

Transwarp Data Hub Version 4.6

Hyperbase使用手册

星环信息科技（上海）有限公司

版本号 T00146x-04-012, 2016-08-26

目录

1. 简介	2
1.1. Hyperbase功能特性	2
1.2. 关于本手册	3
Transwarp HBase使用手册	4
2. Transwarp HBase入门	5
2.1. 安装Transwarp HBase	5
2.2. Transwarp HBase服务的角色	5
2.3. 和Transwarp HBase交互	7
3. Transwarp HBase数据模型	8
3.1. Transwarp HBase数据对象	8
3.2. Transwarp HBase中的表	9
3.2.1. 概念上的表	9
3.2.2. 物理上的表	12
3.2.3. Transwarp HBase表映射成二维表	12
3.3. Transwarp HBase中的索引	13
4. Transwarp HBase简明教程	15
4.1. Transwarp HBase Shell简明教程	15
4.1.1. 建表	15
4.1.2. 填入数据	16
4.1.3. 扫描整张表	16
4.1.4. 删除表	17
4.2. Transwarp HBase SQL简明教程	17
4.2.1. 连接到Inceptor Engine	18
4.2.2. HBase映射表	19
4.2.3. Hyperdrive映射表	19
5. Transwarp HBase Schema设计	22
5.1. 模式(Schema) 创建	22
5.1.1. 模式更新	22
5.2. 表模式经验法则	22
5.3. 列族的数量	23
5.3.1. 列族的基数 (Cardinality)	23
5.4. 行键(RowKey)设计	23
5.4.1. Hotspotting	23
5.4.1.1. Salting	23
5.4.1.2. Hashing	24
5.4.1.3. 反转键 (Reversing the Key)	24
5.4.2. 单调递增行键/时序数据	25
5.4.3. 尽量最小化行和列的大小	25
5.4.3.1. 列族	25

5.4.3.2. 属性	25
5.4.3.3. 行键长度	25
5.4.3.4. 字节模式	25
5.4.4. 倒序时间戳	26
5.4.5. 行键和列族	27
5.4.6. 行键永远不变	27
5.4.7. 行键和region split之间的关系	27
5.5. 版本数量	28
5.5.1. 最大版本数	28
5.5.2. 最小版本数	28
5.6. 支持数据类型	29
5.7. 存活时间 (TTL)	29
5.8. 保留删除的单元	29
6. Transwarp HBase架构	32
6.1. Transwarp HBase架构组件	32
6.1.1. Region	32
6.1.2. Transwarp HBase HMaster	32
6.1.3. Zookeeper:协调者	33
6.2. 各组件如何协调工作?	33
6.3. Transwarp HBase的首次读/写	34
6.4. Transwarp HBase Meta表	34
6.5. RegionServer组件	35
6.6. Transwarp HBase写操作	35
6.7. Transwarp HBase MemStore	36
6.8. Transwarp HBase Region Flush	36
6.9. Transwarp HBase HFile	37
6.9.1. Transwarp HBase Region Flush	37
6.9.2. HFile索引	38
6.10. Transwarp HBase读合并 (Read Merge)	38
6.11. Transwarp HBase Compaction	39
6.11.1. Transwarp HBase Minor Compaction	39
6.11.2. Transwarp HBase Major Compaction	39
6.12. Region 拆分 (split)	40
6.13. HDFS数据备份	40
6.14. Transwarp HBase崩溃修复	40
6.15. 数据修复	41
6.16. Transwarp HBase架构评测	41
6.16.1. 优势	41
6.16.2. 不足	42
7. Transwarp HBase安全	43
7.1. Transwarp HBase Shell交互的认证和授权	43
7.1.1. 认证: 获取Kerberos Ticket	43

7.1.2. 权限管理	43
7.1.3. Transwarp HBase中权限作用的级别	44
7.2. Transwarp HBase中的“角色”	44
7.3. hbase:acl表	45
7.3.1. 通过Transwarp HBase Shell授予权限	45
7.3.1.1. Global (全局) 权限的授予	46
7.3.1.2. Namespace (命名空间) 权限的授予	46
7.3.1.3. Table (表) 权限的授予	47
7.3.1.4. 列族权限的授予	47
7.3.1.5. 列权限的授予	47
7.3.2. 通过Transwarp HBase Shell收回权限	48
7.3.2.1. 收回指定权限	48
7.3.2.2. 一次性收回权限	48
7.3.3. 通过Transwarp HBase Shell查看权限	49
7.3.3.1. 查看hbase:acl表	49
7.3.3.2. user_permission 命令	49
7.4. API交互的认证和权限管理	50
7.4.1. 通过认证访问Transwarp HBase	50
7.4.2. 安全模式下的基本操作	52
7.4.3. 权限管理API示例	55
8. Transwarp HBase Shell命令	58
8.1. Transwarp HBase Shell表管理命令	58
8.1.1. list	58
8.1.2. create	59
8.1.3. describe	59
8.1.4. exists	60
8.1.5. show_filters	60
8.1.6. enable	60
8.1.7. disable	60
8.1.8. enable_all	61
8.1.9. is_enabled	61
8.1.10. disable_all	61
8.1.11. is_disabled	61
8.1.12. delete_table	61
8.1.13. drop_all	62
8.1.14. alter	62
8.1.15. alter_async 和 alter_status	63
8.2. Transwarp HBase Shell数据操作命令	63
8.2.1. count	63
8.2.2. put	63
8.2.3. get	64

8.2.4. <code>scan</code>	64
8.2.5. <code>delete</code>	65
8.2.6. <code>deleteall</code>	65
8.2.7. <code>truncate</code> 和 <code>truncate_preserve</code>	65
8.3. Transwarp HBase Shell Namespace相关命令	66
8.3.1. <code>alter_namespace</code>	66
8.3.2. <code>create_namespace</code>	66
8.3.3. <code>describe_namespace</code>	66
8.3.4. <code>drop_namespace</code>	66
8.3.5. <code>list_namespace</code>	66
8.3.6. <code>list_namespace_tables</code>	67
8.4. Transwarp HBase Shell索引命令	67
8.4.1. 全局索引指令	67
8.4.1.1. 创建全局索引	67
8.4.1.2. 生成全局索引	68
8.4.1.3. 查看全局索引	70
8.4.1.4. 删除全局索引	71
8.4.2. 本地索引指令	71
8.4.2.1. 生成本地索引	71
8.4.3. 全文索引指令	72
8.4.3.1. 生成全文索引	72
8.5. Transwarp HBase Shell快照相关命令	73
8.5.1. <code>snapshot</code>	73
8.5.2. <code>list_snapshots</code>	73
8.5.3. <code>delete_snapshot</code>	74
8.5.4. <code>clone_snapshot</code>	74
8.5.5. <code>restore_snapshot</code>	74
9. Transwarp HBase Shell常规指令	75
9.1. <code>status</code>	75
9.2. <code>version</code>	75
9.3. <code>whoami</code>	75
9.4. Hyperdrive相关命令	76
9.4.1. <code>create_table</code>	76
9.4.2. <code>hput</code>	77
9.4.3. <code>hget</code>	78
9.4.4. <code>hscan</code>	78
9.4.5. <code>hdelete</code>	79
10. Transwarp HBase JSON配置使用说明	81
10.1. JSON配置操作简介	81
10.1.1. 表数据 vs 表的扩展数据	81
10.1.2. 表的元数据 vs 表的扩展元数据	81

10.1.3. JSON配置的命令行指令	81
10.2. JSON配置详解	83
10.2.1. 扩展元数据JSON串的基本格式	83
10.2.2. base模块	85
10.2.3. fulltextindex模块	87
10.2.4. globalindex模块	89
10.2.5. lob模块	93
10.3. JSON配置操作模板	97
10.4. JSON配置简单使用实例	102
10.5. JSON配置迁移扩展元数据	108
11. Transwarp HBase API使用说明	111
11.1. HBaseConfiguration	111
11.2. HBaseAdmin	112
11.3. HyperbaseAdmin	114
11.4. IndexRebuilder	116
11.5. HTableDescriptor	118
11.6. HColumnDescriptor	119
11.7. HTable	120
11.8. Put	121
11.9. Get	122
11.10. Scan	123
11.11. Result	124
11.12. Object Store使用方法	125
12. Transwarp HBase SQL使用说明	131
12.1. DESC FORMATTED	131
12.2. Transwarp HBase SQL DDL	131
12.2.1. 建表: CREATE TABLE	131
12.2.1.1. 建HBase映射表	132
12.2.1.2. 建Hyperdrive映射表	133
12.2.2. 编辑表: ALTER TABLE	135
12.2.3. 删除表: DROP TABLE	136
12.2.4. 清空表: TRUNCATE	136
12.3. Transwarp HBase SQL索引DDL	137
12.3.1. 建索引: CREATE INDEX	137
12.3.1.1. 为HBase映射表建索引	137
12.3.1.2. 为Hyperdrive映射表建索引	138
12.3.2. 删除索引: DROP INDEX	140
12.3.3. 生成索引	140
12.4. Transwarp HBase SQL DML	140
12.4.1. 插入数据: INSERT	141
12.4.2. 更新表: UPDATE	141

12.4.3. 删除记录: DELETE	142
12.4.4. 查询: SELECT	142
12.4.4.1. 指定HBase映射表的索引	143
12.4.4.2. 指定Hyperdrive映射表的索引	147
13. Transwarp HBase故障诊断	149
13.1. 一般准则	149
13.2. 日志	149
13.2.1. 日志位置	149
13.2.2. 日志级别	150
13.2.3. JVM Garbage Collection Log (垃圾收集日志)	150
13.3. 资源	153
13.3.1. IRC	153
13.3.2. JIRA	153
13.4. 工具	153
13.4.1. 内置工具	153
13.4.2. 外部工具	154
13.5. 客户端	163
13.5.1. 因 hbase.client.scanner.max.result.size 不匹配使得扫描结果缺失	163
13.5.2. ScannerTimeoutException/UnknownScannerException	163
13.5.3. Thrift 和 Java APIs的性能偏差	163
13.5.4. Scanner.next 时出现 LeaseException	163
13.5.5. Shell或客户端应用抛出很多不太重要的异常	163
13.5.6. 压缩时客户端长时停顿	163
13.5.7. 安全客户端不能连接	164
13.5.8. ZooKeeper 客户端连接错误	164
13.5.9. 客户端内存耗尽, 但堆大小看起来不太变化	165
13.5.10. 安全客户端不能连接	165
13.6. MapReduce	165
13.6.1. 你认为自己在用集群, 实际上你在用本地(Local)	165
13.7. NameNode	166
13.7.1. 表和region的HDFS 工具	167
13.7.2. 浏览 HDFS, 查看Transwarp HBase对象	167
13.7.3. 意想不到的文件系统峰值	168
13.8. 网络	168
13.8.1. 网络峰值(Network Spikes)	168
13.8.2. 回环IP(Loopback IP)	169
13.8.3. 网络接口	169
13.9. RegionServer	169
13.9.1. 启动错误	169
13.9.2. 运行时错误	170
13.9.3. 由反向DNS引起的快照问题	172
13.9.4. 终止错误	172

13.10. Master	172
13.10.1. 启动错误	172
13.10.2. 终止错误	173
13.11. ZooKeeper	173
13.11.1. 启动错误	173
14. Transwarp HBase工具	174
14.1. 分布式存储运维工具 (DSTools)	174
14.1.1. 工具修复常见异常情况:	174
14.1.2. 工具操作步骤说明:	175
14.1.3. runDSTools.sh工具操作步骤:	175
14.1.4. runHFileCheck.sh工具操作步骤:	177
14.2. 在Transwarp HBase集群之间迁移数据	179
14.2.1. 用 import/export 迁移数据	179
14.2.2. 用快照迁移数据	180
14.2.3. 直接 CopyTable 迁移数据	181
14.3. SQL BulkLoad	183
14.3.1. HBase SQL BulkLoad	183
14.3.2. Hyperdrive SQL Bulkload	186
14.4. 运维管理工具	190
15. Transwarp HBase常见问题	191
Transwarp ES使用手册	194
16. Transwarp ES入门	195
16.1. Transwarp ES是什么	195
16.2. 安装Transwarp ES	195
16.3. 了解您的Transwarp ES服务	196
16.4. 配置Transwarp ES	197
16.5. 和Transwarp ES交互	201
16.5.1. REST API	201
16.5.2. ESDrive SQL	202
17. Transwarp ES数据模型	204
17.1. Transwarp ES中的数据对象	204
17.1.1. Transwarp ES Index (索引)	204
17.1.2. Transwarp ES Type	204
17.1.3. Transwarp ES Document	204
17.1.4. Field	204
17.2. 将Index映射为二维表	204
18. Transwarp ES API简明教程	210
18.1. 编入Document	210
18.2. 获取整个Document	211
18.3. 获取部分Document	212
18.4. 查看Document是否存在	213
18.5. 删除Document	213

