Rust异步编程入门

异步编程让程序不通过多线程达到类似多线程的效果

苏林

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

自我介绍

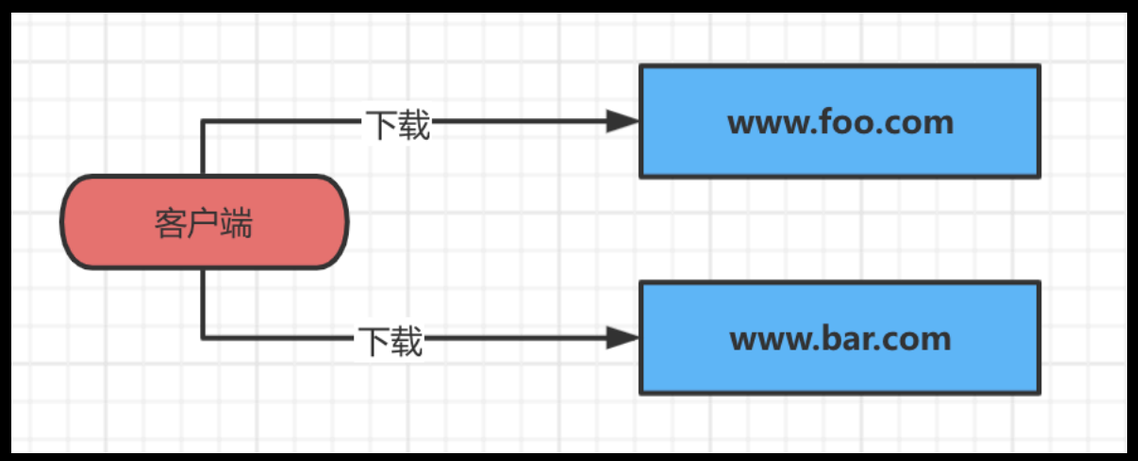
- 前折800互联网研发团队负责人, 10余年一线研发经验
- 目前是多点Dmall技术Leader
- 具有多年的软件开发经验, 熟悉Ruby、Java、Rust等开发语言
- 同时也参与过Rust中文社区日报维护工作.

