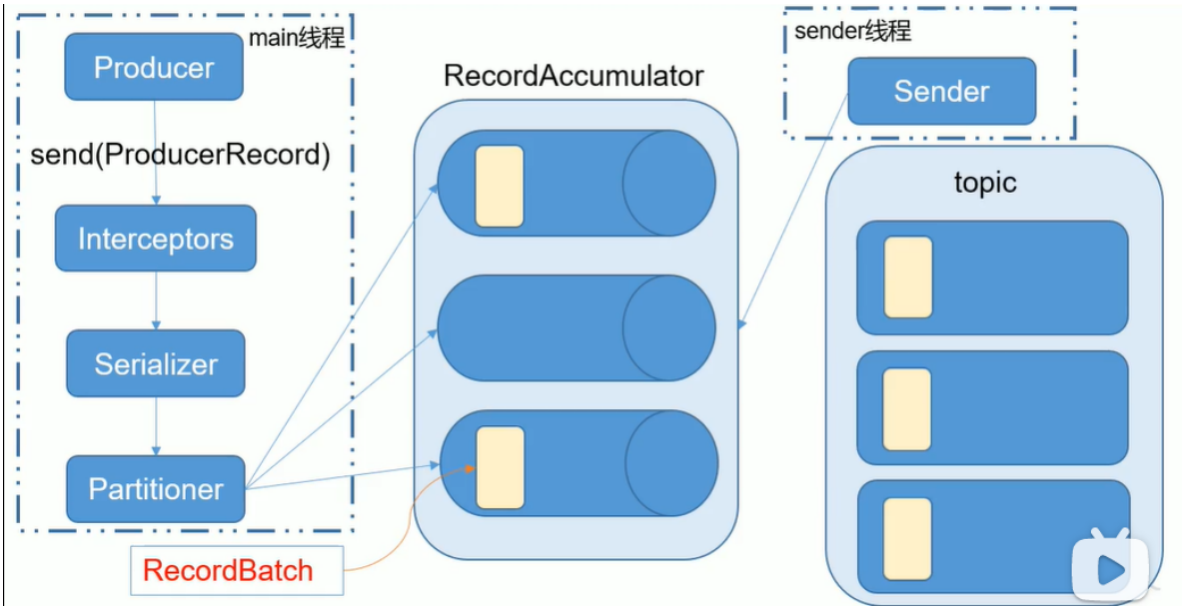
KafkaAPI 1 API生产者

消息发送流程

> Kafka 的 Producer 发送消息采用的是 异步发送 的方式。在消息发送的过程中，涉及到了两个线程——main()线程和Sender 线程，以及一个线程共享变量——RecordAccumulator，main线程将消息发送给RecordAccumulator，Sender 线程不断从 RecordAccumulator中拉取消息发送到Kafka broker。

kafka发送消息流程



相关参数：

batch.size： 只有数据积累到batch.size 之后，sender 才会发送数据

Linger.ms：如果数据迟迟未达到batch.size，sendoer等到linger.time之后就发送数据。

异步发送API

- **导入依赖**

```
<dependencies>
 <!--https://mvnrepository.com/artifact/org.apache.kafka/afka-clients -->
    <dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-clients</artifactId>
        <version>0.11.0.0</version>
    </dependency>
 </denpendencies>
```

