# Hbase实战之新浪微博

# 1、Hbase的namespace介绍

## 1、namespace基本介绍

在HBase中，namespace命名空间指对一组表的逻辑分组，类似RDBMS中的database，方便对表在业务上划分。Apache HBase从0.98.0, 0.95.2两个版本号開始支持namespace级别的授权操作，HBase全局管理员能够创建、改动和回收namespace的授权。

## 2、namespace的作用

1、配额管理：限制一个namespace可以使用的资源，包括region和table

2、命名空间安全管理：提供了另一个层面的多租户安全管理

3、Region服务器组：一个命名或一张表，可以被固定到一组RegionServers上，从而保证了数据隔离性

## 3、namespace的基本操作

创建namespace

hbase>create\_namespace 'nametest'

查看namespace

hbase>describe\_namespace 'nametest'

列出所有namespace

hbase>list\_namespace

在namespace下创建表

hbase>create 'nametest:testtable', 'fm1'

查看namespace下的表

hbase>list\_namespace\_tables 'nametest'

删除namespace

hbase>drop\_namespace 'nametest'

# 2、Hbase的数据版本的确界以及TTL

## 1、数据的确界

在Hbase当中，我们可以为数据设置上界和下界，其实就是定义数据的历史版本保留多少个，通过自定义历史版本保存的数量，我们可以实现历史多个版本的数据的查询

1、版本的下界

默认的版本下界是0，即禁用。row版本使用的最小数目是与生存时间（TTL Time To Live）相结合的，并且我们根据实际需求可以有0或更多的版本，使用0，即只有1个版本的值写入cell。

2、版本的上界

之前默认的版本上界是3，也就是一个row保留3个副本（基于时间戳的插入）。该值不要设计的过大，一般的业务不会超过100。如果cell中存储的数据版本号超过了3个，再次插入数据时，最新的值会将最老的值覆盖。（现版本已默认为1）

## 2、数据的TTL

在实际工作当中经常会遇到有些数据过了一段时间我们可能就不需要了，那么这时候我们可以使用定时任务去定时的删除这些数据，或者我们也可以使用Hbase的TTL（Time To Live）功能，让我们的数据定期的会进行清除

使用代码来设置数据的确界以及设置数据的TTL如下

## 3、通过代码实现版本设定以及数据的TTL

### 第一步：创建maven工程并添加jar包

创建maven工程，导入jar包坐标

<dependencies>  
 <!-- https://mvnrepository.com/artifact/org.apache.hbase/hbase-client -->  
 <dependency>  
 <groupId>org.apache.hbase</groupId>  
 <artifactId>hbase-client</artifactId>  
 <version>2.0.0</version>  
 </dependency>  
 <!-- https://mvnrepository.com/artifact/org.apache.hbase/hbase-server -->  
 <dependency>  
 <groupId>org.apache.hbase</groupId>  
 <artifactId>hbase-server</artifactId>  
 <version>2.0.0</version>  
 </dependency>  
  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.12</version>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.testng</groupId>  
 <artifactId>testng</artifactId>  
 <version>6.14.3</version>  
 <scope>test</scope>  
 </dependency>  
  
 <!-- https://mvnrepository.com/artifact/org.apache.hbase/hbase-mapreduce -->  
 <dependency>  
 <groupId>org.apache.hbase</groupId>  
 <artifactId>hbase-mapreduce</artifactId>  
 <version>2.0.0</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.apache.hadoop</groupId>  
 <artifactId>hadoop-client</artifactId>  
 <version>2.7.5</version>  
 </dependency>  
 <dependency>  
 <groupId>org.apache.hadoop</groupId>  
 <artifactId>hadoop-hdfs</artifactId>  
 <version> 2.7.5</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.apache.hadoop</groupId>  
 <artifactId>hadoop-common</artifactId>  
 <version>2.7.5</version>  
 </dependency>  
</dependencies>  
<build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.0</version>  
 <configuration>  
 <source>1.8</source>  
 <target>1.8</target>  
 <encoding>UTF-8</encoding>  
 <!-- <verbal>true</verbal>-->  
 </configuration>  
 </plugin>  
 <!--将我们其他用到的一些jar包全部都打包进来 -->  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-shade-plugin</artifactId>  
 <version>2.4.3</version>  
 <executions>  
 <execution>  
 <phase>package</phase>  
 <goals>  
 <goal>shade</goal>  
 </goals>  
 <configuration>  
 <minimizeJar>false</minimizeJar>  
 </configuration>  
 </execution>  
 </executions>  
 </plugin>  
 </plugins>  
</build>