分享内容

- 为什么需要异步
- async/await 介绍
- 异步编程模型
- 带领大家实现一个简化版的 future

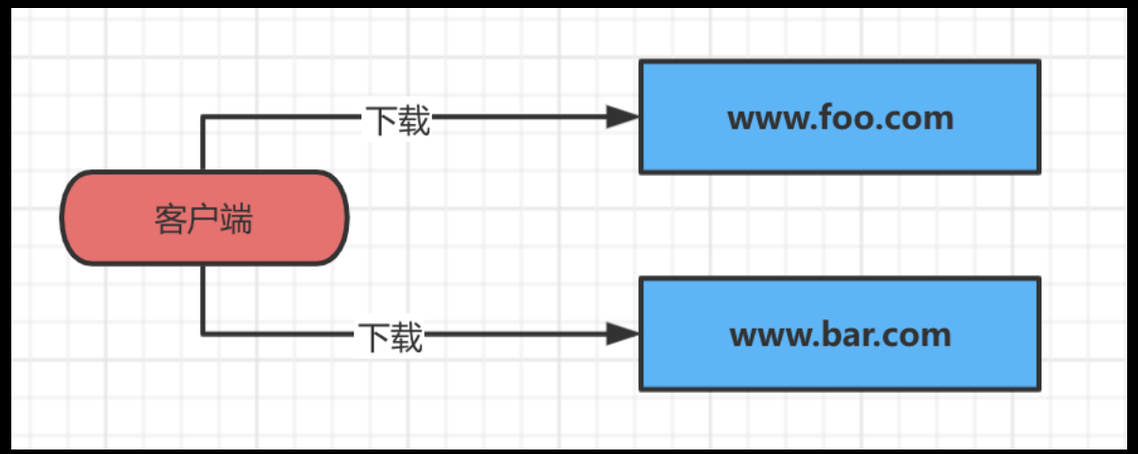
为什么需要异步

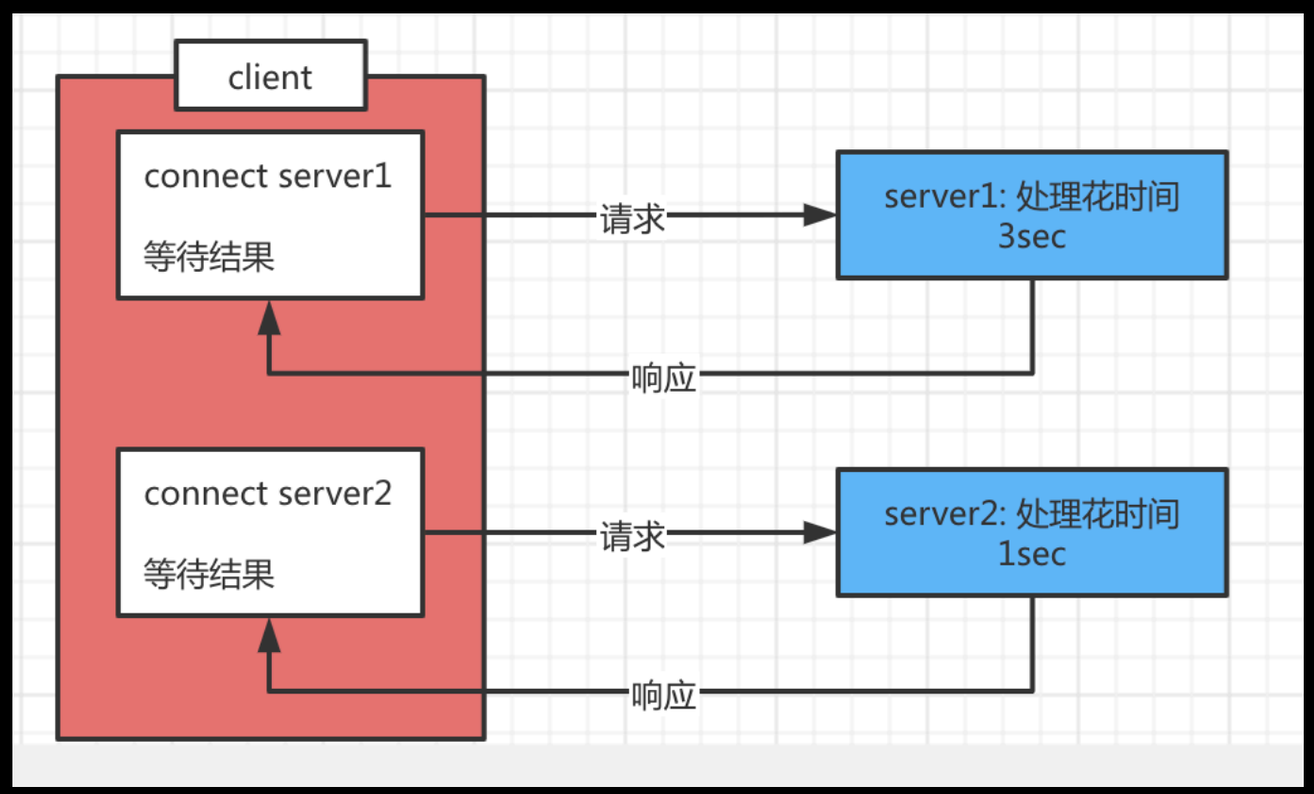
通过官方教程的一个例子, 来理解一下



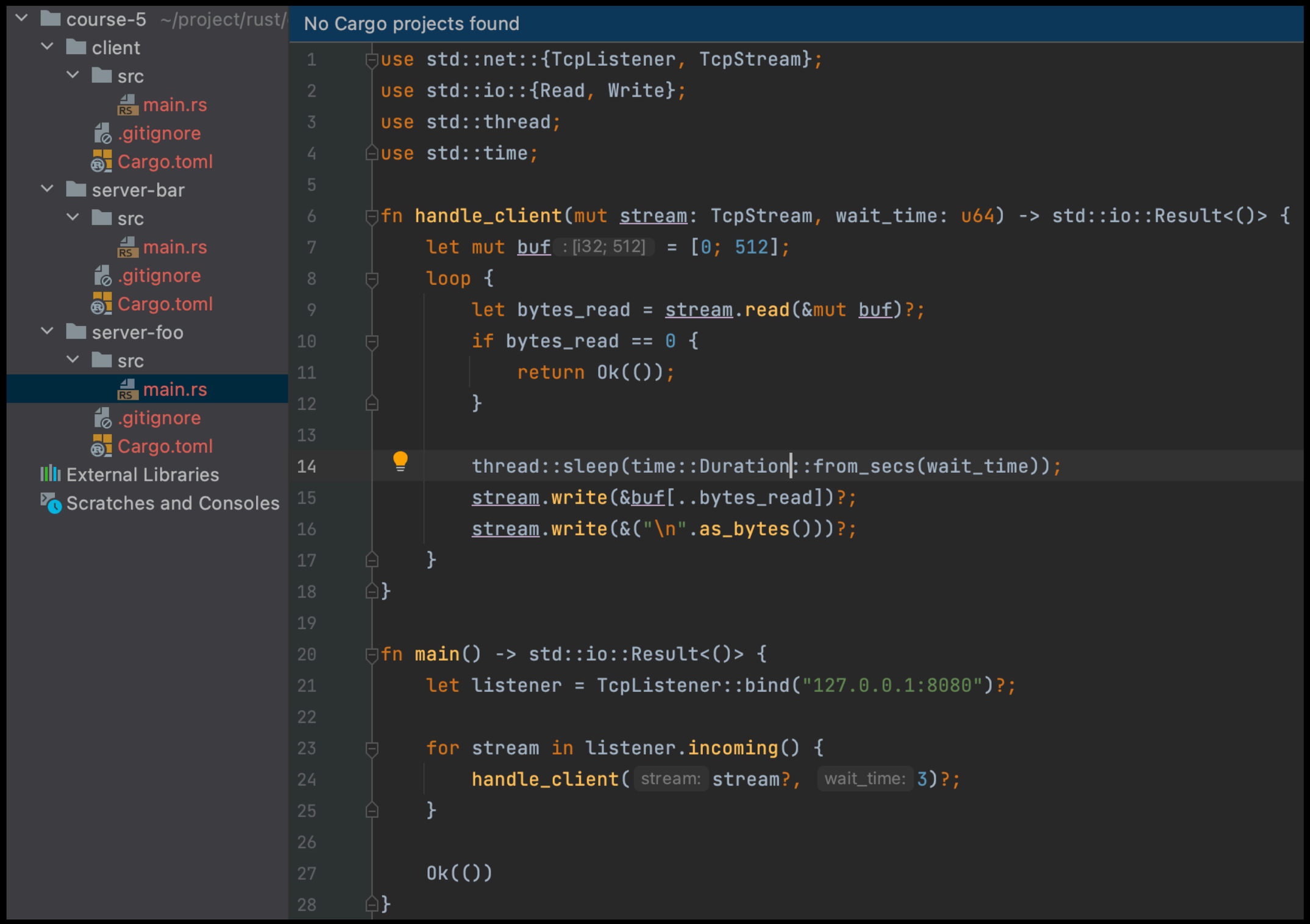
为什么需要异步

通过官方教程的一个例子, 来理解一下

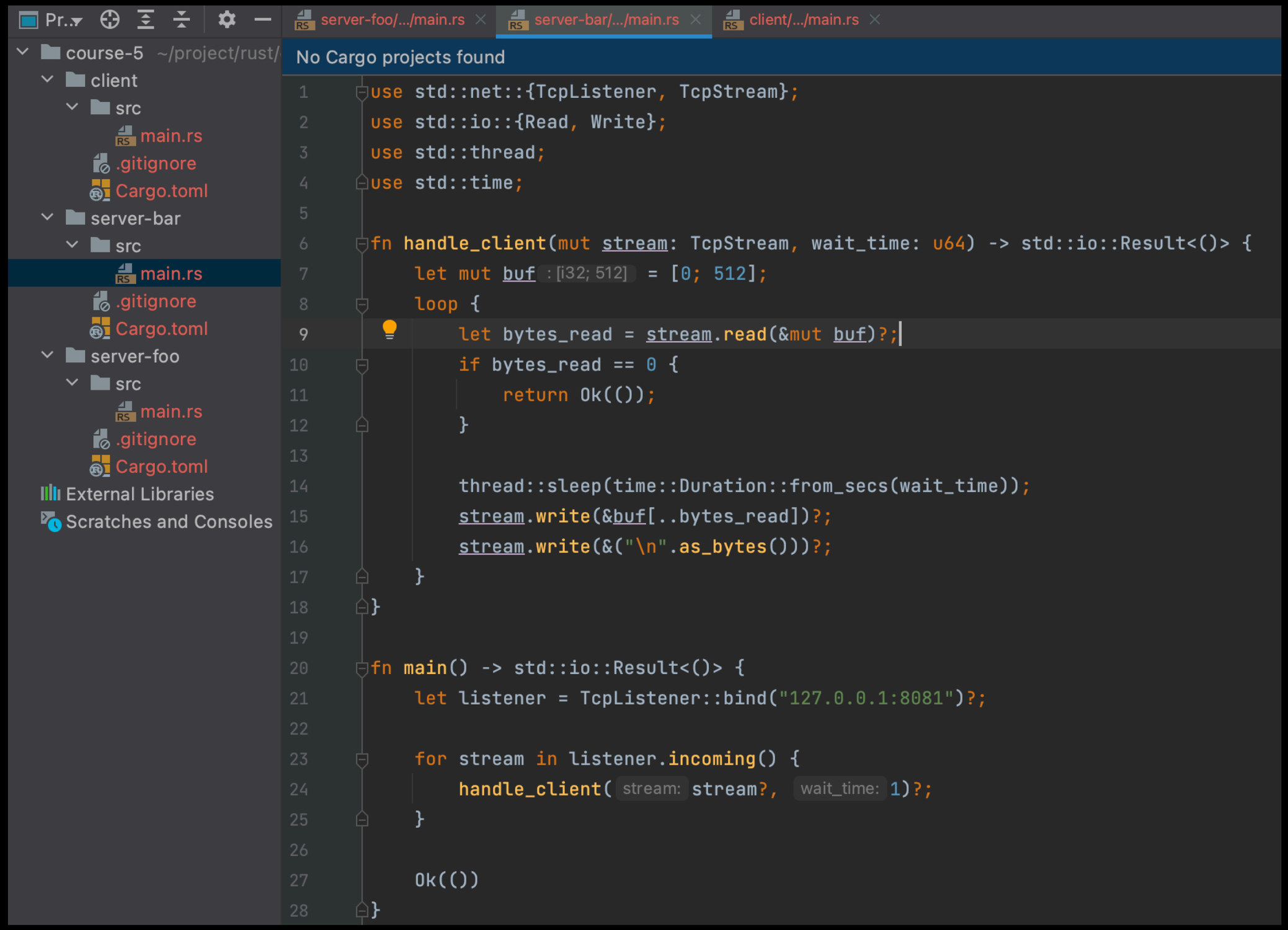




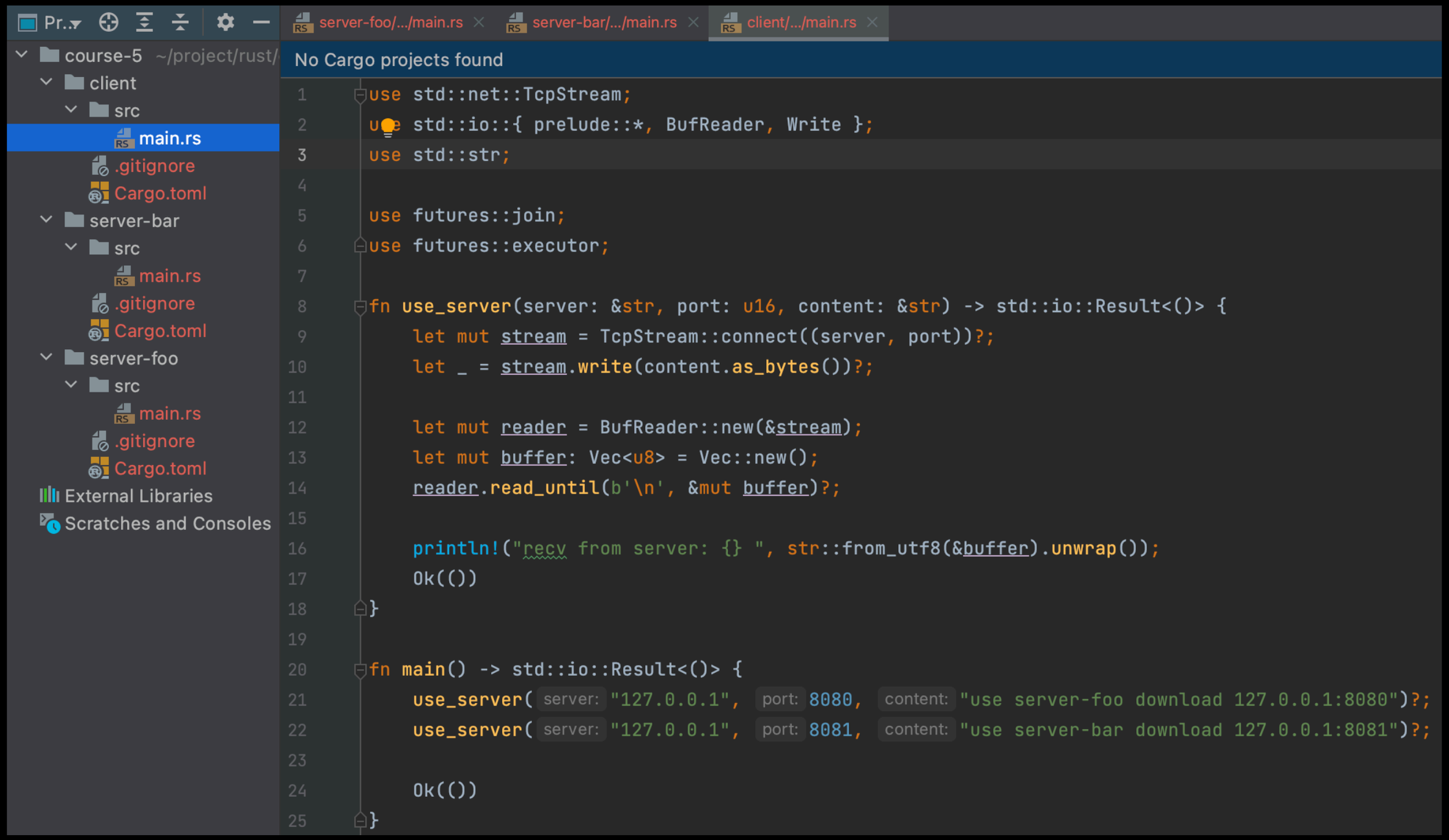
为什么需要异步


course-5 ~/project/rust/
client
src
main.rs
.gitignore
Cargo.toml
server-bar
src
main.rs
.gitignore
Cargo.toml
server-foo
src
main.rs
.gitignore
Cargo.toml
External Libraries
Scratches and Consoles
No Cargo projects found
use std::net::{TcpListener, TcpStream};
use std::io::{Read, Write};
use std::thread;
use std::time;
fn handle_client(mut stream: TcpStream, wait_time: u64) -> std::io::Result<()> {
let mut buf : [i32; 512] = [0; 512];
loop {
let bytes_read = stream.read(&mut buf)?;
if bytes_read == 0 {
return Ok(());
}
thread::sleep(time::Duration::from_secs(wait_time));
stream.write(&buf[..bytes_read])?;
stream.write(&("\n".as_bytes()))?;
}
}
fn main() -> std::io::Result<()> {
let listener = TcpListener::bind("127.0.0.1:8080")?;