18.6. 更新Document	214
18.7. 新建一个Document	215
18.8. 轻量检索	215
18.9. 总结	217
19. Transwarp ES架构	218
19.1. 分片 (Shard) 和副本 (Replica)	218
20. Transwarp ES检索简介	219
20.1. 空检索	219
20.2. 检索请求的格式	220
21. URI检索	222
21.1. 输出结果分页	223
21.2. Query String语法	223
21.2.1. 检索字段的指定	223
21.2.2. 逻辑运算符	224
21.2.3. 偏好运算符	225
21.2.4. 括号的使用	225
21.2.5. 通配符	225
21.2.6. 正则表达式	226
21.2.7. 范围查询	226
21.2.8. 保留字符	226
21.2.9. 复杂查询	226
22. Request Body检索 (Elasticsearch 1.3.1)	227
22.1. Query DSL	227
22.1.1. 查询语句	228
22.1.2. 查询语句的组合和嵌套	228
22.1.3. Query和Filter	229
23. Request Body检索 (Elasticsearch 2.0.0)	231
23.1. Query DSL	231
23.1.1. 查询语句	232
23.1.2. 查询语句的组合和嵌套	232
23.1.3. Query Context和Filter Context	233
24. 映射 (mapping) 与分词 (analysis) 基础	235
24.1. 数据类型差异	235
24.2. 确切值(Exact values) vs. 全文文本(Full text)	236
24.3. 倒排索引(inverted index)	237
24.4. 分词与分词器	239
24.4.1. 内建的分词器	239
24.4.2. 分词器的使用	240
24.4.3. 测试分词器	241
24.4.4. 指定分词器	242
24.5. 映射 (mapping)	242
24.5.1. 核心简单字段类型	242

24.5.2. 查看mapping	243
24.5.3. 自定义字段mapping	243
24.5.4. 更新mapping	245
24.5.5. 测试mapping	246
24.6. 复合核心字段类型	246
24.6.1. 多值字段	246
24.6.2. 空字段	247
24.6.3. 多层对象	247
24.6.4. 内部对象的mapping	247
24.6.5. 内部对象是怎样被索引的	248
24.6.6. 内部对象数组	249
25. Mapping操作	250
25.1. 如何显式定义Mapping	250
25.2. 定义元数据字段Mapping (META-FIELD_MAPPING)	251
25.3. 定义字段Mapping (FIELD_MAPPING)	253
26. 常见Transwarp ES查询语句	254
26.1. Full Text查询语句	254
26.1.1. match 查询语句	254
26.2. Term-level查询语句	256
26.2.1. term 查询语句	256
26.2.2. terms 查询语句	257
26.2.3. range 查询语句	257
26.2.4. wildcard 查询语句	258
26.3. 复合查询语句	258
26.3.1. bool 查询语句 (Elasticsearch 1.3.1)	258
26.3.2. bool 查询语句 (Elasticsearch 2.0.0)	260
27. 常见Transwarp ES API列表	263
27.1. API使用约定	263
27.1.1. 多Index的使用	263
27.1.2. 输出格式	263
27.2. Cluster级别API	263
27.2.1. Cluster health API	264
27.2.2. Cluster stats API	265
27.2.3. Cluster settings API	265
27.2.4. Nodes stats API	265
27.3. Index级别API	266
27.3.1. 创建Index	266
27.3.2. 删除Index	266
27.3.3. 查看Index是否存在	267
27.3.4. Index settings API	267
27.3.5. Index optimize API	267

27.3.6. Index mapping API	268
27.4. Document API	268
27.4.1. 编入API	268
27.4.2. 获取API	269
27.4.3. 删除API	269
28. Transwarp ESDrive SQL使用说明	270
28.1. Transwarp ESDrive SQL 快速入门	270
28.2. Transwarp ESdrive SQL DDL	272
28.2.1. 建表: CREATE	272
28.2.2. 修改表: ALTER	275
28.2.3. 删除表: DROP	276
28.2.4. 清空表: TRUNCATE	276
28.3. Transwarp ESdrive SQL DML	276
28.3.1. 插入: INSERT	276
28.3.2. 删除表中记录: DELETE	277
28.3.3. 查询: SELECT	277
28.3.3.1. CONTAINS	279
28.3.3.2. MATCHES	287
28.4. Transwarp ESdrive 实战	290
28.4.1. 创建内表和外表	290
28.4.2. 插入数据	292
28.4.3. 数据查询	292
29. Transwarp ES常见问题	294
客户服务	297

免责声明

本说明书依据现有信息制作, 其内容如有更改, 恕不另行通知。星环信息科技(上海)有限公司在编写该说明书的时候已尽最大努力保证期内容准确可靠, 但星环信息科技(上海)有限公司不对本说明书中的遗漏、 不准确或印刷错误导致的损失和损害承担责任。具体产品使用请以实际使用为准。

注释: Hadoop® 和 SPARK® 是ApacheTM 软件基金会在美国和其他国家的商标或注册的商标。 Java® 是 Oracle公司在美国和其他国家的商标或注册的商标。 Intel® 和Xeon® 是英特尔公司在美国、 中国和其他国 家的商标或注册的商标。

版权所有 © 2013年-2016年星环信息科技(上海)有限公司。保留所有权利。

©星环信息科技(上海)有限公司版权所有, 并保留对本说明书及本声明的最终解释权和修改权。本说明书 的版权归星环信息科技(上海)有限公司所有。未得到星环信息科技(上海)有限公司的书面许可, 任何人 不得以任何方式或形式对本说明书内的任何部分进行复制、 摘录、 备份、 修改、 传播、 翻译成其他语言、 或 将其全部或部分用于商业用途。

手册版本信息

版本号: T00146x-04-012

发布日期: 2016-08-26

1. 简介

Transwarp Hyperbase实时数据库是建立在Apache HBase和Elasticsearch基础之上，融合了多种索引技术、分布式事务处理、全文实时搜索、图形数据库在内的实时NoSQL数据库。Hyperbase可以高效地支持企业的在线OLTP应用、高并发OLAP应用、批处理应用、全文搜索或高并发图形数据库检索应用，结合Inceptor高速SQL引擎，是企业创建可扩展在线运营数据库（Operational Database）或者实时分析型数据库(ODS – Operational Data Store)的最佳选择。

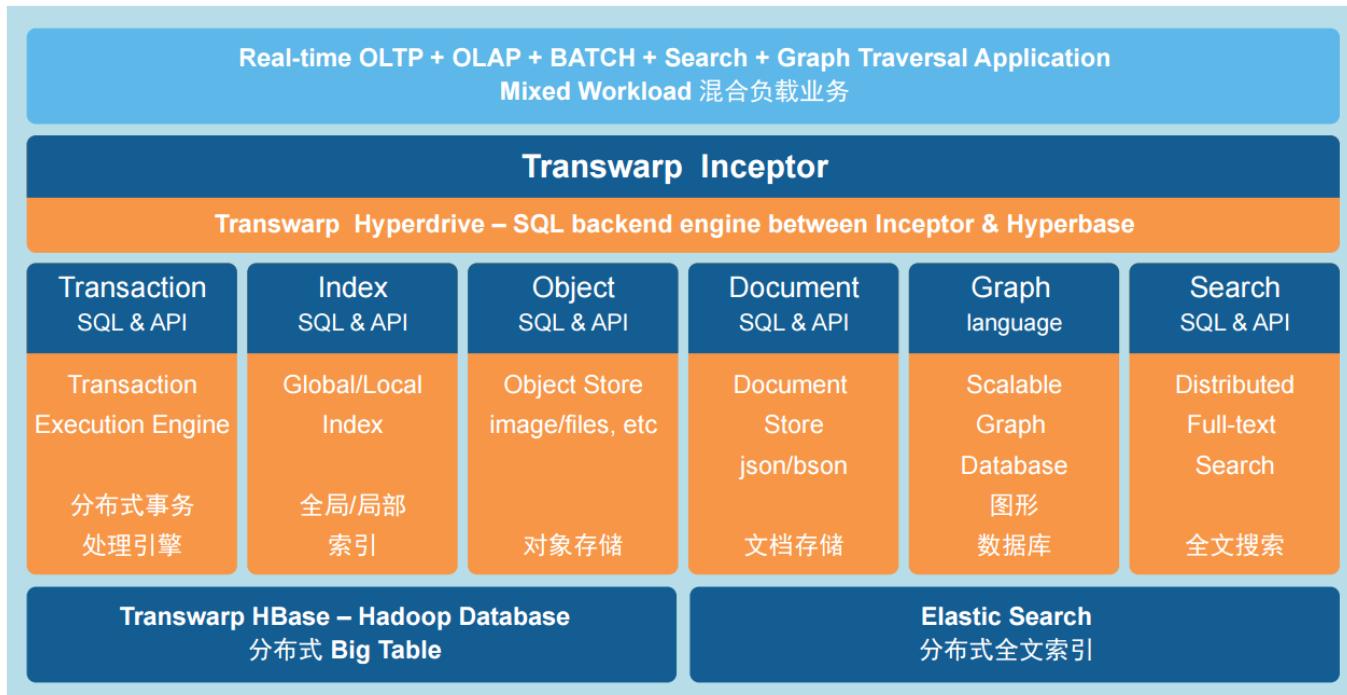


图 1. Hyperbase架构图

1.1. Hyperbase功能特性

- **SQL支持**

通过Inceptor支持采用SQL进行批处理和高并发查询，批处理比Map/Reduce快10倍。可从Hyperbase的行存储转换成Holodesk的列存储，同时支持在线查询和高速OLAP分析。

- **索引**

支持全局、局部、高维索引和高级过滤器，可用于高并发低延时的OLAP查询。

- **CRUD**

支持通过SQL高并发毫秒级数据插入/修改/查询/删除。

- **全文检索**

基于Lucene的分布式全文索引，可结合Big Table实时创建索引并进行搜索。

- **多数据类型支持**

支持文档型数据（如JSON/BSON）的存储、索引和搜索，支持对象数据（图片、音视频、二进制文档等）的存储、检索和自动回收。

1.2. 关于本手册

本手册分为两部分：Transwarp HBase使用手册和Transwarp ES使用手册，它们将分别介绍 Transwarp HBase 和Transwarp ES的基础知识和使用方法。

Transwarp HBase使用手册

2. Transwarp HBase入门

2.1. 安装Transwarp HBase

Transwarp HBase的安装通过管理界面完成，可以在第一次安装Transwarp Data Hub集群时安装，也可以向安装好的集群另外安装Transwarp HBase服务。Transwarp Data Hub集群的安装在《Transwarp Data Hub安装手册》中有详细的介绍，这里不赘述。

2.2. Transwarp HBase服务的角色

登陆您的Transwarp Data Hub集群，点击一个Transwarp HBase服务，进入服务主页后选择“角色”：

角色名称	节点名称	机柜名称	服务链接	健康状况	操作
region server (tw-node118)	tw-node118	/racknroll	Link	● Running	▶ ■ ×
region server (tw-node119)	tw-node119	/racknroll	Link	● Running	▶ ■ ×
region server (tw-node120)	tw-node120	/racknroll	Link	● Running	▶ ■ ×
HMaster (tw-node118)	tw-node118	/racknroll	Link	● Running	▶ ■ × ⚡
HMaster (tw-node119)	tw-node119	/racknroll	Link	● Running	▶ ■ × ⚡
HMaster (tw-node120)	tw-node120	/racknroll	Link	● Running	▶ ■ × ⚡
chronos server (tw-node118)	tw-node118	/racknroll	N/A	● Running	▶ ■ ×
region server (tw-node3114)	tw-node3114	/racknroll	Link	● Running	▶ ■ ×
region server (tw-node3115)	tw-node3115	/racknroll	Link	● Running	▶ ■ ×
region server (tw-node3116)	tw-node3116	/racknroll	Link	● Running	▶ ■ ×

图 2. Transwarp HBase服务的角色

可以看到Transwarp HBase服务有三种角色：

- HMaster: Transwarp HBase的Master/Slave结构中的Master。点击一个HMaster的“服务链接”可以进入它的Web UI:

The screenshot shows the Transwarp HBase Master UI at the URL 172.16.1.119:60010/master-status?filter=all. The interface includes a navigation bar with Home, Table Details, Local Logs, Log Level, Debug Dump, Metrics Dump, and HBase Configuration. Below this is a section titled "Master" with a single entry "test-01". Under "Tasks", there is a table with columns Start Time, Description, State, and Status. One task listed is "Master startup" which is "RUNNING (since 3hrs, 51mins, 20sec ago)". The status message indicates another master is active. The "Software Attributes" section contains a table with columns Attribute Name, Value, and Description, listing various system properties like HBase Version, Hadoop Version, and Zookeeper Quorum.

