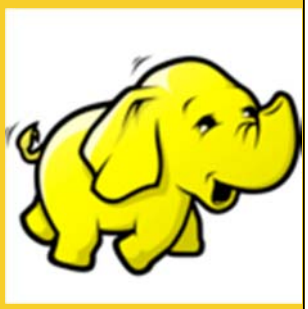
HBase Java Client API

Basic CRUD operations

Originals of Slides and Source Code for Examples:
http://www.coreservlets.com/hadoop-tutorial/

© 2012 coreservlets.com and Dima May



Agenda

- Create via Put method
- Read via Get method
- Update via Put method
- Delete via Delete method

Java Client API Overview

- HBase is written in Java
  - No surprise that it has a Java Native API
- Supports programmatic access to Data Manipulation Language (DML)
  - CRUD operations plus more
- Everything that you can do with HBase Shell and more....
- Java Native API is the fastest way to access HBase

Using Client API

1. **Create a Configuration object**
   - Recall Configuration from HDFS object
   - Adds HBase specific props
2. **Construct HTable**
   - Provide Configuration object
   - Provide table name
3. **Perform operations**
   - Such as put, get, scan, delete, etc...
4. **Close HTable instance**
   - Flushes all the internal buffers
   - Releases all the resources

Using Client API

1. **Create a Configuration object**

```
Configuration conf = HbaseConfiguration.create();
```

2. **Construct HTable**

```
HTable hTable = new HTable(conf, tableName);
```

3. **Perform operations**

```
hTable.getTableName();
```

4. **Close HTable instance**

```
hTable.close();
```

ConstructHTable.java

```
public class ConstructHTable {
  public static void main(String[] args) throws IOException {
    Configuration conf = HbaseConfiguration.create();

    HTable hTable = new HTable(conf, "-ROOT-");

    System.out.println("Table is: " +
                       Bytes.toString(hTable.getTableName()));

    hTable.close();
  }
}
```

Seeds configuration object with required information to establish client connection

Table name

Release all the resources

ConstructHTable.java Output

```
$ yarn jar $PLAY_AREA/HadoopSamples.jar hbase.ConstructHTable
12/01/15 13:22:03 INFO zookeeper.ZooKeeper: Client
environment:zookeeper.version=3.3.3-cdh3u2--1, built on
10/14/2011 03:25 GMT
...
...
...
12/01/15 13:22:03 INFO zookeeper.ClientCnxn: Session
establishment complete on server localhost/127.0.0.1:2181,
sessionid = 0x134e27760560013, negotiated timeout = 40000
Table is: -ROOT-
```

1: Create Configuration Object

- **Client Code Configuration**
- **HbaseConfiguration extends Hadoop's Configuration class**
  - Still fully compatible with Configuration
- **How did HbaseConfiguration.create() seed Configuration object?**
  - Loads hbase-default.xml and hbase-site.xml from Java CLASSPATH
    - hbase-default.xml is packaged inside HBase jar
    - hbase-site.xml will need to be added to the CLASSPATH
    - hbase-site.xml overrides properties in hbase-default.xml

1: Create Configuration Object

- **How did hbase-site.xml get on CLASSPATH?**
  - Recall that we executed the code via yarn script

```
$ yarn jar $PLAY_AREA/HadoopSamples.jar hbase.ConstructHTable
```

  - Hadoop's scripts are configured to put hbase's CLASSPATH onto it's CLASSPATH
  - Specified in <hadoop_install>/conf/hadoop-env.sh

```
export HADOOP_CLASSPATH=
       $HBASE_HOME/*:$HBASE_HOME/conf:$HADOOP_CLASSPATH
```

  - To check what's on Hadoop's CLASSPATH
    - $ yarn classpath
    - $ yarn classpath | grep hbase

1: Create Configuration Object

- **If you already have a Configuration it's easy to add HBase configuration**

```
Configuration newConf = Configuration.create(existingConf);
```

  - Provided configuration takes precedence over files loaded from CLASSPATH
    - hbase-default.xml and hbase-site.xml
  - Creates a new Configuration object and merges with the provided instance
- **You can manually override properties**

```
Configuration conf = HbaseConfiguration.create();
conf.set("hbase.zookeeper.quorum", "node1,node2");
```

