FREE eBook

LEARNING hbase

Free unaffiliated eBook created from
Stack Overflow contributors.

#hbase

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: hbase

It is an unofficial and free hbase ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official hbase.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with hbase

Remarks

This section provides an overview of what hbase is, and why a developer might want to use it.

It should also mention any large subjects within hbase, and link out to the related topics. Since the Documentation for hbase is new, you may need to create initial versions of those related topics.

Examples

Installing HBase in Standalone

HBase Standalone is a mode which allow you to get rid of HDFS and to test HBase before deploying in a cluster, **It is not production oriented.**

Installing HBase in standalone is extremely simple. First you have to download the HBase archive named `hbase-X.X.X-bin.tar.gz` available on one of the apache mirrors.

Once you have done this, execute this shell command

```
tar xzvf hbase-X.X.X-bin.tar.gz
```

It will export the archive in your directory, you can put it wherever you want.

Now, go to the HBase directory you have exported and edit the file conf/hbase-env.sh

```
cd hbase-X.X.X
vi -o conf/hbase-env.xml
```

In this file, uncomment the line and change the path of JAVA_HOME

```
JAVA_HOME=/usr    #The directory must contain bin/java
```

Almost there ! now edit the file `conf/hbase-sitexml` and put the folowing lines

```
<configuration>
  <property>
    <name>hbase.rootdir</name>
    <value>file:///home/user/hbase</value>
  </property>
  <property>
    <name>hbase.zookeeper.property.dataDir</name>
    <value>/home/user/zookeeper</value>
  </property>
</configuration>
```

You can put those directories wherever you want to, just be sure to remember it if you want to

check logs etc.

Your HBase is now ready to run ! Just execute the command

```
bin/start-hbase.sh
```

and if you want to stop HBase

```
bin/stop-hbase.sh
```

Now your HBase is launched on your localhost and you can access it (using the Java API or the HBase shell). To run HBase shell, use

```
bin/hbase shell
```

Have fun using HBase !

Installing HBase in cluster

TODO

Read Getting started with hbase online: https://riptutorial.com/hbase/topic/4369/getting-started-with-hbase

Chapter 2: Using the Java API

Syntax

- HBaseConfiguration.create(); //Create a configuration file
- Configuration.set(String key, String value); //Add a key to the configuration
- ConnectionFactory.createConnection(HBaseConfiguration configuration); //Connects to HBase
- Connection.getAdmin(); //Instanciate a new Admin
- new HTableDescriptor(Table.valueOf(String tableName));; //Create a table descriptor
- HTableDescriptor.addFamily(new HColumnDescriptor(String familyName)); //Add a family to the table descriptor
- Admin.createTable(HTableDescriptor descriptor); //Create a table as described in the descriptor
- Admin.deleteTable(TableName.valueOf(String tableName)); //Delete a table
- Connection.getTable(TableName.valueOf(String tableName)); //Get a Table Object
- new Get(Bytes.toBytes(String row_key)); //Create a new Get
- table.get(Get get) //Returns a Result
- new Put(String row_key); //Create a new Put
- table.put(Put put); //Insert the row(s)
- new Scan(); //Create new Scan
- table.getScanner(Scan scan); //Return a ResultScanner
- new Delete(Bytes.toBytes(String row_key)); //Create a new Delete
- table.delete(Delete delete); //Delete a row from the table

Parameters

Parameter	Possible Values
CompareOp	CompareOp.EQUAL , CompareOp.GREATER , CompareOp.GREATER_OR_EQUAL , CompareOp.LESS , CompareOp.LESS_OR_EQUAL , CompareOp.NOT_EQUAL ,

Parameter	Possible Values
	CompareOp.NO_OP (no operation)

Remarks

This topic show various examples of how to use the Java API for HBase. In this topic you will learn to create and delete a table, insert, query and delete rows from a table but also use the Scans filters.

You will notice than many methods of this API take Bytes as parameters for example the columnFamily name, this is due to HBase implementation. For optimization purpose, instead of storing the values as String, Integer or whatever, it stores a list of Bytes, that is why you need to parse all those values as Bytes. To do this, the easiest method is to use `Bytes.toBytes(something)`.