- Region Server: Transwarp HBase的Master/Slave结构中的Slave。点击任意一个Region Server的“服务链接”可以进入它的Web UI:

The screenshot shows the Transwarp HBase Region Server UI at the URL 172.16.1.119:60030/rs-status. The interface includes a navigation bar with Home, Local Logs, Log Level, Debug Dump, Metrics Dump, and HBase Configuration. Below this is a section titled "RegionServer test-03,60020,1466566747164". Under "Server Metrics", there is a table with tabs for Base Stats, Memory, Requests, hlogs, Storefiles, and Queues. The "Requests Per Second" tab is selected, showing 0 requests per second. The "Tasks" section contains a table with columns Start Time, Description, State, and Status. Four tasks listed are all "COMPLETE (since 33sec ago)" and describe initializing regions with specific IDs.

- Chronos Server: Transwarp HBase中提供全局唯一且单调递增的timestamp的服务，用于保证事务处理的一致性。

这三个角色的功能以及其他Transwarp HBase架构的细节请参考[Transwarp HBase架构](#)。

2.3. 和Transwarp HBase交互

和Transwarp HBase交互有以下三种方式：

- SQL（推荐方式）：我们的SQL引擎Inceptor Engine提供了丰富的SQL语法，并对SQL的执行进行了充分的优化，使用SQL和Transwarp HBase交互在正确性和性能方面都有很好的保证。细节请参考[Transwarp HBase SQL使用说明](#)。
- Shell: Transwarp HBase提供交互式Shell以及一系列Shell指令用于数据操作，细节请参考[Transwarp HBase Shell命令](#)
- Java API: Transwarp HBase支持Apache HBase原生的API，同时还提供多种自有的API，细节请参考[Transwarp HBase API使用说明](#)。

3. Transwarp HBase数据模型

Transwarp HBase以“表”为结构来组织数据，“表”中有“行”和“列”，但是Transwarp HBase中的这些概念和关系型数据库的二维表不同。Transwarp HBase表应当被看做一个“稀疏的，分布式的，持久的，多维度有序map”。下面，我们围绕这个描述来解说Transwarp HBase中的数据模型。

3.1. Transwarp HBase数据对象

Transwarp HBase支持所有Apache HBase中的所有数据对象，同时还提供Transwarp HBase独有的对象——Hyperdrive表和索引。这些数据对象如下：

- **命名空间（Namespace）**：对表的逻辑分组，类似于关系型数据库中的Database概念。Namespace可以帮助用户在多租户场景下做到更好的资源和数据隔离。
- **表（Table）**：一张表由多个行组成。Transwarp HBase中的表分为两种：
 - **HBase表**：HBase表没有数据类型，表中单元格的值都存储为byte[]。在和二维表之间序列化和反序列化时会引起一些数据类型识别上的问题，
 - **Hyperdrive表**：Hyperdrive表是星环信息科技（上海）有限公司为SQL对接Transwarp HBase专门设计的表。Hyperdrive表有数据类型（包括：BOOLEAN、TINYINT、SMALLINT、INTEGER、BIGINT、DATE、TIMESTAMP、DECIMAL、FLOAT、DOUBLE、STRING、VARCHAR和STRUCT），有自己的编码和解码方式，解决了 **NULL** 等特殊类型在Transwarp HBase中的表示；在SQL条件到Transwarp HBase条件的转换上提供更全面的支持；对索引的支持也更加全面。Hyperdrive表的所有操作几乎都可以使用SQL来完成。
- **行（Row）**：每一个行由一个独特的row key和多个列组成。表中各行数据按row key排序，所以row key的设计对Transwarp HBase的性能影响非常重要。[Transwarp HBase Schema设计](#)中会进行详细介绍。
- **列族（Column Family）**：表中数据以列族分组。物理上，同一列族的数据存储在一起。每个列族都可以有自己的存储属性，例如数据是否缓存在内存中，数据是否压缩等等。列族必须在建表时申明。一个列族下可以有多个列，这些列的列名前缀相同，都为列族名。
- **列限定名（Column Qualifier）**：统一列族下的列由列限定名指定。例如一个名为 **info** 的列族下可以有 **info:name** 列、**info:age** 列和 **info:gender** 列。这里 **name**, **age** 和 **gender** 即为列限定名。
- **列（Column）**：一个列由一个列族和一个列限定符唯一指定，列名以列族名和列限定名组成，中间以“：“隔开，
- **单元格（Cell）**：一组Row Key、列族和列限定符指定唯一的单元格。单元格中存放一个值和一个timestamp。
- **时间戳（Timestamp）**：单元格的值可以有不同版本，各个版本由时间戳区分。默认时间戳为单元格值被写入时的时间。
- **索引（Index）**：为了弥补Apache HBase本身只能利用原表row key的条件查询的局限，Transwarp HBase中增加了下面几种索引。通过将查询字段作为索引的row key，可以提速对任何指定字段的查询：
 - **全局索引（Global Index）**：全局索引中的数据做为另一张相对独立的HBase表存在，它的row key即为原表的索引字段，整个过程对用户完全透明的完成。
 - **本地索引（Local Index）**：本地索引中的数据保存在原表内部，从索引到取原表数据这一步可以节省网

络开销。

- **全文索引（Fulltext Index）：** 全文索引利用Elasticsearch作为索引数据的存储，用于加速对指定字段的模糊查询。

3.2. Transwarp HBase中的表

本节通过一个存放用户信息的表bank_info来介绍Transwarp HBase表中数据的结构。本节中介绍的概念对HBase表和Hyperdrive表都适用，下面除非另外指出，Transwarp HBase表代表HBase表以及Hyperdrive表。

3.2.1. 概念上的表

下面是一张概念上的bank_info表：

表1. 概念上的银行用户表bank_info

Account Number (row key)	Personal (column family)		Contact (column family)		Balance (column family)	Timestamp
	name (column qualifier)	password (column qualifier)	email (column qualifier)	cellphone (column qualifier)	balance (column qualifier)	
0001		5678				t16
					10000.00	t05
				12345678912		t04
			zs@mail.com			t03
		1234				t02
	Zhang San					t01
0002					56000.00	t18
			ls@sample.com			t17
					1000.00	t10
				13513572468		t09
			ls@school.edu			t08
		2468				t07
	Li Si					t06
0003					500.00	t15
				13612345678		t14
			ww@mail.com			t13
		1357				t12
	Wang Wu					t11

表bank_info的row key为账户号码Account Number；各行以账户号码排序。该表有三个列族：

- Personal：含有两列，列限定名分别为name和password。
- Contact：含有两列，列限定名分别为email和cellphone。
- Balance：一列，列限定名为balance。

从timestamp上看：

- Row key为0001的行有6个版本。
- Row key为0002的行有7个版本。
- Row Key为0003的行有5各版本。

表中无值的单元格在系统中是不存在的，表的“稀疏性”就体现在这里。上面的表以一个二维表形式展现，但是

在Transwarp HBase中，这并不准确。Transwarp HBase表更接近多维map，由多层 键值对 组成：

多维map形式的bank_info表

```
{Table: {RowKey: {ColumnFamily: {ColumnQualifier: {Timestamp: Value}}}}}
```

所以，bank_info表还可以表示如下：

```
bank_info:{  
    0001:{  
        Personal: {  
            name: {t01: Zhang San}  
            password: {t16: 5678  
                        t02: 1234}  
        }  
        Contact: {  
            email: {t03: zs@mail.com}  
            cellphone: {t04: 12345678912}  
        }  
        Balance: {  
            balance: {t05: 10000.00}  
        }  
    }  
    0002:{  
        Personal: {  
            name: {t06: Li Si}  
            password: {t07: 2468}  
        }  
        Contact: {  
            email: {t08: ls@sample.com  
                        t17: ls@school.edu}  
            cellphone: {t09: 13513572468}  
        }  
        Balance: {  
            balance: {t18: 56000.00  
                        t10: 1000.00}  
        }  
    }  
    0003:{  
        Personal: {  
            name: {t11: Wang Wu}  
            password: {t12: 1357}  
        }  
        Contact: {  
            email: {t13: ww@hmail.com}  
            cellphone: {t14: 13612345678}  
        }  
        Balance: {  
            balance: {t15: 500.00}  
        }  
    }  
}
```

3.2.2. 物理上的表

在实际的存储上，[概念上的银行用户表bank_info](#)中空出的单元格是不存在的，物理上的bank_info如下，bank_info中的数据的存储按Column Family组织

表 2. Column Family: Personal

Row Key	Timestamp	Column
0001	t16	Personal:password=5678
0001	t02	Personal:password=1234
0001	t01	Personal:name=Zhang San
0002	t07	Personal:password=2468
0002	t06	Personal:name=Li Si
0003	t12	Personal:password=1357
0003	t11	Personal:name=Wang Wu

表 3. Column Family: Contact

Row Key	Timestamp	Column
0001	t04	Contact:cellphone=12345678912
0001	t03	Contact:email= zs@mail.com
0002	t17	Contact:email= ls@sample.com
0002	t09	Contact:cellphone=13513572468
0002	t08	Contact:email= ls@school.edu
0003	t14	Contact:cellphone=13612345678
0003	t13	Contact:email= ww@hmail.com

表 4. Column Family: Balance

Row Key	Timestamp	Column
0001	t05	Balance:balance=10000.00
0002	t18	Balance:balance=56000.00
0002	t10	Balance:balance=1000.00
0003	t15	Balance:balance=500.00

3.2.3. Transwarp HBase表映射成二维表

通过Inceptor Engine使用SQL对Transwarp HBase表进行操作时，Inceptor Engine中的表将是Transwarp HBase表的二维映射，映射表的列将为原表的列（Column Family:Column Qualifier）。映射表的单元格中将只有原表单元格中 timestamp最新 的值。将bank_info映射为二维表的逻辑如下图所示：

