있다.
    (Rust는 암시적인 타입 변환이 일어나지 않는다.)

고정된 크기를 갖는 수치 타입

- 점검, 순환, 포화, 넘침 산술
  - Rust에서 정수 산술 연산이 오버플로를 일으키면?
    - 디버그 빌드에서는 패닉에 빠짐
    - 릴리즈 빌드에서는 연산의 결과가 끝에서 끝으로 순환됨 (Wrap Around)
  - 정수 산술 메소드는 크게 네 가지 범주로 나뉨
    - 점검(Checked) : 결과를 `Option`에 담아 반환
    - 순환(Wrapping) : 수학적으로 옳은 결과를 주어진 값의 범위로 나눈 나머지에 해당하는 값을 반환
    - 포화(Saturating) : 표현할 수 있는 값 중에서 수학적으로 옳은 결과에 가장 가까운 값을 반환
    - 넘침(Overflowing) : (`result`, `overflow`) 튜플을 반환
      - `result` : 함수의 순환 버전이 반환하는 값
      - `overflow` : 오버플로 발생 여부를 나타내는 `bool` 값

고정된 크기를 갖는 수치 타입

- 부동소수점 타입
  - `f32` : IEEE 단정밀도 (= `float`), `f64` : IEEE 배정밀도 (= `double`)
  - 특별한 값들은 상수로 정의해 두고 있다.
  - `std::f32::const`와 `std::f64::const` 모듈에 다양한 수학 상수가 정의되어 있다.

`bool` 타입

- 두 가지 값 : `true`, `false`
- C/C++의 경우 문자, 정수, 부동소숫점 수, 포인터를 `bool` 값으로 암시적 변환하므로 `if`나 `while`의 조건으로 쓸 수 있지만, Rust는 암시적인 타입 변환이 일어나지 않고 아주 엄격해서 `bool` 표현식만 가능하다.

문자

- Rust의 문자 타입 char는 유니코드 문자 하나를 32비트 값으로 표현한다. (즉, 4바이트다.)
- 원한다면 문자의 유니코드 코드 포인트를 16진법으로 쓸 수도 있다.
  - 문자의 코드 포인트가 U+0000에서 U+007F 사이에 있다면, '\xHH'와 같은 식으로 쓸 수 있다.
  - 유니코드 문자는 전부 '\u{HHHHHH}'와 같은 식으로 쓸 수 있다.
- as 변환 연산자를 쓰면 char를 정수 타입으로 변환할 수 있는데,
  이때 타입의 크기가 32비트보다 작으면 문자 값의 상위 비트들이 잘려 나간다.
- 반면 as 연산자를 써서 char로 변환할 수 있는 타입은 u8 뿐이다.
- 표준 라이브러리는 문자와 관련해 여러 가지 유용한 메소드들을 제공한다.

튜플

- 다양한 타입의 값들을 한데 모아 묶은 것
- 튜플 값 `t`가 있을 때 안에 있는 요소는 `t.0`, `t.1`과 같은 식으로 접근할 수 있다.
- Rust에서는 함수가 여러 개의 값을 반환할 때 주로 튜플 타입을 쓴다.
- 자주 쓰이는 또 다른 튜플 타입으로 제로 튜플인 `()`가 있다.
  가질 수 있는 값이 하나뿐이라서 통상 유닛 타입(Unit Type)이라고 부른다.

포인터 타입

- 레퍼런스
  - Rust의 기본 포인터 타입이라고 생각하고 들어가면 이해하기 쉽다.
  - 표현식 &x는 x의 레퍼런스를 생성하는데, Rust에서는 이를 두고 x의 레퍼런스를 빌려 온다고 표현한다.
    반대로 레퍼런스 r이 있을 때 표현식 *r은 r이 가리키는 값을 나타낸다.
  - Rust 레퍼런스의 종류에는 2가지가 있다.
    - `&T` : 변경할 수 없는 공유된 레퍼런스. 같은 값을 찾모하는 공유된 레퍼런스는 동시에 여러 개 있을 수 있지만,
      모두 읽기 전용이라서 C의 `const T*`처럼 가리키는 값을 수정하는 데 쓸 수는 없다.
    - `&mut T` : 변경할 수 있는 배타적인 레퍼런스. C의 `T*`처럼 가리키는 값을 읽고 수정할 수 있지만,
      이 레퍼런스가 존재하는 동안은 같은 값을 참조하는 다른 레퍼런스를 가질 수 없다.
  - Rust는 공유된 레퍼런스와 변경할 수 있는 레퍼런스라는 이 양분된 개념을 가지고
    “싱글 라이터 또는 멀티플 리터(Single Writer or Multiple Reader)” 규칙을 구현한다.
    이 규칙에 의하면 값을 읽고 쓸 수 있는 상태가 되거나 여러 리더끼리 공유할 수 있는 상태가 되거나
    둘 중 하나만 선택해야 하고, 절대로 이 두 상태를 동시에 가져갈 수는 없다.

