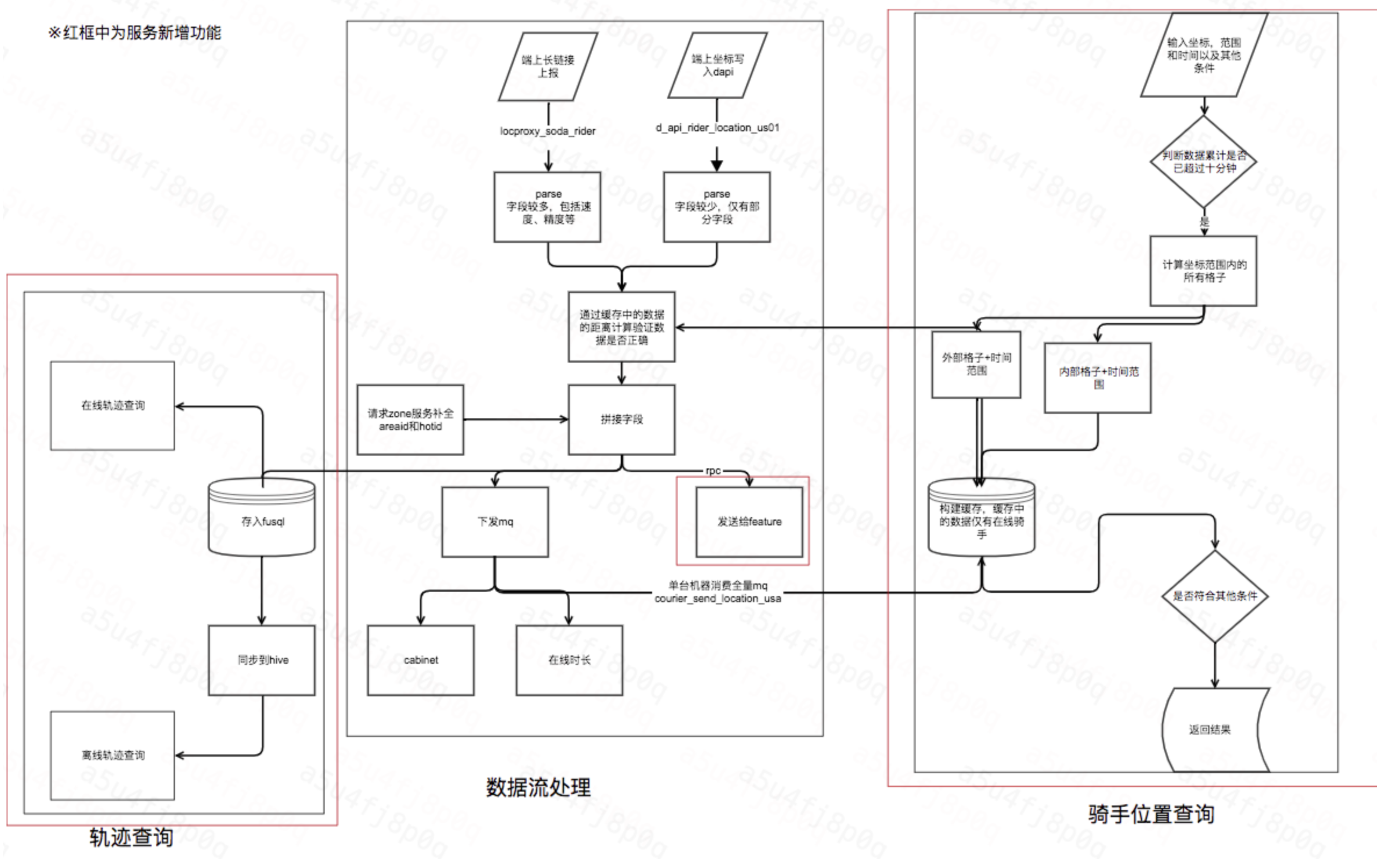
范英昊工作总结

一.UDP兜底服务

需求背景：


※红框中为服务新增功能
端上长链接上报
locproxy_soda_rider
端上坐标写入dapi
d_api_rider_location_us01
parse 字段较多，包括速度、精度等
parse 字段较少，仅有部分字段
通过缓存中的数据的距离计算验证数据是否正确
请求zone服务补全areaid和hotid
拼接字段
rpc
下发mq
发送给feature
单台机器消费全量mq courier_send_location_usa
cabinet
在线时长
数据流处理
在线轨迹查询
存入fusql
同步到hive
离线轨迹查询
轨迹查询
输入坐标、范围和时间以及其他条件
判断数据累计是否已超过十分钟
是
计算坐标范围内的所有格子
外部格子+时间范围
内部格子+时间范围
构建缓存，缓存中的数据仅有在线骑手
是否符合其他条件
返回结果
骑手位置查询

如图所示，courier-dataflow是一个数据流式处理项目，提供高性能、易扩展的数据处理服务，使用的是go-stream框架，支撑每天5000+骑手的每5s一次的数据上报，单日处理坐标3千万+条。

而courier-lbs用于骑手的位置查询，接口能力： 5000+骑手数，高峰qps250+ ，单坐标返回周边骑手信息耗时10ms以内。

但现在目前来说courier-dataflow向courier-lbs当中传输坐标时只有一个mq，所以courier-dataflow需要一个udp的兜底服务，上传的坐标隔一段时间向courier-lbs当中发送一次。

需求分解

任务完成：

1.courier-dataflow项目(存储并发送数据)

（1）获取courier-lbs机器的ip地址

采用odin API查询方式，先获得ns节点机器的列表

```go
func GetMachinesByNS(API string, serviceName string) []MachineInfo {
   //url
   var url = fmt.Sprintf("%s%s", API, serviceName)
   //url
   response, err := http.Get(url)
   if err != nil {
      log.Errorf("_GetMachinesByNS_failure||err=%+v",err)
      return nil
   }
   //response.Body
   defer response.Body.Close()
   //response
   body, err := ioutil.ReadAll(response.Body)
   if err != nil {
      log.Errorf("_GetMachinesByNS_ioutil_failure||err=%+v",err)
      return nil
   }
   //
   var a []MachineInfo
   if err = json.Unmarshal(body, &a); err != nil {
      log.Errorf("_GetMachinesByNS_unmarshal_failure||err=%+v",err)
      return nil
   }
   return a
}
```

然后根据所处环境，获得机器的ip地址

```
func GetMachinesIp(API string, serviceName string, cluster string) []
string {
   var res = GetMachinesByNS(API, serviceName)
   if res==nil{
      log.Errorf("_GetMachinesIp_failure||the res is empty!")
      return nil
   }
   var ip []string
   if cluster=="us"{
      for i := 0; i < len(res); i++ {
         name := strings.Split(res[i].Name, ".")
         last_name := strings.Split(name[0], "-")
         if last_name[len(last_name)-1] != "pre" {
            ip = append(ip, res[i].Ip)
         }
      }
   }else{
      for i := 0; i < len(res); i++ {
         name := strings.Split(res[i].Name, ".")
         last_name := strings.Split(name[0], "-")
         if last_name[len(last_name)-1] == "pre" {
            ip = append(ip, res[i].Ip)
         }
      }
   }
   return ip
}
```

（2）获得IP地址以后，需要根据ip地址建立client链接，此处使用的是github开源的thriftudp工具，链接如：https://github.com/x-mod/thriftudp

遇到问题：

1在根据ip建立transport.NewTUDPClientTransport过程中发现，如果多次建立transport的话会发生端口不够用的情况，解决方法：采用map，map的key是ip，value是transport

2每隔一段时间发送数据，而在发送前都会查询ip地址，并建立相应的map，而在这个过程中，如果上次发送数据还没结束，那么会出现读写并发问题，因此采用syn.map形式

```
func (u *UDPPool) NewUDPClient(IP string) *courierudp.CourierUDPClient {
   if _, ok := u.transports.LoadOrStore(IP, IP); !ok {
      tr, err := transport.NewTUDPClientTransport(IP, "")
      if err != nil {
         log.Errorf("_create_UDP_client_failure||err=%+v", err)
      }
      u.transports.Store(IP, tr)
   }
   if tr, ok := u.transports.Load(IP); ok {
      tr := tr.(*transport.TUDPTransport)
      client := courierudp.NewCourierUDPClientFactory(tr, thrift.
NewTCompactProtocolFactory())
      return client
   }
   return nil
}
```

3.与courier-lbs建立链接以后，需要进行数据的发送

(1)首先要建立相应的数据链路

1建立一个channel

```
MessageSendByUDP                  chan interface{}
```

2对channel建立一个数据链路，消费由dapi上传的骑手数据

```
// 30thrift
func NewDapiLocationFlow() {
   source := ext.NewChanSource(DapiLocationIn)
   flow1 := flow.NewFilter(validate, 1000)
   flow02 := flow.NewMap(getLastRiderInfo, 2000)
   flow2 := flow.NewFilter(checkLocation, 1000)
   flow3 := flow.NewMap(decorate, 1000)
   flow4 := flow.NewMap(getZoneID, 2000)
   flow05 := flow.NewMap(getRiderLoad, 1000)
   flow5 := flow.NewMap(getHexID, 1000)
   flow7 := flow.NewMap(getNoticeMsg, 1000)
   //flow8 := flow.NewFilter(reduceFrequencyFilter, 1000)
   flow9 := flow.NewFilter(reduceFrequencyByReportTimeFilter, 1000)
   sink1 := ext.NewChanSink(MessageSendQueue)
   sink2 := ext.NewChanSink(FeatureWriteQueue)
   sink3 := ext.NewChanSink(MysqlSaveQueue)
   sink4 := ext.NewChanSink(MessageSendDecreaseFrequencyQueue)
   //sink5 := ext.NewChanSink(TrackSaveQueue)
   sink6 := ext.NewChanSink(MessageSendByUDP)
   go func() {
      //flows := flow.FanOut(source.Via(flow1).Via(flow02).Via(flow2).Via
(flow3).Via(flow4).Via(flow05).Via(flow5), 4)
      flows := flow.FanOut(source.Via(flow1).Via(flow02).Via(flow2).Via
(flow3).Via(flow4).Via(flow05).Via(flow5), 5)
      go flows[0].Via(flow7).To(sink1)
      go flows[1].To(sink2)
      go flows[2].Via(flow9).To(sink3)
      // flowflowchannelflowchannel
      go flows[3].Via(
         flow.NewFilter(reduceFrequencyByReportTimeFilter, 1000)).
         Via(flow.NewMap(getNoticeMsg, 1000)).
         To(sink4)
      //go flows[4].To(sink5)
      go flows[4].To(sink6)
   }()
}
```

(2)对传入channel进行消费策略：先将数据写到一个buffer当中，到达一定时间则全部发送；buffer当中只存最新的骑手数据

遇到问题：

1如何存储到buffer当中，以及读写并发问题：

```
type RiderSyncMap struct {
   items map[int64]*courierudp.RiderInfo
   lock  sync.Mutex
}

func (r *RiderSyncMap) New() *RiderSyncMap {
   r.items = make(map[int64]*courierudp.RiderInfo, 0)
   return r
}

func (r *RiderSyncMap) Add(it *courierudp.RiderInfo) {
   r.lock.Lock()
   defer r.lock.Unlock()
   if _, ok := r.items[it.RiderId]; ok {
      if *it.ActionTime > *r.items[it.RiderId].ActionTime {
         r.items[it.RiderId] = it
      }
   } else {
      r.items[it.RiderId] = it
   }
}

func (r *RiderSyncMap) GetAll() map[int64]*courierudp.RiderInfo {
   return r.items
}

func (r *RiderSyncMap) Clear() {
   r.lock.Lock()
   defer r.lock.Unlock()
   r.items = map[int64]*courierudp.RiderInfo{}
}

func (r *RiderSyncMap) PopAll() map[int64]*courierudp.RiderInfo {
   r.lock.Lock()
   defer r.lock.Unlock()
   var bf = r.items
   r.items = map[int64]*courierudp.RiderInfo{}
   return bf
}
```

2如何设计定时器

```
go func() {
   var ridermap = client.RiderSyncMap{}
   var UDPClientPool = client.UDPPool{}
   ridermap.New()
   var ticker = time.NewTicker(time.Second * time.Duration(conf.
UDPSendTime))
   for {
      select {
      case in := <-MessageSendByUDP:
         acInfo, ok := in.(ActionInfo)
         if !ok {
            log.Errorf("_MessageSendByUDP_fail||%+v", in)
         }
         info, ok := acInfo.Action.(action.LocationReportAction)
         if !ok {
            log.Errorf("_MessageSendByUDP_fail||%+v||%+v", trace.
ContextString(acInfo.ctx), in)
         }
         ridermap.Add(info.Convert2courierRiderInfo())
      case <-(ticker).C:
         go func() {
            var ips []string
            ips = utils.GetMachinesIp(API, ServiceName, conf.Cluster)
            var bf = ridermap.PopAll()
            values := make([]*courierudp.RiderInfo, 0, len(bf))
            for _, value := range bf {
               values = append(values, value)
            }
            for i := 0; i < len(ips); i++ {
               //var cli = client.NewUDPClient(ips[i])
               var addr = fmt.Sprintf("%s:%d", ips[i], conf.UDPSendPort)
               var cli = UDPClientPool.NewUDPClient(addr)
               if cli == nil {
                  log.Errorf("_UDPClientCreate_fail||ip=%s", ips[i])
                  continue
               }
               if len(bf) != 0 {
                  if err := cli.SyncRiders(context.TODO(), values); err !=
nil {
                     log.Errorf("_courier-dataflow_SyncRiders_fa")
                  }
               }
            }
         }()
      }
   }
```

2.courier-lbs项目(消费数据)

(1) 建立server端，接受从courier-dataflow过来的数据,将接受过来的数据塞入channel

```
func (s *Service) SyncRiders(ctx context.Context, riders []*courierudp.
RiderInfo) (err error) {
   log.Debugf("_SyncRiders||riders=%d", len(riders))
   for i:=0;i<len(riders);i++{
      model.UDPIn<-*riders[i]
   }
   return nil
}

func NewServer() {
   routine.Main(context.TODO(), routine.ExecutorFunc(func(ctx context.
Context) error {
      var addr=fmt.Sprintf("0.0.0.0:%d",conf.UDPServerPort)
      srv := thriftudp.NewServer(
         thriftudp.ListenAddress(addr),
         thriftudp.Processor(
            courierudp.NewCourierUDPProcessor(&Service{}),
            2),
      )
      if err := srv.Open(ctx); err != nil {
         return err
      }
      fmt.Printf("udp serving a %s\n",addr)
      return srv.Serv(ctx)
   }))
}
```

(2)建立数据链路，写了一个decode函数，仿照原逻辑塞入mq的channel

```
source := ext.NewChanSource(UDPIn)
flow1 := flow.NewMap(decoder_udp, 1000) //
flow2 := flow.NewFilter(validate, 1000)
sink1 := ext.NewChanSink(CacheQueue) //
sink2 := ext.NewChanSink(Grid7Queue)
go func() {
   flows := flow.FanOut(source.Via(flow1).Via(flow2), 2)
   go flows[0].To(sink1)
   go flows[1].To(sink2)
}()
```

二. 骑手轨迹可视化

需求背景

根据CityID 取出一小时内的所有骑手（RiderID）的轨迹，进行可视化

需求分解：

1.需要根据城市id获取整个城市内部的骑手id

2.需要根据骑手id获取骑手一段时间内的轨迹坐标

3.将上述两个需求进行结合，达到根据城市id获取骑手一段时间内的轨迹任务

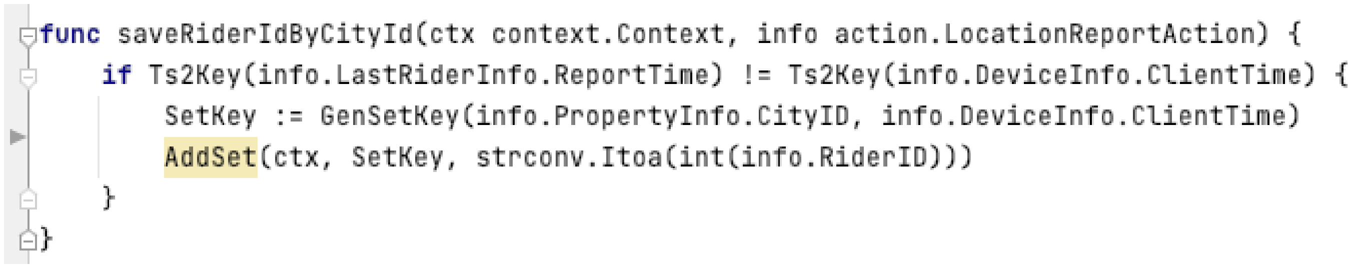
任务完成：

1.courier-dataflow项目(存数据)

