Rust ASYNC Workshop

Devconf.cz 2023

Fernando Fernandez Mancera

Gris Ge

Quique Llorente

Red Hat

Agenda

- Quick wrap up of Rust ASYNC
- Consuming ASYNC functions in crates
- Exposing ASYNC functions in crate
- Tasks for Coding

Tasks for Practise

- Task A: Use `reqwest` or other to get the title of https://www.devconf.cz
- Task B: Create a rust library providing async sleep
- Task C: Create pull request for async **futures::stream::Stream** API for librabc

Quick wrap up of Rust ASYNC

- The `async/await` syntax
- The `Future` trait
- Executor – tokio/async-std/smol/etc
- Example code stored in Github

Quick wrap up of Rust ASYNC

- The `async/await`

```
use std::future::Future;

fn foo() -> impl Future<Output = u8> {
    async {
        println!("foo() 1");
        1
    }
}
async fn bar() -> u8 {
    println!("bar() 2");
    2
}
#[tokio::main]
async fn main() {
    foo().await;
    bar().await;
}
```

Quick wrap up of Rust ASYNC

- The `async/await`
- `join`

```
use std::future::Future;
use std::thread::sleep;
use std::time::Duration;
use futures::future::join;
fn foo() -> impl Future<Output = ()> {
    async {
        sleep(Duration::from_secs(3));
        println!("foo() slept 3");
    }
}
async fn bar() {
    sleep(Duration::from_secs(1));
    println!("bar() slept 1");
}
#[async_std::main]
async fn main() {
    join(foo(), bar()).await;
}
```

Quick wrap up of Rust ASYNC

- The `Future` trait
  - Pin
  - Context waker
  - poll

```
pub trait Future {
    type Output;

    fn poll(
        self: Pin<&mut Self>,
        cx: &mut Context<'_>,
    ) -> Poll<Self::Output>;
}

pub enum Poll<T> {
    Ready(T),
    Pending,
}
```

Quick wrap up of Rust ASYNC

- The `Future` trait
  - Use thread

```
struct SharedState {
    completed: bool,
    waker: Option<Waker>,
}
struct AsyncSleep {
    shared_state: Arc<Mutex<SharedState>>,
}
impl AsyncSleep {
    fn sleep(dur: Duration) -> Self {
     let shared_state = Arc::new(Mutex::new(SharedState {
          completed: false,
          waker: None,
     }));
     let thread_state = shared_state.clone();
     std::thread::spawn(move || {
          std::thread::sleep(dur);
          let mut state = thread_state.lock().unwrap();
          state.completed = true;
          if let Some(waker) = state.waker.take() {
              waker.wake()
          }
     });
     Self { shared_state }
    }
}
```

Quick wrap up of Rust ASYNC

- The `Future` trait
  - Use thread

```
impl Future for AsyncSleep {
    type Output = ();
    fn poll(
        self: Pin<&mut Self>,
        cx: &mut Context<'_>,
    ) -> Poll<Self::Output> {
        let mut shared_state = self
            .shared_state
            .lock()
            .unwrap();
        if shared_state.completed {
            Poll::Ready(())
        } else {
            shared_state.waker = Some(
                cx.waker().clone(),
            );
            Poll::Pending
        }
    }
}
```

Quick wrap up of Rust ASYNC

- The `Future` trait
  - `tokio::AsyncFd`
  - `smol::Async`

```
#[cfg(feature = "smol")]
use smol::Async as DefaultAsync;
#[cfg(not(feature = "smol"))]
use tokio::io::unix::AsyncFd as DefaultAsync;

struct AsyncSleep;
impl AsyncSleep {
     async fn sleep(dur: Duration) {
          let timer = TimerFd::new(..)
          .unwrap();
          timer.set( .. );
          let _ = DefaultAsync::new(timer)
               .unwrap()
               .readable()
               .await
               .unwrap();
     }
}
```

Consuming ASYNC functions in crates

- Choose a executor:

  `tokio/smol/async-std/etc`
- Use `async/await` keywords

Exposing ASYNC functions in crate

- Thread based independent waker
- Using reactor inside of async runtime

Tasks for Practise

- Task A: Use `reqwest` or other to get the title of https://www.devconf.cz
- Task B: Create a rust library providing async sleep
- Task C: Create pull request for async **futures::stream::Stream** API for librabc

Tasks for Practise

- github.com/cathay4t/librabc
  - `RabcClient::new() -> Self`
  - `RabcClient::poll() -> Vec<RabcEvent>`
  - `RabcClient::process() -> Option<String>`

Thank you!
Enjoy Hacking!
Feel free to request help!

Red Hat