KIP-740: Clean up public API in TaskId

Status

Current state: *Adopted (3.0)*

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

The TaskMetadata class in KafkaStreams is used for encoding information about a particular task, such as its offsets, topic partitions, and taskId. For some reason, the taskId is stored and exposed as a String object, rather than using the actual TaskId class. We should move towards returning this as a TaskId object, since that's literally what it is, and because if/when we add additional fields to the TaskId it will become more and more unwieldy to parse the string encoding in order to extract the actual information about the task.

For example, we are currently working on a feature that will make Kafka Streams more resilient to changes in independent modules of the topology, in order to enable some kinds of topological upgrades. This feature requires adding an additional "namedTopology" field to the TaskId, but as we are still in the experimental and design phase, all APIs for this feature must be internal until we feel it's ready for a KIP and a solidified API. However, we do want to be able to access this information in some way before this KIP is accepted, as part of the reason we are rolling in out in phases like this is so we can do stringent testing beyond just unit/integration tests. Therefore we need some way to access new fields without adding them as public APIs, for example by using reflection on a TaskId object.

At the same time, throughout this unexpectedly long and complicated analysis, it came to our attention how awkward the existing TaskId class is for a public API. It's clearly intended to be a simple metadata class to expose the topicGroupId (subtopology) and partition number, yet it has plenty of totally irrelevant (to a user) utility methods such as de/serialization, and instead of exposing the fields via getters it just makes them public. We should take this opportunity to clean up the TaskId class and move some of those inappropriate methods to an internal utility class.

Public Interfaces

Clean up the public TaskId class by deprecating inappropriate APIs, introducing the required new ones, and replacing topicGroupId with the more common and useful term "subtopology":

org.apache.kafka.streams.processor.TaskId

```
/**
 * The task ID representation composed as subtopology (aka topic group ID) plus the assigned partition ID.
 */
public class TaskId {

    /** The ID of the subtopology, aka topicGroupId. */
    @Deprecated
    public final int topicGroupId;

    /** The ID of the partition. */
    @Deprecated
    public final int partition;

    public int subtopology();

    public int partition();

        /**
     * Experimental feature -- will return null
     */
    public String topologyName();

    /**
     * @deprecated since 3.0, for internal use, will be removed
     */
    @Deprecated
    public void writeTo(final DataOutputStream out, final int version);

    /**
     * @deprecated since 3.0, for internal use, will be removed
     */
    @Deprecated
    public static TaskId readFrom(final DataInputStream in, final int version);

    /**
     * @deprecated since 3.0, for internal use, will be removed
     */
    @Deprecated
    public void writeTo(final ByteBuffer buf, final int version);

    /**
     * @deprecated since 3.0, for internal use, will be removed
     */
    @Deprecated
    public static TaskId readFrom(final ByteBuffer buf, final int version);
}
```

Deprecate the taskId() getter that returns the task id as a string and add an API to return the actual TaskId, also deprecate the string constructor and replace with one that accepts a TaskId:

TaskMetadata

```
class TaskMetadata {

        /**
     * @deprecated since 3.0, not intended for public use
     */
    @Deprecated
    public TaskMetadata(final String taskId,
                        final Set<TopicPartition> topicPartitions,
                        final Map<TopicPartition, Long> committedOffsets,
                        final Map<TopicPartition, Long> endOffsets,
                        final Optional<Long> timeCurrentIdlingStarted);

    /**
     * @return the basic task metadata such as subtopology and partition id
     */
    public TaskId getTaskId() {
        return taskId;
    }

    /**
     * @return the basic task metadata such as subtopology and partition id
     * @deprecated please use {@link #getTaskId()} instead.
     */
    @Deprecated
    public String taskId() {
        return taskId.toString();
    }
}
```

Note that since the name taskId() was already taken, we have to choose a different name for the getter on TaskMetadata, namely getTaskId(). Since this is outside the Kafka standards (which does not use the 'get' prefix for getters), we will actually migrate back to the plain taskId() getter later – once the deprecation period has elapsed, instead of removing the deprecated `String taskId()` we will replace it with a new `TaskId taskId()` API and then deprecate the temporary getTaskId().

Proposed Changes

To separate the implementation from the the public API, we will deprecate the various functional methods on the TaskId class and move their implementation to an internal utility class. Similarly, we will deprecate the public fields and introduce getters for those fields in their place. Those fields will be made private once the deprecation period has passed.

To migrate from the String to TaskId in TaskMetadata, we'll need to deprecate the existing taskId() getter method and add a TaskId getter in its place. Unfortunately the appropriate name for this getter in the Kafka standard is just `taskId()` – which is already taken. We plan to introduce a temporary getter called `getTaskId()` for the deprecation cycle of the `String taskId()` API, and then once that is over we can migrate back to a `TaskId taskId()` getter and then deprecate the temporary getter. It's a bit awkward, but we feel it's better to leave things in the ideal state than to go out of the way to avoid deprecations, especially when it's just one relatively-uncommon method.

Compatibility, Deprecation, and Migration Plan

The TaskId readFrom/writeTo methods and public topicGroupId fields will all be deprecated and removed in a future release. The TaskMetadata.taskId() method will also be deprecated. However, once enough time has passed for it to be removed, we will instead replace it with a new method signature that returns a TaskId instead of a String, and then deprecate the temporary TaskMetadata.getTaskId() API introduced in this KIP.

Rejected Alternatives

Quite a few alternatives were discussed during this seemingly simple KIP. At a high level, they either involved restructuring the existing TaskId class into a hierarchy, or removing any form of TaskId from the API altogether and just decompose any APIs into separate methods for each of its fields. The former was rejected because it would mean that users could not rely on fundamental public contracts like the equals(), toString(), and compareTo() APIs, as they would become implementation details in the internal subclass/implementing class and therefore without public contract. The latter was rejected because the TaskId is and has been a fundamental concept in Kafka Streams, and something we regularly explain to users as a first-class citizen with a specific string representation that may help them make sense of anything from the logs to the local state and directory structure. Also, we feel that the taskid is a natural key on the task space of Kafka Streams and a natural way to think about the tasks themselves, more than just a simple data container class.