KIP-147: Add missing type parameters to StateStoreSupplier factories and KGroupedStream/Table methods

Status

Current state: *Discarded (covered by KIP-182)*

Discussion thread: *here*

JIRA: TBD

PR: here

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

Note: This KIP may become obsolete as the discussion on "Streams DSL/StateStore Refactoring" supersedes it.

Motivation

The recommended practice to create a StateStoreSupplier is as per the following example:

```
StateStoreSupplier countStore = Stores.create("Counts")
    .withKeys(Serdes.String())
    .withValues(Serdes.Long())
    .persistent()
    .build();
```

However, StateStoreSupplier is a generic interface that takes the StateStore type as a parameter:

```
public interface StateStoreSupplier<T extends StateStore>
```

In the above example that type parameter is lost as the build() method returns a raw type.

As StateStoreSupplier is passed to count/reduce/aggregate etc. methods on KGroupedStream or KGroupedTable, the compiler cannot detect if a supplier for the wrong kind of store is provided.

The other parameters to those methods, such as Serdes, Reducers, etc are type-parameterised by the key and value types allowing compile-time type checks.

The StateStoreSupplier argument stands out as a raw type. Making it type-parameterised will help detect at compile time errors such as when someone refactors their app to use a different typo of aggregations (e.g. TimeWindowed vs SessionWindowed) and forgets to change the StateStoreSupplier passed in.

Public Interfaces

- **KGroupedStream** - deprecate the following methods:
    - KTable<K, Long> count(final StateStoreSupplier<KeyValueStore> storeSupplier);
    - <W extends Window> KTable<Windowed<K>, Long> count(final Windows<W> windows,
      final StateStoreSupplier<WindowStore> storeSupplier);
    - KTable<Windowed<K>, Long> count(final SessionWindows sessionWindows,
      final StateStoreSupplier<SessionStore> storeSupplier);

- KTable<K, V> reduce(final Reducer<V> reducer,
  final StateStoreSupplier<KeyValueStore> storeSupplier);
- <W extends Window> KTable<Windowed<K>, V> reduce(final Reducer<V> reducer,
  final Windows<W> windows,
  final StateStoreSupplier<WindowStore> storeSupplier);
- KTable<Windowed<K>, V> reduce(final Reducer<V> reducer,
  final SessionWindows sessionWindows,
  final StateStoreSupplier<SessionStore> storeSupplier);
- <VR> KTable<K, VR> aggregate(final Initializer<VR> initializer,
  final Aggregator<? super K, ? super V, VR> aggregator,
  final StateStoreSupplier<KeyValueStore> storeSupplier);
- <W extends Window, VR> KTable<Windowed<K>, VR> aggregate(final Initializer<VR> initializer,
  final Aggregator<? super K, ? super V, VR> aggregator,
  final Windows<W> windows,
  final StateStoreSupplier<WindowStore> storeSupplier);
- <T> KTable<Windowed<K>, T> aggregate(final Initializer<T> initializer,
  final Aggregator<? super K, ? super V, T> aggregator,
  final Merger<? super K, T> sessionMerger,
  final SessionWindows sessionWindows,
  final Serde<T> aggValueSerde,
  final StateStoreSupplier<SessionStore> storeSupplier);

- **KGroupedStream** - add the following replacement methods:
  - KTable<K, Long> count(final **Typed**StateStoreSupplier<KeyValueStore**<K, Long>**> storeSupplier);
  - <W extends Window> KTable<Windowed<K>, Long> count(final Windows<W> windows,
    final **Typed**StateStoreSupplier<WindowStore**<K, Long>**> storeSupplier);
  - KTable<Windowed<K>, Long> count(final SessionWindows sessionWindows,
    final **Typed**StateStoreSupplier<SessionStore**<K, Long>**> storeSupplier);
  - KTable<K, V> reduce(final Reducer<V> reducer,
    final **Typed**StateStoreSupplier<KeyValueStore**<K ,V>**> storeSupplier);
  - <W extends Window> KTable<Windowed<K>, V> reduce(final Reducer<V> reducer,
    final Windows<W> windows,
    final **Typed**StateStoreSupplier<WindowStore**<K, V>**> storeSupplier);
  - KTable<Windowed<K>, V> reduce(final Reducer<V> reducer,
    final SessionWindows sessionWindows,
    final **Typed**StateStoreSupplier<SessionStore**<K, V>**> storeSupplier);
  - <VR> KTable<K, VR> aggregate(final Initializer<VR> initializer,
    final Aggregator<? super K, ? super V, VR> aggregator,
    final **Typed**StateStoreSupplier<KeyValueStore**<K, VR>**> storeSupplier);
  - <W extends Window, VR> KTable<Windowed<K>, VR> aggregate(final Initializer<VR> initializer,
    final Aggregator<? super K, ? super V, VR> aggregator,
    final Windows<W> windows,
    final **Typed**StateStoreSupplier<WindowStore**<K, VR>**>
    storeSupplier);
  - <T> KTable<Windowed<K>, T> aggregate(final Initializer<T> initializer,
    final Aggregator<? super K, ? super V, T> aggregator,
    final Merger<? super K, T> sessionMerger,
    final SessionWindows sessionWindows,
    final Serde<T> aggValueSerde,
    final **Typed**StateStoreSupplier<SessionStore**<K, T>**> storeSupplier);
- **KGroupedTable** - deprecate the following methods:
  - KTable<K, Long> count(final StateStoreSupplier<KeyValueStore> storeSupplier);
  - KTable<K, V> reduce(final Reducer<V> adder,
    final Reducer<V> subtractor,
    final StateStoreSupplier<KeyValueStore> storeSupplier);
  - <VR> KTable<K, VR> aggregate(final Initializer<VR> initializer,
    final Aggregator<? super K, ? super V, VR> adder,
    final Aggregator<? super K, ? super V, VR> subtractor,
    final StateStoreSupplier<KeyValueStore> storeSupplier);

