探讨为什么Pin在Rust异步编程中如此重要

为深入理解Tokio打基础

苏林

Datafuse Rust语言中文社区 知数堂

分享内容

- Async/Await背后是什么?
- 自引用结构体的问题?
- 为什么需要pin以及pin是什么?

回顾一下async/await相应语法

```rust
use futures::future::{self, Future};

fn main() {
    let _ = example(100);
}

async fn example(min_len: usize) -> String {
    let content = async_read_file("foo.txt").await;
    if content.len() < min_len {
        content + &async_read_file("bar.txt").await
    } else {
        content
    }
}

fn async_read_file(name: &str) -> impl Future<Output = String> {
    future::ready(String::from(name))
}

```

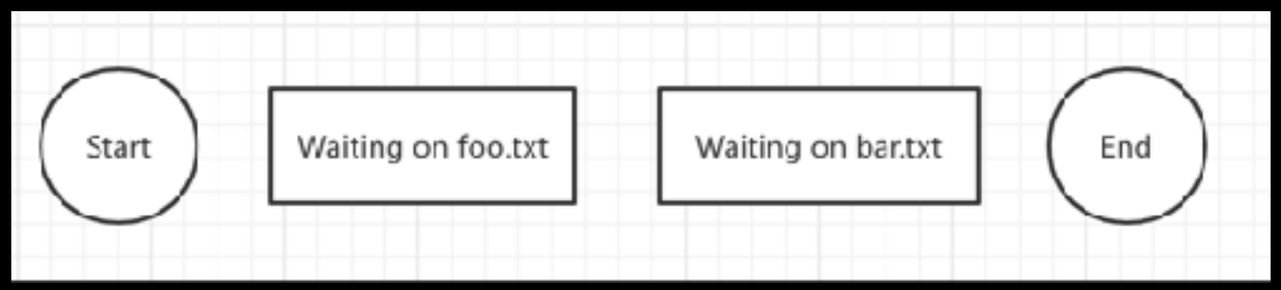
async/await背后是什么？状态机转换

```rust
use futures::future::{self, Future};

fn main() {
    let _ = example(100);
}

async fn example(min_len: usize) -> String {
    let content = async_read_file("foo.txt").await;
    if content.len() < min_len {
        content + &async_read_file("bar.txt").await
    } else {
        content
    }
}

fn async_read_file(name: &str) -> impl Future<Output = String> {
    future::ready(String::from(name))
}
```