포인터 타입

- 박스
  - 힙에 값을 할당하는 가장 간단한 방법이다.
  - 박스를 통해 힙에 할당된 메모리는 범위를 벗어나는 즉시 해제된다.
    단, 반환되거나 하는 등의 이유로 이미 이동된 경우는 예외다.
- 원시 포인터
  - Rust는 원시 포인터(Raw Pointer)인 `*mut T`와 `*const T`도 가지고 있다.
  - 원시 포인터가 가리키는 대상을 추적하지 않으므로 쓰는 건 안전하지 않다.
  - 원시 포인터는 `unsafe` 블록 안에서만 참조할 수 있다.
    `unsafe` 블록은 개발자 스스로가 안전성을 담보한다는 전제하에
    언어의 고급 기능을 쓸 수 있게 해주는 Rust의 옵트인(Opt-in) 매커니즘이다.

배열, 벡터, 슬라이스

- Rust에는 일련의 값을 메모리에 표현할 때 쓸 수 있는 타입이 3가지 있다.
  - `[T; N]` : T 타입의 값을 N개 갖는 배열을 표현한다. 배열의 크기는 컴파일 시점에 결정되는 상수로 타입의 일부다.
    따라서 배열에 새 요소를 추가하거나 배열의 크기를 축소할 수 없다.
  - `Vec<T>` : 동적으로 할당되고 확장될 수 있는 일련의 T 타입 값으로 벡터라고 부른다.
    벡터는 요소를 전부 힙에 저장하기 때문에 새 요소를 추가하거나, 다른 벡터를 가져와 넣거나,
    기존 요소를 삭제하는 등 크기 변경이 뒤따르는 작업을 수행할 수 있다.
  - `&[T]`, `&mut [T]` : 배열이나 벡터 같은 값의 일부분을 가리키는 레퍼런스로, 각각 공유된 T 슬라이스와
    변경할 수 있는 T 슬라이스라고 부른다. 슬라이스는 참조 대상의 첫 번째 요소를 가리키는 포인터와
    그 위치부터 접근할 수 있는 요소의 개수를 합친 것이라고 보면 된다.
  - 인덱스로 접근할 때 타입은 반드시 `usize` 값이어야 하고, 범위 바깥에 접근하면 패닉에 빠진다.

배열, 벡터, 슬라이스

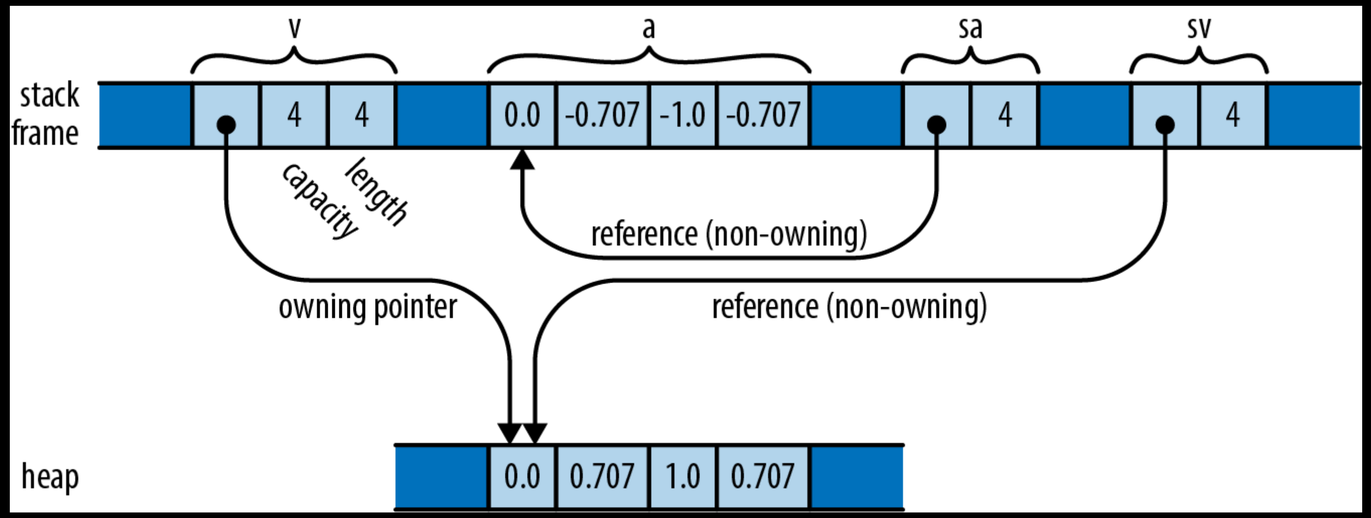
- 배열
  - 배열 값을 작성하는 방법은 여러 가지가 있다.
    - 일련의 값을 대괄호 안에 적어 넣는 방법
    - 각 요소가 가져야 하는 값과 길이를 대괄호 안에 적어 넣는 방법
  - 요소의 반복 처리, 검색, 정렬, 채움, 필터링 등 배열을 다룰 때 쓰이는 유용한 메소드들은
    사실 배열이 아니라 슬라이스에서 호출하는 거다.
    하지만 Rust는 메소드를 찾을 때 암묵적으로 배열 레퍼런스를 슬라이스로 바꾸기 떄문에 바로 호출할 수 있다.