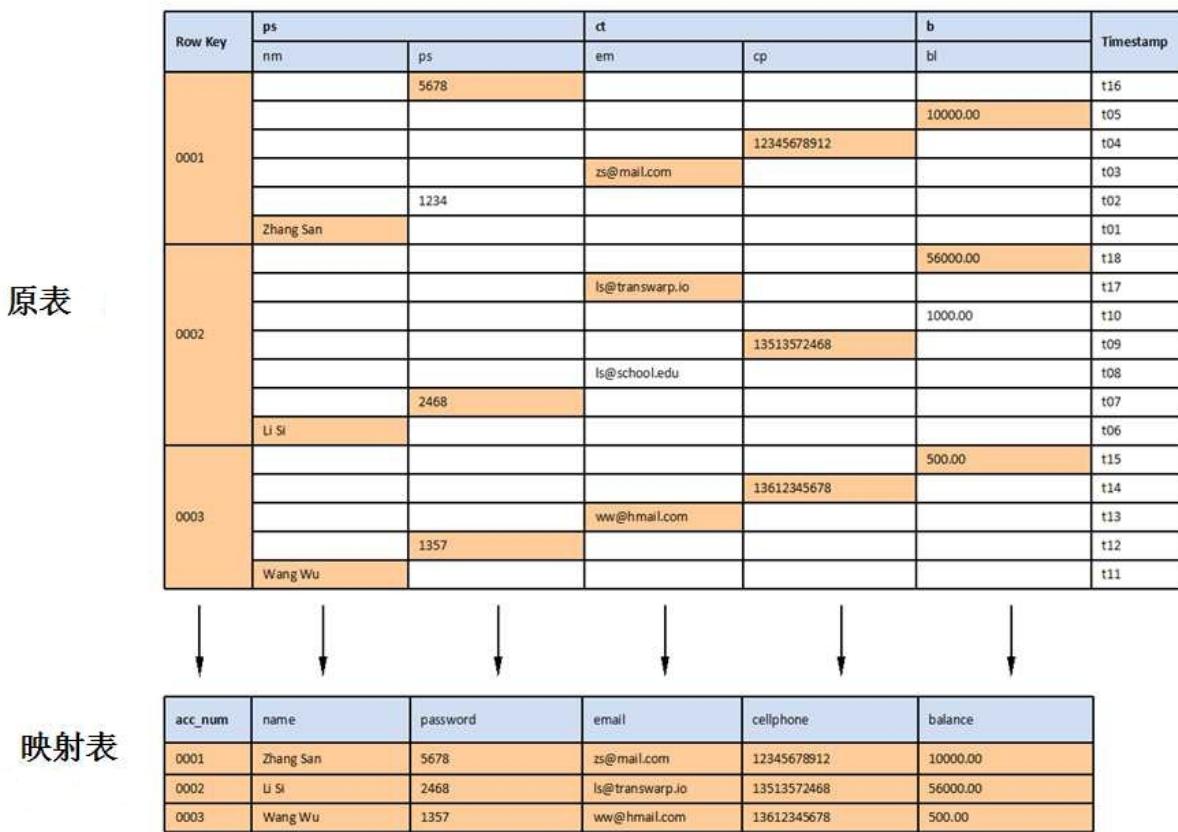


图 3. Transwarp HBase表映射为二维表

3.3. Transwarp HBase中的索引

Transwarp HBase中增加了下面几种索引：

- 全局索引（Global Index）**：全局索引中的数据做为另一张相对独立的HBase表存在，它的row key即为原表的索引字段，整个过程对用户完全透明的完成。
- 本地索引（Local Index）**：本地索引中的数据作为一个Column Family保存在原表内部，从索引到取原表数据这一步可以节省网络开销。HBase表不支持本地索引，仅Hyperdrive表支持本地索引。
- 全文索引（Fulltext Index）**：全文索引利用Elasticsearch作为索引数据的存储，用于加速对指定字段的模糊查询。

HBase表中的数据是按照Row Key排列的，这是的对row key的查询非常高效。Transwarp HBase中索引的设计是为了弥补Apache HBase只能利用原表row key的条件查询的局限。索引的Row Key为查询字段和原表Row Key的拼接，这样，通过合理地设计索引，Transwarp HBase可以加速对任意字段的查询。

索引的创建和删除

创建索引时，索引数据将由原表中的哪个或哪些字段生成。那么索引就可以加速对这个字段或这些字段的查询。一张表可以有多个全局索引或者本地索引，但是只能有一个全文索引。

索引的创建和删除可以通过[Transwarp HBase Shell命令](#), [SQL](#)以及[API](#)。

索引数据的生成

刚创建好的索引中没有数据。索引中的数据在两种情况下生成：

- 原表中有新的数据插入时，Transwarp HBase会为新插入的数据自动生成索引。
- 对索引执行 **重建（rebuild）** 操作，Transwarp HBase会为表中已有的所有数据生成索引。索引的重建需要通过[Transwarp HBase Shell命令](#)或者[API](#)执行。

索引的使用

在使用SQL查询时，您可以在SQL中指定使用哪个索引来加速查询，相关的SQL语法请参考[指定HBase映射表的索引](#)和[指定Hyperdrive映射表的索引](#)。

4. Transwarp HBase简明教程

本章介绍Shell和SQL交互的最基础的使用方法，目的是让您可以迅速地熟悉Transwarp HBase的使用方式。详细的Shell指令和SQL手册请参考[Transwarp HBase Shell命令](#)和[Transwarp HBase SQL使用说明](#)。

4.1. Transwarp HBase Shell简明教程

在安装了Transwarp HBase的节点上执行 `hbase shell` 操作可以进入命令行。下面介绍如何通过命令行进行一些简单操作，包括建表，填入数据和删除数据。



和SQL不同，Transwarp HBase Shell指令区分大小写，例如 `create` 指令不能写成 `CREATE`。

4.1.1. 建表

语法：`create`

```
create '<table>', '<column_family>' [, '<column_family>', ...]
```

现在我们以前文中提到的bank_info为例建表：

Shell指令创建bi表

```
create 'bi', 'ps', 'ct', 'bl'
```

说明：Transwarp HBase中表、列族和列限定符名都要尽量短，以减少读写时的I/O负载。所以我们将bank_info, personal, contact, balance缩短为bi, ps, ct和bl。建表以后，可以通过 `describe` 查看表的描述：

```
describe 'bi'
DESCRIPTION
ENABLED
'bi',
{NAME => 'bl', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW',
REPLICATION_SCOPE => '0', COMPRESSION => 'NONE', VERSIONS => '1', TTL =>
'FOR true EVER', MIN_VERSIONS => '0', KEEP_DELETED_CELLS => 'false',
BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'},
{NAME => 'ct', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW',
REPLICATION_SCOPE => '0', COMPRESSION => 'NONE', VERSIONS => '1', TTL =>
'FOREVER', MIN_VERSIONS => '0', KEE
P_DELETED_CELLS => 'false', BLOCKSIZE => '65536', IN_MEMORY => 'false',
BLOCKCACHE => 'true'},
{NAME => 'ps', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW',
REPLICATION_SCOPE => '0', COMPRESSION => 'NONE', VERSIONS => '1', TTL =>
'FOREVER', MIN_VERSIONS => '0', KEEP_DELETED_CELLS => 'false', BLOCKSIZE
=> '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
```

表的描述由表中的列族属性组成，各列族的属性由 `{}` 归为一组。`{}` 内，属性的赋值由 `ATTRIBUTE => 'value'` 表示（注意，属性值要放在单引号中），例如 `NAME => 'bl'` 指定了列族的名称。建表时，可以通过直接指定列族属性来定义列族，`NAME` 为必须赋值的属性，其余属性有预设好的默认值。所以上面的建表语句也可以像下面这样写：

```
create 'bi', {NAME=>'ps'}, {'NAME'=>'ct'}, {'NAME'=>'bl'}
```

4.1.2. 填入数据

下面，我们用 `put` 指令向表内填入数据：

语法：`put`

```
put '<table>', '<row_key>', '<column_family:column_qualifier>',
  '<cell_value> [, <timestamp>]
```

说明：末尾的 `timestamp` 是可选的。如果用户不指定 `timestamp`, `timestamp` 默认使用数据写入的时间戳。我们建议用户不要自己定义 `timestamp`。

下面向bi表中填入数据：

向bi表 `put` 数据

```
put 'bi', '0001', 'ps:nm', 'Zhang San'
put 'bi', '0001', 'ps:pw', '1234'
put 'bi', '0001', 'ct:em', 'zs@mail.com'
put 'bi', '0001', 'ct:cp', '12345678912'
put 'bi', '0001', 'bl:bl', '10000.00'
put 'bi', '0002', 'ps:nm', 'Li Si'
put 'bi', '0002', 'ps:pw', '2468'
put 'bi', '0002', 'ct:em', 'ls@school.edu'
put 'bi', '0002', 'ct:cp', '13513572468'
put 'bi', '0002', 'bl:bl', '1000.00'
put 'bi', '0003', 'ps:nm', 'Wang Wu'
put 'bi', '0003', 'ps:pw', '1357'
put 'bi', '0003', 'ct:em', 'ww@hmail.com'
put 'bi', '0003', 'ct:cp', '13612345678'
put 'bi', '0003', 'bl:bl', '500.00'
put 'bi', '0001', 'ps:pw', '5678' ①
put 'bi', '0002', 'ct:em', 'ls@sample.com' ②
put 'bi', '0002', 'bl:bl', 56000 ③
```

① 用户Zhang San修改密码

② 用户Li Si修改邮箱

③ 用户Li Si的存款发生改变

4.1.3. 扫描整张表

扫描整张表（查看表中所有数据）中数据的指令为 `scan`:

扫描bi表

```

scan 'bi'
ROW
  001
value=10000.00
  001
value=12345678912
  001
value=zs@mail.com
  001
value=Zhang San
  001
value=5678
  002
value=56000
  002
value=13513572468
  002
value=ls@sample.com
  002
value=Li Si
  002
value=2468
  003
value=500.00
  003
value=13612345678
  003
value=ww@hmail.com
  003
value=Wang Wu
  003
value=1357

```

COLUMN+CELL

```

column=bl:bl, timestamp=1422973263541,
column=ct:cp, timestamp=1422973240271,
column=ct:em, timestamp=1422973219713,
column=ps:nm, timestamp=1422973047378,
column=ps:pw, timestamp=1422973504458,
column=bl:bl, timestamp=1422973543652,
column=ct:cp, timestamp=1422973339893,
column=ct:em, timestamp=1422973526791,
column=ps:nm, timestamp=1422973288836,
column=ps:pw, timestamp=1422973299902,
column=bl:bl, timestamp=1422973464663,
column=ct:cp, timestamp=1422973432593,
column=ct:em, timestamp=1422973416242,
column=ps:nm, timestamp=1422973393169,
column=ps:pw, timestamp=1422973405207,

```

注意，`scan` 只会显示单元格值中拥有最新 `timestamp` 的版本。

4.1.4. 删除表

在删除一张表之前，要先用 `disable` 将这张表下线，下线后使用 `drop` 删除表。

下线和删除bi表

```

disable 'bi'
drop 'bi'

```

4.2. Transwarp HBase SQL简明教程

SQL中对HBase映射表的DDL和Hyperdrive映射表的DDL不同，下面分别介绍。

4.2.1. 连接到Inceptor Engine

您需要先连接到Transwarp Data Hub中的Inceptor Engine才能使用SQL和Transwarp HBase交互。Inceptor Engine的连接方式有很多种，如果您想要跟随本手册学习Hyperdrive SQL，您可以直接在您的集群上使用命令行连接。其他的Inceptor Engine连接方式可以参考《Inceptor使用手册》。

如果Inceptor Engine使用InceptorServer 1，打开命令行的命令为：

```
transwarp -t -h <server_host>
```

这里，`<server_host>` 是您要使用的Inceptor Engine所在的节点的hostname或者ip。

如果您使用 InceptorServer 2，您连接Inceptor Engine的方式根据您的Inceptor Engine服务使用的认证方式有关：

- 没有认证，执行：

```
beeline -u "jdbc:hive2://<server_host>:10000/<database>"
```

这里，`<database>` 处提供您想要连接到的Inceptor Engine数据库的名字，比如`default`。连接成功后您还可以在命令行中使用 `USE <database>` 来切换使用的数据库。

- LDAP认证，执行：

```
beeline -u "jdbc:hive2://<server_host>:10000/default" -n <username> -p <password>
```

在 `<username>` 和 `<password>` 处分别提供登陆的用户名和密码，比如下面命令以hive 用户身份连接localhost上的Inceptor Engine中的 `default` 数据库：

```
beeline -u "jdbc:hive2://localhost:10000/default" -n hive -p 123
```

- Kerberos认证，执行：

```
beeline -u "jdbc:hive2://<server_host>:10000/<database>;principal=<principal_name>"
```

`<principal_name>` 处填写您想要连接的Inceptor Engine的principal。比如下面命令用hive@tw-node119@TDH这个principal连接本地的Inceptor Engine中的default数据库：

```
beeline -u "jdbc:hive2://localhost:10000/default;principal=hive@tw-node119@TDH"
```

