HBase 读写性能优化

**写（put）**

一、介绍

[**Hbase**](http://lib.csdn.net/base/hbase)的客户端包中集中了CURD操作，用户可以通过其中不同种类的API进行CURD操作。[**hbase**](http://lib.csdn.net/base/hbase)数据插入使用Put对象，Put对象在进行数据插入时，首先会向Hbase集群发送一个RPC请求，得到响应后将Put类中的数据通过序列化的方式传给HBase集群，数据先顺序写入HLog，再写入对应的缓存Memstore，当Memstore中数据大小达到一定阈值(128M)之后，系统会异步将Memstore中数据flush到HDFS形成小文件。

二、数据插入详解

HBase客户端拥有多重方式进行数据插入，通过调整不同的属性从而实现不同插入方式。

1、单行插入：put(Put p)

单行插入即每一次插入一行数据

**[java]** [view plain](http://blog.csdn.net/u011518678/article/details/50790193) [copy](http://blog.csdn.net/u011518678/article/details/50790193)

1. **public** **void** put(String tableName,String rowKey,String family,String column,String value)
2. {
3. Configuration conf=init();
4. **try** {
5. HTable table=**new** HTable(conf,TableName.valueOf(tableName));
6. HBaseAdmin admin=**new** HBaseAdmin(conf);
7. //判断表是否存在，如果不存在进行创建
8. **if**(!admin.tableExists(Bytes.toBytes(tableName)))
9. {
10. HTableDescriptor tableDescriptor=**new** HTableDescriptor(Bytes.toBytes(tableName));
11. HColumnDescriptor columnDescriptor=**new** HColumnDescriptor(Bytes.toBytes(family));
12. tableDescriptor.addFamily(columnDescriptor);
13. admin.createTable(tableDescriptor);
14. }
15. table.setAutoFlush(**true**);
16. //进行数据插入
17. Put put=**new** Put(Bytes.toBytes(rowKey));
18. put.add(Bytes.toBytes(family),Bytes.toBytes(column),Bytes.toBytes(value));
19. table.put(put);
20. table.close();
21. } **catch** (IOException e) {
22. // TODO Auto-generated catch block
23. e.printStackTrace();
24. }
25. }

HTable.put(Put p)方法想表中添加一行数据。在此过程中会发送一次RPC操作进行请求，并将Put中的数据序列化以后传送给相应的服务器进行数据插入。

2、批量插入:put(List<Put> list)

批量插入中生成一个List容器，然后将多行数据全部转载到该容器中，然后通过客户端的代码一次将多行数据进行提交

**[html]** [view plain](http://blog.csdn.net/u011518678/article/details/50790193) [copy](http://blog.csdn.net/u011518678/article/details/50790193)

1. public void putList(String tableName,String[] rowKeys,String[] families,String[] columns,String[] values)
2. {
3. Configuration conf=init();
4. try {
5. HBaseAdmin admin=new HBaseAdmin(conf);
6. HTable table=new HTable(conf,tableName.valueOf(tableName));
7. int length=rowKeys.length;
8. List**<Put>** putList=new ArrayList**<>**();
9. if(!admin.tableExists(Bytes.toBytes(tableName)))
10. {
11. System.err.println("the "+tableName+" is not exist");
12. System.exit(1);
13. }
14. for(int i=0;i**<length**;i++)
15. {
16. Put put=new Put(Bytes.toBytes(rowKeys[i]));
17. put.add(Bytes.toBytes(families[i]),Bytes.toBytes(columns[i]),Bytes.toBytes(values[i]));
18. putList.add(put);
19. }
20. table.put(putList);
21. table.close();
22. } catch (Exception e) {
23. // TODO: handle exception
24. }
25. }

多行插入的本质就是对List容器中的所有对象进行迭代，然后通过 HTable.put（Put p）方法进行多次插入操作。这样的批量操作将会发送多次PRC请求。

3、检查并写入：checkAndPut(byte[] row, byte[] family, byte[] qualifier, byte[] value, Put put)

该方法提供了一种原子性操作，即该操作如果失败，则操作中的所有更改都失效。该函数在多个客户端对同一个数据进行修改时将会提供较高的效率。

**[html]** [view plain](http://blog.csdn.net/u011518678/article/details/50790193) [copy](http://blog.csdn.net/u011518678/article/details/50790193)

