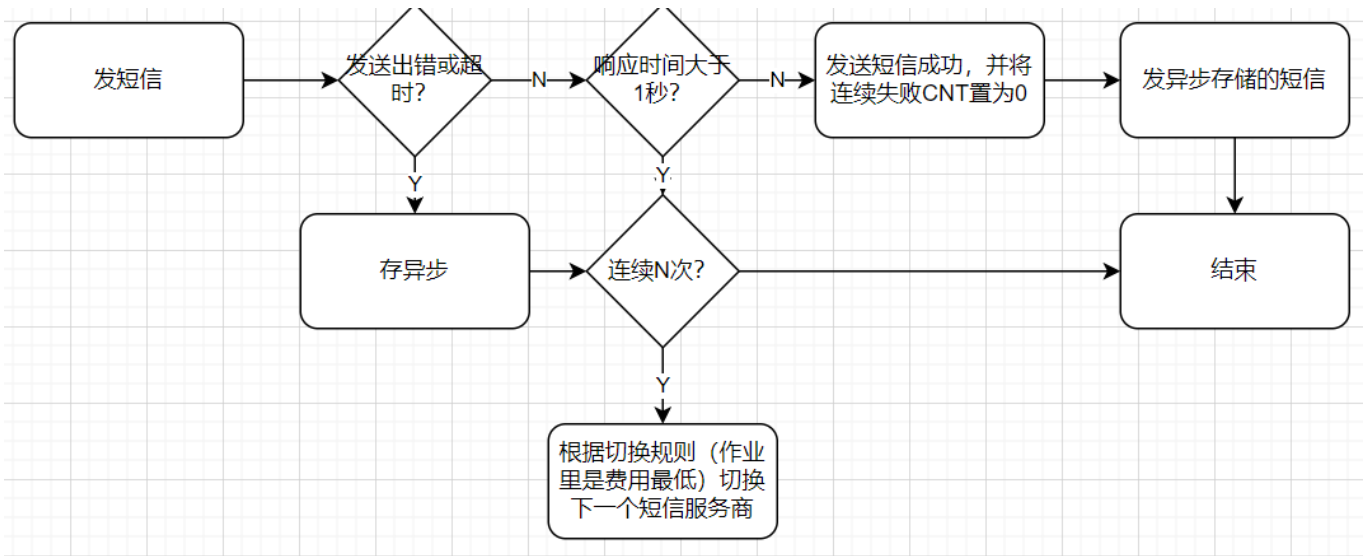


1，设计流程图



2，判断服务商崩溃的标准

符合以下任意条件即认定该次发送失败，将该次请求存到数据库中。如果连续失败N次，则切换服务商：

- 发送超时
- 发送失败
- 响应时间超过1秒
- 限流超出

```
func (s *ExtensiveFailoverSMSService) Send(ctx context.Context, biz string, args []string, numbers
...string) error {
    //先看限流
    limited, err := s.limiter.Limit(ctx, "sms:cnt")
    if err != nil {
        return fmt.Errorf("短信服务判断是否限流出现问题, %w", err)
    }
    if limited {
        // 转异步
        s.repo.Create(ctx, domain.SMSAsync{
            Biz:    biz,
            Args:   strings.Join(args, "###"),
            Numbers: strings.Join(numbers, ";"),
            Ctime:  time.Now(),
        })
        return errLimited
    }

    idx := atomic.LoadInt32(&s.idx)
    cnt := atomic.LoadInt32(&s.cnt)
    if cnt > s.threshold {
        // 这里要切换, 根据切换规则切换下一个服务商
        newIdx, err := s.SwitchNext(ctx, idx)
```

```

if err != nil {
    return err
}
if atomic.CompareAndSwapInt32(&s.idx, idx, newIdx) {
    // 我成功往后挪了一位
    atomic.StoreInt32(&s.cnt, 0)
}
// else 就是出现并发，别人换成功了
// 这个时候只需要重新Load idx就可以了
idx = atomic.LoadInt32(&s.idx)
}

var svc async.SMSService = s.svcs[idx]
t1 := time.Now()
err = svc.Send(ctx, biz, args, numbers...)
t2 := time.Now()
switch err {
case context.DeadlineExceeded:
    atomic.AddInt32(&s.cnt, 1)
    //转异步
    return err
case nil:
    // 该短信服务商cnt要加一
    atomic.AddInt32(s.feeSvcs[idx].GetCnt(), 1)
    // 响应时间超过设定的时长也需要加一
    if t2.Sub(t1) > time.Duration(s.resThreshold)*time.Second {
        atomic.AddInt32(&s.cnt, 1)
        //但是这里不用转异步，因为已经发出去了
    } else {
        // 你的连续状态被打断了
        atomic.StoreInt32(&s.cnt, 0)
        // 发送异步存储的短信
        svc.StartAsync(ctx)
    }

    return nil
default:
    // 不知道什么错误
    // 你可以考虑，换下一个，语义则是：
    // - 超时错误，可能是偶发的，我尽量再试试
    // - 非超时，我直接下一个
    atomic.AddInt32(&s.cnt, 1)
    //转异步
    return err
}
}
}

```

3，切换服务商

新建一个FeeService, 计算每个服务商发单条短信的费用（假定服务商根据发送条数来计算单条短信的费用，发得越多越便宜），在切换下一个服务商的时候会比较剩下的服务商的短信费率，找到费率最低的服务商。

