KIP-439: Cleanup built-in Store interfaces

Status

Current state: *"Under Discussion"*

Discussion thread: [DISCUSS] KIP-439: Deprecate Interface WindowStoreIterator

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

Released: 2.4

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

Motivation

The window and session store interfaces have confusing method names and return types:

- The `ReadOnlyWindowStore` interface has multiple methods to `fetch()` data. However, the return types are mixed up. Two methods return `WindowStoreIterator` while all others return `KeyValueIterator`.
- The `ReadOnlyWindowStore` interface has method `fetch()`, `fetchAll()`, and `all()` that all do range queries with different parameters.
- The `ReadOnlySessionStore` interface has `fetch()` method while the `SessionStore` interface has `findSession()` methods that all do range queries with different parameters
- The `WindStore#fetch(K, long)` and `SessionStore#fetchSession(k, long, long)` return a single value of a stored window or session but don't do a range query.

We should align the return types and method nodes of both interfaces to get a unified API.

Additionally, the provided metrics with name `fetch` is used for all above mentioned methods (ie, implicit roll-up) making it hard to reason about the metrics.

Public Interfaces

We propose to deprecate interface `WindowStoreIterator` and the following methods. Furthermore, new `range(...)` and `get()` methods are added to replace existing ones.

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


@Deprecate
public interface WindowStoreIterator<V> extends KeyValueIterator<Long, V>, Closeable;

public interface ReadOnlyWindowStore<K, V> {
  // already deprecated (cf. KIP-358)
  @Deprecate
  WindowStoreIterator<V> fetch(K key, long timeFrom, long timeTo);
  @Deprecate
  KeyValueIterator<Windowed<K>, V> fetch(K from, K to, long timeFrom, long timeTo);
  @Deprecate
  KeyValueIterator<Windowed<K>, V> fetchAll(long timeFrom, long timeTo);

  // kept
  KeyValueIterator<Windowed<K>, V> all();

  // newly deprecated
```

```
  @Deprecate
  V fetch(K key, long time);
  @Deprecate
  WindowStoreIterator<V> fetch(K key, Instant from, Instant to) throws IllegalArgumentException;
  @Deprecate
  KeyValueIterator<Windowed<K>, V> fetch(K from, K to, Instant fromTime, Instant toTime) throws
IllegalArgumentException;
  @Deprecate
  KeyValueIterator<Windowed<K>, V> fetchAll(Instant from, Instant to) throws IllegalArgumentException;

  // new
  V get(K key, long windowStartTime);
  KeyValueIterator<Windowed<K>, V> range(K key, Instant from, Instant to) throws IllegalArgumentException;
  KeyValueIterator<Windowed<K>, V> range(K from, K to, Instant from, Instant to) throws
IllegalArgumentException;
  KeyValueIterator<Windowed<K>, V> range(Instant from, Instant to) throws IllegalArgumentException;
}

// note: `WindowStore` does not deprecate overloads with primitive types
//       hence, we need to deprecate them now, and add corresponding `range()` methods
public interface WindowStore<K, V> extends ReadOnlyWindowStore<K, V> {
  // deprecate
  @Deprecate
  WindowStoreIterator<V> fetch(K key, long timeFrom, long timeTo);
  @Deprecate
  KeyValueIterator<Windowed<K>, V> fetch(K from, K to, long timeFrom, long timeTo);
  @Deprecate
  KeyValueIterator<Windowed<K>, V> fetchAll(long timeFrom, long timeTo);

  // new
  KeyValueIterator<Windowed<K>, V> range(K key, long fromTime, long toTime);
  KeyValueIterator<Windowed<K>, V> range(K from, K to, long fromTime, long toTime);
  KeyValueIterator<Windowed<K>, V> range(long fromTime, long toTime);
}

public interface ReadOnlySessionStore<K, V> {
  // deprecated
  @Deprecate
  KeyValueIterator<Windowed<K>, AGG> fetch(K key);
  @Deprecate
  KeyValueIterator<Windowed<K>, AGG> fetch(K from, K to);

