разное

2 февраля 2021 г. 14:29

Использование утилит тестирования библиотеки Kafka Streams

Using Kafka Streams' testing utilities

To use Kafka Streams' testing utilities, you'll need to update your build.gradle file with the following:

```
testCompile  group:'org.apache.kafka', name:'kafka-streams',
➥ version:'1.0.0', classifier:'test'

testCompile  group:'org.apache.kafka', name:'kafka-clients',
➥ version:'1.0.0', classifier:'test'
```

If you're using Maven, use this code:

```
<dependency>
   <groupId>org.apache.kafka</groupId>
   <artifactId>kafka-streams</artifactId>
   <version>1.0.0</version>
   <scope>test</scope>
   <classifier>test</classifier>
 </dependency>

<dependency>
   <groupId>org.apache.kafka</groupId>
   <artifactId>kafka-clients</artifactId>
   <version>1.0.0</version>
   <scope>test</scope>
   <classifier>test</classifier>
 </dependency>
```

тестирование топологии

2 февраля 2021 г. 14:31

пример

Listing 8.1 Setup method for topology test

```
@Before
public  void setUp() {

 // properties construction left out for clarity
 StreamsConfig streamsConfig = new StreamsConfig(props);
 Topology topology = ZMartTopology.build();

 topologyTestDriver =
➥ new ProcessorTopologyTestDriver(streamsConfig, topology);
}
```

Refactored ZMart topology: now you can get the topology from the method call.

Creates the ProcessorTopologyTestDriver

Listing 8.2 Testing the topology

```
@Test
 public void testZMartTopology() {

 // serde creation left out for clarity

 Purchase purchase = DataGenerator.generatePurchase();

 topologyTestDriver.process("transactions",
                            null,
                            purchase,
                            stringSerde.serializer(),
                            purchaseSerde.serializer());

 ProducerRecord<String, Purchase> record =
➥ topologyTestDriver.readOutput("purchases",
                                  stringSerde.deserializer(),
                                  purchaseSerde.deserializer());

 Purchase expectedPurchase =
➥ Purchase.builder(purchase).maskCreditCard().build();
 assertThat(record.value(), equalTo(expectedPurchase));
```

Creates a test object; reuses the generation code from running the topology

Sends an initial record into the topology

Reads a record from the purchases topic

Converts the test object to the expected format

Verifies that the record from the topology matches the expected record

Listing 8.3 Testing the rest of the topology

```
@Test
 public void testZMartTopology() {

 // continuing test from the previous section

  RewardAccumulator expectedRewardAccumulator =
➥ RewardAccumulator.builder(expectedPurchase).build();

 ProducerRecord<String, RewardAccumulator> accumulatorProducerRecord =
➥ topologyTestDriver.readOutput("rewards",
                              stringSerde.deserializer(),
                              rewardAccumulatorSerde.deserializer());

 assertThat(accumulatorProducerRecord.value(),
➥ equalTo(expectedRewardAccumulator));

 PurchasePattern expectedPurchasePattern =
➥ PurchasePattern.builder(expectedPurchase).build();

 ProducerRecord<String, PurchasePattern> purchasePatternProducerRecord =
➥ topologyTestDriver.readOutput("patterns",
                                  stringSerde.deserializer(),
                                  purchasePatternSerde.deserializer());

 assertThat(purchasePatternProducerRecord.value(),
➥ equalTo(expectedPurchasePattern));
 }
```

Reads a record from the rewards topic

Verifies the rewards topic output matches expectations

Reads a record from the patterns topic

Verifies the patterns topic output matches expectations

Testing a state store in the topology

2 февраля 2021 г. 14:33

Testing a state store in the topology

Listing 8.4 Testing the state store

```
StockTransaction stockTransaction =
➥ DataGenerator.generateStockTransaction();

topologyTestDriver.process("stock-transactions",
                stockTransaction.getSymbol(),
                stockTransaction,
                stringSerde.serializer(),
                stockTransactionSerde.serializer());

KeyValueStore<String, StockPerformance> store =
➥ topologyTestDriver.getKeyValueStore("stock-performance-store");

assertThat(store.get(stockTransaction.getSymbol()),
➥ notNullValue());
```

