> HBase 数据模型的关键在于稀疏、分布式、多维、排序的映射。数据存储在 HDFS ，需要依赖 Zookeeper 进行管理。其中映射 map指代非关系型数据库的 key-Value 结构。

HBase 安装

安装 HBase

将 HBase 上传至新建的目录 `/opt/software/` 目录下，并解压至 `/opt/module` 目录下。

配置环境变量

```
export HBASE_HOME=/opt/module/hbase
export PATH=$PATH:$HBASE_HOME/bin
```

HBase 配置

修改配置文件

配置文件目录:

```
/opt/module/hbase/conf
```

1. 配置内部环境变量文件 `hbase-env.sh`

```
# 1.修改JDK的默认路径
export JAVA_HOME=[JKD安装路径](比赛中JDK路径为/opt/module/jdk1.8.0_261)

# 2.找到文件底部的 HBASE_MANAGES_ZK 参数，取消注释并设置为 false (默认为true)
# (该配置项说明：是否使用HBase自带的zookeeper，需要设置为否)
export HBASE_MANAGES_ZK=false
```

2. 修改配置文件 `hbase-site.xml`

```
<configuration>

    <!-- 指定HDFS的路径 -->
    <property>
        <name>hbase.rootdir</name>
        <value>hdfs://master:9090/hbase</value>
        <!-- 注：hdfs://master:9090/hbase必须与Hadoop集群的core-site.xml文件配置中的端口号保持一致，并且该项不识别IP，只能使用hostname。 -->
    </property>

    <!-- 指定启用分布式集群模式 -->
    <property>
        <name>hbase.cluster.distributed</name>
        <value>true</value>
    </property>
```

```
    <!-- 指定zookeeper集群 -->
    <property>
        <name>hbase.zookeeper.quorum</name>
        <value>master,slave1,slave2</value>
    </property>

    <!-- 指定zookeeper的临时文件夹 -->
    <property>
        <name>hbase.zookeeper.property.dataDir</name>
        <value>/opt/module/hbase/dataDir</value>
    </property>

</configuration>
```

3. 修改 Hbase 集群文件 `regionservers` ，添加集群

```
master
slave1
slave2
```

分发至其他集群

将环境变量以及 Hbase 分发至其他集群，并使用 `source /root/.bash_profile` 重新加载环境变量。

启动 HBase

Hbase 依赖于 HDFS 以及 Zookeeper ，在启动 Hbase 之前需要确保 **Hadoop** 以及 **Zookeeper** 完全分布式已经正常启动。

集群启动命令：

```
start-hbase.sh
```

停止服务：

```
stop-hbase.sh
```

查看进程：

若服务成功启动，在 master 主机上将会运行以下进程：

- HMaster
- HRegionServer

在 slave 主机上将会运行以下进程：

- HRegionServer

查看 WebUI

在浏览器输入 `主机IP:16010` 即可进入到 HBase 的 WebUI界面。

HBase 使用

HBase Shell

进入 HBase Shell

```
hbase shell
```

查看命令帮助

显示所有命令

```
help
```

查看指定命令帮助

```
help '命令名'
```

命名空间操作

注：如果使用表的时候不指定命名空间，则会使用默认命名空间 **default** 。

创建命名空间

```
create_namespace '命名空间名'
```

查看所有的命名空间

```
list_namespace
```

查看所有表

```
list
```

表结构操作

创建表

使用 `create` 创建表

```
create '命名空间:表名',{NAME => '列族1' , [属性 => 值]... } , {NAME => '列族1'} ...
```

案例：

在 bigdata 命名空间中创建表格 student，两个列族。info 列族数据维护的版本数为 5 个， 如果不写默认版本数为 1。

```
create 'bigdata:student',{NAME => 'info', VERSIONS => 5},{NAME => 'msg'}
```

修改表结构

使用 `alter` 修改表结构

增加列族和修改列族参数都使用覆盖的方法：

```
alter 'student1', {NAME => '列族名', VERSIONS => 3}
```

删除列族：

```
alter 'student1', 'delete' => '列族名'
```

删除表

使用 `drop` 删除表

shell 中删除表格，需要先使用 `disable` 将表格状态设置为不可用。

```
disable 'student1'
```

```
drop 'student1'
```

