Kafka

作者：Hefery
原文链接：https://ld246.com/article/1644686800721
来源网站：链滴

为什么要学习Kafka?

系统学习Kafka已成为刚需

- 企业要求掌握Kafka（核心API+原理）
- 工作中要用到Kafka（倾目实战+配置经验）
- 面试里要问到Kafka（底层实现+面试点）

Kafka入门

介绍并安装kafka

kafka的简单介绍

A distributed streaming platform
Kafka是基于zookeeper的分布式消息系统
Kafka具有高吞吐率、高性能、实时及高可靠等特点

kafka的基本概念

- Topic：一个虚拟的概念，由1到多个Partitions组成
- Partition：实际消息存储单位
    - 特征
        - 每一个分区都是顺序，不可变的队列
        - 可以持续增加消息
        - 消息：尾部追加，头部消费
        - 每一个消息都会配分一个唯一标识，俗称“offset”
    - 规则
        - Producer的消息会放置在哪一个Partition上，根据系统默认规则
        - Partition的数量在运行期间可以增加，但是不能减少
        - 每一个Partition都有一个物理日志文件
    - 优点：集群情况下，日志可根据Partition拆分，避免单个日志文件过大；并行处理单元，提高吐量
- Segment：
- Producer：消息生产者
- Consumer：消息消费者

kafka的安装部署

1. JDK （略）

2. Zookeeper

解压：tar -zxvf apache-zookeeper-3.5.7-bin.tar.gz -C /opt/install/

配置：cd conf; cp zoo_sample.cfg zoo.cfg

```
# The number of milliseconds of each tick
tickTime=2000
# The number of ticks that the initial
# synchronization phase can take
initLimit=10
# The number of ticks that can pass between
# sending a request and getting an acknowledgement
syncLimit=5
# the directory where the snapshot is stored.
# do not use /tmp for storage, /tmp here is just
# example sakes.
dataDir=/tmp/zookeeper
# the port at which the clients will connect
clientPort=2181
# the maximum number of client connections.
# increase this if you need to handle more clients
#maxClientCnxns=60
#
# Be sure to read the maintenance section of the
# administrator guide before turning on autopurge.
#
# http://zookeeper.apache.org/doc/current/zookeeperAdmin.html#sc_maintenance
#
# The number of snapshots to retain in dataDir
#autopurge.snapRetainCount=3
# Purge task interval in hours
# Set to "0" to disable auto purge feature
#autopurge.purgeInterval=1
```

启动：cd bin; ./zkServer.sh start

```
ZooKeeper JMX enabled by default
Using config: /opt/install/apache-zookeeper-3.5.7-bin/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
```

验证：./zkCli.sh 或 ps -ef | grep zookeeper

3. Kafuka

解压：tar -zxvf kafka_2.11-2.4.0.tgz -C /opt/install/

配置：vi config/server.properties

```
listeners=PLAINTEXT://IP:9092
advertised.listeners=PLAINTEXT://IP:9092
zookeeper.connect=IP:2181
```

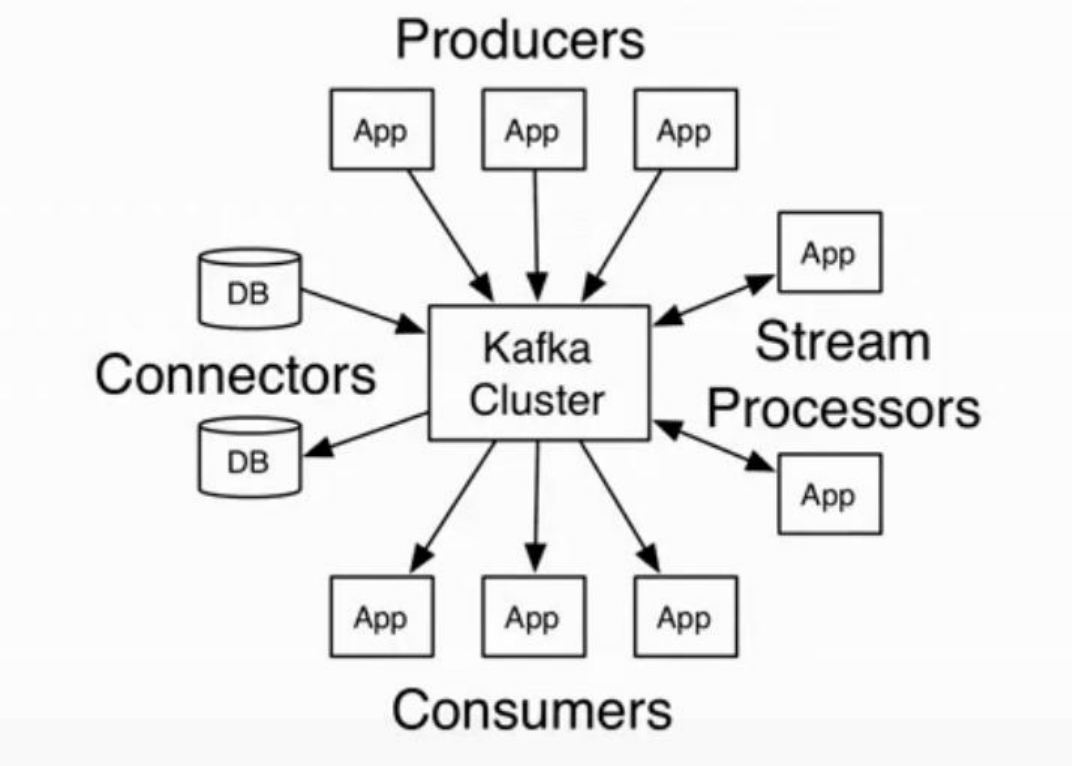
熟悉kafka常见控制台操作：

```
# 启动kafka
bin/kafka-server-start.sh config/server.properties &
```

```
# 停止kafka
bin/kafka-server-stop.sh

# 查看kafka是否启动成功
ps -ef | grep kafka

# 创建Topic
bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --
opic hefery-topic

# 查看已创建Topic信息
bin/kafka-topics.sh --list --zookeeper localhost:2181

# 发送消息
bin/kafka-console-producer.sh --broker-list IP:9092 --topic hefery-topic

# 接收消息
bin/kafka-console-consumer.sh --bootstrap-server IP:9092 --topic hefery-topic --from-beginn
ng
```