  - Usually not necessary and not recommended

1: Create Configuration Object

- **Share Configuration instance as much as possible**
  - HTables created with the same Connection object will share the same underlying Connection
    - Connection to Zookeeper and HbaseMaster
    - Represented by HConnection class
    - Managed by HConnectionManager class
    - Internally connections are cached in a map that uses Configuration instances as a key
  - When re-using Configuration object for multiple HTable instances
    - Call HTable.close so HConnectionManager removes this particular instance from the list of HTables requiring Hconnection
  - When all HTables closed for a particular Connection object then HConnectionManager can close the connection
    - If close is not called then Connection will be open until the client process ends
      - Could lead to running out of connections and causing IOException

2: Construct HTable

- **org.apache.hadoop.hbase.client.HTable**
  - Client interface to a single HBase table
  - Exposes CRUD operations
  - Simple by design and easy to use :)
  - Operations that change data are atomic on per-row-basis
    - There is no built-in concept of a transaction for multiple rows or tables
    - 100% consistency per-row - a client will either write/read the entire row OR have to wait
    - Not a problem when having many readers for a given row but will cause contention when lots of writers attempt to write to the same exact row
    - Doesn't matter on the number of columns written per request, the request will be fully atomic

2: Construct HTable

- **Creating HTable instance is not free**
  - Actually quite costly – scans catalog .META. Table
    - Checks that table exists and enabled
  - Create once (per thread) and re-use for as long as possible
  - If you find yourself constructing many instances consider using HTablePool (utility to re-use multiple HTable instances)
- **HTable is NOT thread safe**
  - Create 1 instance per thread
- **HTable supports CRUD batch operations**
  - Not atomic
  - For performance and convenience

Using Client API Review

1. **Create a Configuration object**
   - Configuration conf = HbaseConfiguration.create();
2. **Construct HTable**
   - HTable hTable = new HTable(conf, tableName);
3. **Perform operations**
   - hTable.getTableName();
4. **Close HTable instance**
   - hTable.close();

Create/Save Data to HBase

1. **Construct HTable instance**
   - Create Put instance
2. **Add cell values and their coordinates**
   - Specify family:column as a coordinate
3. **Call put on HTable instance**
4. **Close HTable**

1: Construct HTable

- **Create Configuration**
- **Construct HTable**

```
Configuration conf = HBaseConfiguration.create();
HTable hTable = new HTable(conf, "HBaseSamples");
```


2: Create Put Instance

- **Put is a save operation for a single row**
- **Must provide a row id to the constructor**
  - Row id is raw bytes: can be anything like number or UUID
    - You are responsible for converting the id to bytes
    - HBase comes with a helper class Bytes that provides static methods which handles various conversions from and to bytes
      - org.apache.hadoop.hbase.util.Bytes

```
Put put1 = new Put(Bytes.toBytes("row1"));
```

  - Optionally can provide cell's timestamp and an instance of RowLock

```
Put put2 = new Put(Bytes.toBytes("row2"), timestamp);
Put put3 = new Put(Bytes.toBytes("row3"), rowLock);
Put put4 = new Put(Bytes.toBytes("row4"), timestamp, rowLock);
```

3: Add Cell Values and Their Coordinates

- **Add columns to save to Put instance**
  - Provide family:value coordinate and optional timestamp
  - Few options of the add methods
    - Put.add(family, column, value)
    - Put.add(family, column, timestamp, value)
    - Put.add(KeyValue kv)
  - Family, column, and value are raw binary
  - Client's responsibility to convert to binary format
  - KeyValue class as its internal cell's representation
    - For advanced usage, not usually required

```
put1.add(toBytes("test"), toBytes("col1"), toBytes("val1"));
put1.add(toBytes("test"), toBytes("col2"), toBytes("val2"));
```


4: Call Put on HTable Instance

- **Provide initialized Put object to HTable**
- **The operation is synchronous**

```
...
hTable.put(put1);
```

5. Close HTable

- **Release resource held by HTable**
- **Inform HConnectionManager that this instance won't be using connection**

```
hTable.close();
```

- **Utilize try/finally block**

```
HTable hTable = new HTable(conf, "HBaseSamples");
try {
      // to stuff with table
} finally {
      hTable.close();
}
```

