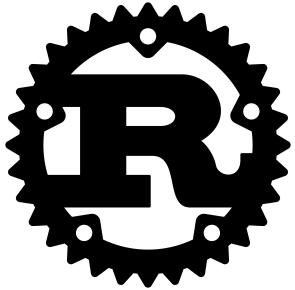

TinTin

第五课：生命期简介、Option、Result与错误处理、宏简介

Mike Tang

daogangtang@gmail.com

2023-5-31

lifetime 与 scope

Lifetimes are named regions of code that a reference must be valid for. Those regions may be fairly complex, as they correspond to paths of execution in the program. While lifetimes and scopes are often referred to together, they are not the same.

https://doc.rust-lang.org/nomicon/lifetimes.html

- 因为Rust从所有权出发，发展了引用及其相关规则。光有scope不够用
- lifetime - 更精细和更紧凑的分析
- 主要是对引用进行分析，只有引用情况才比较复杂
- 对于资源所有权变量，这两个概念往往混用

示例

https://play.rust-lang.org/?version=stable&mode=debug&edition=2021&gist=dbb3c80ba388266f5f7fc264a11b7151

生命周期标注 ‘a

生命周期 lifetime 标注是Rust语言在引用上做的一种标注。用于指代一段最小代码区域，对于一个正确的Rust程序，被标注的那个引用在这片代码区域中必须有效。‘a实际是对这片代码区域的命名。

‘a就是Rust采用的对lifetime作分析的形式化方法。

我们所指的lifetime，一般就是指endtime，也即引用或资源结束的时刻。因为我们一般不需要关心lifetime开始的时刻：只要产生了引用，那么在那之前资源的lifetime一定开始了。

注意：只在引用上做标注。

从一个引用参数的函数中返回引用

情况1:

https://play.rust-lang.org/?version=stable&mode=debug&edition=2021&gist=12e02f087def6453a08e15ee81392bf0

从一个引用参数的函数中返回引用

情况2：

https://play.rust-lang.org/?version=stable&mode=debug&edition=2021&gist=9185806d3c59a8d9746332c94f488240

https://play.rust-lang.org/?version=stable&mode=debug&edition=2021&gist=2c830ef199d0059fc6db3e62db489564

从一个引用参数的函数中返回引用

情况3：

https://play.rust-lang.org/?version=stable&mode=debug&edition=2021&gist=c034bb6f97cb1ec78f1d77beedac5a59

资源先于返回的引用释放的情况

https://play.rust-lang.org/?version=stable&mode=debug&edition=2021&gist=a59d1d7d47e95900f6112d77360953e0

TinTin

空值

空值，是任何一个程序语言中都会出现的概念，可以说到处都是。空字符串，空数组，空哈稀，对象刚创建还没初始化时可能的值 nil, null, NULL, 0, false等等。有些语言甚至不仔细分辨这些类型。

10亿美元错误

Tony Hoare 说引入Null引用是一个10亿美元的错误。

In his 2009 presentation “Null References: The Billion Dollar Mistake,” Tony Hoare, the inventor of null, has this to say:

> I call it my billion-dollar mistake. At that time, I was designing the first comprehensive type system for references in an object-oriented language. My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn’t resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.

```
> typeof NaN
"number"
> 9999999999999999
10000000000000000
> 0.5+0.1==0.6
true
> 0.1+0.2==0.3
false
> Math.max()
-Infinity
> Math.min()
Infinity
> []+[]
""
> []+{}
"[object Object]"
> {}+[]
0
> true+true+true===3
true
> true-true
0
```

```
> true==1
true
> true===1
false
> (!+[]+[]+![]).length
9
> 9+"1"
"91"
> 91-"1"
90
> []==0
true
```

Option<T>

Rust中丢弃了Null值，变量声明时必须初始化。使用这样一个枚举Option来代替。来自函数式语言。

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

我们看到，它带有一个泛型参数。正如空值在程序语言中无处不在，Option在Rust中也是无处不在。

Option<T> 相当于把原来的类型T包裹了一层。将空概念从原来的类型值隔离开来，单独建立了一个维度。（可以看成一个正交的二维图像）

let a: Option<u32> = None;

Option的匹配

https://play.rust-lang.org/?version=stable&mode=debug&edition=2021&gist=73fd3f066f5bbdbce4aae9245cd52450

