

34 | 并发处理：从 atomics 到 Channel，Rust 都提供了什么工具？（下）

 time.geekbang.org/column/article/442217

陈天 · Rust 编程第一课

陈天

Tubi TV 研发副总裁

3669 人已学习

新人首单¥59

登录后，你可以任选4讲全文学习

推荐试读

换一换

开篇词 | 让Rust成为你的下一门主力语言

免费

06 | get hands dirty：SQL查询工具怎么一鱼多吃？

07 | 所有权：值的生杀大权到底在谁手上？

课程目录

已更新 40 讲/共 52 讲

开篇词 (1讲)

免费 开篇词 | 让Rust成为你的下一门主力语言

推荐试读

前置篇 (3讲)

01 | 内存：值放堆上还是放栈上，这是一个问题

02 | 串讲：编程开发中，那些你需要掌握的基本概念

加餐 | Rust真的值得我们花时间学习么？

基础篇 (21讲)

03 | 初窥门径：从你的第一个Rust程序开始！

04 | get hands dirty：来写个实用的CLI小工具

05 | get hands dirty：做一个图片服务器有多难？

06 | get hands dirty：SQL查询工具怎么一鱼多吃？

推荐试读

07 | 所有权：值的生杀大权到底在谁手上？

推荐试读

08 | 所有权：值的借用是如何工作的？

09 | 所有权：一个值可以有多个所有者么？

10 | 生命周期：你创建的值究竟能活多久？

推荐试读

11 | 内存管理：从创建到消亡，值都经历了什么？

加餐 | 愚昧之巅：你的Rust学习常见问题汇总

12 | 类型系统：Rust的类型系统有什么特点？

推荐试读

13 | 类型系统：如何使用trait来定义接口？

推荐试读

14 | 类型系统：有哪些必须掌握的Trait？

15 | 数据结构：这些浓眉大眼的结构竟然都是智能指针？

16 | 数据结构：Vec<T>、&[T]、Box<[T]>，你真的了解集合容器么？

17 | 数据结构：软件系统核心部件哈希表，内存如何布局？

18 | 错误处理：为什么Rust的错误处理与众不同？

19 | 闭包：FnOnce、FnMut 和 Fn，为什么有这么多类型？

20 | 4 Steps：如何更好地阅读Rust源码？

21 | 阶段实操：构建一个简单的 KV server (1) - 基本流程（上）

22 | 阶段实操：构建一个简单的 KV server (1) -基本流程（下）

期中周 (2讲)

加餐 | 期中测试：来写一个简单的 grep 命令行

加餐 | 期中测试：参考实现讲解

进阶篇 (11讲)

23 | 类型系统：如何在实战中使用泛型编程？

24 | 类型系统：如何在实战中使用 Trait Object？

25 | 类型系统：如何围绕 Trait 来设计和架构系统？

加餐 | Rust 2021 版次问世了！

26 | 阶段实操：构建一个简单的 KV server (2) - 高级 trait 技巧

27 | 生态系统：有哪些常有的 Rust 库可以为我所用？

28 | 网络开发：如何使用 Rust 处理网络请求？（上）

29 | 网络开发：如何使用 Rust 处理网络请求？（下）

30 | Unsafe Rust：如何用 C++ 的方式打开 Rust？

31 | FFI：Rust 如何和你的语言架起沟通桥梁？

32 | 实操项目：使用 PyO3 开发 Python3 模块

并发篇 (2讲)