Kafka核心API解读及开发

Kafka客户端操作



Kafka客户端API类型

- AdminClient API：允许管理和检测Topic、broker以及其它Kafka对象
- Producer API：发布消息到1个或多个topic

- Consumer API：订阅一个或多个topic，并处理产生的消息
- Streams API：高效地将输入流转换到输出流
- Connector API：从一些源系统或应用程序中拉取数据到kafka

AdminClient API

API	作用
AdminClient	AdminClient客户端对象
NewTopic	创建Topic
CreateTopicsResult	创建Topic的返回结果
ListTopicsResult	查询Topic列表
ListTopicsOptions	查询Topic列表及选项
DescribeTopicsResult	查询Topics详情
DescribeConfigsResult	查询Topics配置项

```
public class AdminSample {

    public final static String TOPIC_NAME = "lalala-topic";

    public static void main(String[] args) throws Exception {
        // 设置AdminClient
        AdminClient adminClient = AdminSample.adminClient();
        //System.out.println("AdminClient：" + adminClient);
        // 创建Topic实例
        //createTopic();
        // 获取Topic列表
        //topicList();
        // 删除Topic实例
        //delTopics();
        // 描述Topic
        //describeTopics();
        // 查询Config
        //describeConfig();
        // 修改Config
        //alterConfig();
        // 增加partition数量
        incrPartitions(2);
    }

    /*
        增加partition数量
     */
    public static void incrPartitions(int partitions) throws Exception{
        AdminClient adminClient = adminClient();
        Map<String, NewPartitions> partitionsMap = new HashMap<>();
        NewPartitions newPartitions = NewPartitions.increaseTo(partitions);
        partitionsMap.put(TOPIC_NAME, newPartitions);

        CreatePartitionsResult createPartitionsResult = adminClient.createPartitions(partitionsM
```

```
p);
        createPartitionsResult.all().get();
    }

    /*
        修改Config信息
     */
    public static void alterConfig() throws Exception{
        AdminClient adminClient = adminClient();
        // Map<ConfigResource,Config> configMaps = new HashMap<>();

        // 组织两个参数
        // ConfigResource configResource = new ConfigResource(ConfigResource.Type.TOPIC,
OPIC_NAME);
        // Config config = new Config(Arrays.asList(new ConfigEntry("preallocate","true")));
        // configMaps.put(configResource,config);
        // AlterConfigsResult alterConfigsResult = adminClient.alterConfigs(configMaps);

        /*
            从 2.3以上的版本新修改的API
         */
        Map<ConfigResource,Collection<AlterConfigOp>> configMaps = new HashMap<>();
        // 组织两个参数
        ConfigResource configResource = new ConfigResource(ConfigResource.Type.TOPIC, TOP
C_NAME);
        AlterConfigOp alterConfigOp =
                new AlterConfigOp(new ConfigEntry("preallocate","false"),AlterConfigOp.OpType.SE
);
        configMaps.put(configResource,Arrays.asList(alterConfigOp));

        AlterConfigsResult alterConfigsResult = adminClient.incrementalAlterConfigs(configMap
);
        alterConfigsResult.all().get();
    }

    /*
        查看配置信息
        ConfigResource(type=TOPIC, name='jiangzh-topic') ,
        Config(
            entries=[
             ConfigEntry(
               name=compression.type,
               value=producer,
               source=DEFAULT_CONFIG,
               isSensitive=false,
               isReadOnly=false,
               synonyms=[]),
             ConfigEntry(
               name=leader.replication.throttled.replicas,
               value=,
               source=DEFAULT_CONFIG,
               isSensitive=false,
               isReadOnly=false,
               synonyms=[]), ConfigEntry(name=message.downconversion.enable, value=true, sou
```

```
ce=DEFAULT_CONFIG, isSensitive=false, isReadOnly=false, synonyms=[]), ConfigEntry(name
min.insync.replicas, value=1, source=DEFAULT_CONFIG, isSensitive=false, isReadOnly=false, s
nonyms=[]), ConfigEntry(name=segment.jitter.ms, value=0, source=DEFAULT_CONFIG, isSensi
ive=false, isReadOnly=false, synonyms=[]), ConfigEntry(name=cleanup.policy, value=delete, s
urce=DEFAULT_CONFIG, isSensitive=false, isReadOnly=false, synonyms=[]), ConfigEntry(nam
=flush.ms, value=9223372036854775807, source=DEFAULT_CONFIG, isSensitive=false, isRea
Only=false, synonyms=[]), ConfigEntry(name=follower.replication.throttled.replicas, value=, s
urce=DEFAULT_CONFIG, isSensitive=false, isReadOnly=false, synonyms=[]), ConfigEntry(nam
=segment.bytes, value=1073741824, source=STATIC_BROKER_CONFIG, isSensitive=false, isRe
dOnly=false, synonyms=[]), ConfigEntry(name=retention.ms, value=604800000, source=DEF
ULT_CONFIG, isSensitive=false, isReadOnly=false, synonyms=[]), ConfigEntry(name=flush.mes
ages, value=9223372036854775807, source=DEFAULT_CONFIG, isSensitive=false, isReadOnly
false, synonyms=[]), ConfigEntry(name=message.format.version, value=2.4-IV1, source=DEF
ULT_CONFIG, isSensitive=false, isReadOnly=false, synonyms=[]), ConfigEntry(name=file.delete
delay.ms, value=60000, source=DEFAULT_CONFIG, isSensitive=false, isReadOnly=false, synon
ms=[]), ConfigEntry(name=max.compaction.lag.ms, value=9223372036854775807, source=D
FAULT_CONFIG, isSensitive=false, isReadOnly=false, synonyms=[]), ConfigEntry(name=max.m
ssage.bytes, value=1000012, source=DEFAULT_CONFIG, isSensitive=false, isReadOnly=false, s
nonyms=[]), ConfigEntry(name=min.compaction.lag.ms, value=0, source=DEFAULT_CONFIG, i
Sensitive=false, isReadOnly=false, synonyms=[]), ConfigEntry(name=message.timestamp.type,
