KIP-795: Add public APIs for AbstractCoordinator

Status

Current state: *Discarded*

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

After Kafka moved the rebalancing responsibilities from brokers to clients, many other applications (Kafka Connect, Kafka Streams, Confluent Schema Registry) have relied on the Group Membership Protocol to implement resource allocation amongst distributed processes.

Kafka Connect's WorkerCoordinator class extends **AbstractCoordinator**, and uses the rebalance mechanism to distribute tasks to its workers. In the same spirit, Confluent Kafka Registry - and its SchemaRegistryCoordinator - relies in the **AbstractCoordinator** class for leadership election (which instance of a Schema Registry cluster can accept writes). These two are in addition to the ConsumerCoordinator that the Kafka Consumer uses internally, and indirectly Kafka Streams.

We've adopted this generic, extensible protocol to implement a framework that solves both the distributed resource management and leader election use cases our engineering teams face when building their systems. The powerful primitives exposed by the AbstractCoordinator class have made it relatively easy to build distributed resource management systems on top of Apache Kafka.

A good example of how other projects can leverage **AbstractCoordinator** APIs to implement resource management is this PR to add HA capabilities to Kafka Monitor. We're looking into adding these capabilities to other systems in our Kafka Infrastructure (Kafka Cruise Control being the next one).

We think it's time for the **AbstractCoordinator** to become part of Kafka's public API, so we can ensure backwards compatibility in future versions of the client libraries. In fact, we were inspired by Gwen Shapira's talk at the strangeloop conference in '18 [https://www.youtube.com/watch?v=MmLezWRI3Ys] which gave us a new perspective when redesigning our existing distributed leader election system.

Finally, we feel that the protocol is well designed and extensible, so advertising Kafka's Coordinator capabilities might be a good idea, as it's applicable to many use cases.

Public Interfaces

This KIP introduces a new Coordinator interface in the org.apache.kafka.consumer.clients.consumer package.

