KIP-783: Add TaskId field to StreamsException

Status

Current state: *Adopted (3.1.0)*

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

Currently only some exceptions that occur during processing are wrapped as a StreamsException before being handed up to the uncaught exception handler. Unfortunately we don't make any guarantees about which exceptions will or will not be wrapped as a StreamException, which complicates the logic required by a user's custom exception handler and makes it difficult to enforce any kind of compatibility. I have also found the meaning of "StreamsException" to be rather ambiguous – is it just a basic exception type of Kafka Streams, or does it mean the exception came from Kafka Streams vs user code, or does it indicate unrecoverable error? This has been a repeated point of confusion across both users and devs, making it unclear how /when to wrap exceptions for devs, and when to unwrap/how to handle exceptions by users.

It would be cleaner to ensure that all exceptions thrown to the user/handler are wrapped (exactly once) as a StreamsException, and standardize on its definition/use.

Further, many exceptions can be traced back to a particular task that's experiencing an error specific to that part of the topology, or that partition, etc. It can be helpful to have that information when determining how to handle the exception, as well as for debugging purposes. We propose to add a TaskId field to the StreamsException class to help users identify the source of an exception.

Public Interfaces

This KIP will add a new API to the StreamsException class, to expose the TaskId:

StreamsException.java

```
class StreamsException {
    // New constructors
    public StreamsException(final String message, final TaskId taskId);
    public StreamsException(final Throwable throwable, final TaskId taskId);
    public StreamsException(final String message, final Throwable throwable, final TaskId taskId);

    /**
     * @return  the {@link TaskId} that this exception originated from, or {@link Optional#empty()} if the
exception
     *          cannot be traced back to a particular task. Note that the {@code TaskId} being empty does not
     *          guarantee that the exception wasn't directly related to a specific task.
     */
     public Optional<TaskId> taskId() {
        return taskId;
    }
}
```

Proposed Changes

This KIP proposes to

1. Guarantee that every exception that is thrown up to the uncaught exception handler, whether that be the new StreamsUncaughtExceptionHandler or the old generic UncaughtExceptionHandler, is wrapped as a StreamsException.
2. Standardize definition of StreamsException: a top-level exception that indicates an occur has occurred during Streams internal processing, and wraps that error alongside any available info/context. Note that this does not mean the error came from Streams itself.
3. Add a new TaskId field to the StreamsException class, with a getter API to expose it and corresponding constructors. This field will be set for any exception that originates from, or is tied to, a specific task. For example:
    a. Task timeout (ie exceeds the configured *task.timeout.ms* value
    b. User processing error or other exception thrown from Task#process
    c. Exceptions arising from task management, such as suspending/closing/flushing/etc

Note: some exceptions that can theoretically be traced back to a particular task are going to be out of scope for this KIP. For example, some Consumer /Producer exceptions may refer to a topic that corresponds to a specific task. Tackling all possible task-specific exceptions may be considered for followup work, but will not be included in this KIP.

Compatibility, Deprecation, and Migration Plan

Theoretically, a user could have some logic in their exception handlers that check the specific type of the throwable, and act upon it accordingly – for example, if it's a TimeoutException, then they opt to replace the thread. After this KIP, all exceptions will be of the type StreamsException, and this logic would break. However, since we currently make no guarantee that a TimeoutException (or any exception for that matter) will or will not be wrapped as a StreamsException, a user would in fact already need to consider this and check the throwable to determine if it needs to be unwrapped first. This case is actually what this KIP is trying to improve, by giving users a guarantee that they will always be receiving an exception of type StreamsException, and will need to check it for a cause before determining how to react.

Rejected Alternatives

None