## 优质IT资源微信x92%3713



## 《手写OS操作系统》小班二期招生,全程直播授课,大牛带你掌握硬核技术!



慕课网首页

免费课

实战课

体系课

慕课教程

专栏

手记 企业服务

Q

. .

我自

쿻

②

|.|

0

从所有教程的词条中查询…

首页 > 慕课教程 > Go工程师体系课全新版 > 5. 基于redis实现分布式锁

## 全部开发者教程 ::= 不一致? 3. CAP和BASE理论 4. 两\_三阶段提交 5. tcc分布式事务

7. 基于可靠消息的最终一致性- 最常用

6. 基于本地消息表的最终一致

8. 最大努力通知

9. mq (message queue) 的 使用场景

10. mq技术选型

11. rocketmq安装和配置

12. rocketmq的基本概念

13. rocketmq的消息类型

```
bobby・更新于 2022-11-16
                                                 ↓ 上一节 4. mysgl的for u...6. redlock详解 下一节 →
redsync项目地址
 <> 代码块
      package main
  1
  2
       import (
  3
          goredislib "github.com/go-redis/redis/v8"
  4
           "github.com/go-redsync/redsync/v4"
  5
           "github.com/go-redsync/redsync/v4/redis/goredis/v8"
  6
  7
  R
  9
      func main() {
          // Create a pool with go-redis (or redigo) which is the pool redisync will
 10
          // use while communicating with Redis. This can also be any pool that
 11
          // implements the `redis.Pool` interface.
 12
          client := goredislib.NewClient(&goredislib.Options{
 13
               Addr: "localhost:6379",
 14
          })
 15
          pool := goredis.NewPool(client) // or, pool := redigo.NewPool(...)
 16
 17
          // Create an instance of redisync to be used to obtain a mutual exclusion
 18
          // lock.
 19
           rs := redsync.New(pool)
 20
 21
          // Obtain a new mutex by using the same name for all instances wanting the
 22
 23
          // same lock.
          mutexname := "my-global-mutex"
          mutex := rs.NewMutex(mutexname)
 25
 26
          // Obtain a lock for our given mutex. After this is successful, no one else
 27
           // can obtain the same lock (the same mutex name) until we unlock it.
 28
          if err := mutex.Lock(); err != nil {
 29
 30
               panic(err)
 31
 32
          // Do your work that requires the lock.
 33
 34
           // Release the lock so other processes or threads can obtain a lock.
 35
           if ok, err := mutex.Unlock(); !ok || err != nil {
 36
               panic("unlock failed")
 37
 38
 39
      }
```

redsync源码解读





□ 标记书签



## 优质IT资源微信。2023713

将获取和设置值变成原子性的操作

- 2. 如果我的服务挂掉了- 死锁
  - 1. 设置过期时间
  - 2. 如果你设置了过期时间,那么如果过期时间到了我的业务逻辑没有执行完怎么办?
    - 1. 在过期之前刷新一下
    - 2. 需要自己去启动协程完成延时的工作
      - 1. 延时的接口可能会带来负面影响 如果其中某一个服务hung住了, 2s就能执行完,但 是你hung住那么你就会一直去申请延长锁,导致别人永远获取不到锁,这个很要命
- 3. 分布锁需要解决的问题 lua脚本去做
  - 1. 互斥性 setnx
  - 2. 死锁
  - 3. 安全性
    - 1. 锁只能被持有该锁的用户删除,不能被其他用户删除
      - 1. 当时设置的value值是多少只有当时的g才能知道
      - 2. 在删除的时取出redis中的值和当前自己保存下来的值对比一下
- 4. 即使你这样实现了分布式但是还是会有问题 redlock

✔ 我要提出意见反馈

企业服务 网站地图 网站首页 关于我们 联系我们 讲师招募 帮助中心 意见反馈 代码托管

Copyright © 2022 imooc.com All Rights Reserved | 京ICP备 12003892号-11 京公网安备11010802030151号

⊡

?

 $\Box$ 

0