### MarsBase二级索引

#### 1设计思路：

##### 1.1索引表

一个用户表只拥有一个索引表，对该表创建的所有索引，其数据都保存在同一张HBase表，没有直接将索引数据放到用户表里是考虑到用户表海量数据的情况下再加入索引数据的话，用户表会变得很大。

但是把索引数据放到单独的表里需要考虑一下几种情况：

(1)实时写入时，索引数据与用户数据的一致性

(2)需要自定义负载均衡器，使主表数据与对应的索引数据在同一个RS上，减少查询时的网络传输。

(3) split时必须保证数据region和索引region的切分点逻辑上一样。

##### 1.2索引rowKey设计

【索引region的startkey】+【一个0字节】+【索引名】+【最大索引名的填充】+【【索引列的值】】+【主表的rowkey】

说明：

【一个0字节】是考虑第一个region没有startKey的情况

【索引名】+【最大索引名的填充】 索引名最长可以有18个字节，少于18个字节的话，0字节补齐

【【索引列的值】】表示被索引的所有列的列值的集合，暂时没有设计成定长

【主表的rowkey】表示主表数据的rowKey

问题：

对多列进行索引时，索引rowKey中【【索引列的值】】是由多个列的value拼接的，查询时如何分割？

第一种想法：列值之间使用分隔符，不适用type为String

第二种想法：列值设置为定长，不够用空字节补齐，同样不适用type为String

第三种想法：将每个列值长度保存在索引表的列中，扫描索引表时需要对扫描结果再进行过滤

（未解决）

##### 1.3索引数据实时写及一致性控制

实时写入使用HBase RegionObserver协处理器，主表数据写入之前，在协处理器中构建索引数据的写入任务，主表数据写入成功之后执行索引数据的写入。

一致性控制：索引表的操作和主表操作同时写入WAL并同步，这样主表数据写入完成时，索引的WAL也已经写到磁盘了，即使之后索引数据写入时出现异常，也可以根据WAL重新写入，实现最终一致性。

##### 1.4 split操作

禁止索引表的自动Split和手动Split，只能由用户表Split时触发。

索引表Split之后，第二个子Region的索引数据，需要修改RowKey的第一部分【startKey】为SplitKey。

##### 1.5负载均衡

自定义负载均衡器，遵循一个原则：

用户Region使用HBase默认的分配策略，索引Region根据用户Region的分配情况来进行移动（跟随）。

#### 2目前实现：

##### 2.1索引创建

(1)若索引表存在，只添加元数据到mysql

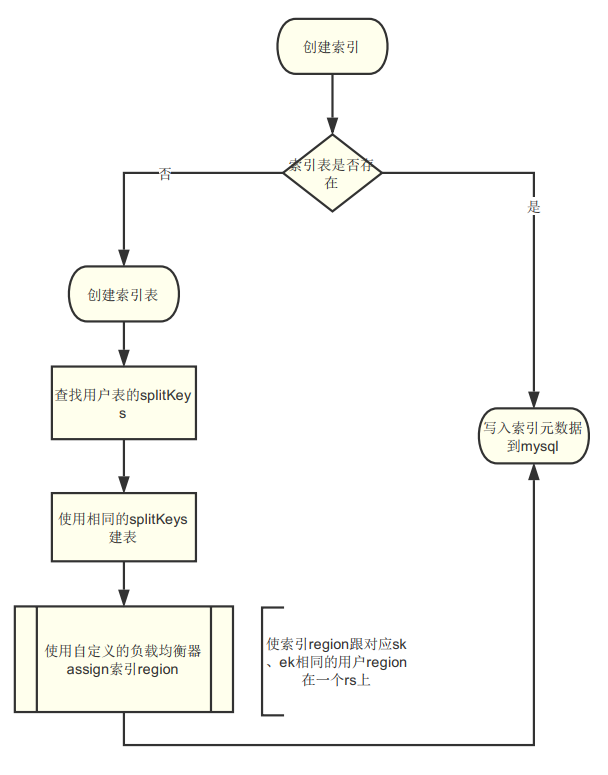
(2)若索引表不存在，先创建索引表，然后添加元数据到mysql

(3)查找对应用户表的splitKey

(4)根据用户表的splitKey创建索引表

(5)自定义Balancer修改索引region的注册策略：将索引region注册到与其startKey、endKey相同的用户region所在的regionServer上，这样保证用户表数据与索引数据在同一个服务器上，减少查询时的网络传输。

