目录

概述

数据订阅

数据订阅功能可以帮助您获取KRDS for MySQL的实时增量数据，您可以根据自身业务需求自由消费增量数据，例如缓存更新策略、业务异步解耦、异构数据源数据实时同步及含复杂ETL的数据实时同步等多种业务场景。

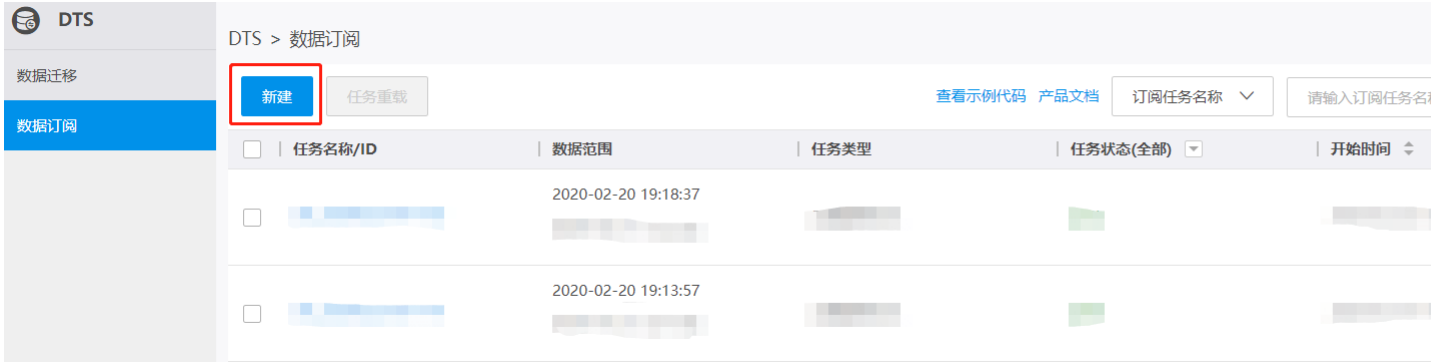
创建订阅任务

源库支持的实例类型

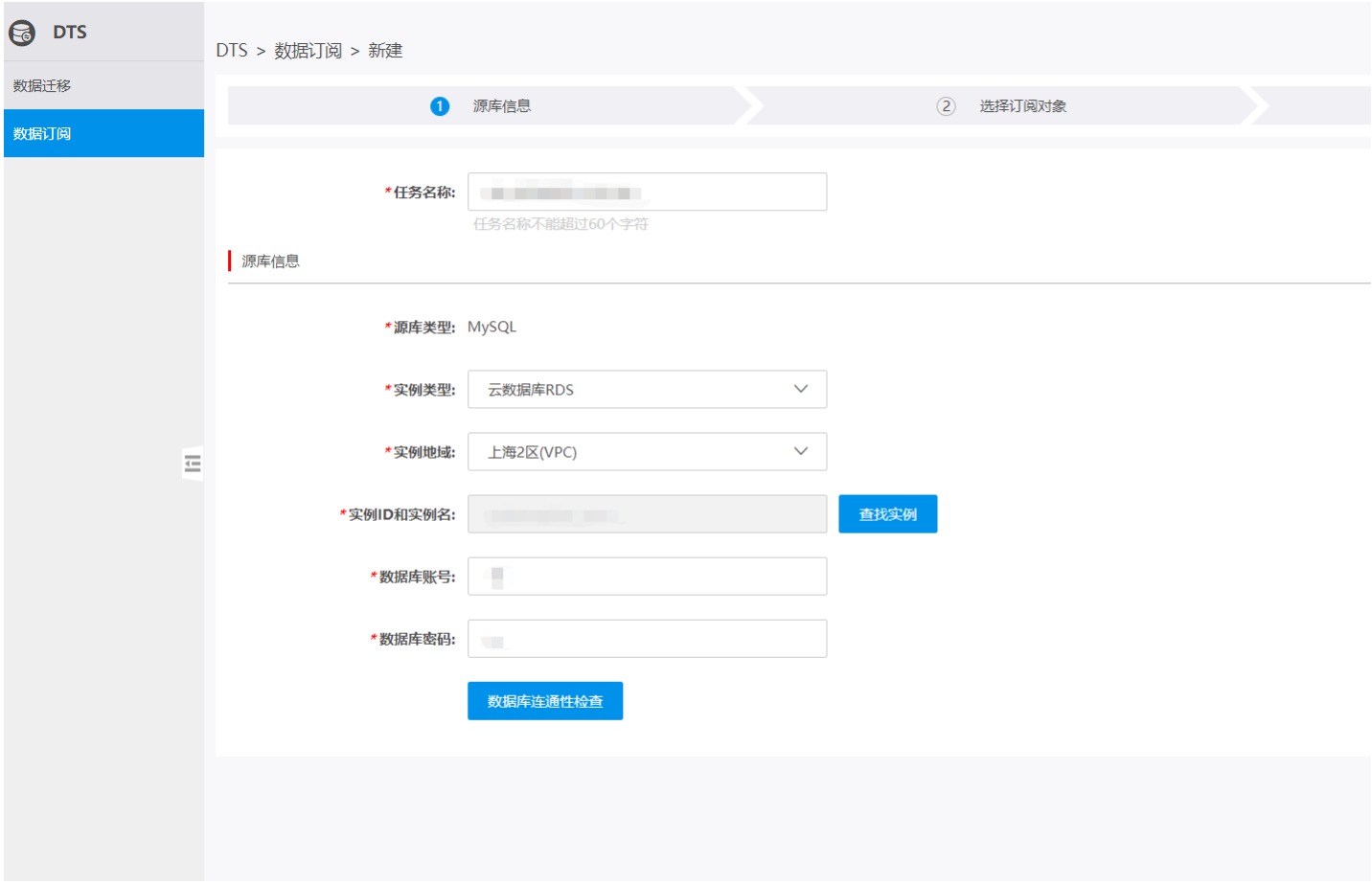
进行数据订阅操作的MySQL数据库支持以下实例类型： KRDS for MySQL

