参考：<https://portal.huaweicloud.com/blogs/e4d29a6ea7d511e69023286ed488c65c>

**HBase（四） HBase JAVA API - Schema操作**

【摘要】 HBase JAVA API - Schema操作...

**Schema操作**

**表**

表描述通过HTableDescriptor对象描述。属性可以通过setter方法设置。

**表名**

setName(byte[] name)   
表名不能以.或-开头，只能包含字母和数字，一般在构造函数就设置。   
表名会作为存储系统中存储路径的一部分来使用（me:类似包名），因此必须要符合文件名规范。

**列族**

addFamily(HColumnDescriptor family)

**文件大小设置**

setMaxFileSize(long maxFileSize)   
这个参数设置表的region大小，方法名有问题，并不直观，事实上是每个存储单元的大小限制。如果一个列族的存储单元已用空间超过大小限制，region会产生拆分。默认是256M。   
当表的数据量非常大的时候，可以适当调高这个参数。   
这个参数是个预期值，可能实际大小会超过这个设置。比如，参数为10M，然后插入了一行20M数据，由于一行数据不能跨region存储，所以也就不会被拆分

**只读**

setReadOnly(boolean readOnly)   
默认所有表都可以写，特殊时候可以设置只读

**memstore flush大小**

setMemStoreFlushSize(long memstoreFlushSize)   
Hbase内存中预留了写缓存区，写操作会写入缓冲区，然后按条件触发顺序写入文件，这个过程是flush的过程。   
这个参数默认为64MB，参数越大，生成的存储文件越大，文件数量越小，同时也导致更长时间的阻塞时间。这时候region server不能持续接收新的数据，请求被阻塞的时间也响应增加。同时，系统崩溃时，WAL恢复的时间也增加。

**延时日志写**

setDeferredLogFlush(boolean isDeferredLogFlush)   
Hbase有两种把WAL保存磁盘的方式，一种是延时日志写，即deferred log flushing。另一种相反。这个参数默认是false。

**任意键值设置**

setValue(String key, String value)   
setValue(byte[] key, byte[] value)   
setValue(ImmutableBytesWritable key, ImmutableBytesWritable value)   
这些数据保存在预定义的表里，可以被查询，可以通过加载协处理器的方式进行查询。内部使用ImmutableBytesWritable 类型存储，目的是为了能够序列化。

**列族**

列族通过HColumnDescriptor对象描述。这也是个失败的命名，叫做HFamilyDescriptor恐怕更好。列族定义所有列的共享信息，并且可以通过客户端创建任意数量的列，通常定义某列都要和列族名合并在一起，中间用:分割，即family:qualifier。列族名也必须是可见字符，因为也是要在底层存储系统中使用的，列族会映射到独立的存储文件。   
Hbase虽然支持空列名，但是一开始就该制定列名的，因为后期无法简单的重命名空列。   
同表一样，列族也有一系列的属性可以设置：

**列族名**

一般在构造函数中传入，因为列族不能被重命名。如果需要，通常是新建一个列族，然后复制数据到新的列族。

**最大版本数**

setMaxVersions(int maxVersions)   
限定了每个值能保留的最大版本数量。Hbase会删除超过最大版本的数据。默认是3，如果不需要访问旧数据，也可以设置为1.

**压缩**

setCompactionCompressionType(Compression.Algorithm type)   
setCompressionType(Compression.Algorithm type)   
Hbase支持插件式的压缩算法，默认是”不压缩”，即Compression.Algorithm. NONE 。其他选项有GZ ，LZ4 ，LZO ，SNAPPY ，除了GZ使用本地库或java库，其他需要相关类库支持。

**块大小**

setBlocksize(int s)   
Hbase的存储文件被划分为若干个小存储块，这些块在get或scan会被加载到内存中，类似于Oracle中的块。默认大小是64KB。这里列族的块，不同于HDFS的块，也没有依赖

**开启块缓存**

setBlockCacheEnabled(boolean blockCacheEnabled)   
Hbase顺序读取一个数据块到内存缓存中，其读取相邻的数据时就可以在内存中读取而不需要查询磁盘，可以有效减少IO。参数默认为true，即每次读取的块都会缓存到内存中。   
但是如果只需要块中的某个部分，比如特定顺序访问列族，需要设为禁用

**保存时间TTL**

setTimeToLive(int timeToLive)   
相对于版本数的另一种判断数据删除的方法。TTL设置了一个基于时间戳的临界值，Hbase会自动检查TTL是否达到上限，在major合并的时候，TTL超时的数据会被删除。参数单位是秒，默认值是Integer.MAX\_VALUE，可以理解为默认为永久保存。

**列族缓存**

setInMemory(boolean inMemory)   
默认为false。注意，设置为true也不意味着列族的所有存储块都会加载到内存中，也不意味着会长期保存，这只是一种高优先级的承诺。在正常数据读的过程中，块数据会被加载到缓存中并长期保存在内存，除非heap压力过大，才会强制从内存移除这部分数据。   
这个参数适合数据量比较小的列族，如常用的、检索频繁的信息。

