# 1.宕机

## 观点1

**总结**

Hbase挂掉的可能性有很多，主要由zk或者hdfs的问题导致，因此zk、hdfs的可用对于hbase极其重要，关于zk：

1.zk如果停止了服务则在很多时候会导致master、rs挂掉，hbase集群基本上就失去了服务的能力，因此zk一定要是稳定可靠的，当client已经于rs建立了链接，这时zk挂掉，如果不进行split等小数与zk交互失败会导致触发rs的abort()的操作时rs还是可以提供服务的；

2.如果rs/master进行了长时间的gc或者改动了服务器时间，导致出现zk的session超时会导致rs/master停止服务，目前已经出现了2次因为服务器时间变化导致hbase停止服务的事故;

3.别轻易人为改变zk的hbase节点数据，master/rs在进行很多操作时会比较依赖zk的数据，如果发现不符合预期可能会导致master/rs停止服务，尤其是master。

    Master通过ZK知道RS是否可用，一般情况下RS在停止服务时均会正常退出，在正常退出时会从ZK中删除/hbase/rs/$regionserver的节点，Master会监听该节点的被删除，从而较快的(速度取决于所有region关闭时间)对该RS负责的region进行重新分配，如果是强制退出，比如 kill -9或者出现HRegionServer挂掉的第8条时则只有等待ZK的session超时时才会删除RS在ZK的节点（RS在ZK中添加节点时采用的是CreateMode.EPHEMERAL模式，该模式创建的节点会在session关闭时自动删除），那时Master才会进行重新assign。

    Kill RS的进程也是正常退出（不能使用kill -9强制退出），RS使用Runtime的addShutdownHook方法注册了jvm关闭钩子，在关闭钩子中会执行RS的退出逻辑，实际上hbase-daemon.sh的停止RS就是采用kill。

**一、引言**  
本文记录了困扰团队两周的**HBase**随机宕机事件的解决方案，并回顾了JVM **GC**调优基础知识，供各位参考。  
**二、实验环境**  
16台虚拟机，每台4G内存，1核CPU，400G硬盘  
Ubuntu 14.04 LTS (GNU/Linux 3.13.0-29-generic x86\_64)  
  
CDH5.2.0套装（包括相应版本的Hadoop，HIVE，**Hbase**，Mahout，Sqoop，Zookeeper等）  
Java  1.7.0\_60  64-Bit Server  
**三、异常现场**  
在以上实验环境中执行计算任务，计算任务涉及HIVE、Mahout、**Hbase** bulkload、MapReduce，工作流驱动通过Shell脚本控制，整个任务执行过程涉及基础行为数据160万条，业务数据40万条。  
  
多次执行任务过程中反复随机出现以下各类异常，仅用文字描述，就不拷贝异常现场了，大家各自对号入座：  
1.**Hbase**的Regionserver进程随机挂掉（该异常几乎每次都发生，只是挂掉的Regionser节点不同）  
2.HMaster进程随机挂掉  
3.主备Namenode节点随机挂掉  
4.Zookeeper节点随机挂掉  
5.Zookeeper连接超时  
6.JVM **GC**睡眠时间过长  
7.datanode写入超时  
等等  
通过调研分析和调试，发现问题解决需从以下几个方面着手：  
1.**Hbase**的ZK连接超时相关参数调优：默认的ZK超时设置太短，一旦发生FULL **GC**，极其容易导致ZK连接超时；  
2.**Hbase**的JVM **GC**相关参数调优：可以通过**GC**调优获得更好的**GC**性能，减少单次**GC**的时间和FULL **GC**频率；  
3.ZK Server调优：这里指的是ZK的服务端调优，ZK客户端（比如**Hbase**的客户端）的ZK超时参数必须在服务端超时参数的范围内，否则ZK客户端设置的超时参数起不到效果；  
4.HDFS读写数据相关参数需调优；  
5.YARN针对各个节点分配资源参数调整：YARN需根据真实节点配置分配资源，之前的YARN配置为每个节点分配的资源都远大于真实虚拟机的硬件资源；  
6.集群规划需优化：之前的集群规划中，为了充分利用虚拟机资源，NameNode、NodeManager、DataNode，RegionServer会混用同一个节点，这样会导致这些关键的枢纽节点通信和内存压力过大，从而在计算压力较大时容易发生异常。正确的做法是将枢纽节点（NameNode，ResourceManager，HMaster）和数据+计算节点分开。  
**四、为了解决该问题而实施的各类配置及集群调整**  
**HBase**  
**hbase-site.xml**