- **KGroupedTable** - add the following replacement methods:
  - KTable<K, Long> count(final **Typed**StateStoreSupplier<KeyValueStore**<K, Long>**> storeSupplier);
  - KTable<K, V> reduce(final Reducer<V> adder,
    final Reducer<V> subtractor,
    final **Typed**StateStoreSupplier<KeyValueStore**<K, V>**> storeSupplier);
  - <VR> KTable<K, VR> aggregate(final Initializer<VR> initializer,
    final Aggregator<? super K, ? super V, VR> adder,
    final Aggregator<? super K, ? super V, VR> subtractor,
    final **Typed**StateStoreSupplier<KeyValueStore**<K, VR>**> storeSupplier);
- **KTable** - deprecate the following methods:
  - KTable<K, V> filter(final Predicate<? super K, ? super V> predicate, final StateStoreSupplier<KeyValueStore> storeSupplier);
  - KTable<K, V> filterNot(final Predicate<? super K, ? super V> predicate, final StateStoreSupplier<KeyValueStore> storeSupplier);
  - <VR> KTable<K, VR> mapValues(final ValueMapper<? super V, ? extends VR> mapper,
    final Serde<VR> valueSerde,
    final StateStoreSupplier<KeyValueStore> storeSupplier);
  - KTable<K, V> through(final String topic,
    final StateStoreSupplier<KeyValueStore> storeSupplier);
  - KTable<K, V> through(final StreamPartitioner<? super K, ? super V> partitioner,
    final String topic,
    final StateStoreSupplier<KeyValueStore> storeSupplier);
  - KTable<K, V> through(final Serde<K> keySerde, Serde<V> valSerde,
    final String topic,
    final StateStoreSupplier<KeyValueStore> storeSupplier);
  - KTable<K, V> through(final Serde<K> keySerde,
    final Serde<V> valSerde,
    final StreamPartitioner<? super K, ? super V> partitioner,
    final String topic,
    final StateStoreSupplier<KeyValueStore> storeSupplier);
  - <VO, VR> KTable<K, VR> join(final KTable<K, VO> other,
    final ValueJoiner<? super V, ? super VO, ? extends VR> joiner,
    final StateStoreSupplier<KeyValueStore> storeSupplier);
  - <VO, VR> KTable<K, VR> leftJoin(final KTable<K, VO> other,
    final ValueJoiner<? super V, ? super VO, ? extends VR> joiner,
    final StateStoreSupplier<KeyValueStore> storeSupplier);
  - <VO, VR> KTable<K, VR> outerJoin(final KTable<K, VO> other,
    final ValueJoiner<? super V, ? super VO, ? extends VR> joiner,
    final StateStoreSupplier<KeyValueStore> storeSupplier);
- **KTable** - add the following replacement methods:
  - KTable<K, V> filter(final Predicate<? super K, ? super V> predicate, final **Typed**StateStoreSupplier<KeyValueStore**<K, V>**> storeSupplier);
  - KTable<K, V> filterNot(final Predicate<? super K, ? super V> predicate, final **Typed**StateStoreSupplier<KeyValueStore**<K, V>**> storeSupplier);
  - <VR> KTable<K, VR> mapValues(final ValueMapper<? super V, ? extends VR> mapper,
    final Serde<VR> valueSerde,
    final **Typed**StateStoreSupplier<KeyValueStore**<K, VR>**> storeSupplier);
  - KTable<K, V> through(final String topic,
    final **Typed**StateStoreSupplier<KeyValueStore**<K, V>**> storeSupplier);
  - KTable<K, V> through(final StreamPartitioner<? super K, ? super V> partitioner,
    final String topic,
    final **Typed**StateStoreSupplier<KeyValueStore**<K, V>**> storeSupplier);
  - KTable<K, V> through(final Serde<K> keySerde, Serde<V> valSerde,
    final String topic,
    final **Typed**StateStoreSupplier<KeyValueStore**<K, V>**> storeSupplier);
  - KTable<K, V> through(final Serde<K> keySerde,
    final Serde<V> valSerde,
    final StreamPartitioner<? super K, ? super V> partitioner,
    final String topic,
    final **Typed**StateStoreSupplier<KeyValueStore**<K, V>**> storeSupplier);
  - <VO, VR> KTable<K, VR> join(final KTable<K, VO> other,
    final ValueJoiner<? super V, ? super VO, ? extends VR> joiner,
    final **Typed**StateStoreSupplier<KeyValueStore**<K, VR>**> storeSupplier);
  - <VO, VR> KTable<K, VR> leftJoin(final KTable<K, VO> other,
    final ValueJoiner<? super V, ? super VO, ? extends VR> joiner,
    final **Typed**StateStoreSupplier<KeyValueStore**<K, VR>**> storeSupplier);
  - <VO, VR> KTable<K, VR> outerJoin(final KTable<K, VO> other,
    final ValueJoiner<? super V, ? super VO, ? extends VR> joiner,
    final **Typed**StateStoreSupplier<KeyValueStore**<K, VR>**> storeSupplier);
- **KStreamBuilder** - deprecate the following method:
  - <K, V> GlobalKTable<K, V> globalTable(final Serde<K> keySerde,
    final Serde<V> valSerde,
    final String topic,
    final StateStoreSupplier<KeyValueStore> storeSupplier)
- **KStreamBuilder** - add the following replacement method:
  - <K, V> GlobalKTable<K, V> globalTable(final Serde<K> keySerde,
    final Serde<V> valSerde,
    final String topic,
    final **Typed**StateStoreSupplier<KeyValueStore**<K, V>**> storeSupplier)
- Deprecate **Stores** class
- Add a new replacement **TypedStores** class with the following extra public interfaces in addition to those equivalent to those in **Stores** class:

```
public interface PersistentWindowFactory<K, V> {


/**
* Caching should be enabled on the created store.
* @return the factory to create a persistent window store
*/
PersistentWindowFactory<K, V> enableCaching();

/**
* Indicates that a changelog should not be created for the key-value store
* @return the factory to create a persistent window store
*/
PersistentWindowFactory<K, V> disableLogging();

/**
* Indicates that a changelog should be created for the store. The changelog will be created
* with the provided cleanupPolicy and configs.
*
* Note: Any unrecognized configs will be ignored.
* @param config any configs that should be applied to the changelog
* @return the factory to create a persistent window store
*/
PersistentWindowFactory<K, V> enableLogging(final Map<String, String> config);


/**
* Return the instance of StateStoreSupplier of new window store.
* @return the key-value store; never null
*/
TypedStateStoreSupplier<WindowStore<K, V>> build();
}
```

```
public interface PersistentSessionFactory<K, V> {
/**
* Indicates that a changelog should be created for the store. The changelog will be created
* with the provided cleanupPolicy and configs.
*
* Note: Any unrecognized configs will be ignored.
* @param config any configs that should be applied to the changelog
* @return the factory to create a persistent key-value store
*/
PersistentSessionFactory<K, V> enableLogging(final Map<String, String> config);
/**
* Indicates that a changelog should not be created for the key-value store
* @return the factory to create a persistent session store
*/
PersistentSessionFactory<K, V> disableLogging();
/**
* Caching should be enabled on the created store.
* @return the factory to create a persistent session store
*/
PersistentSessionFactory<K, V> enableCaching();
/**
* Return the instance of StateStoreSupplier of new session store.
* @return the key-value store; never null
*/
TypedStateStoreSupplier<SessionStore<K, V>> build();

}
```

