通过对futures库分析加深对 Rust 异步运行时的理解

大家可以通过 Tokio 在实战中去理解异步

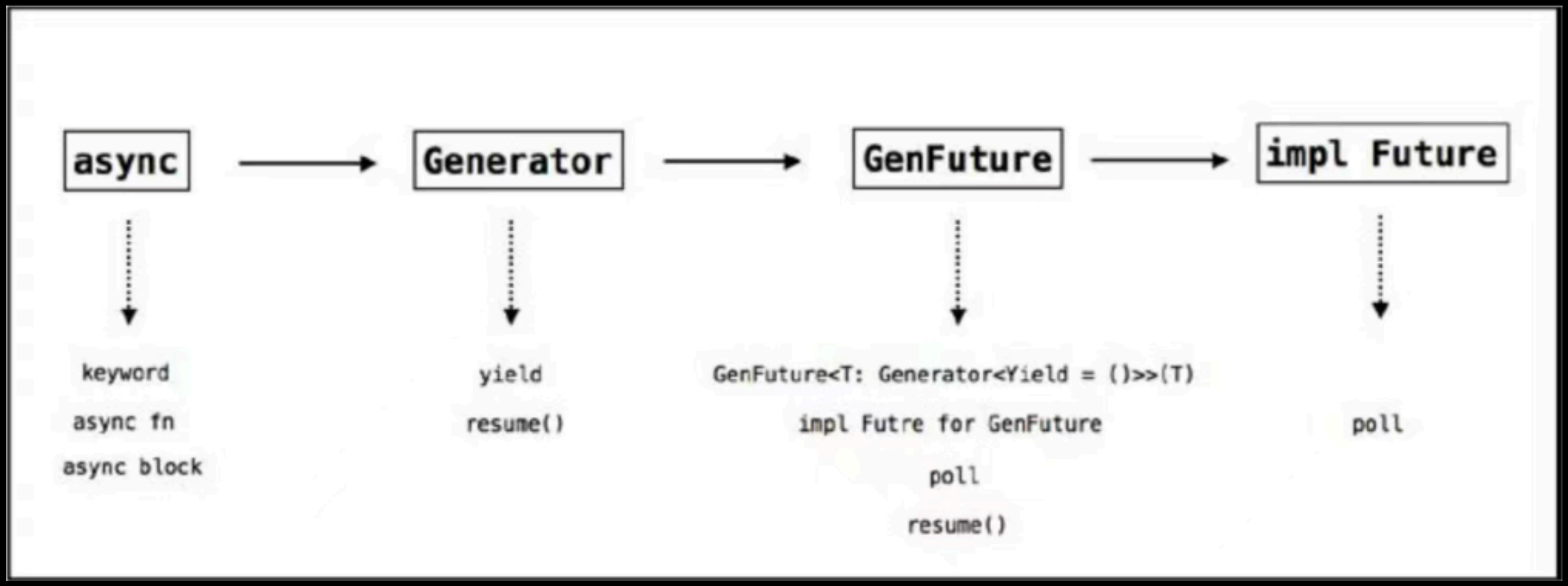
苏林

公开课的内容

1、对Rust异步编程进行整体重点的回顾.

2、一起探讨futures-rs库.