for stream in listener.incoming() {
handle_client(stream: stream?, wait_time: 3)?;
}
Ok(())
}

为什么需要异步

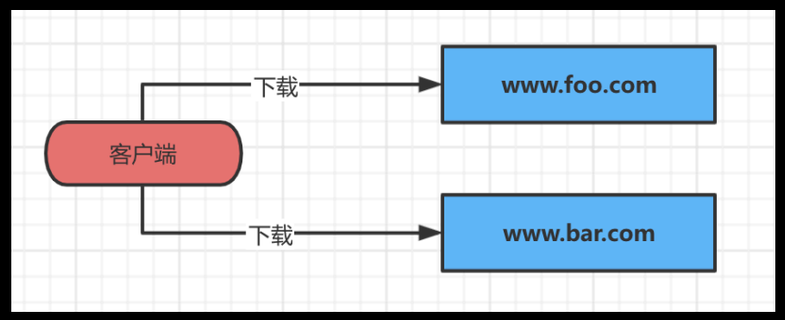

Pr..
server-foo/.../main.rs
server-bar/.../main.rs
client/.../main.rs
course-5 ~/project/rust/
client
src
main.rs
.gitignore
Cargo.toml
server-bar
src
main.rs
.gitignore
Cargo.toml
server-foo
src
main.rs
.gitignore
Cargo.toml
External Libraries
Scratches and Consoles
No Cargo projects found
use std::net::{TcpListener, TcpStream};
use std::io::{Read, Write};
use std::thread;
use std::time;
fn handle_client(mut stream: TcpStream, wait_time: u64) -> std::io::Result<()> {
let mut buf : [i32; 512] = [0; 512];
loop {
let bytes_read = stream.read(&mut buf)?;
if bytes_read == 0 {
return Ok(());
}
thread::sleep(time::Duration::from_secs(wait_time));
stream.write(&buf[..bytes_read])?;
stream.write(&("\n".as_bytes()))?;
}
}
fn main() -> std::io::Result<()> {
let listener = TcpListener::bind("127.0.0.1:8081")?;
for stream in listener.incoming() {
handle_client(stream: stream?, wait_time: 1)?;
}
Ok(())
}