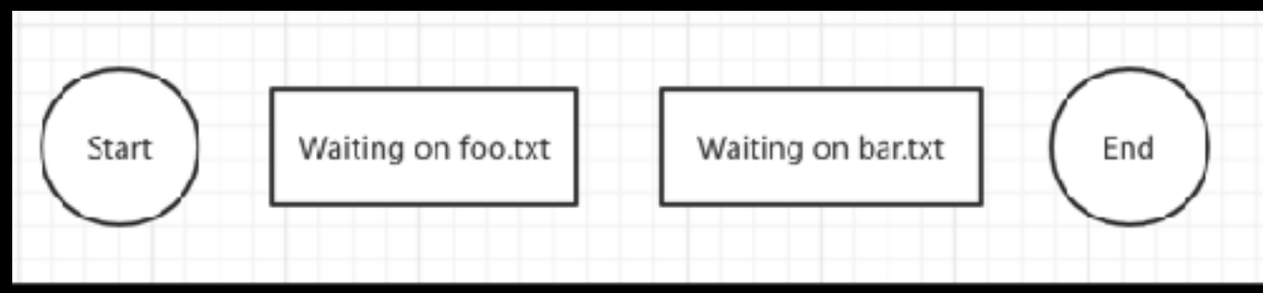
async/await背后是什么？状态机转换

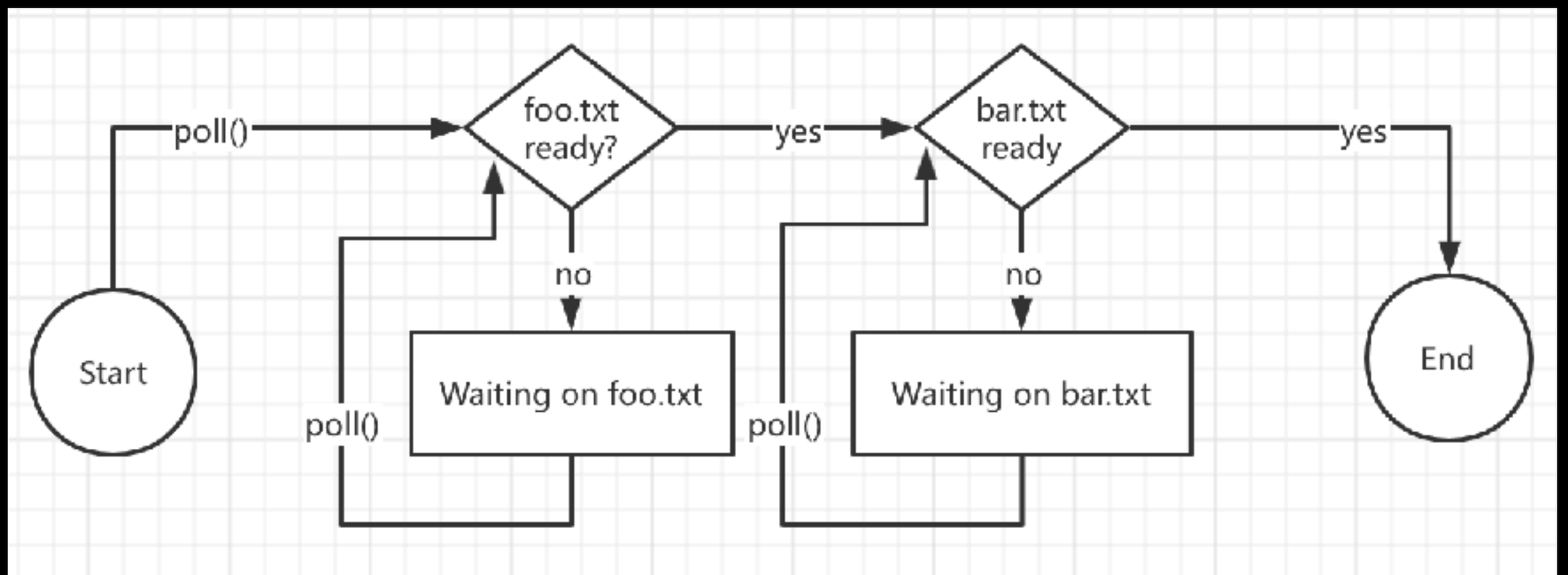
```rust
use futures::future::{self, Future};

fn main() {
    let _ = example(100);
}

async fn example(min_len: usize) -> String {
    let content = async_read_file("foo.txt").await;
    if content.len() < min_len {
        content + &async_read_file("bar.txt").await
    } else {
        content
    }
}

fn async_read_file(name: &str) -> impl Future<Output = String> {
    future::ready(String::from(name))
}
```





async/await背后是什么？状态机转换 保存状态

```rust
async fn example(min_len: usize) -> String {
    let content = async_read_file("foo.txt").await;
    if content.len() < min_len {
        content + &async_read_file("bar.txt").await
    } else {
        content
    }
}
```

```rust
// 编译器生成的状态结构体:

struct StartState {
    min_len: usize,
}

struct WaitingOnFooTxtState {
    min_len: usize,
    foo_txt_future: impl Future<Output = String>,
}

struct WaitingOnBarTxtState {
    content: String,
    bar_txt_future: impl Future<Output = String>,
}

struct EndState {}
```

async/await背后是什么？完整的状态机类型

```rust
async fn example(min_len: usize) -> String {
    let content = async_read_file("foo.txt").await;
    if content.len() < min_len {
        content + &async_read_file("bar.txt").await
    } else {
        content
    }
}
```

```rust
// 编译器生成的状态结构体：

struct StartState {
    min_len: usize,
}

struct WaitingOnFooTxtState {
    min_len: usize,
    foo_txt_future: impl Future<Output = String>,
}

struct WaitingOnBarTxtState {
    content: String,
    bar_txt_future: impl Future<Output = String>,
}

struct EndState {}
```