value=CreateTime, source=DEFAULT_CONFIG, isSensitive=false, isReadOnly=false, synonyms
[]),
            ConfigEntry(name=preallocate, value=false, source=DEFAULT_CONFIG, isSensitive=fa
se, isReadOnly=false, synonyms=[]), ConfigEntry(name=min.cleanable.dirty.ratio, value=0.5, s
urce=DEFAULT_CONFIG, isSensitive=false, isReadOnly=false, synonyms=[]), ConfigEntry(nam
=index.interval.bytes, value=4096, source=DEFAULT_CONFIG, isSensitive=false, isReadOnly=f
lse, synonyms=[]), ConfigEntry(name=unclean.leader.election.enable, value=false, source=DE
AULT_CONFIG, isSensitive=false, isReadOnly=false, synonyms=[]), ConfigEntry(name=retenti
n.bytes, value=-1, source=DEFAULT_CONFIG, isSensitive=false, isReadOnly=false, synonyms=[
), ConfigEntry(name=delete.retention.ms, value=86400000, source=DEFAULT_CONFIG, isSensi
ive=false, isReadOnly=false, synonyms=[]), ConfigEntry(name=segment.ms, value=604800000
 source=DEFAULT_CONFIG, isSensitive=false, isReadOnly=false, synonyms=[]), ConfigEntry(n
me=message.timestamp.difference.max.ms, value=9223372036854775807, source=DEFAULT
CONFIG, isSensitive=false, isReadOnly=false, synonyms=[]), ConfigEntry(name=segment.index
bytes, value=10485760, source=DEFAULT_CONFIG, isSensitive=false, isReadOnly=false, synon
ms=[])])
     */
    public static void describeConfig() throws Exception{
        AdminClient adminClient = adminClient();
        // TODO 这里做一个预留，集群时会讲到
//        ConfigResource configResource = new ConfigResource(ConfigResource.Type.BROKER,
OPIC_NAME);

        ConfigResource configResource = new ConfigResource(ConfigResource.Type.TOPIC, TOP
C_NAME);
        DescribeConfigsResult describeConfigsResult = adminClient.describeConfigs(Arrays.asLis
(configResource));
        Map<ConfigResource, Config> configResourceConfigMap = describeConfigsResult.all().
et();
        configResourceConfigMap.entrySet().stream().forEach((entry)->{
            System.out.println("configResource : "+entry.getKey()+" , Config : "+entry.getValue());
        });
    }
```

```
    /*
        描述Topic
        name ： jiangzh-topic ,
        desc: (name=jiangzh-topic,
            internal=false,
            partitions=
                (partition=0,
                 leader=192.168.220.128:9092
                 (id: 0 rack: null),
                 replicas=192.168.220.128:9092
                 (id: 0 rack: null),
                 isr=192.168.220.128:9092
                 (id: 0 rack: null)),
                 authorizedOperations=[])
     */
    public static void describeTopics() throws Exception {
        AdminClient adminClient = adminClient();
        DescribeTopicsResult describeTopicsResult = adminClient.describeTopics(Arrays.asList(T
PIC_NAME));
        Map<String, TopicDescription> stringTopicDescriptionMap = describeTopicsResult.all().
et();
        Set<Map.Entry<String, TopicDescription>> entries = stringTopicDescriptionMap.entrySe
();
        entries.stream().forEach((entry)->{
            System.out.println("name ： "+entry.getKey()+" , desc: "+ entry.getValue());
        });
    }

    /**
     * 删除Topic
     * @throws Exception
     */
    public static void delTopics() throws Exception {
        AdminClient adminClient = adminClient();
        DeleteTopicsResult deleteTopicsResult = adminClient.deleteTopics(Arrays.asList(TOPIC_
AME));
        deleteTopicsResult.all().get();
    }

    /**
     * 获取Topic列表
     * @throws ExecutionException
     * @throws InterruptedException
     */
    public static void topicList() throws ExecutionException, InterruptedException {
        AdminClient adminClient = adminClient();
        // 是否查看internal选项
        ListTopicsOptions options = new ListTopicsOptions();
        options.listInternal(true);
        //ListTopicsResult listTopicsResult = adminClient.listTopics();
        ListTopicsResult listTopicsResult = adminClient.listTopics(options);
        Set<String> names = listTopicsResult.names().get();
        Collection<TopicListing> topicListings = listTopicsResult.listings().get();
        KafkaFuture<Map<String, TopicListing>> mapKafkaFuture = listTopicsResult.namesToLi
```

```
tings();
        // 打印names
        names.stream().forEach(System.out::println);
        // 打印topicListings
        topicListings.stream().forEach((topicList)->{
            System.out.println(topicList);
        });
    }

    /**
     * 创建Topic实例
     */
    public static void createTopic() {
        AdminClient adminClient = adminClient();
        // bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partiti
ns 1 --topic hefery-topic
        Short rs = 1;  // 副本因子
        NewTopic newTopic = new NewTopic(TOPIC_NAME, 1, rs);
        CreateTopicsResult topics = adminClient.createTopics(Arrays.asList(newTopic));
        System.out.println("CreateTopicsResult:" + topics);
    }

    /**
     * 设置AdminClient
     * @return
     */
    public static AdminClient adminClient() {
        Properties properties = new Properties();
        properties.setProperty(AdminClientConfig.BOOTSTRAP_SERVERS_CONFIG, "192.168.153.
28:9092");
        AdminClient adminClient = AdminClient.create(properties);
        return adminClient;
    }

}
```