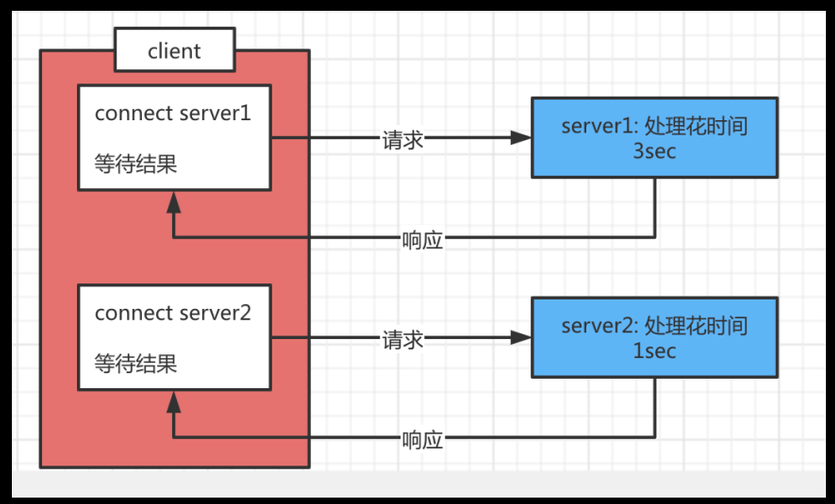
为什么需要异步

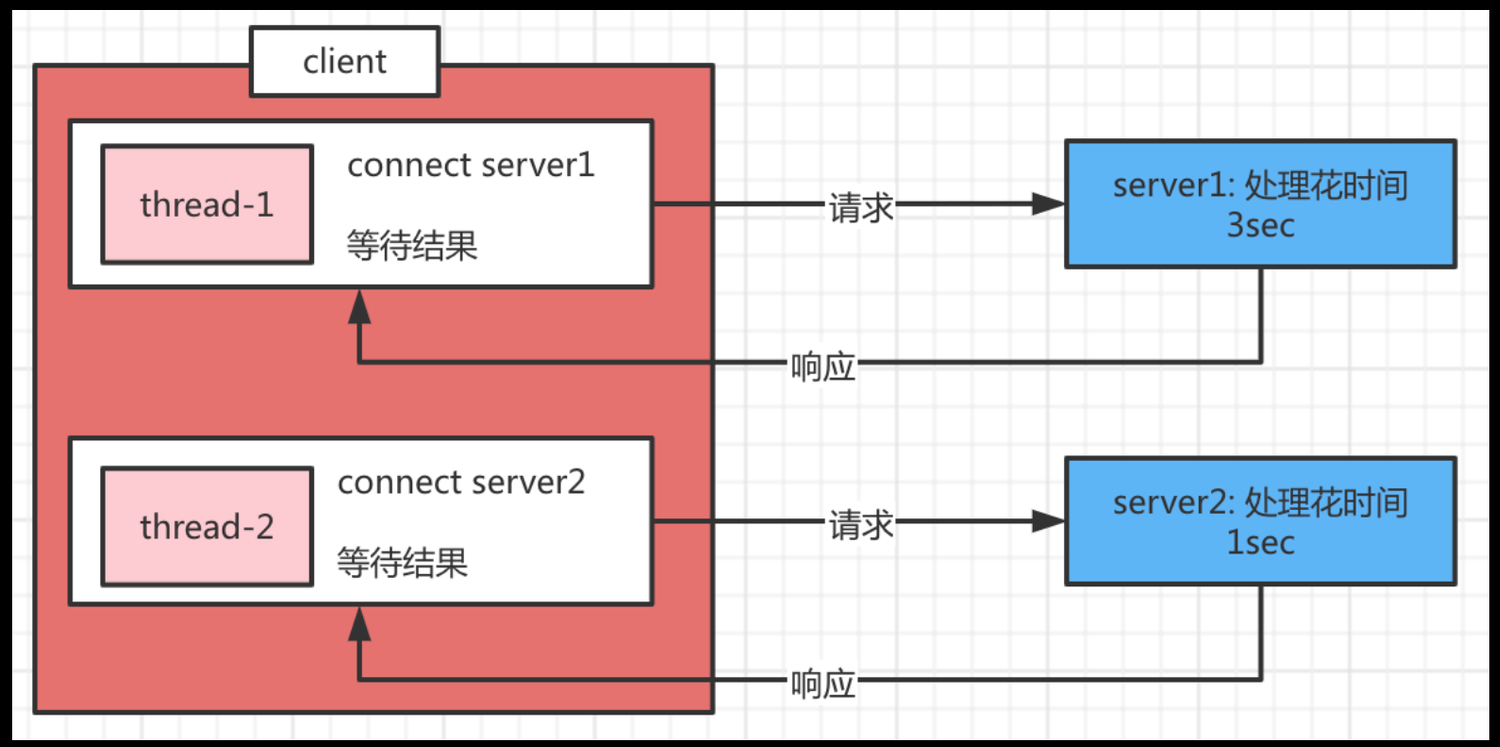

Pr..
course-5 ~/project/rust/
client
src
main.rs
.gitignore
Cargo.toml
server-bar
src
main.rs
.gitignore
Cargo.toml
server-foo
src
main.rs
.gitignore
Cargo.toml
External Libraries
Scratches and Consoles
server-foo/.../main.rs
server-bar/.../main.rs
client/.../main.rs
No Cargo projects found
use std::net::TcpStream;
use std::io::{ prelude::*, BufReader, Write };
use std::str;
use futures::join;
use futures::executor;
fn use_server(server: &str, port: u16, content: &str) -> std::io::Result<()> {
let mut stream = TcpStream::connect((server, port))?;
let _ = stream.write(content.as_bytes())?;
let mut reader = BufReader::new(&stream);
let mut buffer: Vec<u8> = Vec::new();
reader.read_until(b'\n', &mut buffer)?;
println!("recv from server: {} ", str::from_utf8(&buffer).unwrap());
Ok(())
}
fn main() -> std::io::Result<()> {
use_server(server: "127.0.0.1", port: 8080, content: "use server-foo download 127.0.0.1:8080")?;
use_server(server: "127.0.0.1", port: 8081, content: "use server-bar download 127.0.0.1:8081")?;
Ok(())
}