Generates a test record

Processes the record with the test driver

Retrieves the state store from the test topology

Asserts the store contains the expected value

Testing processors and transformers

2 февраля 2021 г. 14:33

Listing 8.5 Testing the `init` method

```
// some details left out for clarity
private ProcessorContext processorContext =
➥ mock(ProcessorContext.class);
private MockKeyValueStore<String, Tuple<List<ClickEvent>,
➥ List<StockTransaction>>> keyValueStore =
➥ new MockKeyValueStore<>();

private AggregatingMethodHandleProcessor processor =
➥ new AggregatingMethodHandleProcessor();

 @Test
 @DisplayName("Processor should initialize correctly")
 public void testInitializeCorrectly() {
  processor.init(processorContext);
  verify(processorContext).schedule(eq(15000L), eq(STREAM_TIME),
➥  isA(Punctuator.class));
  verify(processorContext).getStateStore(TUPLE_STORE_NAME);
 }
```

Mocks the ProcessorContext with Mockito

A mock KeyValueStore object

The class under test

Calls the init method on the processor, triggering method calls on ProcessorContext

Verifies the parameters for the ProcessorContext.schedule method

Verifies retrieving the state store

Listing 8.6 Testing the `punctuate` method

```
@Test
@DisplayName("Punctuate should forward records")
public void testPunctuateProcess(){
  when(processorContext.getStateStore(TUPLE_STORE_NAME))
                       .thenReturn(keyValueStore);

  processor.init(processorContext);
  processor.process("ABC", Tuple.of(clickEvent, null));
  processor.process("ABC", Tuple.of(null, transaction));

  Tuple<List<ClickEvent>,List<StockTransaction>> tuple =
➥ keyValueStore.innerStore().get("ABC");
  List<ClickEvent> clickEvents = new ArrayList<>(tuple._1);
  List<StockTransaction> stockTransactions = new ArrayList<>(tuple._2);

  processor.cogroup(124722348947L);

  verify(processorContext).forward("ABC",
➥ Tuple.of(clickEvents, stockTransactions));

  assertThat(tuple._1.size(), equalTo(0));
  assertThat(tuple._2.size(), equalTo(0));
}
```

Sets mock behavior to return a KeyValueStore when called

Calls init method on processor

Processes a ClickEvent and a StockTransaction

Extracts the entries put into the state store in the process method

Calls the co-group method, which is the method used to schedule punctuate

Validates that the ProcessorContext forwards the expected records

Validates that the collections within the tuple are cleared out

Integration testing

2 февраля 2021 г. 14:37

Вы можете воспользоваться встраиваемым (embedded) кластером Kafka с помощью тестовых библиотек Kafka.

- Под термином «встраиваемый» я понимаю тут большое приложение вроде Kafka или ZooKeeper, работающее в локальном автономном режиме, то есть «встраивание» его в существующее приложение.
- Благодаря встроенному кластеру Kafka появляется возможность выполнения комплексного теста, для которого требуется кластер Kafka, на своей локальной машине в любой момент, как отдельно, так и в составе группы тестов. Это существенно сокращает цикл разработки. Приступим к созданию комплексного теста.
- Для добавления встраиваемого брокера Kafka в тест достаточно одной строки кода, как показано в листинге 8.8

Listing 8.8 Adding the embedded Kafka broker

```
private static final int NUM_BROKERS = 1;

@ClassRule
public static final EmbeddedKafkaCluster EMBEDDED_KAFKA
➥= new EmbeddedKafkaCluster(NUM_BROKERS);
```

Defines the number of brokers

The JUnit ClassRule annotation

Creates an instance of the EmbeddedKafkaCluster

- Самое важное в этом примере — аннотация ClassRule. Полное описание фреймворков тестирования и JUnit выходит за рамки данной книги, но я уделю немного времени разъяснениям по поводу важности @ClassRule и ее роли в тесте.
  Различие между аннотациями @Rule и @ClassRule заключается в частоте вызовов методов before() и after(). Аннотация @Rule выполняет методы before() и after() для каждого отдельного теста в классе. Аннотация же @ClassRule выполняет методы before() и after() однократно; метод before() выполняется до всех тестов, а метод after() — по завершении последнего теста в классе. Создание EmbeddedKafkaCluster требует довольно много ресурсов, так что имеет смысл делать это лишь один раз для каждого тестового класса.

