KIP-159: Introducing Rich functions to Streams

Status

Current state: *"Under Discussion"*

Discussion thread: here

JIRA: ⚠ Unable to render Jira issues macro, execution error.

⚠ Unable to render Jira issues macro, execution error.

⚠ Unable to render Jira issues macro, execution error.

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

Motivation

This KIP combines KIP-149 and provides a hybrid solution to rich functions in Streams and accessing read-only keys within ValueJoiner, ValueTransformer, ValueMapper interfaces.

Rich functions are one of the essential parts of stream processing. There are several use-cases where users cannot express their business logic with current un-rich methods. For example:

- having access to RecordContext within an operator
- having access to a read-only key for ValueJoiner, ValueTransformer, ValueMapper interfaces

Rich Interfaces

```
public interface RichInitializer<V, K> {
    V apply(K key);
}

public interface RichValueMapper<V, VR, K> {
    VR apply(final V value, final K key, final RecordContext recordContext);
}

public interface RichValueJoiner<V1, V2, VR, K> {
    VR apply(final V1 value1, final V2 value2, final K key, final RecordContext recordContext);
}

public interface RichKeyValueMapper<K, V, VR> {
    VR apply(final K key, final V value, final RecordContext recordContext);
}

public interface RichReducer<V, K> {
    V apply(final V value1, final V value2, final K key, final RecordContext recordContext);
}

public interface RichAggregator<K, V, VA> {
    VA apply(final K key, final V value, final VA aggregate, final RecordContext recordContext);
}

public interface RichForeachAction<K, V> {
    void apply(final K key, final V value, final RecordContext recordContext);
}

public interface RichPredicate<K, V> {
    boolean test(final K key, final V value, final RecordContext recordContext);
}


public interface RichMerger<K, V> {
        V apply(final K aggKey, final V aggOne, final V aggTwo, final RecordContext recordContext);
}



public interface RichValueTransformer<V, VR, K> {
        void init(final ProcessorContext context);

        VR transform(final V value, final K key);

        void close();
}



public interface RichValueTransformerSupplier<V, VR, K> {

    RichValueTransformer<V, VR, K> get();
}
```

Public Interfaces

KStream

```
KStream<K, V> filter(RichPredicate<? super K, ? super V> predicate);
KStream<K, V> filterNot(RichPredicate<? super K, ? super V> predicate);
<KR> KStream<KR, V> selectKey(RichKeyValueMapper<? super K, ? super V, ? extends KR> mapper);
<KR, VR> KStream<KR, VR> map(RichKeyValueMapper<? super K, ? super V, ? extends KeyValue<? extends KR, ?
extends VR>> mapper);
<VR> KStream<K, VR> mapValues(RichValueMapper<? super V, ? extends VR, ? super K> mapper);
<KR, VR> KStream<KR, VR> flatMap(final RichKeyValueMapper<? super K, ? super V, ? extends Iterable<? extends
KeyValue<? extends KR, ? extends VR>>> mapper);

<VR> KStream<K, VR> flatMapValues(final RichValueMapper<? super V, ? extends Iterable<? extends VR>, ? super K>
mapper);



void foreach(final RichForeachAction<? super K, ? super V> action);
KStream<K, V> peek(final RichForeachAction<? super K, ? super V> action);
KStream<K, V>[] branch(final RichPredicate<? super K, ? super V>... predicates);

<VR> KStream<K, VR> transformValues(final RichValueTransformerSupplier<? super V, ? extends VR, ? super K>
valueTransformerSupplier,
                                    final String... stateStoreNames);
<KR> KGroupedStream<KR, V> groupBy(final RichKeyValueMapper<? super K, ? super V, KR> selector);
<KR> KGroupedStream<KR, V> groupBy(final RichKeyValueMapper<? super K, ? super V, KR> selector,
                                   final Serialized<KR, V> serialized);


<VO, VR> KStream<K, VR> join(final KStream<K, VO> otherStream,
                             final RichValueJoiner<? super V, ? super VO, ? extends VR, ? super K> joiner,
                             final JoinWindows windows);
<VO, VR> KStream<K, VR> join(final KStream<K, VO> otherStream,
                             final RichValueJoiner<? super V, ? super VO, ? extends VR, ? super K> joiner,
                             final JoinWindows windows,
                             final Joined<K, V, VO> joined);


<VO, VR> KStream<K, VR> leftJoin(final KStream<K, VO> otherStream,
                                 final RichValueJoiner<? super V, ? super VO, ? extends VR, ? super K> joiner,
                                 final JoinWindows windows);
<VO, VR> KStream<K, VR> leftJoin(final KStream<K, VO> otherStream,
                                 final RichValueJoiner<? super V, ? super VO, ? extends VR, ? super K> joiner,
                                 final JoinWindows windows,
                                 final Joined<K, V, VO> joined);


<VO, VR> KStream<K, VR> outerJoin(final KStream<K, VO> otherStream,
                                  final RichValueJoiner<? super V, ? super VO, ? extends VR, ? super K> joiner,
                                  final JoinWindows windows);
<VO, VR> KStream<K, VR> outerJoin(final KStream<K, VO> otherStream,
                                  final RichValueJoiner<? super V, ? super VO, ? extends VR, ? super K> joiner,
                                  final JoinWindows windows,
                                  final Joined<K, V, VO> joined);


<VT, VR> KStream<K, VR> join(final KTable<K, VT> table,
                             final RichValueJoiner<? super K, ? super V, ? super VT, ? extends VR> joiner);
<VT, VR> KStream<K, VR> join(final KTable<K, VT> table,
                             final RichValueJoiner<? super K, ? super V, ? super VT, ? extends VR> joiner,
                             final Joined<K, V, VT> joined);


<VT, VR> KStream<K, VR> leftJoin(final KTable<K, VT> table,
                                 final RichValueJoiner<? super K, ? super V, ? super VT, ? extends VR> joiner);
<VT, VR> KStream<K, VR> leftJoin(final KTable<K, VT> table,
                                 final RichValueJoiner<? super K, ? super V, ? super VT, ? extends VR> joiner,
```

```
                                final Joined<K, V, VT> joined);

<GK, GV, RV> KStream<K, RV> join(final GlobalKTable<GK, GV> globalKTable,
                                final RichKeyValueMapper<? super K, ? super V, ? extends GK> keyValueMapper,
                                final RichValueJoiner<? super K, ? super V, ? super GV, ? extends RV> joiner);

<GK, GV, RV> KStream<K, RV> leftJoin(final GlobalKTable<GK, GV> globalKTable,
                                     final RichKeyValueMapper<? super K, ? super V, ? extends GK>
keyValueMapper,
                                     final RichValueJoiner<? super K, ? super V, ? super GV, ? extends RV>
valueJoiner);
```

