kafka

`ablecloud`

版本: kafka_2.12-0.11.0.0

---

1. zookeeper

配置文件

```
# zookeeper数据目录
dateDir=/data/zookeeper
# 端口
clientPort=2181
# 每个ip的连接数限制开关，0表示关闭
maxClientCnxns=0
# 集群配置
# server.A=B:C:D
# A：第几台服务器
# B：服务器IP
# C：这个服务器与集群中的Leader服务器交换信息的端口
# D：leader挂掉时用来进行选举leader的端口
server.1=10.0.0.1:2888:3888
server.2=10.0.0.2:2888:3888
server.3=10.0.0.2:2888:3888
```

创建serverId标识

```
echo "1" > /data/zookeeper/myid
```

启动

```
./bin/zookeeper-server-start.sh config/zookeeper.properties
```

---

2. kafka

配置文件

```
# zookeeper
zookeeper.connect=10.0.0.1:2888:3888,10.0.0.2:2888:3888,10.0.0.3:2888:3888
# broker id，如果没有设置，会生成一个唯一的id
broker.id=0
# log dirs
log.dirs=/data/kafka/logs
# 自动创建topic
auto.create.topics.enable=true
# 允许删除topic
delete.topic.enable=true
# Listener List
PLAINTEXT://10.0.0.1:9092
# log保存时长
log.retention.hours=168
# topic默认partition count
num.partitions=2
# topic默认副本数
default.replication.factor=2
```

启动

```
./bin/kafka-server-start.sh conf/server.properties
```

---

3. 升级

2.11-0.8.2.1 升级到 2.11-0.11.0.0

如果可以接受服务很短的down掉时间的话，可以将broker直接替换重启，然后换用新的客户端。

> 注意: 新的consumer默认将offset存储在kafka，而不是zookeeper

4. go client producer

https://github.com/confluentinc/confluent-kafka-go

> 版本: v0.11.0

4.1 go producer 实例1

```
// kafka broker
broker := "10.0.0.1:9092,10.0.0.2:9092"
// topic
topic := "test"
// 配置
configMap := &kafka.ConfigMap{
    "bootstrap.servers": broker,
}
// 结果事件channel
deliveryChan := make(chan kafka.Event)

// 创建producer
p, err := kafka.NewProducer(configMap)
if err != nil {
    fmt.Println("create producer failed: ", err)
    return
}

// 发送请求
value := "Hello World"
msg := &kafka.Message {
    // kafka.PartitionAny指的是任一个partition，也可以直接指定partition id
    TopicPartition: kafka.TopicPartition{Topic: &topic, Partition: kafka.PartitionAny},
    Value: []byte(value),
}
err := p.Produce(msg, deliveryChan)
if err != nil {
    fmt.Println("can't add the msg to queue")
```

```
        return //一般为添加到队列失败，根据需求选择重试还是忽略
    }

    // 处理结果
    e := <-deliveryChan
    m := e.(*kafka.Message)

    if m.TopicPartition.Error != nil {
        fmt.Println("Delivery failed: ", m.TopicPartition.Error)
    } else {
        fmt.Println("Delivery success")
    }

    // 关闭
    close(deliveryChan)
```

4.2 go producer 实例2(channel方式)

```
    // kafka broker
    broker := "10.0.0.1:9092,10.0.0.2:9092"
    // topic
    topic := "test"
    // 配置
    configMap := &kafka.ConfigMap{
        "bootstrap.servers": broker,
    }

    // 处理结果
    go func() {
        for e := range p.Events() {
            switch ev := e.(type) {
            case *kafka.Message:
                if ev.TopicPartition.Error != nil {
                    fmt.Println("Delivery failed: ", ev.TopicParition.Error
    )
                } else {
                    fmt.Println("Delivery success")
                }

            default:
                // 忽略
```

```
                fmt.Println("ignored event: ", ev)
            }
        }
    } ()

    // 发送请求
    value := "Hello World"
    msg := &kafka.Message {
        // kafka.PartitionAny指的是任一个partition，也可以直接指定partition id
        TopicPartition: kafka.TopicPartition{Topic: &topic, Partition: kafk
    a.PartitionAny},
        Value: []byte(value),
    }
    p.ProduceChannel() <- msg

    // 关闭
    p.Close()
```

4.3 go client producer 常用配置

最重要的配置

- bootstrap.servers
- acks
- client.id
- retries
- max.in.flight
- request.timeout.ms
- delivery.report.only.error

