KIP-445: In-memory Session Store

- Status
- Motivation
- Public Interfaces
- Proposed Changes
- Compatibility, Deprecation, and Migration Plan
- Rejected Alternatives

Status

Current state: Adopted (2.3.0)

Discussion thread: here

JIRA: KAFKA-8029

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

Motivation

Currently we provide three types of built-in state stores (key-value, window, and session), for which the key-value and window varieties offer both persistent (RocksDB) and in-memory versions. The session store however only has the RocksDB implementation at this point. We should finish rounding out the API and allow users to take full advantage of in-memory performance.

Public Interfaces

This KIP adds one public method to Stores to return an in-memory SessionBytesStoreSupplier

```
    public static SessionBytesStoreSupplier inMemorySessionStore(final String name, final Duration
retentionPeriod);
```

Proposed Changes

The implementation of the in-memory SessionStore can largely follow that of the current InMemoryWindowStore, with a few adjustments; we need to store the start and end timestamp of each record and be able to serve range queries according to earliestSessionEndTime and final long latestSessionStartTime. To do so efficiently I propose adding one layer to the nested NavigableMap approach of the Window Store, where we store records in a doubly-nested map from endTimestamp key startTimestamp. This will allow us to efficiently remove expired records and conserve memory.

Compatibility, Deprecation, and Migration Plan

- N/A

Rejected Alternatives

One alternative that was considered is implementing the underlying byte store similar to how the caching layer stores records, ie as a single map with <Windowed<Key>, value> pairs. These could be sorted according to endTimestamp first, followed by key (and then startTimestamp) in order to preserve ease of removing expired records. This could arguably result in cleaner/less code than a doubly-nested map, however it seems likely to be less clear overall in particular and make it more complicated to fetch records within a certain key and/or time range without iterating over most of the entire map (as the caching layer effectively does currently)

Also, it was discussed whether we really need to include the innermost nested map <startTimestamp value> : each endTimestamp, key pair **should** be unique as these sessions would be merged into one, however currently users can get access to a session store through a processor and are not disallowed from inserting records corresponding to overlapping sessions. We may want to move the session-merging logic from the processor into the underlying store at some point, however for now we should continue to allow arbitrary sessions to be inserted. This can easily be returned to and refactored if choose to at a later time.