0.8.0 SimpleConsumer Example

Using SimpleConsumer

Why use SimpleConsumer?

The main reason to use a SimpleConsumer implementation is you want greater control over partition consumption than Consumer Groups give you.

For example you want to:

1. Read a message multiple times
2. Consume only a subset of the partitions in a topic in a process
3. Manage transactions to make sure a message is processed once and only once

Downsides of using SimpleConsumer

The SimpleConsumer does require a significant amount of work not needed in the Consumer Groups:

1. You must keep track of the offsets in your application to know where you left off consuming.
2. You must figure out which Broker is the lead Broker for a topic and partition
3. You must handle Broker leader changes

Steps for using a SimpleConsumer

1. Find an active Broker and find out which Broker is the leader for your topic and partition
2. Determine who the replica Brokers are for your topic and partition
3. Build the request defining what data you are interested in
4. Fetch the data
5. Identify and recover from leader changes

Finding the Lead Broker for a Topic and Partition

The easiest way to do this is to pass in a set of known Brokers to your logic, either via a properties file or the command line. These don't have to be all the Brokers in the cluster, rather just a set where you can start looking for a live Broker to query for Leader information.

```
private PartitionMetadata findLeader(List<String> a_seedBrokers, int a_port, String a_topic, int a_partition) {
        PartitionMetadata returnMetaData = null;
        loop:
        for (String seed : a_seedBrokers) {
            SimpleConsumer consumer = null;
            try {
                consumer = new SimpleConsumer(seed, a_port, 100000, 64 * 1024, "leaderLookup");

                List<String> topics = Collections.singletonList(a_topic);
                TopicMetadataRequest req = new TopicMetadataRequest(topics);
                kafka.javaapi.TopicMetadataResponse resp = consumer.send(req);

                List<TopicMetadata> metaData = resp.topicsMetadata();

                for (TopicMetadata item : metaData) {
                    for (PartitionMetadata part : item.partitionsMetadata()) {
                        if (part.partitionId() == a_partition) {
                            returnMetaData = part;
                            break loop;
                        }
                    }
                }
            } catch (Exception e) {
                System.out.println("Error communicating with Broker [" + seed + "] to find Leader for [" +
a_topic
                        + ", " + a_partition + "] Reason: " + e);
            } finally {
                if (consumer != null) consumer.close();
            }
        }
        if (returnMetaData != null) {
            m_replicaBrokers.clear();
            for (kafka.cluster.Broker replica : returnMetaData.replicas()) {
                m_replicaBrokers.add(replica.host());
            }
        }
        return returnMetaData;
    }
```

The call to topicsMetadata() asks the Broker you are connected to for all the details about the topic we are interested in.

The loop on partitionsMetadata iterates through all the partitions until we find the one we want. Once we find it, we can break out of all the loops.

Finding Starting Offset for Reads

Now define where to start reading data. Kafka includes two constants to help, kafka.api.OffsetRequest.EarliestTime() finds the beginning of the data in the logs and starts streaming from there, kafka.api.OffsetRequest.LatestTime() will only stream new messages. Don't assume that offset 0 is the beginning offset, since messages age out of the log over time.

```
public static long getLastOffset(SimpleConsumer consumer, String topic, int partition,
                                  long whichTime, String clientName) {
        TopicAndPartition topicAndPartition = new TopicAndPartition(topic, partition);
        Map<TopicAndPartition, PartitionOffsetRequestInfo> requestInfo = new HashMap<TopicAndPartition,
PartitionOffsetRequestInfo>();
        requestInfo.put(topicAndPartition, new PartitionOffsetRequestInfo(whichTime, 1));
        kafka.javaapi.OffsetRequest request = new kafka.javaapi.OffsetRequest(requestInfo, kafka.api.
OffsetRequest.CurrentVersion(),clientName);
        OffsetResponse response = consumer.getOffsetsBefore(request);

        if (response.hasError()) {
            System.out.println("Error fetching data Offset Data the Broker. Reason: " + response.errorCode
(topic, partition) );
            return 0;
        }
        long[] offsets = response.offsets(topic, partition);
        return offsets[0];
    }
```