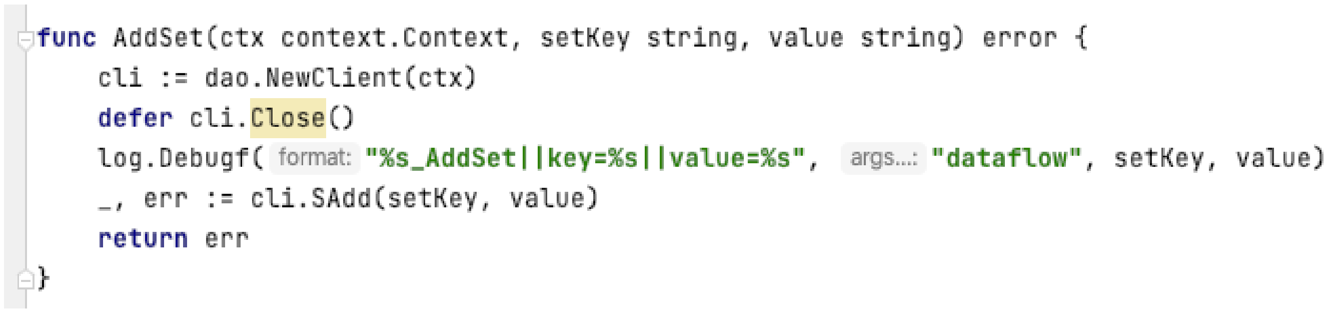
courier-dataflow是一个数据流式处理项目，对于本需求而言，courier-dataflow的上游是dapi和端上长链接，这二者可以上传数据骑手的的一些数据以供下游处理。

由于我们需要geojson的格式来进行可视化。所以我们在存数据的时候，要根据CityID存整点小时内的所有RiderID。

根据CityID和当前整点小时数设置SetKey。然后调用Redis SAdd，以RiderID为value存起来。



```go
func saveRiderIdByCityId(ctx context.Context, info action.LocationReportAction) {
    if Ts2Key(info.LastRiderInfo.ReportTime) != Ts2Key(info.DeviceInfo.ClientTime) {
        SetKey := GenSetKey(info.PropertyInfo.CityID, info.DeviceInfo.ClientTime)
        AddSet(ctx, SetKey, strconv.Itoa(int(info.RiderID)))
    }
}
```



```go
func AddSet(ctx context.Context, setKey string, value string) error {
    cli := dao.NewClient(ctx)
    defer cli.Close()
    log.Debugf( format: "%s_AddSet||key=%s||value=%s",  args...: "dataflow", setKey, value)
    _, err := cli.SAdd(setKey, value)
    return err
}
```

其保存形式为：

courier_rider_set_52140500_2021_07_08_15

具体测试可以根据courier-track的根据城市id获取骑手id的http接口测试

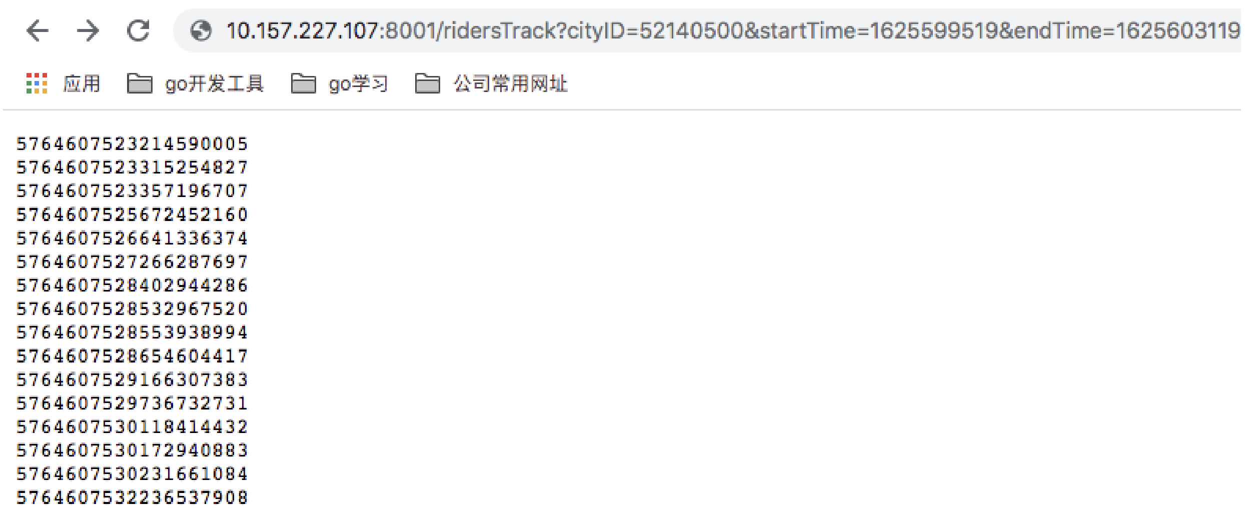
2.courier-track项目（取数据）

courier-track是courier-dataflow下游的消费项目，其主要的作用是对骑手轨迹进行一些处理。

本次修改主要包括三个http接口：

第一个：根据城市id获取骑手的id

测试url:http://10.157.227.107:8001/riderIDset?cityID=52140500&startTime=1625599519&endTime=1625603119

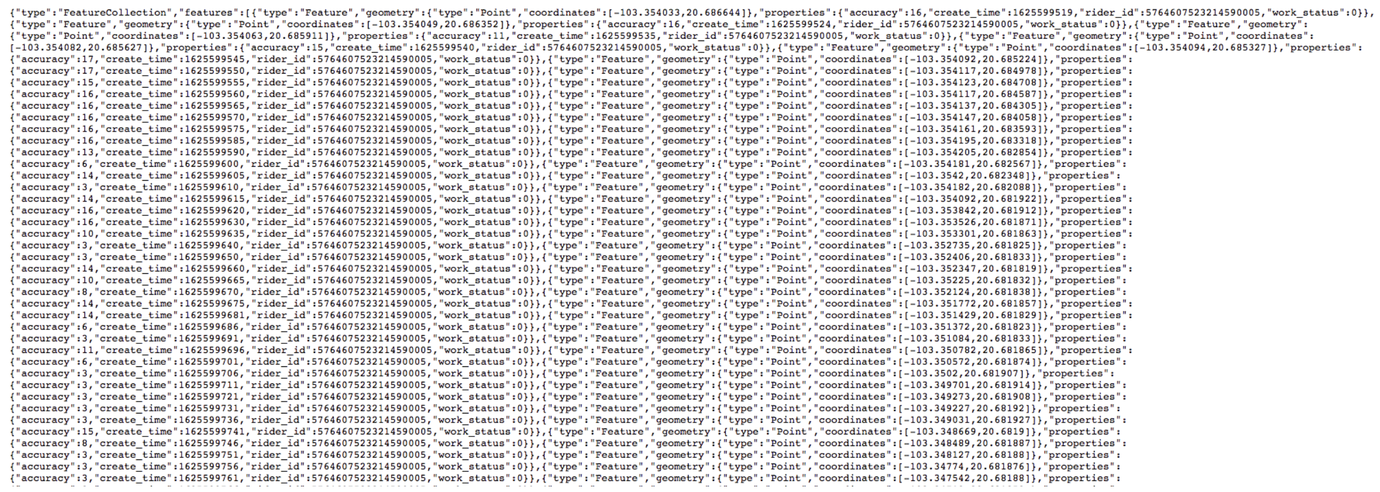


```
10.157.227.107:8001/ridersTrack?cityID=52140500&startTime=1625599519&endTime=1625603119
```

```
5764607523214590005
5764607523315254827
5764607523357196707
5764607525672452160
5764607526641336374
5764607527266287697
5764607528402944286
5764607528532967520
5764607528553938994
5764607528654604417
5764607529166307383
5764607529736732731
5764607530118414432
5764607530172940883
5764607530231661084
5764607532236537908
```

第二个：根据骑手id获取骑手轨迹

测试url:http://10.157.227.107:8001/riderTrack?riderID=5764607523214590005&startTime=1625599519&endTime=1625603119

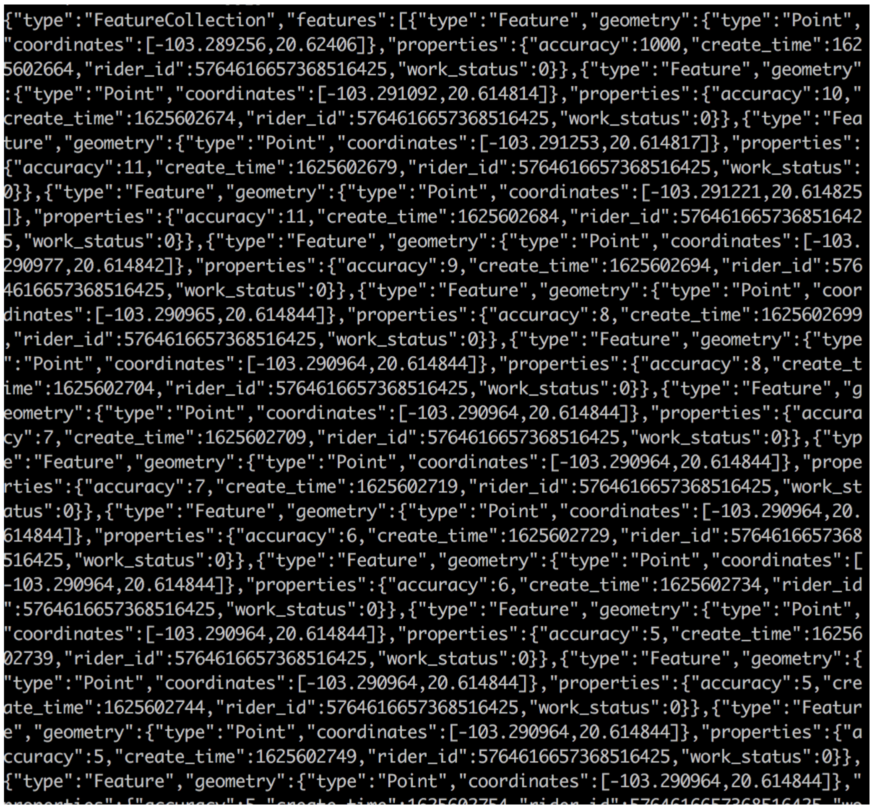


```
{"type":"FeatureCollection","features":[{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.354033,20.686644]},"properties":{"accuracy":16,"create_time":1625599519,"rider_id":5764607523214590005,"work_status":0}},
{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.354049,20.686352]},"properties":{"accuracy":16,"create_time":1625599524,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":
{"type":"Point","coordinates":[-103.354063,20.685911]},"properties":{"accuracy":11,"create_time":1625599535,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":
[-103.354082,20.685627]},"properties":{"accuracy":15,"create_time":1625599540,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.354094,20.685327]},"properties":
{"accuracy":17,"create_time":1625599545,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.354092,20.685224]},"properties":
{"accuracy":17,"create_time":1625599550,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.354117,20.684978]},"properties":
{"accuracy":15,"create_time":1625599555,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.354123,20.684708]},"properties":
{"accuracy":16,"create_time":1625599560,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.354117,20.684587]},"properties":
{"accuracy":16,"create_time":1625599565,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.354137,20.684305]},"properties":
{"accuracy":16,"create_time":1625599570,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.354147,20.684058]},"properties":
{"accuracy":16,"create_time":1625599575,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.354161,20.683593]},"properties":
{"accuracy":16,"create_time":1625599585,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.354195,20.683318]},"properties":
{"accuracy":13,"create_time":1625599590,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.354205,20.682854]},"properties":
{"accuracy":6,"create_time":1625599600,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.354181,20.682567]},"properties":
{"accuracy":14,"create_time":1625599605,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.3542,20.682348]},"properties":
{"accuracy":3,"create_time":1625599610,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.354182,20.682088]},"properties":
{"accuracy":14,"create_time":1625599615,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.354092,20.681922]},"properties":
{"accuracy":16,"create_time":1625599620,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.353842,20.681912]},"properties":
{"accuracy":16,"create_time":1625599630,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.353526,20.681871]},"properties":
{"accuracy":10,"create_time":1625599635,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.353301,20.681863]},"properties":
{"accuracy":3,"create_time":1625599640,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.352735,20.681825]},"properties":
{"accuracy":3,"create_time":1625599650,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.352406,20.681833]},"properties":
{"accuracy":14,"create_time":1625599660,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.352347,20.681819]},"properties":
{"accuracy":10,"create_time":1625599665,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.35225,20.681832]},"properties":
{"accuracy":8,"create_time":1625599670,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.352124,20.681838]},"properties":
{"accuracy":14,"create_time":1625599675,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.351772,20.681857]},"properties":
{"accuracy":14,"create_time":1625599681,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.351429,20.681829]},"properties":
{"accuracy":6,"create_time":1625599686,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.351372,20.681823]},"properties":
{"accuracy":3,"create_time":1625599691,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.351084,20.681833]},"properties":
{"accuracy":11,"create_time":1625599696,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.350782,20.681865]},"properties":
{"accuracy":6,"create_time":1625599701,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.350572,20.681874]},"properties":
{"accuracy":3,"create_time":1625599706,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.3502,20.681907]},"properties":
{"accuracy":3,"create_time":1625599711,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.349701,20.681914]},"properties":
{"accuracy":3,"create_time":1625599721,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.349273,20.681908]},"properties":
{"accuracy":3,"create_time":1625599731,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.349227,20.68192]},"properties":
{"accuracy":3,"create_time":1625599736,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.349031,20.681927]},"properties":
{"accuracy":15,"create_time":1625599741,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.348669,20.6819]},"properties":
{"accuracy":8,"create_time":1625599746,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.348489,20.681887]},"properties":
{"accuracy":3,"create_time":1625599751,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.348127,20.68188]},"properties":
{"accuracy":3,"create_time":1625599756,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.34774,20.681876]},"properties":
{"accuracy":3,"create_time":1625599761,"rider_id":5764607523214590005,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.347542,20.68188]},"properties":
```

第三个：根据城市id获取骑手id

测试url：http://10.157.227.107:8001/ridersTrack?cityID=52140500&startTime=1625599519&endTime=1625603119