为什么需要异步

通过官方教程的一个例子, 来理解一下

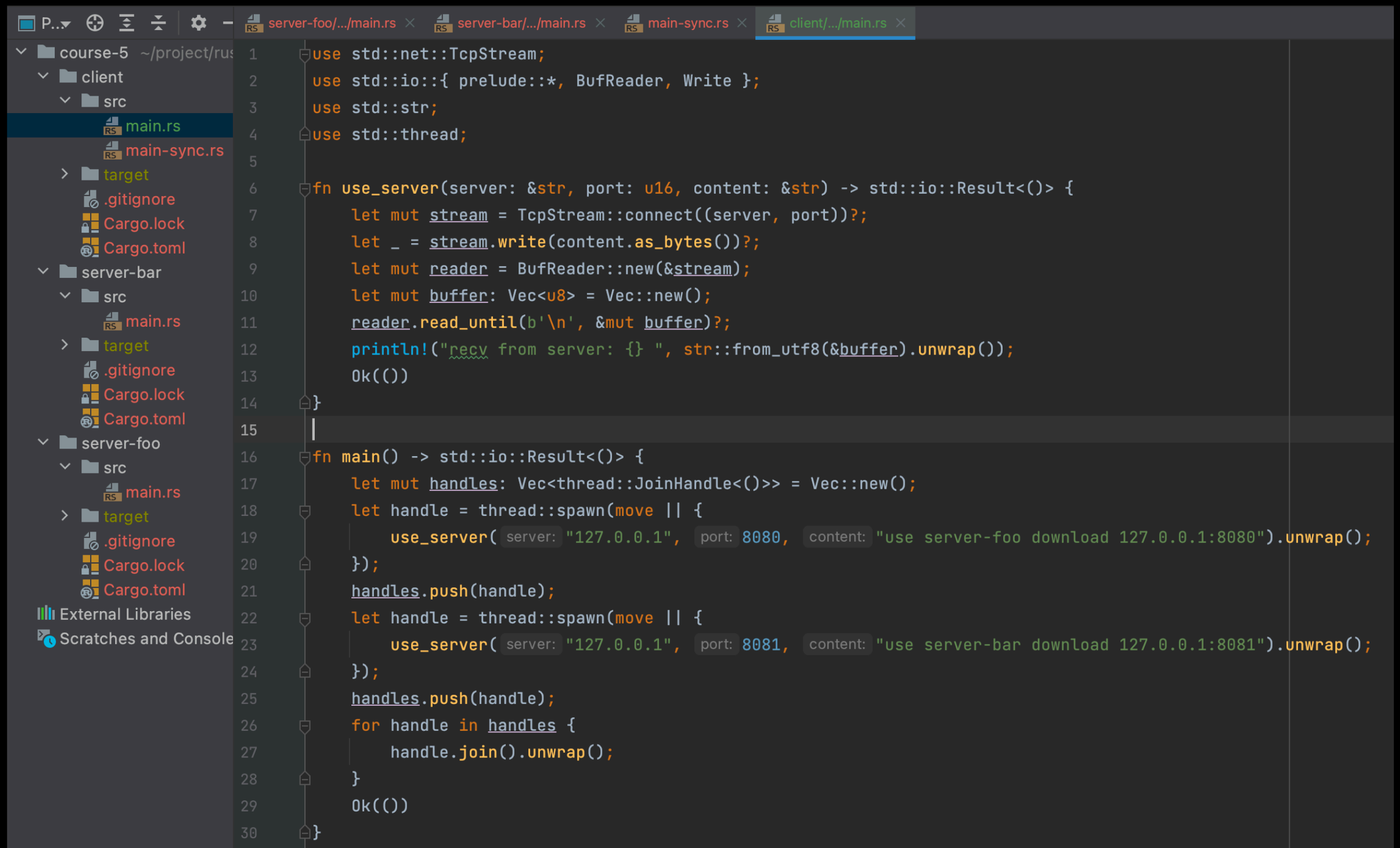






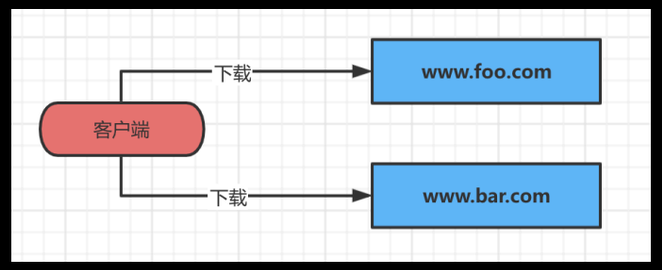
为什么需要异步

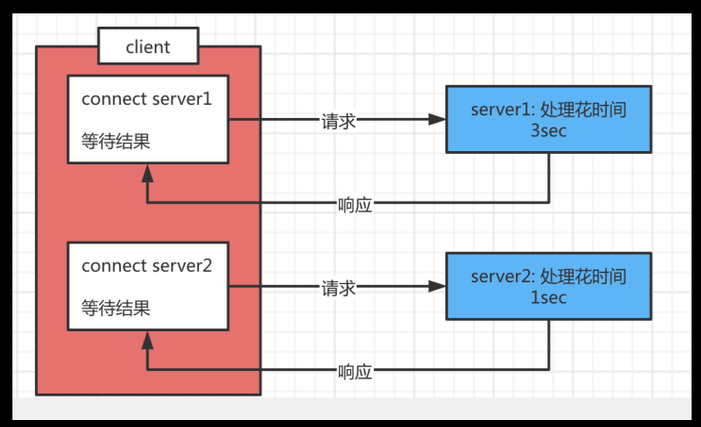
通过官方教程的一个例子，来理解一下

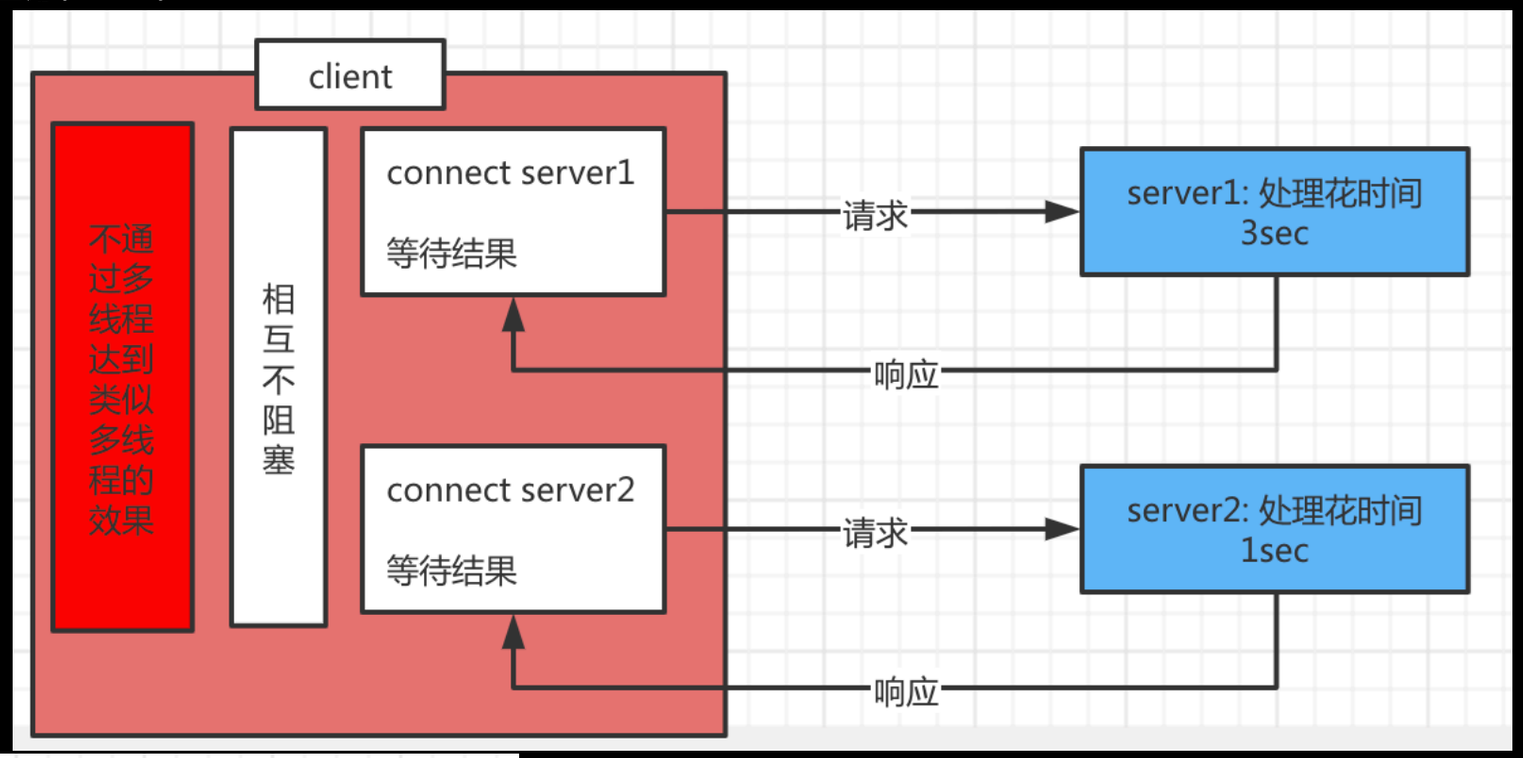


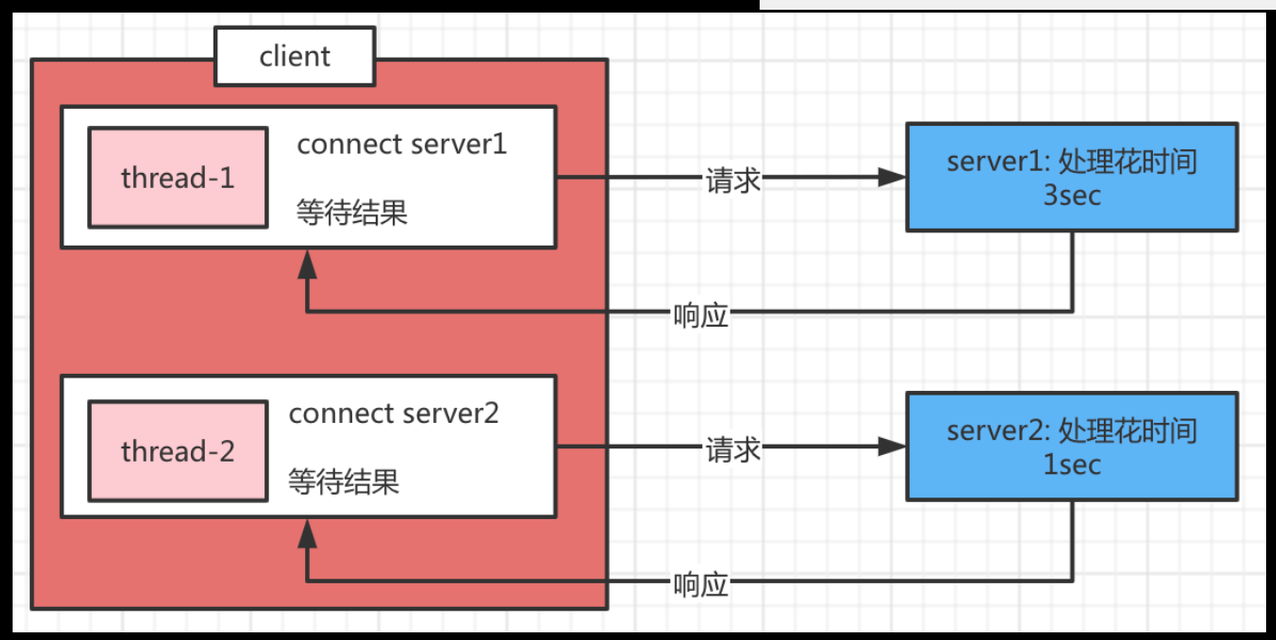
为什么需要异步

通过官方教程的一个例子, 来理解一下



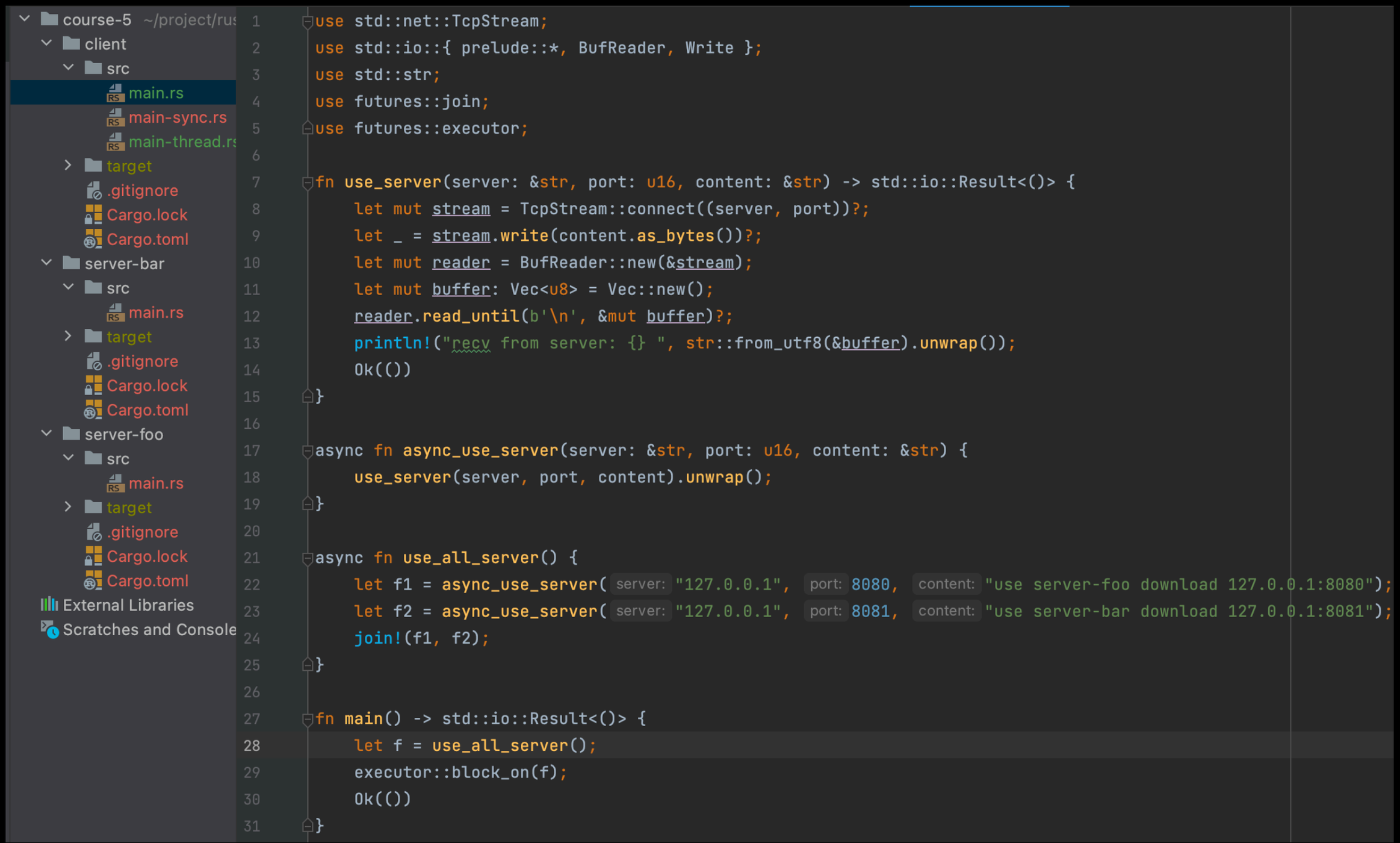