Error Handling

Since the SimpleConsumer doesn't handle lead Broker failures, you have to write a bit of code to handle it.

```
           if (fetchResponse.hasError()) {
                numErrors++;
                // Something went wrong!
                short code = fetchResponse.errorCode(a_topic, a_partition);
                System.out.println("Error fetching data from the Broker:" + leadBroker + " Reason: " + code);
                if (numErrors > 5) break;

                if (code == ErrorMapping.OffsetOutOfRangeCode())  {
                    // We asked for an invalid offset. For simple case ask for the last element to reset
                    readOffset = getLastOffset(consumer,a_topic, a_partition, kafka.api.OffsetRequest.
LatestTime(), clientName);
                    continue;
                }
                consumer.close();
                consumer = null;
                leadBroker = findNewLeader(leadBroker, a_topic, a_partition, a_port);
                continue;
            }
```

Here, once the fetch returns an error, we log the reason, close the consumer then try to figure out who the new leader is.

```
 private String findNewLeader(String a_oldLeader, String a_topic, int a_partition, int a_port) throws Exception
{
        for (int i = 0; i < 3; i++) {
            boolean goToSleep = false;
            PartitionMetadata metadata = findLeader(m_replicaBrokers, a_port, a_topic, a_partition);
            if (metadata == null) {
                goToSleep = true;
            } else if (metadata.leader() == null) {
                goToSleep = true;
            } else if (a_oldLeader.equalsIgnoreCase(metadata.leader().host()) && i == 0) {
                // first time through if the leader hasn't changed give ZooKeeper a second to recover
                // second time, assume the broker did recover before failover, or it was a non-Broker issue
                //
                goToSleep = true;
            } else {
                return metadata.leader().host();
            }
            if (goToSleep) {
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException ie) {
                }
            }
        }
        System.out.println("Unable to find new leader after Broker failure. Exiting");
        throw new Exception("Unable to find new leader after Broker failure. Exiting");
    }
```

This method uses the findLeader() logic we defined earlier to find the new leader, except here we only try to connect to one of the replicas for the topic /partition. This way if we can’t reach any of the Brokers with the data we are interested in we give up and exit hard.

Since it may take a short time for ZooKeeper to detect the leader loss and assign a new leader, we sleep if we don’t get an answer. In reality ZooKeeper often does the failover very quickly so you never sleep.

Reading the Data

Finally we read the data being streamed back and write it out.

```
            // When calling FetchRequestBuilder, it's important NOT to call .replicaId(), which is meant for
internal use only.
            // Setting the replicaId incorrectly will cause the brokers to behave incorrectly.
            FetchRequest req = new FetchRequestBuilder()
                    .clientId(clientName)
                    .addFetch(a_topic, a_partition, readOffset, 100000)
                    .build();
            FetchResponse fetchResponse = consumer.fetch(req);

            if (fetchResponse.hasError()) {
                    // See code in previous section
            }
            numErrors = 0;

            long numRead = 0;
            for (MessageAndOffset messageAndOffset : fetchResponse.messageSet(a_topic, a_partition)) {
                long currentOffset = messageAndOffset.offset();
                if (currentOffset < readOffset) {
                    System.out.println("Found an old offset: " + currentOffset + " Expecting: " + readOffset);
                    continue;
                }
                readOffset = messageAndOffset.nextOffset();
                ByteBuffer payload = messageAndOffset.message().payload();

                byte[] bytes = new byte[payload.limit()];
                payload.get(bytes);
                System.out.println(String.valueOf(messageAndOffset.offset()) + ": " + new String(bytes, "UTF-
8"));
                numRead++;
                a_maxReads--;
            }

            if (numRead == 0) {
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException ie) {
                }
            }
```

Note that the ‘readOffset’ asks the last read message what the next Offset would be. This way when the block of messages is processed we know where to ask Kafka where to start the next fetch.