AliFeeService的例子：

```

import (
    "context"
    "errors"
    "github.com/bolognagene/geektime-gocamp/geektime-gocamp/webook/webook/internal/service/sms"
)

type AliyunFeeService struct {
    cnt  int32 // 该短信服务商发送短信计数
    name string // 服务商名字
}

func NewAliyunFeeService() sms.FeeService {
    return &AliyunFeeService{}
}

func (s *AliyunFeeService) Fee(ctx context.Context, args ...any) (float32, error) {
    // 阿里云根据短信发送数量来计算每条的费用
    cnt, ok := args[0].(*int32)

    if !ok {
        return 1.0, errors.New("输入参数类型不对")
    }

    var fee float32
    if *cnt < 100000 {
        fee = 0.045
    } else if *cnt < 300000 {
        fee = 0.04
    } else if *cnt < 500000 {
        fee = 0.039
    } else if *cnt < 1000000 {
        fee = 0.038
    } else if *cnt < 3000000 {
        fee = 0.037
    } else {
        fee = 0.036
    }

    return fee, nil
}

func (s *AliyunFeeService) GetName() string {
    return s.name
}

func (s *AliyunFeeService) GetCnt() *int32 {
    return &s.cnt
}

```

切换下一个服务商：

```

// SwitchNext
// 根据费用切换到下一个
func (s *ExtensiveFailoverSMSService) SwitchNext(ctx context.Context, idx int32) (int32, error) {
    // 待写
    //return (idx + 1) % int32(len(f.svcs)), nil
    var i int32

```

```

idxAndFees := make([]idxAndFee, len(s.feeSvcs))

for i = 0; i < int32(len(s.feeSvcs)); i++ {

    fee, err := s.feeSvcs[i].Fee(ctx, s.feeSvcs[i].GetCnt())
    if err != nil {
        fee = 1.0 //自动排到最后
    }

    idxAndFees[i] = idxAndFee{
        idx: i,
        fee: fee,
    }
}

sort.Slice(idxAndFees, func(i, j int) bool {
    return idxAndFees[i].fee < idxAndFees[j].fee
})

for i = 0; i < int32(len(idxAndFees)); i++ {
    if idxAndFees[i].idx == idx {
        nextIdx := (i + 1) % int32(len(idxAndFees))
        return idxAndFees[nextIdx].idx, nil
    }
}

return (idx + 1) % int32(len(s.svcs)), nil
}

```

4，发送异步短信

当成功发送一条短信的时候，证明短信服务商已能正常工作（切换成功），这个时候会将异步存储的请求启一个goroutine发送出去，发送成功的会从数据库里删掉，发送失败的则不变，等待下一次发送。

发送短信成功时调用StartAsync

```

func (s *SMSService) Send(ctx context.Context, biz string, args []string, numbers
...string) error {
.....
case nil:
    // 该短信服务商cnt要加一
    atomic.AddInt32(s.feeSvcs[idx].GetCnt(), 1)
    // 响应时间超过设定的时长也需要加一
    if t2.Sub(t1) > time.Duration(s.resThreshold)*time.Second {
        atomic.AddInt32(&s.cnt, 1)
        //但是这里不用转异步，因为已经发出去了
    } else {
        // 你的连续状态被打断了
        atomic.StoreInt32(&s.cnt, 0)
        // 发送异步存储的短信
        svc.StartAsync(ctx)
    }
}
}

```

StartAsync:

```
func (s *SMSService) StartAsync(ctx context.Context) {
    go func() {
        smsAsyncs, err := s.repo.FindAll(ctx)
        if err != nil {
            return
        }
        for _, smsAsync := range smsAsyncs {
            // 在这里发送, 并且控制重试
            var nums []string
            numbers := strings.Split(smsAsync.Numbers, ";")
            for _, number := range numbers {
                nums = append(nums, number)
            }

            cnt := 0
            for cnt < s.retryMax {
                err = s.svc.Send(ctx, smsAsync.Biz, strings.Split(smsAsync.Args, "###"),
                    nums...)
                if err == nil {
                    // 发送成功删除记录
                    s.repo.Remove(ctx, smsAsync)
                    break
                }
                cnt++
            }
        }
    }()
}
```

5, 初始化

在ioc的sms.go里InitSMSService, 初始化Ali, tencent和cloopen的短信服务(这里用的memory模拟的), 然后初始化ExtensiveFailoverSMSService.

```
// 由于我的阿里云、腾讯云启动不成功(没有申请相关的数据)因此这里用memory来模拟阿里、腾讯和容联云的短信服务
func InitSMSService(cmd redis.Cmdable, repo repository.SMSAsyncRepository) sms.Service {
    // 换内存, 还是换别的
    //svc := ratelimit.NewRatelimitSMSService(memory.NewService(),
    // limiter.NewRedisSlidingWindowLimiter(cmd, time.Second, 100))
    //return retryable.NewService(svc, 3)
    //return memory.NewService()

    // aliyun服务
    aliService := memory.NewAliyunService()
    asyncAliService := async.NewSMSService(aliService, repo, 3)
    aliFeeService := memory.NewAliyunFeeService()

    // tencent服务
    tencentService := memory.NewTencentService()
    asyncTencentService := async.NewSMSService(tencentService, repo, 3)
    tencentFeeService := memory.NewTencentFeeService()
}
```

```
// 容联云服务
cloopenService := memory.NewCloopenService()
asyncCloopenService := async.NewSMSService(cloopenService, repo, 3)
cloopenFeeService := memory.NewCloopenFeeService()

//限流服务
limiter := ratelimit.NewRedisSlidingWindowLimiter(cmd, 1*time.Second, 3000)

//组装failoverService
var smses []async.SMSService
var fees []sms.FeeService
smses = append(smses, *asyncAliService, *asyncTencentService, *asyncCloopenService)
fees = append(fees, aliFeeService, tencentFeeService, cloopenFeeService)
return failover.NewExtensiveFailoverSMSService(smses, fees, limiter, repo, 2, 1)
}
```