建议用curl访问，数据太多，http显示不过来。



```
{"type":"FeatureCollection","features":[{"type":"Feature","geometry":{"type":"Point",
"coordinates":[-103.289256,20.62406]},"properties":{"accuracy":1000,"create_time":162
5602664,"rider_id":5764616657368516425,"work_status":0}},{"type":"Feature","geometry"
:{"type":"Point","coordinates":[-103.291092,20.614814]},"properties":{"accuracy":10,"
create_time":1625602674,"rider_id":5764616657368516425,"work_status":0}},{"type":"Fea
ture","geometry":{"type":"Point","coordinates":[-103.291253,20.614817]},"properties":
{"accuracy":11,"create_time":1625602679,"rider_id":5764616657368516425,"work_status":
0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.291221,20.614825
]},"properties":{"accuracy":11,"create_time":1625602684,"rider_id":576461665736851642
5,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.
290977,20.614842]},"properties":{"accuracy":9,"create_time":1625602694,"rider_id":576
4616657368516425,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coor
dinates":[-103.290965,20.614844]},"properties":{"accuracy":8,"create_time":1625602699
,"rider_id":5764616657368516425,"work_status":0}},{"type":"Feature","geometry":{"type
":"Point","coordinates":[-103.290964,20.614844]},"properties":{"accuracy":8,"create_t
ime":1625602704,"rider_id":5764616657368516425,"work_status":0}},{"type":"Feature","g
eometry":{"type":"Point","coordinates":[-103.290964,20.614844]},"properties":{"accura
cy":7,"create_time":1625602709,"rider_id":5764616657368516425,"work_status":0}},{"typ
e":"Feature","geometry":{"type":"Point","coordinates":[-103.290964,20.614844]},"prope
rties":{"accuracy":7,"create_time":1625602719,"rider_id":5764616657368516425,"work_st
atus":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.290964,20.
614844]},"properties":{"accuracy":6,"create_time":1625602729,"rider_id":5764616657368
516425,"work_status":0}},{"type":"Feature","geometry":{"type":"Point","coordinates":[
-103.290964,20.614844]},"properties":{"accuracy":6,"create_time":1625602734,"rider_id
":5764616657368516425,"work_status":0}},{"type":"Feature","geometry":{"type":"Point",
"coordinates":[-103.290964,20.614844]},"properties":{"accuracy":5,"create_time":16256
02739,"rider_id":5764616657368516425,"work_status":0}},{"type":"Feature","geometry":{
"type":"Point","coordinates":[-103.290964,20.614844]},"properties":{"accuracy":5,"cre
ate_time":1625602744,"rider_id":5764616657368516425,"work_status":0}},{"type":"Featur
e","geometry":{"type":"Point","coordinates":[-103.290964,20.614844]},"properties":{"a
ccuracy":5,"create_time":1625602749,"rider_id":5764616657368516425,"work_status":0}},
{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.290964,20.614844]},"
```

3.pyinsight项目(可视化)

可视化的工具 Uber Kepler

Uber 开源了其内部可视化工具包 ketoper.gl，这是一个基于 deck.gl 构建的 React 组件，高性能，用于大规模地理定位数据集的可视化探索。它对 GPU 功能的支持允许应用程序立即渲染数百万个数据点。

根据经纬度以及时间戳 做成 轨迹运动的视频

输入数据设置成geoJson格式。因为是多个骑手放在一个大的geoJson文件，可以添加多个feature（一条轨迹一个feature）

```
geojson = {"type": "FeatureCollection", "features": []}


ft = {"type": "Feature",
      "properties": {
          "vendor": "A"
 },
      "geometry": {
          "type": "LineString",
          "coordinates": [lng, lat, 0, time], ....
      }
}
```

学习url:https://www.cnblogs.com/feffery/p/12987968.html

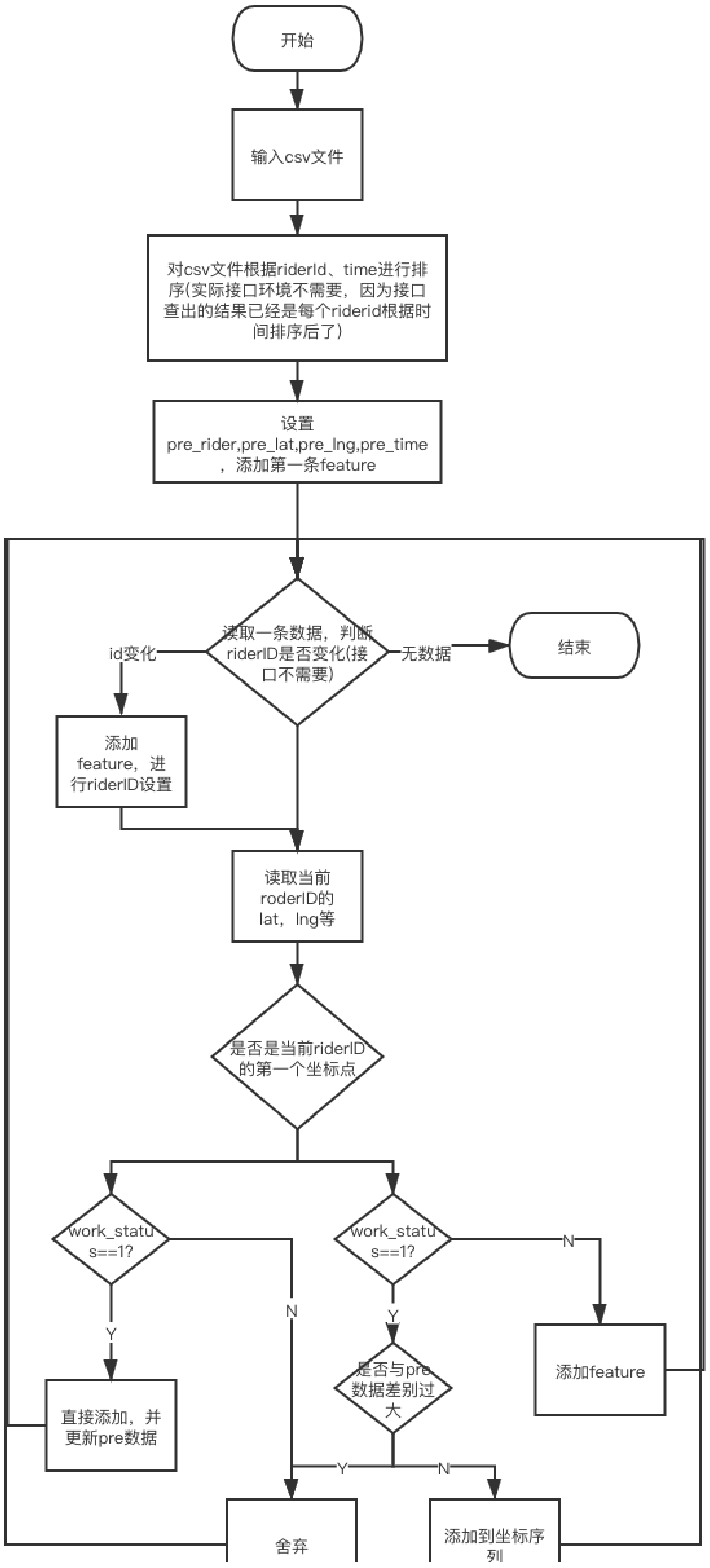
pyinsight项目主要是用来对查询出来的数据进行处理，根据城市id查询出的骑手轨迹数据特别类似从hive表查询出来的骑手轨迹.csv，但csv首先需要根据骑手id和创建时间进行一个排序，所以在本地开发可以通过csv数据进行开发及测试。

对数据进行处理如下：

一条骑手的轨迹坐标如下：

{"type":"Feature","geometry":{"type":"Point","coordinates":[-103.289256,20.62406]},"properties":{"accuracy":1000,"create_time":1625602664,"rider_id":5764616657368516425,"work_status":0}}

处理流程如下：


开始
输入csv文件
对csv文件根据riderId、time进行排序(实际接口环境不需要，因为接口查出的结果已经是每个riderid根据时间排序后了)
设置pre_rider,pre_lat,pre_lng,pre_time，添加第一条feature
读取一条数据，判断riderID是否变化(接口不需要)
id变化
无数据
结束
添加feature，进行riderID设置
读取当前roderID的lat，lng等
是否是当前riderID的第一个坐标点
work_status==1?
Y
N
直接添加，并更新pre数据
work_status==1?
Y
N
是否与pre数据差别过大
Y
N
添加feature
舍弃
添加到坐标序列

4.遇到问题及解决方法：

1.feature的添加

ft就是geojson当中的features部分，不可以将ft设置为全局变量

```
geojson = {"type": "FeatureCollection", "features": []}

ft = {"type": "Feature",
      "properties": {
          "vendor": "A"
      },
      "geometry": {
          "type": "LineString",
          "coordinates": []
      }
      }

@city_track.route('/riderIDset/cityID=<city_id>&start=<start>', methods=['GET'])
def draw_map(city_id, start, end):
    ...
    geojson['features'].append(ft)
    ...
    lng = temp["features"][i]["geometry"]["coordinates"][0]
    lat = temp["features"][i]["geometry"]["coordinates"][1]
    time = temp["features"][i]["properties"]["create_time"]
    coordinate = [lng, lat, 0, time]
    ...
    geojson['features'][j]['geometry']['coordinates'].append(coordinate)
```

因为如果ft设置成全局变量，则当中的coordinates会一直保存，即每一个riderid都会继承上一个riderid的全部轨迹

应该为：

```
geojson['features'].append({"type": "Feature",
                            "properties": {
                                "RiderID": []
                            },
                            "geometry": {
                                "type": "LineString",
                                "coordinates": []
                            }
                            })
```

2.″流星”问题

″流星“问题：即两点之间间隔过大，可能会出现某一个坐标点慢悠悠飘向另一个较远的坐标点的情况。

出现原因：

1.骑手的第一个坐标点直接进行插入，但可能这时候该坐标点恰好是″偏离点″，即偏离正常轨迹的坐标点，这时可能会出现流星问题(小概率)。

2.某两个坐标点的中间若干坐标点缺失，这时，这两个坐标点会出现流星问题(大概率)。

3.某一个骑手可能工作一段时间后，下线，但下线过程中坐标还是在一直上报，虽然我们已经根据工作状态进行坐标过滤，但如果他再次进入工作，那么，两次工作之间可能会出现流星问题。即：

(广州)11111111000000000(北京)1111111

3.解决措施：

该数据是在81270100_0-12.csv文件中得出的，其中包括529684条骑手数据，其中work_status=0的个数为95379，workstatus=2的个数为9568，共利用了234053数据，利用distance和位置少于10个的点过滤了20万的数据

线上数据测试，测试参数：http://127.0.0.1:5002/ridersTrack/city_id=52140500&start=1625599519&end=1625603119，其中包括46万骑手数据，work_status=0的个数为6万条，distance过滤掉2万条，有效数据39万条

1.根据距离进行过滤(解决原因2)

```
def distance(lon1, lat1, lon2, lat2):  # 1122
    """
    Calculate the great circle distance between two points
    on the earth (specified in decimal degrees)
    """
    #
    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])

    # haversine
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat / 2) ** 2 + cos(lat1) * cos(lat2) * sin(dlon / 2) ** 2
    c = 2 * asin(sqrt(a))
    r = 6371  #
    return c * r * 1000
```

两点之间距离个数：

大于800:243个

大于400:441个

大于200:960个

大于100:4106个

但根据线上情况，发现距离差别可能会更大一点。

2.根据时间进行过滤

大于50s的坐标点为20万，所以暂时舍弃根据时间判断

3.如果骑手workstatus=0，则另起一条轨迹(解决原因3)

5.效果展示

1.本地csv数据展示效果：

花博記念公園 鶴見緑地
Higashiosaka
Osaka
OSAKA CASTLE PARK
梅南二丁目
04/03/2
02:49:11
04/02/21 15:59:59pm

2. 线上数据展示效果：

ÁREA NATURAL PROTEGIDA BOSQUE EL NIXTICUIL – SAN ESTÉBAN – EL DIENTE
BASE AÉREA MILITAR NO. 5 ZAPOPAN
Zapopan
Guadalajara
07/06/21
19:52:34pm

6. 后续改进措施

1. 重新定义坐标过滤算法

参考链接：https://blog.csdn.net/hello_json/article/details/79984081

2.对courier-track当中返回骑手坐标进行过滤，不再包含work_status=0的坐标

测试链接：http://10.157.227.107:8001/ridersTrack?cityID=52140500&startTime=1625809208&endTime=1625809458

结论：总共work_status数量为2605条，其中work_status=0的数量为1982条，占比76%

测试链接：http://10.157.227.107:8001/ridersTrack?cityID=52140500&startTime=1625599519&endTime=1625600119

结论：总共work_status数量为68788条，其中work_status=0的数量为7125条，占比10.35%

三.内部工具建设

3.1 骑手信息

需求背景

1.对骑手召回问题可能会发生的问题进行定位

2.向cache中直接插入骑手数据，方便测试使用

需求分解

主要包含三个http接口：1.向cache中插入骑手数据，方便测试 2.骑手未召回问题定位 3.查询骑手信息

骑手信息查询

feature:

courier:

关联服务：courier-lbs

接口名：http://127.0.0.1:8001/get_rider_info

测试地址：http://127.0.0.1:8001/get_rider_info?rider_id=1152921549011943531&source=db

返回值：

```
{
        "code": 200,
        "msg": "!!",
        "is_pre": true,
        "data_source": "DB",
        "virtual_coordinates": {
                "apollo_lat": 30.70464775,
                "apollo_lng": 104.0398417
        },
        "riderInfo": {
                "rider_id": 1152921549011943531,
                "cabinet_id": 0,
                "vehicle_type": 101,
                "distance": 0,
                "action_time": 1532918461,
                "channel": 0,
                "lat": 30.6494564,
                "lng": 103.9947651,
                "gid": 0,
                "city_id": 17,
                "rider_load": 0,
                "auto_status": 0,
                "work_type": 0,
                "hot_area_id": 0,
                "pos_status": 0,
                "cash_status": 0,
                "order_type_switch": 0,
                "bind_area_type": 1
        }
}
```

作用：可以根据is_pre和virtual_coordinates判断骑手在apollo环境中是否存在坐标，而riderInfo是从courier服务中查出的结果。

rider:

接口提供人：冯艳慧

接口名：http://10.14.128.18:8000/sailing/d-api/usa/iapi/rider/detail

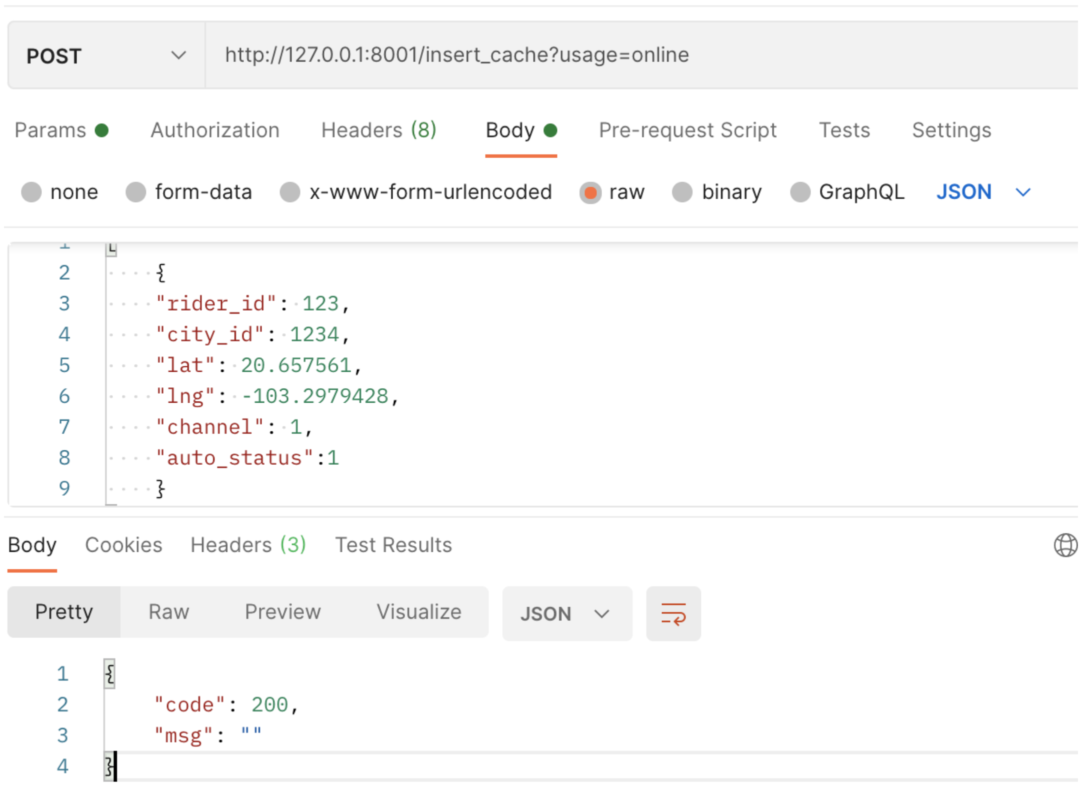
测试命令：curl -d "id=5764607751271483489" http://10.14.128.18:8000/sailing/d-api/usa/iapi/rider/detail （到pyinsight的服务器上去看）

```
2000
```