Also note that we are explicitly checking that the offset being read is not less than the offset that we requested. This is needed since if Kafka is compressing the messages, the fetch request will return an entire compressed block even if the requested offset isn't the beginning of the compressed block. Thus a message we saw previously may be returned again. Note also that we ask for a fetchSize of 100000 bytes. If the Kafka producers are writing large batches, this might not be enough, and might return an empty message set. In this case, the fetchSize should be increased until a non-empty set is returned.

Finally, we keep track of the # of messages read. If we didn't read anything on the last request we go to sleep for a second so we aren't hammering Kafka when there is no data.

Running the example

The example expects the following parameters:

- Maximum number of messages to read (so we don’t loop forever)
- Topic to read from
- Partition to read from
- One broker to use for Metadata lookup
- Port the brokers listen on

Full Source Code

```
package com.test.simple;

import kafka.api.FetchRequest;
import kafka.api.FetchRequestBuilder;
import kafka.api.PartitionOffsetRequestInfo;
import kafka.common.ErrorMapping;
```

```
import kafka.common.TopicAndPartition;
import kafka.javaapi.*;
import kafka.javaapi.consumer.SimpleConsumer;
import kafka.message.MessageAndOffset;

import java.nio.ByteBuffer;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class SimpleExample {
    public static void main(String args[]) {
        SimpleExample example = new SimpleExample();
        long maxReads = Long.parseLong(args[0]);
        String topic = args[1];
        int partition = Integer.parseInt(args[2]);
        List<String> seeds = new ArrayList<String>();
        seeds.add(args[3]);
        int port = Integer.parseInt(args[4]);
        try {
            example.run(maxReads, topic, partition, seeds, port);
        } catch (Exception e) {
            System.out.println("Oops:" + e);
             e.printStackTrace();
        }
    }

    private List<String> m_replicaBrokers = new ArrayList<String>();

    public SimpleExample() {
        m_replicaBrokers = new ArrayList<String>();
    }

    public void run(long a_maxReads, String a_topic, int a_partition, List<String> a_seedBrokers, int a_port)
throws Exception {
        // find the meta data about the topic and partition we are interested in
        //
        PartitionMetadata metadata = findLeader(a_seedBrokers, a_port, a_topic, a_partition);
        if (metadata == null) {
            System.out.println("Can't find metadata for Topic and Partition. Exiting");
            return;
        }
        if (metadata.leader() == null) {
            System.out.println("Can't find Leader for Topic and Partition. Exiting");
            return;
        }
        String leadBroker = metadata.leader().host();
        String clientName = "Client_" + a_topic + "_" + a_partition;

        SimpleConsumer consumer = new SimpleConsumer(leadBroker, a_port, 100000, 64 * 1024, clientName);
        long readOffset = getLastOffset(consumer,a_topic, a_partition, kafka.api.OffsetRequest.EarliestTime(),
clientName);

        int numErrors = 0;
        while (a_maxReads > 0) {
            if (consumer == null) {
                consumer = new SimpleConsumer(leadBroker, a_port, 100000, 64 * 1024, clientName);
            }
            FetchRequest req = new FetchRequestBuilder()
                    .clientId(clientName)
                    .addFetch(a_topic, a_partition, readOffset, 100000) // Note: this fetchSize of 100000 might
need to be increased if large batches are written to Kafka
                    .build();
            FetchResponse fetchResponse = consumer.fetch(req);

            if (fetchResponse.hasError()) {
                numErrors++;
                // Something went wrong!
                short code = fetchResponse.errorCode(a_topic, a_partition);
```

```
                System.out.println("Error fetching data from the Broker:" + leadBroker + " Reason: " + code);
                if (numErrors > 5) break;
                if (code == ErrorMapping.OffsetOutOfRangeCode())  {
                    // We asked for an invalid offset. For simple case ask for the last element to reset
                    readOffset = getLastOffset(consumer,a_topic, a_partition, kafka.api.OffsetRequest.
LatestTime(), clientName);
                    continue;
                }
                consumer.close();
                consumer = null;
                leadBroker = findNewLeader(leadBroker, a_topic, a_partition, a_port);
                continue;
            }
            numErrors = 0;

            long numRead = 0;
            for (MessageAndOffset messageAndOffset : fetchResponse.messageSet(a_topic, a_partition)) {