Producer API

熟练掌握Kafka之Producer API

了解Producer各项重点配置

熟练Producer负载均衡等高级特性

Kafka配置解析

Producer发送模式

- 同步发送

```
public class ProducerSample {

    private final static String TOPIC_NAME="hefery-topic";
```

```
    public static void main(String[] args) {
        // 同步：Producer异步阻塞发送
        producerSyncSend();
    }

    /*
        同步：Producer异步阻塞发送
     */
    public static void producerSyncSend() throws ExecutionException, InterruptedException {
        Properties properties = new Properties();
        properties.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,"192.168.153.128:9092");
        properties.put(ProducerConfig.ACKS_CONFIG,"all");
        properties.put(ProducerConfig.RETRIES_CONFIG,"0");
        properties.put(ProducerConfig.BATCH_SIZE_CONFIG,"16384");
        properties.put(ProducerConfig.LINGER_MS_CONFIG,"1");
        properties.put(ProducerConfig.BUFFER_MEMORY_CONFIG,"33554432");

        properties.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,"org.apache.kafka.comm
n.serialization.StringSerializer");
        properties.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,"org.apache.kafka.co
mon.serialization.StringSerializer");

        // Producer的主对象
        Producer<String,String> producer = new KafkaProducer<>(properties);

        // 消息对象 - ProducerRecoder
        for(int i=0;i<10;i++){
            String key = "key-"+i;
            ProducerRecord<String,String> record =
                    new ProducerRecord<>(TOPIC_NAME,key,"value-"+i);

            Future<RecordMetadata> send = producer.send(record);
            RecordMetadata recordMetadata = send.get();
            System.out.println(key + "partition : "+recordMetadata.partition()+" , offset : "+recor
Metadata.offset());
        }

        // 所有的通道打开都需要关闭
        producer.close();
    }

}
```

- 异步发送

```
public class ProducerSample {

    private final static String TOPIC_NAME="hefery-topic";

    public static void main(String[] args) {
        // Producer异步发送演示
        producerSend();
    }

    /**
```

```
     * Producer异步发送演示
     */
    public static void producerSend() {
        Properties properties = new Properties();
        properties.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,"192.168.153.128:9092");
        properties.put(ProducerConfig.ACKS_CONFIG,"all");
        properties.put(ProducerConfig.RETRIES_CONFIG,"0");
        properties.put(ProducerConfig.BATCH_SIZE_CONFIG,"16384");
        properties.put(ProducerConfig.LINGER_MS_CONFIG,"1");
        properties.put(ProducerConfig.BUFFER_MEMORY_CONFIG,"33554432");

        properties.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,"org.apache.kafka.comm
n.serialization.StringSerializer");
        properties.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,"org.apache.kafka.co
mon.serialization.StringSerializer");
        // Producer的主对象
        Producer<String,String> producer = new KafkaProducer<>(properties);

        // 消息对象 - ProducerRecoder
        for(int i=0;i<10;i++){
            ProducerRecord<String,String> record = new ProducerRecord<>(TOPIC_NAME,"key-
+i,"value-"+i);
            producer.send(record);
        }

        // 所有的通道打开都需要关闭
        producer.close();
    }

}
```