1. <property>
2. <name>zookeeper.session.**timeout**</name>
3. <value>300000</value>
4. </property>
5. <property>
6. <name>**hbase**.zookeeper.property.tickTime</name>
7. <value>60000</value>
8. </property>
9. <property>
10. <name>**hbase**.hregion.memstroe.mslab.enable</name>
11. <value>true</value>
12. </property>
13. <property>
14. <name>**hbase**.zookeeper.property.maxClientCnxns</name>
15. <value>10000</value>
16. </property>
17. <property>
18. <name>**hbase**.client.scanner.**timeout**.period</name>
19. <value>240000</value>
20. </property>
21. <property>
22. <name>**hbase**.rpc.**timeout**</name>
23. <value>280000</value>
24. </property>
25. <property>
26. <name>**hbase**.hregion.max.filesize</name>
27. <value>107374182400</value>
28. </property>
29. <property>
30. <name>**hbase**.regionserver.handler.count</name>
31. <value>100</value>
32. </property>
33. <property>
34. <name>dfs.client.socket-**timeout**</name>
35. <value>300000</value>
36. <description>Down the DFS **timeout** from 60 to 10 seconds.</description>
37. </property>

*复制代码*

**hbase-env.sh**

1. export **HBASE**\_HEAPSIZE=2048M
2. export **HBASE**\_HOME=/home/fulong/**Hbase**/**hbase**-0.98.6-cdh5.2.0
3. export **HBASE**\_LOG\_DIR=${**HBASE**\_HOME}/logs
4. export **HBASE**\_OPTS="-server -Xms1g -Xmx1g -XX:NewRatio=2 -XX:PermSize=128m -XX:MaxPermSize=128m -verbose:**gc** -Xloggc:$**HBASE**\_HOME/logs/hbasegc.log -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UseParNewGC -XX:+CMSParallelRemarkEnabled -XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFraction=75 -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=$**HBASE**\_HOME/logs"

*复制代码*

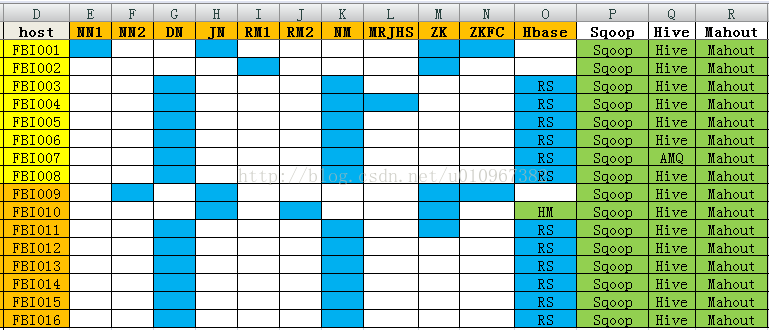
**zookeeper**  
  
**zoo.cfg**

1. syncLimit=10
2. #New in 3.3.0: the maximum session **timeout** in milliseconds that the server will allow the client to negotiate. Defaults to 20 times the tickTime.
3. maxSessionTimeout=300000
4. # the directory where the snapshot is stored.
5. # do not use /tmp for storage, /tmp here is just
6. # example sakes.
7. dataDir=/home/fulong/Zookeeper/CDH/zookdata
8. # the port at which the clients will connect
9. clientPort=2181  
   修改以下两个文件是为了跟踪ZK日志，ZK的默认日志查看不方便。  
   **log4j.properties**
10. zookeeper.root.logger=INFO,CONSOLE,ROLLINGFILE
11. zookeeper.console.threshold=INFO
12. zookeeper.log.dir=/home/fulong/Zookeeper/CDH/zooklogs
13. zookeeper.log.file=zookeeper.log
14. zookeeper.log.threshold=DEBUG
15. zookeeper.tracelog.dir=/home/fulong/Zookeeper/CDH/zooklogs
16. zookeeper.tracelog.file=zookeeper\_trace.log
17. log4j.appender.ROLLINGFILE=org.apache.log4j.RollingFileAppender
18. log4j.appender.ROLLINGFILE.Threshold=${zookeeper.log.threshold}
19. log4j.appender.ROLLINGFILE.File=${zookeeper.log.dir}/${zookeeper.log.file}
20. # Max log file size of 10MB
21. log4j.appender.ROLLINGFILE.MaxFileSize=50MB
22. <b>zkEnv.sh</b>
23. if [ "x${ZOO\_LOG4J\_PROP}" = "x" ]
24. then
25. ZOO\_LOG4J\_PROP="INFO,CONSOLE,ROLLINGFILE"
26. fi
27. 备注：修改完以上两个文件后，并没有如愿的见到ZK的Log4j日志文件，原因待进一步调研。  
      
