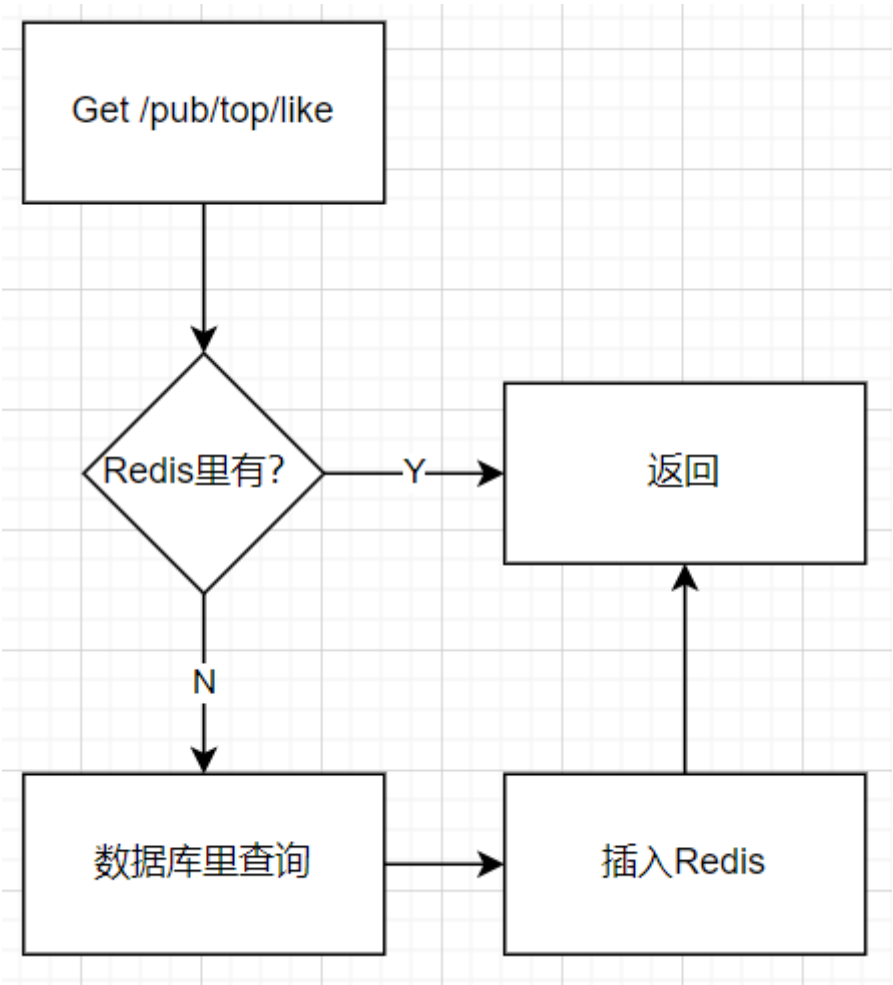
1，设计思路

用一个Redis的ZSet来维持一个TopLike的文章列表，在这个ZSet没到期的时间内，对文章点赞，如果该文章也在ZSet里，也会对ZSet里的该文章的Score加一，这样就不用每次查询数据库了。