- **Most examples emit try/finally constructs in favor of readability**

PutExample.java

Static import of Bytes class

```
import static org.apache.hadoop.hbase.util.Bytes.*;
public class PutExample {
  public static void main(String[] args) throws IOException {
     Configuration conf = HBaseConfiguration.create();
     HTable hTable = new HTable(conf, "HBaseSamples");

     Put put1 = new Put(toBytes("row1"));

     put1.add(toBytes("test"), toBytes("col1"), toBytes("val1"));
     put1.add(toBytes("test"), toBytes("col2"), toBytes("val2"));

     hTable.put(put1);

     hTable.close();
  }
}
```

Create put with id "row1"

Add "val1" to test:col1 column
Add "val2" to test:col2 column

Save row to HBase

PutExample.java Output

```
$ yarn jar $PLAY_AREA/HadoopSamples.jar hbase.PutExample
$ hbase shell
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.90.4-cdh3u2, r, Thu Oct 13 20:32:26 PDT 2011

hbase(main):001:0> get 'HBaseSamples', 'row1'
ROW        COLUMN+CELL
row1       column=test:col1, timestamp=1326663102473, value=val1
row1       column=test:col2, timestamp=1326663102473, value=val2
1 row(s) in 0.3340 seconds
```


Retrieving Data

- **API supports**
  - Get a single row by id
  - Get a set of rows by a set of row ids
    - Implemented via batching and will be covered later
  - Scan an entire table or a sub set of rows
    - To scan a portion of the table provide start and stop row ids
    - Recall that row-ids are ordered by raw byte comparison
    - In case of string based ids, the order is alphabetical
- **That's it**
  - Very limited simple API

Retrieve a Single Row

1. **Construct HTable instance**
2. **Create Get instance**
3. **Optionally narrow down result**
   - Specify family:column coordinate
   - Optionally add filters
4. **Request and get results**
   - Call get on HTable
   - Result instance is returned and will contain the data
5. **Close HTable**

2: Create Get Instance

- **Retrieve a single row**
- **Construct a Get Instance by providing row id**
  - Row id is in raw binary format
- **Optional parameter for a row lock**

```
Get get = new Get(toBytes("row1"));
```

3: Optionally Narrow Down Result

- **Only retrieve the data that you need**
  - If not specified then an entire row is retrieved
  - Important, as HBase allows you to scale to millions of rows
  - Can narrow down by family, column(s), time range and max versions
  - Can provide more than one narrow down criteria
  - Family and column name parameters are in raw bytes
- **Narrow down by family**
  - get.addFamily(family)
- **Narrow down by column**
  - get.addColumn(family, column)

3: Optionally Narrow Down Result

- **Narrow down by time range**
  - get.setTimeRange(minStamp, maxStamp)
- **Specify number of versions returned**
  - get.setMaxVersions(maxVersions)
  - By default set to 1: only returns the latest version
- **Can retrieve multiple families and columns**
  - get.addFamily(family)
  - get.addFamily(family1)
  - get.addColumn(family2, column1)
  - get.addColumn(family2, column2)
  - get.setTimeRange(minStamp, maxStamp)

4: Request and Get Results

- **Utilize get methods on HTable**
  - Provide assembled Get instance
  - Returns Result object with all the matching cells

```
Result result = hTable.get(get);
byte [] rowId = result.getRow();
byte [] val1 =
     result.getValue(toBytes("test"), toBytes("col1"));
byte [] val2 =
     result.getValue(toBytes("test"), toBytes("col2"));
```


4: Request and Get Results

- **Result class**
  - Allows you to access everything returned
  - Result is NOT Thread safe
- **Methods of interest**
  - Result.getRow() - get row's id
  - Result.getValue(family, column) - get a value for a chosen cell
  - Result.isEmpty() - true if the result is empty false otherwise
  - Result.size() - returns number of cells
  - Result.containsColumn(family:column) true if column exists
  - There are a number of methods that provide access to underlying KeyValue objects
    - are for advanced usage and usually not required

GetExample.java

```
public static void main(String[] args) throws IOException {
  Configuration conf = HBaseConfiguration.create();
  HTable hTable = new HTable(conf, "HBaseSamples");


  Get get = new Get(toBytes("row1"));
  Result result = hTable.get(get);
  print(result);


  get.addColumn(toBytes("test"), toBytes("col2"));
  result = hTable.get(get);
  print(result);


  hTable.close();
}
```

