howtoprogram 9:06 am on August 27, 2016

Create Multi-threaded Apache Kafka Consumer

In previous posts, I introduced about how to get started with Apache Kafka by installing and using Java client API 0.9 as well. In this post, I'd like to share how to create multi-threaded Apache Kafka consumer.

You can take a look at previous related posts by access below links.

Getting started with Apache Kafka 0.9

Apache Kafka 0.9 Java Client API Example

1. Why do we need multi-thread consumer model?

Suppose we implement a notification module which allow users to subscribe for notifications from other users, other applications..Our module reads messages which will be written by other users, applications to a Kafka clusters. In this case, we can get all notifications of the others written to a Kafka topic and our module will create a consumer to subscribe to that topic.

Everything seems to be fine at the beginning. However, what will happen if the number of notifications produced by other applications, users…is increased fast and exceed the rate that can be processed by our module?

Well, everything is still…not bad. All the messages/notifications that haven't been processed by our module, are still in the Kafka topic. However, things get more danger when the number of messages is too much. Some of them will be lost when the retention policy is met (Note that Kafka retention policy can be time-based, partition size-based, key-based). And more important, when our notification module falls very far behind the income notifications/messages, it is not a true "notification" module anymore.

It's time to think about the multi-thread consumer model.

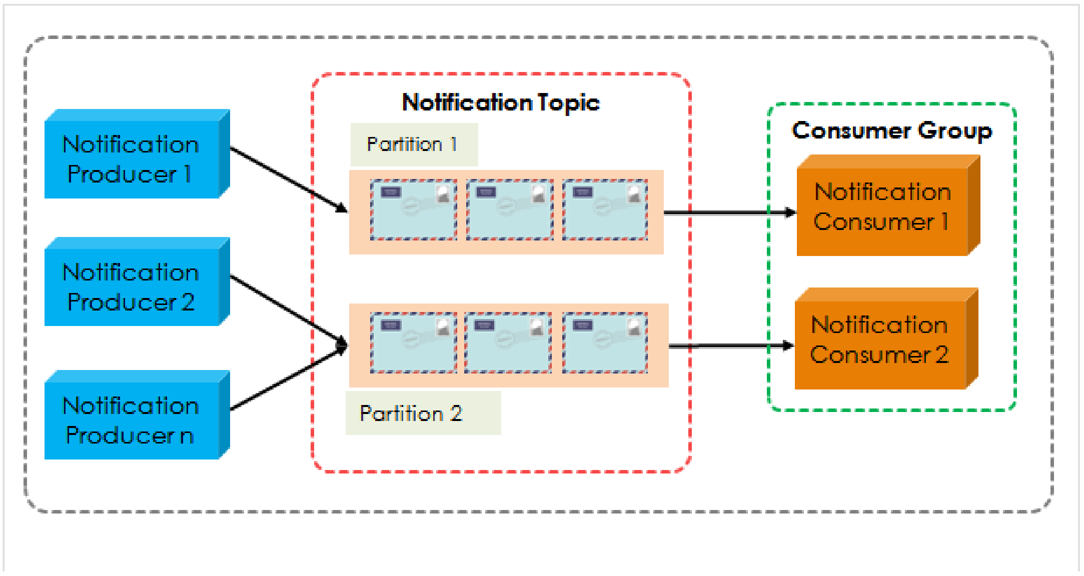
2. Multi-threaded Apache Kafka consumer model

There are 2 possible models I'd like to mention in this post.