**流程图**：



##### 2.2索引数据实时写入并保证一致性

###### 2.2.1索引rowKey格式：

【索引region的startkey】+【一个0字节】+【索引名】+【最大索引名的填充】+【【索引列的值】】+【主表的rowkey】

(1)【一个0字节】是考虑第一个region没有startKey的情况

(2)【索引名】+【最大索引名的填充】 索引名长度目前定义的是18个字节，定长

(3)【【索引列的值】】表示被索引的所有列的列值的集合，暂时没有设计成定长

(4)【主表的rowkey】表示主表数据的rowKey

###### 2.2.2 索引列值（mf：mc）格式：

主表数据rowKey的长度+startKey的长度+索引数据rowKey总长度，共6个字节

###### 2.2.3数据写入:

(1) 入口：主表put数据

(2) 协处理器前钩子函数：preBatchMutate(..)

根据用户region找到对应的索引region（startKey与endKey相同的）

构建索引rowKey

创建索引Put对象（日志级别设为SkipWAL）及索引WALEdit对象。

将索引Put对象、WALEdit对象及索引region一起保存到ThreadLocal对象中

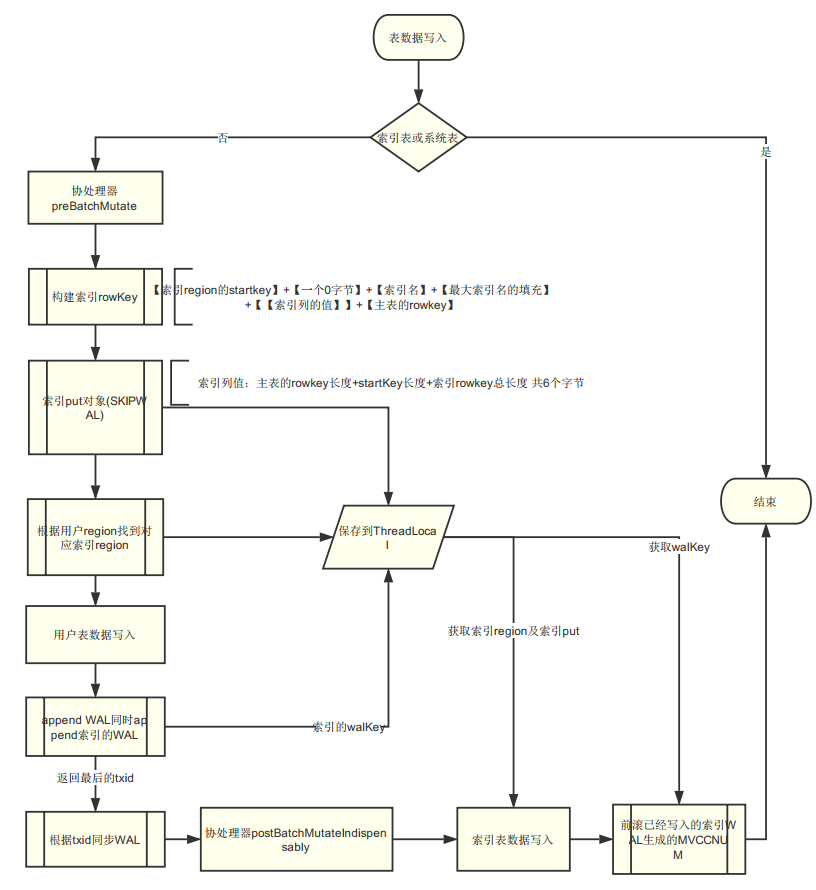
(3) 主表数据写入主流程：doMiniBatchMutation（..）修改HRegion类中的此方法，在第四步Append the final edit to WAL时，将索引的WALEdit也 append到WAL，同时返回最后的txId.以保证在同步WAL（syncOrDefer(txid, durability);）时，用户reigon的WAL数据与索引region的WAL数 据都被同步到磁盘。

将上一步的索引WALKey保存到ThreadLocal对象中。

(4) 写入成功后执行后钩子postBatchMutateIndispensably（..）从ThreadLocal中拿到索引region对象及索引Put对象（数组），执行索引region的 batchMutate（..）方法