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

async/await背后是什么？完整的状态机类型

```
impl Future for ExampleStateMachine {
    type Output = String; // return type of `example`

    fn poll(self: Pin<&mut Self>, cx: &mut Context) -> Poll<Self::Output> {
        loop {
            match self { // TODO: handle pinning
                ExampleStateMachine::Start(state) => {…}
                ExampleStateMachine::WaitingOnFooTxt(state) => {…}
                ExampleStateMachine::WaitingOnBarTxt(state) => {…}
                ExampleStateMachine::End(state) => {…}
            }
        }
    }
}
```

async/await背后是什么? 完整的状态机类型

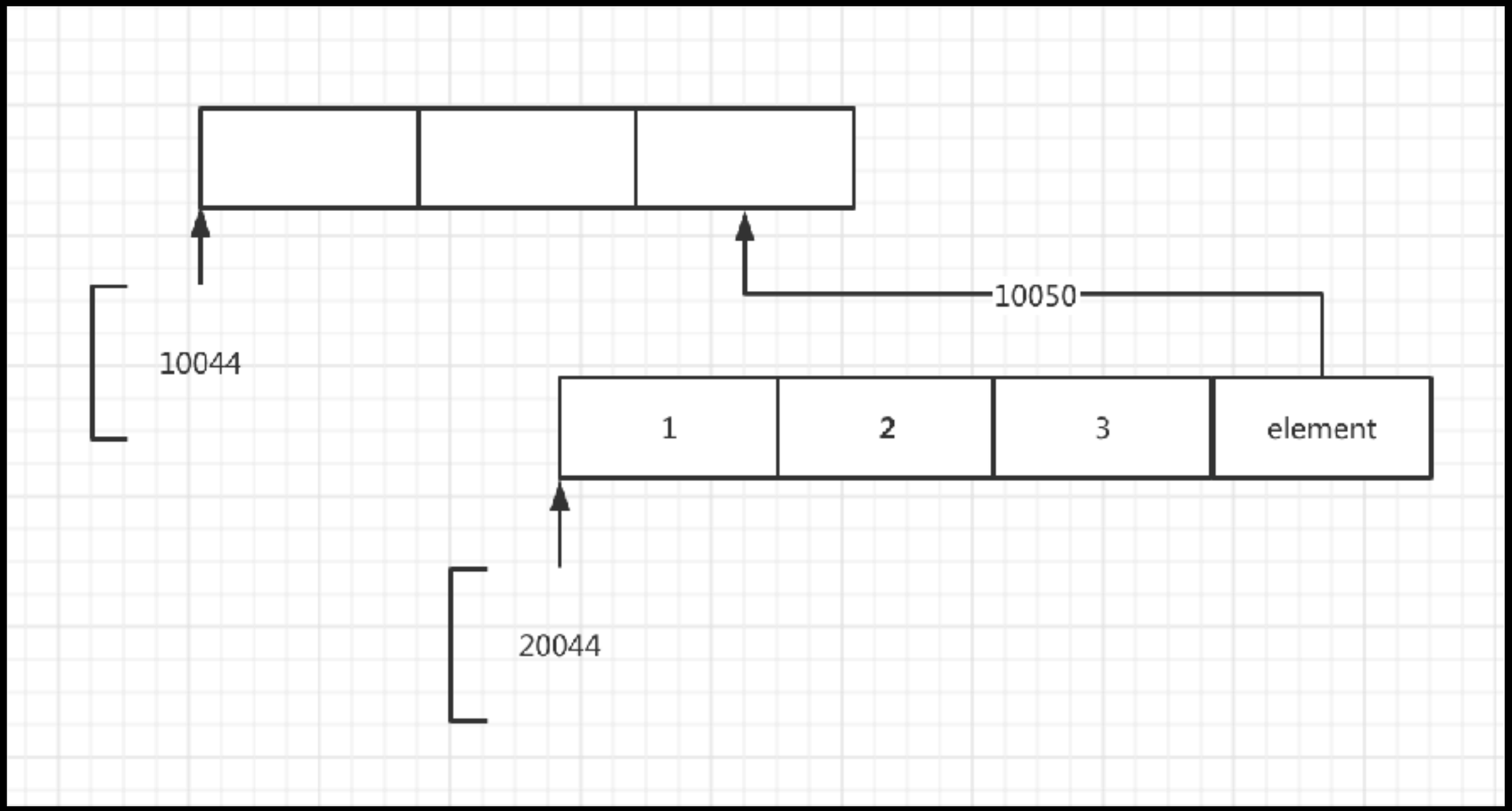
```
ExampleStateMachine::Start(state) => {
    // from body of `example`
    let foo_txt_future = async_read_file("foo.txt");
    // `.await` operation
    let state = WaitingOnFooTxtState {
        min_len: state.min_len,
        foo_txt_future,
    };
    *self = ExampleStateMachine::WaitingOnFooTxt(state);
}
```