                long currentOffset = messageAndOffset.offset();
                if (currentOffset < readOffset) {
                    System.out.println("Found an old offset: " + currentOffset + " Expecting: " + readOffset);
                    continue;
                }
                readOffset = messageAndOffset.nextOffset();
                ByteBuffer payload = messageAndOffset.message().payload();

                byte[] bytes = new byte[payload.limit()];
                payload.get(bytes);
                System.out.println(String.valueOf(messageAndOffset.offset()) + ": " + new String(bytes, "UTF-
8"));
                numRead++;
                a_maxReads--;
            }

            if (numRead == 0) {
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException ie) {
                }
            }
        }
        if (consumer != null) consumer.close();
    }

    public static long getLastOffset(SimpleConsumer consumer, String topic, int partition,
                                     long whichTime, String clientName) {
        TopicAndPartition topicAndPartition = new TopicAndPartition(topic, partition);
        Map<TopicAndPartition, PartitionOffsetRequestInfo> requestInfo = new HashMap<TopicAndPartition,
PartitionOffsetRequestInfo>();
        requestInfo.put(topicAndPartition, new PartitionOffsetRequestInfo(whichTime, 1));
        kafka.javaapi.OffsetRequest request = new kafka.javaapi.OffsetRequest(
                requestInfo, kafka.api.OffsetRequest.CurrentVersion(), clientName);
        OffsetResponse response = consumer.getOffsetsBefore(request);

        if (response.hasError()) {
            System.out.println("Error fetching data Offset Data the Broker. Reason: " + response.errorCode
(topic, partition) );
            return 0;
        }
        long[] offsets = response.offsets(topic, partition);
        return offsets[0];
    }

    private String findNewLeader(String a_oldLeader, String a_topic, int a_partition, int a_port) throws
Exception {
        for (int i = 0; i < 3; i++) {
            boolean goToSleep = false;
            PartitionMetadata metadata = findLeader(m_replicaBrokers, a_port, a_topic, a_partition);
            if (metadata == null) {
                goToSleep = true;
            } else if (metadata.leader() == null) {
                goToSleep = true;
```

```
            } else if (a_oldLeader.equalsIgnoreCase(metadata.leader().host()) && i == 0) {
                // first time through if the leader hasn't changed give ZooKeeper a second to recover
                // second time, assume the broker did recover before failover, or it was a non-Broker issue
                //
                goToSleep = true;
            } else {
                return metadata.leader().host();
            }
            if (goToSleep) {
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException ie) {
                }
            }
        }
        System.out.println("Unable to find new leader after Broker failure. Exiting");
        throw new Exception("Unable to find new leader after Broker failure. Exiting");
    }

    private PartitionMetadata findLeader(List<String> a_seedBrokers, int a_port, String a_topic, int
a_partition) {
        PartitionMetadata returnMetaData = null;
        loop:
        for (String seed : a_seedBrokers) {
            SimpleConsumer consumer = null;
            try {
                consumer = new SimpleConsumer(seed, a_port, 100000, 64 * 1024, "leaderLookup");
                List<String> topics = Collections.singletonList(a_topic);
                TopicMetadataRequest req = new TopicMetadataRequest(topics);
                kafka.javaapi.TopicMetadataResponse resp = consumer.send(req);

                List<TopicMetadata> metaData = resp.topicsMetadata();
                for (TopicMetadata item : metaData) {
                    for (PartitionMetadata part : item.partitionsMetadata()) {
                        if (part.partitionId() == a_partition) {
                            returnMetaData = part;
                            break loop;
                        }
                    }
                }
            } catch (Exception e) {
                System.out.println("Error communicating with Broker [" + seed + "] to find Leader for [" +
a_topic
                        + ", " + a_partition + "] Reason: " + e);
            } finally {
                if (consumer != null) consumer.close();
            }
        }
        if (returnMetaData != null) {
            m_replicaBrokers.clear();
            for (kafka.cluster.Broker replica : returnMetaData.replicas()) {
                m_replicaBrokers.add(replica.host());
            }
        }
        return returnMetaData;
    }
}
```