1. public void checkAndPut(String tableName,String row,String family,String column,String value)
2. {
3. Configuration conf=init();
4. try {
5. HBaseAdmin admin=new HBaseAdmin(conf);
6. if(!admin.tableExists(Bytes.toBytes(tableName)))
7. {
8. System.err.println("the table "+tableName+" is not exist");
9. System.exit(1);
10. }
11. HTable table=new HTable(conf, TableName.valueOf(tableName));
12. Put put=new Put(Bytes.toBytes(row));
13. put.addColumn(Bytes.toBytes(family),Bytes.toBytes(column),Bytes.toBytes(value));
14. table.checkAndPut(Bytes.toBytes(row),Bytes.toBytes(family),Bytes.toBytes(column),null, put);
15. table.flushCommits();
16. } catch (Exception e) {
17. // TODO: handle exception
18. e.printStackTrace();
19. }
20. }

上述代码实现了只有在写入位置的值为Null的时候将才会将数据写入到数据库中。

注意：需要注意的时，checkAndPut方法以及类似的方法（统称为compact and set（CAS）操作）都只能对一行进行原子性操作。当checkAndPut函数中的参数row和参数put中的row不相同时，即该操作已经不再同一行中时，则会抛出异常。

**[java]** [view plain](http://blog.csdn.net/u011518678/article/details/50790193) [copy](http://blog.csdn.net/u011518678/article/details/50790193)

1. org.apache.hadoop.hbase.DoNotRetryIOException: org.apache.hadoop.hbase.DoNotRetryIOException: Action's getRow must match the passed row
2. at org.apache.hadoop.hbase.regionserver.HRegion.checkAndMutate(HRegion.java:3276)
3. at org.apache.hadoop.hbase.regionserver.RSRpcServices.mutate(RSRpcServices.java:2102)
4. at org.apache.hadoop.hbase.protobuf.generated.ClientProtos$ClientService$2.callBlockingMethod(ClientProtos.java:32203)
5. at org.apache.hadoop.hbase.ipc.RpcServer.call(RpcServer.java:2114)
6. at org.apache.hadoop.hbase.ipc.CallRunner.run(CallRunner.java:101)
7. at org.apache.hadoop.hbase.ipc.RpcExecutor.consumerLoop(RpcExecutor.java:130)
8. at org.apache.hadoop.hbase.ipc.RpcExecutor$1.run(RpcExecutor.java:107)
9. at java.lang.Thread.run(Thread.java:745)

4、缓存块操作

2方法虽然提供了批量操作，但实际的RPC请求次数没有任何的减少，因此put(List)和多次put(Put p)方法理论上的速率是相同的。而Put对象提供了一种可以打开Put缓存区的方式来提高数据提交的速率。该方式在客户端的内存中提供一块缓存区域，客户端并设置其大小，然后在用户每次进行提交时并不立刻将数据提交给Hbase集群中，而是当所有该缓存区已经满溢的时候将缓存区中的数据通过一次RPC操作，一次提交到HBase集群中去。所以缓存块在进行大量put请求，且数据量较小时将会明显提高效率。

**[java]** [view plain](http://blog.csdn.net/u011518678/article/details/50790193) [copy](http://blog.csdn.net/u011518678/article/details/50790193)

1. **public** **void** startBufferAndInsert(String tableName,String[] rows,String[] families,String[] columns,String[] values)
2. {
3. Configuration conf=init();
4. **try** {
5. //检查制定的表是否村存在
6. HBaseAdmin admin=**new** HBaseAdmin(conf);
7. **if**(!admin.tableExists(Bytes.toBytes(tableName)))
8. {
9. System.err.println("the table "+tableName+" is not exist");
10. System.exit(1);
11. }
12. admin.close();
13. //创建表连接
14. HTable table=**new** HTable(conf,TableName.valueOf(tableName));
15. //将数据自动提交功能关闭
16. table.setAutoFlush(**false**);
17. //设置数据缓存区域
18. table.setWriteBufferSize(64\*1024\*1024);
19. //然后开始写入数据
20. **int** length=rows.length;
21. **for**(**int** i=0;i<length;i++)
22. {
23. Put put=**new** Put(Bytes.toBytes(rows[i]));
24. put.addColumn(Bytes.toBytes(families[i]),Bytes.toBytes(columns[i]),Bytes.toBytes(values[i]));
25. table.put(put);
26. }
27. //刷新缓存区
28. table.flushCommits();
29. //关闭表连接
30. table.close();
31. } **catch** (Exception e) {
32. // TODO: handle exception
33. e.printStackTrace();
34. }
35. }