- Multiple consumers with their own threads (Model #1)
- Single consumer, multiple worker processing threads (Model #2)

Both of them have their own pros and cons

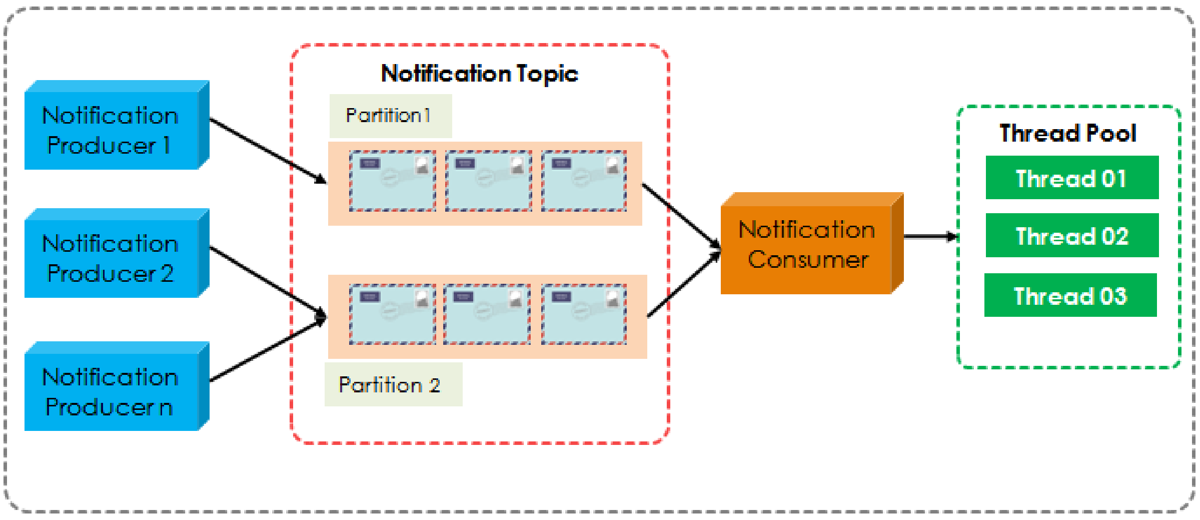
2.1 Model #1. Multiple consumers with their own threads



Multiple consumers with their own threads

Pros	Cons
Easy to implement	The total of consumers is limited the total partitions of the topic. The redundant consumers may not be used.
Implementing in-order processing on per-partition is easier.	More TCP connections to the brokers

2.2 Model #2. Single consumer, multiple worker processing threads



Single consumer, multiple worker processing threads

Pros	Cons
Be flexible in scale out the number of processing thread	It's not easy to implement in-order processing on per partition. Let's say there are 2 messages on the same partitions being processed by 2 different threads. To guarantee the order, those 2 threads must be coordinated somehow.

3. Implementation

Below is the implementation detail of both 2 models.

3.1. Prerequisite

- Apache Kafka 0.9/0.10 broker installed on local machine or remote. You can refer to this link for setting up.
- JDK 7/8 installed on your development PC.
- Eclipse 4 (I am using Eclipse Mars 4.5)
- Maven 3

3.2 Source Code Structure

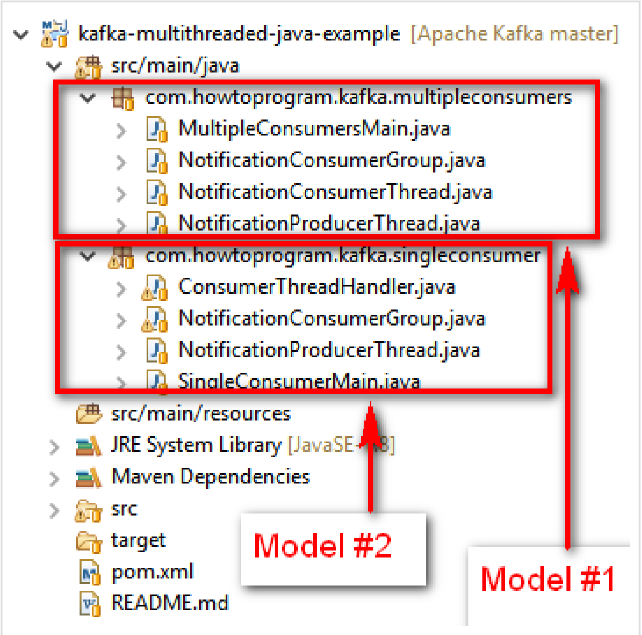
The example source code is added in the Github. You can use git to pull the repository to your PC or simple download the **zip** version of the example and extract to your PC.