33 | 并发处理：从 atomics 到 Channel，Rust 都提供了什么工具？（上）

34 | 并发处理：从 atomics 到 Channel，Rust 都提供了什么工具？（下）

陈天 · Rust 编程第一课

15

1.0x

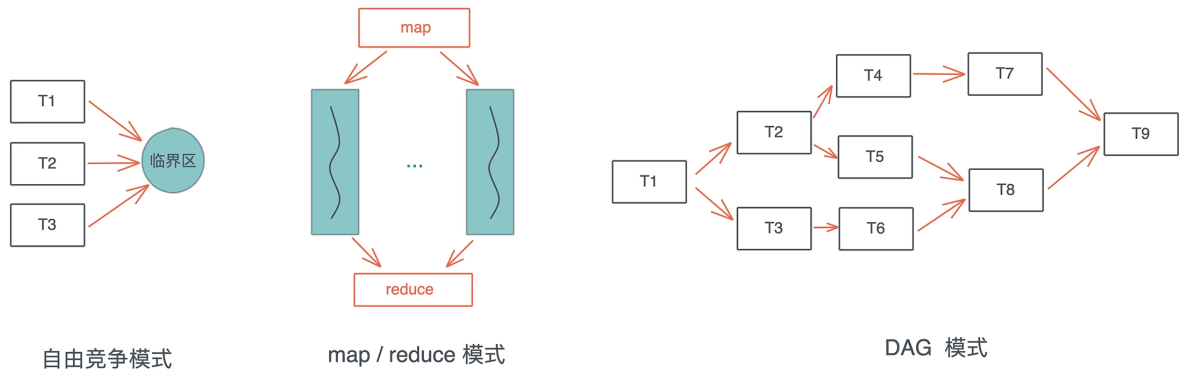
[登录](#) | [注册](#)

陈天 2021-11-15



你好，我是陈天。

对于并发状态下这三种常见的工作模式：自由竞争模式、map/reduce 模式、DAG 模式，我们的难点是如何在这些并发的任务中进行同步。atomic / Mutex 解决了自由竞争模式下并发任务的同步问题，也能够很好地解决 map/reduce 模式下的同步问题，因为此时同步只发生在 map 和 reduce 两个阶段。



然而，它们没有解决一个更高层次的问题，也就是 DAG 模式：如果这种访问需要按照一定顺序进行或者前后有依赖关系，该怎么做？

这个问题的典型场景是生产者 - 消费者模式：生产者生产出来内容后，需要有机制通知消费者可以消费。比如 socket 上有数据了，通知处理线程来处理数据，处理完成之后，再通知 socket 收发的线程发送数据。

Condvar

所以，操作系统还提供了 Condvar。Condvar 有两种状态：

等待 (wait)：线程在队列中等待，直到满足某个条件。

通知 (notify)：当 condvar 的条件满足时，当前线程通知其他等待的线程可以被唤醒。通知可以是单个通知，也可以是多个通知，甚至广播（通知所有人）。

在实践中，Condvar 往往和 Mutex 一起使用：Mutex 用于保证条件在读写时互斥，Condvar 用于控制线程的等待和唤醒。我们来看一个例子：

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

开篇词 | 让 Rust 成为你的下一门主力语言

免费

06 | get hands dirty: SQL 查询工具怎么一鱼多吃？

07 | 所有权：值的生杀大权到底在谁手上？

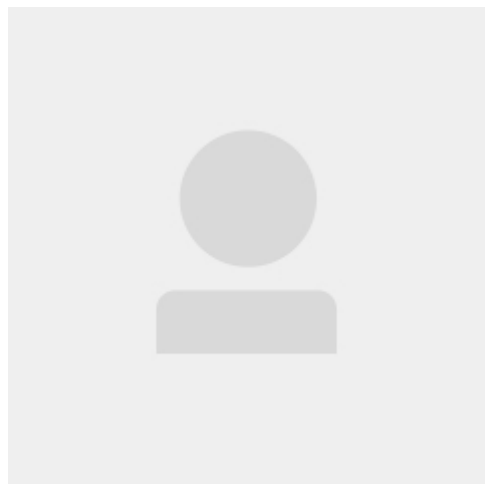
10 | 生命周期：你创建的值究竟能活多久？

12 | 类型系统：Rust 的类型系统有什么特点？

13 | 类型系统：如何使用trait来定义接口？

该试读文章来自付费专栏《陈天·Rust 编程第一课》，如需阅读全部文章，
请订阅文章所属专栏，新人首单¥59

立即订阅



登录 后留言

精选留言(3)



终生侧隐

```
#[test]
fn test_mpsc() {
    let (a2btx, a2brx) = mpsc::channel();
    let (b2atx, b2arx) = mpsc::channel();

    let threada = thread::spawn(move || {
        a2btx.send("hello world!".to_string()).unwrap();
        for re in b2arx {
            println!("{}", re);
            thread::sleep(Duration::from_secs(1));
            a2btx.send("hello world!".to_string()).unwrap();
        }
    });

    let threadb = thread::spawn(move || {
        for re in a2brx {
            println!("{}", re);
            thread::sleep(Duration::from_secs(1));
            b2atx.send("goodbye!".to_string()).unwrap();
        }
    });

    thread::sleep(Duration::from_secs(10));
    return
}
```

