CnosDB

Use Rust To Make A TSDB

rust入门基础（十二）

Lecturer: Segment

Date: 2022.08.10

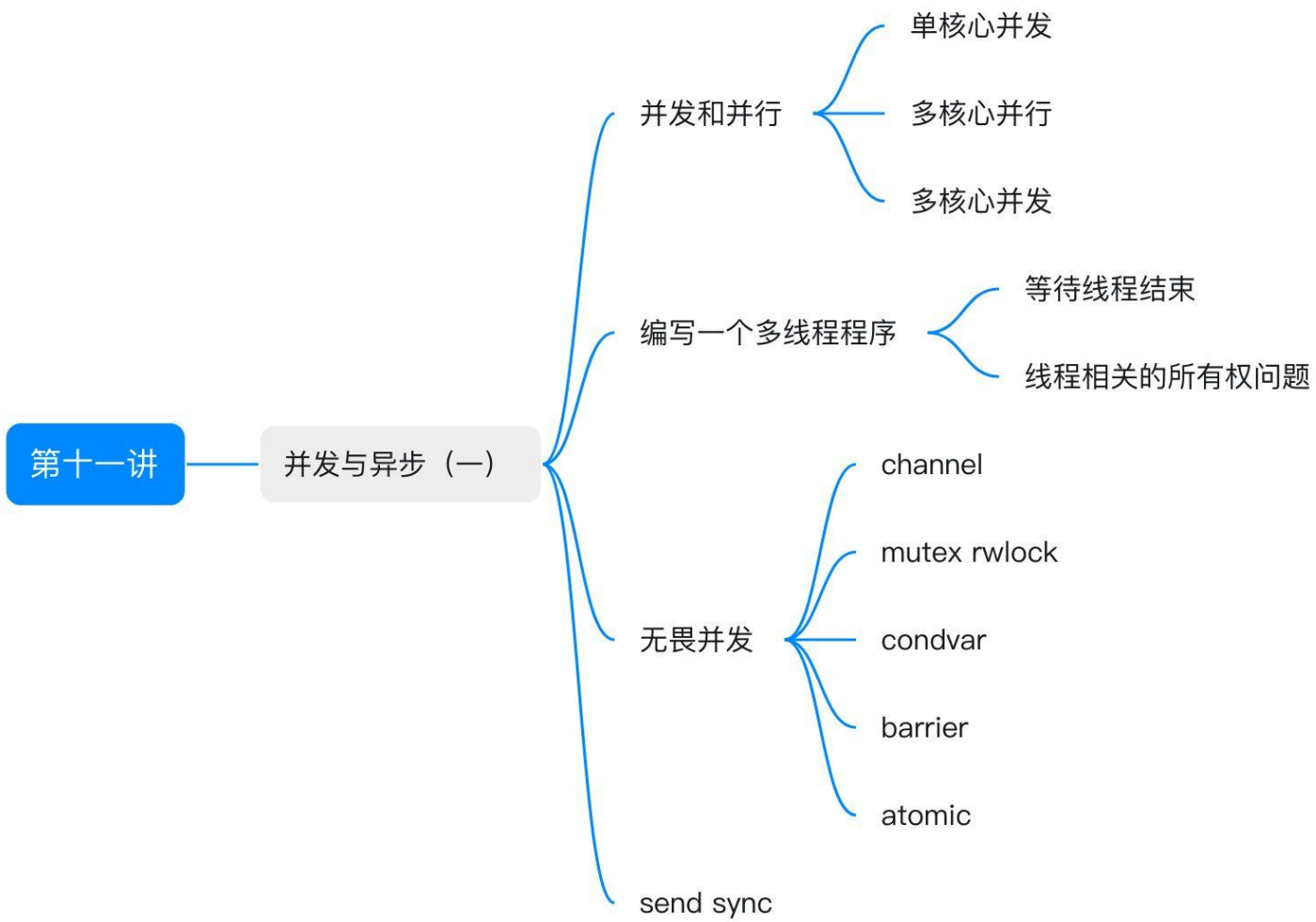
Welcome to follow the GitHub repo

欢迎关注我们的代码仓库

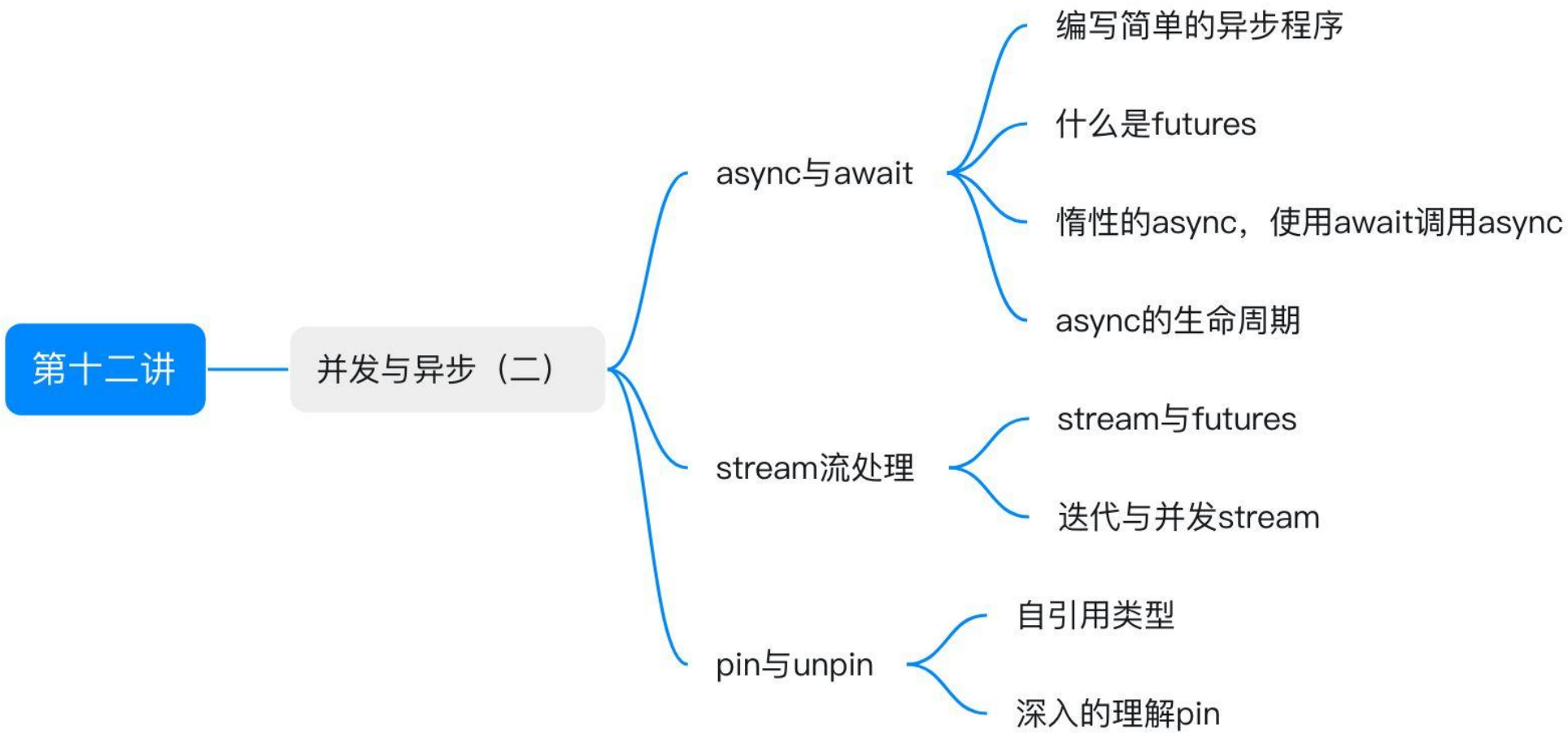
https://github.com/**cnosdb/cnosdb**

GitHub

回顾


第十一讲
并发与异步（一）
并发和并行
单核心并发
多核心并行
多核心并发
编写一个多线程程序
等待线程结束
线程相关的所有权问题
无畏并发
channel
mutex rwlock
condvar
barrier
atomic
send sync

并发与异步（二）


第十二讲
并发与异步（二）
async与await
编写简单的异步程序
什么是futures
惰性的async，使用await调用async
async的生命周期
stream流处理
stream与futures
迭代与并发stream
pin与unpin
自引用类型
深入的理解pin

Rust中的async

CnosDB

- Future在rust中是惰性的
- async在rust中没有开销
- rust并不是默认自带异步运行时
- 运行时同时支持单线程和多线程

Rust的Stream

CnosDB

Stream 特征类似于 Future 特征，但是前者在完成前可以生成多个值，这种行为跟标准库中的 Iterator 特征倒是颇为相似。

```
trait Stream {
    type Item;
    fn poll_next(self: Pin<&mut Self>, cx: &mut Context<'_>)
        -> Poll<Option<Self::Item>>;
}
```

Stream的并发

CnosDB

```
stream.for_each_concurrent(MAX_CONCURRENT_JUMPERS, |num| async move {
        jump_n_times(num).await?;
        report_n_jumps(num).await?;
        Ok(())
    }).await?;
```

自引用类型

CnosDB

- struct SelfRef {
- value: String,
- pointer_to_value: *mut String,
- }

异步中的自引用

CnosDB

```
async {
    let mut x = [0; 128];
    let read_into_buf_fut = read_into_buf(&mut x);
    read_into_buf_fut.await;
    println!("{:?}", x);
}
```

异步中的自引用

CnosDB

- struct ReadIntoBuf<'a> {
- buf: &'a mut [u8], // 指向下面的`x`字段
- }

- struct AsyncFuture {
- x: [u8; 128],
- read_into_buf_fut: ReadIntoBuf<'what_lifetime?>,
- }

CnosDB

Q&A

Welcome to follow the GitHub repo

欢迎关注我们的代码仓库

https://github.com/**cnosdb/cnosdb**

GitHub