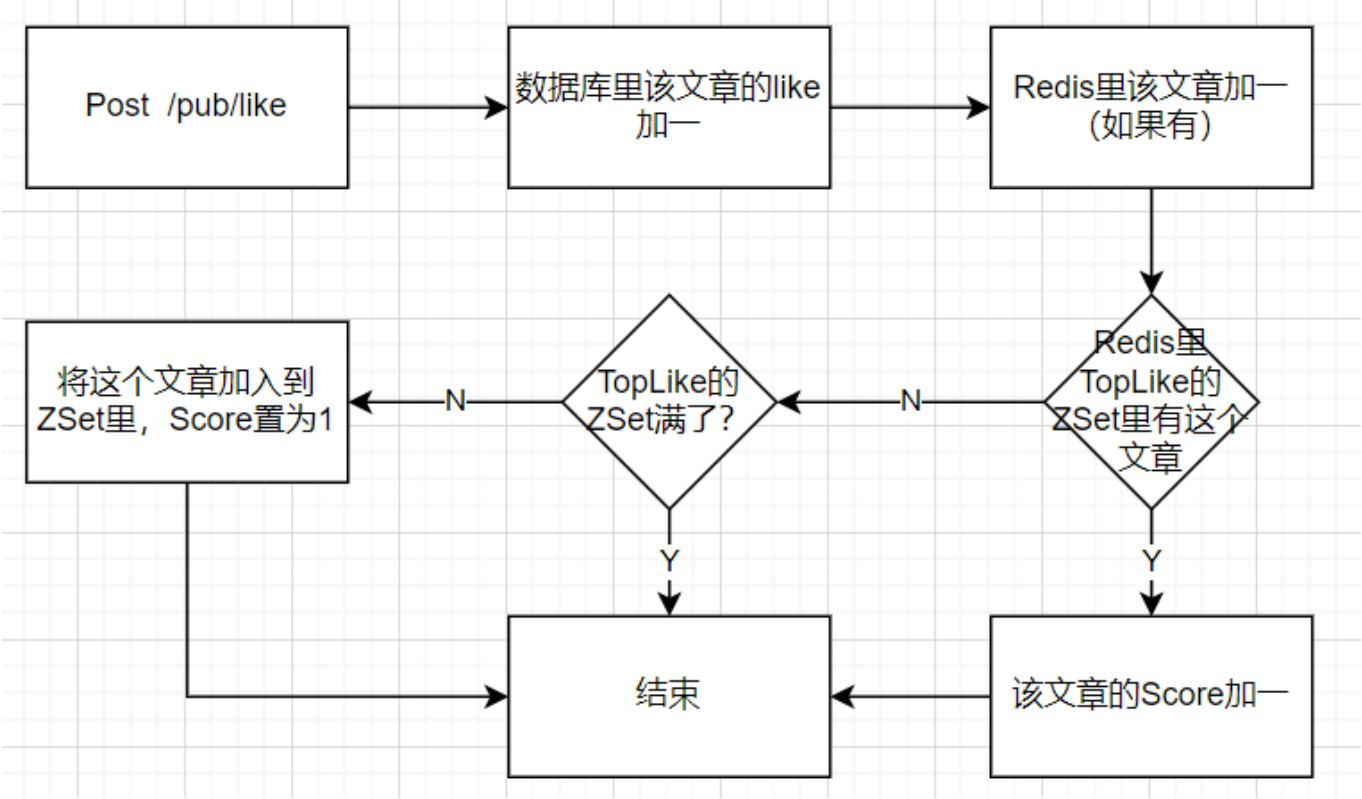
对该ZSet的Key到期时间进行订阅，如果到期，会触发再从数据库里计算一次Toplike的文章列表和点赞数，并写入Redis里。

