KIP-167: Add interface for the state store restoration process

Current state: *Accepted [VOTE]: 167 Add interface for the state store restoration process*

Discussion thread:*here*

JIRA: Unable to render Jira issues macro, execution error.

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

Motivation

Currently, when restoring a state store in a Kafka Streams application, we put one key-value pair at a time into the store.

This proposal aims to make this recovery more efficient for certain persistent stores that can be optimized for bulk writes by adding a new interface with "restoreAll" functionality.

Additionally this proposal will add an interface used as an event listener to do the following:

1. Notification when the restoration (bulk or not) process starts.
2. Intermediate notification as batches are restored with number of records and last offset restored.
3. Notification when the restoration (again bulk or not) process ends.

The proposed listener interface will be available for two use cases:

1. External or user notification of state restoration progress for monitoring purposes when the application is fully online. This will require adding a setter method on the `KafkaStreams` instance described in the next section.
2. Internal or per state store notification so the state store can perform any required resource management at the beginning or end of the restoration. Closing and re-opening a RocksDB database to use bulk loading configurations is one intended result of providing this listener.

We'll outline these use cases in more detail below.

NOTE: This work is building off ideas originally proposed in Unable to render Jira issues macro, execution error. and will be incorporated into this KIP.

Public Interfaces

This KIP will introduce the following interfaces:

- The `BatchingStateRestoreCallback interface`.
- The `StateRestoreListener` interface.

```
public interface BatchingStateRestoreCallback extends StateRestoreCallback {

   void restoreAll(Collection<KeyValue<byte[], byte []>> records);
}
```

This interface will allow for state stores to implement a bulk loading approach during the restore phase. The `StateRestoreCallback` interface is kept as is for backwards compatibility

```
public interface StateRestoreListener {

    void onRestoreStart(TopicPartition topicPartition, StateStore storeName, long startingOffset, long
endOffset);

    void onBatchRestored(TopicPartition topicPartition, String storeName, long batchEndOffset, long
numRestored);

    void onRestoreEnd(TopicPartition topicPartition, String storeName, long totalRestored);
}
```

The `onBatchRestored` method is called after records retrieved from each `poll()` call have been restored. This is to give users a sense of progress being made in the restore process.

The number of times `onBatchRestored` is called is (`Total records in change log / MAX_POLL_RECORDS`).

The changes also include adding a `setter` method on the `KafkaStreams` object, named `setGlobalStateRestoreListener` to reinforce the fact the listener is for the entire application

```
public void setGlobalStateRestoreListener(final StateRestoreListener stateRestoreListener)
```

Default Implementations

As a convenience for users wanting to leverage the `StaterRestoreListener` for state store callbacks as part of this KIP we'll also add the following abstract classes:

For single action state restoration, there is `AbstractNotifyingRestoreCallback`

```
public abstract  class AbstractNotifyingRestoreCallback  implements StateRestoreCallback, StateRestoreListener {


    @Override
    public void onRestoreStart(TopicPartition topicPartition, String storeName, long startingOffset, long
endingOffset) {

    }

    @Override
    public void onBatchRestored(TopicPartition topicPartition, String storeName, long batchEndOffset, long
numRestored) {

    }

    @Override
    public void onRestoreEnd(TopicPartition topicPartition, String storeName, long totalRestored) {

    }
}
```

For the corresponding bulk action state restoration, we have `AbstractBatchingRestoreCallback`

```
public abstract class AbstractBatchingRestoreCallback implements BatchingStateRestoreCallback,
StateRestoreListener {

    @Override
    public void restore(byte[] key, byte[] value) {
        throw new UnsupportedOperationException("Single restore not supported");
    }

    @Override
    public void onRestoreStart(TopicPartition topicPartition, String storeName, long startingOffset, long
endingOffset) {

    }

    @Override
    public void onBatchRestored(TopicPartition topicPartition, String storeName, long batchEndOffset, long
numRestored) {

    }

    @Override
    public void onRestoreEnd(TopicPartition topicPartition, String storeName, long totalRestored) {

    }
}
```

StateRestoreListener Use Cases

The first use case is user updates of the restore progress - In this case users of a Kafka Streams application want to receive updates of the restoration progress and publish those updates to a UI for example. The `StateRestoreListener` set via the `KafkaStreams.setGlobalStateRestoreListener` method functions as a single, global listener reporting on the restoration status for all state stores in an application. Additionally, the `StateRestoreListener` also reports on the bootstrapping progress of any `GlobalKTables` defined in the application.

The second use case is internal state store management, closing and re-opening a RocksDB instance for bulk loading with different configuration settings for example. In this case implementors of a custom store want notification of restoration start, progress and ending for state manage purposes. In this case, the `StateRestoreListener` implementation is used internally by the given state store. In this use case, users can specify a `StateStoreListener` per store, but the intent here is not for reporting but for internal state management.

To use the listener functionality users will implement the `StateRestoreListener` interface in addition to the `StateRestoreCallback` or `BatchingStateRestoreCallback` interfaces when constructing their callbacks. Providing the callback is still done via the `ProcessorContext.register` method.

During the restoration process the type of the `restoreCallback` is inspected and if it implements the `StateRestoreListener` then the listener methods are executed. With this in mind, the `StateStoreListener` API can be called in two places (although two different implementations);

1. If the instance level listener is set via the `KSteam.setStateRestoreListener` method, then that listener will be executed for each `poll` call.
2. If the provided state-store-level callback extends the `StateRestoreListener` interface, then those listener methods triggered for each poll call that is restoring that specific store as well.

Compatibility, Deprecation, and Migration Plan

- Since the `BatchingStateResoreCallback` extends the `StateRestoreCallback` there should be no impact to classes already implementing this interface.
- The `StateRestoreContext` interface is an addition to the code base so no impact is expected.
- The addition of a setter method on the `KafkaStreams` object adds no impact to existing code.

- Abstract classes implementing the different callback approaches and the `StateRestoreListener` interface with no-op methods are provided.

Rejected Alternatives

N/A