배열, 벡터, 슬라이스

- 벡터
  - 크기 조절이 가능한 T 타입 배열로 요소는 모두 힙에 할당된다.
  - 벡터를 만드는 방법 또한 여러 가지다.
    - 가장 간단한 방법은 `vec!` 매크로를 쓰는 것이다.
    - 이터레이터가 넘겨주는 값을 가지고 벡터를 만들 수도 있다.
  - `Vec<T>`는 세 가지 값으로 구성되어 있다.
    - 요소들을 담아 두기 위해서 힙에 할당해 소유하는 버퍼의 포인터
    - 버퍼가 저장할 수 있는 요소의 개수를 뜻하는 용량 (Capacity)
    - 버퍼가 지금 실제로 가지고 있는 요소의 개수를 뜻하는 길이 (Len)
  - 만약 벡터에 담아야 할 요소의 개수를 미리 알고 있다면, `Vec::new` 대신 `Vec::with_capacity`를 호출해서 애초에 모든 요소를 저장할 수 있을 만큼 큰 버퍼를 가진 벡터를 만드는 게 좋다.

배열, 벡터, 슬라이스

- 슬라이스
  - 배열이나 벡터의 한 영역으로 별도의 길이 지정 없이 [T]라고 쓴다.
  - 팻 포인터(Fat Pointer)로 슬라이스의 첫 번째 요소를 가리키는 포인터와 그 안에 있는 요소의 개수로 구성된다.