- 编写代码

  需要用到的类：

  KafkaProducer：需要创建一个生产者对象，用来发送数据

ProducerConfig：获取所需的一系列配置参数

ProducerRedcord：每条数据都要封装成一个ProducerRecord对象

```
package com.atguigu.producer;

public class MyProducer{

    public static void main(String[] args){

        Properties props = new Properties();
        // Kafka 服务端的主机名和端口号
        //props.put("bootstrap.servers", "hadoop103:9092");
        props.put(ProducerConfig.BOOTSRAP_SERVERS_CONFIG,"hadoop102:9092");
 //也可以使用ProducerConfig类
        // 等待所有副本节点的应答,ack应答级别
        props.put("acks", "all");
        // 消息发送最大尝试次数
        props.put("retries", 0);
        // 一批消息处理大小
        props.put("batch.size", 16384);
        // 请求延时，等待时间
        props.put("linger.ms", 1);
        // 发送缓存区内存大小
        props.put("buffer.memory", 33554432);
        // key 序列化
        props.put("key.serializer",
"org.apache.kafka.common.serialization.StringSerializer");
        // value 序列
        props.put("value.serializer",
"org.apache.kafka.common.serialization.StringSerializer");

        //创建生产者对象

        KafkaProducer<String,String> producer = new KafkaProducer<String,
String>();

        //发送数据
        for(int i = 0; i < 10; i++){
            producer.send(new ProducerRescord<>("first", "autuigu--")+i);
        }
        //关闭
        producer.close();

    }
}
```

- 开启消费者

```
bin/kafka-console-consumer.sh --zookeeper hadoop102:2181 --topic first
```

- 运行发送

```
autuigu--0
autuigu--2
autuigu--4
autuigu--6
autuigu--8
autuigu--1
autuigu--3
autuigu--5
autuigu--7
autuigu--9
```

回调代码

- 开启消费者

```
bin/kafka-console-consumer.sh --zookeeper hadoop102:2181 --topic first
```

- 写代码

```
package com.atguigu.producer;

public class CallBackProducer{

    public static void main(String[] args){

        Properties props = new Properties();
        // Kafka 服务端的主机名和端口号
        //props.put("bootstrap.servers", "hadoop103:9092");
        props.put(ProducerConfig.BOOTSRAP_SERVERS_CONFIG,"hadoop102:9092");
 //也可以使用ProducerConfig类
        // key 序列化
        props.put("key.serializer",
"org.apache.kafka.common.serialization.StringSerializer");
        // value 序列
        props.put("value.serializer",
"org.apache.kafka.common.serialization.StringSerializer");

        //创建生产者对象

        KafkaProducer<String,String> producer = new KafkaProducer<String,
String>();

        //发送数据
        for(int i = 0; i < 10; i++){
            producer.send(new ProducerRescord<>("topic" , "autuigu--")+i),
new Callback(){

                @Override
                public void onCompletion(RecordMetadata metadata, Exception
exception){
                    if(exception == null){
                        System.out.print(metadata.partition()+"---
"+metadata.offset());
```

```
                    }
                }
            });
        }
        //关闭
        producer.close();

    }
}
```

- 结果

```
2-0
2-1
2-2
2-3
1-0
1-1
1-2
0-0
0-1
0-2
```

分区策略

重点：修改 new ProducerRedcord<>()函数：

- **new ProducerRedcord<>(topic, partition, key, value)**

```
指定分区发送
```

- **new ProducerRedcord<>(topic, key, value)**

```
将key的hashcode % n来分区传递
```

- **new ProducerRedcord<>(topic, value)**

```
采用轮询的方式，一个分区一个信息
```

- 代码

```
package com.atguigu.producer;

public class CallBackProducer{

    public static void main(String[] args){

        Properties props = new Properties();
        // Kafka 服务端的主机名和端口号
        //props.put("bootstrap.servers", "hadoop103:9092");
```

```
        props.put(ProducerConfig.BOOTSRAP_SERVERS_CONFIG,"hadoop102:9092");
 //也可以使用ProducerConfig类
        // key 序列化
        props.put("key.serializer",
"org.apache.kafka.common.serialization.StringSerializer");
        // value 序列
        props.put("value.serializer",
"org.apache.kafka.common.serialization.StringSerializer");

        //创建生产者对象

        KafkaProducer<String,String> producer = new KafkaProducer<String,
String>();

        //发送数据,这里不一样
        for(int i = 0; i < 10; i++){
            producer.send(new ProducerRescord<>("aaa", 0, "autuigu--")+i),
new Callback(){

                @Override
                public void onCompletion(RecordMetadata metadata, Exception
exception){
                    if(exception == null){
                        System.out.print(metadata.partition()+"---
"+metadata.offset());
                    }
                }
            });
        }
        //关闭
        producer.close();

    }
}
```