查看表结构

使用 `describe` 查看表结构

```
discribe 'bigdata:student'
```

表数据操作

查看表数据

使用 `scan` 读取多行数据，能够读取多行数据，不建议扫描过多的数据。推荐使用 `startRow` 和 `stopRow` 来控制读取的数据，默认范围左闭右开。

```
scan '表名',{STARTROW => '1001',STOPROW => '1002'}
```

（见API）

HBase API

创建连接

```
// 1.创建连接配置对象
// (包路径 import org.apache.hadoop.conf.Configuration)
val conf = new Configuration()

// 2.添加配置参数
// 参数的value为集群地址，与Hbase的配置文件保持一致
conf.set("hbase.zookeeper.quorum","master,slave1,slave2")

// 3.创建连接对象
// (包路径 import org.apache.hadoop.hbase.client.ConnectionFactory)
val connection = ConnectionFactory.createConnection(conf)

// 4.使用连接
(代码块...)

// 5.关闭连接
connection.close()
```

获取操作对象

HBase 的操作 API 都被封装在 **Admin** 与 **Table** 这两个操作对象当中，分别对应元数据的管理与表数据的管理。使用 Connection 获取这两个对象，通过调用对象的方法，即可对 HBase 进行操作。

表结构操作

获取 Admin

```
// 通过 Connection 获取 Admin
val admin = connection.getAdmin

// 使用 Admin
(...)

// 关闭 Admin
admin.close()
```

创建命名空间

使用 `admin.createNamespace(NamespaceDescriptor)` 方法创建命名空间。该方法需要传入一个 NamespaceDescriptor 对象。

```
// 构造 NamespaceDescriptor 对象
val DB_bigdata = NamespaceDescriptor.create("命名空间名")
 .addConfiguration("参数","值")
 .addConfiguration("...","...")
 .build()

// 创建命名空间
admin.createNamespace(DB_bigdata)
```

创建表名对象

使用 `TableName.valueOf("命名空间", "表名")` 获取 TableName 表名对象。

判断表格存在

使用 `admin.tableExists(TableName)` 方法判断指定表是否存在，返回布尔类型。

```
var bool = admin.tableExists(TableName.valueOf("命名空间", "表名"))
```

表数据操作

获取 Table

```
// 通过 Connection 获取 Table
val table = connection.getTable(TableName.valueOf(namespace,tablename))

// 使用 Table
(...)

// 关闭 Table
table.close()
```

写入数据

```
// 创建Put对象
val put = new put(Bytes.toBytes("行号"))

// 给put对象添加数据
put.addColumn(Bytes.toBytes("列族"),Bytes.toBytes("列名"),Bytes.toBytes("Value"))

// 调用put方法插入Put对象，完成数据写入
table.put(put)
```

读取数据

读取一整列

```
// 创建Get对象
val get = new Get(Bytes.toBytes("rowKey"));

// 如果不对Get对象添加参数，直接调用get方法读取数据，读到的是一整列的数据
table.get(get)
```

读取指定列

```
// 创建Get对象
val get = new Get(Bytes.toBytes("rowKey"));

// 指定读某列的数据
get.addColumn()

// 调用get方法读取数据
table.get(get)
```

扫描数据

单列过滤扫描

整行过滤扫描

删除数据

集成 Hive

前置准备

在 `hive-site.xml` 中添加 zookeeper 的属性

```
<property>
    <name>hive.zookeeper.quorum</name>
    <value>master,slave1,slave2</value>
</property>
<property>
    <name>hive.zookeeper.client.port</name>
    <value>2181</value>
</property>
```

将 Hive 数据同步至 Hbase

建立 Hive 表，关联 HBase 表，插入数据到 Hive 表的同时能够影响 HBase 表。

```
CREATE TABLE hive_hbase_emp_table(
 empno int,
 ename string,
 job string,
 mgr int,
 hiredate string,
 sal double,
 comm double,
 deptno int
)
WITH SERDEPROPERTIES
("hbase.columns.mapping"=":key,info:ename,info:job,info:mgr,info:hiredate,info:sal,inf
o:co
mm,info:deptno")
TBLPROPERTIES ("hbase.table.name" = "hbase_emp_table");
```

完成之后，可以分别进入 Hive 和 HBase 查看，都生成了对应的表。

Hive 建立外表关联 Hbase