### 第二步：代码开发

**public class** HBaseVersionAndTTL {  
 **public static void** main(String[] args) **throws** IOException, InterruptedException {  
 Configuration configuration = HBaseConfiguration.*create*();  
 configuration.set(**"hbase.zookeeper.quorum"**,**"node01:2181,node02:2181,node03:2181"**);  
 Connection connection = ConnectionFactory.*createConnection*();  
 Admin admin = connection.getAdmin();  
 **if**(!admin.tableExists(TableName.*valueOf*(**"version\_hbase"**))){  
 HTableDescriptor hTableDescriptor = **new** HTableDescriptor(TableName.*valueOf*(**"version\_hbase"**));  
 HColumnDescriptor f1 = **new** HColumnDescriptor(**"f1"**);  
 f1.setMinVersions(3);  
 f1.setMaxVersions(5);  
 *//针对某一个列族下面所有的列设置TTL* f1.setTimeToLive(30);  
 hTableDescriptor.addFamily(f1);  
 admin.createTable(hTableDescriptor);  
 }  
 Table version\_hbase = connection.getTable(TableName.*valueOf*(**"version\_hbase"**));  
 Put put = **new** Put(**"1"**.getBytes());  
 *//针对某一条具体的数据设置TTL  
 //put.setTTL(3000);* put.addColumn(**"f1"**.getBytes(),**"name"**.getBytes(),System.*currentTimeMillis*(),**"zhangsan"**.getBytes());  
 version\_hbase.put(put);  
 Thread.*sleep*(1000);  
 Put put2 = **new** Put(**"1"**.getBytes());  
 put2.addColumn(**"f1"**.getBytes(),**"name"**.getBytes(),System.*currentTimeMillis*(),**"zhangsan2"**.getBytes());  
 version\_hbase.put(put2);  
 Get get = **new** Get(**"1"**.getBytes());  
 get.setMaxVersions();  
 Result result = version\_hbase.get(get);  
 Cell[] cells = result.rawCells();  
 **for** (Cell cell : cells) {  
 System.***out***.println(Bytes.*toString*(CellUtil.*cloneValue*(cell)));  
 }  
 version\_hbase.close();  
 connection.close();  
 }  
}