- 结果

```
autuigu-- -- 0
autuigu-- -- 1
autuigu-- -- 2
autuigu-- -- 3
autuigu-- -- 4
autuigu-- -- 5
autuigu-- -- 6
autuigu-- -- 7
autuigu-- -- 8
autuigu-- -- 9
```

自定义分区

```
package com.atguigu.partitioner;

public class MyPartitioner implements Partitioner{
```

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

    }


    @Override
    public int partition(String topic, Object key, byte[] keyBytes, Object
value, byte[] valueBytes, Cluster cluster) {
        // 控制分区
        return 0;
    }

    @Override
    public void close() {
    }
}
```

- 调用代码

```
package com.atguigu.producer;

public class  PartitionerProducer{

    public static void main(String[] args){

        Properties props = new Properties();
        // Kafka 服务端的主机名和端口号
        //props.put("bootstrap.servers", "hadoop103:9092");
        props.put(ProducerConfig.BOOTSRAP_SERVERS_CONFIG,"hadoop102:9092");  //
也可以使用ProducerConfig类
        // key 序列化
        props.put("key.serializer",
"org.apache.kafka.common.serialization.StringSerializer");
        // value 序列
        props.put("value.serializer",
"org.apache.kafka.common.serialization.StringSerializer");
        //添加分区器
        props.put("PARTITIONER_CLASS_CONFIG",
"com.atguigu.partitioner.MyPartitioner");


        //创建生产者对象
        KafkaProducer<String,String> producer = new KafkaProducer<String,
String>();

        //发送数据,这里不一样
        for(int i = 0; i < 10; i++){
            producer.send(new ProducerRescord<>("aaa", 0, "autuigu--")+i), new
Callback(){

                @Override
                public void onCompletion(RecordMetadata metadata, Exception
exception){
                    if(exception == null){
                        System.out.print(metadata.partition()+"---
"+metadata.offset());
```

```
                    }
                }
            });
        }
        //关闭
        producer.close();
    }
}
```

- 答案

```
1 - 16
2 - 17
3 - 18
...
```

同步发送 (少用)

同步发送的意思就是，一条信息发送之后，会阻塞当前线程，直接返回ack

由于send方法返回的是一个Future对象，根据Future对象的特点，我们也可以实现同步发送的效果，只需要调用Future对象的get方法即可。

```
package com.atguigu.producer;

public class MyProducer{

    public static void main(String[] args){

        Properties props = new Properties();
        // Kafka 服务端的主机名和端口号
        //props.put("bootstrap.servers", "hadoop103:9092");
        props.put(ProducerConfig.BOOTSRAP_SERVERS_CONFIG,"hadoop102:9092");  //
也可以使用ProducerConfig类
        // 等待所有副本节点的应答,ack应答级别
        props.put("acks", "all");
        // 消息发送最大尝试次数
        props.put("retries", 0);
        // 一批消息处理大小
        props.put("batch.size", 16384);
        // 请求延时，等待时间
        props.put("linger.ms", 1);
        // 发送缓存区内存大小
        props.put("buffer.memory", 33554432);
        // key 序列化
        props.put("key.serializer",
"org.apache.kafka.common.serialization.StringSerializer");
        // value 序列
        props.put("value.serializer",
"org.apache.kafka.common.serialization.StringSerializer");

        //创建生产者对象
```

```
        KafkaProducer<String,String> producer = new KafkaProducer<String,
String>();

        //发送数据, 同步发送数据
        for(int i = 0; i < 10; i++){
            Future<RecordMetadata> first = new producer.send(new
ProducerRescord<>("first", "autuigu--")+i));

            try{
                RecordMetadata recordMetadata = first.get(); //阻塞其他线程
            }catch( ExcutionException e){
                e.printStackTrace();
            }
        }
        //关闭
        producer.close();

    }
}
```