配置详细说明

- bootstrap.servers
  用来获取整个集群信息的种子结点，没必要将所有机器都配置上，可以选择2到3台机器。
  配置格式: 10.0.0.1:9092,10.0.0.2:9092

- acks
  - 0: producer发送record后，不需要等待server的确认，每一条记录的offset被设置为-1
  - 1: producer发送record后，leader写完log就返回确认，不会等待follower将record写入log
  - all或-1: producer发送record后，所有的副本都写完log后，才返回确认
    默认为1
- client.id
  客户端标识，用于记录日志和排查问题
  默认为rdkafka
- retries
  失败重试次数，当retries>0: 当发送消息失败后，会重试，此时如果配置max.in.flight如果值不是1,则重试会导致请求乱序
  默认为0
- max.in.flight
  单个connection中，最大允许的没有响应的请求数目
  默认为1000000
- request.timeout.ms
  producer发送消息超时时间
  默认为5000
- delivery.report.only.error
  消息发送成功，不report; 消息发送失败，report
  默认为false
- compression.codec
  - none: 不压缩
  - gzip: gzip
  - lz4: lz4
  - snappy: snappy
    默认为none
- batch.num.messages
  最大批量大小
  默认值为10000
- linger.ms

消息发送延迟时间，与batch.num.messages配合使用
默认为0

- queue.buffering.max.kbytes
  消息发送队列中允许的最大字节数
  默认为4000000
- queue.buffering.max.messages
  producer队列中允许的最大消息数量
  默认值为1000000
- receive.message.max.bytes
  最大接收的消息大小。
  默认值为100000000
- message.max.bytes
  kafka消息最大值
  默认值1000000
- retry.backoff.ms
  发送消息重试等待间隔
  默认为100
- message.timeout.ms
  本地消息超时时间，如果是0,则表示无限值
  默认为300000
- heartbeat.interval.ms
  group session keepalive heartbeat interval
  默认为1000
- session.timeout.ms
  Client group session and failure detection timeout.
  默认为30000
- group.id
  client group id
- log_level
  日志级别
  默认为6
- reconnection.backoff.jitter.ms
  连接断开后，基础的连接等待等待间隔

默认为500

- socket.max.fails
  失败多少次后，断开连接（连接断开后，会自动重连），0表示不断开
  默认为3,
- socket.nagle.disable
  tcp nodelay
  默认为false
- socket.keepalive.enable
  tcp enable
  默认为false
- socket.send.buffer.bytes
  send buffer大小
  默认为0,使用系统配置
- socket.receive.buffer.bytes
  receive buffer大小
  默认为0,使用系统配置
- socket.blocking.max.ms
  socket请求最长阻塞时间
  默认为1000
- socket.timeout.ms
  网络请求超时时间
  默认为60000
- topic.blacklist
  topic黑名单
  如果topic不存在，则忽略
- topic.metadata.refresh.sparse
  占用带宽少的元数据更新请求
  默认为true
- topic.metadata.refresh.fast.interval.ms
  当一个topic失去leader后，获取元数据的初始间隔
  默认为250
- metadata.request.timeout.ms
  元数据请求超时间

默认为60000

- topic.metadata.refresh.interval.ms
  元数据更新间隔
  默认为300000
- metadata.max.age.ms
  元数据cache时间
  默认为-1，意思是topic.metadata.refresh.interval.ms*3
- message.copy.max.bytes
  执行copy的最大的消息大小。（对于大于这个值的，执行zero-copy)
  默认值为65535

常用配置组合

- 确保消息只发送一次，并且不能重复，不能丢失; 性能要求不高; 超时时间为3s
  - bootstap.servers: 10.0.0.1:9092,10.0.0.2:9092,10.0.0.3:9092
  - acks: -1
  - retries: 1
  - max.in.flight: 1
  - request.timeout.ms: 3000
- 消息允许重复，不允许乱序，允许挂掉一个副本后少量数据丢失，超时时间为3s
  - bootstap.servers: 10.0.0.1:9092,10.0.0.2:9092,10.0.0.3:9092
  - acks: 1
  - retries: 1
  - max.in.flight: 1
  - request.timeout.ms: 3000
- 消息允许重复，允许乱序，允许挂掉一个副本后少量数据丢失，超时时间为3s
  - bootstap.servers: 10.0.0.1:9092,10.0.0.2:9092,10.0.0.3:9092
  - acks: 1
  - retries: 1
  - max.in.flight: 10000
  - request.timeout.ms: 3000
- 消息允许重复，允许乱序，发送失败不重试，允许挂掉一个副本后少量数据丢失，超时时