# 3、hbase微博实战案例

## 1、需求分析

1) 微博内容的浏览，数据库表设计

2) 用户社交体现：关注用户，取关用户

3) 拉取关注的人的微博内容

## 2、代码实现

### 2.1 准备工作

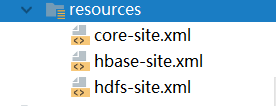
#### 第一步：创建maven工程并导入jar包

直接使用在版本确界当中创建的工程以及导入的jar包即可

#### 第二步：拷贝三个配置文件到maven工程的下

将node01服务器的/export/servers/hbase-2.0.0/conf 这个路径下的三个配置文件，分别是

core-site.xml、hdfs-site.xml、hbase-site.xml三个配置文件，拷贝到maven工程的resources资源目录下



### 2.2 代码设计总览：

1) 创建命名空间以及表名的定义

2) 创建微博内容表

3) 创建用户关系表

4) 创建用户微博内容接收邮件表

5) 发布微博内容

6) 添加关注用户

7) 移除（取关）用户

8) 获取关注的人的微博内容

### 2.3 创建命名空间以及表名的定义

**代码实现：**

*//微博内容表的表名*private static final byte[] *TABLE\_CONTENT* = Bytes.*toBytes*("weibo:content");  
*//用户关系表的表名*private static final byte[] *TABLE\_RELATIONS* = Bytes.*toBytes*("weibo:relations");  
*//微博收件箱表的表名*private static final byte[] *TABLE\_RECEIVE\_CONTENT\_EMAIL* = Bytes.*toBytes*("weibo:receive\_content\_email");  
*/\*\*  
 \* 初始化命名空间  
 \*/*public void initNameSpace() throws IOException {  
 Connection connection = getConnection();  
 Admin admin = connection.getAdmin();  
 NamespaceDescriptor namespaceDescriptor = NamespaceDescriptor.*create*("weibo2").addConfiguration("creator","jim").build();  
 admin.createNamespace(namespaceDescriptor);  
 admin.close();  
 connection.close();  
}  
  
public Connection getConnection() throws IOException {  
 Configuration configuration = HBaseConfiguration.*create*();  
 configuration.set("hbase.zookeeper.quorum","node01:2181,node02:2181,node03:2181");  
 Connection connection = ConnectionFactory.*createConnection*();  
 return connection;  
}

### 2.4 创建微博内容表

**表结构：**

|  |  |
| --- | --- |
| **方法名** | creatTableeContent |
| Table Name | weibo:content |
| RowKey | 用户ID\_时间戳 |
| ColumnFamily | info |
| ColumnLabel | 标题,内容,图片 |
| Version | 1个版本 |

**代码实现：**

/\*\*  
 \* 创建微博内容存储表  
 \*  
 \* 方法名 creatTableeContent  
 Table Name weibo:content  
 RowKey 用户ID\_时间戳  
 ColumnFamily info  
 ColumnLabel 标题,内容,图片  
 Version 1个版本  
  
 \*  
 \*/  
**public void** creatTableeContent() **throws** IOException {  
 //获取连接  
 Connection connection = getConnection();  
 //获取管理员对象  
 Admin admin = connection.getAdmin();  
 //得到HTableDescriptor对象  
 HTableDescriptor hTableDescriptor = **new** HTableDescriptor(TableName.valueOf(**"weibo:content"**));  
 //添加列族info  
 HColumnDescriptor info = **new** HColumnDescriptor(**"info"**);  
 //设置版本确界  
 info.setMaxVersions(1);  
 info.setMinVersions(1);  
 info.setBlockCacheEnabled(**true**);  
 //设置数据压缩  
 // info.setCompressionType(Compression.Algorithm.SNAPPY);  
 info.setBlocksize(2048\*1024);  
 hTableDescriptor.addFamily(info);  
 //创建表  
 admin.createTable(hTableDescriptor);  
 admin.close();  
 connection.close();  
}

### 2.5 创建用户关系表

**表结构：**

|  |  |
| --- | --- |
| **方法名** | createTableRelations |
| Table Name | weibo:relations |
| RowKey | 用户ID |
| ColumnFamily | attends、fans |
| ColumnLabel | 关注用户ID，粉丝用户ID |
| ColumnValue | 用户ID |
| Version | 1个版本 |

**代码实现：**

/\*\*  
 \* 创建用户关系表  
 \* 方法名 createTableRelations  
 Table Name weibo:relations  
 RowKey 用户ID  
 ColumnFamily attends、fans  
 ColumnLabel 关注用户ID，粉丝用户ID  
 ColumnValue 用户ID  
 Version 1个版本  
  
 \*/  
**public void** createTableRelations() **throws** IOException {  
 //获取连接  
 Connection connection = getConnection();  
 //获取管理员对象  
 Admin admin = connection.getAdmin();  
 //得到表定义  
 HTableDescriptor hTableDescriptor = **new** HTableDescriptor(TableName.valueOf(**TABLE\_RELATIONS**));  
 HColumnDescriptor attends = **new** HColumnDescriptor(**"attends"**);  
 HColumnDescriptor fans = **new** HColumnDescriptor(**"fans"**);  
  
 attends.setBlocksize(2048\*1024);  
 attends.setBlockCacheEnabled(**true**);  
 attends.setMinVersions(1);  
 attends.setMaxVersions(1);  
  
  
 fans.setBlocksize(2048\*1024);  
 fans.setBlockCacheEnabled(**true**);  
 fans.setMinVersions(1);  
 fans.setMaxVersions(1);  
  
 hTableDescriptor.addFamily(attends);  
 hTableDescriptor.addFamily(fans);  
 admin.createTable(hTableDescriptor);  
 admin.close();  
 connection.close();  
}

