Kafka 0.9 consumer protocol

WARN: This is an obsolete design. The design that's implemented in Kafka 0.9.0 is described in this wiki.

This page describes the algorithmic workflow of a lightweight consumer which removes the rebalance logic to the broker-side coordinator and the ZK dependency. It is based on the detailed design of the consumer coordinator described here.

This page is used for people to write the consumer client interface with other languages.

A consumer will have the following key components:

1. A coordinator connector which keeps a socket channel with the current coordinator
2. A list of fetchers used for sending fetch requests to brokers

And it keeps the following information:

1. Current server cluster info, including a list of current brokers with their address and the current coordinator id
2. Current consumed partition info, including the topic, broker-partition id, fetched offset, consumed offset, fetch size

Upon startup, a consumer will do the following:

```
Consumer.onStartup:

Input: a list of unknown brokers

1. Construct initial server cluster info from the unknown broker list (while the current coordinator id is
still unknown)

2. Consult known brokers in the cluster to update server cluster info:

2.1. In a round robin fashion, pick a broker in the cluster server, create a socket channel with that broker.

2.2. If the channel can be successfully created, send a cluster metadata request to the broker, receive the
response and update the server cluster (with the current coordinator id) accordingly.

2.3. If the channel cannot be created, try another broker and redo 2.2.

2.4. If all the brokers in the current cluster cannot be connected, restart from the first broker and redo 2.1,
the consumer will keep trying to consult the brokers until it succeeds or is shut down.

3. Try to connect to the current coordinator and send a register request with its interested topic count, its
session timeout, etc.

4. If the registry is rejected by the coordinator, trigger the lost-connection procedure.

5. If it is accepted, initialize and start the coordinator connector with the channel.


Consumer.onLostConnection:

1. Close all fetchers and clean their corresponding queues

2. Shutdown the coordinator connector.

3. Retry step 2 to 5 of the onStart procedure.
```

Upon startup, the consumer's coordinator connector will do the following:

```
Connector.onStartup:

Input: the socket channel to the current coordinator

while (isRunning):

  1. Try block reading from the channel.

  2. Handle the read request from the coordinator.

  3. If no request is received within the session timeout or if the connection is closed or failed, trigger the lost-connection procedure of the consumer.


Connector.handlePingRequest:

1. Send the current consuming partitions' consumed offset as the response back to the coordinator.


Connector.handleStopFetcherRequest:

1. Stop the fetchers of the consumer and clean the corresponding queues.

2. Send the consumed partitions' consumed offset as the response back to the coordinator.


Connector.handleStartFetcherRequest:

1. Read the assigned partition info from the request, and refresh the consumed partition info.

2. Start fetchers with the new consumed partition info.

3. Send the response back to the coordinator.
```