KTable

```
KTable<K, V> filter(final RichPredicate<? super K, ? super V> predicate);
KTable<K, V> filter(final RichPredicate<? super K, ? super V> predicate,
                    final Materialized<K, V, KeyValueStore<Bytes, byte[]>> materialized);
KTable<K, V> filterNot(final RichPredicate<? super K, ? super V> predicate);
KTable<K, V> filterNot(final RichPredicate<? super K, ? super V> predicate,
                       final Materialized<K, V, KeyValueStore<Bytes, byte[]>> materialized);

<VR> KTable<K, VR> mapValues(final RichValueMapper<? super V, ? extends VR, ? super K> mapper);
<VR> KTable<K, VR> mapValues(final RichValueMapper<? super V, ? extends VR, ? super K> mapper,
                             final Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized);

<KR> KStream<KR, V> toStream(final RichKeyValueMapper<? super K, ? super V, ? extends KR> mapper);


<KR, VR> KGroupedTable<KR, VR> groupBy(final RichKeyValueMapper<? super K, ? super V, KeyValue<KR, VR>>
selector);
<KR, VR> KGroupedTable<KR, VR> groupBy(final RichKeyValueMapper<? super K, ? super V, KeyValue<KR, VR>>
selector,
                                       final Serialized<KR, VR> serialized);


<VO, VR> KTable<K, VR> join(final KTable<K, VO> other,
                            final RichValueJoiner<? super V, ? super VO, ? extends VR, ? super K> joiner);
<VO, VR> KTable<K, VR> join(final KTable<K, VO> other,
                            final RichValueJoiner<? super V, ? super VO, ? extends VR, ? super K> joiner,
                            final Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized);


<VO, VR> KTable<K, VR> leftJoin(final KTable<K, VO> other,
                                final RichValueJoiner<? super V, ? super VO, ? extends VR, ? super K> joiner);
<VO, VR> KTable<K, VR> leftJoin(final KTable<K, VO> other,
                                final ValueJoiner<? super K, ? super V, ? super VO, ? extends VR> joiner,
                                final Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized);


<VO, VR> KTable<K, VR> outerJoin(final KTable<K, VO> other,
                                 final RichValueJoiner<? super V, ? super VO, ? extends VR, ? super K> joiner);
<VO, VR> KTable<K, VR> outerJoin(final KTable<K, VO> other,
                                 final RichValueJoiner<? super V, ? super VO, ? extends VR, ? super K> joiner,
                                 final Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized);
```