### 2.6 创建微博收件箱表

**表结构：**

|  |  |
| --- | --- |
| **方法名** | createTableReceiveContentEmails |
| Table Name | weibo:receive\_content\_email |
| RowKey | 用户ID |
| ColumnFamily | info |
| ColumnLabel | 用户ID |
| ColumnValue | 取微博内容的RowKey |
| Version | 1000 |

**代码实现：**

/\*\*  
 \* 表结构：  
 方法名 createTableReceiveContentEmails  
 Table Name weibo:receive\_content\_email  
 RowKey 用户ID  
 ColumnFamily info  
 ColumnLabel 用户ID  
 ColumnValue 取微博内容的RowKey  
 Version 1000  
  
 \*/  
  
**public void** createTableReceiveContentEmails() **throws** IOException {  
 //获取连接  
 Connection connection = getConnection();  
 //得到管理员对象  
 Admin admin = connection.getAdmin();  
 //获取HTableDescriptor  
 HTableDescriptor hTableDescriptor = **new** HTableDescriptor(TableName.valueOf(**TABLE\_RECEIVE\_CONTENT\_EMAIL**));  
 //定义列族名称  
 HColumnDescriptor info = **new** HColumnDescriptor(**"info"**);  
 info.setMaxVersions(1000);  
 info.setMinVersions(1000);  
 info.setBlockCacheEnabled(**true**);  
 info.setBlocksize(2048\*1024);  
 hTableDescriptor.addFamily(info);  
 //创建表  
 admin.createTable(hTableDescriptor);  
 admin.close();  
 connection.close();  
}

### 2.7 发布微博内容

a、微博内容表中添加1条数据

b、微博收件箱表对所有粉丝用户添加数据

**代码实现：**

*/\*\*  
 \* 发布微博内容  
 \** ***@param uid*** *\** ***@param content*** *\*/***public void** publishWeiBo(String uid ,String content) **throws** IOException {  
 Connection connection = getConnection();  
 Table table = connection.getTable(TableName.*valueOf*(***TABLE\_CONTENT***));  
 String rowkey = uid + **"\_"**+ System.*currentTimeMillis*();  
 Put put = **new** Put(rowkey.getBytes());  
 put.addColumn(**"info"**.getBytes(),**"content"**.getBytes(),System.*currentTimeMillis*(),content.getBytes());  
 table.put(put);  
 *//微博用户关系表* Table table\_relations = connection.getTable(TableName.*valueOf*(***TABLE\_RELATIONS***));  
 Get get = **new** Get(uid.getBytes());  
 get.addFamily(**"fans"**.getBytes());  
 Result result = table\_relations.get(get);  
 Cell[] cells = result.rawCells();  
 **if**(cells.**length** <= 0){  
 **return** ;  
 }  
 List<**byte**[]> allFans = **new** ArrayList<**byte**[]>();  
 *//将所有的粉丝都获取到，然后将数据保存到粉丝表当中去* **for** (Cell cell : cells) {  
 **byte**[] bytes = CellUtil.*cloneQualifier*(cell);  
 allFans.add(bytes);  
 }  
 Table table\_receive\_content\_email = connection.getTable(TableName.*valueOf*(***TABLE\_RECEIVE\_CONTENT\_EMAIL***));  
 List<Put> putFansList = **new** ArrayList<>();  
 **for** (**byte**[] allFan : allFans) {  
 Put put1 = **new** Put(allFan);  
 put1.addColumn(**"info"**.getBytes(),Bytes.*toBytes*(uid),System.*currentTimeMillis*(),rowkey.getBytes());  
 putFansList.add(put1);  
 }  
 table\_receive\_content\_email.put(putFansList);  
}