回顾async/await编译器背后做的一些事



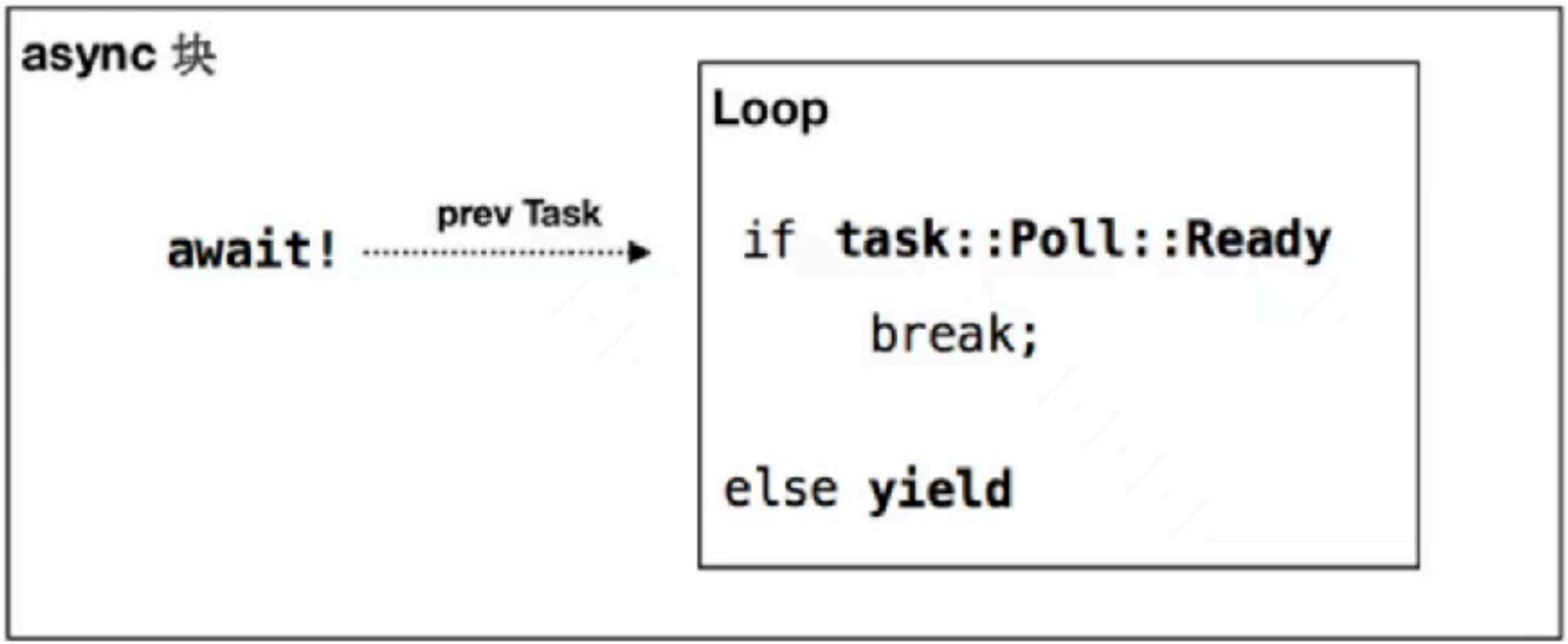
由async生成Future过程示意图

回顾async/await编译器背后做的一些事

```rust
impl<T: Generator<ResumeTy, Yield = ()>> Future for GenFuture<T> {
    type Output = T::Return;
    fn poll(self: Pin<&mut Self>, cx: &mut Context<'_>) -> Poll<Self::Output> {
        // SAFETY: Safe because we're !Unpin + !Drop, and this is just a field projection.
        let gen : Pin<&mut T> = unsafe { Pin::map_unchecked_mut(self, |s : &mut GenFuture<T>| &mut s.0) };

        // Resume the generator, turning the `&mut Context` into a `NonNull` raw pointer. The
        // `.await` lowering will safely cast that back to a `&mut Context`.
        match gen.resume( arg: ResumeTy(NonNull::from( reference: cx).cast::<Context<'static>>())) {
            GeneratorState::Yielded(()) => Poll::Pending,
            GeneratorState::Complete(x : <T as Generator<ResumeTy>>::Return ) => Poll::Ready(x),
        }
    }
}
```