- differences in PersistentKeyValueFactory in **TypedStores** versus that in **Stores**:
  - **PersistentWindowFactory**<K, V> windowed(final long windowSize, long retentionPeriod, int numSegments, boolean retainDuplicates);
  - **PersistentSessionFactory**<K, V> sessionWindowed(final long retentionPeriod);
  - **Typed**StateStoreSupplier**<KeyValueStore<K, V>>** build();
- differences in InMemoryKeyValueFactory in **TypedStores** versus that in **Stores**:

- **Typed**StateStoreSupplier**<KeyValueStore<K, V>>** build();

Proposed Changes

Pull Request to demonstrate the changes: **https://github.com/apache/kafka/pull/2992/files**

The new usage would be e.g.:

```
TypedStateStoreSupplier<KeyValueStore<String, Long>> countStore = TypedStores.create("Counts")
    .withKeys(Serdes.String())
    .withValues(Serdes.Long())
    .persistent()
    .build();
```

```
TypedStateStoreSupplier<WindowStore<String, Long>> windowedStore = TypedStores.create("WindowedCounts")
    .withKeys(Serdes.String())
    .withValues(Serdes.Long())
    .persistent()
        .windowed(1000, 10000, 10, false)
    .build();
```

```
TypedStateStoreSupplier<SessionStore<String, Long>> sessionStore = TypedStores.create("SessionWindowedCounts")
    .withKeys(Serdes.String())
    .withValues(Serdes.Long())
    .persistent()
        .sessionWindowed(60000)
    .build();
```

Compatibility, Deprecation, and Migration Plan

Changes are intended to be backwards-compatible.

Test Plan

Only re-run of existing tests is envisaged at this time.

Rejected Alternatives

Add type parameters to current method parameters. This has been rejected as a backwards-incompatible change.

<K, V>