从C++到Rust，异常处理panic ，下集

原创 Ajonbin AJonbin的杂货铺 2024年01月09日 21:09 美国

上集讲了用除零来触发一个panic，今天继续。

除了程序出错意外，你可以手动触发一个panic，这样你就知道程序出现了一个不该出现的情况。类似于assert的效果。

panic!宏就是用来干这个的。结尾的就是Rust中宏的标记，以后再慢慢聊。就像println!一样，它以!结尾，它也是个宏。

我们在上集的程序了里在添加一个bin，来试试panic!宏。

```
[package]
name = "hello_panic"
version = "0.1.0"
edition = "2021"

[dependencies]

[[bin]]
name = "panic_macro"
path = "src/panic_macro.rs"
```


先复习下前面内容，我们在Cargo.toml里增加一个bin crate，panic_macro。注意这里是[[bin]]，因为同lib段不同，bin段是可以有多个的。

然后增加文件src/panic_macro.rs。

```
1  fn trigger_panic_by_macro(){
2    panic!("Something wrong, trigger a panic by macro");
3  }
4
5  fn main(){
6    let ret = std::panic::catch_unwind(||trigger_panic_by_macro());
7
8    println!("Is trigger_panic_by_macro paniced? {}", !ret.is_ok());
9
10   println!("=====> Bye Panic Macro");
11 }
```

1 2 3 4

我们先运行下，看看结果，再来解释代码。因为我们有两个bin，运行时要加上--bin来指定panic_macro。

```
hello_panic$cargo run --bin panic_macro
   Compiling hello_panic v0.1.0
    Finished dev [unoptimized + debuginfo] target(s) in 0.85s
     Running `target/debug/panic_macro`
thread 'main' panicked at src/panic_macro.rs:2:3:
Something wrong, trigger a panic by macro
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
Is trigger_panic_by_macro paniced? true
=====> Bye Panic Macro
```


在panic_macro.rs的标记1处，我们调用了panic!宏，并给出了错误信息。在运行结果中，我们可以看到对应的panic被触发，对应的消息被打印出来。

在第6行中，我们调用了catch_unwind在判断并截获panic。在标记2处，我们将catch_unwind的结果赋值给变量ret，这是一个Result<>类型。关于Result我们稍后会详细讲。目前只要知道Result一个Ok或Err的Enum类型。

第6行，标记3处，| | trigger_panic_byMacro() 是Rust中的closure，它是一个可以被调用的函数类型，类似于C++中的lambda函数。也是下次详细讲。

第8行，标记4处，通过Result::is_ok()来判断ret是Ok还是Err。如果是不是Ok则说明通过catch_unwind调用的函数里发生了panic。

panic， No Panic!

修改于2024年01月09日