第二十一讲：异步编程 (Asynchronous Programming)

第 3 节：Generators and async/await

向勇、陈渝

清华大学计算机系

xyong,yuchen@tsinghua.edu.cn

2020 年 5 月 5 日

提纲

Ref:

- Futures Explained in 200 Lines of Rust
- Writing an OS in Rust - Async/Await
- 零成本异步 I/O

Concurrency in Rust

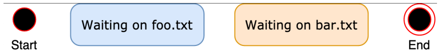
- Stackful coroutines (green threads)
- Using combinators
- Stackless coroutines (generators)

State Machine Transformation in Future

- Async in Rust is implemented using Generators
- Generators in Rust are implemented as state machines
- Compiler transforms the body of the ‘async‘ function into a state machine, with each ‘.await‘ call representing a different state.

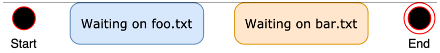
State Machine Transformation in Future

- Async in Rust is implemented using Generators
- Generators in Rust are implemented as state machines
- Compiler transforms the body of the ‘async‘ function into a state machine, with each ‘.await‘ call representing a different state.
- Each state represents a different pause point of the function

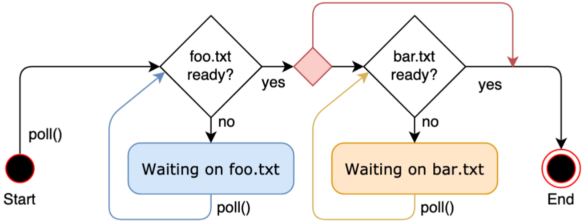


State Machine Transformation in Future

- Async in Rust is implemented using Generators
- Generators in Rust are implemented as state machines
- Compiler transforms the body of the ‘async‘ function into a state machine, with each ‘.await‘ call representing a different state.
- Each state represents a different pause point of the function



- Arrows represent state switches and diamond shapes represent alternative ways



State Machine Type: enum ExampleStateMachine

- Create a state machine and combine them into an ‘enum‘

```
//rust code
enum ExampleStateMachine {
    Start(StartState),
    WaitingOnFooTxt(WaitingOnFooTxtState),
    WaitingOnBarTxt(WaitingOnBarTxtState),
    End(EndState),
}
```

State Machine Type: impl Future for ExampleStateMachine

- Generates an implementation of the state transitions in the ‘poll‘ function

```
    impl Future for ExampleStateMachine {
    type Output = String; // return type of `example`
    fn poll(self: Pin<&mut Self>, cx: &mut Context) -> Poll<Self::Outpu
        loop {
            match self { // TODO: handle pinning
                ExampleStateMachine::Start(state) => {···}
                ExampleStateMachine::WaitingOnFooTxt(state) => {···}
                ExampleStateMachine::WaitingOnBarTxt(state) => {···}
                ExampleStateMachine::End(state) => {···}
            }
        }
    }
}
```

Example of Generator

```
fn main() {
    let a: i32 = 4;
    let mut gen = move || {
        println!("Hello");
        yield a * 2;
        println!("world!");
    };
    if let GeneratorState::Yielded(n) = gen.resume() {
        println!("Got value {}", n);
    }
    if let GeneratorState::Complete(()) = gen.resume() {
        ()
    };
}
```