### 2.8 添加关注用户

a、在微博用户关系表中，对当前主动操作的用户添加新关注的好友

b、在微博用户关系表中，对被关注的用户添加新的粉丝

c、微博收件箱表中添加所关注的用户发布的微博

**代码实现：**

/\*\*  
 \* 添加关注用户，一次可能添加多个关注用户  
 \* A 关注一批用户 B,C ,D  
 \* 第一步：A是B,C,D的关注者 在weibo:relations 当中attend列族当中以A作为rowkey，B,C,D作为列名，B,C,D作为列值，保存起来  
 \* 第二步：B,C,D都会多一个粉丝A 在weibo:relations 当中fans列族当中分别以B,C,D作为rowkey，A作为列名，A作为列值，保存起来  
 \* 第三步：A需要获取B,C,D 的微博内容存放到 receive\_content\_email 表当中去，以A作为rowkey，B,C,D作为列名，获取B,C,D发布的微博rowkey，放到对应的列值里面去  
 \*  
 \*  
 \* **@param uid** \* **@param attends** \*/  
**public void** addAttends(String uid ,String ...attends) **throws** IOException {  
 Connection connection = getConnection();  
 Table relation\_table = connection.getTable(TableName.valueOf(**TABLE\_RELATIONS**));  
 //用户关注人,attend列族当中添加数据  
 Put put = **new** Put(uid.getBytes());  
 **for** (String attend : attends) {  
 put.addColumn(**"attends"**.getBytes(),attend.getBytes(),attend.getBytes());  
 }  
 relation\_table.put(put);  
 //粉丝fans添加粉丝，A 关注B，那么自然B就需要添加一个粉丝A  
 **for** (String attend : attends) {  
 Put put1 = **new** Put(attend.getBytes());  
 put1.addColumn(**"fans"**.getBytes(),uid.getBytes(),uid.getBytes());  
 relation\_table.put(put1);  
 }  
  
 //获取uid的所有关注人的收件箱，放到收件箱列表weibo:receive\_content\_email里面去  
 //A 关注B，那么A需要获取B所有的微博内容  
 Table table\_content = connection.getTable(TableName.valueOf(**TABLE\_CONTENT**));  
 Scan scan = **new** Scan();  
 List<**byte**[]> rowkeyBytes = **new** ArrayList<>();  
 **for** (String attend : attends) {  
 RowFilter rowFilter = **new** RowFilter(CompareOperator.**EQUAL**,**new** SubstringComparator(attend+**"\_"**));  
 scan.setFilter(rowFilter);  
 ResultScanner scanner = table\_content.getScanner(scan);  
 **for** (Result result : scanner) {  
 //获取到数据的rowkey  
 **byte**[] rowkey = result.getRow();  
 rowkeyBytes.add(rowkey);  
 }  
 }  
  
  
  
  
 **if**(rowkeyBytes.size() > 0){  
 Table table\_receive\_content = connection.getTable(TableName.valueOf(**TABLE\_RECEIVE\_CONTENT\_EMAIL**));  
 List<Put> recPuts = **new** ArrayList<Put>();  
  
 **for** (**byte**[] rowkeyByte : rowkeyBytes) {  
 Put put1 = **new** Put(uid.getBytes());  
 String rowKeyStr = Bytes.toString(rowkeyByte);  
 String attendUid = rowKeyStr.substring(0, rowKeyStr.indexOf(**"\_"**));  
 **long** timestamp = Long.parseLong(rowKeyStr.substring(rowKeyStr.indexOf(**"\_"**) + 1));  
  
 put1.addColumn(**"info"**.getBytes(),attendUid.getBytes(), timestamp,rowkeyByte);  
 recPuts.add(put1);  
 }  
  
 table\_receive\_content.put(recPuts);  
  
