KIP-267: Add Processor Unit Test Support to Kafka Streams Test Utils

Status

Current state: *Merged (target: 1.2: https://github.com/apache/kafka/pull/4736)*

Discussion thread: [DISCUSS] KIP-267: Add Processor Unit Test Support to Kafka Streams Test Utils

JIRA: **KAFKA-6473** - Getting issue details... STATUS

Released: 1.2.0

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

Motivation

KIP-247 adds support for testing a complete topology, but authors of Processor, Transformer, and ValueTransformer implementations can benefit from writing lighter and faster unit tests.

This isn't impossible today, but it requires writing some fairly complicated mock code for the ProcessorContext. We want to simplify this task by providing a general purpose ProcessorContext for unit testing.

Public Interface

```
package org.apache.kafka.streams.processor;

public class MockProcessorContext extends ProcessorContext {
    public MockProcessorContext(final String applicationId, final TaskId taskId, final Properties config);
    public MockProcessorContext(final String applicationId, final TaskId taskId, final Properties config, final
File stateDir);

    @Override StreamsMetrics metrics(); // return a StreamsMetrics instance for the processor and test code to
use

    // high level metadata inherited from ProcessorContext (set in constructor) ===

    @Override String applicationId();
    @Override TaskId taskId();
    @Override Map<String, Object> appConfigs();
    @Override Map<String, Object> appConfigsWithPrefix(String prefix);
    @Override Serde<?> keySerde();
    @Override Serde<?> valueSerde();
    @Override File stateDir();

    // record metadata ============================================================

    // setters provided for test code to use
    void setRecordMetadata(String topic, int partition, long offset, long timestamp);
    void setRecordMetadataTopic(String topic);
    void setRecordMetadataPartition(int partition);
    void setRecordMetadataOffset(long offset);
    void setRecordMetadataTimestamp(long timestamp);
```

```
    // getters inherited from ProcessorContext
    @Override String topic();
    @Override int partition();
    @Override long offset();
    @Override long timestamp();

    // mocked methods ============================================================

    // StateStore setter and getter
    @Override void register(StateStore store, boolean loggingEnabledIsDeprecatedAndIgnored,
StateRestoreCallback stateRestoreCallback);
    @Override StateStore getStateStore(String name);


    // Punctuator capture: NOTE: punctuators will not be triggered automatically. Instead, they'll be captured
so test code can retrieve and trigger them if desired.
    @Override Cancellable schedule(long intervalMs, PunctuationType type, Punctuator callback);
    @Override void schedule(long interval); // throw UnsupportedOperationException
    List<CapturedPunctuator> scheduledPunctuators();

    // captures forward() data
    @Override <K, V> void forward(K key, V value);
    @Override <K, V> void forward(K key, V value, int childIndex);
    @Override <K, V> void forward(K key, V value, String childName);

    // returns captured forward data
    <K, V> List<KeyValue<K, V>> forwarded();
    <K, V> List<KeyValue<K, V>> forwarded(int childIndex);
    <K, V> List<KeyValue<K, V>> forwarded(String childName);

    // clears captured forward data
    void resetForwards();

    // captures whether commit() gets called
    @Override void commit();

    // true iff commit() has been called
    boolean committed();

    // resets whether commit() has been called
    void resetCommits();

        // structure for capturing punctuators in to schedule()
    public static class CapturedPunctuator {
        public CapturedPunctuator(final long intervalMs, final PunctuationType punctuationType, final
Punctuator punctuator) {}

        public long getIntervalMs();
        public PunctuationType getPunctuationType();
        public Punctuator getPunctuator();
    }
}
```

Proposed Changes

We add the above **new class** to the `kafka-streams-test-utils` artifact (introduced in 1.1.0).

In the initial release, we mark our new class with annotation `@Evolving`.

All the methods annotated with `@Override` are implementing the ProcessorContext interface.

We add new methods for the purposes of setting some properties (like topic, partition, etc.) that the processor may want to access.

We also capture data that the processor may put into the context, like k/v pairs in calls to `forward()`, but also state stores that get registered, punctuators that get scheduled, and whether `commit()` gets called.

We provide a mechanism to reset the commit and forwarded data captures to make it easy to make assertions about the behavior of the processor after various calls to `Processor#process()`.

There is one method we plan not to implement:

- `void schedule(long interval)`: it's not possible to implement it without providing a handle on the processor to the context. This is a little tricky, since the mocked ProcessorContext may be used to test not just a Processor, but also a Transformer or a ValueTransformer. Since this method is deprecated in the interface, we feel it's reasonable to update client code to use Punctuators instead. This method will throw an UnsupportedOperationException.

Also note that we won't automatically trigger sheduled punctuators. Instead, we'll just capture them so test code can retrieve and trigger them if desired. This keeps the mock truly a "mock" and not a "driver", which maintains the simplicity of the code and enables users to write linear and sensible tests. If folks want to have their punctuators triggered automatically, they can use the TopologyTestDriver, which does that. This is a potential sharp edge, so I'm planning to make sure it's well documented in both the javadoc and html docs.

Example Usage

This mocked ProcessorContext would enable unit tests like the following:

```
// ============================
// Initialize the test harness

final Properties config = new Properties();
final MockProcessorContext context = new MockProcessorContext("test-app", new TaskId(0, 0), config);
final KeyValueStore<String, Long> myStore = ...;
context.register(myStore, false, null);
final Processor<String, Long> processor = new MyProcessor();
processor.init(context);

// ==============================
// Verify some processor behavior
processor.process("a", 1L);
processor.process("a", 2L);
final KeyValue<Object, Object> capture1 = context.forwarded().get(0);
Assert.assertEquals(new KeyValue<>("a", 1L), capture1);
final KeyValue<Object, Object> capture2 = context.forwarded().get(1);
Assert.assertEquals(new KeyValue<>("a", 2L), capture2);

Assert.assertTrue(context.committed());

context.resetForwards();
context.resetCommits();

processor.process("a", 3L);
final KeyValue<Object, Object> capture3 = context.forwarded().get(0);
Assert.assertEquals(new KeyValue<>("a", 3L), capture3);
Assert.assertFalse(context.committed());

// ================================
// Verify some Punctuator behavior
context.scheduledPunctuators().get(0).getPunctuator().punctuate(0L);
Assert.assertNull(myStore.get("a")); // assuming the Processor registers a Punctuator that clears this key
```

Compatibility, Deprecation, and Migration Plan

We are only adding a new class. There are no compatiblity issues.

Test Plan

We need to test all added classes with unit tests. Integration or system test are not required.

Rejected Alternatives

None.