为什么需要异步

通过官方教程的一个例子，来理解一下



async\await 简单介绍

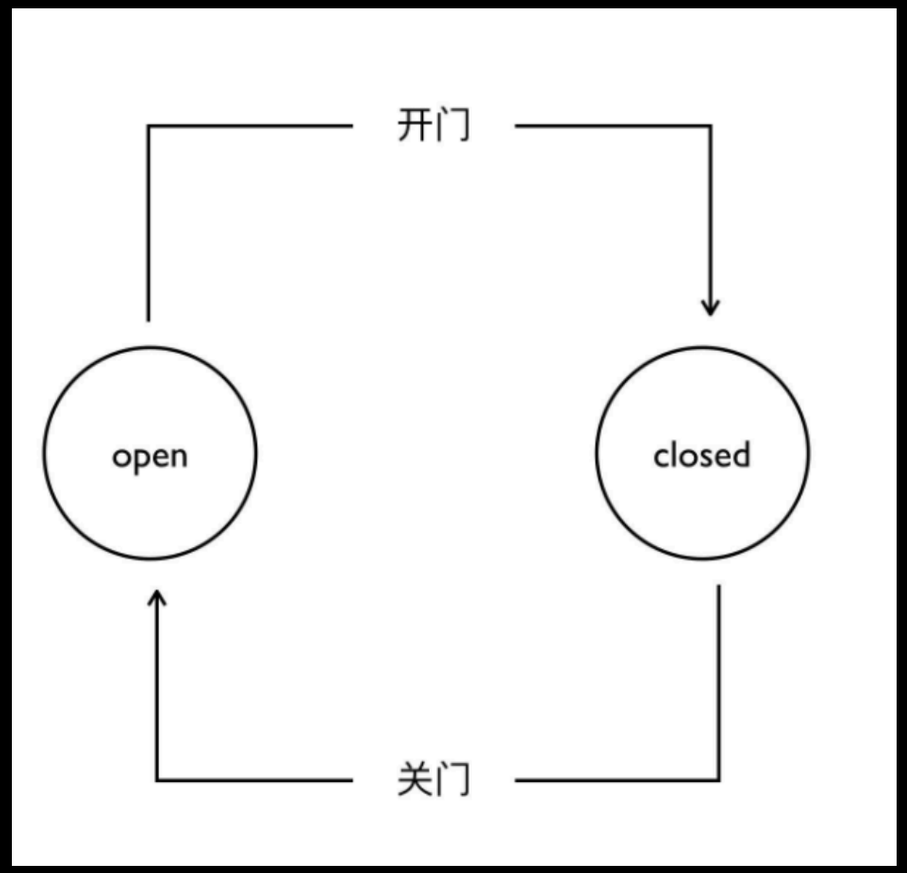
async/await是Rust编写异步的内置工具
async将一个代码块转化为实现了future特征的状态机
那么转化为future后有什么作用呢?

async\await 简单介绍

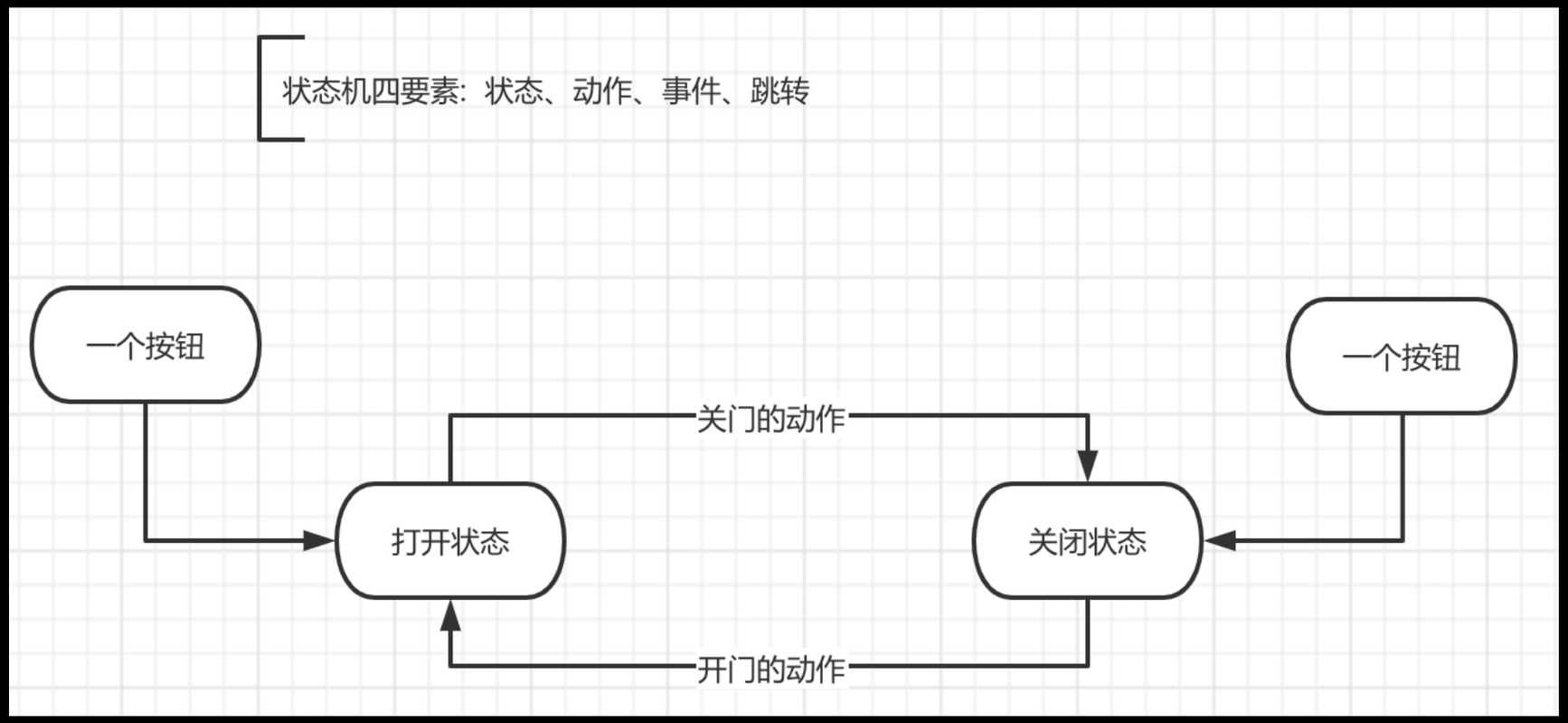
async/await是Rust编写异步的内置工具
async将一个代码块转化为实现了future特征的状态机.
那么转化为future后有什么作用呢?