**Bloom过滤器**

setBloomFilterType(StoreFile.BloomType bt)   
默认是关闭状态，即StoreFile.BloomType. NONE。其他可选项为ROW（Bloom enabled with Table row as Key）、ROWCOL（Bloom enabled with Table row & column (family+qualifier) as Key）。   
由于列的数量一般多于行（除非每行只有一列），所以ROWCOL选项会占用大量的空间，但是粒度会更细，使用时需要考量。

**复制范围**

setScope(int scope)   
hbase提供replication功能，能跨集群同步。本参数默认为0，即关闭状态。还有个选项就是1，即开启状态。其实更像一个bool参数

**代码示例**

public static void main(String[] args) throws Exception {   
Configuration conf = HBaseConfiguration.create();   
HBaseAdmin admin = new HBaseAdmin(conf);   
// 这个是隐式的实现，客户端不能显示的调用，是框架在特殊情况下隐式调用   
// admin.abort("cause", new Exception());   
byte[] tableName = Bytes.toBytes("t2");   
// 简单建表   
HTableDescriptor tableDesc = new HTableDescriptor(tableName);   
HColumnDescriptor f1 = new HColumnDescriptor("f1");   
tableDesc.addFamily(f1);   
admin.createTable(tableDesc);   
System.out.println(admin.tableExists(tableName));   
System.out.println(admin.isTableAvailable(tableName));   
// 修改表结构，添加一个列族   
HColumnDescriptor f2 = new HColumnDescriptor("f2");   
tableDesc.addFamily(f2);   
// 禁用状态才能修改表结构   
// 表禁用时，region服务器会将内存中还未提交的已修改数据刷入磁盘，然后关闭所有region，并更新这表的元数据，标记下线   
// 表禁用可能会费时，时长取决于多少数据还在内存中未flush到磁盘   
admin.disableTable(tableName);   
admin.modifyTable(tableName, tableDesc);   
System.out.println(admin.getTableDescriptor(tableName));   
admin.deleteColumn("t2", "f1");   
System.out.println(admin.getTableDescriptor(tableName));   
f2.setCacheDataOnWrite(true);   
admin.modifyColumn(tableName, f2);   
System.out.println(admin.getTableDescriptor(tableName));   
admin.addColumn(tableName, f1);   
System.out.println(admin.getTableDescriptor(tableName));   
// 启用在转移表的region到其他可用服务器的场景也比较有用   
admin.enableTable(tableName);   
// 删除表前也要禁用，删除启用的表会抛异常。表被删除，数据也同时被删除   
admin.disableTable(tableName);   
admin.deleteTable(tableName);   
System.out.println(admin.tableExists(tableName));   
// 建表同时预分区，表创建的时候就划分若干个特定的region，分别是起始、终止行健和region数，region数要大于3   
admin.createTable(tableDesc, Bytes.toBytes(1l), Bytes.toBytes(100l), 10);   
printTableRegions(tableName, admin);   
admin.disableTable(tableName);   
admin.deleteTable(tableName);   
// 预分区的另一种方式，直接给出已拆分好的region边界列表   
// 上面的方法起始就是先调用Bytes.split()方法计算出边界，然后调用这种方式而已   
byte[][] regions = new byte[][] { Bytes.toBytes("A"), Bytes.toBytes("D"), Bytes.toBytes("G"), Bytes.toBytes("K"), Bytes.toBytes("O"), Bytes.toBytes("T") };   
admin.createTable(tableDesc, regions);   
printTableRegions(tableName, admin);   
admin.disableTable(tableName);   
admin.deleteTable(tableName);   
// 异步建表，内部实现中，上面的同步模式就是异步的简单封装，循环不断检查这个任务是否完成   
admin.createTableAsync(tableDesc, regions);   
// 关闭HBaseAdmin实例所有资源，包括与远程服务器的连接   
admin.close();   
}   
private static void printTableRegions(byte[] tableName, HBaseAdmin admin) throws IOException {   
System.out.println("Printing regions of table: " + Bytes.toString(tableName));   
List hrlList = admin.getConnection().locateRegions(tableName);   
System.out.println("region cnt: " + hrlList.size());   
  
// 下面输出可以看到hbase设计region边界的规则 第一个region的起始行健和最后一个region的终止行健都是空字节   
// 前一个region的终止行健与后一个region的起始行健是相同的 终止行健不包含在region内，而起始行健包括，描述为[ )   
// 样例输出： start key: , end key: 1 start key: 1, end key: 13 …… start key:   
// 85, end key: 100 start key: 100, end key:   
  
for (HRegionLocation hrl : hrlList) {   
HRegionInfo ri = hrl.getRegionInfo();   
byte[] sk = ri.getStartKey();   
byte[] ek = ri.getEndKey();   
System.out.println(" start key: " + (sk.length == 8 ? Bytes.toLong(sk) : Bytes.toString(sk)) + ", end key: " + (ek.length == 8 ? Bytes.toLong(ek) : Bytes.toString(ek)));   
}   
}

作者 | 林钰鑫

转载请注明出处：华为云博客 https://portal.hwclouds.com/blogs