Coordinator.java

```
/*
 * Licensed to the Apache Software Foundation (ASF) under one or more
 * contributor license agreements. See the NOTICE file distributed with
 * this work for additional information regarding copyright ownership.
 * The ASF licenses this file to You under the Apache License, Version 2.0
 * (the "License"); you may not use this file except in compliance with
 * the License. You may obtain a copy of the License at
 *
 *    http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
```

```java
 */
package org.apache.kafka.clients.consumer;

import java.io.Closeable;
import java.nio.ByteBuffer;
import java.util.List;
import java.util.Map;
import java.util.Objects;
import org.apache.kafka.clients.consumer.internals.ConsumerCoordinator;
import org.apache.kafka.common.message.JoinGroupRequestData.JoinGroupRequestProtocol;
import org.apache.kafka.common.message.JoinGroupResponseData;
import org.apache.kafka.common.message.JoinGroupResponseData.JoinGroupResponseMember;

/**
 * Coordinator interface implements group management for a single group member by interacting with
 * a designated Kafka broker (the coordinator). Group semantics are provided by extending this class.
 * See {@link ConsumerCoordinator} for example usage.
 *
 * From a high level, Kafka's group management protocol consists of the following sequence of actions:
 *
 * <ol>
 *     <li>Group Registration: Group members register with the coordinator providing their own metadata
 *         (such as the set of topics they are interested in).</li>
 *     <li>Group/Leader Selection: The coordinator select the members of the group and chooses one member
 *         as the leader.</li>
 *     <li>State Assignment: The leader collects the metadata from all the members of the group and
 *         assigns state.</li>
 *     <li>Group Stabilization: Each member receives the state assigned by the leader and begins
 *         processing.</li>
 * </ol>
 *
 * To leverage this protocol, an implementation must define the format of metadata provided by each
 * member for group registration in {@link #metadata()} and the format of the state assignment provided
 * by the leader in {@link #performAssignment(String, String, List)} and becomes available to members in
 * {@link #onJoinComplete(int, String, String, ByteBuffer)}.
 *
 * Note on locking: this class shares state between the caller and a background thread which is
 * used for sending heartbeats after the client has joined the group. All mutable state as well as
 * state transitions are protected with the class's monitor. Generally this means acquiring the lock
 * before reading or writing the state of the group (e.g. generation, memberId) and holding the lock
 * when sending a request that affects the state of the group (e.g. JoinGroup, LeaveGroup).
 */
public interface Coordinator extends Closeable {

    /**
     * Unique identifier for the class of supported protocols (e.g. "consumer" or "connect").
     * @return Non-null protocol type name
     */
    String protocolType();

    /**
     * Get the current list of protocols and their associated metadata supported
     * by the local member. The order of the protocols in the map indicates the preference
     * of the protocol (the first entry is the most preferred). The coordinator takes this
     * preference into account when selecting the generation protocol (generally more preferred
     * protocols will be selected as long as all members support them and there is no disagreement
     * on the preference).
     * @return Non-empty map of supported protocols and metadata
     */
    List<JoinGroupMetadata> metadata();

    /**
     * Invoked prior to each group join or rejoin. This is typically used to perform any
     * cleanup from the previous generation (such as committing offsets for the consumer)
     * @param generation The previous generation or -1 if there was none
     * @param memberId The identifier of this member in the previous group or "" if there was none
     */
    void onJoinPrepare(int generation, String memberId);

    /**
     * Perform assignment for the group. This is used by the leader to push state to all the members
     * of the group (e.g. to push partition assignments in the case of the new consumer)
     * @param leaderId The id of the leader (which is this member)
     * @param protocol The protocol selected by the coordinator
     * @param allMemberMetadata Metadata from all members of the group
     * @return A map from each member to their state assignment
     */
    Map<String, ByteBuffer> performAssignment(String leaderId,
                                                 String protocol,
                                                 List<AssignmentMetadata> allMemberMetadata);

    /**
     * Invoked when a group member has successfully joined a group.
     *
     * @param generation The generation that was joined
     * @param memberId The identifier for the local member in the group
     * @param protocol The protocol selected by the coordinator
     * @param memberAssignment The assignment propagated from the group leader
     */
    void onJoinComplete(int generation,
                         String memberId,
                         String protocol,
                         ByteBuffer memberAssignment);

    /**
     * Invoked prior to each leave group event. This is typically used to clean up assigned partitions;
     * note it is triggered by the consumer's API caller thread (i.e. background heartbeat thread would
     * not trigger it even if it tries to force leaving group upon heartbeat session expiration)
     */
    default void onLeavePrepare() {}

    /**
     * Wrapper for {@link JoinGroupRequestProtocol} protocol class.
     */
    class JoinGroupMetadata {

        private final JoinGroupRequestProtocol delegate;

        public JoinGroupMetadata() {
            this.delegate = new JoinGroupRequestProtocol();
        }

        public JoinGroupMetadata(String name, byte[] metadata) {
            this();
            this.delegate.setName(name).setMetadata(metadata);
        }

        public String name() {
            return delegate.name();
        }

        public ByteBuffer metadata() {
            return ByteBuffer.wrap(delegate.metadata());
        }

        public JoinGroupMetadata setName(String name) {
            delegate.setName(name);
            return this;
        }

        public JoinGroupMetadata setMetadata(byte[] v) {
            delegate.setMetadata(v);
            return this;
        }

        @Override
        public boolean equals(Object o) {
            if (this == o) {
                return true;
            }
            if (o == null || getClass() != o.getClass()) {
                return false;
            }
            JoinGroupMetadata that = (JoinGroupMetadata) o;
            return Objects.equals(delegate, that.delegate);
        }

        @Override
        public int hashCode() {
            return Objects.hash(delegate);
        }

        @Override
        public String toString() {
            return "JoinGroupMetadata{" +
                "delegate=" + delegate +
                '}';
        }
    }

    /**
     * Wrapper for {@link JoinGroupResponseMember} protocol class.
     */
    class AssignmentMetadata {

        private final JoinGroupResponseData.JoinGroupResponseMember delegate;

        public AssignmentMetadata() {
            this.delegate = new JoinGroupResponseMember();
        }

        public String memberId() {
            return delegate.memberId();
        }

        public String groupInstanceId() {
            return delegate.groupInstanceId();
        }

        public ByteBuffer metadata() {
            return ByteBuffer.wrap(delegate.metadata());
        }

        public AssignmentMetadata setMemberId(String memberId) {
            delegate.setMemberId(memberId);
            return this;
        }

        public AssignmentMetadata setGroupInstanceId(String groupInstanceId) {
            delegate.setGroupInstanceId(groupInstanceId);
            return this;
        }

        public AssignmentMetadata setMetadata(byte[] metadata) {
            delegate.setMetadata(metadata);
            return this;
        }

        @Override
        public boolean equals(Object o) {
            if (this == o) {
                return true;
            }
            if (o == null || getClass() != o.getClass()) {
                return false;
            }
            AssignmentMetadata that = (AssignmentMetadata) o;
            return Objects.equals(delegate, that.delegate);
        }

        @Override
        public int hashCode() {
            return delegate.hashCode();
        }

        @Override
        public String toString() {
            return "AssignmentMetadata{" +
                "delegate=" + delegate +
                '}';
        }
    }
}
```

