# Split

# Region 概念

Region是表获取和分布的基本元素，由每个列族的一个Store组成。对象层级图如下：

Table (HBase table)

Region (Regions for the table)

Store (Store per ColumnFamily for each Region for the table)

MemStore (MemStore for each Store for each Region for the table)

StoreFile (StoreFiles for each Store for each Region for the table)

Block (Blocks within a StoreFile within a Store for each Region for the table)

# Region 大小

Region的大小是一个棘手的问题，需要考量如下几个因素。

* Region是HBase中分布式存储和负载均衡的最小单元。不同Region分布到不同RegionServer上，但并不是存储的最小单元。
* Region由一个或者多个Store组成，每个store保存一个columns family，每个Strore又由一个memStore和0至多个StoreFile 组成。memStore存储在内存中， StoreFile存储在HDFS上。
* HBase通过将region切分在许多机器上实现分布式。也就是说，你如果有16GB的数据，只分了2个region， 你却有20台机器，有18台就浪费了。
* region数目太多就会造成性能下降，现在比以前好多了。但是对于同样大小的数据，700个region比3000个要好。
* region数目太少就会妨碍可扩展性，降低并行能力。有的时候导致压力不够分散。这就是为什么，你向一个10节点的HBase集群导入200MB的数据，大部分的节点是idle的。
* RegionServer中1个region和10个region索引需要的内存量没有太多的差别。

最好是使用默认的配置，可以把热的表配小一点(或者受到split热点的region把压力分散到集群中)。如果你的cell的大小比较大(100KB或更大)，就可以把region的大小调到1GB。region的最大大小在hbase配置文件中定义：

**<property>**

**<name>**hbase.hregion.max.filesize**</name>**

**<value>**10 \* 1024 \* 1024 \* 1024**</value>**

**</property>**

说明：

1. 当region中的StoreFile大小超过了上面配置的值的时候，该region就会被拆分，具体的拆分策略见下文。
2. 上面的值也可以针对每个表单独设置，例如在hbase shell中设置：

**create** 't','f'

disable 't'

**alter** 't', **METHOD** => 'table\_att', MAX\_FILESIZE => '134217728'

enable 't'

# Region 拆分策略

Region的分割操作是不可见的，因为Master不会参与其中。RegionServer拆分region的步骤是，先将该region下线，然后拆分，将其子region加入到META元信息中，再将他们加入到原本的RegionServer中，最后汇报Master。

执行split的线程是CompactSplitThread。

## 自定义拆分策略

可以通过设置RegionSplitPolicy的实现类来指定拆分策略，RegionSplitPolicy类的实现类有：

ConstantSizeRegionSplitPolicy

IncreasingToUpperBoundRegionSplitPolicy

DelimitedKeyPrefixRegionSplitPolicy

KeyPrefixRegionSplitPolicy

对于split，并不是设置了hbase.hregion.max.filesize（默认10G）为很大就保证不split了，需要有以下的算法：

* IncreasingToUpperBoundRegionSplitPolicy，**0.94.0默认region split策略**。根据公式min(r^2\*flushSize，maxFileSize)确定split的maxFileSize，其中r为在线region个数，maxFileSize由hbase.hregion.max.filesize指定。
* ConstantSizeRegionSplitPolicy，仅仅当region大小超过常量值（hbase.hregion.max.filesize大小）时，才进行拆分。
* DelimitedKeyPrefixRegionSplitPolicy，保证以分隔符前面的前缀为splitPoint，保证相同RowKey前缀的数据在一个Region中
* KeyPrefixRegionSplitPolicy，保证具有相同前缀的row在一个region中（要求设计中前缀具有同样长度）。指定rowkey前缀位数划分region，通过读取table的prefix\_split\_key\_policy.prefix\_length属性，该属性为数字类型，表示前缀长度，在进行split时，按此长度对splitPoint进行截取。此种策略比较适合固定前缀的rowkey。当table中没有设置该属性，或其属性不为Integer类型时，指定此策略效果等同与使用IncreasingToUpperBoundRegionSplitPolicy。

### IncreasingToUpperBoundRegionSplitPolicy

这是**0.94.0默认region split策略**。根据根据公式min(r^2\*flushSize，maxFileSize)确定split的maxFileSize，这里假设flushSize为128M：

