**KStream Branch Function**

**Kafka Streams Transformations** provide the ability to perform actions on Kafka Streams such as filtering and updating values in the stream.  Kafka Stream’s transformations contain operations such as `filter`, `map`, `flatMap`, etc. and have similarities to functional combinators found in languages such as Scala.

Kafka Stream Transformations are available from `KTable` or `KStream` and will result in one or more `KTable`, `KStream` or `KGroupedTable` depending on the transformation function.

Here, we are going to discuss **Branch Function** in Kstream.

**Branch Function**

The branch function is used to split a KStream based on the supplied predicates into one or more KStream instances.

Predicates

One or more Kstream Instances

**KStream<K,V>[] branch​(Predicate<? super K,? super V>... predicates)**

Creates an array of KStream from this stream by branching the records in the original stream based on the supplied predicates. Each record is evaluated against the supplied predicates, and predicates are evaluated in order. Each stream in the result array corresponds position-wise (index) to the predicate in the supplied predicates. The branching happens on first-match: A record in the original stream is assigned to the corresponding result stream for the first predicate that evaluates to true, and is assigned to this stream only. A record will be dropped if none of the predicates evaluate to true. This is a stateless record-by-record operation.

Parameters:

predicates - the ordered list of Predicate instances

Returns:

multiple distinct substreams of this KStream

**Example**

In this Kafka Streams Transformations, the `branch` example had three predicates: two filters for key name and one default predicate for everything else.

This is the example implementation

val results: Array[KStream[String, String]] = inputStream.branch(  
 (key, value) => key.startsWith(keyFilter1),  
 (key, value) => key.startsWith(keyFilter2),  
 (key, value) => true )

The expected results for filters on “sensor-1” and “sensor-2” and a default.

storeOne.get("sensor-1") shouldBe "MN"  
storeOne.get("sensor-11") shouldBe "IL"  
storeTwo.get("sensor-2") shouldBe "WI"

Predicates are evaluated in order. A record is placed to one and only one output stream on the first match: if the n-th predicate evaluates to true, the record is placed to n-th stream. If no predicate matches, the the record is dropped.

Branching is useful, for example, to route records to different downstream topics.

KStream<String, Long> stream = ...;  
KStream<String, Long>[] branches = stream.branch(  
 (key, value) -> key.startsWith("A"), /\* first predicate \*/  
 (key, value) -> key.startsWith("B"), /\* second predicate \*/  
 (key, value) -> true /\* third predicate \*/ );

// KStream branches[0] contains all records whose keys start with "A"

// KStream branches[1] contains all records whose keys start with "B"

// KStream branches[2] contains all other records

// Java 7 example: cf. `filter` for how to create `Predicate` instances