Producer API

Apache Kafka

Kafka Producer send() Method

- Asynchronously sends a record to a topic:

```
new ProducerRecord(TOPIC, reading.getId(), reading)
```

- Allows sending many records without blocking for a broker response
- send() method returns a Future
- Two forms:
  - send() method without a callback
  - send() method with a callback – the callback gets invoked when the broker has acknowledged the send [ACK = -1 (all ISR), 0 or 1 (leader)]
- Callbacks for records sent to same partition are executed in the sent order
- Callback receives RecordMetadata which contains a record's partition, offset, and timestamp, and possible Exception if there was an error sending.

Kafka send() Method Exceptions

- InterruptException - If thread is interrupted while blocking
- SerializationException - If key or value can not be serialized using configured serializers
- TimeoutException – when fetching metadata or allocating memory exceeds max.block.ms, or getting **acks** from Broker exceed timeout.ms, etc.
- KafkaException - when Kafka error occurs, but not in public API
- AuthenticationException - if authentication fails
- AuthorizationException - the producer is not allowed to write
- IllegalStateException - if a transactional.id has been configured and no transaction has been started, or when send() invoked on closed producer

Kafka Producer flush() and close() Methods

```
Runtime.getRuntime().addShutdownHook(new Thread(() -> {
    executor.shutdown();
    try {
        executor.awaitTermination(200, TimeUnit.MILLISECONDS);
        log.info("Flushing and closing producer");
        producer.flush();
        producer.close(10_000, TimeUnit.MILLISECONDS);
    } catch (InterruptedException e) {
        log.warn("shutting down", e);
    }
}));
```

Kafka Producer partitionsFor() Method

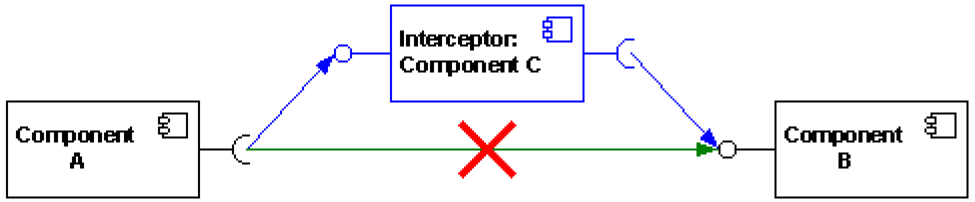
- partitionsFor(topic) - returns meta-data for partitions:

public List<PartitionInfo> partitionsFor(String topic)

- Used by producers that implement their own partitioning – for custom partitioning
- PartitionInfo consists of topic, partition, leader node (Node), replica nodes (Node[]) and inSyncReplica nodes.
- Node consists of id, host, port, and rack

Kafka Producer Interceptors

- Interceptor design pattern:



- Activate interceptors by adding them to interceptor.classes property of the producer

```
props.put(
        ProducerConfig.INTERCEPTOR_CLASSES_CONFIG,
        CountingProducerInterceptor.class.getName());
```

- Producer interceptor methods:

```
public ProducerRecord<K, V> onSend(ProducerRecord<K, V> record)
public void onAcknowledgement(RecordMetadata metadata, Exception exception)
```

Kafka Producer flush() and close() Methods

```
Runtime.getRuntime().addShutdownHook(new Thread(() -> {
    executor.shutdown();
    try {
        executor.awaitTermination(200, TimeUnit.MILLISECONDS);
        log.info("Flushing and closing producer");
        producer.flush();
        producer.close(10_000, TimeUnit.MILLISECONDS);
    } catch (InterruptedException e) {
        log.warn("shutting down", e);
    }
}));
```

Kafka Producer metrics() Method

- metrics() - used to get a map of metrics:

public Map<MetricName,? extends Metric> metrics()

- Returns a full set of producer metrics.
- MetricName consists of name, group, description, and tags (Map).
- Metric consist of a MetricName and a Measurable value (double) or Object (gauge).

Kafka Transactions Simple Example

```
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 (ProducerFencedException | OutOfOrderSequenceException | AuthorizationException e) {
    // We can't recover from these exceptions, so our only option is to close the producer and exit.
    producer.close();
} catch (KafkaException e) {
    // For all other exceptions, just abort the transaction and try again.
    producer.abortTransaction();
}
producer.close();
```

Kafka Transactions – Atomic Read & Write to Topic

```
// Poll and validate deposit events
Deposits = validate(consumer.poll(100));

// Atomically send valid deposits and commit offsets
producer.beginTransaction();
producer.send(validatedDeposits);
producer.sendOffsetsToTransaction(offsets(consumer));
producer.endTransaction();
```

Thank’s for Your Attention!

Trayan Iliev

IPT – Intellectual Products & Technologies

http://iproduct.org/

http://robolearn.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

https://www.facebook.com/IPT.EACAD