DataSet Programming API

Module Objectives

In this module, you will learn:

- The Continuuity Reactor Java `Table` API
- Reading, writing and deleting data from DataSets
- Scanning, incrementing and swapping data from DataSets

Continuuity Reactor Table API

The `Table` API provides basic methods to perform:

- **Read, Write and Delete** operations

plus special

- **Scan**
- **Atomic increment** and
- **Compare-and-swap** operations

Each basic operation has a **method that takes an operation-type object** as a parameter plus handy **methods for working directly with byte arrays**

If your application code already deals with byte arrays, you can use the latter methods to save a conversion

---

Table Read, Write and Delete

```
// Read
public Row get(Get get)
public Row get(byte[] row)
public byte[] get(byte[] row, byte[] column)
public Row get(byte[] row, byte[][] columns)
public Row get(byte[] row, byte[] startColumn,
               byte[] stopColumn, int limit)

// Write
public void put(Put put)
public void put(byte[] row, byte[] column, byte[] value)
public void put(byte[] row, byte[][] columns, byte[][] values)

// Delete
public void delete(Delete delete)
public void delete(byte[] row)
public void delete(byte[] row, byte[] column)
public void delete(byte[] row, byte[][] columns)
```

Table Scan, Increment, Compare and Swap

```
// Scan
public Scanner scan(byte[] startRow, byte[] stopRow)

// Increment
public Row increment(Increment increment)
public long increment(byte[] row, byte[] column, long amount)
public Row increment(byte[] row, byte[][] columns, long[] amounts)

// Compare and Swap
public boolean compareAndSwap(byte[] row, byte[] column,
                              byte[] expectedValue, byte[] newValue)
```

Table Read (1 of 3)

A `get` operation reads all columns or selection of columns of a single row:

```
Table t;
byte[] rowKey1;
byte[] columnX;
byte[] columnY;
int n;

// Read all columns of a row
Row row = t.get(new Get("rowKey1"));

// Read specified columns from a row
Row rowSelection = t.get(new Get("rowKey1").add("column1").add("column2"));

// Reads a column range from x (inclusive) to y (exclusive)
// with a limit of n return values
rowSelection = t.get(rowKey1, columnX, columnY; n);

// Read only one column in one row byte[]
value = t.get(rowKey1, columnX);
```

Table Read (2 of 3)

- `Row` object provides access to the Row data including its columns
- If only a selection of row columns is requested, the returned Row object will contain only those columns
- Row object provides an extensive API for accessing returned column values:

```
// Get column value as a byte array
byte[] value = row.get("column1");

// Get column value of a specific type
String valueAsString = row.getString("column1");
Integer valueAsInteger = row.getInt("column1");
```

Table Read (3 of 3)

- When requested, the value of a column is converted to a specific type automatically
- If the column is absent in a Row, the returned value is `null`
- To return primitive types, the corresponding methods accepts default value to be returned when the column is absent:

```
// Get column value as a primitive type or 0 if column is absent
long valueAsLong = row.getLong("column1", 0);
```

Table Write

A `put` operation writes data into a row:

```
// Write a set of columns with their values
t.put(new Put("rowKey1").add("column1", "value1").add("column2", 55L));
```

Table Delete

A `delete` operation removes an entire row or a subset of its columns:

```
// Delete the entire row
t.delete(new Delete("rowKey1"));

// Delete a selection of columns from the row
t.delete(new Delete("rowKey1").add("column1").add("column2"));
```

Specifying a set of columns helps to perform delete operation faster

When you want to delete all the columns of a row and you know all of them, passing all of them will make the deletion faster

---

Table Scan (1 of 2)

A `scan` operation fetches a subset of rows or all of the rows of a Table:

```
byte[] startRow;
byte[] stopRow;
Row row;

// Scan all rows from startRow (inclusive) to stopRow (exclusive)
Scanner scanner = t.scan(startRow, stopRow);
try {
  while ((row = scanner.next()) != null) {
    LOG.info("column1: " + row.getString("column1", "null"));
  }
} finally {
  scanner.close();
}
```

Table Scan (2 of 2)

To scan a set of rows not bounded by `startRow` and/or `stopRow` you can pass `null` as their value:

```
byte[] startRow;

// Scan all rows of a table
Scanner allRows = t.scan(null, null);

// Scan all columns up to stopRow (exclusive)
Scanner headRows = t.scan(null, stopRow);

// Scan all columns starting from startRow (inclusive)
Scanner tailRows = t.scan(startRow, null);
```

Table Increment

An increment operation increments a `long` value of one or more columns by either `1L` or an integer amount *n*

If a column doesn't exist, it is created with an assumed value before the increment of zero:

```
// Write long value to a column of a row
t.put(new Put("rowKey1").add("column1", 55L));

// Increment values of several columns in a row
t.increment(new Increment("rowKey1").add("column1", 1L).add("column2", 23L));
```

If the existing value of the column cannot be converted to a `long`, a `NumberFormatException` will be thrown

---

Table Compare and Swap

A swap operation compares the existing value of a column with an expected value, and if it matches, replaces it with a new value

The operation returns `true` if it succeeds and `false` otherwise:

```
byte[] expectedCurrentValue;
byte[] newValue;
if (!t.compareAndSwap(rowKey1, columnX,
      expectedCurrentValue, newValue)) {
  LOG.info("Current value was different from expected");
}
```

Module Summary

You should now be able to:

- Use the Continuuity Reactor Java `Table` API
- Perform reading, writing and deleting data from DataSets
- Understand Scanning, incrementing and swapping data from DataSets

---

Module Completed