2021-11-15



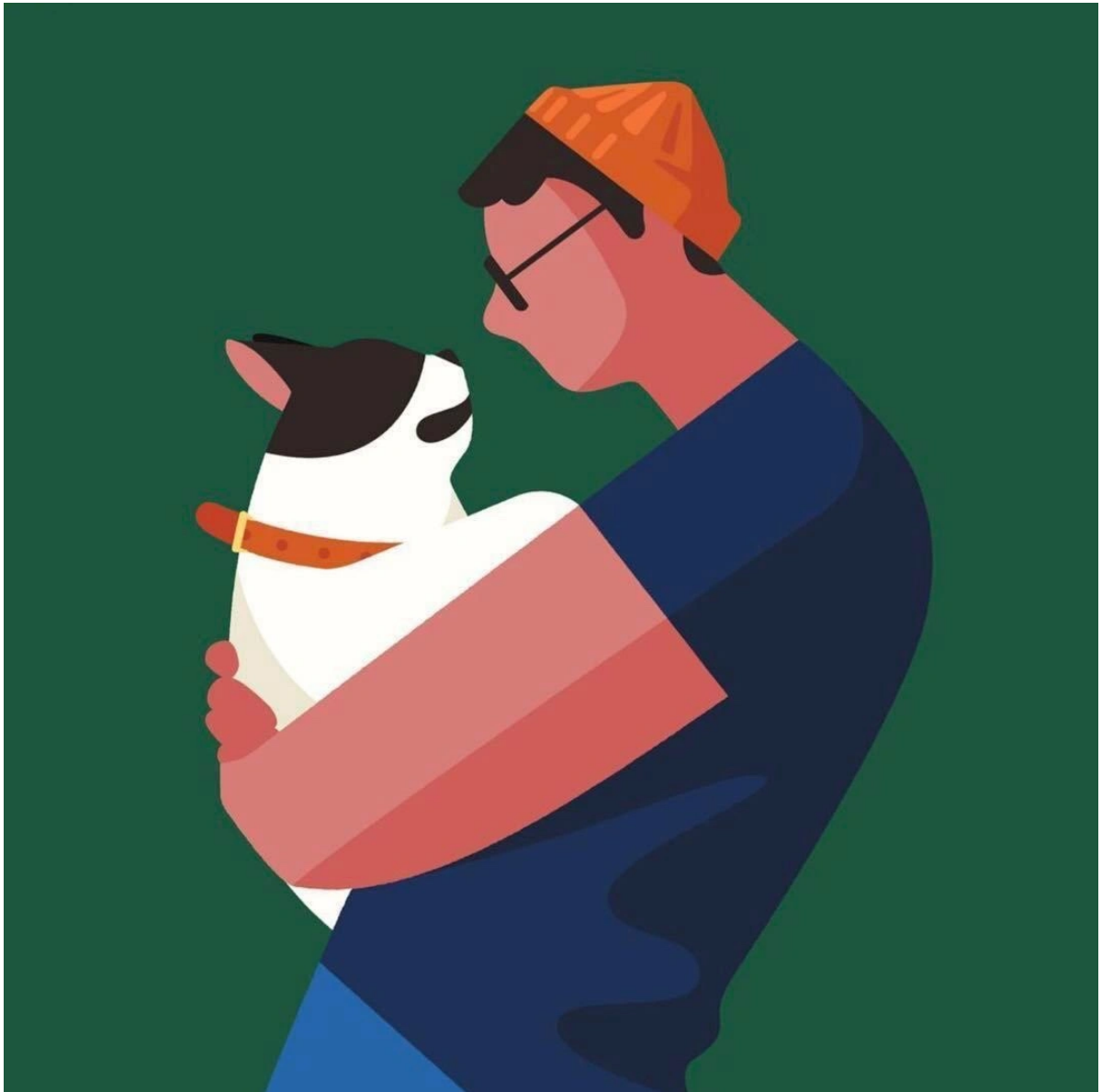
罗杰



对，合理的使用 Channel，不应该死搬硬套。

2021-11-15

•



GengTeng

笨拙地用 Channel 叠加 Channel 来应对所有的场景?Go: 你直接说我名儿得了。

2021-11-15

1

收起评论