 }  
  
}

### 2.9 移除（取关）用户

a、在微博用户关系表中，对当前主动操作的用户移除取关的好友(attends)

b、在微博用户关系表中，对被取关的用户移除粉丝

c、微博收件箱中删除取关的用户发布的微博

**代码实现：**

/\*\*  
 \* 取消关注 A取消关注 B,C,D这三个用户  
 \* 其实逻辑与关注B,C,D相反即可  
 \* 第一步：在weibo:relation关系表当中，在attends列族当中删除B,C,D这三个列  
 \* 第二步：在weibo:relation关系表当中，在fans列族当中，以B,C,D为rowkey，查找fans列族当中A这个粉丝，给删除掉  
 \* 第三步：A取消关注B,C,D,在收件箱中，删除取关的人的微博的rowkey  
 \*/  
**public void** attendCancel(String uid,String ...cancelAttends) **throws** IOException {  
 Connection connection = getConnection();  
 Table table\_relations = connection.getTable(TableName.valueOf(**TABLE\_RELATIONS**));  
  
 //移除A关注的B,C,D这三个用户  
 **for** (String cancelAttend : cancelAttends) {  
 Delete delete = **new** Delete(uid.getBytes());  
 delete.addColumn(**"attends"**.getBytes(),cancelAttend.getBytes());  
 table\_relations.delete(delete);  
 }  
  
 //B,C,D这三个用户移除粉丝A  
 **for** (String cancelAttend : cancelAttends) {  
 Delete delete = **new** Delete(cancelAttend.getBytes());  
 delete.addColumn(**"fans"**.getBytes(),uid.getBytes());  
 table\_relations.delete(delete);  
 }  
  
 //收件箱表当中 A移除掉B,C,D的信息  
 Table table\_receive\_connection = connection.getTable(TableName.valueOf(**TABLE\_RECEIVE\_CONTENT\_EMAIL**));  
  
 **for** (String cancelAttend : cancelAttends) {  
 Delete delete = **new** Delete(uid.getBytes());  
 delete.addColumn(**"info"**.getBytes(),cancelAttend.getBytes());  
 table\_receive\_connection.delete(delete);  
  
 }  
 table\_receive\_connection.close();  
 table\_relations.close();  
 connection.close();  
}

### 2.10 获取关注的人的微博内容

a、从微博收件箱中获取所关注的用户的微博RowKey

b、根据获取的RowKey，得到微博内容

**代码实现：**

/\*\*  
 \* 某个用户获取收件箱表内容  
 \* 例如A用户刷新微博，拉取他所有关注人的微博内容  
 \* A 从 weibo:receive\_content\_email 表当中获取所有关注人的rowkey  
 \* 通过rowkey从weibo:content表当中获取微博内容  
 \*/  
**public void** getContent(String uid) **throws** IOException {  
 //从weibo:receive\_content\_email 表当中获取用户id为uid的人的所有的微博列表  
 Connection connection = getConnection();  
 //从 weibo:receive\_content\_email  
 Table table\_receive\_content\_email = connection.getTable(TableName.valueOf(**TABLE\_RECEIVE\_CONTENT\_EMAIL**));  
 //定义list集合里面存储我们的所有的Get对象，用于下一步的查询  
 List<Get> rowkeysList = **new** ArrayList<Get>();  
 Get get = **new** Get(uid.getBytes());  
 //设置最大版本为5个  
 get.setMaxVersions(5);  
 Result result = table\_receive\_content\_email.get(get);  
 Cell[] cells = result.rawCells();  
 **for** (Cell cell : cells) {  
 **byte**[] rowkeys = CellUtil.cloneValue(cell);  
 Get get1 = **new** Get(rowkeys);  
 rowkeysList.add(get1);  
 }  
 //从weibo:content表当中通过用户id进行查询微博内容  
 //table\_content内容表  
 Table table\_content = connection.getTable(TableName.valueOf(**TABLE\_CONTENT**));  
 //所有查询出来的内容进行打印出来  
 Result[] results = table\_content.get(rowkeysList);  
 **for** (Result result1 : results) {  
 **byte**[] value = result1.getValue(**"info"**.getBytes(), **"content"**.getBytes());  
 **byte**[] row = result1.getRow();  
 String rowkey = Bytes.toString(row);  
 String[] split = rowkey.split(**"\_"**);  
 Content content = **new** Content();  
 content.setUid(split[0]);  
 content.setTimeStamp(Long.parseLong(split[1]));  
 content.setContent(Bytes.toString(value));  
 System.**out**.println(content.toString());  
 }  
}