bobby · 更新于 2022-11-16

上一节 4. mysql的for u... 6. redlock详解 下一节

redsync项目地址

代码块

```
package main

import (
    goredislib "github.com/go-redis/redis/v8"
    "github.com/go-redsync/redsync/v4"
    "github.com/go-redsync/redsync/v4/redis/goredis/v8"
)

func main() {
    // Create a pool with go-redis (or redigo) which is the pool redisync will
    // use while communicating with Redis. This can also be any pool that
    // implements the `redis.Pool` interface.
    client := goredislib.NewClient(&goredislib.Options{
        Addr: "localhost:6379",
    })
    pool := goredis.NewPool(client) // or, pool := redigo.NewPool(...)

    // Create an instance of redisync to be used to obtain a mutual exclusion
    // lock.
    rs := redsync.New(pool)

    // Obtain a new mutex by using the same name for all instances wanting the
    // same lock.
    mutexname := "my-global-mutex"
    mutex := rs.NewMutex(mutexname)

    // Obtain a lock for our given mutex. After this is successful, no one else
    // can obtain the same lock (the same mutex name) until we unlock it.
    if err := mutex.Lock(); err != nil {
        panic(err)
    }

    // Do your work that requires the lock.

    // Release the lock so other processes or threads can obtain a lock.
    if ok, err := mutex.Unlock(); !ok || err != nil {
        panic("unlock failed")
    }
}
```

redsync源码解读

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

4. mysql的for update实现悲观锁 ◂ 上一节 下一节 ▸ 6. redlock详解

我要提出意见反馈