(5) 索引数据写入主流程：doMiniBatchMutation（..）从ThreadLocal中拿到索引WALKey，同步其中的mvccNum

**流程图**



##### 2.3带索引的split

禁用索引表的自动split和手动split，仅在用户表split时触发

###### 2.3.1入口：用户表split

（1）校验splitKey是否合格

（2） 创建两个子region

###### 2.3.2协处理器钩子：preSplitBeforePONR

（1）遍历索引数据，将splitKey与索引rowKey包含的主数据rowKey用二进制比较器进行比较：当主键大于等于splitKey时，会修改索引rowKey， 将第一部分的startkey改为splitKey

（2） 将修改flush到磁盘

（3） 执行索引region split流程，创建两个子region

（4）从元数据中下线索引父reigon

###### 2.3.3设置PONR（不可回退点）

###### 2.3.4从元数据下线用户父region

###### 2.3.5协处理器钩子：preSplitAfterPONR

（1） 上线索引子region

###### 2.3.6上线用户子region

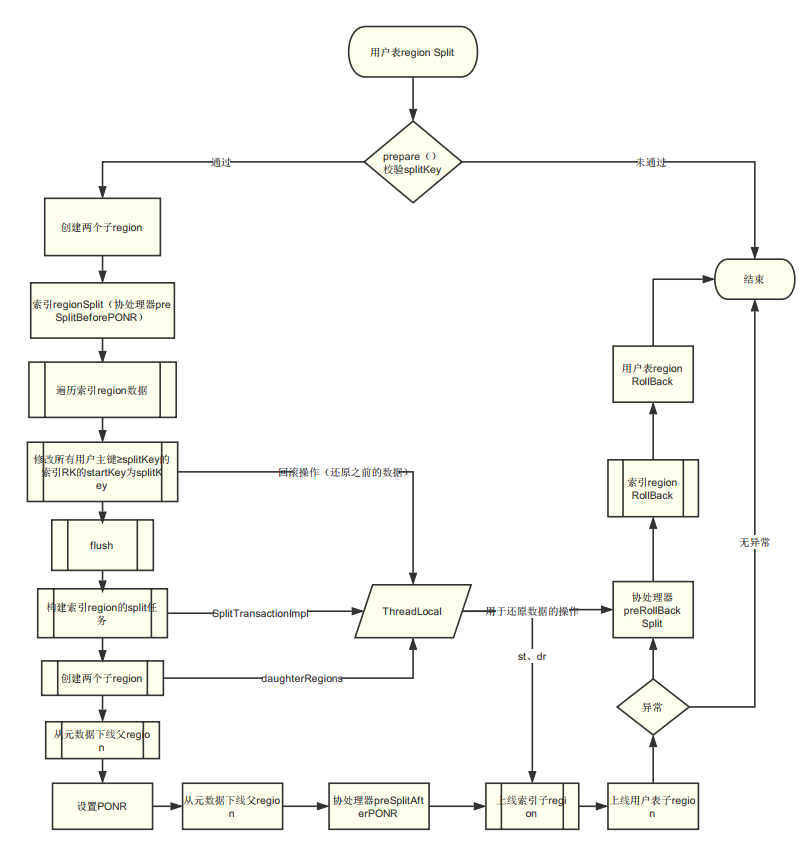
###### 2.3.7如果遇到异常，执行用户region的RollBack

###### 2.3.8协处理器钩子：preRollBackSplit

（1）恢复索引region中的数据

（2）执行索引region的RollBack

**流程图**

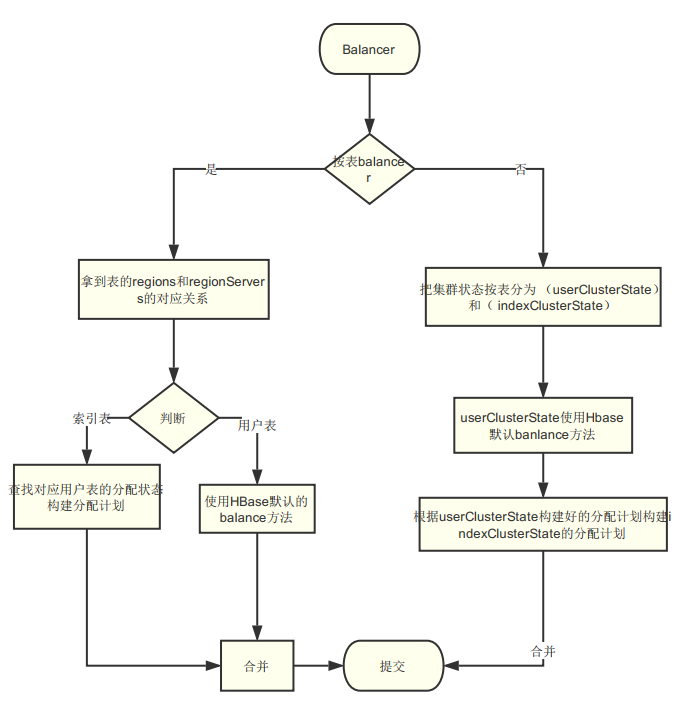


