package connetor 使用说明

依赖包

使用前先用go get 下载下列软件包：

```
"github.com/Shopify/sarama"
"github.com/bsm/sarama-cluster"
"github.com/collinmsn/thrift-client-pool"
"github.com/goojiayong/hbase"
"common-core/aclog"
```

使用thrift 生成"hbase"软件包：

1. 下载hbase的thrift接口说明文件。（ https://github.com/apache/hbase/blob/rel/1.2.5/hbase-thrift/src/main/resources/org/apache/hadoop/hbase/thrift2/hbase.thrift ）
2. 使用 "thrift --gen go hbase.thrift" 命令生成"hbase"软件包。
3. 将生成的“hbase”软件包放在GOPATH目录下。

使用示例

simple-example.go

package connetor 文件

- connetor.go
- hbaseOperation.go
- hbaseType.go
- kafkaConsumer.go
- kafkaProducer.go
- kafkaType.go
- monitorDataType.go
- transformData.go

主要接口说明

type Connetor

```
type Connetor struct {
   totalRuntineNum uint32 //总的hbase协程数量
   clientPool  *thrift_client_pool.ChannelClientPool //hbase client 池
   kafkaCfg *KafkaConfig //kafka的配置信息
   hbaseCfg *HbaseConfig //hbase的配置信息
   putRuntineNum uint32 //写入数据到hbase表的协程数量
   scanRuntineNum uint32 //查询hbase表的协程数量
```

```
    KafkaConsumer chan sarama.ConsumerMessage //从kafka中读取数据的通道
    KafkaProducer chan string //写入数据到kafka中的通道

    HbasePut chan []byte //写入数据到hbase的通道
    HbaseScan chan *ScanConfig //查询hbaser的通道，传输查询的条件
}
```

负责连接kafka和hbase。从kafka中读写数据和hbase写入数据都通过Connetor的方法进行。

func CreateConnetor

```
func CreateConnetor(kafkaCfg *KafkaConfig, hbaseCfg *HbaseConfig) *Connetor
```

创建一个Connetor。需要传入kafka和hbase的配置参数。

func StartConnetor

```
func (connetor *Connetor)StartConnetor()
```

启动Connetor。主要是启动了一个kafka Consumer、一个Producer的协程和多个hbase 读写数据的协程。数据通过connetor 的KafkaConsumer、KafkaProducer、HbasePut和HbaseScan数据通道传输给相应的协程。

func ConnetKafkatoHbase

```
func (connetor *Connetor)ConnetKafkatoHbase()
```

接收kafka中的数据，并将测点数据写入到hbase表中。

func ConnetHbasetoKafka

```
func (connetor *Connetor)ConnetHbasetoKafka(table string, pointName string,
    minTime time.Time, maxTime time.Time)
```

从hbase表中查询数据，并将查询到的数据格式化为JSON字符串，然后再写入kafka中。

func ConnetScanHbase

```
func (connetor *Connetor)ConnetScanHbase(table string, pointName string,
    minTime time.Time, maxTime time.Time) *ScanConfig
```

查询hbase表中的数据，可通过filters 传入查询时的过滤条件。

func Close

```
func (connetor *Connetor)Close()
```

用于Connetor使用结束后, 关闭Connetor。

type KafkaConfig

```
type KafkaConfig struct {
  Brokers string  //设置kafka的broker地址（如：192.168.1.32:9092,192.168.1.33:9092）
   GroupId string //设置kafka消费时的GroupId（非必须设置）
   ConsumerTopic string //设置kafka消费数据的Topic名称
   ProducerTopic string //设置kafka生产数据的Topic名称
   consumerMode cluster.ConsumerMode
   cmessages chan sarama.ConsumerMessage
   pmessages chan string
   cdone     chan bool
   pdone     chan bool
}
```

保存kafka的配置参数和数据传输通道。

func NewKafkaConfig

```
func NewKafkaConfig() *KafkaConfig
```

type HbaseConfig

```
type HbaseConfig struct {
   Hosts string  //设置hbase的regionserver地址
   Table string  //设置要读写的hbase的表名
   MaxIdle uint32 //最大的闲置hbase client连接数量
   MaxOpen uint32 //最大打开的hbase client连接数量和hbase读写总的协程数
   ConnectTimeout time.Duration //hbase socket 打开时的超时时间
   ReadTimeout time.Duration //hbase 读写时的超时时间
   PutFamilys  []string //hbase 写数据时指定的列族名称
}
```

保存hbase的配置参数。

func NewHbaseConfig

```
func NewHbaseConfig() *HbaseConfig
```

type ScanFilters

```
type ScanFilters struct {
   Table string //表名，当表名为空字符串时，默认查询hbaseConfig中的表名。
   PointName string //测点名称
   MinStamp int64 //时间段的开始时间（13位数，精确到毫秒的时间戳）
   MaxStamp int64 //时间段的结束时间（13位数，精确到毫秒的时间戳）
}
```

保存查询hbase表时的过滤条件。

func NewScanFilters

```
func NewScanFilters(table string, pointName string,
    minTime time.Time, maxTime time.Time) *ScanFilters
```

func ScanConfig

```
type ScanConfig struct {
   TScan *hbase.TScan //查询hbase表的核心参数
   Table string //表名
   ReadNum int32 //需要读取多少行数据（暂未使用）
   HaveReads int32 //已经记取了多少行数据(暂未使用)
   OneReadNum int32 //读取数据时，每一次读取多少行数据，默认为5000行
   Results chan []*hbase.TResult_ //从hbase表中读取的原始数据会放在此通道中
   MonitorDatas chan []string //将原始数据格式化成JSON字符串的数据会放在此通道中
    endScan bool //数据是否读取完成
    endMarshaler bool //数据是否转换完成
    endPrintMonitorDatas bool //数据是否全部取出
}
```

保存查询hbase表时的配置参数。

func NewScanConfig

```
func NewScanConfig(args ...interface{}) *ScanConfig
```

func MarshalResults

```
func (ScanCfg *ScanConfig)MarshalResults()
```

将ScanCfg.Results 通道中的原始测点数据格式化成JSON字符串存放在MonitorDatas通道中。

func PrintMonitorDatas

```
func (ScanCfg *ScanConfig)PrintMonitorDatas()
```

将MonitorDatas通道中的数据打印在屏幕中。

func SetScanFilters

```
func (ScanCfg *ScanConfig)SetScanFilters(filters *ScanFilters)
```

设置查询hbase表的过滤条件。

func ScanClose

```
func (ScanCfg *ScanConfig)ScanClose()
```

结束此次查询。