任务完成

1. 向cache中插入骑手数据，方便测试使用

使用及效果如图：

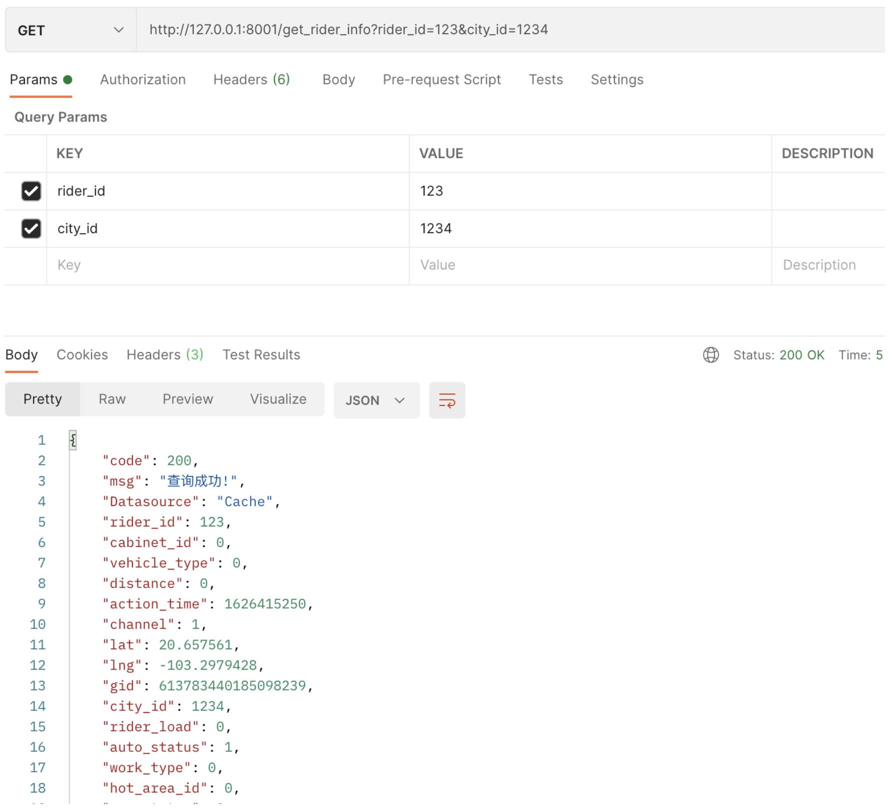


POST http://127.0.0.1:8001/insert_cache?usage=online

```
    {
    "rider_id": 123,
    "city_id": 1234,
    "lat": 20.657561,
    "lng": -103.2979428,
    "channel": 1,
    "auto_status":1
    }
```

```
{
    "code": 200,
    "msg": ""
}
```

2. 查询单个骑手的实时信息



GET http://127.0.0.1:8001/get_rider_info?rider_id=123&city_id=1234

	KEY	VALUE	DESCRIPTION
✓	rider_id	123	
✓	city_id	1234	
	Key	Value	Description

Status: 200 OK Time: 5

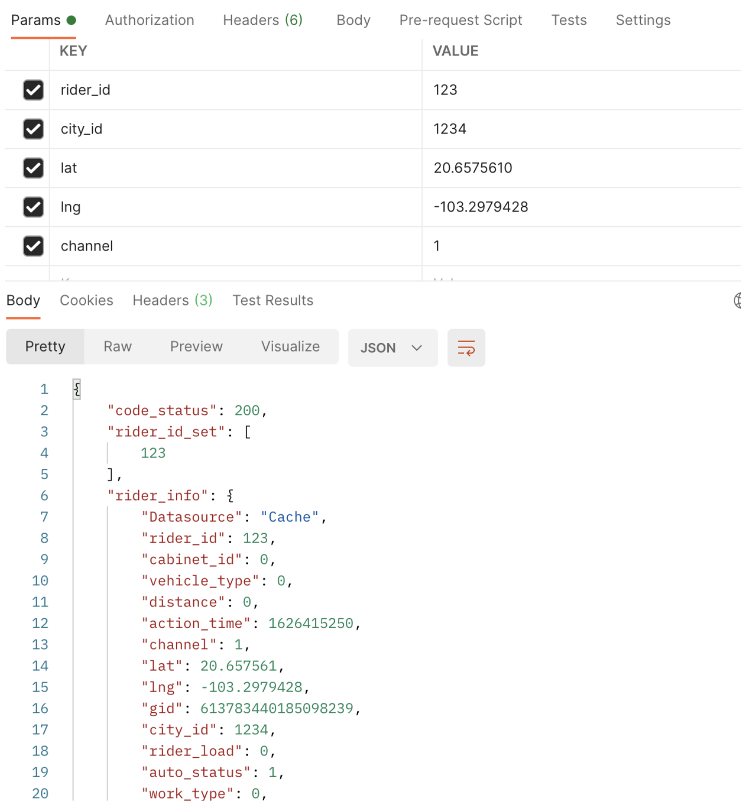
```
{
    "code": 200,
    "msg": "查询成功!",
    "Datasource": "Cache",
    "rider_id": 123,
    "cabinet_id": 0,
    "vehicle_type": 0,
    "distance": 0,
    "action_time": 1626415250,
    "channel": 1,
    "lat": 20.657561,
    "lng": -103.2979428,
    "gid": 613783440185098239,
    "city_id": 1234,
    "rider_load": 0,
    "auto_status": 1,
    "work_type": 0,
    "hot_area_id": 0,
```

3. 骑手召回问题分析

针对骑手没有被召回的情况进行分析，可能存在以下情况

```
if analysisInfo.CodeStatus == 400 {
   dist := geo.PointDistance(geo.Point{X: float64(lat), Y: float64(lng)},
geo.Point{X: analysisInfo.RiderInfo.Lat, Y: analysisInfo.RiderInfo.Lng})
   if analysisInfo.RiderInfo.Datasource == "" {
      analysisInfo.Reason = "!"
   } else if analysisInfo.RiderInfo.CityID != int64(cityID) {
      analysisInfo.Reason = "IDID!"
   } else if analysisInfo.RiderInfo.Channel != int64(channel) {
      analysisInfo.Reason = "channelchannel!"
   } else if analysisInfo.RiderInfo.AutoStatus == 0 {
      analysisInfo.Reason = "AutoStatus0!"
   } else if analysisInfo.EnvType == "pre" {
      analysisInfo.Reason = "pre!"
   } else if analysisInfo.ApolloConf != "" {
      analysisInfo.Reason = "Apollod_test_dispatch!"
   } else if dist > request.Radius {
      analysisInfo.Reason = "!!"
   } else if time.Now().Unix()-analysisInfo.RiderInfo.ActionTime > 300 {
      analysisInfo.Reason = "5!"
   } else {
      analysisInfo.Reason = "ID!(read.goSearchRiderByCoordsisUseCachetrue)"
   }
}
```

使用及效果如图：



KEY	VALUE
rider_id	123
city_id	1234
lat	20.6575610
lng	-103.2979428
channel	1

```
{
    "code_status": 200,
    "rider_id_set": [
        123
    ],
    "rider_info": {
        "Datasource": "Cache",
        "rider_id": 123,
        "cabinet_id": 0,
        "vehicle_type": 0,
        "distance": 0,
        "action_time": 1626415250,
        "channel": 1,
        "lat": 20.657561,
        "lng": -103.2979428,
        "gid": 613783440185098239,
        "city_id": 1234,
        "rider_load": 0,
        "auto_status": 1,
        "work_type": 0,
```

其中，参数rider_id是想要召回的骑手，city_id和lat，lng是召回的区域，这些都是必须的参数。

而在结果当中，code_status是有两种形式，一种是200，表示成功召回，第二种是400，表示没有成功召回。

而rider_id_set表示召回的骑手id

rider_info表示想要召回骑手的详细信息。

接下来进行一些原因的测试：

1. 如果在cache和数据库中都没有这个骑手，则会

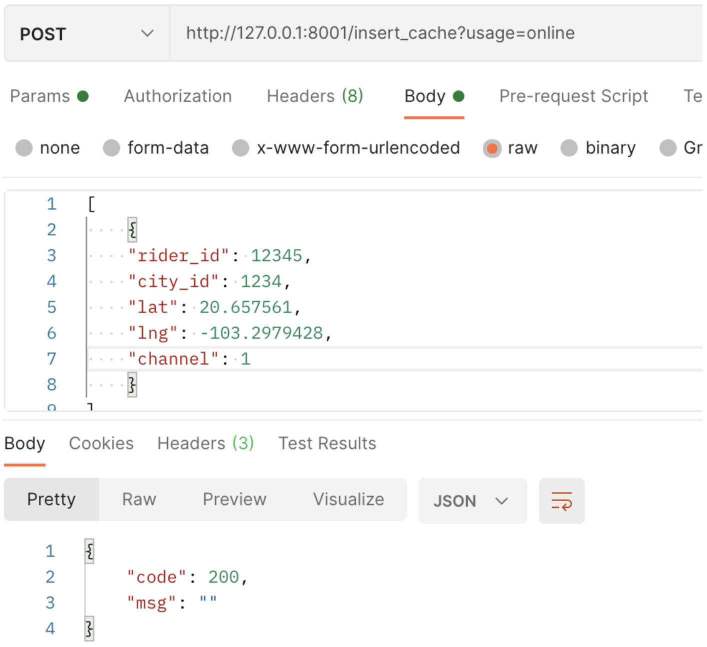
```
    "code_status": 400,
    "rider_id_set": null,
    "rider_info": {
        "Datasource": "无该数据",
        "rider_id": 12345,
        "cabinet_id": 0,
        "vehicle_type": 0,
        "distance": 0,
        "action_time": 0,
        "channel": 0,
        "lat": 0,
        "lng": 0,
        "gid": 0,
        "city_id": 0,
        "rider_load": 0,
        "auto_status": 0,
        "work_type": 0,
        "hot_area_id": 0,
        "pos_status": 0,
        "cash_status": 0,
        "order_type_switch": 0,
        "bind_area_type": 0
    },
    "environment_type": "骑手在线上环境",
    "work_status": "",
    "last_upload_time": "",
    "apollo_lat": 0,
    "apollo_lng": 0,
    "apollo_Conf": "",
    "reason": "数据库以及缓存中不存在该骑手!"
```

2. 如果表明有这个骑手，但这个骑手可能不在这个city或者channel不相符，则会

```
"code_status": 400,
"rider_id_set": null,
"rider_info": {
    "Datasource": "DB",
    "rider_id": 1234,
    "cabinet_id": 0,
    "vehicle_type": 101,
    "distance": 0,
    "action_time": 1619323213,
    "channel": 0,
    "lat": 34.6986025,
    "lng": 135.5053972,
    "gid": 0,
    "city_id": 0,
    "rider_load": 0,
    "auto_status": 1,
    "work_type": 0,
    "hot_area_id": 0,
    "pos_status": 0,
    "cash_status": 0,
    "order_type_switch": 0,
    "bind_area_type": 1
},
"environment_type": "骑手在线上环境",
"work_status": "在线",
"last_upload_time": "2021-04-25 12:00:13",
"apollo_lat": 0,
"apollo_lng": 0,
"apollo_Conf": "",
"reason": "输入城市ID与查询出骑手所在城市ID不符!"
```

channel类似

3. 在测试插入骑手时auto_status需要为1，否则无法召回

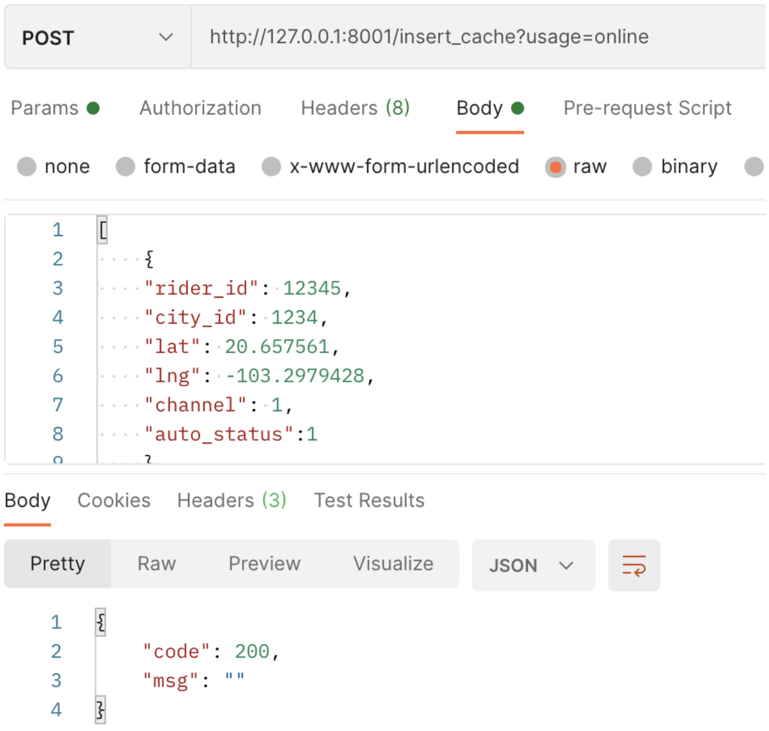


```
http://127.0.0.1:8001/insert_cache?usage=online
```

```
[
    {
    "rider_id": 12345,
    "city_id": 1234,
    "lat": 20.657561,
    "lng": -103.2979428,
    "channel": 1
    }
]
```

```
{
    "code": 200,
    "msg": ""
}
```

```
"code_status": 400,
"rider_id_set": null,
"rider_info": {
    "Datasource": "Cache",
    "rider_id": 12345,
    "cabinet_id": 0,
    "vehicle_type": 0,
    "distance": 0,
    "action_time": 1626415977,
    "channel": 1,
    "lat": 20.657561,
    "lng": -103.2979428,
    "gid": 613783440185098239,
    "city_id": 1234,
    "rider_load": 0,
    "auto_status": 0,
    "work_type": 0,
    "hot_area_id": 0,
    "pos_status": 0,
    "cash_status": 0,
    "order_type_switch": 0,
    "bind_area_type": 0
},
"environment_type": "骑手在线上环境",
"work_status": "在线",
"last_upload_time": "2021-07-16 14:12:57",
"apollo_lat": 0,
"apollo_lng": 0,
"apollo_Conf": "",
"reason": "骑手AutoStatus为0, 不在线, 无法召回!"
```



POST http://127.0.0.1:8001/insert_cache?usage=online

Params Authorization Headers (8) **Body** Pre-request Script

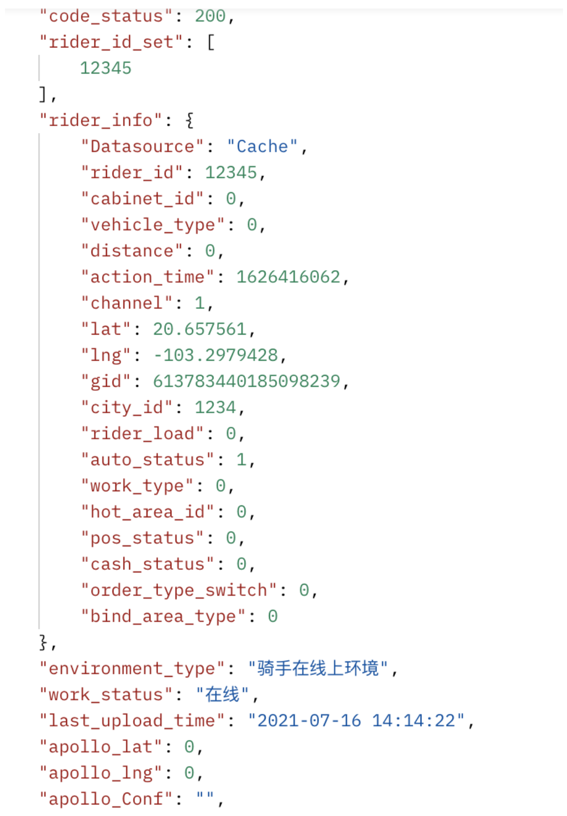
none form-data x-www-form-urlencoded [✓ raw] binary

```
[
    {
    "rider_id": 12345,
    "city_id": 1234,
    "lat": 20.657561,
    "lng": -103.2979428,
    "channel": 1,
    "auto_status":1
```