- 异步回调发送

```
public class ProducerSample {

    private final static String TOPIC_NAME="hefery-topic";

    public static void main(String[] args) {
        /// Producer异步发送带回调函数
        producerSendWithCallback();
        // Producer异步发送带回调函数和Partition负载均衡
        producerSendWithCallbackAndPartition();
    }

    /*
        Producer异步发送带回调函数和Partition负载均衡
     */
    public static void producerSendWithCallbackAndPartition(){
        Properties properties = new Properties();
        properties.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,"192.168.153.128:9092");
        properties.put(ProducerConfig.ACKS_CONFIG,"all");
        properties.put(ProducerConfig.RETRIES_CONFIG,"0");
        properties.put(ProducerConfig.BATCH_SIZE_CONFIG,"16384");
        properties.put(ProducerConfig.LINGER_MS_CONFIG,"1");
        properties.put(ProducerConfig.BUFFER_MEMORY_CONFIG,"33554432");
```

```
        properties.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,"org.apache.kafka.comm
n.serialization.StringSerializer");
        properties.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,"org.apache.kafka.co
mon.serialization.StringSerializer");
        properties.put(ProducerConfig.PARTITIONER_CLASS_CONFIG,"com.imooc.jiangzh.kafka.p
oducer.SamplePartition");

        // Producer的主对象
        Producer<String,String> producer = new KafkaProducer<>(properties);

        // 消息对象 - ProducerRecoder
        for(int i=0;i<10;i++){
            ProducerRecord<String,String> record =
                    new ProducerRecord<>(TOPIC_NAME,"key-"+i,"value-"+i);

            producer.send(record, new Callback() {
                @Override
                public void onCompletion(RecordMetadata recordMetadata, Exception e) {
                    System.out.println(
                            "partition : "+recordMetadata.partition()+" , offset : "+recordMetadata.offset
));
                }
            });
        }

        // 所有的通道打开都需要关闭
        producer.close();
    }

    /*
        Producer异步发送带回调函数
     */
    public static void producerSendWithCallback(){
        Properties properties = new Properties();
        properties.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,"192.168.153.128:9092");
        properties.put(ProducerConfig.ACKS_CONFIG,"all");
        properties.put(ProducerConfig.RETRIES_CONFIG,"0");
        properties.put(ProducerConfig.BATCH_SIZE_CONFIG,"16384");
        properties.put(ProducerConfig.LINGER_MS_CONFIG,"1");
        properties.put(ProducerConfig.BUFFER_MEMORY_CONFIG,"33554432");

        properties.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,"org.apache.kafka.comm
n.serialization.StringSerializer");
        properties.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,"org.apache.kafka.co
mon.serialization.StringSerializer");

        // Producer的主对象
        Producer<String,String> producer = new KafkaProducer<>(properties);

        // 消息对象 - ProducerRecoder
        for(int i=0;i<10;i++){
            ProducerRecord<String,String> record =
                    new ProducerRecord<>(TOPIC_NAME,"key-"+i,"value-"+i);
```

```
            producer.send(record, new Callback() {
                @Override
                public void onCompletion(RecordMetadata recordMetadata, Exception e) {
                    System.out.println(
                            "partition : "+recordMetadata.partition()+" , offset : "+recordMetadata.offset
));
                }
            });
        }

        // 所有的通道打开都需要关闭
        producer.close();
    }

}
```