这里可以设置TopLike的ZSet永不过期，然后用一个定时任务来定时从数据库里计算Toplike，但这里是为了学习如果订阅Key到期事件这么设计的

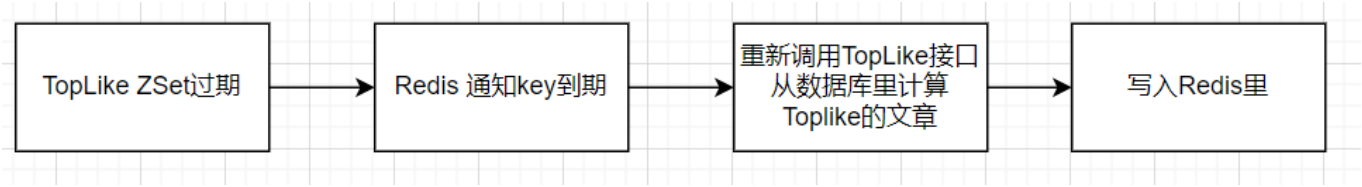
- 查询Top Like文章的流程：



- 给某文章点赞时，除了常规的计数加一，还要在TopLike的ZSet里加一



- ZSet过期时



2，代码设计

2.1，查询Top Like文章

ArticleHandler里注册路由，并调用Handler层的TopLike函数 （其中N是要返回多少个文章，比如Like前十的文章，那么N是10；Limit是一个扩大量，比如为了返回前十的文章，我查询100个文章，然后在这个周期我只统计这100个文章的Like变化。N和Limit的值可以在配置文件中更改。）

```
gpub.GET("/top/like", ginx.WrapFunc(h.TopLike, "TopLikeArticle", h.l))
......
func (h *ArticleHandler) TopLike(ctx *gin.Context) (ginx.Result, error) {

 data, err := h.interSvc.TopLike(ctx, h.biz, TopLikeN.Load(), TopLikeLimit.Load())

 if err != nil {
  return ginx.Result{
   Code: codes.ArticleInternalServerError,
   Msg:  "系统错误",
  }, err
 }

 return ginx.Result{
  Code: codes.ArticleOK,
  Msg:  "OK",
  Data: data,
 }, nil
}
```

InteractiveService层的TopLike函数，直接调用repo层的TopLike函数

```
func (svc *interactiveService) TopLike(ctx context.Context, biz string, n,
limit int64) ([]domain.TopWithScore, error) {
 return svc.repo.GetTopLike(ctx, biz, n, limit)
}
```

CachedInteractiveRepository的TopLike函数，先查询Redis，没有则查询数据库，然后回写到缓存

```
func (repo *CachedInteractiveRepository) GetTopLike(ctx context.Context, biz string, n int64,
limit int64) ([]domain.TopWithScore, error) {
 // 从缓存里读取TopLikeN
 data, err := repo.cache.GetTopLike(ctx, biz, n)

 if err == nil && data != nil {
  return data, nil
 }

 // 否则从数据库里查询
 intrs, err := repo.dao.GetTopLike(ctx, biz, limit)
 if err != nil {
  return nil, err
 }

 data = make([]domain.TopWithScore, len(intrs))
 for i, z := range intrs {
  data[i] = repo.ToTopWithScore(z)
 }

 go func() {
  // 回写到缓存中
  err1 := repo.cache.SetTopLike(ctx, biz, data)
  if err1 != nil {
   repo.l.Debug("设置toplike缓存失败", logger.String("biz", biz),
    logger.Error(err1))
  }
 }()

 if int64(len(data)) > n {
  return data[:n], nil
 }
 return data, nil

}
```

数据库的查询

```
func (dao *GORMInteractiveDAO) GetTopLike(ctx context.Context, biz string,
limit int64) ([]Interactive, error) {
 var data []Interactive
 err := dao.db.WithContext(ctx).Model(&Interactive{}).
  Where("biz = ?", biz).Limit(int(limit)).Order("like_cnt DESC").
  Find(&data).Error

 return data, err
}
```

从Redis里得到ZSet

```
func (r *RedisInteractiveCache) GetTopLike(ctx context.Context, biz string,
n int64) ([]domain.TopWithScore, error) {
 data, err := r.client.ZRevRangeWithScores(ctx, fmt.Sprintf("top_like_%s", biz), 0, n).Result()
 if err == nil && len(data) != 0 {
  ts := make([]domain.TopWithScore, n)
  for i, z := range data {
   tws, _ := r.ToTopWithScore(z)
   ts[i] = tws
  }
  return ts, err
 }
 return nil, err
}
```

回写到Redis

```
func (r *RedisInteractiveCache) SetTopLike(ctx context.Context, biz string, intrs
[]domain.TopWithScore) error {
 zs := make([]redis.Z, len(intrs))
 for i, intr := range intrs {
  zs[i] = r.ToRedisZ(intr)
 }

 zargs := redis.ZAddArgs{
  Members: zs,
 }

 err := r.client.ZAddArgs(ctx, fmt.Sprintf("top_like_%s", biz), zargs).Err()
 if err != nil {
  return err
 }

 return r.client.Expire(ctx, fmt.Sprintf("top_like_%s", biz), time.Minute*30).Err()
}
```