KGroupedStream

```
KTable<K, V> reduce(final RichReducer<V, K> reducer);

KTable<K, V> reduce(final RichReducer<V, K> reducer,
                    final Materialized<K, V, KeyValueStore<Bytes, byte[]>> materialized);

<VR> KTable<K, VR> aggregate(final RichInitializer<VR, K> initializer,
                             final RichAggregator<? super K, ? super V, VR> aggregator,
                             final Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized);

<VR> KTable<K, VR> aggregate(final RichInitializer<VR, K> initializer,
                             final RichAggregator<? super K, ? super V, VR> aggregator);
```

SessionWindowedKStream

There are 3 rich interfaces in aggregate() methods. So converting all possible combinations to their rich counterparts can cause a lot of overloads. So, I propose to overload one method with all rich interfaces.

```
<T> KTable<Windowed<K>, T> aggregate(final RichInitializer<T, Windowed<K>> initializer,
                                     final RichAggregator<? super K, ? super V, T> aggregator,
                                     final RichMerger<? super K, T> sessionMerger);
<VR> KTable<Windowed<K>, VR> aggregate(final RichInitializer<VR, Windowed<K>> initializer,
                                       final RichAggregator<? super K, ? super V, VR> aggregator,
                                       final RichMerger<? super K, VR> sessionMerger,
                                       final Materialized<K, VR, SessionStore<Bytes, byte[]>> materialized);

KTable<Windowed<K>, V> reduce(final RichReducer<V, K> reducer);
KTable<Windowed<K>, V> reduce(final RichReducer<V, K> reducer,
                              final Materialized<K, V, SessionStore<Bytes, byte[]>> materializedAs);


,
```

TimeWindowedKStream

```
<VR> KTable<Windowed<K>, VR> aggregate(final RichInitializer<VR, K> initializer,
                                       final RichAggregator<? super K, ? super V, VR> aggregator);
<VR> KTable<Windowed<K>, VR> aggregate(final RichInitializer<VR, K> initializer,
                                       final RichAggregator<? super K, ? super V, VR> aggregator,
                                       final Materialized<K, VR, WindowStore<Bytes, byte[]>> materialized);

KTable<Windowed<K>, V> reduce(final RichReducer<V, K> reducer);
KTable<Windowed<K>, V> reduce(final RichReducer<V, K> reducer,
                              final Materialized<K, V, WindowStore<Bytes, byte[]>> materialized);
```

KGroupedTable

```
KTable<K, V> reduce(final RichReducer<V, K> adder,
                    final RichReducer<V, K> subtractor,
                    final Materialized<K, V, KeyValueStore<Bytes, byte[]>> materialized);

KTable<K, V> reduce(final RichReducer<V, K> adder,
                    final RichReducer<V, K> subtractor);

<VR> KTable<K, VR> aggregate(final RichInitializer<VR> initializer,
                             final RichAggregator<? super K, ? super V, VR> adder,
                             final RichAggregator<? super K, ? super V, VR> subtractor,
                             final Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized);
<VR> KTable<K, VR> aggregate(final RichInitializer<VR> initializer,
                             final RichAggregator<? super K, ? super V, VR> adder,
                             final RichAggregator<? super K, ? super V, VR> subtractor);
```

Proposed changes

- Move `RecordContext` from `.processor.internals` to `.processor`

- Make record context open to public

Currently we set record context through InternalProcessorContext (`StreamTask.updateProcessorContext()`) :

```
// the below code snippet already exists, this is just for background.
private void updateProcessorContext(final StampedRecord record, final ProcessorNode currNode) {
    processorContext.setRecordContext(new ProcessorRecordContext(record.timestamp, record.offset(), record.
partition(), record.topic()));
    processorContext.setCurrentNode(currNode);
}
```

Sample processor should look like this:

```
class KStreamKTableJoinProcessor<K1, K2, V1, V2, R> extends AbstractProcessor<K1, V1> {

    ...
    private RecordContext recordContext;       // this line is added in this KIP

    ...

    @Override
        public void process(final K1 key, final V1 value) {

            recordContext = new RecordContext() {              // recordContext initialization is added in
this KIP

                    @Override
                    public long offset() {
                        return context().recordContext().offset();
                    }

                    @Override
                    public long timestamp() {
                        return context().recordContext().timestamp();
                    }

                    @Override
                    public String topic() {
                        return context().recordContext().topic();
                    }

                    @Override
                    public int partition() {
                        return context().recordContext().partition();
                    }
      };



    if (key != null && value != null) {
        final V2 value2 = valueGetter.get(keyMapper.apply(key, value));
        if (leftJoin || value2 != null) {
            context().forward(key, joiner.apply(value, value2, recordContext));
        }
    }
}


}
```

Rejected Alternatives

Not yet.