After having the source code, you can import the source code into Eclipse and run the test.

To import:

- Menu ***File*** –> ***Import*** –> ***Maven*** –> ***Existing Maven Projects***
- Browse to your source code location
- Click ***Finish*** button to finish the importing

Here is the project structure in my Eclipse:



```
kafka-multithreaded-java-example [Apache Kafka master]
  src/main/java
    com.howtoprogram.kafka.multipleconsumers
      MultipleConsumersMain.java
      NotificationConsumerGroup.java
      NotificationConsumerThread.java
      NotificationProducerThread.java
    com.howtoprogram.kafka.singleconsumer
      ConsumerThreadHandler.java
      NotificationConsumerGroup.java
      NotificationProducerThread.java
      SingleConsumerMain.java
  src/main/resources
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
  pom.xml
  README.md
```

Create Multi-threaded Apache Kafka Consumer – Source Code

The source code includes the implementation for both above models. The package **com.howtoprogram.kafka.multipleconsumers** contains all source code for the **Model #1: Multiple consumers with their own threads** and the package **com.howtoprogram.kafka.singleconsumer** contain all the source code for the **Model #2: Single consumer, multiple worker processing threads**

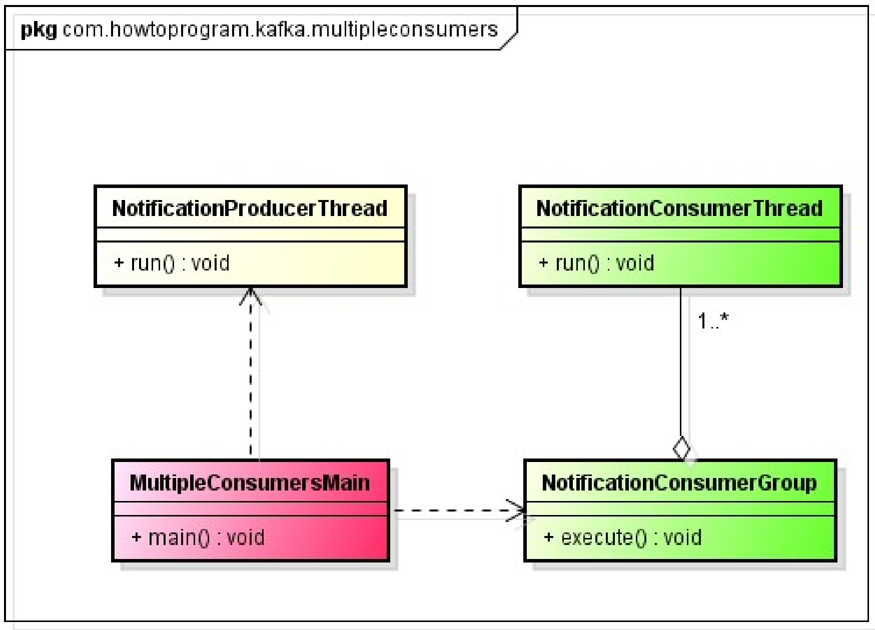
3.3 Maven pom.xml

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0
                                   http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.howtoprogram</groupId>
    <artifactId>kafka-multithreaded-java-example</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>Kafka-MultiThread-Java-Example</name>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka-clients</artifactId>
            <version>0.9.0.1</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
</project>
```

We use the **kafka-clients-0.9.0.1** library for this example. The Java compiler **1.8**

3.4. Multiple consumers with their own threads (Model #1)

3.4.1 Class Diagram

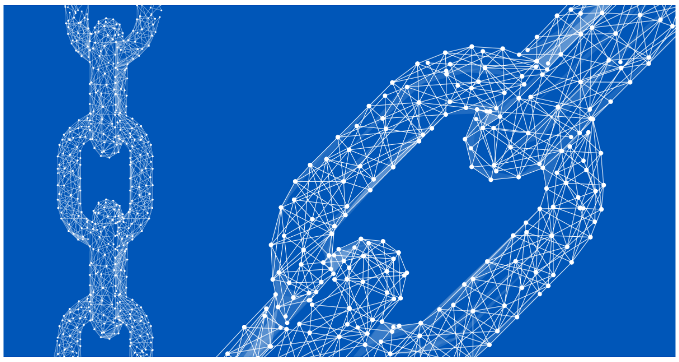

pkg com.howtoprogram.kafka.multipleconsumers
NotificationProducerThread
+ run() : void
NotificationConsumerThread
+ run() : void
1..*
MultipleConsumersMain
+ main() : void
NotificationConsumerGroup
+ execute() : void

Multiple consumers with their own threads – Class Diagram

The source code for this part includes 4 classes:

NotificationProducerThread.java is a producer thread, produces message to the Kafka brokers

NotificationConsumerThread.java is a consumer thread, consumes message from Kafka brokers



NotificationConsumerGroup.java create a group of NotificationConsumerThread(s)

MultipleConsumersMain.java contains the **main** method, run the program to produce and consume messages.

3.4.2 NotificationProducerThread.java

```java
package com.howtoprogram.kafka.multipleconsumers;

import java.util.Properties;

import org.apache.kafka.clients.producer.Callback;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.producer.RecordMetadata;

public class NotificationProducerThread implements Runnable {