Создание топиков для тестирования

Listing 8.9 Creating the topics for testing

```
@BeforeClass
public static void setUpAll() throws Exception {
  EMBEDDED_KAFKA.createTopic(YELL_A_TOPIC);
  EMBEDDED_KAFKA.createTopic(OUT_TOPIC);
}
```

BeforeClass annotation

Creates the first source topic

Creates the output topic

с коэффициентом репликации 1, так что можно воспользоваться удобным методом `EmbeddedKafkaCluster.createTopic(String name)`. Если же нам требуется более одной секции или коэффициент репликации больше 1, то настройки по умолчанию нам не подойдут. В подобном случае можно воспользоваться одним из следующих перегруженных методов `createTopic`:

- `EmbeddedKafkaCluster.createTopic(String topic, int partitions, int replication);`
- `EmbeddedKafkaCluster.createTopic(String topic, int partitions, int replication, Properties topicConfig)`.

Тестирование топологии

TESTING THE TOPOLOGY

All the pieces are in place. Now you can follow these steps to execute the integration test:

1. Start the Kafka Streams application.
2. Write some records to the source topic and assert the correct results.
3. Create a new topic matching your pattern.
4. Write some additional records to the newly created topic and assert the correct results.

Let's start with the first two parts of the test (found in src/java/bbejeck/chapter_3/KafkaStreamsYellingIntegrationTest.java).

Listing 8.10 Starting the application and asserting the first set of values

```
// some setup code left out for clarity

kafkaStreams = new KafkaStreams(streamsBuilder.build(), streamsConfig);
```



```
kafkaStreams.start();

List<String> valuesToSendList =
➥ Arrays.asList("this", "should", "yell", "at", "you");
List<String> expectedValuesList =
➥ valuesToSendList.stream()
                   .map(String::toUpperCase)
                   .collect(Collectors.toList());

IntegrationTestUtils.produceValuesSynchronously(YELL_A_TOPIC,
                                                valuesToSendList,
                                                producerConfig,
                                                mockTime);
int expectedNumberOfRecords = 5;
List<String> actualValues =
➥ IntegrationTestUtils.waitUntilMinValuesRecordsReceived(
➥ consumerConfig, OUT_TOPIC, expectedNumberOfRecords);

assertThat(actualValues, equalTo(expectedValuesList));
```

Starts the Kafka Streams application

Specifies the list of values to send

Creates the list of expected values

Produces the values to embedded Kafka

Consumes records from Kafka

Asserts the values read are equal to the expected values

фазы тестирования динамического поведения

- Динамическое добавление топика
- Мы создали новый топик, соответствующий шаблону для узла-источника потокового приложения. После этого мы прошли через те же самые шаги заполнения нового топика данными и потребления записей из топика, обеспечивающего данными узел-источник потокового приложения. В конце теста мы проверяем, совпадают ли прочитанные результаты с ожидаемыми.

Listing 8.11 Starting the application and asserting values

```
EMBEDDED_KAFKA.createTopic(YELL_B_TOPIC);

valuesToSendList = Arrays.asList("yell", "at", "you", "too");

expectedValuesList = valuesToSendList.stream()
                                     .map(String::toUpperCase)
                                     .collect(Collectors.toList());

IntegrationTestUtils.produceValuesSynchronously(YELL_B_TOPIC,
                                                valuesToSendList,
                                                producerConfig,
                                                mockTime);

expectedNumberOfRecords = 4;
List<String> actualValues =
➥IntegrationTestUtils.waitUntilMinValuesRecordsReceived(
➥consumerConfig, OUT_TOPIC, expectedNumberOfRecords);

assertThat(actualValues, equalTo(expectedValuesList));
```

Creates the new topic

Specifies a new list of values to send

Creates the expected values

Produces the values to the source topic of the streaming application

Consumes the results of the streaming application

Asserts that the expected results match the actual results