注意，在Kerberos认证的情况下，您的服务器上需要有一张有效的Kerberos TGT才能连接到Inceptor Engine。

4.2.2. HBase映射表

下面SQL创建一张Transwarp HBase Shell简明教程中使用Shell创建的bi表的HBase映射表：

创建bank_info的HBase映射表

```
CREATE EXTERNAL TABLE bank_info
(acc_num STRING, name STRING, password STRING, email STRING, cellphone
STRING, balance DOUBLE)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler' ①
WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key, ps:nm, ps:pw, ct:em,
ct:cp, bl:bl") ②
TBLPROPERTIES ("hbase.table.name" = "bi"); ③
```

- ① 通过SQL创建HBase表要用 **STORED BY** 指定使用HBase Storage Handler (**org.apache.hadoop.hive.hbase.HBaseStorageHandler** 为HBase Storage Handler使用的类名)。
- ② 定义映射表bank_info和bi表的列映射关系。
- ③ 指定映射表bank_info对应bi表。



HBase表支持的类型：BOOLEAN, TINYINT, SMALLINT, INT, BIGINT, DATE, TIMESTAMP, FLOAT, DOUBLE, STRING, VARCHAR, DECIMAL 和 STRUCT。

这时，我们可以通过bank_info直接读取前文中 **put** 进bi表中的数据：

查看bank_info中的数据

```
SELECT * FROM bank_info;
0001  Zhang San      5678    zs@mail.com    12345678912    10000.0
0002  Li Si          2468    ls@sample.com  13513572468    56000.0
0003  Wang Wu        1357    ww@hmail.com   13612345678    500.0
```

您可以对bank_info使用Inceptor SQL进行查询，例如下面是一个简单的过滤：

```
SELECT * FROM bank_info WHERE acc_num = '0003';
0003  Wang Wu        1357    ww@hmail.com   13612345678    500.0
```

4.2.3. Hyperdrive映射表

本节介绍Hyperdrive映射表的建表、建索引、插入数据和利用索引的查询。

建Hyperdrive映射表hyper_employee

```
CREATE TABLE hyper_employee
(key BIGINT, id STRING, name STRING, obd TIMESTAMP)
STORED AS HYPERDRIVE; ①
```

① 使用 `STORED AS HYPERDRIVE` 来指定建Hyperdrive映射表。



Hyperdrive表支持的类型: TINYINT, SMALLINT, INT, BIGINT, BOOLEAN, TIMESTAMP, DECIMAL, FLOAT, DOUBLE, STRING, VARCHAR, DATE, STRUCT。

下面为hyper_employee建本地索引、全局索引和全文索引。

建Hyperdrive表hyper_employee的本地索引

```
CREATE LOCAL INDEX id_local_index ON hyper_employee (id SEGMENT LENGTH 8);
```

上面语句用id列建局部索引。这里 `SEGMENT LENGTH` 指定id列在索引词条中所占字段的长度。`SEGMENT LENGTH` 当且仅当 使用STRING类型的列建索引时指定，其他类型的列建索引无需也不支持指定 `SEGMENT LENGTH`。

建Hyperdrive表hyper_employee的全局索引

```
CREATE GLOBAL INDEX id_global_index ON hyper_employee (id(8));
```

上面语句用id列建全局索引。[建Hyperdrive表hyper_employee的本地索引](#)中的 `id SEGMENT LENGTH 8` 也可以用这里的 `id(8)` 来代替，两者意思完全相同。

建Hyperdrive表hyper_employee的全文索引

```
CREATE FULLTEXT INDEX ON hyper_employee (id docValues[TRUE]) SHARD NUM 10;
```

建全文索引无需像局部和全局索引一样指定索引名——每张表最多只能有一个全文索引。另外，全文索引也无需指定STRING类型列的SEGMENT LENGTH。全文索引需要在创建时用`SHARD NUM` 指定全文索引的分片数。

要查看索引，您可以使用 `DESCRIBE FORMATTED <table>` 来查看表的元数据，索引信息包含在表的元数据之中。

向hyper_employee表 `INSERT` 数据

```
INSERT INTO hyper_employee(key, id, name, obd) VALUES (1, '123', 'Alice',
'2015-01-04 00:09:00');
INSERT INTO hyper_employee(key, id, name, obd) VALUES (2, '124', 'Bob',
'2015-01-05 00:09:00');
```

您可以对Hyperdrive映射表使用Inceptor SQL进行查询。另外，Hyperdrive表还支持使用提示在查询时指定索引辅助查询。在指定索引查询时，必须要为表起化名，并在提示中使用表化名。

查询时指定本地索引

```
SELECT /*+USE_INDEX(e USING id_local_index)*/ * FROM hyper_employee e WHERE id = '123';
```

查询时指定全局索引

```
SELECT /*+USE_INDEX(e USING id_global_index)*/ * FROM hyper_employee e WHERE id = '123';
```

查询时指定全文索引

```
SELECT /*+USE_INDEX(e USING FULLTEXT)*/ * FROM hyper_employee e WHERE id = '123';
```

删除全局索引

```
DROP INDEX id_global_index ON hyper_employee;
```

删除局部索引

```
DROP INDEX id_local_index ON hyper_employee;
```

删除全文索引

```
DROP FULLTEXT INDEX ON hyper_employee;
```

5. Transwarp HBase Schema设计

5.1. 模式(Schema) 创建

可以使用Transwarp HBase shell或Java API的HBaseAdmin来创建和编辑Transwarp HBase模式。

当修改列族（或其他表元属性）时，建议先将这张表下线（disable）。如：

```
Configuration config = HBaseConfiguration.create();
HBaseAdmin admin = new HBaseAdmin(conf);
String table = "myTable";

admin.disableTable(table);

HColumnDescriptor cf1 = ...;
admin.addColumn(table, cf1);           // 增加新的列族
HColumnDescriptor cf2 = ...;
admin.modifyColumn(table, cf2);       // 改动列族

admin.enableTable(table);
```

5.1.1. 模式更新

当表或列族改变时(如压缩格式，编码方式，block大小)，这些改变将会在下次major compaction及StoreFiles重写时起作用。

5.2. 表模式经验法则

不同的数据集可能有着不同的访问模式和服务级期望，下面这些经验法则只是一些概述：

- region规模大小在10到50GB之间；
- 单元的大小不要超过10MB，如果使用 **Object Store**，可放宽到50MB；不然，可以考虑将单元数据存在HDFS中，或者在Transwarp HBase中存一个指向这些数据的指针；



更多Object Store使用方法详见[Object Store使用方法](#)

- 一个典型的模式每个表中含有1~3个列族。Transwarp HBase表不应该设计成类似RDBM的样式；
- 对于只有1~2个列族的表，50到100个region是一个比较合适的数量。需要提醒的是，每个region都是列族的一个连续段；
- 列族的名字越短越好，因为对每个值（忽略前缀编码， **prefix encoding**），列族名都会存一次。它们不应当像典型RDNMS一样自记录（**self-documenting**）和描述。
- 如果在基于时间的机器上存储数据或日志信息，行键（Row Key）是由设备ID或服务器ID加上时间得到的，那最后能得到这样的模式：除了某个特定的时间段，旧的数据region没有额外的写。在这种情况下，得到的是少量的活跃region和大量的没有新写入的旧region。这时由于资源消耗仅来自于活跃的region，大量

的region能被容纳接受;

5.3. 列族的数量

现在Transwarp HBase并不能很好的处理两个或者三个以上的列族，所以尽量让列族数量少一些。目前，**flush**和**compaction**操作是针对一个region。所以当一个列族操作大量数据的时候会引发一个flush。那些邻近的列族也有进行flush操作，尽管它们没有操作多少数据。**compaction**操作现在是根据一个列族下的全部文件的数量触发的，而不是根据文件大小触发的。当很多的列族在flush和compaction时，会造成很多没用的I/O负载(要想解决这个问题，需要将flush和compaction操作只针对一个列族)。

尽量在模式中只针对一个列族操作。将使用率相近的列归为一列族，这样每次访问时就只用访问一个列族，提高效率。

5.3.1. 列族的基数 (Cardinality)

如果一个表存在多个列族，要注意列族之间基数(如行数)相差不要太大。例如列族A有100万行，列族B有10亿行，按照行键切分后，列族A可能被分散到很多很多region(及RegionServer)，这导致扫描列族A十分低效。

5.4. 行键(RowKey)设计

5.4.1. Hotspotting

HBase的行由行键按字典顺序排序，这样的设计优化了扫描，允许存储相关的行或者那些将被一起读的邻近的行。然而，设计不好的行键是导致hotspotting的常见原因。当大量的客户端流量(**traffic**)被定向在集群上的一个或几个节点时，就会发生hotspotting。这些流量可能代表着读、写或其他操作。流量超过了承载该region的单个机器所能负荷的量，这就会导致性能下降并有可能造成region的不可用。在同一RegionServer上的其他region也可能会受到其不良影响，因为主机无法提供服务所请求的负载。设计使集群能被充分均匀地使用的数据访问模式是至关重要的。

为了防止在写操作时出现 **hotspotting**，设计行键时应该使得数据尽量同时往多个region上写，而避免只向一个region写，除非那些行真的有必要写在一个region里。下面介绍了集中常用的避免hotspotting的技巧，它们各有优劣：

5.4.1.1. Salting

Salting 从某种程度上看与加密无关，它指的是将随机数放在行键的起始处。进一步说，salting给每一行键随机指定了一个前缀来让它与其他行键有着不同的排序。所有可能前缀的数量对应于要分散数据的region的数量。如果有几个“hot”的行键模式，而这些模式在其他更均匀分布的行里反复出现，salting就能到帮助。下面的例子说明了salting能在多个RegionServer间分散负载，同时也说明了它在读操作时候的负面影响。

假设行键的列表如下，表按照每个字母对应一个region来分割。前缀‘a’是一个region，‘b’就是另一个region。在这张表中，所有以‘f’开头的行都属于同一个region。这个例子关注的行和键如下：

```
foo0001
foo0002
foo0003
foo0004
```

现在，假设想将它们分散到不同的region上，就需要用到四种不同的 **salts** : a, b, c, d。在这种情况下，每种字母前缀都对应着不同的一个region。用上这些salts后，便有了下面这样的行键。由于现在想把它们分到四个独立的区域，理论上吞吐量会是之前写到同一region的情况的四倍。

```
a-foo0003
b-foo0001
c-foo0004
d-foo0002
```

如果想新增一行，新增的一行会被随机指定四个可能的salt值中的一个，并放在某条已存在的行的旁边。

```
a-foo0003
b-foo0001
c-foo0003
c-foo0004
d-foo0002
```

由于前缀的指派是随机的，因而如果想要按照字典顺序找到这些行，则需要做更多的工作。从这个角度看，salting增加了写操作的吞吐量，却也增大了读操作的开销。

5.4.1.2. Hashing

可用一个单向的 **hash** 散列来取代随机指派前缀。这样能使一个给定的行在“salted”时有相同的前缀，从某种程度上说，这在分散了RegionServer间的负载的同时，也允许在读操作时能够预测。确定性hash（**deterministic hash**）能让客户端重建完整的行键，以及像正常的一样用Get操作重新获得想要的行。

考虑和上述salting一样的情景，现在可以用单向hash来得到行键foo0003，并可预测得‘a’这个前缀。然后为了重新获得这一行，需要先知道它的键。可以进一步优化这一方法，如使得将特定的键对总是在相同的region。

5.4.1.3. 反转键（Reversing the Key）

第三种预防hotspotting的方法是反转一段固定长度或者可数的键，来让最常改变的部分（最低显著位，**the least significant digit**）在第一位，这样有效地打乱了行键，但是却牺牲了行排序的属性。

5.4.2. 单调递增行键/时序数据

在Tom White的 [Hadoop: The Definitive Guide](#) 一书中，有个章节描述了一个值得注意的问题：在一个集群中，一个导入数据的进程锁住不动，所有的client都在等待一个region(因而也就是一个单个节点)，过了一会后，变成了下一个region…如果使用了单调递增或者时序的key便会造成这样的问题。详情可以参见IKai画的漫画 [monotonically increasing values are bad](#)。使用了顺序的key会将本没有顺序的数据变得有顺序，把负载压在一台机器上。所以要尽量避免时间戳或者序列(e. g. 1, 2, 3)这样的行键。

如果需要导入时间顺序的文件(如log)到Transwarp HBase中，可以学习OpenTSDB的做法。它有一个页面来描述它的HBase模式。OpenTSDB的Key的格式是[metric_type][event_timestamp]，乍一看，这似乎违背了不能将timestamp做key的建议，但是它并没有将timestamp作为key的一个关键位置，有成百上千的metric_type就足够将压力分散到各个region了。因此，尽管有着连续的数据输入流，Put操作依旧能被分散在表中的各个region中。

5.4.3. 尽量最小化行和列的大小

在Transwarp HBase中，值是作为一个单元(Cell)保存在系统的中的，要定位一个单元，需要行，列名和时间戳。通常情况下，如果行和列的名字要是太大(甚至比value的大小还要大)的话，可能会遇到一些有趣的情况。在Transwarp HBase的存储文件([storefiles](#))中，有一个索引用来方便值的随机访问，但是访问一个单元的坐标要是太大的话，会占用很大的内存，这个索引会被用尽。要想解决这个问题，可以设置一个更大的块大小，也可以使用更小的行和列名。压缩也能得到更大指数。

大部分时候，细微的低效不会影响很大。但不幸的是，在这里却不能忽略。无论是列族、属性和行键都会在数据中重复上亿次。

5.4.3.1. 列族

尽量使列族名小，最好一个字符。(如 "d" 表示 data/default).