Get the entire row

Select a single column test:col2

GetExample.java

```
private static void print(Result result) {
  System.out.println("-------------------------------");
  System.out.println("RowId: " + Bytes.toString(result.getRow()));


  byte [] val1 = result.getValue(toBytes("test"), toBytes("col1"));
  System.out.println("test1:col1="+Bytes.toString(val1));


  byte [] val2 = result.getValue(toBytes("test"), toBytes("col2"));
  System.out.println("test1:col2="+Bytes.toString(val2));
}
```

Retrieve row id

Print value test:col1 column

Print value test:col2 column

GetExample.java Output

```
$ yarn jar $PLAY_AREA/HadoopSamples.jar hbase.GetExample
...
...
-----------------------------------
RowId: row1
test1:col1=val1
test1:col2=val2
-----------------------------------
RowId: row1
test1:col1=null
test1:col2=val2
```

test1:col1 wasn't selected the second time

Deleting Data

- **Deletes are per-row-basis**
- **Supports batching**
  - Batching is not atomic, for performance and for convenience
  - More on that later..

Deleting Data

1. **Construct HTable instance**
2. **Create and Initialize Delete**
3. **Call delete on HTable**
   - htable.delete(delete);
4. **Close HTable**

** We are already familiar with HTable usage, and #3 is too elementary so lets focus on step #2

2: Create and Initialize Delete

- **Construct a Delete instance**
  - Similar to Get or Put
  - Delete(byte[] row)
    - Provide a row id to delete/modify
  - Delete(byte[] row, long timestamp, RowLock rowLock)
    - Optional timestamp and RowLock
- **Optionally narrow down the Deletes**

```
Delete delete1 = new Delete(toBytes("anotherRow"));
delete1.deleteColumns(toBytes("family"), toBytes("loan"));
delete1.deleteFamily(toBytes("family"));
```

2: Create and Initialize Delete

- **Narrow down what to delete for a row**
  - If nothing provided then entire row is deleted
  - Delete a subset of a row by narrowing down
    - public Delete deleteFamily(byte[] family)
    - public Delete deleteColumn(byte[] family, byte[] qualifier)
    - public Delete deleteColumns(byte[] family, byte[] qualifier)
  - Notice deleteColumn vs deleteColumns
    - deleteColumns deletes ALL the versions of the cell but deleteColumn only deletes the latest
  - Most of the methods are overloaded to also take timestamp
    - Deletes everything on or before the provided timestamp
    - deleteColumn is an exception where only the exact timestamp match is removed

DeleteExample.java

```
public static void main(String[] args) throws IOException {
   Configuration conf = HBaseConfiguration.create();
   HTable hTable = new HTable(conf, "HBaseSamples");


   Delete delete = new Delete(toBytes("rowToDelete"));
   hTable.delete(delete);


   Delete delete1 = new Delete(toBytes("anotherRow"));
   delete1.deleteColumns(toBytes("metrics"), toBytes("loan"));
   hTable.delete(delete1);


   hTable.close();
}
```

Delete an entire row

Delete one cell rowId “anotherRow” and column metrics:loan

DeleteExample.java Output

```
hbase> put 'HBaseSamples', 'anotherRow', 'metrics:loan', 'deleteme'
hbase> put 'HBaseSamples', 'rowToDelete', 'metrics:loan', 'deleteme'
hbase> put 'HBaseSamples', 'anotherRow', 'metrics:keepMe', 'keepMe'

hbase> scan 'HBaseSamples', {COLUMNS=>['metrics:loan','metrics:keepMe']}
ROW            COLUMN+CELL
anotherRow     column=metrics:keepMe, timestamp=1326689202690, value=keepMe
anotherRow     column=metrics:loan,   timestamp=1326689182059, value=deleteme
rowToDelete    column=metrics:loan,   timestamp=1326689192229, value=deleteme
2 row(s) in 0.3310 seconds

hbase> quit

$ yarn jar $PLAY_AREA/HadoopSamples.jar hbase.DeleteExample

$ hbase shell
hbase> scan 'HBaseSamples', {COLUMNS=>['metrics:loan','metrics:keepMe']}
ROW            COLUMN+CELL
anotherRow     column=metrics:keepMe, timestamp=1326689202690, value=keepMe
1 row(s) in 0.3490 seconds
```


Wrap-Up

Summary

- **We learned how to**
  - Create records
  - Read records
  - Update records
  - Delete records

Questions?