  private final KafkaProducer<String, String> producer;
  private final String topic;
  public NotificationProducerThread(String brokers, String topic) {
    Properties prop = createProducerConfig(brokers);
    this.producer = new KafkaProducer<String, String>(prop);
    this.topic = topic;
  }

  private static Properties createProducerConfig(String brokers) {
    Properties props = new Properties();
    props.put("bootstrap.servers", brokers);
    props.put("acks", "all");
    props.put("retries", 0);
    props.put("batch.size", 16384);
    props.put("linger.ms", 1);
    props.put("buffer.memory", 33554432);
    props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
    props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
    return props;
  }

  @Override
  public void run() {
    System.out.println("Produces 3 messages");
    for (int i = 0; i < 5; i++) {
      String msg = "Message " + i;
      producer.send(new ProducerRecord<String, String>(topic, msg), new Callback() {
        public void onCompletion(RecordMetadata metadata, Exception e) {
          if (e != null) {
            e.printStackTrace();
          }
          System.out.println("Sent:" + msg + ", Partition: " + metadata.partition() + ", Offset: "
              + metadata.offset());
        }
      });

    }
    // closes producer
    producer.close();

  }
}
```

When this producer thread runs, it will produces 5 messages to the broker.

3.4.3 NotificationConsumerThread.java

```
package com.howtoprogram.kafka.multipleconsumers;

import java.util.Arrays;
import java.util.Properties;

import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;

public class NotificationConsumerThread implements Runnable {

  private final KafkaConsumer<String, String> consumer;
  private final String topic;

  public NotificationConsumerThread(String brokers, String groupId, String topic) {
    Properties prop = createConsumerConfig(brokers, groupId);
    this.consumer = new KafkaConsumer<>(prop);
    this.topic = topic;
    this.consumer.subscribe(Arrays.asList(this.topic));
  }

  private static Properties createConsumerConfig(String brokers, String groupId) {
    Properties props = new Properties();
    props.put("bootstrap.servers", brokers);
    props.put("group.id", groupId);
    props.put("enable.auto.commit", "true");
    props.put("auto.commit.interval.ms", "1000");
    props.put("session.timeout.ms", "30000");
    props.put("auto.offset.reset", "earliest");
    props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
    props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
    return props;
  }

  @Override
  public void run() {
    while (true) {
      ConsumerRecords<String, String> records = consumer.poll(100);
      for (ConsumerRecord<String, String> record : records) {
        System.out.println("Receive message: " + record.value() + ", Partition: "
            + record.partition() + ", Offset: " + record.offset() + ", by ThreadID: "
            + Thread.currentThread().getId());
      }
    }