Body Cookies Headers (3) Test Results

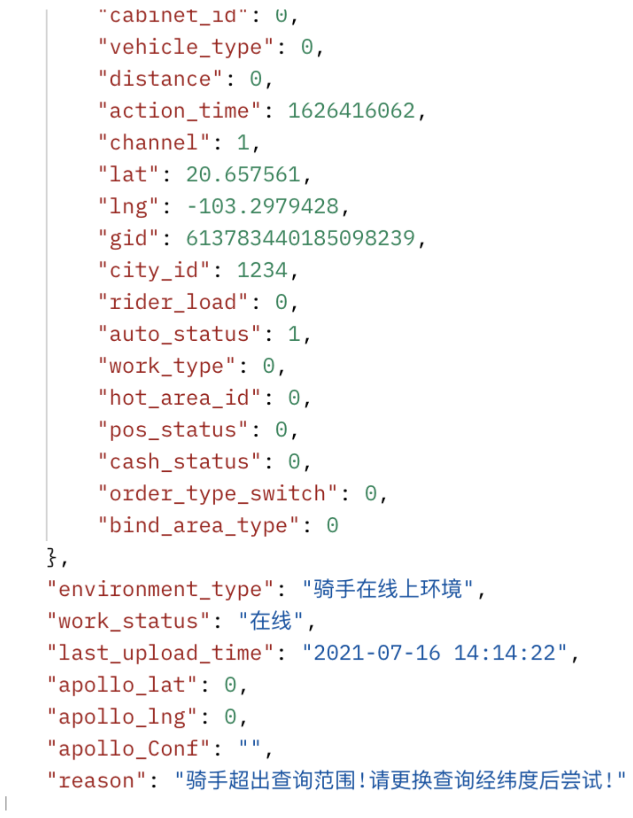
[✓ Pretty] Raw Preview Visualize JSON

```
{
    "code": 200,
    "msg": ""
}
```



```
    "code_status": 200,
    "rider_id_set": [
        12345
    ],
    "rider_info": {
        "Datasource": "Cache",
        "rider_id": 12345,
        "cabinet_id": 0,
        "vehicle_type": 0,
        "distance": 0,
        "action_time": 1626416062,
        "channel": 1,
        "lat": 20.657561,
        "lng": -103.2979428,
        "gid": 613783440185098239,
        "city_id": 1234,
        "rider_load": 0,
        "auto_status": 1,
        "work_type": 0,
        "hot_area_id": 0,
        "pos_status": 0,
        "cash_status": 0,
        "order_type_switch": 0,
        "bind_area_type": 0
    },
    "environment_type": "骑手在线上环境",
    "work_status": "在线",
    "last_upload_time": "2021-07-16 14:14:22",
    "apollo_lat": 0,
    "apollo_lng": 0,
    "apollo_Conf": "",
```

4.如果超过查询范围，即离查询所在地太远，则无法召回，而这可能是由于apollo当中已经设定了相应的测试坐标所导致的。所以，需要查看apollo_Conf中的结果与查询出的rider_info是否对应，如果对应，则证明是apollo中测试数据取代了自己插入的数据。

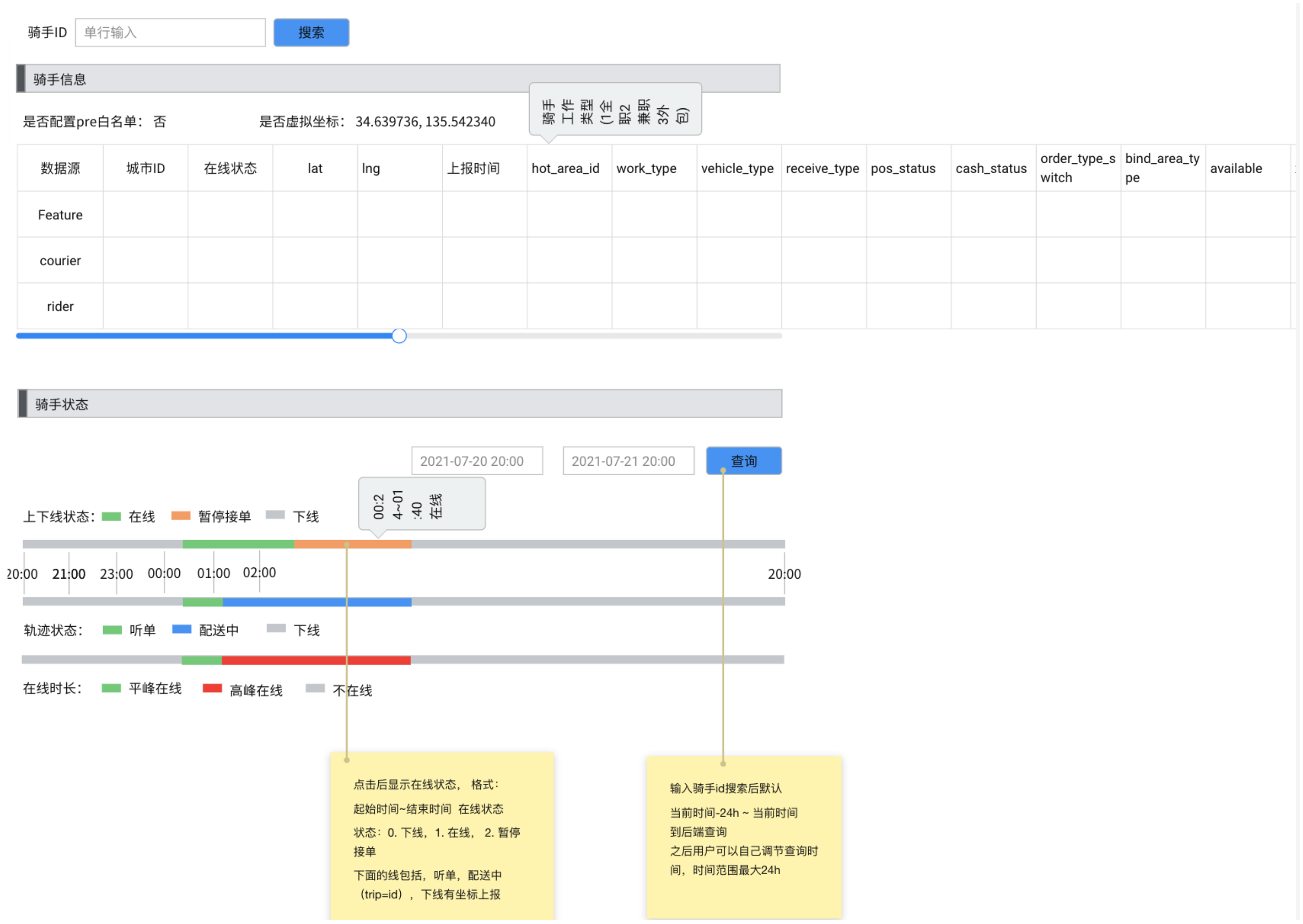


```
        "cabinet_id": 0,
        "vehicle_type": 0,
        "distance": 0,
        "action_time": 1626416062,
        "channel": 1,
        "lat": 20.657561,
        "lng": -103.2979428,
        "gid": 613783440185098239,
        "city_id": 1234,
        "rider_load": 0,
        "auto_status": 1,
        "work_type": 0,
        "hot_area_id": 0,
        "pos_status": 0,
        "cash_status": 0,
        "order_type_switch": 0,
        "bind_area_type": 0
    },
    "environment_type": "骑手在线上环境",
    "work_status": "在线",
    "last_upload_time": "2021-07-16 14:14:22",
    "apollo_lat": 0,
    "apollo_lng": 0,
    "apollo_Conf": "",
    "reason": "骑手超出查询范围!请更换查询经纬度后尝试!"
```

5.如果骑手插入的时间过长，即大于5分钟，策略是默认离线，不会进行召回。可看last_upload_time所在。

3.2 时间线

需求背景


骑手ID 单行输入 搜索

骑手信息

是否配置pre白名单：否　　　是否虚拟坐标：34.639736, 135.542340

骑手工作类型(1全职2兼职3外包)

数据源	城市ID	在线状态	lat	lng	上报时间	hot_area_id	work_type	vehicle_type	receive_type	pos_status	cash_status	order_type_switch	bind_area_type	available
Feature														
courier														
rider														

骑手状态

2021-07-20 20:00　2021-07-21 20:00　查询

上下线状态：🟩 在线 🟧 暂停接单 ⬜ 下线

00:24~01:40 在线

20:00 21:00 23:00 00:00 01:00 02:00 20:00

轨迹状态：🟩 听单 🟦 配送中 ⬜ 下线

在线时长：🟩 平峰在线 🟥 高峰在线 ⬜ 不在线

点击后显示在线状态，格式：
起始时间~结束时间 在线状态
状态：0. 下线，1. 在线，2. 暂停接单
下面的线包括，听单，配送中（trip=id），下线有坐标上报

输入骑手id搜索后默认
当前时间-24h ~ 当前时间
到后端查询
之后用户可以自己调节查询时间，时间范围最大24h

需求分解

track-timeline

该需求其实就是为了展示骑手一天的订单状态

1. courier-lbs服务获取骑手一天的轨迹信息，如下所示：

```
Time        int64   `kim:"time_series"`
TripId      int64   `variable:"trip_id"`
Speed       int64   `variable:"speed"`
Lat         float64 `variable:"lat"`
Lng         float64 `variable:"lng"`
Accuarcy    int64   `variable:"accuarcy"`
WorkStatus  int8    `variable:"work_status"`
HexGridId   int64   `variable:"hex_grid_id"`
LocalAreaId int64   `variable:"local_area_id"`
RiderId     int64   `stable:"rider_id" `
CityId      int64   `stable:"city_id" `
Channel     int64   `stable:"channel"`
Property    map[string]interface{}
```

2. 将若干轨迹信息的点，映射track-time到如需求所示的图上

change_status-timeline

1. 数据库查询语句

```
SELECT
  rider_id,
  cabinet_status,
  cabinet_status_before,
  update_time
FROM
  rider_location_change_record
WHERE
  rider_id = {rider_id}
  and cabinet_status!=cabinet_status_before
  and report_time >= {start} and report_time <= {end}
  order by update_time
```

2. 查询出的数据

```
5764611612472248981        0        1        2021-08-06 07:06:16.205
5764611612472248981        1        0        2021-08-06 07:06:19.154
5764611612472248981        0        1        2021-08-06 07:06:21.177
5764611612472248981        1        0        2021-08-06 07:06:23.047
5764611612472248981        0        1        2021-08-06 07:06:25.078
5764611612472248981        1        0        2021-08-06 07:06:26.719
5764611612472248981        0        1        2021-08-06 07:06:28.538
5764611612472248981        1        0        2021-08-06 07:06:30.441
5764611612472248981        0        1        2021-08-06 07:06:32.138
5764611612472248981        1        0        2021-08-06 07:06:34.253
5764611612472248981        0        1        2021-08-06 07:06:36.064
5764611612472248981        1        0        2021-08-06 07:06:37.939
5764611612472248981        0        1        2021-08-06 07:06:40.030
5764611612472248981        1        0        2021-08-06 07:06:44.640
5764611612472248981        0        1        2021-08-06 07:06:46.974
5764611612472248981        1        0        2021-08-06 07:06:48.876
5764611612472248981        0        1        2021-08-06 07:06:50.607
5764611612472248981        1        0        2021-08-06 07:06:52.347
```

online_status_timeline

1. 数据库查询语句

```
SELECT
  entity_id,
  start_time,
  end_time,
  peak_flag
FROM
  {table}
WHERE
  entity_id = {rider_id} and start_time>={start} and end_time<={end}
```

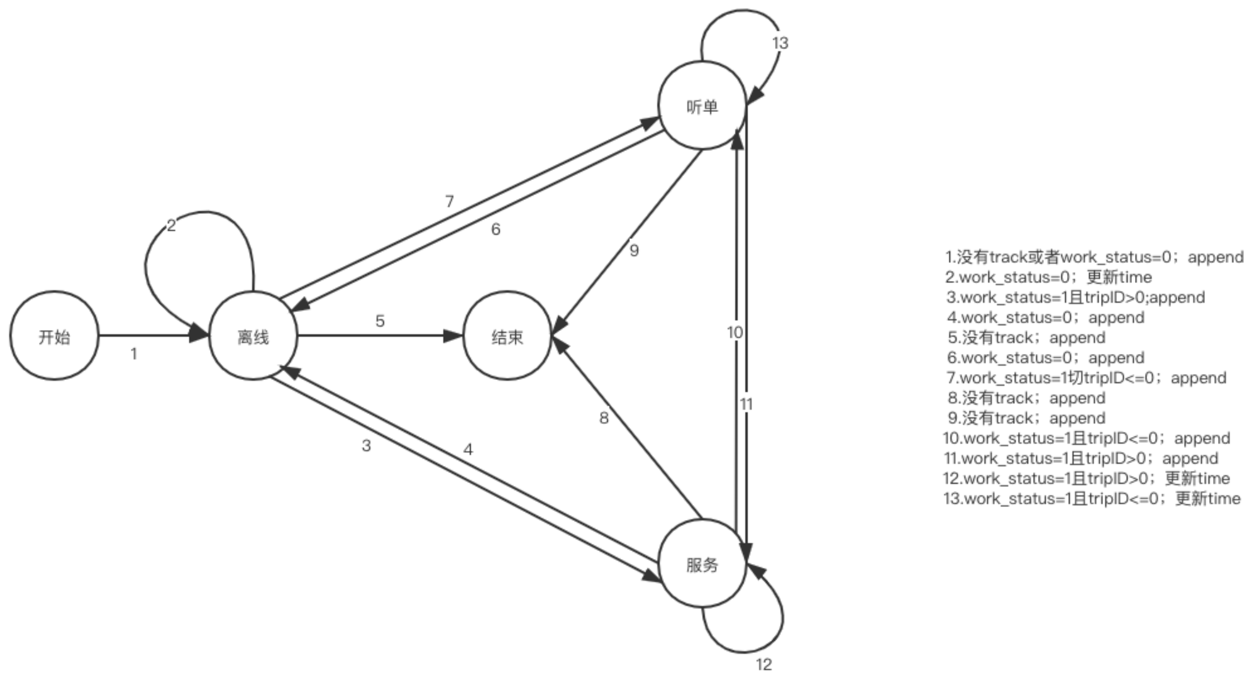
2. 查询出的数据

```
5764608030716988416        1595767380        1595767440        0
5764608030716988416        1595767440        1595767500        0
5764608030716988416        1595767500        1595767560        0
5764608030716988416        1595767560        1595767620        0
5764608030716988416        1595767620        1595767680        0
5764608030716988416        1595767680        1595767740        0
5764608030716988416        1595767740        1595767800        0
5764608030716988416        1595767800        1595767860        0
5764608030716988416        1595767860        1595767920        0
5764608030716988416        1595767920        1595767980        0
5764608030716988416        1595767980        1595768040        0
5764608030716988416        1595768040        1595768100        0
5764608030716988416        1595768100        1595768160        0
5764608030716988416        1595768160        1595768220        0
5764608030716988416        1595768220        1595768280        0
5764608030716988416        1595768280        1595768340        0
5764608030716988416        1595768340        1595768400        0
5764608030716988416        1595768400        1595768460        0
```

