KIP-596: Safely abort Producer transactions during application shutdown

Status

Current state: Under Discussion

Discussion thread:

JIRA: KAFKA-9592

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

Motivation

Currently if a transactional producer hits a fatal exception, the caller usually catches the exception and handle it by invoking abortTransaction to abort the transaction, and then closing the producer, which makes sense and sounds clean. The tricky scenario is that abortTransaction is not a safe call when the producer is already in an error state, which means user has to do another try-catch with the first layer catch block, making the error handling pretty annoying. It is more appropriate and user-friendly to ensure that the producer client does not throw the same exception again while aborting transaction.

Proposed Changes

Our proposal is quite straightforward and simple. To avoid throwing the same exception twice, we would remember whether a fatal exception has already been thrown to the application level, so that in abortTransaction we will not throw it again, thus making the function safe to be called in an error state.

Public Interfaces

We shall modify the comments on abortTransaction and remove all exceptions except TimeoutException and InterruptException.

KafkaProducer.java

```
/**
* Aborts the ongoing transaction. Any unflushed produce messages will be aborted when this call is made.
* This call will be a no-op if previous transactional call has thrown an exception and made the producer in an
error state.
*
* Note that this method will raise {@link TimeoutException} if the transaction cannot be aborted before
expiration
* of {@code max.block.ms}. Additionally, it will raise {@link InterruptException} if interrupted.
* It is safe to retry in either case, but it is not possible to attempt a different operation (such as
commitTransaction)
* since the abort may already be in the progress of completing. If not retrying, the only option is to close
the producer.
*
* @throws TimeoutException if the time taken for aborting the transaction has surpassed <code>max.block.ms<
/code>.
* @throws InterruptException if the thread is interrupted while blocked
*/
public void abortTransaction();
```

Suggested Coding Pattern

With this proposed change, users may use transactional APIs as follows.

Example.java

```
Properties props = new Properties();
props.put("bootstrap.servers", "localhost:9092");
props.put("transactional.id", "my-transactional-id");
Producer<String, String> producer = new KafkaProducer<>(props, new StringSerializer(), new StringSerializer());

producer.initTransactions();

try {
    producer.beginTransaction();
    for (int i = 0; i < 100; i++)
        producer.send(new ProducerRecord<>("my-topic", Integer.toString(i), Integer.toString(i)));
    producer.commitTransaction();
} catch (Exception e) {
    producer.abortTransaction();
    if(e instanceof IllegalStateException ||
        e instanceof ProducerFencedException ||
            e instanceof UnsupportedVersionException ||
        e instanceof AuthorizationException) {
        producer.close();
    }
}
```

Compatibility, Deprecation, and Migration Plan

Our proposed change can be considered as minor. In users' perspective, they have less exceptions to worry and handle when calling abortTransaction.

Rejected Alternatives

1. Internally abort any ongoing transaction within `producer.close`, and comment on `abortTransaction` call to warn user not to do it manually.
2. Similar to 1, but get a new `close(boolean abortTransaction)` API call in case some users want to handle transaction state by themselves.
3. Introduce a new abort transaction API with a boolean flag indicating whether the producer is in error state, instead of throwing exceptions
4. Introduce a public API `isInError` on producer for user to validate before doing any transactional API calls

The alternatives are rejected because they all require bigger changes.