  }


}
```

When this consumer thread runs, it will poll the data from the topics or partitions.

3.4.4 NotificationConsumerGroup.java

```
package com.howtoprogram.kafka.multipleconsumers;

import java.util.ArrayList;
import java.util.List;

public class NotificationConsumerGroup {

  private final int numberOfConsumers;
  private final String groupId;
  private final String topic;
  private final String brokers;
  private List<NotificationConsumerThread> consumers;

  public NotificationConsumerGroup(String brokers, String groupId, String topic,
      int numberOfConsumers) {
    this.brokers = brokers;
    this.topic = topic;
    this.groupId = groupId;
    this.numberOfConsumers = numberOfConsumers;
    consumers = new ArrayList<>();
    for (int i = 0; i < this.numberOfConsumers; i++) {
      NotificationConsumerThread ncThread =
          new NotificationConsumerThread(this.brokers, this.groupId, this.topic);
      consumers.add(ncThread);
    }
  }

  public void execute() {
    for (NotificationConsumerThread ncThread : consumers) {
      Thread t = new Thread(ncThread);
      t.start();
    }
  }

  /**
   * @return the numberOfConsumers
   */
  public int getNumberOfConsumers() {
    return numberOfConsumers;
  }

  /**
   * @return the groupId
   */
  public String getGroupId() {
    return groupId;
  }

}
```

This class creates a group of consumer threads based on the given parameters:
brokers: The Kafka brokers to which consumers group will connect

groupId: The group id. All consumers on this group will have the same groupId

topic: The topic to which the consumers group will fetch the data

numberOfConsumer: the number of consumers will be created for the group

3.4.5 MultipleConsumersMain.java

```
package com.howtoprogram.kafka.multipleconsumers;

public final class MultipleConsumersMain {

  public static void main(String[] args) {

    String brokers = "localhost:9092";
    String groupId = "group01";
    String topic = "HelloKafkaTopic1";
    int numberOfConsumer = 3;


    if (args != null && args.length > 4) {
      brokers = args[0];
      groupId = args[1];
      topic = args[2];
      numberOfConsumer = Integer.parseInt(args[3]);
    }

    // Start Notification Producer Thread
    NotificationProducerThread producerThread = new NotificationProducerThread(brokers, topic);
    Thread t1 = new Thread(producerThread);
    t1.start();

    // Start group of Notification Consumers
    NotificationConsumerGroup consumerGroup =
        new NotificationConsumerGroup(brokers, groupId, topic, numberOfConsumer);

    consumerGroup.execute();

    try {
      Thread.sleep(100000);
    } catch (InterruptedException ie) {

    }
  }
}
```

This class is the entry point, contain the **main** method for testing our source code. In this class we will create a producer thread to produces 5 messages to the topic which has 3 partitions:***HelloKafkaTopic1***.
Then we create a group of 3 consumers on their own thread to consume the message from the ***HelloKafkaTopic1*** topic.

3.4.5 Run the example.

Create a topic ***HelloKafkaTopic1*** with 3 partitions

```
#cd $APACHE_KAFKA_HOME
./bin/kafka-topics.sh --create --zookeeper localhost:2181 \
      --replication-factor 1 --partitions 3 --topic HelloKafkaTopic1
```

Open the ***MultipleConsumersMain.java*** on the eclipse. ***Right click*** –> ***Run As*** –> ***Java Application*** or use the shortcut: ***Alt+Shift+x, j*** to start the main method.

The output on my eclipse is as below:

```
Produces 3 messages
Sent:Message 0, Partition: 2, Offset: 21
Sent:Message 3, Partition: 2, Offset: 22
Sent:Message 1, Partition: 1, Offset: 24
Sent:Message 4, Partition: 1, Offset: 25
Sent:Message 2, Partition: 0, Offset: 21
Receive message: Message 2, Partition: 0, Offset: 21, by ThreadID: 13
Receive message: Message 1, Partition: 1, Offset: 24, by ThreadID: 14
Receive message: Message 4, Partition: 1, Offset: 25, by ThreadID: 14
Receive message: Message 0, Partition: 2, Offset: 21, by ThreadID: 15
Receive message: Message 3, Partition: 2, Offset: 22, by ThreadID: 15
```

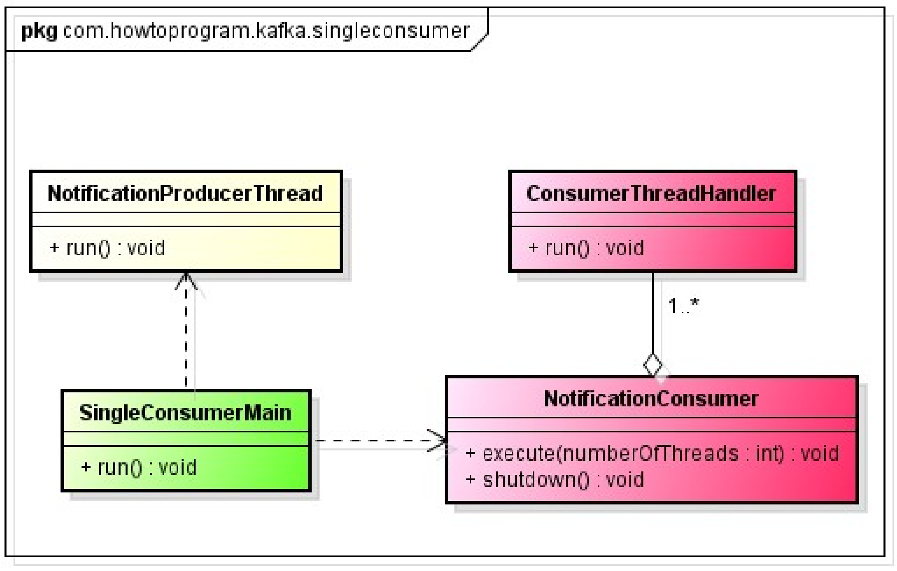
Note that the ***HelloKafkaTopci1*** has 3 partitions. Kafka producer client assigned the Message 0,3 to the partition #2, the Messages 1,4 to the partition #1 and the Message 2 to the partition 0.

The consumer group which includes 3 consumers with their own threads with ThreadID(s): 13, 14, 15. All the messages of the partition #0 were consumed by the consumer thread: #13. The messages of partition #1 were consumed by the consumer thread #1. And messages of partition #2 were consumed by the consumer thread #15.

Note that you may get the different partition numbers and ThreadIDs. However, in this case, each partition will be handled by each consumer thread.

3.5 Model #2: Single consumer, multiple worker processing threads

3.5.1 Class Diagram



Single consumer, multiple worker processing threads – Class Diagram

The source code for this part includes 4 classes too:

NotificationProducerThread.java is a producer thread, produces message to the Kafka brokers.

NotificationConsumer.java is a consumer which has a pool of background threads, receives message from the topic and dispatch to the the pool.

ConsumerThreadHandler.java is a kind of worker thread which handle business processing for the message which is dispatched from the NotificationConsumer.

SingleConsumerMain.java. includes the **main** method, run the program to produce and consume messages.

3.5.2 **NotificationProducerThread.java**

```
package com.howtoprogram.kafka.singleconsumer;

import java.util.Properties;

import org.apache.kafka.clients.producer.Callback;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.producer.RecordMetadata;

public class NotificationProducerThread implements Runnable {

  private final KafkaProducer<String, String> producer;
  private final String topic;

  public NotificationProducerThread(String brokers, String topic) {
    Properties prop = createProducerConfig(brokers);
    this.producer = new KafkaProducer<String, String>(prop);
    this.topic = topic;
  }

  private static Properties createProducerConfig(String brokers) {
    Properties props = new Properties();
    props.put("bootstrap.servers", brokers);
    props.put("acks", "all");
    props.put("retries", 0);
    props.put("batch.size", 16384);
    props.put("linger.ms", 1);
    props.put("buffer.memory", 33554432);
    props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
    props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
    return props;
  }