-> 在同步方法中调用阻塞函数(async转化的函数), 会阻塞整个线程, 但是, 阻塞的future会让出线程控制权, 允许其它future运行.

什么是状态机

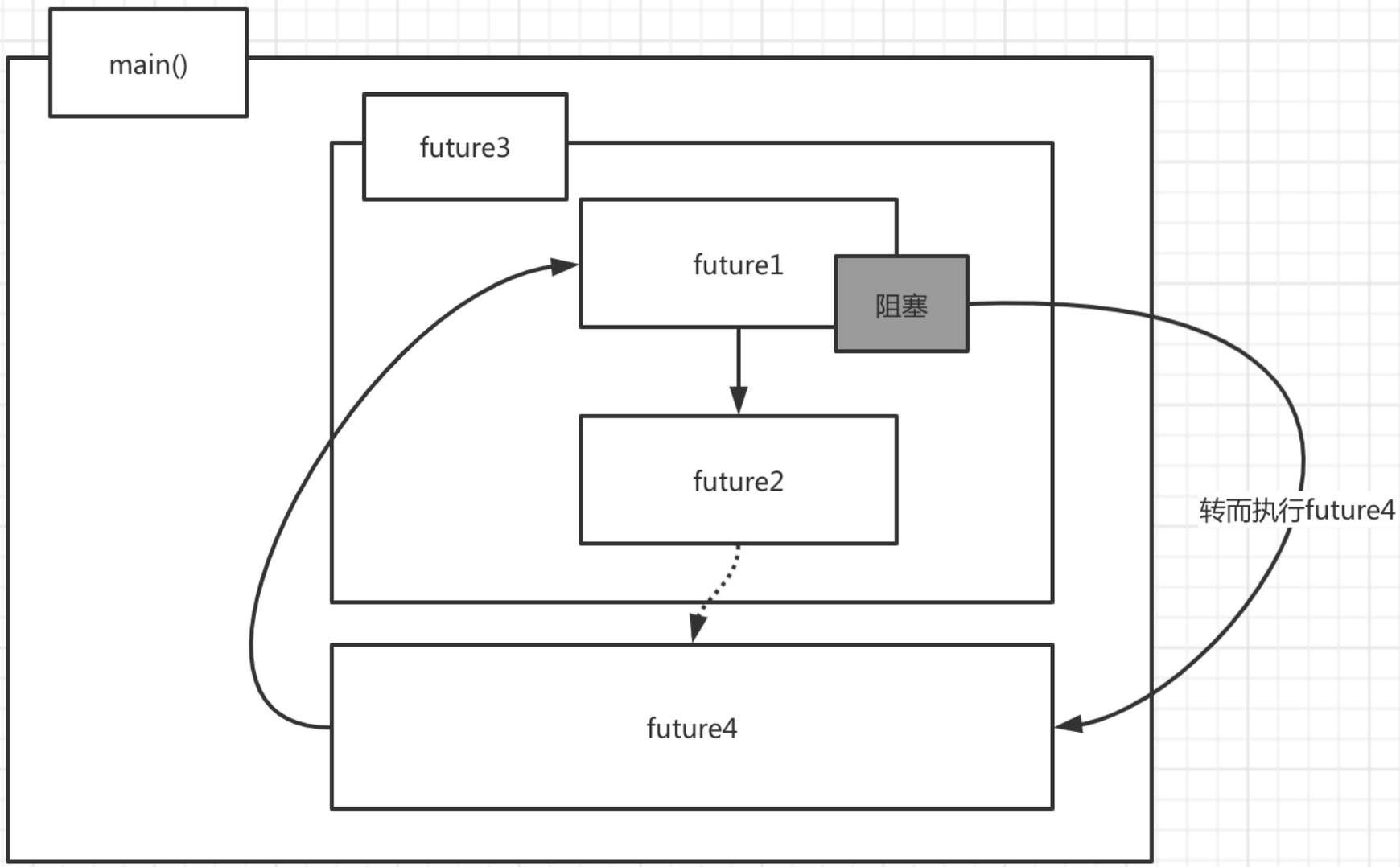


什么是状态机

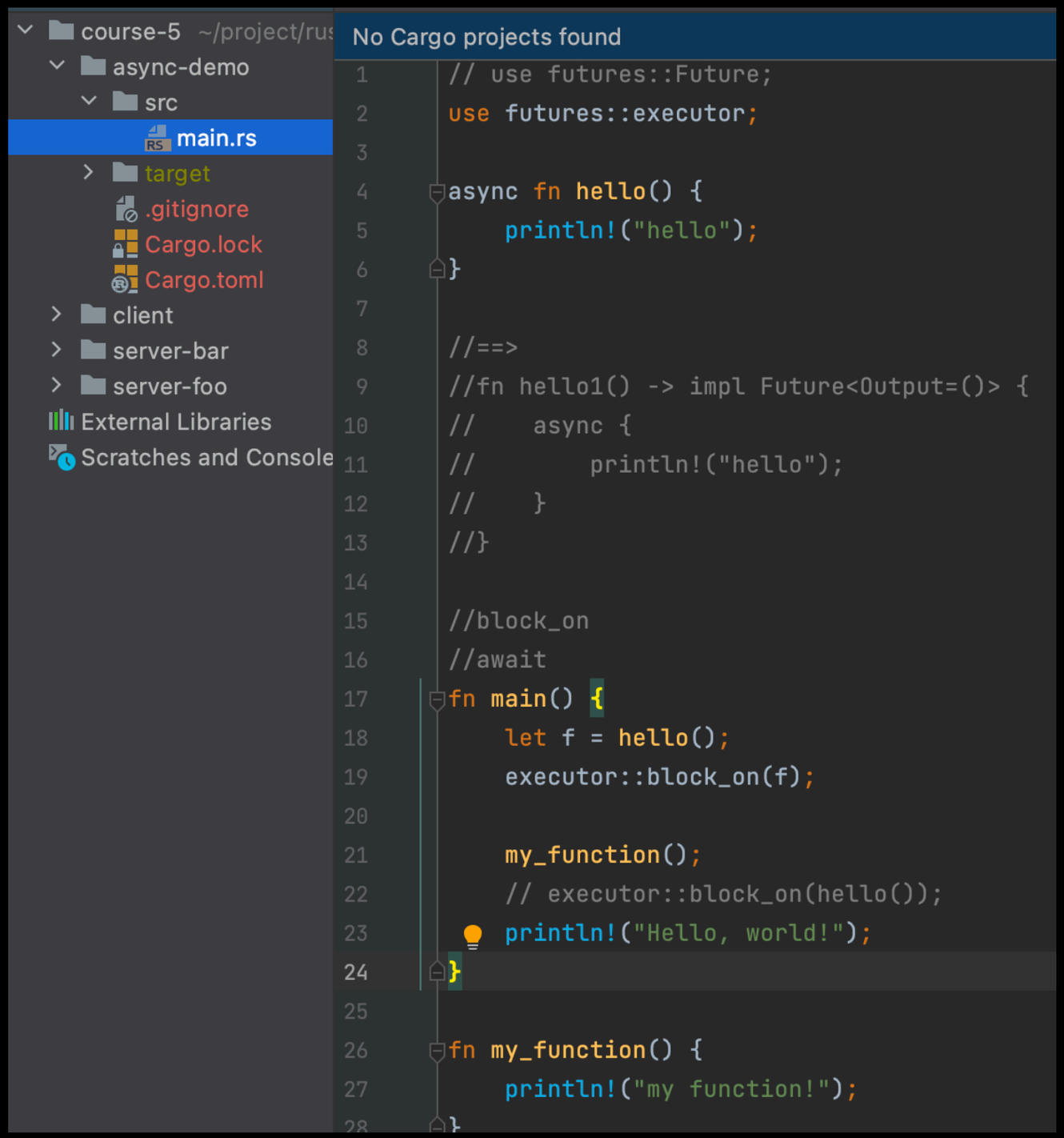


通过一个图来说明 那么转化为future后有什么作用呢?

在同步方法中调用阻塞函数(async转化的函数), 会阻塞整个线程, 但是, 阻塞的future会让出线程控制权, 允许其它future运行.