Producer源码

构建—KafkaProducer

- MetricConfig
- 加载负载均衡器
- 初始化Serializer
- 初始化RecordAccumulator——类似于计数器
- 启动newSender——守护线程

```
public class KafkaProducer<K, V> implements Producer<K, V> {
   KafkaProducer(Map<String, Object> configs, Serializer<K> keySerializer, Serializer<V> valu
Serializer, ProducerMetadata metadata, KafkaClient kafkaClient, ProducerInterceptors intercep
ors, Time time) {
        ProducerConfig config = new ProducerConfig(ProducerConfig.addSerializerToConfig(con
igs, keySerializer, valueSerializer));

        try {
            Map<String, Object> userProvidedConfigs = config.originals();
            this.producerConfig = config;
            this.time = time;
            String transactionalId = userProvidedConfigs.containsKey("transactional.id") ? (String)
serProvidedConfigs.get("transactional.id") : null;
            this.clientId = buildClientId(config.getString("client.id"), transactionalId);
            LogContext logContext;
            if (transactionalId == null) {
                logContext = new LogContext(String.format("[Producer clientId=%s] ", this.clientId))

            } else {
                logContext = new LogContext(String.format("[Producer clientId=%s, transactionall
=%s] ", this.clientId, transactionalId));
            }

            this.log = logContext.logger(KafkaProducer.class);
```

```
            this.log.trace("Starting the Kafka producer");
            Map<String, String> metricTags = Collections.singletonMap("client-id", this.clientId);
            // 1. MetricConfig
            MetricConfig metricConfig = (new MetricConfig()).samples(config.getInt("metrics.num.
amples")).timeWindow(config.getLong("metrics.sample.window.ms"), TimeUnit.MILLISECOND
).recordLevel(RecordingLevel.forName(config.getString("metrics.recording.level"))).tags(metri
Tags);
            List<MetricsReporter> reporters = config.getConfiguredInstances("metric.reporters",
etricsReporter.class, Collections.singletonMap("client.id", this.clientId));
            reporters.add(new JmxReporter("kafka.producer"));
            this.metrics = new Metrics(metricConfig, reporters, time);
            // 2. 加载负载均衡器
            this.partitioner = (Partitioner)config.getConfiguredInstance("partitioner.class", Partitio
er.class);
            long retryBackoffMs = config.getLong("retry.backoff.ms");
            if (keySerializer == null) {
                this.keySerializer = (Serializer)config.getConfiguredInstance("key.serializer", Serialize
.class);
                this.keySerializer.configure(config.originals(), true);
            } else {
                config.ignore("key.serializer");
                this.keySerializer = keySerializer;
            }

            // 3. 初始化Serializer
            if (valueSerializer == null) {
                this.valueSerializer = (Serializer)config.getConfiguredInstance("value.serializer", Seria
izer.class);
                this.valueSerializer.configure(config.originals(), false);
            } else {
                config.ignore("value.serializer");
                this.valueSerializer = valueSerializer;
            }

            userProvidedConfigs.put("client.id", this.clientId);
            ProducerConfig configWithClientId = new ProducerConfig(userProvidedConfigs, false);
            List<ProducerInterceptor<K, V>> interceptorList = configWithClientId.getConfiguredI
stances("interceptor.classes", ProducerInterceptor.class);
            if (interceptors != null) {
                this.interceptors = interceptors;
            } else {
                this.interceptors = new ProducerInterceptors(interceptorList);
            }

            ClusterResourceListeners clusterResourceListeners = this.configureClusterResourceList
ners(keySerializer, valueSerializer, interceptorList, reporters);
            this.maxRequestSize = config.getInt("max.request.size");
            this.totalMemorySize = config.getLong("buffer.memory");
            this.compressionType = CompressionType.forName(config.getString("compression.ty
e"));
            this.maxBlockTimeMs = config.getLong("max.block.ms");
            this.transactionManager = configureTransactionState(config, logContext, this.log);
            int deliveryTimeoutMs = configureDeliveryTimeout(config, this.log);
            this.apiVersions = new ApiVersions();
```

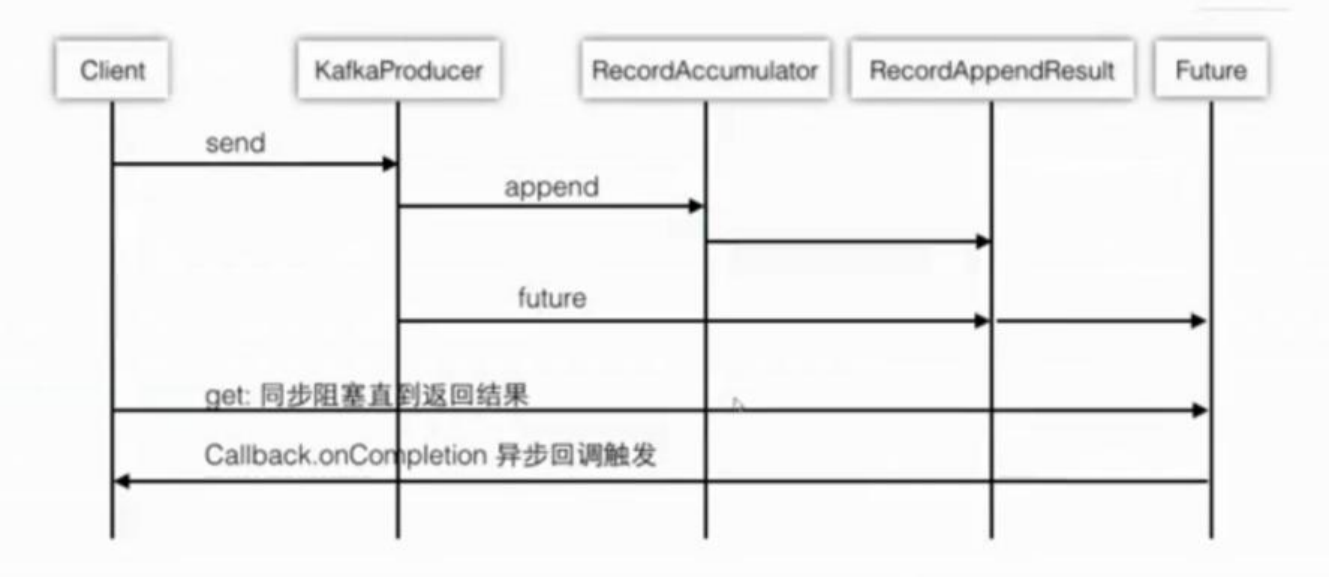
```
            // 4. 初始化RecordAccumulator——类似于计数器
            this.accumulator = new RecordAccumulator(logContext, config.getInt("batch.size"), thi
.compressionType, lingerMs(config), retryBackoffMs, deliveryTimeoutMs, this.metrics, "produ
er-metrics", time, this.apiVersions, this.transactionManager, new BufferPool(this.totalMemoryS
ze, config.getInt("batch.size"), this.metrics, time, "producer-metrics"));
            List<InetSocketAddress> addresses = ClientUtils.parseAndValidateAddresses(config.g
tList("bootstrap.servers"), config.getString("client.dns.lookup"));
            if (metadata != null) {
                this.metadata = metadata;
            } else {
                this.metadata = new ProducerMetadata(retryBackoffMs, config.getLong("metadata
max.age.ms"), logContext, clusterResourceListeners, Time.SYSTEM);
                this.metadata.bootstrap(addresses, time.milliseconds());
            }

            this.errors = this.metrics.sensor("errors");
            // 5. 启动newSender——守护线程
            this.sender = this.newSender(logContext, kafkaClient, this.metadata);
            String ioThreadName = "kafka-producer-network-thread | " + this.clientId;
            this.ioThread = new KafkaThread(ioThreadName, this.sender, true);
            this.ioThread.start();
            config.logUnused();
            AppInfoParser.registerAppInfo("kafka.producer", this.clientId, this.metrics, time.millise
onds());
            this.log.debug("Kafka producer started");
        } catch (Throwable var23) {
            this.close(Duration.ofMillis(0L), true);
            throw new KafkaException("Failed to construct kafka producer", var23);
        }
    }

}
```