2.2，给文章点赞

在CachedInteractiveRepository里启动一个goroutine增加对TopLike的ZSet的操作

```
func (repo *CachedInteractiveRepository) IncrLike(ctx context.Context, biz string, bizId, uid,
limit int64) error {
 // 先插入点赞，然后更新点赞计数，更新缓存
 err := repo.dao.InsertLikeInfo(ctx, biz, bizId, uid)
 if err != nil {
  return err
 }

 // 这种做法，你需要在 repository 层面上维持住事务
 go func() {
  ret, err1 := repo.cache.IncrLikeCntIfPresent(ctx, biz, bizId)
  if err1 != nil {
   repo.l.Debug("增加点赞计数失败", logger.String("biz", biz),
    logger.Int64("bizId", bizId), logger.Error(err1))
  }

  fmt.Sprintf("ret is %d", ret)
 }()

 // 增加Toplike里的计数
 go func() {
  ret2, err2 := repo.cache.IncrTopLike(ctx, biz, bizId, limit)
  if err2 != nil {
   repo.l.Debug("增加TopLike计数失败，可能是该文章不在TopLike中", logger.String("biz", biz),
    logger.Int64("bizId", bizId), logger.Error(err2))
  }

  fmt.Sprintf("ret is %d", ret2)
 }()

 return nil
}
```

Cache里的IncrTopLike函数

```
func (r *RedisInteractiveCache) IncrTopLike(ctx context.Context, biz string, bizId int64,
limit int64) (int, error) {
 return r.client.Eval(ctx, luaIncrTopLike,
  []string{fmt.Sprintf("top_like_%s", biz)},
  1, bizId, limit).Int()
}
```

调用的是luaIncrTopLike脚本，

1，ZCARD判断ZSet (key为top*_like_article*)是否存在，如果不存在返回4；

2，ZSCORE判断文章是否存在，如果存在对Like进行加一或者减一 （ZINCRBY）返回3

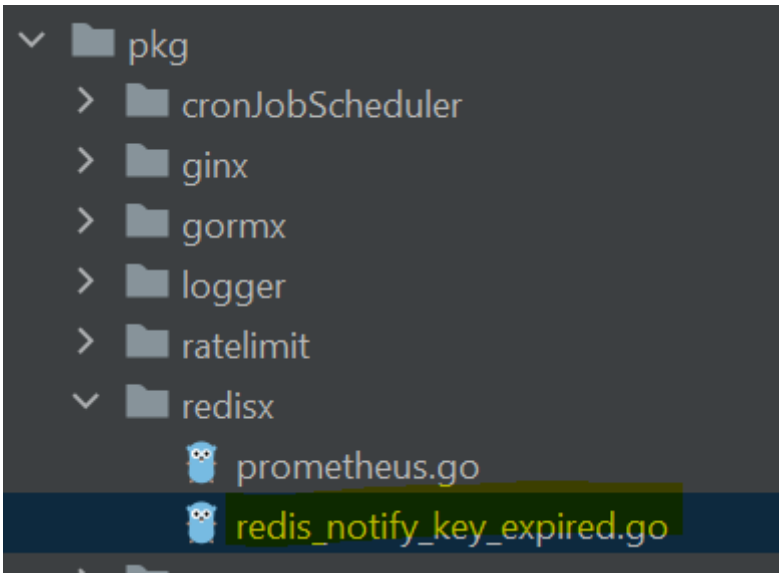
3，如果不存在则判断ZSet是否满了，如果没满且delta为加一，则创建这个member （ZINCRBY, 没有就自动创建），返回2，否则返回1

```
local key = KEYS[1]
-- +1 或者 -1
local delta = tonumber(ARGV[1])
-- 对应到的是 ZSet 中的 member
local member = tonumber(ARGV[2])
local limit = tonumber(ARGV[3])
local count = redis.call("ZCARD", key)
local score = redis.call("ZSCORE", key, member)
-- zset不存在，直接返回
if count == 0 then
    return 4
end
-- 该member不存在 ， 那么只有在ZSET个数小于limit 且 增加1 时才加入ZSET
if score == nil then
    if delta == 1 and count < limit then
        redis.call("ZINCRBY", key, delta, member)
        return 2
    end
else
    redis.call("ZINCRBY", key, delta, member)
    return 3
end

return 1
```