    **HDFS**  
      
    **hdfs-site.xml**
28. <property>
29. <name>dfs.datanode.socket.write.**timeout**</name>
30. <value>600000</value>
31. </property>
32. <property>
33. <name>dfs.client.socket-**timeout**</name>
34. <value>300000</value>
35. </property>
36. <property>
37. <name>dfs.datanode.max.xcievers</name>
38. <value>4096</value>
39. </property>

*复制代码*

**YARN**  
  
**yarn-site.xml**

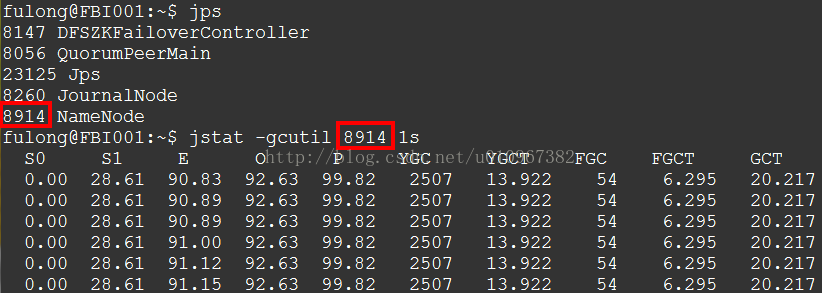
1. <property>
2. <name>yarn.scheduler.minimum-allocation-mb</name>
3. <value>512</value>
4. </property>
5. <property>
6. <name>yarn.scheduler.fair.user-as-default-queue</name>
7. <value>false</value>
8. </property>
9. <property>
10. <name>yarn.resourcemanager.zk-**timeout**-ms</name>
11. <value>120000</value>
12. </property>
13. <property>
14. <name>yarn.nodemanager.resource.memory-mb</name>
15. <value>3072</value>
16. </property>
17. <property>
18. <name>yarn.scheduler.minimum-allocation-mb</name>
19. <value>128</value>
20. </property>
21. <property>
22. <name>yarn.scheduler.maximum-allocation-mb</name>
23. <value>3072</value>
24. </property>
25. <property>
26. <name>yarn.nodemanager.resource.cpu-vcores</name>
27. <value>1</value>
28. </property>
29. <property>
30. <name>yarn.scheduler.maximum-allocation-vcores</name>
31. <value>1</value>
32. </property>
33. <property>
34. <name>yarn.nodemanager.container-monitor.interval-ms</name>
35. <value>300000</value>
36. </property>

*复制代码*  
**集群调整**  
NN Active, NN Standby, RM Active, RM Standby 所在节点均不运行DN,NM,RS  
DN、NM、RS所在节点一一对应。  
调整过后的布局：  
  
说明：如果遇到类似问题，可以重点参考以上配置项，但具体数值请根据具体情况具体分析。  
  
**五、补充回顾--JVM GC**  
  
调优过程中**GC**问题十分明显，未调优之前，频繁出现3~6min的Full **GC**时间，调优过后，**GC**时间能控制在20s以内。  
**GC**调优对于Hadoop集群十分重要，是必须掌握的基础知识，在此简单记录。  
  
完整知识叙述请参见：  
英文原文：<http://www.cubrid.org/blog/dev-platform/understanding-java-garbage-collection/>  
译文地址：<http://www.importnew.com/1993.html>  
  
以及Oracle网站上的相关说明：  
<http://www.oracle.com/technetwork/java/javase/tech/exactoptions-jsp-141536.html>  
<http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html>  
<http://www.oracle.com/technetwork/java/javase/tech/index-jsp-140228.html>  
  
