第五课:如何编写测试 - 2

Mock、Fuzz

苏林

回顾上一次公开课的内容

今天公开课内容

1、通过实例学习Mock

2、通过实例学习Fuzzing — https://rust-fuzz.github.io/book/cargo-fuzz/tutorial.html

什么是Mock、为什么需要Mock

1、加快开发. -> 项目. 某个功能(2名后端研发投入), 2名分工 A 开发2个接口. B开发2个接口. B依赖于A的接口.

A还没有开发完, 但是 提前把trait都已经定义好. trait的实现还没有.

B 就对A的trait进行mock.

2、保证代码都具有测试的能力.

开发一块复杂的功能(拆分成4个方法), 复杂的功能就用在一个方法里面， 最后这个方法写了100行, 可读差.

每个方法，实现单一功能.

入门

Mockall有两种使用方式。最简便的是直接使用 #[automock\]。这种方式可以模拟大多数的特征，或只有一个impl代码块的结构体。其他的情况则可以使用 mock!。

无论使用以上哪种方式，基本的概念大致相同：

创建一个模拟的结构体。命名在原有结构体之前加上“Mock”字样。

在你的测试中，通过模拟出的结构体自带的new或者default进行实例化。

为模拟出的机构体设置期望。每一个期望可以提供所需的参数匹配器、预期的被调用次数、以及在被调用的序列中固定的位置。每一个期望还必需有一个期待的返回值。

在测试时，为被测试的代码提供模拟的结构体。这个模拟结构体会返回事先设置好的返回值。任何违反预期的访问都将会引起panic。

入门

Mockall有两种使用方式。最简便的是直接使用 #[automock\]。这种方式可以模拟大多数的特征，或只有一个impl代码块的结构体。其他的情况则可以使用 mock!。

无论使用以上哪种方式，基本的概念大致相同：

创建一个模拟的结构体。命名在原有结构体之前加上“Mock”字样。

在你的测试中，通过模拟出的结构体自带的new或者default进行实例化。

为模拟出的机构体设置期望。每一个期望可以提供所需的参数匹配器、预期的被调用次数、以及在被调用的序列中固定的位置。每一个期望还必需有一个期待的返回值。

在测试时，为被测试的代码提供模拟的结构体。这个模拟结构体会返回事先设置好的返回值。任何违反预期的访问都将会引起panic。

```
use mockall::*;
use mockall::predicate::*;
#[automock]
trait MyTrait {
    fn foo(&self, x: u32) -> u32;
}

fn call_with_four(x: &MyTrait) -> u32 {
    x.foo(4)
}

let mut mock = MockMyTrait::new();
mock.expect_foo()
    .with(predicate::eq(4))
    .times(1)
    .returning(|x| x + 1);
assert_eq!(5, call_with_four(&mock));
```

什么是Fuzz、为什么需要Fuzz

模糊测试 用于通过提供 随机数据 作为输入 来发现安全性和稳定性问题

123 -> 456

比如开发一个功能. 解析字符. “aaaa”, “bbbb”

随机数据(字符串) 作为输入

Cargo-fuzz 实例

```
#![no_main]
#[macro_use] extern crate libfuzzer_sys;
extern crate url;

fuzz_target!(|data: &[u8]| {
    if let Ok(s) = std::str::from_utf8(data) {
        let _ = url::Url::parse(s);
    }
});
```

afl.rs 实例

```rust
#[macro_use]
extern crate afl;
extern crate url;

fn main() {
    fuzz!(|data: &[u8]| {
        if let Ok(s) = std::str::from_utf8(data) {
            let _ = url::Url::parse(&s);
        }
    });
}
```

QA环节

加群一起交流Rust & Databend

Databend

Rust语言中文社区

知数堂