2.3，ZSet过期

这是一个通用功能，所以主要代码写在pkg下



```
package redisx

import (
 "context"
 "fmt"
 "github.com/bolognagene/geektime-gocamp/geektime-gocamp/webook/webook/internal/key_expired_event"
 "github.com/redis/go-redis/v9"
)

type Handler struct {
 client redis.Cmdable
 evts   []key_expired_event.KeyExpiredEvent
}

func NewHandler(client redis.Cmdable, evts []key_expired_event.KeyExpiredEvent) *Handler {
 return &Handler{
  client: client,
  evts:   evts,
 }
}

func (h *Handler) NotifyKeyExpiredEvent() {
 cli, ok := h.client.(*redis.Client)
 if ok {
  _, err := cli.Do(context.Background(), "CONFIG", "SET", "notify-keyspace-events", "Ex").Result()
  if err != nil {
   panic(err)
  }
  pubSub := cli.PSubscribe(context.Background(), "__keyevent@0__:expired")

  go func() {
   // 创建一个接收通道以接收订阅的消息
   channel := pubSub.Channel()
   // 开始监听订阅的消息
   for msg := range channel {
    fmt.Printf("Received expired key message: %s\n", msg.Payload)

    for _, evt := range h.evts {
     evt.Process(msg.Payload)
    }
   }
  }()

 } else {
  panic("notify redis expired key initialize failed!")
 }
}
```

仿照之前sarama的写法，新建一个key_expired_event文件夹，将Process函数写在这里，在这里调用GetTopLike函数

```
package key_expired_event

import (
 "context"
 "fmt"
 "github.com/bolognagene/geektime-gocamp/geektime-gocamp/webook/webook/internal/repository"
 "github.com/bolognagene/geektime-gocamp/geektime-gocamp/webook/webook/internal/web"
 "github.com/bolognagene/geektime-gocamp/geektime-gocamp/webook/webook/pkg/logger"
)

type TopLikeKey struct {
 repo repository.InteractiveRepository
 l    logger.Logger
 biz  string
}

func NewTopLikeKey(repo repository.InteractiveRepository,
 l logger.Logger, biz string) *TopLikeKey {
 return &TopLikeKey{
  repo: repo,
  l:    l,
  biz:  biz,
 }
}

func (t *TopLikeKey) Process(key string) error {
 // 是过期的key就处理
 if key == fmt.Sprintf("top_like_%s", t.biz) {
  _, err := t.repo.GetTopLike(context.Background(), t.biz,
   web.TopLikeN.Load(), web.TopLikeLimit.Load())
  if err != nil {
   t.l.Debug("top like key到期处理失败", logger.Error(err),
    logger.String("biz", t.biz))
  }
 }

 return nil
}
```

如果调用？同样仿照之前的Sarama，在APP里加上pkg包里的redisx.Handler

```
package main

import (
 "github.com/bolognagene/geektime-gocamp/geektime-gocamp/webook/webook/internal/events"
 "github.com/bolognagene/geektime-gocamp/geektime-gocamp/webook/webook/pkg/redisx"
 "github.com/gin-gonic/gin"
 "github.com/robfig/cron/v3"
)

type App struct {
 web       *gin.Engine
 consumers []events.Consumer
 rh        *redisx.Handler   <--- 这里!
 cron      *cron.Cron
}
```

在main函数里调用起来!

```
func main() {
 initViperFromLocalConfigFile()
 //initViperFromReader()
 //initViperFromProgramArgument()
 //initViperFromRemote()
 app := InitWebServer()
 for _, c := range app.consumers {
  err := c.Start()
  if err != nil {
   panic(err)
  }
 }

 app.rh.NotifyKeyExpiredEvent()   <--- 这里!!

 app.cron.Start()

 app.web.Run(":8077") // 监听并在 0.0.0.0:8077 上启动服务

 // 这里是优雅退出所有的程序
 // 一分钟内你要关完，要退出
 ctx, cancel := context.WithTimeout(context.Background(), time.Minute)
 defer cancel()

 ctx = app.cron.Stop()
 // 想办法 close ??
 // 这边可以考虑超时强制退出，防止有些任务，执行特别长的时间
 tm := time.NewTimer(time.Minute * 10)
 select {
 case <-tm.C:
 case <-ctx.Done():
 }
}
```