使用缓存插入方式时，要注意将table的自动填充属性进行关闭，并且在数据插入完成后进行一次手动的提交操作。将缓存区中的数据手动的提交到HBase服务器中。

三、Put类详解

1、主要属性：KeyValue对象数组

Put类中主要含有一个KeyValue对象数组，KeyValue对象是HBase底层存储的一个重要类，代表了数据在底层存储时的状态。KeyValue对象代表了一个Hbase表中的一个数据单元，即含有行值（row）、列簇（family）、列（column）、时间戳（timestamp）和值（value），从这些信息能够在表中唯一确定一个数据单元。在KeyValue对象中，Key（键）包含了一个value值的row、family、column和timestamp信息，而value则是该表单元格的数据。

当插入一条数据时，其实就是讲KeyValue进行序列化后，然后传递后Hbase集群，集群在根据KeyValue的值进行相应的操作。

2、主要方法

（1）Put(byte[] row) / Put(byte[] row,RowLock lock)

初始化函数，一个Put对象值代表一行数据，但是因为其内部含有多个KeyValue键值对，Put对象可以填充多个列簇(family)、列(qualifier)，甚至是多时间戳(timestampe)数据。第二个函数中有行锁(RowLock)参数，用户可以通过设定该参数指定一个行锁。一般来说系统在进行Put时会对put所在的行添加一个行锁。系统并不提倡用户自己定义行锁，因为可能在多个用户指定多个行锁时会造成死锁的情况，导致系统资源在两个客户端断开连接之前一直被占用。

（2）add(Cell kv) / **add**(byte[] family, byte[] qualifier, long ts, byte[] value) / **addColumn**(byte[] family, [ByteBuffer](http://docs.oracle.com/javase/7/docs/api/java/nio/ByteBuffer.html?is-external=true) qualifier, long ts, [ByteBuffer](http://docs.oracle.com/javase/7/docs/api/java/nio/ByteBuffer.html?is-external=true) value)  / **addImmutable**(byte[] family, byte[] qualifier, long ts, byte[] value, org.apache.[**Hadoop**](http://lib.csdn.net/base/hadoop).hbase.Tag[] tag)

这四种方法都是想Put对象中添加单元数据，即增加一个KeyValue对象。

A、添加Cell对象，Cell类可以引用KeyVakue的对象，因此此处也可以插入KeyValue对象，因此即可将该函数理解为插入一个完整的KeyValue对象即可

B、添加列簇、列、时间戳、值，Put对象会在接收这些数据后将其初始化成一个KeyValue对象，不过该方法已经废弃，不建议使用

C、该方法和第二方法作用相同

D、添加恒定数据，通过该方法添加一个恒定的KeyValue对象值，具体效果没有试验过

（3）**has**(byte[] family, byte[] qualifier, byte[] value)

判断Put对象中是否含有制定的value值/列/列簇，返回值为Boolean值

（4）setWriteToWAL(boolean write)

是否将数据写入到预写日志(Write-Ahead-Log)中，预写日志是一种数据保护措施，如果当HBase某一个节点故障时，可以通过预写日志中的记录的数据操作进行数据恢复。该选项如果打开，数据会被写入到预写日志中，安全性增加，但是会损耗一定性能，如果不打开，则损耗变小，安全性降低。该方法在1.0.1以上版本已经去掉了，所以可以用接下来讲到的setDurability方法来设置。

（5）setTTL(long ttl)

ttl毫秒数。该函数对该行中的所有的KeyValue对象设置TTL，TTL（Time-To-Live）的作用是从数据产生起，经过TTL时间后就会被作为废弃数据而被删除掉。

（6）setDurability(Durability d)

该函数对写入WAL模式进行设定。可填充的值为：Durability.ASYNC\_WAL(异步进行数据写入)、Durability.SYNC\_WAL(同步进行写入)、Durabiliry.SKIP\_WAL(不进行WAL填写)、Durability.FSYNC\_WAL(强制同步进行WAL写入)、Durability.USE\_DEFAULT(默认选项，为SYNC\_WAL选项)

（7）其他方法

A、getRow()  获取创建Put实例时的行健

B、getRowLock()   获取创建Put实例时的行锁

C、getLockId()    返回使用rowlock参数传递给构造函数的可选的锁ID，当违背制定时放回-1L

D、getTimeStamp()   获取相应Put实例的时间戳，改制在构造函数中有ts参数传入，如果没有指定的话则返回 Long.MAX\_VALUE

E、isEmpty()   返回该Put实例中是否含有KeyValue实例

F、numFamilies()   返回该Put实例中的列簇数量

G、size()   返回本次Put对象添加的KeyValue实例的数量