Option的解包

expect()

unwrap()

unwrap_or()

https://doc.rust-lang.org/std/option/enum.Option.html

Result<T, E>

TinTin

错误分两大类：可恢复的和不可恢复的。

可恢复的，比如：文件没找到

不可恢复的，比如：缓冲区访问溢出

有些语言统一用exception来处理。Rust没有这种异常机制（它本身就是一个奇怪的东西）。而是用Result和panic!分别处理它们。

```
enum Result<T, E> {
    Ok(T),
    Err(E),
}
```

示例

```
use std::fs::File;

fn main() {
    let greeting_file_result = File::open("hello.txt");

    let greeting_file = match greeting_file_result {
        Ok(file) => file,
        Err(error) => panic!("Problem opening the file: {:?}", error),
    };
}
```

传递错误

https://play.rust-lang.org/?version=stable&mode=debug&edition=2021&gist=3dcf92f6f2dcb72b52560d11d0c9ac80

使用？传递错误

https://play.rust-lang.org/?version=stable&mode=debug&edition=2021&gist=8f7126b958104ae8d57cbf0e7641124b

如何正确使用？

函数体内的错误类型要统一。

```
fn fn() -> Result<(), std::io::Error> {
    let greeting_file = File::open("hello.txt")?;
    // let a = "100".parse::<u32>()?;
    Ok(())
}
```

如何正确使用？

一个函数返回值为Result或者Option的时候，能用，但不能混合使用。

https://play.rust-lang.org/?version=stable&mode=debug&edition=2021&gist=242e0bc9304fdfad834bfc12fe9e5967

给main函数添加返回值

```
use std::error::Error;

use std::fs::File;

fn main() -> Result<(), std::io::Error> {
    let greeting_file = File::open("hello.txt")?;
    Ok(())
}
```

宏

宏不神秘。

Rust宏是操作代码的工具。也可以叫元编程。

操作代码的token流

宏是外围设施。不是核心设施。

声明宏

简单的例子

https://play.rust-lang.org/?version=stable&mode=debug&edition=2021&gist=29e45ffc560b4ae60284e68233d5926e

TinTin

同一名字，多种形态

https://play.rust-lang.org/?version=stable&mode=debug&edition=2021&gist=dedfe8daca68a5edba48e9e706f34898

符号

item — an item, like a function, struct, module, etc.

block — a block (i.e. a block of statements and/or an expression, surrounded by braces)

stmt — a statement

pat — a pattern

expr — an expression

ty — a type

ident — an identifier

path — a path (e.g., foo, ::std::mem::replace, transmute::<_, int>, ...)

meta — a meta item; the things that go inside #[...] and #![...] attributes

tt — a single token tree

vis — a possibly empty Visibility qualifier

https://play.rust-lang.org/?version=stable&mode=debug&edition=2021&gist=b7ca974a33a0d332888439159cfca4a4

*和+，$()

https://play.rust-lang.org/?version=stable&mode=debug&edition=2021&gist=6dcf38a5081f69de94029f807ae2d0c5

cargo expand

cargo expand

过程宏

宏的实现我们课上不会细讲，首先要认识过程宏。

- 属性宏
- 派生宏
- 类函数宏

虽然属于中高阶内容，但是实际概念理解上并不难，需要一点编译原理的知识。多加练习就好了。

属性宏

```
#[some_attribute_macro(some_argument)]
fn perform_task(){
// some code
}
```

代码顶部那些

```
#[allow(dead_code)]

#![allow(dead_code)]
```

TinTin

derive宏

```
#[derive(Trait)]

struct MyStruct{}
```

TinTin

类函数宏

```
fn foo() {
    tlborm_attribute!(be quick; time is mana);
    let sql = sql!(SELECT * FROM posts WHERE id=1);
}
```

TinTin

示例：利用宏实现DSL

四则计算器

https://play.rust-lang.org/?version=stable&mode=debug&edition=2021&gist=3dd1285d3d9da81a030ea7b8b4a41273

实现一个类CSS语言

https://developerlife.com/2022/08/04/rust-dsl-part-1/

实现一个lisp

https://github.com/JunSuzukiJapan/macro-lisp

附录

一些扩展阅读链接：

https://developerlife.com/2022/03/30/rust-proc-macro/

https://blog.cloudflare.com/writing-complex-macros-in-rust-reverse-polish-notation/

https://betterprogramming.pub/develop-a-rust-macro-to-automatically-write-sql-boilerplate-code-60c25d86adcb

Q&A

TinTin