KIP-646 Serializer API should support ByteBuffer

This page is meant as a template for writing a KIP. To create a KIP choose Tools->Copy on this page and modify with your content and replace the heading with the next KIP number and a description of your issue. Replace anything in italics with your own description.

Status

Current state: *Under Discussion [One of "Under Discussion", "Accepted", "Rejected"]*

Discussion thread: *here*

JIRA: *here*

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

Motivation

This KIP intends opening the door to implement more effective memory management on Kafka Serializer. Kafka Serializer is used to convert data to a temporary byte[] which is filled into producer/socket buffer. The type of byte array, currently, in Kafka Serializer is pure byte[] so it is hard to reuse the existing byte array for the new messages. Hence, we should introduce an new API to Serializer that enables user to mark the start/end of available bytes in the byte array.

The new response type introduced to Serializer.serialize is ByteBuffer since it comes from java standard library and it is common on NIO code.

The default implementation of new API is based on existent method so binary compatibility is safe. Also, in order to simplify the users' migration, the existent serialize methods are marked as deprecated and all of they get default implementation, which throw UnsupportedOperationException. Hence, users who don't want to extend deprecated methods can migrate to new API safely. Of course, **we should not call deprecated serialize methods anymore so the user-defined serializer which extends only new interface is able to work.**

Apart from Serializer, Partitioner also need some updates since it still accept byte[]. The purposed changes are similar to Serializer that we replace byte[] by ByteBuffer. Also, the new API get default implementation based on existent method.

Public Interfaces

org.apache.kafka.common.serialization.Serializer

```
    @Deprecated
    default byte[] serialize(String topic, T data) {
        throw new UnsupportedOperationException("this method is not used on production anymore");
    }

    @Deprecated
    default byte[] serialize(String topic, Headers headers, T data) {
        return serialize(topic, data);
    }

    default ByteBuffer serialize(String topic, T data, Headers headers) {
        byte[] array = serialize(topic, headers, data);
        return array == null ? null : ByteBuffer.wrap(array);
    }
```

org.apache.kafka.clients.producer.Partitioner

```
    @Deprecated
    default int partition(String topic, Object key, byte[] keyBytes, Object value, byte[] valueBytes, Cluster
cluster) {
        throw new UnsupportedOperationException("this method is not used on production anymore");
    }

    default int partition(String topic, Object key, ByteBuffer keyBytes, Object value, ByteBuffer valueBytes,
Cluster cluster) {
        return partition(topic, key, Utils.getOrCopyArray(keyBytes), value, Utils.getOrCopyArray(valueBytes),
cluster);
    }
```

Proposed Changes

1. Serialized#serialize(String, T) is deprecated and has default implementation
2. Serialized#serialize(String, Headers, T) is deprecated and has default implementation
3. Serialized#serialize(String, T, Headers) is an new method. The order of arguments is different from Serialized#serialize(String, Headers, T) so as to keep the same method name.
4. Partitioner#partition(String, Object, byte[], Object, byte[], Cluster) is deprecated and has default implementation
5. Partitioner#partition(String, Object, ByteBuffer, Object, ByteBuffer, Cluster) is an new method.

Compatibility, Deprecation, and Migration Plan

1. All new APIs have default implementation so BC is not broken
2. All deprecated methods have default implementation so users can migrate to new interfaces without keeping deprecated override methods

Rejected Alternatives

None