Producer是线程安全的

Producer并不是接到一条发一条，是批量发送

发送—producer.send(record)

- 计算分区：消息具体进入哪一个partition
- 计算批次：accumulator.append
- 主要内容：创建批次；向批次中追加消息




```
public class KafkaProducer<K, V> implements Producer<K, V> {
    private Future<RecordMetadata> doSend(ProducerRecord<K, V> record, Callback callback)
{
        TopicPartition tp = null;

        try {
            this.throwIfProducerClosed();

            KafkaProducer.ClusterAndWaitTime clusterAndWaitTime;
            try {
                clusterAndWaitTime = this.waitOnMetadata(record.topic(), record.partition(), this.m
xBlockTimeMs);
            } catch (KafkaException var20) {
                if (this.metadata.isClosed()) {
                    throw new KafkaException("Producer closed while send in progress", var20);
                }

                throw var20;
            }

            long remainingWaitMs = Math.max(0L, this.maxBlockTimeMs - clusterAndWaitTime.w
itedOnMetadataMs);
            Cluster cluster = clusterAndWaitTime.cluster;

            byte[] serializedKey;
            try {
                serializedKey = this.keySerializer.serialize(record.topic(), record.headers(), record.key
));
            } catch (ClassCastException var19) {
                throw new SerializationException("Can't convert key of class " + record.key().getClas
().getName() + " to class " + this.producerConfig.getClass("key.serializer").getName() + " speci
ied in key.serializer", var19);
            }

            byte[] serializedValue;
            try {
                serializedValue = this.valueSerializer.serialize(record.topic(), record.headers(), record
```

```
value());
        } catch (ClassCastException var18) {
            throw new SerializationException("Can't convert value of class " + record.value().ge
Class().getName() + " to class " + this.producerConfig.getClass("value.serializer").getName() +
" specified in value.serializer", var18);
        }

        // 计算分区：消息具体进入哪一个partition
        int partition = this.partition(record, serializedKey, serializedValue, cluster);
        tp = new TopicPartition(record.topic(), partition);
        this.setReadOnly(record.headers());
        Header[] headers = record.headers().toArray();
        int serializedSize = AbstractRecords.estimateSizeInBytesUpperBound(this.apiVersions
maxUsableProduceMagic(), this.compressionType, serializedKey, serializedValue, headers);
        this.ensureValidRecordSize(serializedSize);
        long timestamp = record.timestamp() == null ? this.time.milliseconds() : record.timest
mp();
        if (this.log.isTraceEnabled()) {
            this.log.trace("Attempting to append record {} with callback {} to topic {} partition {}"
 new Object[]{record, callback, record.topic(), partition});
        }

        Callback interceptCallback = new KafkaProducer.InterceptorCallback(callback, this.inte
ceptors, tp);
        if (this.transactionManager != null && this.transactionManager.isTransactional()) {
            this.transactionManager.failIfNotReadyForSend();
        }

        // 计算批次：accumulator.append
        RecordAppendResult result = this.accumulator.append(tp, timestamp, serializedKey, se
ializedValue, headers, interceptCallback, remainingWaitMs, true);
        // 主要内容：创建批次
        if (result.abortForNewBatch) {
            int prevPartition = partition;
            this.partitioner.onNewBatch(record.topic(), cluster, partition);
            partition = this.partition(record, serializedKey, serializedValue, cluster);
            tp = new TopicPartition(record.topic(), partition);
            if (this.log.isTraceEnabled()) {
                this.log.trace("Retrying append due to new batch creation for topic {} partition {}.
The old partition was {}", new Object[]{record.topic(), partition, prevPartition});
            }

            interceptCallback = new KafkaProducer.InterceptorCallback(callback, this.intercepto
s, tp);
            result = this.accumulator.append(tp, timestamp, serializedKey, serializedValue, head
rs, interceptCallback, remainingWaitMs, false);
        }

        if (this.transactionManager != null && this.transactionManager.isTransactional()) {
            this.transactionManager.maybeAddPartitionToTransaction(tp);
        }

        if (result.batchIsFull || result.newBatchCreated) {
            this.log.trace("Waking up the sender since topic {} partition {} is either full or getting
```

```
a new batch", record.topic(), partition);
                // 主要内容：向批次中追加消息
                this.sender.wakeup();
            }

            return result.future;
        } catch (ApiException var21) {
            this.log.debug("Exception occurred during message send:", var21);
            if (callback != null) {
                callback.onCompletion((RecordMetadata)null, var21);
            }

            this.errors.record();
            this.interceptors.onSendError(record, tp, var21);
            return new KafkaProducer.FutureFailure(var21);
        } catch (InterruptedException var22) {
            this.errors.record();
            this.interceptors.onSendError(record, tp, var22);
            throw new InterruptException(var22);
        } catch (BufferExhaustedException var23) {
            this.errors.record();
            this.metrics.sensor("buffer-exhausted-records").record();
            this.interceptors.onSendError(record, tp, var23);
            throw var23;
        } catch (KafkaException var24) {
            this.errors.record();
            this.interceptors.onSendError(record, tp, var24);
            throw var24;
        } catch (Exception var25) {
            this.interceptors.onSendError(record, tp, var25);
            throw var25;
        }
    }

}
```