##### 2.4集群负载均衡 Balancer

###### 2.4.1 用户表的region遵循HBase默认的负载均衡策略

###### 2.4.2 索引表的region根据数据表的region的分配状态或计划来进行移动。

**流程图**



### 代码部分

代码位置： 项目：MarsBase-hbase-handler

包名：cn.com.bonc.hbase.handler.index

#### Balancer

位置：cn.com.bonc.hbase.handler.index.balancer

修改了HBase的负载均衡策略加载类—LoadBalancerFactory.java



说明：需要在配置文件hbase-site.xml中配置上边代码中的两个参数，

‘hbase.index.loadbalancer.class’ 表示基于二级索引的负载均衡策略的全路径类名

‘hbase.use.secondary.index’ 当参数为True时表示使用用户指定的负载均衡策略。

新增了类IndexLoadBalancer.java实现了LoadBalancer类中的一些方法。

下面对本类中主要的方法进行剖析：



Balancer可以设置是按表balancer还是按整个集群所有表blancer，所以这里需要分情况处理：





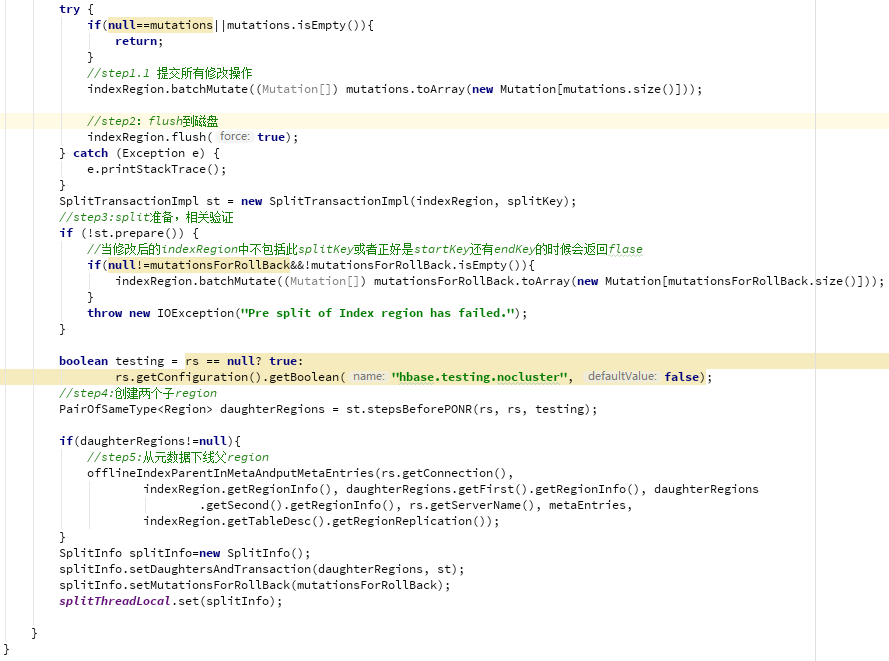
#### 用户表Split触发索引表的Split

代码位置： cn.com.bonc.hbase.handler.index.coprocessor.MindexRegionObserver



协处理器RegionObserver中的方法-preSplitBeforePONR,是执行split操作时调用的钩子方法，调用时机是达到PONR之前。

PONR--不可回滚点



#### 索引数据一致性保证WAL相关代码

见HBase-Index的文档

* 1. 索引实时写入和删除

这部分代码HBase—Index已交给屈箫负责。