以下仅仅描述最重要的基础知识：  
  
  
       每个JVM主要内存区域分为两部分：Permanent Space 和 Heap Space。  
  
       Permanent即持久代（Permanent Generation），主要存放的是Java类定义信息，与垃圾收集器要收集的Java对象关系不大。  
Heap={Old+NEW={Eden,Survivor 0 ,Survivor 1}}，Old即老年代（Old Generation），New即年轻代（Young Generation）。年轻代（Young Generation）用来保存那些第一次被创建的对象，它进一步被分为三个空间：一个伊甸园空间（Eden ），两个幸存者空间（Survivor ）。老年代和年轻代的划分对垃圾收集影响比较大。  
  
  
      基本的执行顺序如下：

1. 绝大多数刚刚被创建的对象会存放在伊甸园（Eden ）空间。
2. 在伊甸园（Eden ）空间执行了第一次**GC**之后，存活的对象被移动到其中一个幸存者空间。
3. 此后，在伊甸园空间执行**GC**之后，存活的对象会被堆积在同一个幸存者空间。
4. 当一个幸存者空间饱和，还在存活的对象会被移动到另一个幸存者空间。之后会清空已经饱和的那个幸存者空间。
5. 在以上的步骤中重复几次依然存活的对象，就会被移动到老年代。

      截止目前版本，Java可配置的垃圾收集器有5种类型：

1. Serial **GC**
2. Parallel **GC**
3. Parallel Old **GC** (Parallel Compacting **GC**)
4. Concurrent Mark & Sweep **GC**  (or “CMS”)
5. Garbage First (G1) **GC**

      其中用得较多比较成熟的是CMS。  
  
      我们可以通过各种工具来监控JVM **GC**情况，比较简单直观的是jstat。比如我们要监控NameNode的**GC**情况，可以先用jps查看到进程号，然后通过jstat查看**gc**情况：  
   
       jstat -gcutil后面跟的参数是JVM进程号，1s是数据刷新时间。命令输出的每一列依次是：幸存者0的空间占用比例，幸存者1的空间占用比例，伊甸园空间的占用比例，老年代空间的占用比例，持久代空间的占用比例，年轻代（S0+S1+E）发生**GC**的次数，年轻代发生**GC**的总时间（秒为单位），FULL **GC**发生的次数，FULL **GC**发生的总时间（秒为单位），**GC**消耗的总时间（秒为单位）。  
  
       最后附上我们本次调优**hbase**设置的JVM参数：  
  
-server                      //开启java服务器模式  
  
  
-Xms1g                    //最小最大堆内存  
-Xmx1g   
  
  
-XX:NewRatio=2        //老年代空间：年轻代空间=2  
  
  
-XX:PermSize=128m        //初始和最大持久代空间，感觉可以进一步缩减，目前观察持久代空间使用没超过30%  
-XX:MaxPermSize=128m  
  
  
-Xloggc:$**HBASE**\_HOME/logs/hbasegc.log   //开启**gc**日志功能，便于调试，基本不会影响性能  
-XX:+PrintGCDetails   
-XX:+PrintGCTimeStamps  
  
  
-XX:+UseParNewGC                                         //开启CMS垃圾回收期  
-XX:+CMSParallelRemarkEnabled   
-XX:+UseConcMarkSweepGC   
-XX:CMSInitiatingOccupancyFraction=75  
  
  
本文转自<http://blog.csdn.net/u010967382/article/details/42394031>

# 2.HBase的long GC与 Zookeeper lease expired的权衡

问题和现象：

这是一个连锁反应：

1）RegionServer在遇到"Stop-The-World" GC时，会停止一切工作，这样与Zookeeper保持的心跳，就会停止。

2）Zookeeper在没有收到注册节点的心跳时，就会删除对应rs对应节点。

3）HMaster的ServerManager会发现这个RegionServer出现了问题，然后交由ServerShutdownHandler处理。

4）HMaster的SplitLogManager和RegionServer的SplitLogWorker组成Master-Slave结构，对HMaster

认定Dead的RegionServer的节点进行处理。

HLOG => {按照Region分割成不同的edits文件}，具体内容可以查看：http://www.cloudera.com/blog/2012/07/hbase-log-splitting/

5）HMaster的AssignmentManager把HLog处理完成的Region分配给一个RegionServer，RegionServer在接到Open Region的请求之后，利用分割出来的edits，实现HLog Replay，将其加载到MemStore，并且flush成一个文件。

6）RegionServer上的所有Region在其它RegionServer上提供服务。

7）RegionServer FullGC结束，然后正常与HMaster进行心跳tryRegionServerReport()，后来收到了一个YouAreDeadException，接到命令后，就开始关闭各种线程，然后退出。