第一次拆分大小为：min(10G，1\*1\*128M)=128M

第二次拆分大小为：min(10G，3\*3\*128M)=1152M

第三次拆分大小为：min(10G，5\*5\*128M)=3200M

第四次拆分大小为：min(10G，7\*7\*128M)=6272M

第五次拆分大小为：min(10G，9\*9\*128M)=10G

第五次拆分大小为：min(10G，11\*11\*128M)=10G

可以看到，只有在第四次之后的拆分大小才为10G

## 配置拆分策略

你可以在hbase配置文件中定义全局的拆分策略，设置hbase.regionserver.region.split.policy的值即可，也可以在创建和修改表时候指定：

*// 更新现有表的split策略*

HBaseAdmin admin = **new** HBaseAdmin( conf);

HTable hTable = **new** HTable( conf, "test" );

HTableDescriptor htd = hTable.getTableDescriptor();

HTableDescriptor newHtd = **new** HTableDescriptor(htd);

newHtd.setValue(HTableDescriptor. SPLIT\_POLICY, KeyPrefixRegionSplitPolicy.class .getName());*// 指定策略*

newHtd.setValue("prefix\_split\_key\_policy.prefix\_length", "2");

newHtd.setValue("MEMSTORE\_FLUSHSIZE", "5242880"); *// 5M*

admin.disableTable( "test");

admin.modifyTable(Bytes. toBytes("test"), newHtd);

admin.enableTable( "test");

说明：

1. 上面的不同策略可以在不同的业务场景下使用，特别是第三种和第四种一般关注和使用的比较少。
2. 如果想关闭自动拆分改为手动拆分，建议同时修改hbase.hregion.max.filesize和hbase.regionserver.region.split.policy值。