  // new
  KeyValueIterator<Windowed<K>, AGG> range(K key);
  KeyValueIterator<Windowed<K>, AGG> range(K from, K to);
}

public interface SessionStore<K, V> extends ReadOnlySessionStore<K, V> {
  // deprecated
  @Deprecate
  KeyValueIterator<Windowed<K>, AGG> findSessions(K key, long earliestSessionEndTime, long
latestSessionStartTime);
  @Deprecate
  KeyValueIterator<Windowed<K>, AGG> findSessions(K keyFrom, K keyTo, long earliestSessionEndTime, long
latestSessionStartTime);
  @Deprecate
  AGG fetchSession(K key, long startTime, long endTime);

  // new
  KeyValueIterator<Windowed<K>, AGG> range(K key, long earliestSessionEndTime, long latestSessionStartTime);
  KeyValueIterator<Windowed<K>, AGG> range(K keyFrom, K keyTo, long earliestSessionEndTime, long
latestSessionStartTime);
  AGG get(K key, long sessionStartTime, long sessionEndTime);
}

// unmodified: just added for completeness
public interface ReadOnlyKeyValueStore<K, V> {
  // kept
  KeyValueIterator<K, V> all();
}
```

```
// Metrics: kafka.streams
type = stream-[storeType]-metrics
client-id = [thread ID]
task-id = [task ID]
[storeType]-id = [store name]
level = DEBUG

// deprecated:
[store-type] = rocksdb-window-state / in-memory-window-state / rocksdb-session-state / in-memory-session-state
name = fetch-latency-avg
name = fetch-latency-max
name = fetch-rate
name = fetch-total

[store-type] = rocksdb-state / in-memory--state
name = all-latency-avg
name = all-latency-max
name = all-rate
name = all-total

name = range-latency-avg
name = range-latency-max
name = range-rate
name = range-total


// new
[store-type] = rocksdb-window-state / in-memory-window-state
name = get-latency-avg
name = get-latency-max
name = get-rate
name = get-total

name = range-single-key-time-range-latency-avg
name = range-single-key-time-range-latency-max
name = range-single-key-time-range-rate
name = range-single-key-time-range-total

name = range-key-and-time-latency-avg
name = range-key-and-time-latency-max
name = range-key-and-time-rate
name = range-key-and-time-total

name = range-time-latency-avg
name = range-time-latency-max
name = range-time-rate
name = range-time-total

name = range-all-latency-avg
name = range-all-latency-max
name = range-all-rate
name = range-all-total

[store-type] = rocksdb-session-state / in-memory-session-state
name = range-key-latency-avg
name = range-key-latency-max
name = range-key-rate
name = range-key-total

name = range-single-key-time-range-latency-avg
name = range-single-key-time-range-latency-max
name = range-single-key-time-range-rate
name = range-single-key-time-range-total

name = range-single-key-latency-avg
name = range-single-key-latency-max
name = range-single-key-rate
name = range-single-key-total
```

```
name = range-key-and-time-latency-avg
name = range-key-and-time-latency-max
name = range-key-and-time-rate
name = range-key-and-time-total

name = get-latency-avg
name = get-latency-max
name = get-rate
name = get-total

[store-type] = rocksdb-state / in-memory-state
name = range-all-latency-avg
name = range-all-latency-max
name = range-all-rate
name = range-all-total

name = range-key-latency-avg
name = range-key-latency-max
name = range-key-rate
name = range-key-total
```

Proposed Changes

We propose to replace `WindowStoreIterator` with `KeyValueIterator` as return type for all range query methods. Additionally, the return type for window /session stores should be changed to `Windowed<K>` key type instead of returning only a `Long` that encoded the window-start timestamp. For this, the `WindowStoreIterator` interface and the corresponding `fetch()` methods are deprecated and new methods with new return types are added.

Furthermore, all methods returning an Iterator should have the same name (`range`), while method that don't return an Iterator should have a different name (`get`).

Additionally, we want to split up metrics to be able to track each method call independently. At the moment, there is a `fetch` metric that is used by all methods called `fetch()/fetchAll()/all()/findSession()` independent of their parameter list. We split the existing `fetch` metric into fine grained metrics per method overload.

Compatibility, Deprecation, and Migration Plan

Because we only deprecate interfaces/methods but don't change any other code this change is backward compatible. Calls to deprecated methods are redirected to new methods and the existing `fetch` metric is still recorded unmodified.

Test Plan

N/A

Rejected Alternatives

N/A