操作步骤

1. 登录数据传输控制台。
2. 在左侧导航栏，单击**数据订阅**。
3. 在数据订阅列表页面上方，点击**新建**



4. 配置订阅任务的源库信息



5. 选择订阅对象

6. 确认任务信息并进行预检查

预检查完成后，数据订阅任务即创建完成，之后可进行新建消费组和Kafka客户端消费订阅数据的操作

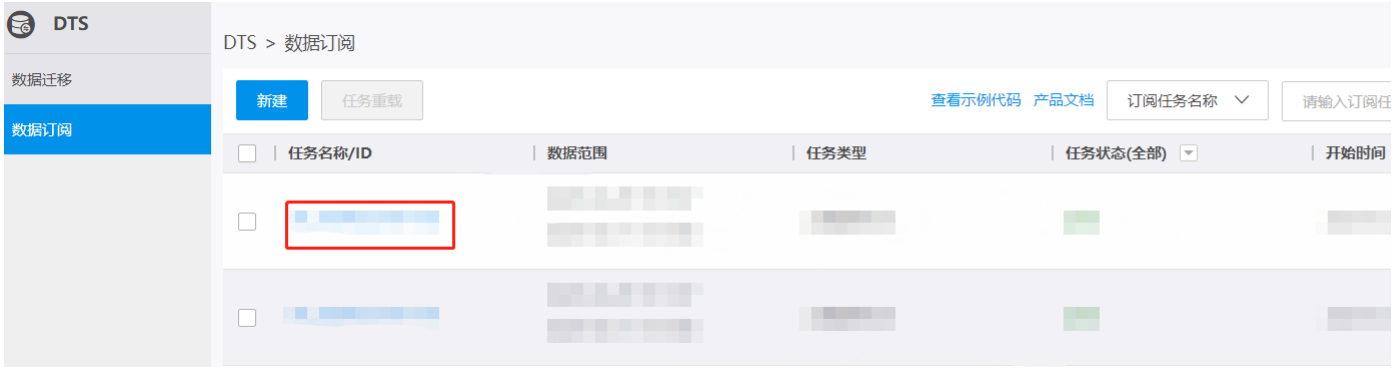
新建消费组

注意事项

- 一个数据订阅实例中可以创建多个消费组，通过创建多个消费组可以实现数据的重复消费。
- 同一个消费组对每条消息只进行一次消费，消费组内的多个消费者互为备份。
- 在同一个消费组中，同一时刻只能有一个消费者进行数据消费，其他消费者作为容灾节点。

操作步骤

1. 登录数据传输控制台。
2. 在左侧导航栏，单击数据订阅。
3. 在订阅任务列表中，单击订阅任务名称/ID



4. 单击左侧导航栏的数据消费。
5. 在数据消费页面，单击右上角的新建消费组。

6. 在弹出的创建消费组对话框，设置消费组信息。
7. 单击创建

Kafka客户端消费订阅数据

前提条件

- 已创建数据订阅任务。详情请见创建订阅任务
- 已创建消费组。详情请见新建消费组

注意事项

一个消费组对同一消息只能消费一次，如果希望实现对数据的多次消费，请新增消费组。

kafka客户端示例代码下载

请查看kafka客户端示例代码 说明：关于代码使用的详细介绍，请参见示例中的说明。

kafka客户端示例代码说明

代码中参数设置项详细说明如下 说明：可以通过DTS控制台获取对应参数的取值，详情请参见下方“获取数据订阅所需信息”。

参数名称	说明
User	消费组对应账号名称
Passwd	该账号对应的密码
BrokerURL	数据订阅的网络地址及端口号信息
GroupID	消费组ID

Topic 数据订阅的消息主题
StartTime 消息消费的起始时间。默认情况下，不需设置该值，即从订阅任务开始订阅全部消息；用户设置该值后，将强制订阅到该时间之后的消息，即为满足不同业务需求设置。