5.4.3.2. 属性

详细属性名(如，“myVeryImportantAttribute”)易读，最好还是用短属性名(e.g., “via”)保存到HBase.

5.4.3.3. 行键长度

让行键短到可读即可，这样对获取数据有帮助(e.g., Get vs. Scan)。短键对访问数据无用，并不比长键对get/scan更好。设计行键需要权衡。

5.4.3.4. 字节模式

long类型有8字节。8字节内可以保存无符号数字到18,446,744,073,709,551,615。如果用字符串保存——假设一个字节一个字符——需要将近3倍的字节数。

下面是示例代码，可以自己运行一下。

```

// long
//
long l = 1234567890L;
byte[] lb = Bytes.toBytes(l);
System.out.println("long bytes length: " + lb.length);    // returns 8

String s = "" + l;
byte[] sb = Bytes.toBytes(s);
System.out.println("long as string length: " + sb.length);    // returns 10

// hash
//
MessageDigest md = MessageDigest.getInstance("MD5");
byte[] digest = md.digest(Bytes.toBytes(s));
System.out.println("md5 digest bytes length: " + digest.length);    // returns 16

String sDigest = new String(digest);
byte[] sbDigest = Bytes.toBytes(sDigest);
System.out.println("md5 digest as string length: " + sbDigest.length);
// returns 26

```

不幸的是，用二进制表示会使数据在代码之外难以阅读。下例便是当需要增加一个值时会看到的shell：

```

hbase(main):001:0> incr 't', 'r', 'f:q', 1
COUNTER VALUE = 1

hbase(main):002:0> get 't', 'r'
COLUMN CELL
      f:q timestamp=1369163040570,
value=\x00\x00\x00\x00\x00\x00\x00\x00\x01
1 row(s) in 0.0310 seconds

```

这个shell尽力在打印一个字符串，但在这种情况下，它决定只将进制打印出来。当在region名内行键会发生相同的情况。如果知道储存的是什么，那自是没问题，但当任意数据都可能被放到相同单元的时候，这将会变得难以阅读。这是最需要权衡之处。

5.4.4. 倒序时间戳

一个数据库处理的通常问题是找到最近版本的值。采用倒序时间戳作为键的一部分可以对此特定情况有很大帮助。该技术包含追加(`Long.MAX_VALUE - timestamp`) 到key的后面，如 `[key] [reverse_timestamp]` 。

表内[key]的最近的值可以用[key]进行Scan，找到并获取第一个记录。由于Transwarp HBase行键是排序的，该键排在任何比它老的行键的前面，所以是第一个。

该技术可以用于代替**版本数**，其目的是保存所有版本到“永远”（或一段很长时间）。同时，采用同样的Scan技术，可以很快获取其他版本。

5.4.5. 行键和列族

行键在列族范围内。所以同样的行键可以在同一个表的每个列族中存在而不会冲突。

5.4.6. 行键永远不变

行键不能改变。唯一可以“改变”的方式是删除然后再插入。这是一个常问问题，所以要注意开始就要让行键正确（且/或在插入很多数据之前）。

5.4.7. 行键和region split之间的关系

如果已经 **pre-split**（预裂）了表，接下来关键要了解行键是如何在region边界分布的。为了说明为什么这很重要，可考虑用可显示的16位字符作为键的关键位置（e.g., "0000000000000000" to "ffffffffffffffff"）这个例子。通过 Bytes.split来分割键的范围（这是当用 `Admin.createTable(byte[] startKey, byte[] endKey, numRegions)` 创建region时的一种拆分手段），这样会分得10个region。

```

48 48 48 48 48 48 48 48 48 48 48 48 48 48 48 48 // 0
54 -10 -10 -10 -10 -10 -10 -10 -10 -10 -10 -10 -10 -10 -10 -10 // 6
61 -67 -67 -67 -67 -67 -67 -67 -67 -67 -67 -67 -67 -67 -67 -68 // =
68 -124 -124 -124 -124 -124 -124 -124 -124 -124 -124 -124 -124 -124 -124 -124 -126 // D
75 75 75 75 75 75 75 75 75 75 75 75 75 75 75 72 // K
82 18 18 18 18 18 18 18 18 18 18 18 18 18 18 14 // R
88 -40 -40 -40 -40 -40 -40 -40 -40 -40 -40 -40 -40 -40 -40 -40 -44 // X
95 -97 -97 -97 -97 -97 -97 -97 -97 -97 -97 -97 -97 -97 -97 -97 -102 // _
102 102 102 102 102 102 102 102 102 102 102 102 102 102 102 102 // f

```

但问题在于，数据将会堆放在前两个region以及最后一个region，这样就会导致某几个region由于数据分布不均匀而特别忙。为了理解其中缘由，需要考虑ASCII Table的结构。根据ASCII表，“0”是第48号，“f”是102号；但58到96号是个巨大的间隙，考虑到在这里仅[0-9]和[a-f]这些值是有意义的，因而这个区间里的值不会出现在键空间（**keyspace**），进而中间区域的region将永远不会用到。为了pre-split这个例子中的键空间，需要自定义拆分。

教程1. 预裂表（**pre-splitting tables**）是个很好的实践，但pre-split时要注意使得所有的region都能在键空间中找到对应。尽管例子中解决的问题是关于16位键的键空间，但其他任何空间也是同样的道理。

教程2. 16位键（通常用到可显示的数据中）尽管通常不可取，但只要所有的region都能在键空间找到对应，它依旧能和预裂表配合使用。

下例说明了如何为16位键预创建合适的拆分：

```

public static boolean createTable(Admin admin, HTableDescriptor table,
byte[][] splits)
throws IOException {
try {
admin.createTable( table, splits );
return true;
} catch (TableExistsException e) {
logger.info("table " + table.getNameAsString() + " already exists");
// the table already exists...
return false;
}
}

public static byte[][] getHexSplits(String startKey, String endKey, int
numRegions) {
byte[][] splits = new byte[numRegions-1][];
BigInteger lowestKey = new BigInteger(startKey, 16);
BigInteger highestKey = new BigInteger(endKey, 16);
BigInteger range = highestKey.subtract(lowestKey);
BigInteger regionIncrement = range.divide(BigInteger.
valueOf(numRegions));
lowestKey = lowestKey.add(regionIncrement);
for(int i=0; i < numRegions-1;i++) {
BigInteger key = lowestKey.add(regionIncrement.multiply
(BigInteger.valueOf(i)));
byte[] b = String.format("%016x", key).getBytes();
splits[i] = b;
}
return splits;
}
}

```

5.5. 版本数量

5.5.1. 最大版本数

行的版本的数量是HColumnDescriptor设置的，每个列族可以单独设置，默认是3。这个设置是很重要的，因为Transwarp HBase是不会去覆盖一个值的，它只会在后面在追加写，用时间戳来区分、过早的版本会在执行major compaction时删除，这些在[Transwarp HBase数据模型](#)有描述。这个版本的值可以根据具体的应用增加减少。

不推荐将版本最大值设到一个很高的水平（如，成百或更多），除非老数据对你很重要。因为这会导致存储文件变得极大。

5.5.2. 最小版本数

和行的最大版本数一样，最小版本数也是通过HColumnDescriptor 在每个列族中设置的。最小版本数缺省值是0，表示该特性禁用。最小版本数参数和存活时间一起使用，允许配置如“保存最后T秒有价值数据，最多N个版本，但最少约M个版本”（M是最小版本数，M<N）。该参数仅在存活时间对列族启用，且必须小于行版本数。

5.6. 支持数据类型

Transwarp HBase通过Put和Result支持“bytes-in/bytes-out”接口，所以任何可被转为字节数组的东西可以作为值存入。输入可以是字符串、数字、复杂对象、甚至图像，它们能转为字节。

存在值的实际长度限制（如：保存10–50MB对象到Transwarp HBase可能对查询来说太长）；搜索邮件列表获取本话题的对话。Transwarp HBase的所有行都遵循[Transwarp HBase数据模型](#)包括版本化。设计时需考虑到这些，以及列族的块大小。

5.7. 存活时间 (TTL)

列族可以设置TTL秒数，Transwarp HBase在超时后将自动删除数据。影响全部行的全部版本 – 甚至当前版本。Transwarp HBase里面TTL时间时区是UTC.

存储文件仅包含有过期的行（expired rows），它们可通过minor compaction删除。设置 `hbase.store.delete.expired.storefile to false` 可禁用此功能；将最小版本数设置成非0值也可达到同样的效果。

Transwarp HBase的最新版本还支持将设定的时间存放在每个结构单元。TTL单元通过Mutation#setTTL作为变更请求（Appends, Increments, Puts, etc.）的一个属性提交，如果TTL的属性被设定了，它将会应用到由于该变更操作更新的所有单元上。cell TTL handling和ColumnFamily TTLs间有两个显著的差别：

1. Cell TTLs的数量级是毫秒而不是秒；
2. 一个cell TTL不能超出ColumnFamily TTLs设置的有效时间。

5.8. 保留删除的单元

默认情况下，删除标记（**delete markers**）会回到起始时间。因此，Get或Scan操作不能看到删除的单元（行或列），甚至当Get或Scan操作在删除单元之前发生，它们也不能看到。

列族会选择性地保留删除单元。在这种情况下，只要操作在删除的时间戳之前，删除单元仍能被重新获得，这使得在删除进行时能进行即时查询。

删除的单元仍然受TTL控制，并永远不会超过“最大版本数”被删除的单元。新“raw” scan 选项返回所有已删除的行和删除标志。

用Transwarp HBase Shell改变KEEP_DELETED_CELLS的值：

```
hbase> hbase> alter 't1', NAME => 'f1', KEEP_DELETED_CELLS => true
```

用API改变KEEP_DELETED_CELLS的值：

```
...
HColumnDescriptor.setKeepDeletedCells(true);
...
```

接下来说明设置KEEP_DELETED_CELLS对表产生的基本影响：首先：

```

create 'test', {NAME=>'e', VERSIONS=>2147483647}
put 'test', 'r1', 'e:c1', 'value', 10
put 'test', 'r1', 'e:c1', 'value', 12
put 'test', 'r1', 'e:c1', 'value', 14
delete 'test', 'r1', 'e:c1', 11

hbase(main):017:0> scan 'test', {RAW=>true, VERSIONS=>1000}
ROW                                         COLUMN+CELL
  r1                                         column=e:c1, timestamp=
  14, value=value                           column=e:c1, timestamp=
  r1                                         column=e:c1, timestamp=
  12, value=value                           column=e:c1, timestamp=
  r1                                         column=e:c1, timestamp=
  11, type=DeleteColumn                    column=e:c1, timestamp=
  r1                                         column=e:c1, timestamp=
  10, value=value                           column=e:c1, timestamp=
1 row(s) in 0.0120 seconds

hbase(main):018:0> flush 'test'
0 row(s) in 0.0350 seconds

hbase(main):019:0> scan 'test', {RAW=>true, VERSIONS=>1000}
ROW                                         COLUMN+CELL
  r1                                         column=e:c1, timestamp=
  14, value=value                           column=e:c1, timestamp=
  r1                                         column=e:c1, timestamp=
  12, value=value                           column=e:c1, timestamp=
  r1                                         column=e:c1, timestamp=
  11, type=DeleteColumn                    column=e:c1, timestamp=
1 row(s) in 0.0120 seconds

hbase(main):020:0> major_compact 'test'
0 row(s) in 0.0260 seconds

hbase(main):021:0> scan 'test', {RAW=>true, VERSIONS=>1000}
ROW                                         COLUMN+CELL
  r1                                         column=e:c1, timestamp=
  14, value=value                           column=e:c1, timestamp=
  r1                                         column=e:c1, timestamp=
  12, value=value                           column=e:c1, timestamp=
1 row(s) in 0.0120 seconds