间为3s
    - bootstap.servers: 10.0.0.1:9092,10.0.0.2:9092,10.0.0.3:9092
    - acks: 1
    - retries: 0
    - max.in.flight: 10000
    - request.timeout.ms: 3000
- 只发送，不关心数据是否写成功
    - bootstap.servers: 10.0.0.1:9092,10.0.0.2:9092,10.0.0.3:9092
    - acks: 0
    - retries: 0
    - max.in.flight: 10000
    - enable.idempontence: false
    - request.timeout.ms: 3000

---

5. go client consumer

5.1 go consumer 实例1

```
broker := "10.0.0.1:9092,10.0.0.2:9092,10.0.0.3:9092"
topics := []string{"test1", "test2"}
groupId := "group1"

// 配置
configMap := &kafka.ConfigMap{
    "bootstrap.servers": broker,
    "group.id": groupId,
    "go.application.rebalance.enable": true,
    "auto.offset.reset": "latest",
}

// 创建consumer
c, err := kafka.NewConsumer(configMap)
if err != nil {
    fmt.Println("create consumer failed: ", err)
```

```
        return
    }

    // 订阅
    err = c.SubscribeTopics(topics, nil)
    if err != nil {
        fmt.Println("subscribe topics failed: ", err)
        return
    }

    // 处理消息
    for {
        ev := c.Pool(100)
        if ev == nil {
            continue
        }

        switch e := ev.(type) {
        case *kafka.Message:
            fmt.Println("consumer message: ", e)
        case kafka.PartitionEOF:
            fmt.Println("reached end: ", e)
        case kafka.Error:
            fmt.Println("error")
        default:
            fmt.Println("ignored")
        }
    }

    // 关闭
    c.Close()
```

5.2 go consumer 实例2

```
broker := "10.0.0.1:9092,10.0.0.2:9092,10.0.0.3:9092"
topics := []string{"test1", "test2"}
groupId := "group1"

// 配置
configMap := &kafka.ConfigMap{
    "bootstrap.servers": broker,
```

```
        "group.id": groupId,
        "go.events.channel.enable": true,
        "go.application.rebalance.enable": true,
        "auto.offset.reset": "latest",
    }

    // 创建consumer
    c, err := kafka.NewConsumer(configMap)
    if err != nil {
        fmt.Println("create consumer failed: ", err)
        return
    }

    // 订阅
    err = c.SubscribeTopics(topics, nil)
    if err != nil {
        fmt.Println("subscribe topics failed: ", err)
        return
    }

    // 处理消息
    for {
        ev := <-c.Events()

        switch e := ev.(type) {
        case kafka.AssignedPartitions:
            fmt.Println("assign partitions: ", e)
            c.Assign(e.Partitions)
        case kafka.RevokedPartitions:
            fmt.Println("revoke partitions: ", e)
            c.Unassign()
        case *kafka.Message:
            fmt.Println("message: ", e)
        case *kafka.PartitionEOF:
            fmt.Println("reached end: ", e)
        case kafka.Error:
            fmt.Println("error: ", e)
            // 自定义处理方式(退出 or 继续 or 重试)
        }
    }

    // 关闭
    c.Close()
```

5.3 go client consumer 常用配置

最重要的配置

- bootstrap.servers
- group.id
- auto.commit.enable
- enable.auto.offset.store
- auto.commit.interval.ms
- auto.offset.reset
- enable.partition.eof

常用配置说明

- bootstrap.servers
  用来获取整个集群信息的种子结点，没必要将所有机器都配置上，可以选择2到3台机器。
  配置格式: 10.0.0.1:9092,10.0.0.2:9092
- group.id
  consumer group id
- auto.commit.enable
  是否要自动提交
  默认为true
- enable.auto.offset.store
  自动存储最后一条消息的offset
  默认为true
- auto.commit.interval.ms
  offset定时提交间隔
  默认为60000
- auto.offset.reset
  当没有初始offset时，offset的起始值，earliest, latest
  默认为largest
- enable.partition.eof
  到partition结尾，报告事件
  默认为true