Please feel free to notice if you see any mistake or misunderstanding.

Examples

Connecting to HBase

If you want to connect to an HBase server, first you need to make sure that the IP of the server is in your /etc/hosts file for example add the line

```
255.255.255.255    hbase
```

Then you can use the Java API to connect to zookeeper, you only have to specify the client port and the zookeeper address

```
Configuration config =  HBaseConfiguration.create();
config.set("hbase.zookeeper.quorum", "hbase");
config.set("hbase.zookeeper.property.clientPort","2181");
```

After you configured the connection, you can test it, using

```
HBaseAdmin.checkHBaseAvailable(config);
```

If you have a problem with your HBase configuration, an exception will be thrown.

Finally to connect to the server, just use

```
Connection connection = ConnectionFactory.createConnection(config);
```

Creating and deleting tables

In HBase, data are stored in tables with columns. Columns are regrouped in column families, which can be for example "personal" or "professional", each of these containing specific

informations.

To create a table, you need to use the Admin Object, create it using :

```
Admin admin = connection.getAdmin();
```

Once you have this admin, you can start creating tables. First of all make sure this table doesn't exist already with the line

```
admin.tableExists(TableName.valueOf("myTable);
```

This method will return true if the table exists. When you have checked this, you can create your table using the lines

```
HTableDescriptor descriptor = new HTableDescriptor(TableName.valueOf("myTable"));
descriptor.addFamily(new HColumnDescriptor("myFamily"));
admin.createTable(descriptor);
```

You need to set at least of family for the table, and HBase reference book recommends not getting over 3 column families else you will lose performances.

Congratulations ! Your table has been created !

If you need to delete your table, you can use

```
this.admin.disableTable(TableName.valueOf(tableName));
this.admin.deleteTable(TableName.valueOf(tableName));
```

Be sure to always disable the table first !

You now know how to manage tables in HBase.

Querying HBase, Get, Put, Delete and Scans

In HBase, you can use 4 types of operations

- **Get** : retrieves a row
- **Put** : inserts one or more row(s)
- **Delete** : delete a row
- **Scan** : retrieves several rows

If you simply want to retrieve a row, given its `row_key` you can use the `Get` object:

```
Get get = new Get(Bytes.toBytes("my_row_key"));
Table table = this.connection.getTable(TableName.valueOf("myTable"));
Result r = table.get(get);
byte[] value = r.getValue(Bytes.toBytes(columnFamily), Bytes.toBytes("myColumn"));
String valueStr = Bytes.toString(value);
System.out.println("Get result :" + valueStr);
```

Here we only get the value from the column we want, if you want to retrieve all the column, use the `rawCell` attribute from the `Get` object:

```
Get get = new Get(Bytes.toBytes(rowKey));
Table table = this.connection.getTable(TableName.valueOf(tableName));
Result r = table.get(get);
System.out.println("GET result :");
    for (Cell c : r.rawCells()) {
        System.out.println("Family : " + new String(CellUtil.cloneFamily(c)));
        System.out.println("Column Qualifier : " + new String(CellUtil.cloneQualifier(c)));
        System.out.println("Value : " + new String(CellUtil.cloneValue(c)));
        System.out.println("----------");
    }
```

Well, we can now retrieve data from our table, row by row, but how do we put some ? You use the `Put` object:

```
Put put = new Put("my_row_key");
put.addColumn(Bytes.toBytes("myFamily"), Bytes.toBytes("myColumn"),
Bytes.toBytes("awesomeValue");
//Add as many columns as you want

Table table = connection.getTable(TableName.valueOf("myTable");
table.put(put);
```

NB : `Table.put` can also take in parameter a list of puts, which is, when you want to add a lot of rows, way more efficient than put by put.

Alright now, I can put some rows and retrieve some from my HBase, but what if I want to get several rows and if I don't know my `row_keys` ?

Captain here ! You can use the `Scan` Object:

A scan basically look all the rows and retrieve them, you can add several parameters it, such as filters and start/end row but we will see that in another example.