获取数据订阅所需信息

下面以KRDS for MySQL数据订阅为例，介绍如何获取数据订阅所需信息。 1、登录数据订阅控制台，在左侧导航栏选择数据订阅，进入数据订阅界面https://dts.console.ksyun.com/#/feed； 2、选择数据订阅所属地域，找到需要订阅的任务，单击该任务进入任务详情页； 3、在订阅任务详情页，可以获得订阅Topic和网络信息；

4、在任务详情页左侧选择数据消费，可以获得消费组ID及消费账号信息；

客户端示例代码

说明

go_demo是对应的go语言实现，后续会补充java、C++、python等语言的实现。如果您在使用过程中发现问题或者有好的实现方式，请联系客服，我们会在第一时间处理。 go_demo这个示例展示了如何使用数据订阅从DTS获取数据并且解析数据。 整个的流程包含了：用户进行参数配置、使用原生的kafka consumer从DTS获取增量数据、将获取到数据进行解析，从中获取对应的库名、表名、SQL语句、点位信息等。 使用的时候请替换： config.User = "" config.Passwd = "" // kafka broker url config.BrokerURL = "" // kafka consumer group name config.GroupID = "" // topic to consume, partition is 0 config.Topic = "" config.StartTime = "" 如果您希望订阅响应topic全部消息（即从订阅任务创建开始所有增量数据），config.StartTime则置为空。如果您希望从可选时间范围内某一时刻开始订阅数据，则将config.StartTime置为某一时刻值，时间格式为"2006-01-02 15:04:05"。 配置完成后，直接运行go run go_demo.go，即可订阅数据。您也可以根据自己的需要对订阅结果进行相应处理。

go_demo

```
package main

import (
    "encoding/json"
    "fmt"
    "github.com/Shopify/sarama"
    cluster "github.com/bsm/sarama-cluster"
    "os"
    "os/signal"
    "time"
)

type KafkaConfig struct {
    User      string
    Passwd    string
    BrokerURL string
    GroupID   string
    Topic     string
    StartTime string
}

func main() {
    var config KafkaConfig
```

```
    config.User = ""
    config.Passwd = ""
    config.BrokerURL = ""
    config.GroupID = ""
    config.Topic = ""
    config.StartTime = ""
    clusterConsumer(config)
}
func clusterConsumer(con KafkaConfig) {
    config := cluster.NewConfig()
    config.Version = sarama.V2_3_0_0
    config.Net.SASL.Enable = true
    config.Net.SASL.Mechanism = "PLAIN"
    config.Consumer.Return.Errors = true
    config.Group.Return.Notifications = true
    config.Net.MaxOpenRequests = 100
    config.Consumer.Offsets.AutoCommit.Interval = 1 * time.Second
    config.Consumer.Offsets.Initial = sarama.OffsetOldest
    config.Net.SASL.User = con.User
    config.Net.SASL.Password = con.Passwd
    consumer, err := cluster.NewConsumer([]string{con.BrokerURL}, con.GroupID, []string{con.Topic}, config)
    if err != nil {
        return
    }
    defer consumer.Close()
    var startTime int64
    if con.StartTime == "" {
        startTime = 0
    } else {
        formatTime, err := time.ParseInLocation("2006-01-02 15:04:05", con.StartTime, time.Local)
        if err != nil {
            fmt.Println(err)
            return
        }
        startTime = formatTime.Unix()
    }
    // trap SIGINT to trigger a shutdown
    signals := make(chan os.Signal, 1)
    signal.Notify(signals, os.Interrupt)
    // consume errors
    go func() {
        for err := range consumer.Errors() {
            panic(err)
        }
    }()
    // consume notifications
    go func() {
        for ntf := range consumer.Notifications() {
            fmt.Printf("Rebalanced: %+v\n", ntf)
        }
    }()
    // consume messages, watch signals
    var successes int
Loop:
    for {
        select {
        case msg, ok := <-consumer.Messages():
            if ok {
                if msg.Timestamp.Unix() < startTime {
                    continue
                }
                table, operation, sql, err := parseData(msg.Value)
                if err != nil {
                    fmt.Println("Create consumer failed,", err)
                    return
                }
                fmt.Fprintf(os.Stdout, "%s:%s/%d/%d\tdb:%s\ttable:%s\toperation:%s\tsql:%s\n",
```

```
                    "group_id", "topic",
                    msg.Partition, msg.Offset, msg.Key, table, operation, sql)
                consumer.MarkOffset(msg, "")
                successes++
            }
        case <-signals:
            break Loop
        }
    }
    fmt.Fprintf(os.Stdout, "%s consume %d messages \n", con.GroupID, successes)
}
func parseData(data []byte) (string, string, string, error) {
    type SubData struct {
        TableName string `json:"table"`
        Operation string `json:"operation"`
        Sql       string `json:"sql"`
    }
    d := SubData{}
    err := json.Unmarshal(data, &d)
    if err != nil {
        return "", "", "", fmt.Errorf("Data format is wrong.")
    }
    return d.TableName, d.Operation, d.Sql, nil
}
```