回顾async/await编译器背后做的一些事

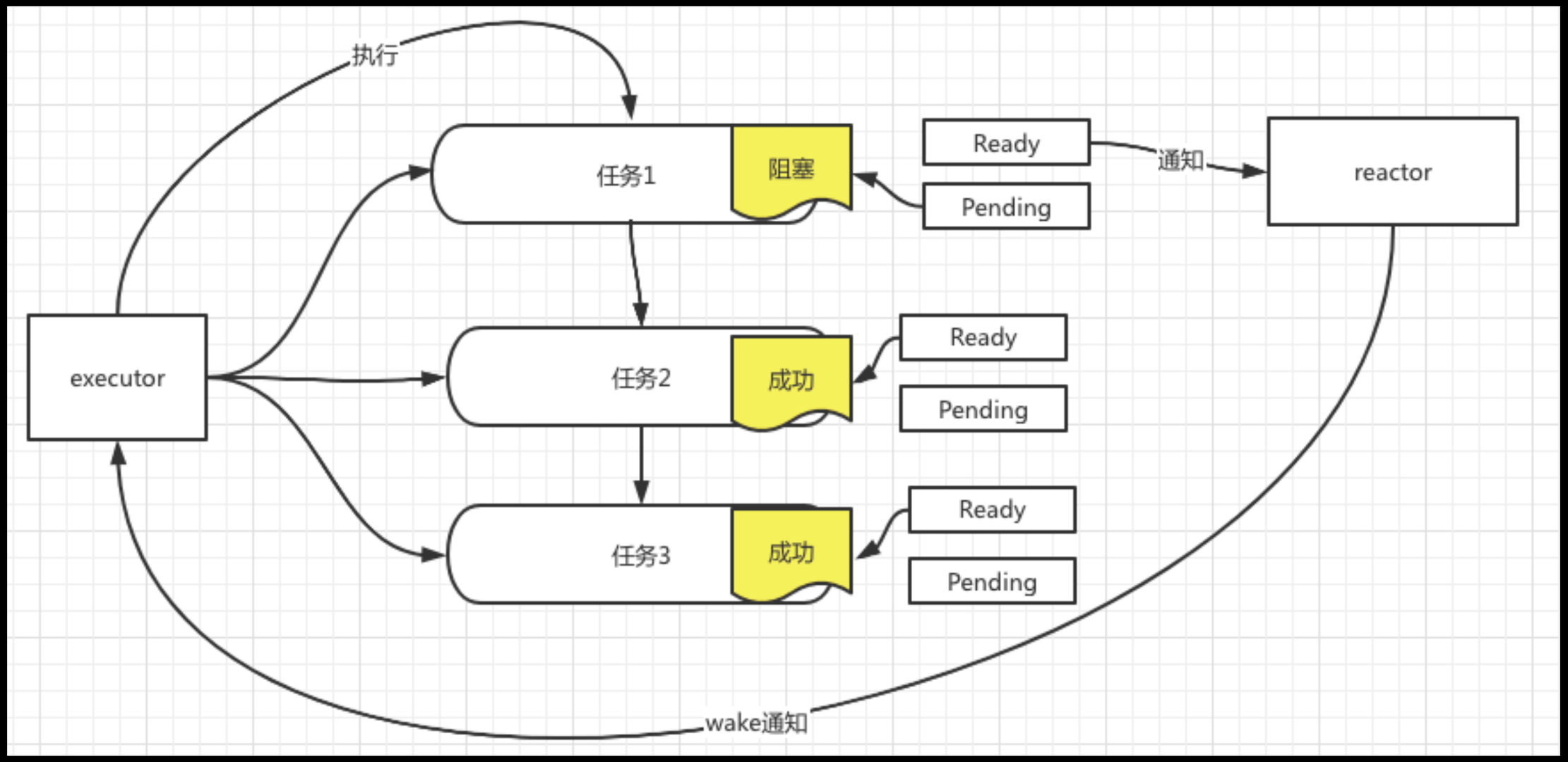


回顾async/await编译器背后做的一些事

2033-experimental-coroutines.md

https://github.com/rust-lang/rfcs/blob/master/text/2033-experimental-coroutines.md

回顾一下之前画过的简单的一张图



运行时

叶子feature

```rust
let mut stream = tokio::net::TcpStream::connect("127.0.0.1:3000");
```

非叶子feature

```rust
let non_leaf = async {
    let mut stream = TcpStream::connect("127.0.0.1:3000").await.unwrap();// <- yield
    println!("connected!");
    let result = stream.write(b"hello world\n").await; // <- yield
    println!("message sent!");
    ...
};
```

运行时

异步系统可以分为三个部分

1、Reactor
2、Executor　　Waker
3、Future

多线程+协程 => 实现高性能I/O服务器的利器

Futures-rs

futures-rs是rust官方提供的一个类库，它是Rust异步编程的基础。包括关键trait的定义如Stream，以及宏如join!， select!以及各种future组合子用来控制异步流程。

```
├── examples //包含两个demo
├── futures //统一对外提供的接口,提供异步编程的核心抽象,使用的是下面的子类库
├── futures-channel //异步通信channel实现:包含onshot和mpsc
├── futures-core //futures库的核心trait和type
├── futures-executor  //异步任务执行器
├── futures-io   //AsyncRead AsyncWrite AsyncSeek AsyncBufRead等trait
├── futures-macro //join! try_join! select! select_biased!宏实现
├── futures-sink //Sink trait
├── futures-task //处理task的工具,如FutureObj/LocalFutureObj struct、 Spawn/LocalSpawn ArcWake trait、
├── futures-test //用于测试futures-rs的通用工具类库(非测试用例)
├── futures-util //通用工具类库、扩展trait(如AsyncReadExt、SinkExt、SpawnExt)
```

Futures-rs

```rust
use futures::channel::mpsc;
use futures::executor;
use futures::executor::ThreadPool;
use futures::StreamExt;

fn main() {
    let pool = ThreadPool::new().expect("Failed to build pool");
    let (tx, rx) = mpsc::unbounded::<i32>();

    // 使用async块创建一个future,返回一个Future的实现
    // 此刻尚未提供executor给这个future,因此它不会运行
    let fut_values = async {
        // 创建另外一个async块,同样会生成Future的实现,
        // 它在又async块里面,因此会在又async块执行后,提供executor给子async块执行
        // executor这段是由Future::poll的第二个参数std::task::Context完成,
        // 它代表了我们的executor,子async块生成的Future实现它能被polled(使用父async块的executor)
        let fut_tx_result = async move {
            (0..100).for_each(|v| {
                tx.unbounded_send(v).expect("Failed to send");
            })
        };

        // 使用thread pool 的spawn方法传输生成的future
        pool.spawn_ok(fut_tx_result);

        let fut_values = rx
            .map(|v| v * 2)
            .collect();

        //使用async块提供的executue去等待fut_values完成
        fut_values.await
    };
    // 真正的去调用上面的fut_values future,执行它的poll方法和Future里的子future的poll方法,最终驱动fut_values旅驱动完
    let values: Vec<i32> = executor::block_on(fut_values);

    println!("Values={:?}", values);
}
```

QA环节

加群一起交流Rust & Datafuse

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