It also moves several classes from the org.apache.kafka.consumer.clients.consumer.internals package to the org.apache.kafka.consumer.clients.consumer package, as detailed on the **Proposed Changes** section below.

Proposed Changes

For a list of the propose changes, please refer to the the pull-request: KIP-795 Make AbstractCoordinator part of the public API #11515

AbstractCoordinator

- Move the AbstractCoordinator class from the o.a.k.c.c.consumer.internals to the o.a.k.c.c.consumer package
- Extract its abstract methods into a new org.apache.kafka.consumer.clients.consumer.Coordinator interface
- Remove references to the protocol types from its public APIs
    - org.apache.kafka.common.message.JoinGroupRequestData.JoinGroupRequestProtocolCollection is replaced by a java.util.Map
    - org.apache.kafka.common.message.JoinGroupResponseData.JoinGroupResponseMember is wrapped into a new **AssignmentMetadata** type
- The visibility of some methods have been raised (from default to protected), so classes extending AbstractCoordinator (e.g. ConsumerCoordinator) can make use of them.

Other classes made public

A few other classes are also moved from org.apache.kafka.clients.consumer.internals package to org.apache.kafka.clients.consumer:

- ConsumerNetworkClient
- Heartbeat
- RequestFuture / RequestFutureAdapter / RequestFutureListener

Other changes

Due to the removal of references to the protocol type classes from AbstractCoordinator, some adjustments needed to be made on the ConsumerCoordinator and Kafka Connect classes (ConnectAssignor, EagerAssignor, IncrementalCooperativeAssignor and WorkerCoordinator) to support the new interface.

These classes are all internal to Kafka, so clients should not be impacted at all.

Changes not included on this KIP changes

This KIP **does not** include any changes to the Admin APIs. Potential changes to the **Admin/KafkaAdminClient** classes (such as adding methods to query for group metadata from brokers) will be addressed in a separate KIP.

Compatibility, Deprecation, and Migration Plan

- As all members of the new interface are public, clients who extend the existing **AbstractCoordinator** class will potentially have to change the visibility of the overridden abstract methods.
- Also, as classes are being relocated, the package names of clients extending **AbstractCoordinator** will have to change accordingly.
- No functional changes are planned as part of this KIP, so these changes won't impact the vast majority of clients.

Rejected Alternatives

The obvious alternative is to keep **AbstractCoordinator** as an internal API. Clients who use these APIs might break in future versions of the library, and this is a risk that might not be and acceptable for many users.