从C++到Rust，错误处理Result，第三集

原创 Ajonbin AJonbin的杂货铺 2024年01月12日 22:30 上海

上两集讲了Result创建和处理方法。

用match来匹配Result这样Enum类型是一种标准的方法。但是每次要写大段代码来匹配不同Enum值，有些累赘了。

而且大部分时候，我们不太关心怎么处理不同的错误，有些意料之外的错误可以直接退出，或者交给”其他函数“处理。

为此，Result提供了一些方法，可以简化我们的代码。

- Result.expect(message)
- Result.unwrap()
- ?运算符

这三种Result的处理方式，是大量存在于各个Rust项目中的。写Rust程序，不可能调用这些函数。

不理解这些函数的行为，你看别人代码的时候就会感觉云里雾里。

先来看看expect(message)和unwrap()

这两个函数功能基本相同，如果result是一个成功Ok()的类型，那么就返回Ok()包含的实际值；如果result是一个Err()类型，那么就会直接触发一个panic。

区别是调用expect(message)时，可以设置一个消息变量，当是Err()时，在触发panic的同时，也会打印出这个消息变量，帮助调试程序。

来看个栗子

我们现在之前的hello_result工程里再添加一个bin crate。

```
[lib]
name = "libdivide"
path = "src/lib/lib.rs"

[[bin]]
name = "my_result"
path = "src/my_result.rs"

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


在method_of_result.rs里，我们调用File::open()打开一个不存在的文件。这样应该返回一个Err()值。

这里我们不用match来匹配这个错误，而是直接调用Result的unwrap()方法。

```
1 fn main(){
2   let ret = std::fs::File::open("./no_such_file").unwrap();
3   println!("{:?}",ret);
4 }
```


来看看运行结果。

```
hello_result$ cargo run --bin method_of_result
   Compiling hello_result v0.1.0
    Finished dev [unoptimized + debuginfo] target(s) in 0.38s
     Running `target/debug/method_of_result`
thread 'main' panicked at src/method_of_result.rs:3:53:
called `Result::unwrap()` on an `Err` value: Os { code: 2, kind: NotFound, message: "No such file or directory" }
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
```


可以看到程序直接因为panic退出了，而这个panic就是由Err::unwrap()产生的。

有一个不太友好的地方，根据错误信息，你只知道是一个文件不存在错误。如果你的程序中需要打开多个文件的的话，你就不能直观的知道是哪个文件不存在。

expect() comes to help! 我们把unwrap()换成expect(message)

```
1 fn main(){
2   let ret = std::fs::File::open("./no_such_file").expect("Open no_such_file failed");
3   println!("{:?}",ret);
4 }
```


expect(message)比 unwrap()多一个参数message。

```
hello_result$ cargo run --bin method_of_result
   Compiling hello_result v0.1.0
    Finished dev [unoptimized + debuginfo] target(s) in 0.38s
     Running `target/debug/method_of_result`
thread 'main' panicked at src/method_of_result.rs:2:53:
Open no_such_file failed: Os { code: 2, kind: NotFound, message: "No such file or directory" }
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
```


同样，Err::expect(message)也会触发panic，而且会将message消息打印出来，这样在panci的时候，你就可以得到你需要的message信息。

看了Err的情况，我们来看看当Result是Ok()的情况下，会有什么表现。

我们还是调用之前写的divide函数。

```
1 use libdivide::divide;
2
3 fn main(){
4   let ret = divide(10, 2).expect("Divide failed");
5   println!("{}",ret);
6 }
```



```
hello_result$ cargo run --bin method_of_result
   Compiling hello_result v0.1.0
    Finished dev [unoptimized + debuginfo] target(s) in 0.38s
     Running `target/debug/method_of_result`
5
```


可以看到，当Result是Ok(v)时，expect()会直接取出Ok() 中实际的成功值，并赋值给变量ret，这里ret的类型不是Result<f32, std::io::Error>，而就是Result<T,E>中的T类型，这里就是f32。

再来看看?运算符

有时候，我们知道某些函数会产生错误，但是我们并不关心，也不想处理，这个时候，我们可以简单的向外层透传这个错误，让上层来处理这个错误。我们只关心正确结果的处理。

这个和C++中try但不catch的情况类似，会不断的向外抛出异常，直到被捕获处理。

这就需要用到?操作符。使用的时候，只需要在目标函数调用之后加上？即可。比如std::io::File::open("")?

当目标函数返回Ok()时，?运算符会把Ok()中的值返回；

当目标函数返回Err()时，?运算符会立刻返回整个Err()，注意不是Err()里包含的错误值。

由于?运算符在错误时，会返回Err()，这个一个Result<>类型，所以调用?运算符的函数的返回值声明为Result<>。

先看代码，对于怎么使用?运算符有个整体概念。

```
1 fn wrap_fopen(file_path: &str) -> Result<std::fs::File, std::io::Error>{
2   let f = std::fs::File::open(file_path)?;
3   Ok(f)
4 }
5
6 fn main()->Result<(), std::io::Error>{
7   let f_ok = wrap_fopen("./Cargo.toml")?;
8   println!("{:?}", f_ok);
9
10   let f_err = wrap_fopen("./no_such_file")?;
11   println!("{:?}", f_err);
12   println!("exit...");
13   Ok(())
14 }
```


第1-4行，我们定义了一个函数wrap_fopen()包装下File::open()。

在第2行，调用File::open()之后马上调用了?运算符。

注意函数的返回值是Result<std::fs::File, std::io::Error>。

之前说过，由于?运算符可能会返回一个Result::Err()，所以wrap_fopen()也必须声明会返回一个Err(e)。并且这个e的类型需要和函数内部可能发生的错误类型一致。这里e的类型是std::io::Error。

第3行，我们构建了一个Ok()，用来返回成功时的File对象。注意，这一行结尾没有分号“;”，表明这就是函数的返回值。

在Rust中，没有“;”的表达式就会被当作函数的返回值，这里相当于return Ok(f);

第6行，我们还可以在main()也加上返回Result<T,E>，这样我们在代码中就可以一路使用?运算符，而不处理Err(e)。这在写一些简单的测试程序是什么有用的。

这样，一旦File::open()产生Err(e)，就会由第2行的?运算符向上传递到wrap_fopen()，再由第10行的?运算符传递到main函数，最终导致程序退出。但不会产生panic。

第6行中，main()函数返回的Result<T,E>中，成功情况下返回的类型是()。这也是一个特殊的类型，空类型，啥类型也不是。

第13行，我们就用Ok(())构建了一个成功的返回值()。注意这里()的数量和不同含义。

我们来看看运行结果。

```
hello_result$ cargo run --bin method_of_result
   Compiling hello_result v0.1.0
    Finished dev [unoptimized + debuginfo] target(s) in 0.38s
     Running `target/debug/method_of_result`
File { fd: 3, path: "/Users/haimhuan/RustLab/hello_result/Cargo.toml", read: true, write: false }
Error: Os { code: 2, kind: NotFound, message: "No such file or directory" }
```


可以看到，最终main函数以为Err()而退出了，但是这样并不会产生panic。

总结下，Result<T,E>是一个枚举Enum类型。

他可以有两种值Ok(r) 和Err(e)。

r是成功时真正的值，类型时T。

e是错误时的错误变量，类型时E。

标准的做法是通过match关键字来匹配Ok()和Err()。

为了方便，Result有一些函数来快速处理Err。

记住expect / unwrap / ?运算符。