If you want to scan all the column values from your table, given a column use the following lines:

```
Table table = this.connection.getTable(TableName.valueOf("myTable"));
Scan scan = new Scan();
scan.addColumn(Bytes.toBytes("myFamily"), Bytes.toBytes("myColumn"));
ResultScanner rs = table.getScanner(scan);
    try {
        for (Result r = rs.next(); r != null; r = rs.next()) {
            byte[] value = r.getValue(Bytes.toBytes("myFamily"), Bytes.toBytes("myCOlumn"));
            String valueStr = Bytes.toString(value);
            System.out.println("row key "+new String(r.getRow()));
            System.out.println("Scan result :" + valueStr);
        }
    } finally {
        rs.close(); // always close the ResultScanner!
   }
```

I really want to insist on the fact that **you must always close the ResultScanner** (same thing

than any ResultSet from a database by the way)

Nearly done ! Now let's learn how to delete a row. You have a `Delete` object for this:

```
Table table = this.connection.getTable(TableName.valueOf("myTable"));
Delete d = new Delete(Bytes.toBytes("my_weird_key"));
table.delete(d);
System.out.prinln("Row " + row_key + " from table " + tableName + " deleted");
```

One last thing: before executing any of the operations, always check that the table exists, or you will get an exception.

That's all for now, you can manage you data in HBase with this example.

Using the Scan filters

Basically, the Scan object retrieves all the rows from the table, but what if you want to retrieve only the rows where the value of a given column is equal to something ? Let me introduce you the **Filters**, they work like the *WHERE* in SQL.

Before starting using the filters, if you know how your *row_keys* are stored, you can set a starting row and an ending one for your Scan, which will optimize your query.

In HBase, *row_keys* are stored in the lexicographic order, but you can still use salting to change the way it is stored, I will not explain salting in this topic, it would take too long and that's not the point.

Let's get back to our row bounds, you have two methods to use to set the starting and ending row

```
Scan scan = new Scan();
scan.setStartRow(Bytes.toBytes("row_10"));
scan.setStopRow(Bytes.toBytes("row_42"));
```

This will change your scanner behavior to fetch all the rows between "row_10" and "row_42".

NB : As in most of the "sub" methods (for example substring), the startRow is inclusive and the stopRow is exclusive.

Now that we can bound our Scan, we should now add some filters to our scans, there are lots of those, but we will see here the most important ones.

- *If you want to retrieve all the rows having a row_key starting by a given pattern*

Use the `RowPrefixFilter` :

```
Scan scan = new Scan();
scan.setRowPrefixFilter(Bytes.toBytes("hello"));
```

With this code, your scan will only retrieve the rows having a row_key starting by "hello".

- *If you want to retrieve all the rows where the value of a given column is equal to something*

Use the `SingleColumnValueFilter` :

```
Scan scan = new Scan();
SingleColumnValueFilter filter = new
SingleColumnValueFilter(Bytes.toBytes("myFamily"),Bytes.toBytes("myColumn"), CompareOp.EQUAL,
Bytes.toBytes("42"));
scan.setFilter(filter);
```

With this code, you will get all the rows where the value of the column myColumn is equal to 42. You have different values for `CompareOp` which are explained in the Parameters section.

-Good, but what if I want to use regular expressions

Use the `RegexStringComparator` filter :

```
Scan scan = new Scan();
RegexStringComparator comparator = new RegexStringComparator(".hello.");
SingleColumnValueFilter filter = new
SingleColumnValueFilter(Bytes.toBytes("myFamily"),Bytes.toBytes("myColumn"), CompareOp.EQUAL,
comparator);
scan.setFilter(filter);
```

And you will get all the rows where the column myColumn contains hello.

Please also notice that the method `Scan.setFilter()` can also take a list of `Filter` as parameters

Read Using the Java API online: https://riptutorial.com/hbase/topic/4448/using-the-java-api

Credits

S. No	Chapters	Contributors
1	Getting started with hbase	Alexi Coard, BusyAnt, Community
2	Using the Java API	Alexi Coard, BusyAnt, KIM, Prutswonder