自引用结构体

```rust
async fn pin_example() -> i32 {
    let array = [1, 2, 3];
    let element = &array[2];
    async_write_file("foo.txt", element.to_string()).await;
    *element
}
```

```rust
struct WaitingOnWriteState {
    array: [1, 2, 3],
    element: 0x1001c, // address of the last array element
}
```

自引用结构体的问题



自引用结构体的问题 — 根源是什么?

Pin的作用

Rust的Pin

编译器为async/await之后生成的 impl Future的结构体 !Unpin

```rust
/// A pinned pointer.
///
/// This is a wrapper around a kind of pointer which makes that pointer "pin" its
/// value in place, preventing the value referenced by that pointer from being moved
/// unless it implements [`Unpin`].
#[stable(feature = "pin", since = "1.33.0")]
#[lang = "pin"]
#[fundamental]
#[repr(transparent)]
#[derive(Copy, Clone)]
pub struct Pin<P> {
    pointer: P,
}

#[stable(feature = "pin", since = "1.33.0")]
impl<P: Deref> Deref for Pin<P> {
    type Target = P::Target;
    fn deref(&self) -> &P::Target {
        Pin::get_ref(Pin::as_ref(self))
    }
}

#[stable(feature = "pin", since = "1.33.0")]
impl<P: DerefMut<Target: Unpin>> DerefMut for Pin<P> {
    fn deref_mut(&mut self) -> &mut P::Target {
        Pin::get_mut(Pin::as_mut(self))
    }
}
```

Pin<P<T>> 取决于T是否 impl Unpin -> P<T>

Rust的Pin

```rust
/// A pinned pointer.
///
/// This is a wrapper around a kind of pointer which makes that pointer "pin" its
/// value in place, preventing the value referenced by that pointer from being moved
/// unless it implements [`Unpin`].
#[stable(feature = "pin", since = "1.33.0")]
#[lang = "pin"]
#[fundamental]
#[repr(transparent)]
#[derive(Copy, Clone)]
pub struct Pin<P> {
    pointer: P,
}

#[stable(feature = "pin", since = "1.33.0")]
impl<P: Deref> Deref for Pin<P> {
    type Target = P::Target;
    fn deref(&self) -> &P::Target {
        Pin::get_ref(Pin::as_ref(self))
    }
}

#[stable(feature = "pin", since = "1.33.0")]
impl<P: DerefMut<Target: Unpin>> DerefMut for Pin<P> {
    fn deref_mut(&mut self) -> &mut P::Target {
        Pin::get_mut(Pin::as_mut(self))
    }
}
```

Pin<P<T>> 取决于T是否 impl Unpin -> P<T>

QA环节

加群一起交流Rust & Datafuse

Datafuse Rust语言中文社区 知数堂 zhishuteng.com