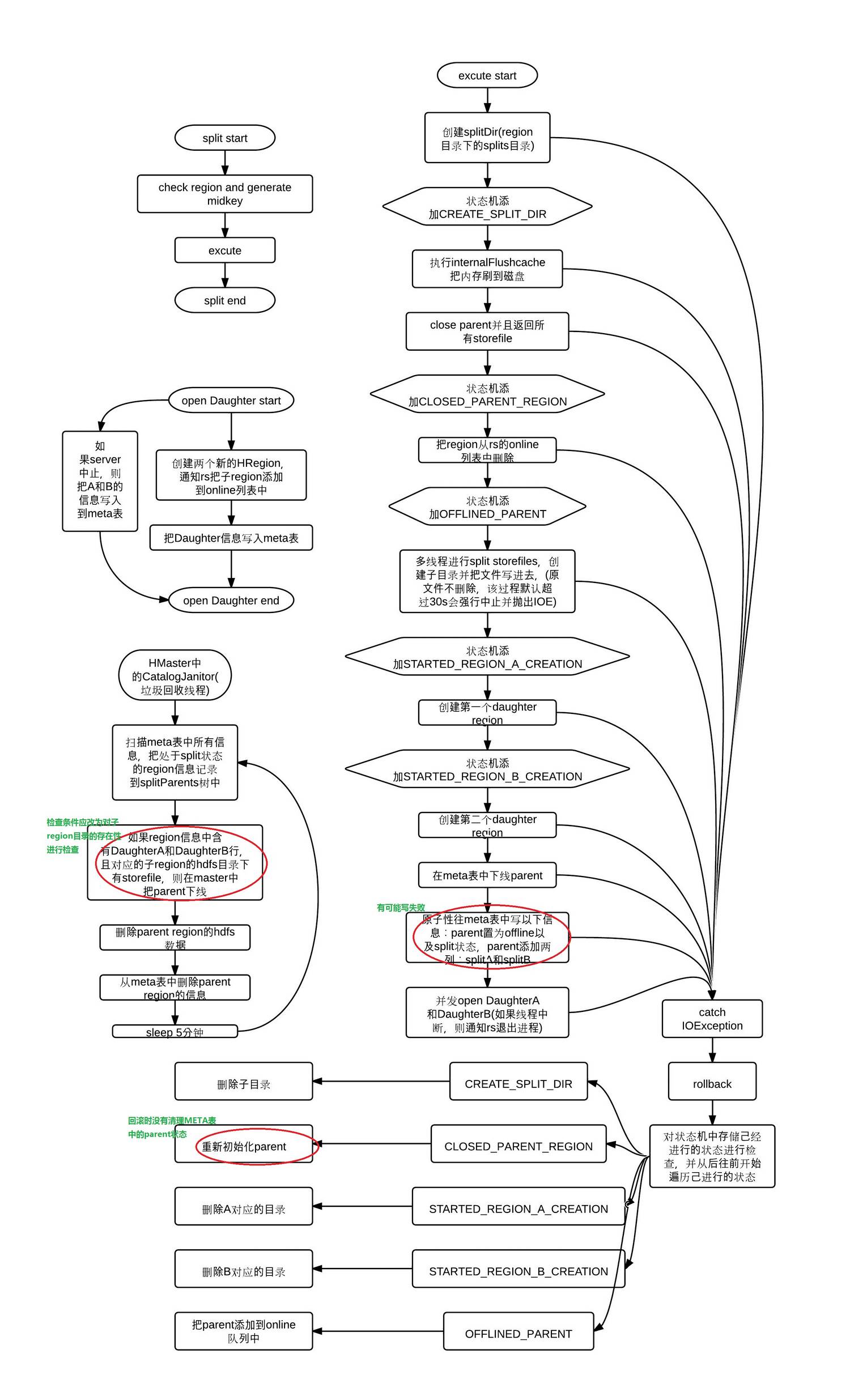
http://koven2049.iteye.com/blog/1199519

hbase的稳定性是近期社区的重要关注点，毕竟稳定的系统才能被推广开来，这里有几次稳定性故障和大家分享。   
    第一次生产故障的现象及原因   
    现象：

**Java代码  [收藏代码](javascript:void())**

1. 1 hbase发现无法写入
2. 2 通过hbck检测发现.META.表中出现空洞，具体log是：;Chain of regions in table ...  is broken; edges does not contain ...
3. 3 此时读写失败

修复方法：直接使用check\_meta.rb重新生成.META.表并修补空洞，但是会引起数据丢失。因为引起该空洞的原因是某个region的parent和daughter都被删掉了   
    查找故障过程非常复杂，具体就不提了，都是内伤啊...   
    故障原因需要从split的原理说起：   
    split是一个分布式的事务过程，由于分布式的复杂性，在每一步都有可能发生异常中止，因此每进行一步就要记录一下当前的状态。如果出错了，就根据己经进行的状态来进行对应的回滚操作。这个记录状态的变量在代码里体现为JournalEntry 。

于是split的过程是这样的：(见图)   
   

* 1 创建splitDir(region目录下的splits目录)
* 2 状态机添加CREATE\_SPLIT\_DIR
* 3 执行internalFlushcache把内存刷到磁盘
* 4 close parent并且返回所有storefile
* 5 状态机添加CLOSED\_PARENT\_REGION
* 6 把region从rs的online列表中删除
* 7 状态机添加OFFLINED\_PARENT
* 8 多线程进行split storefiles，创建子目录并把文件写进去，(原文件不删除，该过程默认超过30s会强行中止并抛出IOE)
* 9 状态机添加STARTED\_REGION\_A\_CREATION
* 10 创建第一个daughter region
* 11 状态机添加STARTED\_REGION\_B\_CREATION
* 12 创建第二个daughter region
* 13 在meta表中下线parent
* 14 原子性往meta表中写以下信息：parent置为offline以及split状态，parent添加两列：splitA和splitB
* 15 并发open DaughterA和DaughterB(如果线程中断，则通知rs退出进程)
* 16 在open期间，如果server中止，则先把A和B的信息写入到meta表中再跳过以下过程
* 17 创建两个新的HRegion，通知rs把子region添加到online列表中
* 18 把Daughter信息写入meta表

    当以上过程中任何一步抛出异常时，regionserver会进入回滚逻辑(rollback)：   
    对状态机中存储己经进行的状态进行检查，并从后往前开始遍历己进行的状态：

* CREATE\_SPLIT\_DIR:删除子目录
* CLOSED\_PARENT\_REGION:重新初始化parent
* STARTED\_REGION\_A\_CREATION:删除A对应的目录
* STARTED\_REGION\_B\_CREATION:删除B对应的目录
* OFFLINED\_PARENT:把parent添加到online队列中