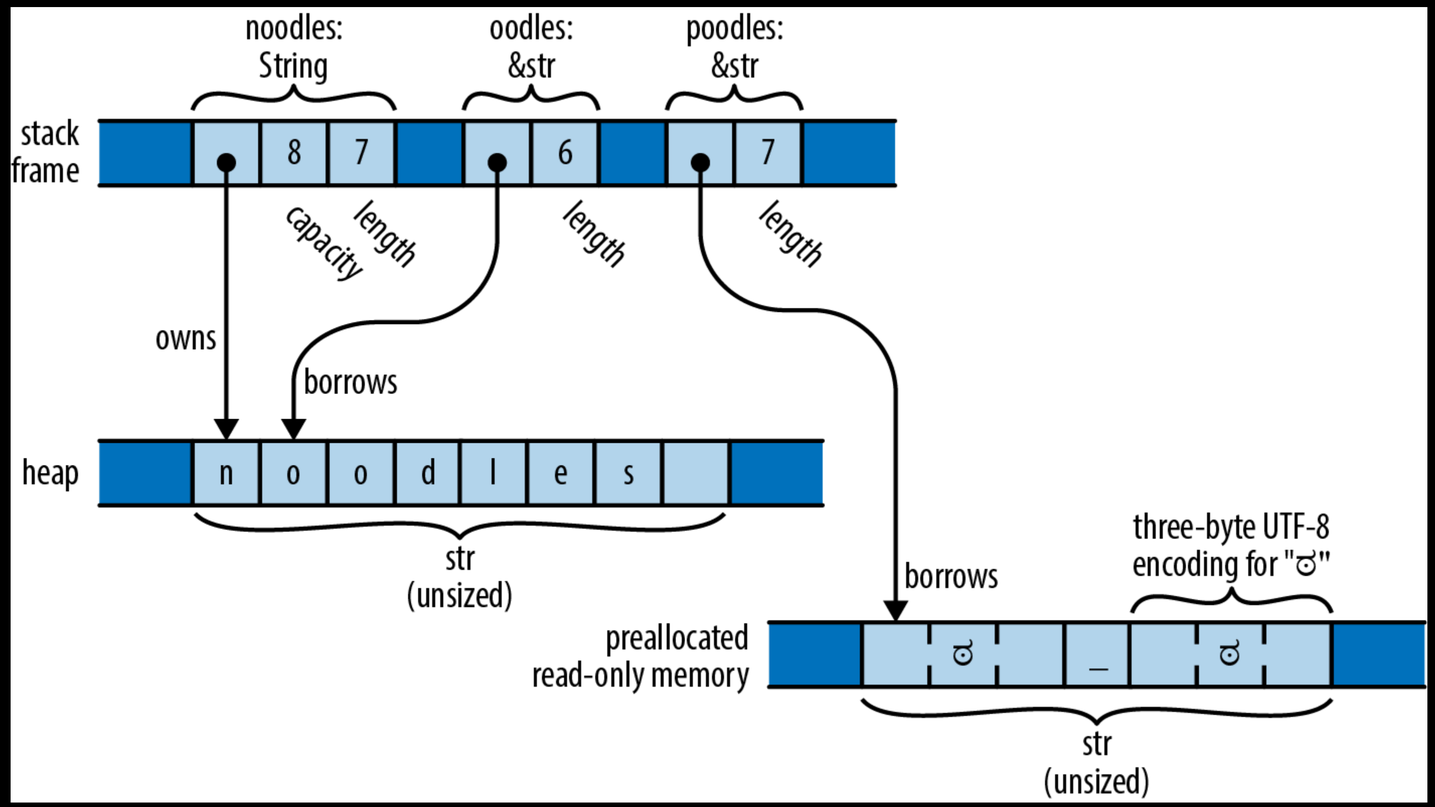


문자열 타입

- 문자열 리터럴
  - 앞뒤로 큰따옴표가 붙는다.
  - 여러 줄에 걸쳐 쓸 수 있다. 이때 새 줄(New Line) 문자와 다음 줄의 선행 공백이 모두 포함된다.
    만약 새 줄 문자와 다음 줄의 선행 공백을 삭제하고 싶다면 문자열 줄의 마지막에 백슬래시를 추가하면 된다.
  - 원시 문자열(Raw String)을 제공한다. 원시 문자열은 맨 앞에 소문자 `r`이 붙는다.

문자열 타입

- 바이트 문자열
  - 유니코드 문자들이 차례로 배열된 것이지만, 그렇다고 해서 메모리에 char 배열로 저장되는 건 아니다.
    그게 아니라 가변 폭 인코딩인 UTF-8을 써서 저장된다.



문자열 타입

- `String`
  - `&str`은 어떤 데이터를 가리키는 팻 포인터라는 점에서 `&[T]`와 유사하다. `String`은 `Vec<T>`와 유사하다.

	Vec<T>	String
Automatically frees buffers	Yes	Yes
Growable	Yes	Yes
`::new()` and `::with_capacity()` type-associated functions	Yes	Yes
`.reserve()` and `.capacity()` methods	Yes	Yes
`.push()` and `.pop()` methods	Yes	Yes
Range syntax `v[start..stop]`	Yes, returns `&[T]`	Yes, returns `&str`
Automatic conversion	`&Vec<T>` to `&[T]`	`&String` to `&str`
Inherits methods	From `&[T]`	From `&str`

문자열 타입

- `String`
  - 다양한 방법으로 만들 수 있다.
    - `.to_string()` 메소드는 `&str`을 `String`으로 변환한다. 이 변환 과정은 문자열 복사를 수반한다.
      `.to_owned()` 메소드는 하는 일과 사용법이 같지만, 다른 타입에도 쓸 수 있다는 차이가 있다.
    - `format!()` 매크로는 `println!()`과 하는 일이 같지만, 텍스트를 `stdout`에 기록하지 않고
      새 `String`에 담아 봔한다는 점과 끝에 새 줄 문자를 자동으로 넣지 않는다는 점이 다르다.
    - 문자열의 배열, 슬라이스, 벡터는 여러 문자열로부터 새 `String`을 만들어 내는
      두 메소드 `.concat()`과 `.join(sep)`를 갖는다.

문자열 타입

- 유사 문자열 타입
  - 프로그램을 만들다 보면 유효한 유니코드가 아닌 문자열을 다뤄야 하는 경우도 더러 있다.
  - Rust는 이런 상황에 쓸 수 있는 유사 문자열 타입 몇 가지를 제공한다.
    - 유니코드 텍스트에는 `String`과 `&str`을 쓴다.
    - 파일 이름을 다룰 때는 `std::path::PathBuf`와 `&Path`를 쓴다.
    - UTF-8로 인코딩되지 않은 바이너리 데이터를 다룰 때는 `Vec<u8>`과 `&[u8]`을 쓴다.
    - 환경 변수 이름과 명령줄 인수를 운영체제의 고유 타입으로 다룰 때는 `OsString`과 `&OsStr`를 쓴다.
    - 널 종료 문자열을 쓰는 C 라이브러리와 연동할 때는 `std::ffi::CString`와 `&CStr`를 쓴다.

타입 별칭

- `type` 키워드는 C++의 `typedef`처럼 기존 타입을 위한 새 이름을 선언하는 데 쓸 수 있다.

감사합니다.

utilForever@gmail.com

https://github.com/utilForever

Facebook, Twitter: @utilForever