任务完成

track_timeline

track-timeline采用有限自动机的方法来完成



1.没有track或者work_status=0；append
2.work_status=0；更新time
3.work_status=1且tripID>0;append
4.work_status=0；append
5.没有track；append
6.work_status=0；append
7.work_status=1切tripID<=0；append
8.没有track；append
9.没有track；append
10.work_status=1且tripID<=0；append
11.work_status=1且tripID>0；append
12.work_status=1且tripID>0；更新time
13.work_status=1且tripID<=0；更新time

(1)获取输入startTime所在天0点的unix时间

```
func CurrentDay(ts int64) time.Time {
   startTime := time.Unix(ts, 0)
   startDay := time.Date(startTime.Year(), startTime.Month(), startTime.
Day(), 0, 0, 0, 0, time.Local)
   return startDay
}
```

(2)根据startTime获取骑手一整天的轨迹

```
deliveryStaus := make([][]TimeLineStatus, 0)
st := CurrentDay(start)
startTime := st.Unix()
endTime := st.Add(time.Hour * 24).Unix()
if endTime > time.Now().Unix() {
   endTime = time.Now().Unix()
}
tracks_, _ := getTrackByRiderID(trace.NewContext(ctx, nil), false,
riderID, startTime, endTime)
tracks := tracks_.(TrackDatas)
tracksStatus := GetTimeLineStatus(ctx, tracks, riderID, st.Unix(), st.Add
(time.Hour*24).Unix())
deliveryStaus = append(deliveryStaus, tracksStatus)
return deliveryStaus
```

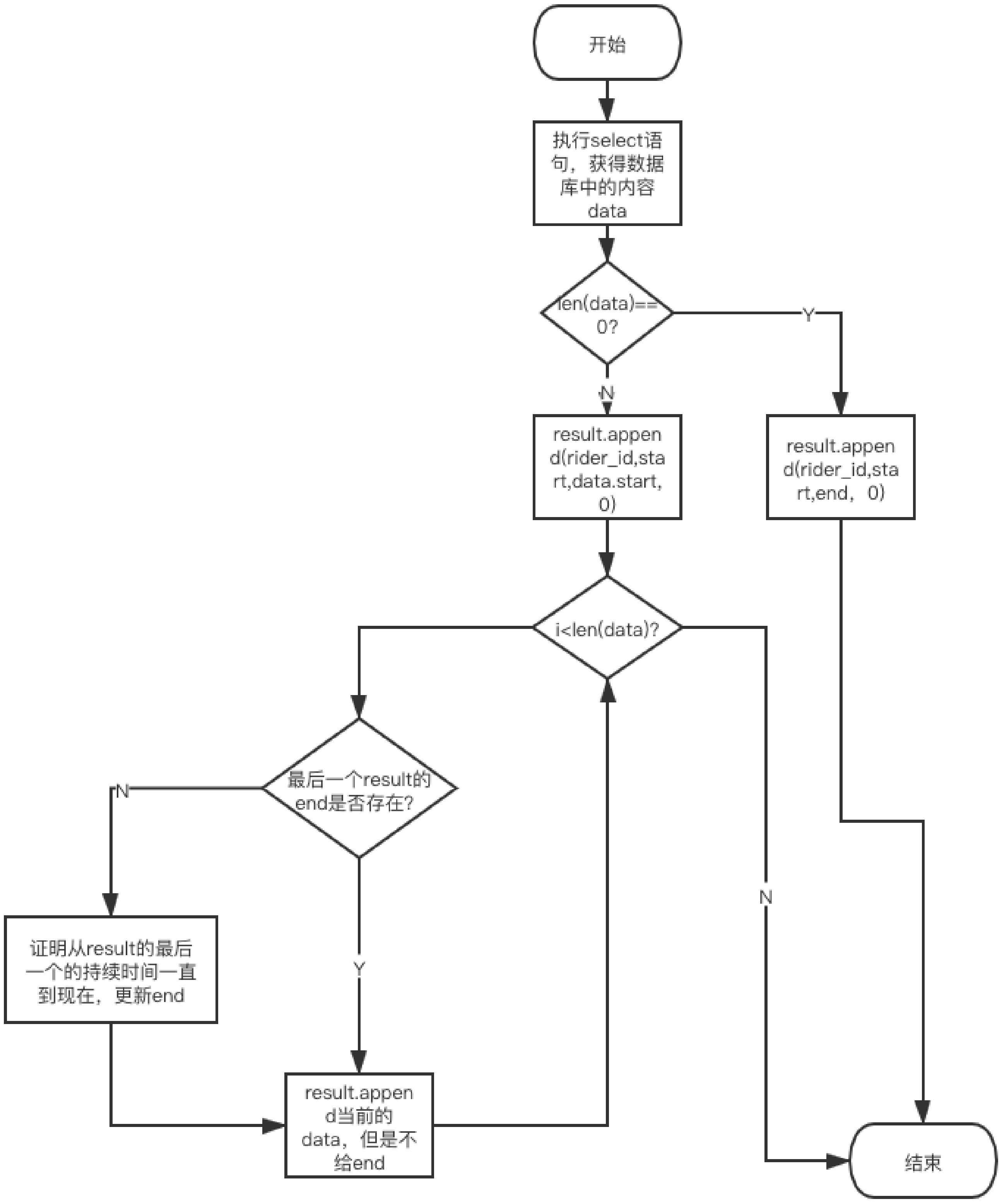
(3)对轨迹进行有限自动机处理

```
res := make([]TimeLineStatus, 0)
var track *TrackData
if len(tracks) == 0 {
   res = append(res, TimeLineStatus{
      Start:   start,
      End:     end,
      Type:    0,
      TripID:  0,
      RiderID: riderID,
      CityID:  0,
      Channel: 1,
   })
   return res
}
//FSM
fsm := fsm.NewFSM(
   "",
   fsm.Events{
      //   Src         Dst
      {Name: "-", Src: []string{""}, Dst: ""},
      {Name: "-", Src: []string{""}, Dst: ""},
      {Name: "-", Src: []string{""}, Dst: ""},
      {Name: "-", Src: []string{""}, Dst: ""},
      {Name: "-", Src: []string{""}, Dst: ""},
      {Name: "-", Src: []string{""}, Dst: ""},
      {Name: "-", Src: []string{""}, Dst: ""},
      {Name: "-", Src: []string{""}, Dst: ""},
      {Name: "-", Src: []string{""}, Dst: ""},
   },
   fsm.Callbacks{
      //
      "before_-": func(e *fsm.Event) {
         res = append(res, TimeLineStatus{
            Start:   track.Time,
            End:     track.Time,
            Type:    1,
            TripID:  track.TripId,
            RiderID: track.RiderId,
            CityID:  track.CityId,
            Channel: track.Channel,
         })
      },
      "before_-": func(e *fsm.Event) {
         res = append(res, TimeLineStatus{
            Start:   track.Time,
            End:     track.Time,
            Type:    2,
            TripID:  track.TripId,
            RiderID: track.RiderId,
            CityID:  track.CityId,
            Channel: track.Channel,
         })
      },
      "before_-": func(e *fsm.Event) {
         res[len(res)-1].End = track.Time
      },
      "before_-": func(e *fsm.Event) {
         res[len(res)-1].End = track.Time
      },
      "before_-": func(e *fsm.Event) {
         res = append(res, TimeLineStatus{
            Start:   track.Time,
            End:     track.Time,
            Type:    2,
            TripID:  track.TripId,
            RiderID: track.RiderId,
            CityID:  track.CityId,
            Channel: track.Channel,
         })
      },
      "before_-": func(e *fsm.Event) {
         res = append(res, TimeLineStatus{
            Start:   track.Time,
            End:     track.Time,
            Type:    1,
            TripID:  track.TripId,
            RiderID: track.RiderId,
            CityID:  track.CityId,
            Channel: track.Channel,
         })
      },
      "before_-": func(e *fsm.Event) {
         res = append(res, TimeLineStatus{
            Start:   track.Time,
            End:     track.Time,
            Type:    0,
            TripID:  track.TripId,
            RiderID: track.RiderId,
            CityID:  track.CityId,
            Channel: track.Channel,
         })
      },
      "before_-": func(e *fsm.Event) {
         res = append(res, TimeLineStatus{
            Start:   track.Time,
            End:     track.Time,
            Type:    0,
            TripID:  track.TripId,
            RiderID: track.RiderId,
            CityID:  track.CityId,
            Channel: track.Channel,
         })
      },
      "before_-": func(e *fsm.Event) {
         res[len(res)-1].End = track.Time
      },
   },
)
//track
var k = 0
for i := 0; i < len(tracks); i++ {
   if tracks[i].WorkStatus != 0 {
      break
   }
   k = i
}
if tracks[k].Time-start > 0 {
   res = append(res, TimeLineStatus{
      Start:   start,
      End:     tracks[k].Time,
      Type:    0,
      TripID:  tracks[k].TripId,
      RiderID: tracks[k].RiderId,
      CityID:  tracks[k].CityId,
      Channel: tracks[k].Channel,
   })
}
for i := k; i < len(tracks); i++ {
   track = tracks[i]
```

```
      if track.WorkStatus != 0 {
         if track.TripId == 0 {
            if fsm.Current() == "" {
               fsm.Event("-")
            } else if fsm.Current() == "" {
               fsm.Event("-")
            } else {
               fsm.Event("-")
            }
         } else {
            if fsm.Current() == "" {
               fsm.Event("-")
            } else if fsm.Current() == "" {
               fsm.Event("-")
            } else {
               fsm.Event("-")
            }
         }
      } else {
         if fsm.Current() == "" {
            fsm.Event("-")
         } else if fsm.Current() == "" {
            fsm.Event("-")
         } else if fsm.Current() == "" {
            fsm.Event("-")
         }
      }
   }
   //track
   if end-tracks[len(tracks)-1].Time > 50 {
      res = append(res, TimeLineStatus{
         Start:   tracks[len(tracks)-1].Time,
         End:     end,
         Type:    0,
         TripID:  track.TripId,
         RiderID: track.RiderId,
         CityID:  track.CityId,
         Channel: track.Channel,
      })
   }
   return res
```

change_status-timeline

骑手上下线状态变化的时间线实现起来比较简单，需要注意的是，查询出来的数据代表的是骑手状态发生变更的时间，流程图如下：



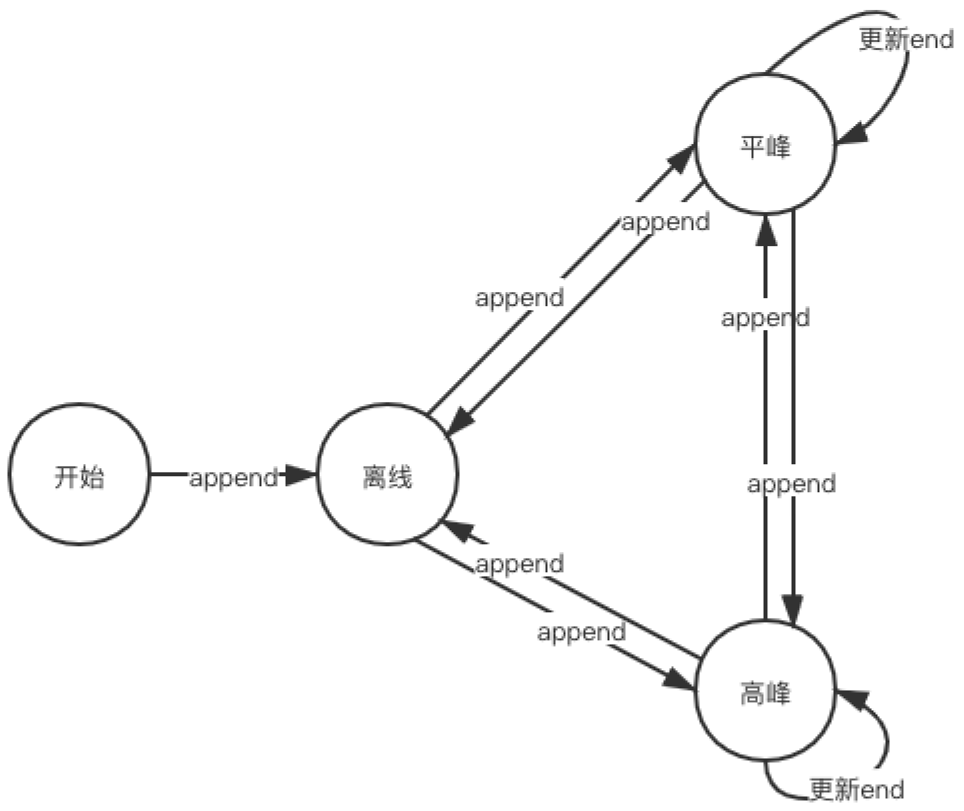
开始
执行select语句，获得数据库中的内容data
len(data)==0?
Y
N
result.append(rider_id,start,data.start,0)
result.append(rider_id,start,end，0)
i<len(data)?
N
最后一个result的end是否存在?
N
Y
证明从result的最后一个的持续时间一直到现在，更新end
result.append当前的data，但是不给end
结束

根据上述流程图实现代码：

```
def get_rider_status_change_timeline(rider_id, start, end):
    data = load_courier_status(rider_id, start, end)
    result = []
    if len(data) != 0:
        result.append({"rider_id": rider_id,
                       "start": time.strftime("%Y-%m-%d %H:%M:%S.000000",
time.localtime(start)),
                       "end": str(data.iloc[0]['update_time']),
                       "type": data.iloc[0]['cabinet_status_before']
                       })
    else:
        result.append({"rider_id": rider_id,
                       "start": time.strftime("%Y-%m-%d %H:%M:%S.000000",
time.localtime(start)),
                       "end": time.strftime("%Y-%m-%d %H:%M:%S.000000",
time.localtime(end)),  # update_time
                       "type": 0
                       })
        logging.info("%s||_get_rider_status_change_timeline||rider_id=%
d||start=%d||end=%d||result_len=%d",
                     time.strftime("%Y-%m-%d-%H_%M_%S", time.localtime()),
rider_id, start, end, len(result))
        return result
    for i in range(len(data)):
        if not result[len(result) - 1]['end']:
            result[len(result) - 1]['end'] = str(data.iloc[i]
['update_time'])
        result.append({"rider_id": rider_id,
                       "start": str(data.iloc[i]['update_time']),
                       "end": [],
                       "type": data.iloc[i]['cabinet_status']
                       })
    if not result[len(result) - 1]['end']:
        result[len(result) - 1]['end'] = time.strftime("%Y-%m-%d %H:%M:%S.
000000", time.localtime(end))
    logging.info("%s||_get_rider_status_change_timeline||rider_id=%
d||start=%d||end=%d||result_len=%d",
                 time.strftime("%Y-%m-%d-%H_%M_%S", time.localtime()),
rider_id, start, end, len(result))
    return result
```

online_status_timeline

online_status_timeline也需要状态机，但其逻辑要比track_timeline简单，状态机如下：



根据上述状态机代码实现如下：

```
def get_rider_online_timeline(rider_id, start, end):
    result = []
    table = "d_online_duration_{}".format(rider_id % 1024)
    data = load_courier_online_time(rider_id, start, end, table)
    if len(data) != 0:
        #
        if start != data.iloc[0]['start_time']:
            result.append({"rider_id": data.iloc[0]['entity_id'],
                           "start": start,
                           "end": data.iloc[0]['start_time'],
                           "type": 0
                           })
        else:
            result.append({"rider_id": data.iloc[0]['entity_id'],
                           "start": data.iloc[0]['start_time'],
                           "end": data.iloc[0]['end_time'],
                           "type": data.iloc[0]['peak_flag'] + 1
                           })
    else:
        #
        result.append({"rider_id": data.iloc[0]['entity_id'],
                       "start": start,
                       "end": end,  # update_time
                       "type": 0  # cabinet_status_before
                       })
        logging.info(
            "%s||_get_rider_online_timeline||rider_id=%d||start=%d||end=%
d||table_name=%s||result_len=%d",
            time.strftime("%Y-%m-%d-%H_%M_%S", time.localtime()),
rider_id, start, end, table, len(result))
        return result
    for i in range(len(data)):
        last_i = len(result) - 1
        if result[last_i]['end'] != data.iloc[i]['start_time'] and result
[last_i]['end'] < data.iloc[i]['start_time']:
            result.append({"rider_id": data.iloc[i]['entity_id'],
                           "start": result[last_i]['end'],
                           "end": data.iloc[i]['start_time'],
                           "type": 0
                           })
        last_i = len(result) - 1
        if result[last_i]['type'] == 0 or result[last_i]['type'] != data.
iloc[i]['peak_flag'] + 1:
            result.append({"rider_id": data.iloc[i]['entity_id'],
                           "start": result[last_i]['end'],
                           "end": data.iloc[i]['end_time'],
                           "type": data.iloc[i]['peak_flag'] + 1
                           })
        else:
            result[last_i]['end'] = data.iloc[i]['end_time']
    if result[len(result) - 1]['end'] != end:
        result.append({"rider_id": data.iloc[0]['entity_id'],
                       "start": result[len(result) - 1]['end'],
                       "end": end,  # update_time
                       "type": 0  # cabinet_status_before
                       })
    logging.info(
        "%s||_get_rider_online_timeline||rider_id=%d||start=%d||end=%
d||table_name=%s||result_len=%d",
        time.strftime("%Y-%m-%d-%H_%M_%S", time.localtime()), rider_id,
start, end, table, len(result))
    return result
```