- offset.store.method
  存储offset的方法，支持file(本地)和broker(kafka)
  默认为broker
- consume.callback.max.messages
  最大允许的接收消息的数量
  默认0（无限制)
- client.id
  客户端标识，用于记录日志和排查问题
  默认为rdkafka
- check.crcs
  是否要校验crc
  默认为true
- enable.partition.eof
  是否要报告数据读完事件
  默认为true
- fetch.message.max.bytes
  一次获取的最大字节数
  默认为1048576
- fetch.min.bytes
  一次获取的最小字节数
  默认为1
- fetch.wait.max.ms
  获取响应，最长填充请求的时间
  默认为100
- fetch.error.backoff.ms
  fetch失败，重试间隔
  默认为500
- queued.max.messages.kbytes
  队列中最大的message大小和
  默认为1000000
- queued.min.messages
  队列中保持的最小的message数
  默认为100000

- enable.auto.offset.store
  自动存储最后一条消息的offset
  默认为true
- heartbeat.interval.ms
  group session keepalive heartbeat interval
  默认为1000
- session.timeout.ms
  Client group session and failure detection timeout.
  默认为30000
- log_level
  日志级别
  默认为6
- reconnection.backoff.jitter.ms
  连接断开后，基础的连接等待等待间隔
  默认为500
- socket.nagle.disable
  tcp nodelay
  默认为false
- socket.keepalive.enable
  tcp enable
  默认为false
- socket.send.buffer.bytes
  send buffer大小
  默认为0,使用系统配置
- socket.receive.buffer.bytes
  receive buffer大小
  默认为0,使用系统配置
- socket.blocking.max.ms
  socket请求最长阻塞时间
  默认为1000
- socket.timeout.ms
  网络请求超时时间
  默认为60000

- topic.blacklist
  topic黑名单
  如果topic不存在，则忽略
- topic.metadata.refresh.sparse
  占用带宽少的元数据更新请求
  默认为true
- topic.metadata.refresh.fast.interval.ms
  当一个topic失去leader后，获取元数据的初始间隔
  默认为250
- metadata.request.timeout.ms
  元数据请求超时间
  默认为60000
- topic.metadata.refresh.interval.ms
  元数据更新间隔
  默认为300000
- metadata.max.age.ms
  元数据cache时间
  默认为-1，意思是topic.metadata.refresh.interval.ms*3
- max.in.flight
  单个connection中，最大允许的没有响应的请求数目
  - metadata.max.age.ms
    元数据cache时间
    默认为-1，意思是topic.metadata.refresh.interval.ms*3
- message.copy.max.bytes
  执行copy的最大的消息大小。（对于大于这个值的，执行zero-copy）
  默认值为65535 默认为1000000
- receive.message.max.bytes
  最大接收的消息大小。
  默认值为100000000

常用配置组合

- 每1秒提交一次offset，数据消费完成自动提交，当初始offset不存在时，从头开始消费
  - bootstrap.servers=10.0.0.1:9092,10.0.0.2:9092
  - group.id=group1
  - auto.commit.enable=true
  - enable.auto.offset.store=true
  - auto.commit.interval.ms=1000
  - aut.offset.reset=ealiest
- 手动提交offset，当初始offset不存在时，从头开始消费
  - bootstrap.servers=10.0.0.1:9092,10.0.0.2:9092
  - group.id=group1
  - auto.commit.enable=false

手动提交offset的方式

- 方式1: 提交consumer的所有offset

```
1.  func (c *Consumer) Commit() ([]TopicPartition, error)
```

- 方式2: 使用message提交指定message所在的topic和offset

```
1.  func (c *Consumer) CommitMessage(m *Message) ([]TopicPartition, error)
```

- 方式3: 提交指定topic的offset

```
1.  func (c *Consumer) CommitOffsets(offsets []TopicPartition) ([]TopicPart
    ition, error)
```

---

6. kafka 常用命令

- 添加topic

```
1.  ./bin/kafka-topics.sh --zookeeper zk_host:port --create --topic my_topi
    c_name --partitions 2 --replication-factor 3
```

- 修改topic

```
./bin/kafka-topics.sh --zookeeper zk_host:port --alter --topic my_topic_name --partitions 4
```

- 查看topic

```
./bin/kafka-topics.sh --zookeeper localhost:2181 --topic foo --describe
```

- 列出topic

```
./bin/kafka-topics.sh --zookeeper localhost:2181 --list
```

- 列出所有consumer group

```
./bin/kafka-consumer-groups.sh --bootstrap-server broker1:9092 --list
```

- 查看consumer group 信息 (offset等)

```
./bin/kafka-consumer-groups.sh --bootstrap-server broker1:9092 --describe --group test-consumer-group
```