以上只是对HBase一个现象的描述，如果正常的情况下，这样的处理是没有问题的。但是，我们在维护的Hbase集群却出现了一个让我们很郁闷的问题。

有应用报出Get、Scan某些数据不见了，查看Region下的RegionServer报出：

DFS Read: java.io.FileNotFoundException: File does not exist: /user/hbase/hbase/OTHER/9cc2c3d381e6891d1798d2cf4fe23859/dat/48bf8d0bbdae4b8b85f66da6f81b66aa

也就是该表格下的dat数据丢失，检查了NameNode查看该文件的情况，发现该数据被原RegionServer的DFSClient删除了，是因为原RegionServer在Long FullGC之后，做了一次minor Compaction，把两个StoreFile文件合并成了一个。而在这个之前，该Region已经被一个新的RegionServer所服务着，这样这个新的RegionServer会认为没有进行Compaction，还是按照接手该Region时的情况进行读取数据，于是，悲剧就产生了。

具体描述：

出问题Region：RegionA

FullGC的RegionServer： RegionServer1

接管RegionA的RegionServer: RegionServer2

RegionServer1 --------------Full---GC------------------->compaction--(delete 原 dat/48bf\*\*\*)->You Are Dead--->Exit

RegionServer2 -------------------------> 接管RegionA---->请求出错

整体上虽然数据没有丢失，但是仍然给线上系统带来一定的影响。分析了整个过程之后，预防这类问题出现应该

从如下几点入手：

1、相关参数

<property>

<name>hbase.regionserver.lease.period</name>

<value>240000</value>

<description>HRegion server lease period in milliseconds. Default is 60 seconds. Clients must report in within this period else they are considered dead.</description>

</property>

hbase.regionserver.lease.period的设置是Tradeoff，因为在HBase Client操作过程中，如果RegionServer的响应速度过慢或者出现了Long FullGC，超过了设置值，则抛出lease expired。

ps：hbase.regionserver.lease.period的设置要和hbase.rpc.timeout保持一致，最好hbase.rpc.timeout的值略大于hbase.regionserver.lease.period

<property>

<name>zookeeper.session.timeout</name>

<value>90000</value>

<description>Default 3 minutes.ZooKeeper session timeout. HBase passes this to the zk quorum as suggested maximum time for a session (This setting becomes zookeeper's 'maxSessionTimeout'). See http://hadoop.apache.org/zookeeper/docs/current/zookeeperProgrammers.html#ch\_zkSessions "The client sends a requested timeout, the server responds with the timeout that it can give the client. " In milliseconds. </description>

</property>

zookeeper.session.timeout指定zk Client与zk Server Quorum之间的会话的最长时间，RegionServerTracker监听rs的znode的变化，超过session time out的值，就认定该RegionServer出错了，然后通过ServerManager的expireServer将其加入Dead RegionServer Lists中。这样等RegionServer发送心跳给HMaster时，发现该RegionServer位于Dead RegionServer Lists中，会抛出YouAreDeadException。然后，RegionServer会开始执行shutdown系统操作。

注意：这里设置的zookeeper.session.timeout与zookeeper设置的ticktime共同影响session的生存周期，一般而言，sessionTimeOut= Min(应用设置的zookeeper.session.timeout，ticktime\*20)

2、tryRegionServerReport()的频率设置高一些。默认间隔时间为：

this.msgInterval = conf.getInt("hbase.regionserver.msginterval", 3 \* 1000);设置成1\*1000ms，可以更快地同步HMaster上RegionServer状态，尽量避免延迟阶段造成的数据不一致。注意，如果是RegionServer的个数超过200，该值不要太小，避免HMaster的压力过大。

3、在compaction删除文件时，tryRegionServerReport一下，保证没有出现已经被认定“Dead”但仍然工作的情况。在任何删除操作之前，首先report一下HMaster，检查是否已经出现YouAreDeadException，然后恢复之前的文件。

3.案例

# 3.Region Server 宕机后,处理过程

对于分布式数据库来说，容错处理是非常重要的一个部分。RegionServer是HBase系统中存在最多的节点，所以对于RegionServer的容错处理对于HBase来说至关重要。本文对RegionServer的容错处理进行Step by Step的分析，希望能解释清除整个过程并加以点评。

