从C++到Rust，异常处理panic ，上集

原创 Ajonbin AJonbin的杂货铺 2024年01月08日 22:47 美国

C++通过try-catch来捕获和处理异常。Rust的异常处理机制有两种：panic和Result类型。

Result是Rust中的enum类型，其中定义了Ok和Err两个可能的值。如果有异常发生，Reulst就会设置Err，来表明异常错误的类型和信息。

Result一般是程序外部错误，是可以预见，有可能发生的错误，比如IO错误，网络连接错误。通过对不同类型错误的处理，来保证程序继续运行。

但今天不讲Result，先放一放。

今天讲讲另一种异常的类型，panic。

和Result不同，panic一般是程序内部，不应该发生的错误。比如数组越界，除0等。

发生这种错误往往是程序无法按照正常的逻辑运行下去，所以一旦发生panic，线程就会最终退出。

从概念上讲，panic有两种处理方式**展开堆栈(Unwinding)**和**退出程序(Aborting)**。

Unwinding the stack展开堆栈是默认的处理方式。

Aborting，直接退出程序，简单粗暴。

让我们来写点代码试试。

新建个工程，再添加一个除零的错误。

```
$ cargo new hello_panic
$ cd hello_panic/
```

```
hello_panic$ vim src/main.rs

1
2 fn divide(divider:u32){
3   let _ = 100/divider;
4 }
5
6 fn will_divide_zero(){
7     let _ = divide(0);
8 }
9
10 fn caller_divide(){
11     will_divide_zero();
12 }
13
14 fn main() {
15     println!("Hello, panic!");
16
17     caller_divide();
18
19 }
```


在main.rs里，我们设计了三个函数，依次调用，最终用100除以0。

运行cargo run 运行一下

程序如约在第3行除0的地方产生了错误，最终退出了程序。

通过提示， 如果设置了RUST_BACKTRACE=1，那么堆栈就会被打印出来。来试试。

panic的默认处理方式Unwinding展开堆栈，可以理解为清理栈空间，它包括了这些步骤：

- 清理当前函数的栈空间，释放所有临时变量，本地变量和函数入参。
- 当前函数清理完成后，会沿着调用栈向上逐步清理调用函数。
- 如果遇到catch_unwind()，则当前清理工作结束。否则清理工作会持续到当前线程thread结束。
- 如果当前线程是主线程，那么程序就会退出了。

就向我们刚才的例子，只有一个主线程，一旦发生panic，就会导致整个程序退出。

刚刚提到，用std::panic::catch_unwind()可以防止线程退出，再来试试

我们修改下main.rs，在调用will_divide_zero时加上catch_unwind()保护。然后我们在最后加一句打印"After panic"。

可以看到运行之后，panic依然发生了，但是程序并没有马上退出，而是正常运行到了最后。

catch_unwind()和unwinding提供了一种优雅地处理错误异常的方式。它可以让程序在遇到panic时可以有机会不退出整个程序。

接着来看看Aborting的情形。

我们需要在Cargo.toml增加一个[profile]段，并通过panic="abort"修改panic默认的处理方式。

需要注意，之前的catch_unwind()的代码并没有删除。你看一旦我们指定abort作为panic的处理方式后，程序及就直接退出了。

最后的一句打印表明，程序收到了SIGABRT信号，结束了自己的生命。