  @Override
  public void run() {
    System.out.println("Produces 5 messages");
    for (int i = 0; i < 5; i++) {
      String msg = "Message " + i;
      producer.send(new ProducerRecord<String, String>(topic, msg), new Callback() {
        public void onCompletion(RecordMetadata metadata, Exception e) {
          if (e != null) {
            e.printStackTrace();
          }
          System.out.println("Sent:" + msg + ", Offset: " + metadata.offset());
        }
      });
      try {
        Thread.sleep(100);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }

    }

    // closes producer
    producer.close();

  }
}
```

This is the producer thread which will produces **5** messages to the broker

3.5.3 ConsumerThreadHandler.java

```
package com.howtoprogram.kafka.singleconsumer;

import org.apache.kafka.clients.consumer.ConsumerRecord;

public class ConsumerThreadHandler implements Runnable {

  private ConsumerRecord consumerRecord;

  public ConsumerThreadHandler(ConsumerRecord consumerRecord) {
    this.consumerRecord = consumerRecord;
  }

  public void run() {
    System.out.println("Process: " + consumerRecord.value() + ", Offset: " + consumerRecord.offset()
        + ", By ThreadID: " + Thread.currentThread().getId());
  }
}
```

This thread processes the message dispatched from the consumer. In this example, it simply print out the messages, offsets on the topic and the current ThreadID.

3.5.4 NotificationConsumer.java

```
package com.howtoprogram.kafka.singleconsumer;

import java.util.Arrays;
import java.util.Properties;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.ThreadPoolExecutor;
import java.util.concurrent.TimeUnit;

import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;

public class NotificationConsumer {

  private final KafkaConsumer<String, String> consumer;
  private final String topic;
  // Threadpool of consumers
  private ExecutorService executor;

  public NotificationConsumer(String brokers, String groupId, String topic) {
    Properties prop = createConsumerConfig(brokers, groupId);
    this.consumer = new KafkaConsumer<>(prop);
    this.topic = topic;
    this.consumer.subscribe(Arrays.asList(this.topic));
  }

  /**
   * Creates a {@link ThreadPoolExecutor} with a given number of threads to consume the messages
   * from the broker.
   *
   * @param numberOfThreads The number of threads will be used to consume the message
   */
  public void execute(int numberOfThreads) {

    // Initialize a ThreadPool with size = 5 and use the BlockingQueue with size =1000 to
    // hold submitted tasks.
    executor = new ThreadPoolExecutor(numberOfThreads, numberOfThreads, 0L, TimeUnit.MILLISECONDS,
        new ArrayBlockingQueue<Runnable>(1000), new ThreadPoolExecutor.CallerRunsPolicy());

    while (true) {
      ConsumerRecords<String, String> records = consumer.poll(100);
      for (final ConsumerRecord record : records) {
        executor.submit(new ConsumerThreadHandler(record));
      }
    }
```

```java
  }

  private static Properties createConsumerConfig(String brokers, String groupId) {
    Properties props = new Properties();
    props.put("bootstrap.servers", brokers);
    props.put("group.id", groupId);
    props.put("enable.auto.commit", "true");
    props.put("auto.commit.interval.ms", "1000");
    props.put("session.timeout.ms", "30000");
    props.put("auto.offset.reset", "earliest");
    props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
    props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
    return props;
  }

  public void shutdown() {
    if (consumer != null) {
      consumer.close();
    }
    if (executor != null) {
      executor.shutdown();
    }
    try {
      if (!executor.awaitTermination(5000, TimeUnit.MILLISECONDS)) {
        System.out
            .println("Timed out waiting for consumer threads to shut down, exiting uncleanly");
      }
    } catch (InterruptedException e) {
      System.out.println("Interrupted during shutdown, exiting uncleanly");
    }
  }

}
```

This class contains a threadpool of ConsumerThreadHandler thread. It receives the message from the topic and dispatch the processing handler for the pool.

3.5.5 SingleConsumerMain.java

