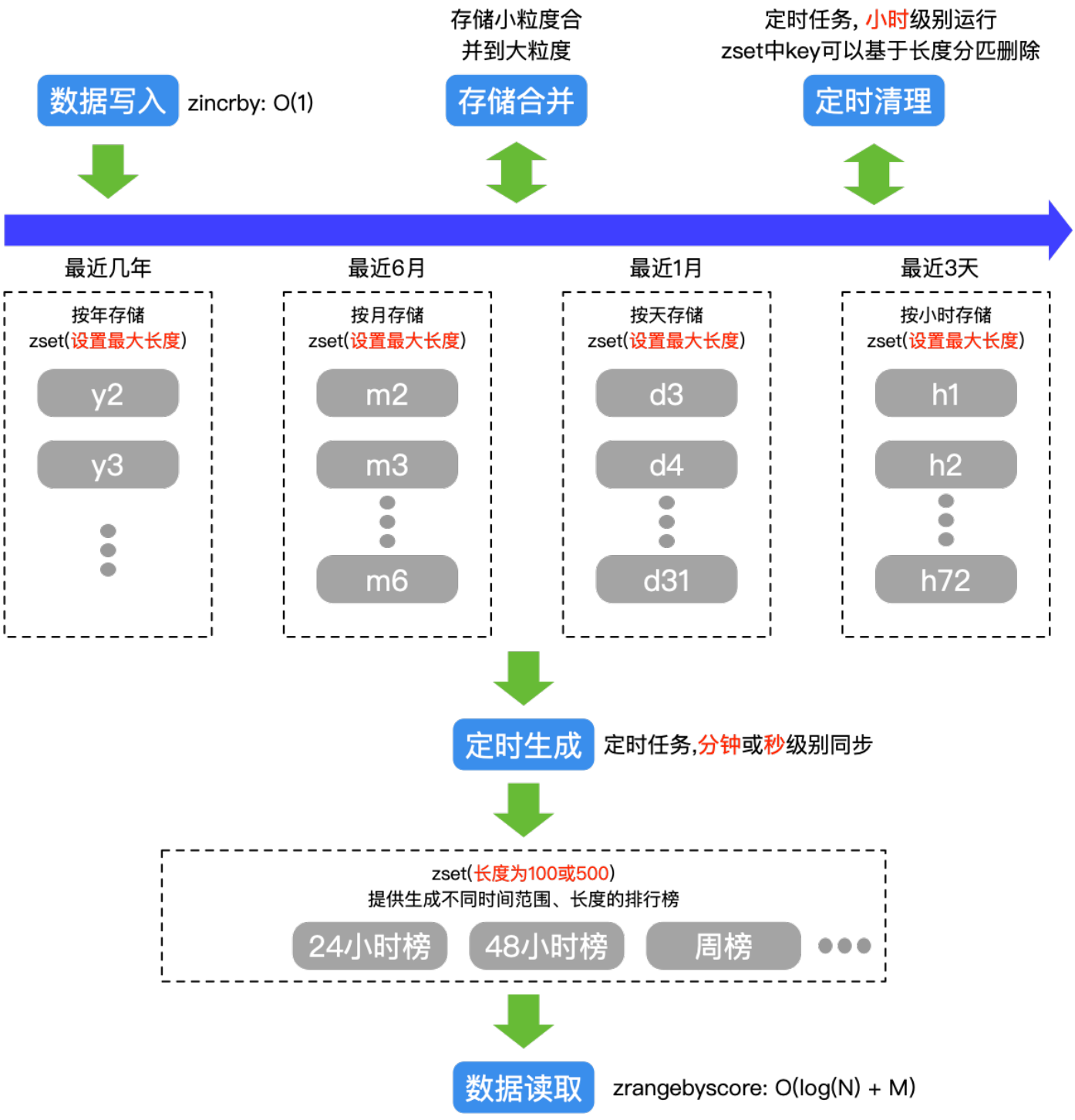
# 通用排行榜设计

## 需求

实时排行(分钟或秒级)，支持灵活**按年、按月(近6月)、按周(近4周)、按天(近3天)、按小时(近72小时)**排行。数据写入和读取分离，时间复杂度稳定。

## 基于redis存储设计



## 数据写入

直接zhincrby对应小时key:

```
// KeyIncr 记录key自增
func KeyIncr(ctx context.Context, appid string, key string, value uint32) error {
  util.ReportMonitor(fmt.Sprintf("%s调用KeyIncr请求总量", appid), 1, 0)
  curRedisKey := GetCurrentStoreKey(appid)

  keyLen, err := bredis.GetRedisPool().ZCount(curRedisKey, "-inf", "+inf")
  if keyLen > StoreControl.Level1Size {
    err = fmt.Errorf("level1 key[%s] is excess size[%d]", curRedisKey,
StoreControl.Level1Size)
    log.Errorf(err.Error())
    util.ReportMonitor("当前存储key超过最大值", 1, 0)
    return err
  }
  _, err = bredis.GetRedisPool().ZIncrBy(curRedisKey, int(value), key)
  if err != nil {
    log.Errorf(err.Error())
    util.ReportMonitor("redis.zset.key自增失败", 1, 0)
    return err
  }
  return nil
}

// GetCurrentStoreKey 获取当前写入redis的key
func GetCurrentStoreKey(appid string) string {
  zoneKey := util.GetCurrentTimeKey(StoreControl.Level1Unit)
  key := RedisKeyStoreLevel1 + appid + ":" + zoneKey
  log.Debugf("current redis key: %s", key)
  return key
}
```