四. go当中的测试覆盖率

1. 需求背景

一个好的测试对于一个项目的稳定性建设来说较为重要，不能完全依赖QA，所以打算运用一个比较完备的测试工具。并且对核心模块memdb进行功能测试。

实现效果：进行make test即将项目中所有的test文件都执行一遍，如果test文件中能够包含项目当中比较核心功能测试的话，那么每次修改完代码进行一次make test就可以很好的防治出错；并且可以通过代码覆盖率来查看有哪些边界情况没有执行到，防治意外情况发生。

2.任务完成

1.以courier-lbs项目举例

首先将所有的test文件调通

2.在makefile当中进行功能建设

```
date
echo "make(install_apolloapollo)"
nohup ./output/bin/courier-lbs >/private/tmp/nohup.out 2>&1 &
echo "{\"toggle\":{\"namespace\":\"courier_location\",\"name\":\"
cache_switcher\",\"version\":0,\"last_modify_time\":1622620819515,\"
log_rate\":0,\"cache_plan\":0,\"rule\":{\"subject\":\"bucket\",\"verb\":\"
=\",\"objects\":[[0,1000]]},\"experiment\":{\"groups\":[{\"name\":\"new\",
\"version\":460387,\"rule\":{\"subject\":\"exp_bucket\",\"verb\":\"=\",\"
objects\":[[0,100]]},\"params\":{\"men_or_db\":1,\"mar_cache_switcher\":
0}}]},\"schema_version\":\"1.0.0\"}}">/private/tmp/xiaoju/ep/as/store/conf
/courier_location/cache_switcher
sleep 2s
go clean -testcache
go test  -coverpkg=./... -coverprofile=/private/tmp/cover.out  ./... ./...
go tool cover -html=/private/tmp/cover.out -o /private/tmp/coverage.html
go tool cover -func=/private/tmp/cover.out -o /private/tmp/cover.txt
tail -n 1 /private/tmp/cover.txt | awk '{print $$1,$$3}'
open /private/tmp/coverage.html
killall courier-lbs
date
```

1第三行是为了后台运行courier-lbs项目，因为在测试文件当中会有一些需要rpc调用的函数，之所以需要rpc调用，是因为普通的测试文件调用函数没法产生日志，只有rpc调用才会产生日志，并使用同样的cache.

2第四行是为了修改本地apollo文件，主要修改地方是"last_modify_time"和"men_or_db",主要是为了serachByCoords函数当中使用cache而不是db

3第六行主要是为了清除go test缓存，防治上次的test影响

4第七行是扫描项目当中所有的文件

5第八行和第九行是为了形成代码覆盖率的html和txt，在txt当中保存了每个函数的覆盖率和调用次数，而在html当中用颜色来表明没有被测试过的逻辑

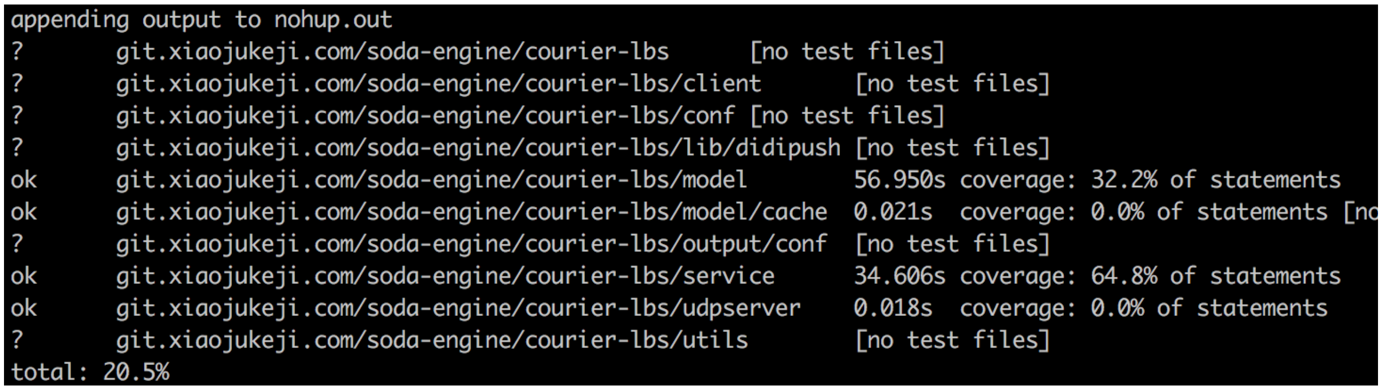
6第十行是输出总体的覆盖率

7第十二行是为了杀死后台启动的courier-lbs进程

3.结果如下

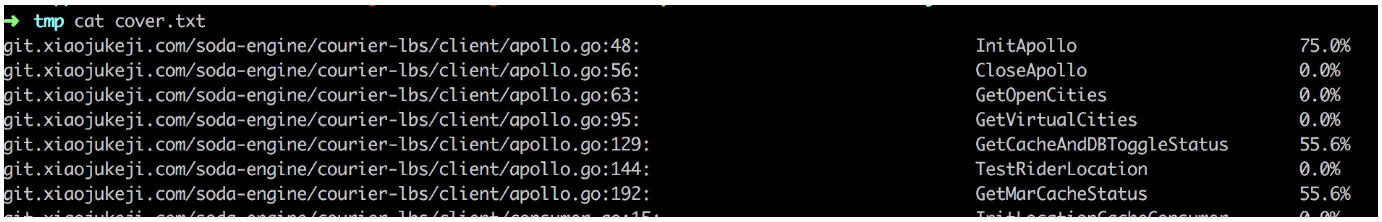
在终端中输入make test

首先后台启动courier-lbs项目，接着进行代码覆盖率的产生



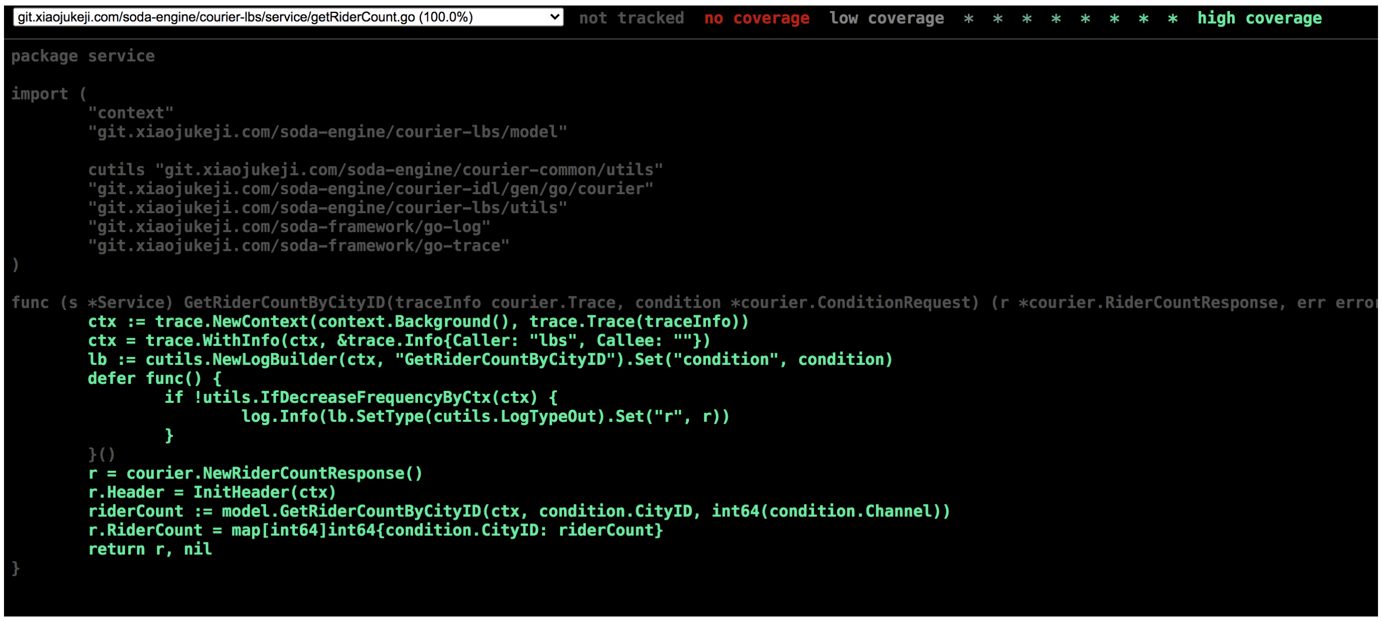
```
appending output to nohup.out
?       git.xiaojukeji.com/soda-engine/courier-lbs      [no test files]
?       git.xiaojukeji.com/soda-engine/courier-lbs/client       [no test files]
?       git.xiaojukeji.com/soda-engine/courier-lbs/conf [no test files]
?       git.xiaojukeji.com/soda-engine/courier-lbs/lib/didipush [no test files]
ok      git.xiaojukeji.com/soda-engine/courier-lbs/model        56.950s coverage: 32.2% of statements
ok      git.xiaojukeji.com/soda-engine/courier-lbs/model/cache  0.021s  coverage: 0.0% of statements [no
?       git.xiaojukeji.com/soda-engine/courier-lbs/output/conf  [no test files]
ok      git.xiaojukeji.com/soda-engine/courier-lbs/service      34.606s coverage: 64.8% of statements
ok      git.xiaojukeji.com/soda-engine/courier-lbs/udpserver    0.018s  coverage: 0.0% of statements
?       git.xiaojukeji.com/soda-engine/courier-lbs/utils        [no test files]
total: 20.5%
```

产生每个文件当中函数的代码覆盖率



```
➜ tmp cat cover.txt
git.xiaojukeji.com/soda-engine/courier-lbs/client/apollo.go:48:                 InitApollo              75.0%
git.xiaojukeji.com/soda-engine/courier-lbs/client/apollo.go:56:                 CloseApollo             0.0%
git.xiaojukeji.com/soda-engine/courier-lbs/client/apollo.go:63:                 GetOpenCities           0.0%
git.xiaojukeji.com/soda-engine/courier-lbs/client/apollo.go:95:                 GetVirtualCities        0.0%
git.xiaojukeji.com/soda-engine/courier-lbs/client/apollo.go:129:                GetCacheAndDBToggleStatus       55.6%
git.xiaojukeji.com/soda-engine/courier-lbs/client/apollo.go:144:                TestRiderLocation       0.0%
git.xiaojukeji.com/soda-engine/courier-lbs/client/apollo.go:192:                GetMarCacheStatus       55.6%
git.xiaojukeji.com/soda-engine/courier-lbs/client/consumer.go:15:               InitLocationCacheConsumer       0.0%
```

最后自动打开覆盖率的html



git.xiaojukeji.com/soda-engine/courier-lbs/service/getRiderCount.go (100.0%)
not tracked
no coverage
low coverage
* * * * * * * *
high coverage

```
package service

import (
        "context"
        "git.xiaojukeji.com/soda-engine/courier-lbs/model"

        cutils "git.xiaojukeji.com/soda-engine/courier-common/utils"
        "git.xiaojukeji.com/soda-engine/courier-idl/gen/go/courier"
        "git.xiaojukeji.com/soda-engine/courier-lbs/utils"
        "git.xiaojukeji.com/soda-framework/go-log"
        "git.xiaojukeji.com/soda-framework/go-trace"
)

func (s *Service) GetRiderCountByCityID(traceInfo courier.Trace, condition *courier.ConditionRequest) (r *courier.RiderCountResponse, err erro
        ctx := trace.NewContext(context.Background(), trace.Trace(traceInfo))
        ctx = trace.WithInfo(ctx, &trace.Info{Caller: "lbs", Callee: ""})
        lb := cutils.NewLogBuilder(ctx, "GetRiderCountByCityID").Set("condition", condition)
        defer func() {
                if !utils.IfDecreaseFrequencyByCtx(ctx) {
                        log.Info(lb.SetType(cutils.LogTypeOut).Set("r", r))
                }
        }()
        r = courier.NewRiderCountResponse()
        r.Header = InitHeader(ctx)
        riderCount := model.GetRiderCountByCityID(ctx, condition.CityID, int64(condition.Channel))
        r.RiderCount = map[int64]int64{condition.CityID: riderCount}
        return r, nil
}
```

3.遇到问题

1.首先来说，courier-lbs当中的某些测试函数需要修改本地apollo来达到测试的目的，我们可以使用echo等linux功能来进行文件书写，来修改本地的apollo。

但当我们使用make或者install_apollo来恢复本地apollo时，会遇到一些问题：

时间线如下：

进程A make test

进程A echo修改本地apollo

进程A 修改好本地apollo

开始进行测试文件的运行

进程A make恢复apollo，但此时进程A并没有修改本地apollo文件，大概需要10s-20s的时间才会修改本地apollo文件

而这时

进程B make test

进程B echo修改本地apollo

进程B 修改好本地apollo

进程A make恢复apollo，修改好本地apollo文件。————————————————这就会产生问题，在courier-lbs当中的话是可以修改本地apollo而不再修改回去的，而如果需要修改回去的话那可以 sleep 10s来解决这个问题

开始进程测试文件的运行

进程B make恢复apollo

2.go测试无法跨包的问题

问题定义：

假设项目结构

`stuff/stuff.go`

`test/stuff/stuff_test.go`

尽管stuff_test从stuff.go执行代码，但显示`coverage: 0.0% of statements`

解决：在go test后面加上 -coverpkg=./...

```
go test  -coverpkg=./... -coverprofile=/private/tmp/cover.out  ./... ./...
```

4.接入策略

1.将项目当中的test文件全部都调试通过

2.在makefile当中加入下述语句即可生成相应的代码覆盖率html，结果保存在tmp当中

```
test:
   date
   echo "make(install_apolloapollo)"
   go clean -testcache
   go test  -coverpkg=./... -coverprofile=/private/tmp/cover.out  ./... .
/...
   go tool cover -html=/private/tmp/cover.out -o /private/tmp/coverage.html
   go tool cover -func=/private/tmp/cover.out -o /private/tmp/cover.txt
   tail -n 1 /private/tmp/cover.txt | awk '{print $$1,$$3}'
   open /private/tmp/coverage.html
   date
```

3.运行make test，即可进行相应的效果实现

4.