```java
package com.howtoprogram.kafka.singleconsumer;

public final class SingleConsumerMain {

  public static void main(String[] args) {

    String brokers = "localhost:9092";
    String groupId = "group01";
    String topic = "HelloKafkaTopic";
    int numberOfThread = 3;

    if (args != null && args.length > 4) {
      brokers = args[0];
      groupId = args[1];
      topic = args[2];
      numberOfThread = Integer.parseInt(args[3]);
    }

    // Start Notification Producer Thread
    NotificationProducerThread producerThread = new NotificationProducerThread(brokers, topic);
    Thread t1 = new Thread(producerThread);
    t1.start();

    // Start group of Notification Consumer Thread
    NotificationConsumer consumers = new NotificationConsumer(brokers, groupId, topic);

    consumers.execute(numberOfThread);

    try {
      Thread.sleep(100000);
    } catch (InterruptedException ie) {

    }
    consumers.shutdown();
  }
}
```

The entry point, contains the main method to run the example. It create a ***NotificationProducerThread*** thread which produces 5 messages to the topic: ***HelloKafkaTopic***. Then, it creates a ***NotificationConsumer*** object which will receive the message from the above topic and dispatch to the pool of 3 *ConsumerThreadHandler* thread for processing.

3.5.6. Run the example.

You can open the ***SingleConsumerMain.java*** on the eclipse. ***Right click*** –> ***Run As*** –> ***Java Application*** or use the shortcut: ***Alt+Shift+x, j*** to start the main method.

The output on my eclipse is as below:

```
Produces 5 messages
Sent:Message 0, Offset: 2602
Sent:Message 1, Offset: 2603
Process: Message 0, Offset: 2602, By ThreadID: 13
Process: Message 1, Offset: 2603, By ThreadID: 14
Sent:Message 2, Offset: 2604
Process: Message 2, Offset: 2604, By ThreadID: 15
Sent:Message 3, Offset: 2605
Process: Message 3, Offset: 2605, By ThreadID: 13
Sent:Message 4, Offset: 2606
Process: Message 4, Offset: 2606, By ThreadID: 14
```

The producer produces 5 messages with offsets from 2602 ~ 2606. Those messages were processed a pool of threads with Ids: 13, 14, 15.

Note that you may get the different Offsets, ThreadID(s).

4 Conclusion

We have taken a look at how to create multi-threaded Apache Kafka consumer with 2 possible models. They have their own pros and cons and depend on the specific circumstance we will decide which one is suitable. Maybe, there are some cases which the model #2 is suitable. In this case, each partition of a topic will be handled by each consumer thread. However, if the number messages for this partition is too much and the consumer fall far behind, we may need to combine both the model #1 and model #2.

Below are the articles related to Apache Kafka topic. If you're interested in them, you can refer to the following links:

Apache Kafka Tutorial

Getting started with Apache Kafka 0.9

Using Apache Kafka Docker

Apache Kafka 0.9 Java Client API Example

Apache Kafka Command Line Interface

How To Write A Custom Serializer in Apache Kafka

Write An Apache Kafka Custom Partitioner

Apache Kafka Connect Example

Apache Kafka Command Line Interface

Apache Flume Kafka Source And HDFS Sink Tutorial

KafkaCurious

Do you have any suggestion on how these consumers can be managed inside J2ee container?

+ 0 − Reply 2 years ago

howtoprogram

Still not understand clearly what you have mentioned about J2ee container. If you want to manage the lifecycle of Kafka consumer with "container", you can take a look at spring-kafka which allow us to manage consumers by Spring container.

+ 0 − Reply 2 years ago

som

good job

+ 0 − Reply 1 year ago

Kyle

I have a doubt about model #2 — how to make sure we have successfully consume the message?

think about we have a job which needs long time to execute, in model #2, we poll data from kafka and then submit it into the thread pool, and then committed, but the polled-out messages have not been consumed, they are still processing. If unfortunately, the progress was killed by -9, we will lost the messages, right ?

What can we do in such case ?

+ 0 − Reply 1 year ago

← Introduction to Docker Compose Write An Apache Kafka Custom Partitioner →