    了解了以上过程，我们来假设以下两种场景：   
    **场景1** 如果split期间如果在第4步(close parent并且返回所有storefile)结束后，或者执行过程中发生异常，那么此时会发生什么呢？   
    状态机里的值是CREATE\_SPLIT\_DIR，因此会删除子目录，但由于parent己经被close或者处于closing状态了，那么就是父子region都没有在线，此时无法读写该region对应的数据   
    场景1还比较好解决，比如直接下线该台rs再重启就能解决了。当然对于在线业务来讲，这个还是不能接受的，因为系统不可用时间太长了。于是需要用到HBASE-4563这个patch来解决它   
    它的原理很简单：把CLOSED\_PARENT\_REGION状态放到close parent前面即可   
  
    **场景2** 如果split期间在第14步期间或以后发生异常。那此时meta表己经将parent下线了，回滚时却没有将parent在meta中的状态改回来，而是删除了子region的目录以及open parent   
(注意在open parent的时候并不会修改meta中的offline及split值)。于是该region就不再上线了，形成了空洞。不过这个时候客户端如果cache了该region的地址，那暂时还不会报错，   
因为regionserver己经把它重新上线了，数据暂时还能读到。   
    但是严重的问题是：master有个垃圾清理线程(CatalogJanitor)会定期对meta表做扫描，默认5min一次。它会扫描meta表中split和offline状态为true的那些region，检查是否有子region指向它，如果没有的话，它会认为这个region是己经split成功的，于是会将它从meta表以及hdfs上删掉。检查子region是否指向它是检查是否有子目录存在并且不为空。在场景2中显然子目录己经不再存在了，因此被认为是split成功的region，于是数据被master删掉...   
    这里的根本原因是没有对写meta这个状态进行记录以及回滚，并且master检查子region是否存在的条件太过简单，需要做调整。具体可参见HBASE-3872以及HBASE-4562(3872试图解决这个问题，但是没有解决掉。4562进行了进一步的处理)   
    修复的原理就是增加一个状态来记录，而当回滚发现这个问题时，让这台regionserver自己挂掉。这样在master恢复它的时候，会执行fixDaughter的逻辑，这个逻辑会完整地恢复这个region，让split成功，子region上线。而CatalogJanitor的逻辑也进行了调整，对数据进行更加严格的检查和保护，避免随意删除数据。   
    场景2是个非常严重的bug，推荐大家都升级一下。因为在复杂的网络环境中这个异常还是比较容易出现的。   
  
    第二次生产故障的现象及原因   
    现象：

**Java代码  [收藏代码](javascript:void())**

1. 1 用户发现tps有下降，且部分写入不正常。
2. 2 通过hbck检查到集群有"Chain of regions in table …contains less elements than are listed in META; visited=”问题存在，意即META表中某些region出错，此时若用户有新的写入，则新的写入有可能会数据丢失。
3. 3 通过1个多小时的修复，仍然没有将该集群状态无损还原。原因是出现了两个region服务同一段数据

    修复方法：找到start\_key和end\_key相同的几个region，把它们的从hdfs上删除掉。然后用add\_table重建meta表（会导致丢失数据）   
    这个过程也是一个hbase的bug产生的，这个bug来自于重启过程。复现问题也很容易，进行以下几步即可复现：   
    1 找到一台正在split的region所在的rs   
    2 kill掉该台rs   
    3 重启整个集群或master进行切换   
    原因分析：   
    当hbase的master在主从切换或者重启的时候，有一个步骤是切换之后的master需要对原来所有的挂掉的regionserver上的region进行processDeadRegion，即重新上线。   
    该过程在0.90.4之前存在一个bug，即会把meta表中所有处在split期间的region也进行处理，虽然region在meta表中处于split状态并不能证明它己经split结束还是正在split（要对split状态进行标记还是很复杂的，因此目前的代码还没有对split状态进行记录，只能通过一些辅助手段，比如检查子region的状态来说明region是否处于split状态），但是万一它己经split结束的话是绝对不应该上线的。因此有可能一个region己经split结束，但它在这个处理过程中又被新起的master上线了，这就导致父子region同时服务了。而父region上线后又有可能继续split，导致状况更加糟糕，同一段数据被两个region服务，等等。   
    正确的处理办法是在重启时检查这些region的子region状态，具体检查方案在hbase-0.90.4中己经给出，可参见HBASE-3946。注意：打上3946的patch以后，还必须要打上3995的patch，否则单元测试无法通过。