注意1:在make test过程中可能会因为前面test文件的一些操作，导致后面test文件发生错误，这也需要相应的调整。

注意2:如果项目当中的test文件需要特殊的条件，如rpc调用、修改本地apollo等，都需要在makefile当中进行相应修改，上述代码只是最基本修改。

5.内存DB的学习和测试

1. schema(模式)及radixtreecache

schema就是数据库对象的集合，这个集合包含了各种对象如：表、视图、存储过程、索引等

初始化schema函数如下：

```
func NewRadixRiderSchema() (string, memdb.DBSchema) {
   table := "riders"
   return table, memdb.DBSchema{
      Tables: map[string]*memdb.TableSchema{
         table: &memdb.TableSchema{
            Name: table,
            Indexes: map[string]*memdb.IndexSchema{
               "id": &memdb.IndexSchema{
                  Name:    "id",
                  Unique:  true,
                  Indexer: &memdb.IntFieldIndex{Field: "RiderID"},
               },
               "grid": &memdb.IndexSchema{
                  Name:    "grid",
                  Unique:  false,
                  Indexer: &memdb.IntFieldIndex{Field: "Grid"},
               },
            },
         },
      },
   }
}
```

schema代码测试函数：

如上述schema初始化函数所示，初始化一个scheme，schema中包含了很多map结构，表的map结构为map[string]*memdb.TableSchema，定义一个名为"riders"的表，这个表也是一个map结构，map结构为map[string]*memdb.IndexSchema，表中有两列，一列是id，一列是grid。

```
func TestInitCache(t *testing.T) {
   level7Table, level7Schema := NewRadixRiderSchema()
   assert.Equal(t, "riders", level7Table, "table name should be same")
   assert.Equal(t, "id", level7Schema.Tables[level7Table].Indexes["id"].
Name, "column name should be same")
   assert.Equal(t, "grid", level7Schema.Tables[level7Table].Indexes
["grid"].Name, "column name should be same")
   Grid7CacheTest = NewRadixTreeCache("level7", level7Table, 7,
level7Schema)
}
```

当初始化完schema后，要根据初始化的schema重新初始化一个radixtreecache（感觉radix_tree和schema没有什么差别啊？）

```
func NewRadixTreeCache(name, table string,gridLevel int, schema memdb.
DBSchema) *RadixTreeCache {
   return &RadixTreeCache{
      Name: name,
      table: table,
        GridLevel: gridLevel,
        SchemaTemplate: schema,
    }
}
```

2. radixDB

获得了radixtreecache之后，再在外面封装一点东西，就变成了radixDB

radixDB其实主要是根据cityID进行分表保存，主要的就是cityID，memdb.MemDB,TTL当中存放的是*******

```
type RadixDB struct {
    Name   string
    table  string
    CityID int64
    DB  *memdb.MemDB
    TTL *SortedSet
    RWLock sync.RWMutex // TTL
}
```

其初始化函数如下：

```
func NewRadixDB(name, table string, cityID int64, schema *memdb.DBSchema)
*RadixDB{
    db, err := memdb.NewMemDB(schema)
    if err != nil{
        log.Errorf("_RadixTreeCache_GetTable_Err||err=%+v", err)
        return nil
    }
    radixDB := &RadixDB{
        Name: name,
        table: table,
        CityID: cityID,
        DB: db,
        TTL:NewSortedSet(),
    }
    radixDB.init()
    return radixDB
}


func (db *RadixDB) init() {
    go db.ExpireLoop()
}


func (db *RadixDB) ExpireLoop() {
    for {
        ctx := trace.NewContext(context.Background(), nil)
        db.Expire(ctx)
        time.Sleep(time.Duration(60) * time.Second)
```

```
   }
}


func (db *RadixDB) Expire(ctx context.Context) {
   startTime := time.Now()
   //  db.RWLock.RUnlock()panic
   db.RWLock.Lock()
   riders := db.TTL.GetValueByScoreRange(0, time.Now().Add(-1*
RadixExpireTime).Unix(), 200)

   deleteCount := 0
   txn := db.DB.Txn(true)
   for _, rider := range riders {
      err := txn.Delete(db.table, rider)
      if err != nil && err != memdb.ErrNotFound {
         continue
      }
      db.TTL.Remove(db.GetTTLKey(rider.(*RadixRider)))
      deleteCount += 1
   }
   txn.Commit()
   db.RWLock.Unlock()

   tags := metrics.Tags{"caller": "expire", "callee": fmt.Sprintf("%d", db.
CityID)}
   metrics.Send(ctx, &metrics.Counter{Name: fmt.Sprintf("%s.radixDB.
expire", db.Name), Count: deleteCount, Tags: tags})
   metrics.Send(ctx, &metrics.Distribution{Name: fmt.Sprintf("%s.radixDB.
expire.time", db.Name), Value: time.Now().Sub(startTime),
      Percentages: []metrics.Percentage{metrics.P99, metrics.P95, metrics.
P75, metrics.P50},
   })
}
```

测试函数如下：

```
func TestNewRadixDB(t *testing.T) {
   TestInitCache(t)
   var cityID = int64(52140500)
   radixDB := NewRadixDB(Grid7CacheTest.Name, Grid7CacheTest.table,
cityID, &Grid7CacheTest.SchemaTemplate)
   assert.Equal(t, Grid7CacheTest.Name, radixDB.Name, "db name should be
same")
   assert.Equal(t, Grid7CacheTest.table, radixDB.table, "table name should
be same")
   assert.Equal(t, cityID, radixDB.CityID, "cityID should be same")
}
```

用的时候一般就是用radixDB

获取radixDB，sync.map，key是cityID，value是memdb

```
func (c *RadixTreeCache) GetDB(cityID int64) *RadixDB {
   _db, ok := c.Sharding.Load(cityID)
   if !ok {
      radixDB := NewRadixDB(c.Name, c.table, cityID, &c.SchemaTemplate)
      c.Sharding.Store(cityID, radixDB)
      return radixDB
   }
   return _db.(*RadixDB)
}
```

3.增、删、改、查

1.insert函数

```
func (c *RadixTreeCache) Insert(cityID int64, rider *RadixRider) error{
   db := c.GetDB(cityID)
   txn := db.DB.Txn(true)
   err := txn.Insert(db.table, rider)
   txn.Commit()
   if err == nil{
      db.RWLock.Lock()
      db.TTL.SetByCurTime(db.GetTTLKey(rider), rider)
      db.RWLock.Unlock()
   }
   return err
}
```

其实主要是先根据cityID获取对应的cityID的DB

然后根据DB获取相应的txn，根据txn进行插入，txn是一个事物，使用txn可以对表进行相应的插入、删除、修改功能。

测试函数：

```
func TestInsert(t *testing.T) {
   TestInitCache(t)
   var cityID = int64(12345)
   NewRadixDB(Grid7CacheTest.Name, Grid7CacheTest.table, cityID,
&Grid7CacheTest.SchemaTemplate)
   rider := RadixRider{
      RiderID:    12345,
      Lat:        20.6575610,
      Lng:        -103.2979428,
      Time:       time.Now().Unix(),
      Channel:    0,
      RiderLoad:  0,
      AutoStatus: 1,
   }
   Grid7CacheTest.InsertWithCalculateGrid(cityID, &rider)
   riders, err := Grid7CacheTest.GetWithPrefixWithFilter(cityID, "id", func
(rider *RadixRider) bool {
      return true
```

```
    }, rider.RiderID)
    if err == nil {
        for i := 0; i < len(riders); i++ {
            assert.Equal(t, rider, *riders[i], "insert value not equal
serarched value") //
        }
    }

}
```

2. 删除函数

类似上述函数，使用txn进行删除，并相应的删除TTLSet当中的内容

```
func (db *RadixDB) Remove(riderID int64) error {
    txn := db.DB.Txn(true)
    defer txn.Commit()
    rider := &RadixRider{RiderID: riderID}
    err := txn.Delete(db.table, rider)
    if err != nil && err != memdb.ErrNotFound {
        return err
    }
    db.RWLock.Lock()
    db.TTL.Remove(db.GetTTLKey(rider))
    db.RWLock.Unlock()
    return nil
}
```

测试函数：

```
func TestRemove(t *testing.T) {
    TestInsert(t)
    var cityID = int64(12345)
    var riderID = int64(12345)
    Grid7CacheTest.GetDB(cityID).Remove(riderID)
    db := Grid7CacheTest.GetDB(cityID)
    txn := db.DB.Txn(false)
    it, err := txn.Get(db.table, "id", riderID)
    assert.Equal(t, nil, err, "txnrider")
    assert.Equal(t, nil, it.Next(), "txn")
}
```

3. 查找函数

```
func (c *RadixTreeCache) GetWithPrefixWithFilter(cityID int64, index
string, filter func(rider *RadixRider) bool, args...interface{}) ([]
*RadixRider, error) {
    var riders []*RadixRider
```

```
   it, err := c.GetWithPrefix(cityID, index, args...)
   if err!= nil{
      return nil, err
   }
   for obj := it.Next(); obj != nil; obj=it.Next(){
      if filter(obj.(*RadixRider)) {
         riders = append(riders, obj.(*RadixRider))
      }
   }
   return riders, nil
}


func (c *RadixTreeCache) GetWithPrefix(cityID int64, index string, args...
interface{}) (memdb.ResultIterator, error) {
   db := c.GetDB(cityID)
   txn := db.DB.Txn(false)
   return txn.Get(db.table, index, args...)
}
```

测试函数:

4. 修改函数

```
func UpdateRadixTreeAutoStatus(tree *RadixTreeCache,cityID int64, riderID
int64, autoStatus int64) error {
   db := tree.GetDB(cityID)
   txn := db.DB.Txn(true)
   defer txn.Commit()
   it, err := txn.Get(db.table, "id", riderID)
   if err != nil {
      return err
   }
   var rider *RadixRider
   for obj := it.Next(); obj != nil; obj=it.Next(){
      rider = obj.(*RadixRider)
      rider.AutoStatus = autoStatus
      return txn.Insert(db.table, rider)
   }
   return nil
}
```

测试函数:

```
func TestUpdateRadixTreeAutoStatus(t *testing.T) {
   TestInsert(t)
   var cityID = int64(12345)
   var riderID = int64(12345)
   var autoStatus = int64(1)
   db := Grid7CacheTest.GetDB(cityID)
   txn := db.DB.Txn(false)
   it, err := txn.Get(db.table, "id", riderID)
   assert.Equal(t, nil, err, "_TestUpdateRadixTreeAutoStatus||rider")
   rider := *it.Next().(*RadixRider)
   rider.AutoStatus = autoStatus
   UpdateRadixTreeAutoStatus(Grid7CacheTest, cityID, riderID, autoStatus)
   it, err = txn.Get(db.table, "id", riderID)
   _rider := *it.Next().(*RadixRider)
   assert.Equal(t, rider, _rider, "_TestUpdateRadixTreeAutoStatus||
autostatus")
}
```

五. 紧急需求

5.1 D端骑手坐标流获取不到问题

```
# -*- coding: utf-8 -*
import time
import logging
import requests
import prestodb
import os
from prestodb import dbapi
import pandas as pd
import sys
logging.basicConfig(stream=sys.stdout, level=logging.INFO)
conn=prestodb.dbapi.connect(
    host='',
    port=,
    user='',
    password="",
    resource_group="",
    real_account='',
    catalog=''
)

def GetReq(deliveryId, cityId):
    # url
    url = "http://xx.xxx.xxx.xxx:xxxx/track?delivery={}&city={}".format
(deliveryId, cityId)
    try:
        req = requests.get(url)
    except Exception as ex:
        return None
    return req.json()
```

```
def GetCount(body):
    # url
    if body != None:
        return len(body['features'])
    else:
        return -1

def GetDeliveryCount(deliveryId, cityId):
     #print(deliveryId)
     req = GetReq(deliveryId, cityId)
     count = GetCount(req)
     return count


def GetData(date='',country='', city_id='',length=0):
    cur = conn.cursor()
    sql="select country_code, city_id, delivery_id, curr_rider_id,
create_time, accept_time,update_time from soda_international_dwd.
dwd_delivery_d_increment where concat_ws('-', year, month, day) = '%s' and
status = 160"%(date)#.format("2021-05-10")
    cur.execute(sql)
    #ij
    rows = cur.fetchall()
    df_res = pd.DataFrame(
        columns=['country_code', 'city_id', 'delivery_id',
'curr_rider_id', 'create_time', 'accept_time', 'cost_time', 'count'])
    logging.info("read date={%s} total size=%d",date,len(rows))
    start = t0 = time.time()
    t1=0
    err_num = 0
    if length==0 or length>len(rows):
        length=len(rows)
    for i in range(length):
        if country!="" and country != rows[i][0]:
            continue
        if city_id!="" and city_id != str(rows[i][1]):
            continue
        count = GetDeliveryCount(rows[i][2], rows[i][1])
        if count < 10:
            err_num += 1
        df_res = df_res.append({
            'country_code': rows[i][0],
            'city_id': rows[i][1],
            'delivery_id': rows[i][2],
            'curr_rider_id': rows[i][3],
            'create_time': rows[i][4],
            'accept_time': rows[i][6],
            'cost_time': rows[i][6]-rows[i][4],
            'count': count}, ignore_index=True)
        if len(df_res) % 100 == 0:
            t1 = time.time()
            #print(len(df_res))
            logging.info("err=%d len=%d total=%d cost=%d sec",err_num,len
(df_res),i,int(t1-t0))
            t0 = t1
    logging.info("filter date=%s country=%s city_id=%s size=%d",date,
country,city_id,len(df_res))
    file_name="delivery_{}_{}_{}".format(country,city_id,date)
    #print(file_name)
    df_res.to_csv(file_name, index=False, sep=',')
    logging.info("date=%s country=%s city=%s  row err/total=%d/%d
total_cost=%d sec",date,country,city_id,err_num,len(df_res),int(t1-start))
    conn.close()

GetData("2021-05-10","BR","55000116",0)
```

5.2 courier-dataflow添加mysql字段

```
var writeMysql = func(acInfo ActionInfo) error {
   log.Debug("__________writeMysql______________")
   info, ok := acInfo.Action.(action.LocationReportAction)
   if !ok {
      return nil
   }
   cols := []string{"rider_id", "report_time", "lat", "lng", "grid_id",
"cabinet_status", "local_hot_area_id"}
   values := []interface{}{info.RiderID, info.CreateTime, info.Lat, info.
Lng, info.HexGrid, info.PropertyInfo.WorkStatus, info.PropertyInfo.
LocalHotAreaID}

   // cabinet_statusmqcabinet_statuscabinet_status
   // cabinet_status
   // update_time
   //
   updateValues := []string{
      fmt.Sprintf("rider_id=%d", info.RiderID),
      fmt.Sprintf("report_time=IF(@update_record := (cabinet_status=%d or
(cabinet_status<> %d and (UNIX_TIMESTAMP(update_time) < %d))), %d,
report_time)", info.PropertyInfo.WorkStatus, info.PropertyInfo.WorkStatus,
info.CreateTime-60, info.CreateTime),
      fmt.Sprintf("lat=IF(@update_record, %.7f, lat)", info.Lat),
      fmt.Sprintf("lng=IF(@update_record, %.7f, lng)", info.Lng),
      fmt.Sprintf("grid_id=IF(@update_record, %d, grid_id)", info.HexGrid),
      fmt.Sprintf("cabinet_status=IF(@update_record, %d, cabinet_status)",
info.PropertyInfo.WorkStatus),
      fmt.Sprintf("local_hot_area_id=IF(@update_record, %d,
local_hot_area_id)", info.PropertyInfo.LocalHotAreaID),
   }
   extra := "%v ON DUPLICATE KEY UPDATE " + strings.Join(updateValues[:],
",")
   _, err := cdao.InsertRiderWithExtra(acInfo.ctx, cols, values, extra)

   //_, err = cdao.UpsertRider(acInfo.ctx, cols, values, cols,
updateValues)
```

```
    if err != nil {
        log.Errorf("_writeMysql_Err||%+v||body=%+v||err=%+v", trace.
ContextString(acInfo.ctx), info, err)
        return err
    }
    err = model.DeleteRiderCache(acInfo.ctx, info.RiderID)
    if err != nil {
        log.Errorf("_writeMysql_delete_redis_err||err=%v", err)
    }
    syncToFeature(acInfo)
    return nil
}
```