我们假设在HBase运行的过程中有一个RegionServer突然Crash, 基于这个场景进行分析。

1. RegionServer Crash了

Crash的原因可能有很多种，程序自身挂掉，OS挂掉，网络断掉，电源断掉，等等。但从MasterServer的角度看来只有一种现象，那就是RegionServer在Zookeeper上面注册的Node消失了。我们知道当RegionServer启动的时候会产生一个StartCode，并在Zookeeper上面注册一个EPHEMERAL类型的节点。

一旦RS和ZK之间的通信消失，EPHEMERAL的节点就会被自动删除。而MasterServer则会捕获这个Node消失的事件。

捕获这个事件之后，所有的事情就交给ServerManager.expireServer()来处理了。

2. ServerManager处理

首先ServerManager会更新自己的列表，包括deadserver list, online server list 以及 server connection list. 当然在我们试图Stop整个Cluster的时候也会收到同样的请求，不予理睬就好了。

并不是所有的RS Crash都做相同的处理，有些RS比较特殊，他们正在管理-ROOT-表或者.META.表。如果我们重新分配Region, 就需要修改.META.表，如果要访问.META.表先要查询-ROOT-表 （参见以前的文章，client如何路由到正确的RS）。问题是现在管理-ROOT-表或者.META.表的RS挂了，显然第一要务是让-ROOT-和.META.有所归属，能够正常的对他们进行读写。

接下来ServerManager通过传递Event的方式将任务交给Excutor线程来处理，具体调用MetaServerShutdownHandler或者ServerShutdownHandler的process函数。

3. ServerShutdownHandler处理

MetaServerShutdownHandler是继承于ServerShutdownHandler的，除了打上META的标志以外其他都一样，所以我们只介绍ServerShutdownHandler的处理。

处理的第一步是Split HLog。当RS Crash的时候所有MemCache里面的内容都会被丢掉，这些内容还没有来得及Flush到HFile里面。感谢WAL机制，所有的内容都可以在HLog File里面找到。问题是原来管理这些HLog文件的RS已经挂掉了，需要将这些HLog交给新的RS去处理。往往这些HLog不会交给同一个新的RS去处理，因为HLog可能包含多个Region的内容，而这些Region可能会分配给不同的RS。这样看来最好的方式是让HLog里面的每一个Entry跟随Region，Region被分配给哪个RS，就让那个RS来处理这个Entry。事实上MasterServer也确实是这么做的。具体步骤如下：

1) 从Crash的RS的HLog目录下读取每个HLog文件

2) 根据文件中每个Entry所隶属的Region找到Region文件的存储目录。将Entry写到一个叫做 “recovered.edits” 的文件夹中。Entry写入的格式依然是HLog的格式。

3) 将Region分配给某个具体的RS，剩下的任务由RS处理

4. Assign RegionServer

前面有提到过，如果Crash的RS正在Handle -ROOT-或者.META.，需要特殊处理。特殊处理的方式就是先assign -ROOT-和.META.，并等待他们online。这样可以保证后需的assign工作。

具体Assign的过程可以单独用一章来讲，这里不做详细介绍。

5. RegionServer加载新的Region

一旦某个RS被assign了一个新的Region, 它就会试图加载这个新Region. 在加载的过程中RS会查看 “recovered.edits” 目录，试图从HLog中恢复丢失的数据。具体过程如下：

1) 扫描目录中的每个HLog文件

2) 跳过那些Sequence ID小于Region MAX Seq ID的Entry，因为这些Entry已经在HFile里面了。

3) 找到丢失的Entry并写入MemStore

4) Flush MemStore到HFile

5) 删除目录下的文件

具体可以参考 HRegion.replayRecoveredEdits

6. 分析后的思考

写到这里整个过程已经基本上结束了，但除了分析过程，还有许多我们可以思考的地方。

1) 恢复一个RegionServer需要多少时间？

等待ZK Node消失 + Split Log + Assign Region + Recover HLog +Region Online

这是从正常情况看来需要花费的时间，Log越多，Region越多，需要花费的时间越长。别忘了如果RS正在Handle -ROOT-或者.META.，需要的时间会更多。

2) RegionServer的问题真的可以都被侦测吗？

如果RS已经停止服务，但依然存活，ZK Node 就不会消失，这种情况应该发生过。目前只能作为Bug去修理。也许额外的Monitor是一个弥补的办法（定期Scan什么的）。