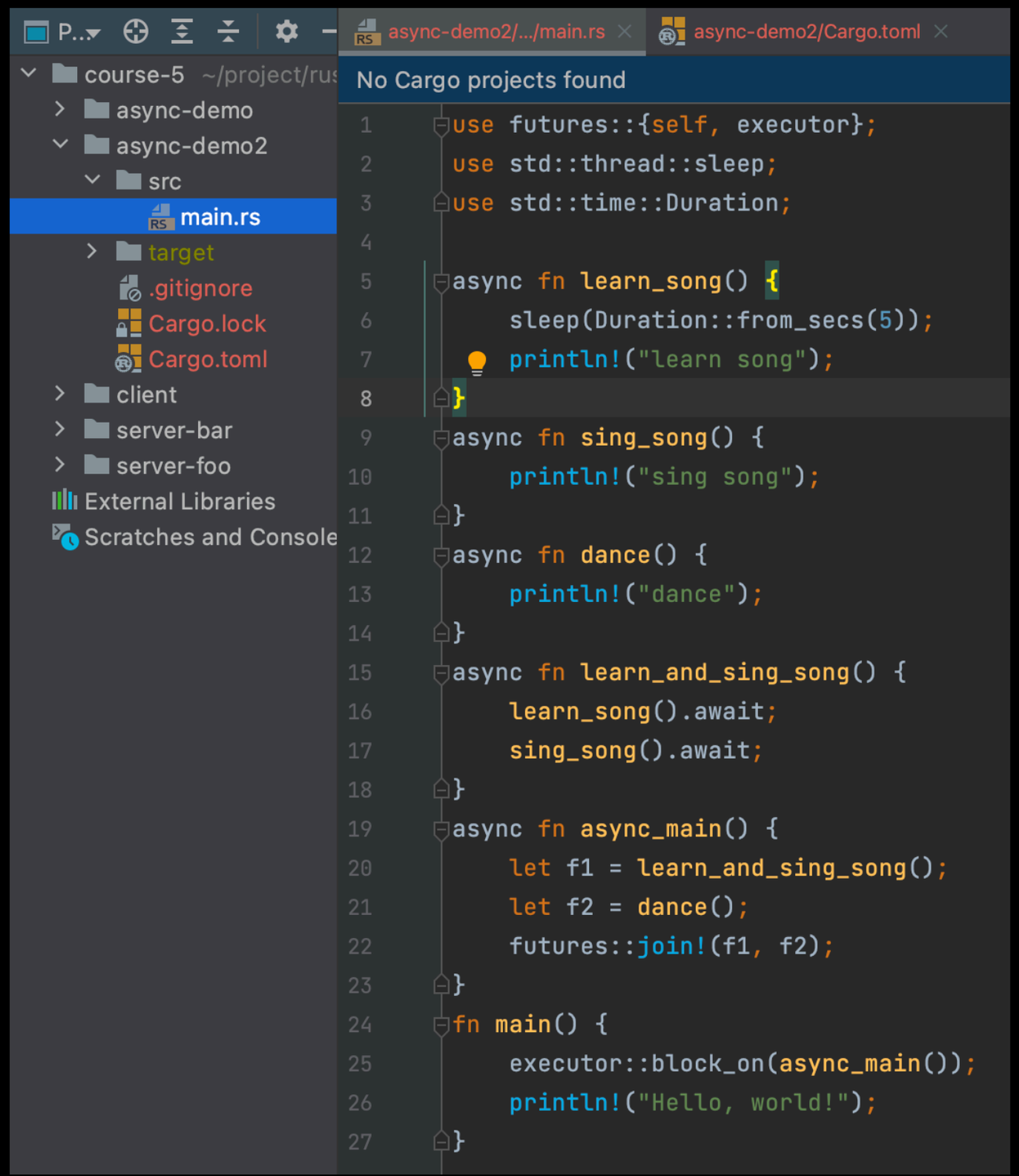


通过一段代码，了解async关键字的作用会



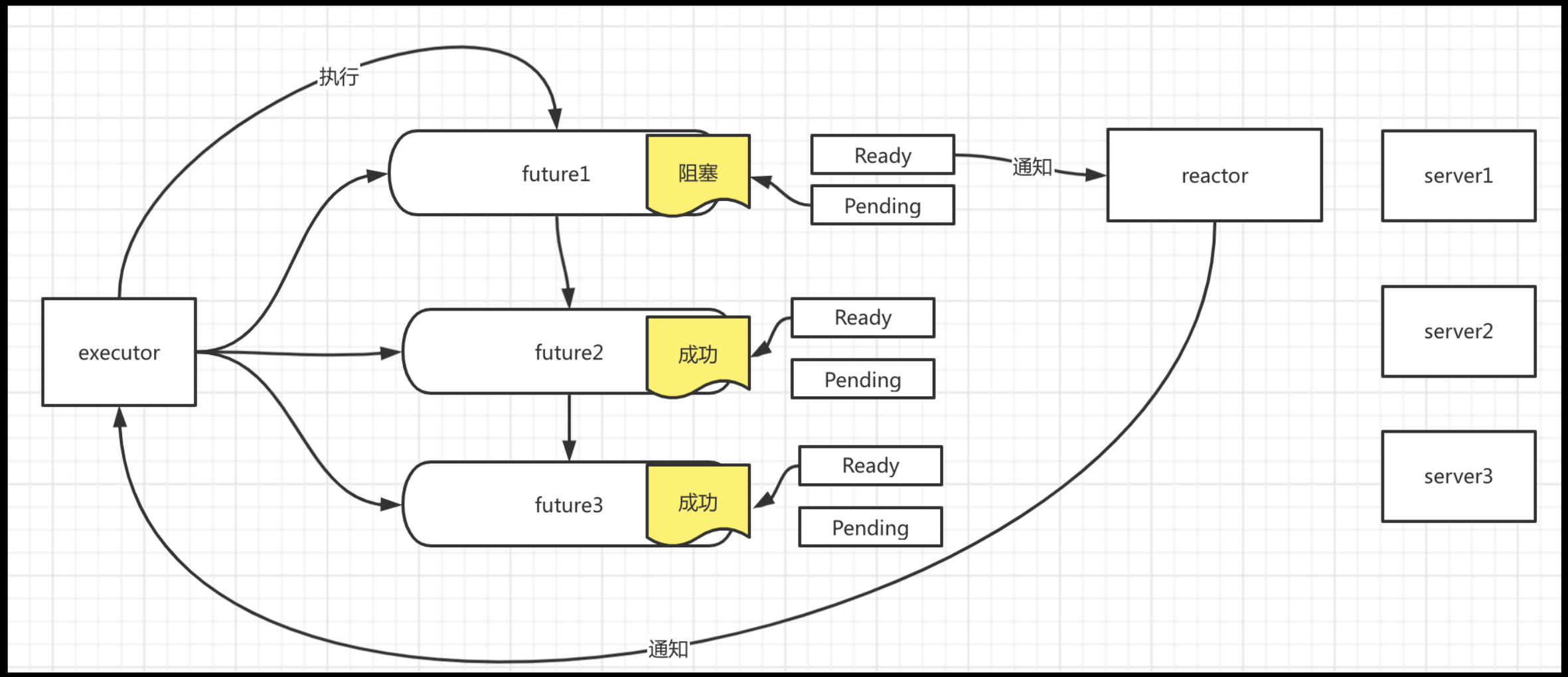
通过block_on来执行

通过await来执行


P...
course-5 ~/project/rus
async-demo
async-demo2
src
main.rs
target
.gitignore
Cargo.lock
Cargo.toml
client
server-bar
server-foo
External Libraries
Scratches and Console
async-demo2/.../main.rs
async-demo2/Cargo.toml
No Cargo projects found
use futures::{self, executor};
use std::thread::sleep;
use std::time::Duration;
async fn learn_song() {
sleep(Duration::from_secs(5));
println!("learn song");
}
async fn sing_song() {
println!("sing song");
}
async fn dance() {
println!("dance");
}
async fn learn_and_sing_song() {
learn_song().await;
sing_song().await;
}
async fn async_main() {
let f1 = learn_and_sing_song();
let f2 = dance();
futures::join!(f1, f2);
}
fn main() {
executor::block_on(async_main());
println!("Hello, world!");
}

join!语法

- 异步编程模型


执行
future1
阻塞
Ready
Pending
通知
reactor
server1
executor
future2
成功
Ready
Pending
server2
future3
成功
Ready
Pending
server3
通知

- 带领大家实现一个简化版的Rust异步编程框架

QA环节

加群一起交流Rust & Datafuse

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