```

注意删除单元是如何被清除的。

现在仅用KEEP_DELETED_CELLS设置来运行相同的例子：

```

hbase(main):005:0> create 'test', {NAME=>'e', VERSIONS=>2147483647,
KEEP_DELETED_CELLS => true}
0 row(s) in 0.2160 seconds

=> Hbase::Table - test
hbase(main):006:0> put 'test', 'r1', 'e:c1', 'value', 10
0 row(s) in 0.1070 seconds

```

```

hbase(main):007:0> put 'test', 'r1', 'e:c1', 'value', 12
0 row(s) in 0.0140 seconds

hbase(main):008:0> put 'test', 'r1', 'e:c1', 'value', 14
0 row(s) in 0.0160 seconds

hbase(main):009:0> delete 'test', 'r1', 'e:c1', 11
0 row(s) in 0.0290 seconds

hbase(main):010:0> scan 'test', {RAW=>true, VERSIONS=>1000}
ROW
COLUMN+CELL
  r1
column=e:c1, timestamp=14, value=value
  r1
column=e:c1, timestamp=12, value=value
  r1
column=e:c1, timestamp=11, type=DeleteColumn
  r1
column=e:c1, timestamp=10, value=value
1 row(s) in 0.0550 seconds

hbase(main):011:0> flush 'test'
0 row(s) in 0.2780 seconds

hbase(main):012:0> scan 'test', {RAW=>true, VERSIONS=>1000}
ROW
COLUMN+CELL
  r1
column=e:c1, timestamp=14, value=value
  r1
column=e:c1, timestamp=12, value=value
  r1
column=e:c1, timestamp=11, type=DeleteColumn
  r1
column=e:c1, timestamp=10, value=value
1 row(s) in 0.0620 seconds

hbase(main):013:0> major_compact 'test'
0 row(s) in 0.0530 seconds

hbase(main):014:0> scan 'test', {RAW=>true, VERSIONS=>1000}
ROW
COLUMN+CELL
  r1
column=e:c1, timestamp=14, value=value
  r1
column=e:c1, timestamp=12, value=value
  r1
column=e:c1, timestamp=11, type=DeleteColumn
  r1
column=e:c1, timestamp=10, value=value
1 row(s) in 0.0650 seconds

```

当移除删除单元的唯一因素是来自于删除标记，那么KEEP_DELETED_CELLS可以用来避免Transwarp HBase移除这些单元。当写入的版本超过设置的最大值或者TTL和Cell超过了设置的超时(`timeout`)，通过KEEP_DELETED_CELLS保留的删除单元依旧会被移除。

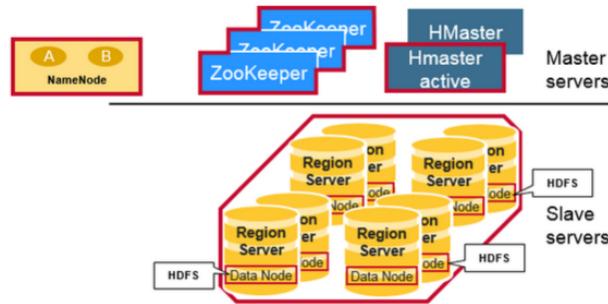
6. Transwarp HBase架构

6.1. Transwarp HBase架构组件

从物理架构上看，Transwarp HBase是包含三类服务器的主从式架构。RegionServer 负责响应用户I/O请求，并向 HDFS 中读写数据。每当访问数据时，客户端（Client）会直接与RegionServer建立连接。region的分配、DDL（创建、删除表）操作则由 HMaster 来处理。Zookeeper，它作为HDFS的一部分，负责维护集群的健康状态、避免HMaster单点问题。

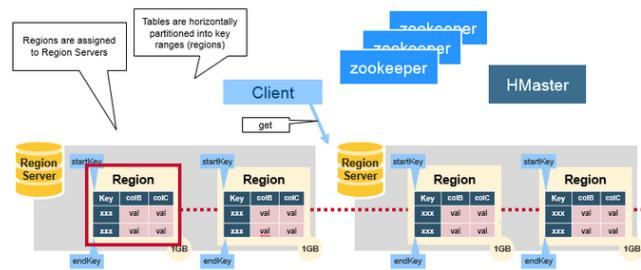
Hadoop DataNode 存储有RegionServer正在管理的数据。所有Transwarp HBase的数据都存储在HDFS文件中。RegionServer和HDFS DataNode并置，这使得由RegionServer处理的数据具有 **数据局部性**（**data locality**，数据被放在需要它的地方的附近）。Transwarp HBase的数据在写入的时候可从本地获取，但当它所属的region被移走时，则需要从远端获取数据，直到等到 **Compaction** 操作。

NameNode为组成文件的物理数据块维护着它们的 **元数据（metadata）** 信息。



6.1.1. Region

Transwarp HBase表按照 **行键（Row Key）** 被水平地分割成一个个region。每个region包含了表中从该region的开始键到结束键之间的所有行。这些region被指派给集群中的节点，这些节点便是RegionServer，它们负责数据的读写。推荐每个RegionServer管理1000个左右的region，除非它的region并不活跃。



6.1.2. Transwarp HBase HMaster

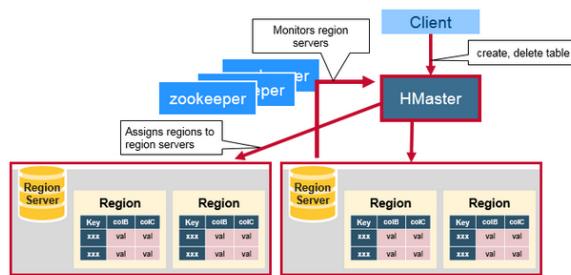
Region的分配、DDL（创建、删除表）操作由HBase Master处理。每个master的职责都包括有：

- 协调RegionServer

- 在启动或需重分配region的时候负责region的分配，以实现数据恢复和负载平衡。
- 监视所有在集群中的RegionServer（监听来自Zookeeper的消息）。

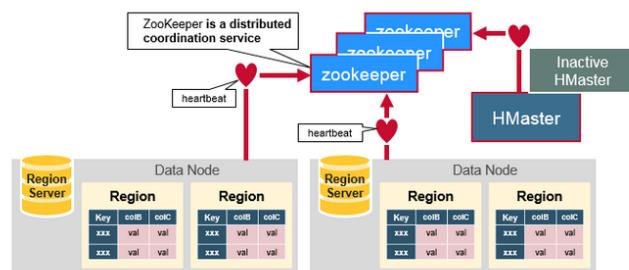
- 管理功能

- 管理用户对表的增删改查操作



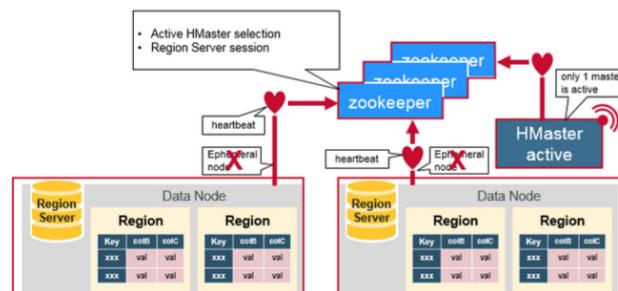
6.1.3. Zookeeper:协调者

Transwarp HBase用Zookeeper来提供分布式协调服务，以维护集群中服务器的状态。Zookeeper留存了服务器健康状态与是否可用的信息，并提供服务器故障通知。Zookeeper用consensus协议来保障共享状态。需要注意的是，一份consensus协议需要3个或5个机器的参与。



6.2. 各组件如何协调工作？

Zookeeper用来协调分布式系统成员的共享状态信息。RegionServer和处于活跃状态的HMaster通过一个 **会话 (session)** 与Zookeeper建立连接。Zookeeper通过 **heartbeat** 来为活跃的会话维护短暂的临时节点。



每个RegionServer都会创建一个短暂的临时节点。HMaster监视这些节点以发现可用的RegionServer以及可能的服务器故障。接着，HMaster也会创建一个短暂节点。Zookeeper找到第一个节点，并用它来保证只有一个master处于活跃状态。活跃的HMaster会发送heartbeat给Zookeeper；同时，不活跃的HMaster会监听着活跃HMaster故障的消息。

如果RegionServer或处于活跃状态的HMaster发送heartbeat失败，那么会话将会终止，对应的临时节点也会被删除。用于更新（updates）的监听器将被告知这些节点已被删除。同时，活跃状态的HMaster会监听着RegionServer，并将它们从故障中修复。非活跃状态的HMaster则会监听着活跃状态的HMaster是否发生故障，如果故障它们会替补上去，进而成为处于活跃状态的HMaster。

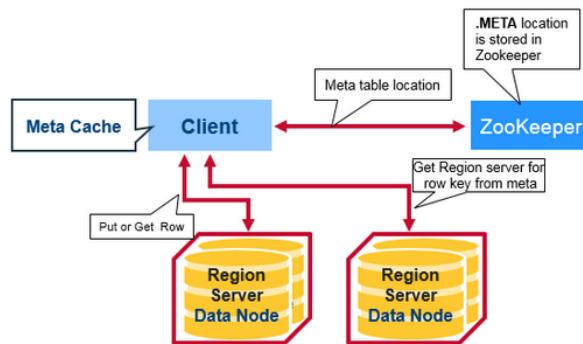
6.3. Transwarp HBase的首次读/写

META表 是一个特殊的Transwarp HBase编目表，它保存着集群中region的地址。META表的地址会保存在Zookeeper中。

当客户端首次在Transwarp HBase中读/写时，会执行如下操作：

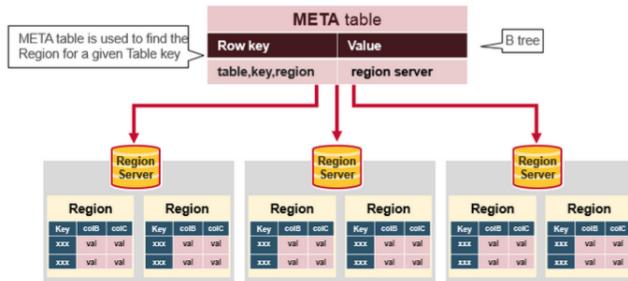
1. 客户端获取ZooKeeper中的META表所记录的RegionServer；
2. 客户端查询 **METAServer**，以找到它想要的行键位置上的RegionServer。它将会连同META表的位置一起缓存（cache）这些信息；
3. 客户端从相应的RegionServer得到对应的行键。

在以后的读取中，客户端就可以通过cache重新获得META的地址以及之前读到的行键信息。除非有由region移动所引起的miss，那么随着时间的推移，客户端就不必再去查询META表了；取而代之的是它会反复查询并不断更新cache。



6.4. Transwarp HBase Meta表

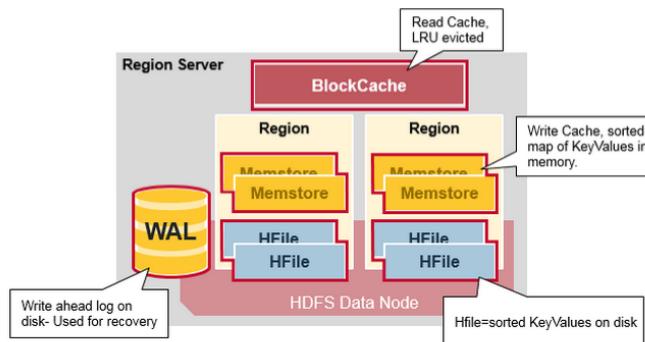
- META表是一个特殊的Transwarp HBase表，它维护着记录系统中所有region的一个列表；
- META表的数据结构类似于B树；
- META表的构架组成如下：
 - 键：region的开始键，region ID
 - 值：RegionServer