## 存储合并

定时检查，合并存储，从小粒度合并到大粒度，周期可以和生成排行榜一致：

这里逻辑稍微复杂一点

```
  // 检查并合并排行榜key存储
  ranklogic.KeyCheckAndMergeWithLock(appid)
```

### 定时清理

这里主要考量存储大小保护，定期清理没用的key，周期可以大一些，比如1小时或1天

### 生成排行榜

读数据，生成排行榜。这里可以根据实时性需求配置分钟级或秒级：

```
// GenRankByHours 生成小时级别的排行榜
func GenRankByHours(appid string, num int) error {
  startTime := time.Now()
  defer func() {
    costTime := time.Now().Sub(startTime).Milliseconds()
    config.LogTaskTimer.Infof("gen rank by hours, appid: %s, num: %d, cost_time: %d",
appid, num, costTime)
  }()

  rankType := fmt.Sprintf("%dh", num)
  rankRedisKey := GetRankKey(appid, rankType)
  lockKey := RedisKeyMergeLock + rankRedisKey
  lockValue, errLock := lock.GetLock(lockKey, lock.TRANSACTION_LOCK_EXPIRE)
  defer lock.ReleaseLock(lockKey, lockValue) //释放lock
  if errLock != nil {
    util.ReportMonitor(fmt.Sprintf("%s生成%d天排行榜抢锁失败", appid, num), 1, 0)
    config.LogTaskTimer.Warnf("Get Lock Failed, err: %v", errLock)
    return nil // 加锁失败返回nil
  }

  validKeys, err := getValidKeys(appid, rankType)
  if err != nil {
    config.LogTaskTimer.Warnf(err.Error())
    return err
  }

  config.LogTaskTimer.Debugf("GenRankByHours src list: %+v, target key: %s", validKeys,
rankRedisKey)

  rankRedisKeyTemp := fmt.Sprintf("%s:temp", rankRedisKey)
  err = KeyMerge(validKeys, rankRedisKeyTemp)
  if err != nil {
    config.LogTaskTimer.Errorf(err.Error())
    return err
  }
  // 删除临时redis key
  defer func() {
    _, errTempDel := bredis.GetRedisPool().Del([]string{rankRedisKeyTemp})
    if err != nil {
      config.LogTaskTimer.Error("temp rankRedisKeyTempkey del failed, err:%v",
errTempDel)
```

```
      err = errTempDel
    }
  }()
  rankData, err := bredis.GetRedisPool().ZRevRangeWithScores(rankRedisKeyTemp, 0, -1)
  if err != nil {
    config.LogTaskTimer.Errorf(err.Error())
    return err
  }
  var keyInfos []*pb.KeyInfo
  for _, v := range rankData {
    // 基于redis中zset的score，这里转换肯定没问题
    score, _ := strconv.ParseUint(v["value"], 10, 64)
    keyInfos = append(keyInfos, &pb.KeyInfo{
      Key:   util.InterfaceToString(v["member"]),
      Score: score,
    })
  }
  keyInfosJson, err := codec.Marshal(codec.SerializationTypeJSON, keyInfos)
  if err != nil {
    config.LogTaskTimer.Errorf(err.Error())
    return err
  }
  err = bredis.GetRedisPool().Set(rankRedisKey, string(keyInfosJson))
  if err != nil {
    config.LogTaskTimer.Errorf(err.Error())
    return err
  }
  return err
}

func getValidKeys(appid, rankType string) ([]string, error) {
  storeKeys := GetStoreKeysByConf()
  timeBegin, err := util.GetBeforeTime(rankType)
  if err != nil {
    config.LogTaskTimer.Warnf("Get Lock Failed")
    return nil, err
  }
  timeBeginStr := timeBegin.Format("2006010215")
  var validKeys []string
  maxValidDayKeyNum := 0
  // 获取有效的2级存储key
  for index, dayKey := range storeKeys.DayKeys {
    dayKeyInt := util.InterfaceToInt(fmt.Sprintf("%s00", dayKey))
    if dayKeyInt >= util.InterfaceToInt(timeBeginStr) {
      dayRedisKey := RedisKeyStoreLevel2 + appid + ":" + dayKey
      validKeys = append(validKeys, dayRedisKey)
      if index == len(storeKeys.DayKeys)-1 {
        maxValidDayKeyNum = dayKeyInt
      }
```

```
    }
  }
  // 获取有效的1级存储key
  for _, hourKey := range storeKeys.HourKeys {
    hourKeyInt := util.InterfaceToInt(hourKey)
    if hourKeyInt >= util.InterfaceToInt(timeBeginStr) {
      hourRedisKey := RedisKeyStoreLevel1 + appid + ":" + hourKey
      // 2级别存储补充00转为int: 2020061700 + 100 = 2020061800
      if maxValidDayKeyNum+100 > hourKeyInt {
        continue
      }
      validKeys = append(validKeys, hourRedisKey)
    }
  }
  return validKeys, nil
}
```