Producer发送原理解析

- 直接发送
- 负载均衡
- 异步发送

Producer自定义Partition负载均衡

```
public class SamplePartition implements Partitioner {
    @Override
    public int partition(String topic, Object key, byte[] keyBytes, Object value, byte[] valueBytes,
Cluster cluster) {
        /*
            key-1 key-2 key-3
         */
        String keyStr = key + "";
```

```
        String keyInt = keyStr.substring(4);
        System.out.println("keyStr : "+keyStr + "keyInt : "+keyInt);

        int i = Integer.parseInt(keyInt);

        return i%2;
    }

    @Override
    public void close() {

    }

    @Override
    public void configure(Map<String, ?> configs) {

    }
}

public class ProducerSample {

    private final static String TOPIC_NAME="jiangzh-topic";
    public static void main(String[] args) throws ExecutionException, InterruptedException {
        // Producer异步发送带回调函数和Partition负载均衡
        producerSendWithCallbackAndPartition();
    }


    /*
        Producer异步发送带回调函数和Partition负载均衡
     */
    public static void producerSendWithCallbackAndPartition(){
        Properties properties = new Properties();
        properties.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,"192.168.220.128:9092");
        properties.put(ProducerConfig.ACKS_CONFIG,"all");
        properties.put(ProducerConfig.RETRIES_CONFIG,"0");
        properties.put(ProducerConfig.BATCH_SIZE_CONFIG,"16384");
        properties.put(ProducerConfig.LINGER_MS_CONFIG,"1");
        properties.put(ProducerConfig.BUFFER_MEMORY_CONFIG,"33554432");

        properties.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,"org.apache.kafka.comm
n.serialization.StringSerializer");
        properties.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,"org.apache.kafka.co
mon.serialization.StringSerializer");
        properties.put(ProducerConfig.PARTITIONER_CLASS_CONFIG,"com.imooc.jiangzh.kafka.p
oducer.SamplePartition");

        // Producer的主对象
        Producer<String,String> producer = new KafkaProducer<>(properties);

        // 消息对象 - ProducerRecoder
        for(int i=0;i<10;i++){
            ProducerRecord<String,String> record =
                    new ProducerRecord<>(TOPIC_NAME,"key-"+i,"value-"+i);
```

```
            producer.send(record, new Callback() {
                @Override
                public void onCompletion(RecordMetadata recordMetadata, Exception e) {
                    System.out.println(
                            "partition : "+recordMetadata.partition()+" , offset : "+recordMetadata.offset
));
                }
            });
        }

        // 所有的通道打开都需要关闭
        producer.close();
    }
}
```

Producer发送消息传递保障

- 最多一次：收到0到1次
- 至少一次：收到1到多次
- 正好一次：有且仅有一次

Producer发送-项目应用

```
请求参数实例：
{
    templateId:"001",
    result:[
        {"questionId":"1","question":"今天几号","answer":"A"},
        {"questionId":"2","question":"你喜爱的颜色","answer":"B"}
    ]
}

public class KafkaTest {
    // kafka producer将数据推送至Kafka Topic
    public void templateReported(JSONObject reportInfo) {
        log.info("templateReported : [{}]", reportInfo);
        String topicName = "hefery-topic";
        // 发送Kafka数据
        String templateId = reportInfo.getString("templateId");
        JSONArray reportData = reportInfo.getJSONArray("result");

        // 如果templateid相同，后续在统计分析时，可以考虑将相同的id的内容放入同一个partition
便于分析使用
        ProducerRecord<String,Object> record = new ProducerRecord<>(topicName,templateId
reportData);

        /*
            1、Kafka Producer线程安全，建议多线程复用，如果每个线程都创建，出现大量的上下文
换或争抢的情况，影响Kafka效率
            2、Kafka Producer的key是一个很重要的内容：
                2.1 我们可以根据Key完成Partition的负载均衡
```

```
            2.2 合理的Key设计，可以让Flink、Spark Streaming之类的实时分析工具做更快速处理
          3、ack - all， kafka层面上就已经有了只有一次的消息投递保障，如果想不丢数据，最好自处理异常
         */
        try{
            producer.send(record);
        }catch (Exception e){
            // 将数据加入重发队列， redis，es，...
        }
    }
}
```

Consumer API

Kafka底层实现

Kafka设计原理

Kafka集群配置及监控