6.5. RegionServer组件

每个RegionServer都运行在HDFS的一个数据节点上，RegionServer由如下部分构成：

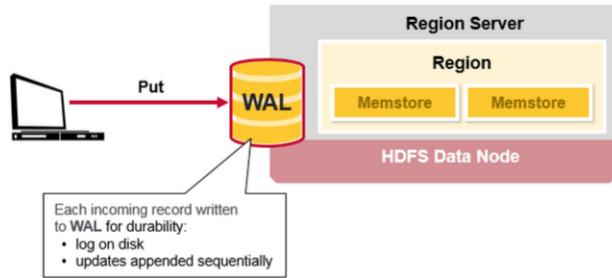
- **WAL** : Write Ahead Log是分布式文件系统上的文件。WAL保存着一些临时的新数据，以用作之后故障的修复。
- **BlockCache** : BlockCache是读操作时的cache，它保存着内存中经常被读的数据。当cache填满后，**最近最少 (Least Recently Used)** 使用的数据将被剔除。
- **MemStore** : MemStore是写操作时的cache。它保存着尚未写入磁盘的新数据，这些数据在被写入磁盘前是有序的。每个region的每一**列族 (column family)** 都有一个MemStore。
- **Hfiles** : Hfiles保存着磁盘上有序**键值对 (KeyValues)** 的行信息。



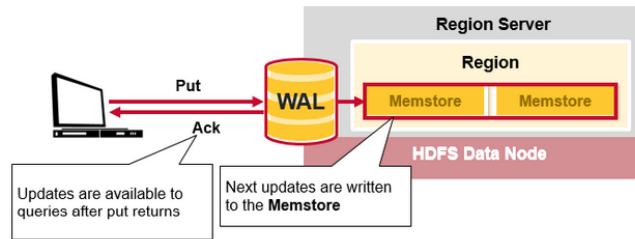
6.6. Transwarp HBase写操作

当客户端发出Put操作请求时，首先数据会被写入WAL：

- 更改被添加在磁盘上WAL文件的末端；
- WAL恢复服务器崩溃事务中的**未持久化 (not-yet-persisted)** 的数据。

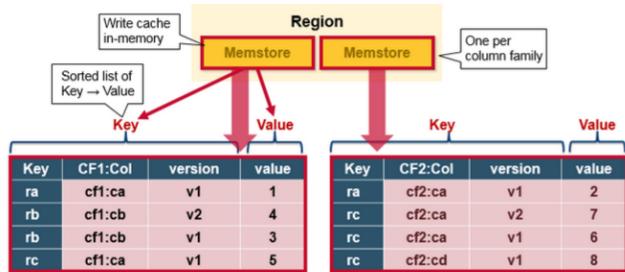


一旦数据被写入WAL，它们就会被放在MemStore中。接着，Put请求的 确认信息（ACK） 将会回复给客户端。



6.7. Transwarp HBase MemStore

MemStore将内存中的更新（updates）按照有序键值对保存，这和它在HFile中的储存是一样的模式。每一列族有一个MemStore， 更新按照每一列族有序储存。



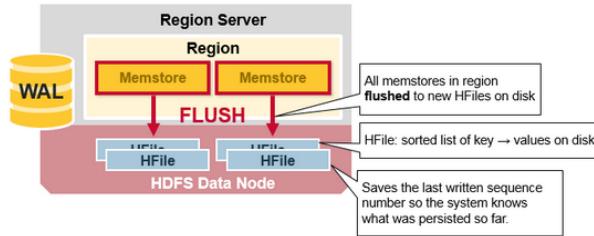
6.8. Transwarp HBase Region Flush

当MemStore积攒了足够多的数据，所有的有序集会被写入HDFS的一个新HFile中。HBase的每一列族中有很多HFiles，里面包含有一定数量的 单元数据（cells），或说键值对。当MemStores中的有序键值对被flush到磁盘上时，就生成了这些HFiles文件。

这便是HBase的列族有数量限制的原因之一。每个列族有一个MemStore，当一个MemStore被填满后，它们会全部被flush到磁盘上。MemStore仍保存有最后写入的序列号，因而系统能够知道哪些数据依旧是持久的（写在磁盘上的）。

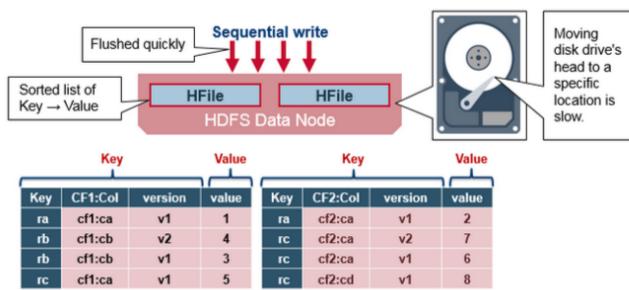
最高序列号在每个HFile中被存为一个 元字段（meta field），这些字段反映出何处持久性终止了以及应该在何处继续。当一个region启动的时候，它的序列号被读取，而它的最高序列号则被作为新更改（edits）时的

序列号。



6.9. Transwarp HBase HFile

数据被保存在HFile中，它们包含有序键/值。当MemStore积攒了足够多的数据，所有的有序集会被写入HDFS的一个新HFile中。写入是顺序写入；由于它避免了磁盘驱动器磁头的移动，所以写入的速度非常快。

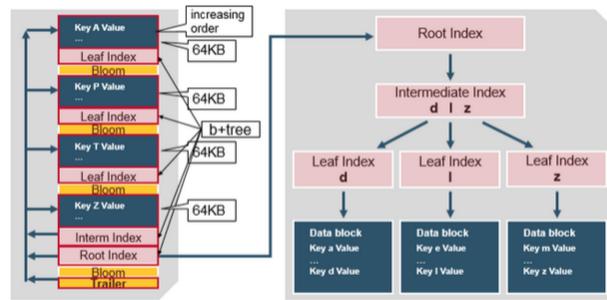


6.9.1. Transwarp HBase Region Flush

每个HFile包含有一个多层次索引（index），这使得HBase不用去读完全部的文件就能找到想要的数据。多层次索引的结构类似于一个B+树：

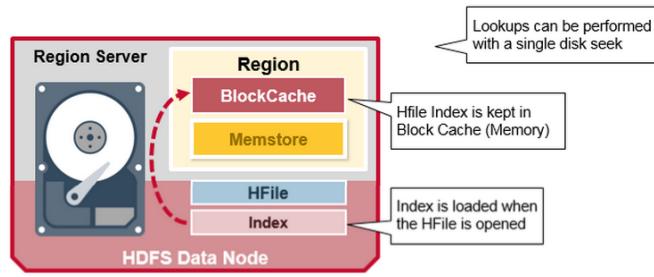
- 键值对被按照升序储存；
- 索引通过行键指向64KB-blocks（块）中的键值数据；
- 每一个block都有它自己的叶子索引；
- 每个block的最后一个键被放在中间索引（intermediate index）中；
- 根索引指向中间索引。

HFile文件是不定长的，长度固定的只有其中的两块：**Trailer** 和 **FileInfo**。Trailer指向元块（meta blocks），它被写在文件的最后。Trailer还有着类似于bloom filter的信息以及时间段信息（time range info）。Bloom filter可用于跳过那些没有确定键值的文件。时间段信息可用于跳过那些不在所需时间段内的文件。



6.9.2. HFile索引

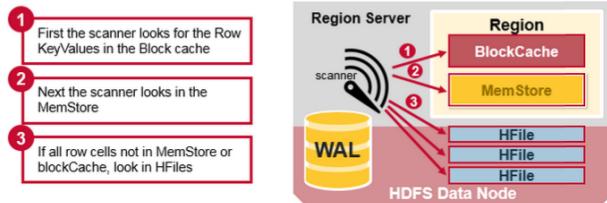
之前讨论过的“索引”在HFile被打开和保存至内存中时加载进来。这使得查找能在单个磁盘上进行。



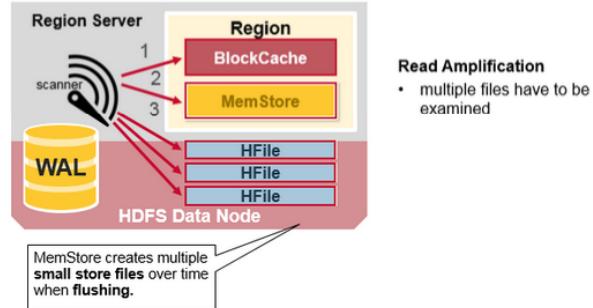
6.10. Transwarp HBase读合并（Read Merge）

之前已介绍过，每行对应的键值对单元能被放置在多处。行单元被保存在HFiles中；最近更新的行单元被放在MemStore中；最近读取单元被放在BlockCache中。那么当需要读取一行时，系统是如何找到相应的单元并返回的呢？读操作将来自BlockCache、MemStore和HFiles的键值对合并，步骤如下：

- 首先，扫描器从BlockCache中找到行单元，最近读取的键值对就被缓存在此。当内存需要时，最近最少使用（Least Recently Used）的数据将会被剔除。
- 接着，扫描器会检查MemStore，MemStore中包含有大多数最近写入的数据。
- 如果扫描器没能在BlockCache和MemStore中找全所有的行单元，HBase就会用BlockCache索引和bloom filters把Hfiles加载进内存，因为这些Hfiles中可能包含所需的行单元。



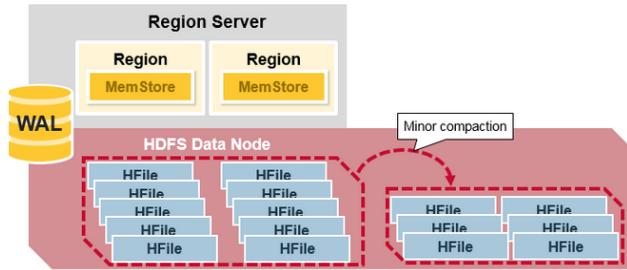
之前有提到，每个MemStore中有多个HFiles，这意味着在读操作时，需要检查多个文件，这将会影响时间性能。这被称之为 **读出放大（read amplification）**。



6.11. Transwarp HBase Compaction

6.11.1. Transwarp HBase Minor Compaction

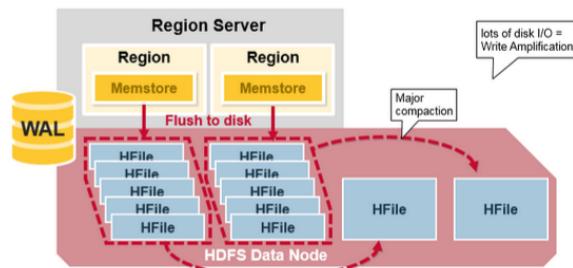
HBase会自动拾取一些较小的HFiles，并将它们重新写入一些较大的HFiles中。这个过程被称为 **minor compaction**。Minor compaction通过重写操作，利用合并排序将较小的文件转化为较大却数量较少的文件之中。



6.11.2. Transwarp HBase Major Compaction

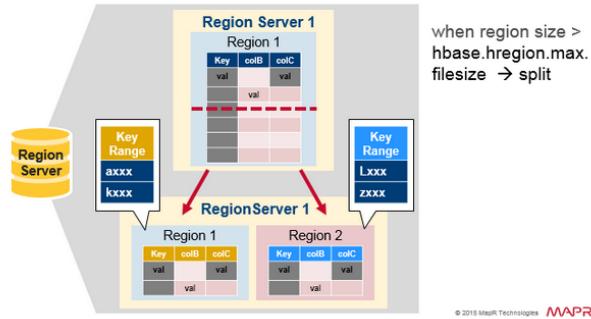
Major Compaction 将一个region里的全部HFiles，按照每一列族合并和重写成一个HFiles。在这个过程中，会将已经删除或者过期的单元剔除。这样优化了读操作的性能；然而，由于major compaction重写了所有的文件，因而在这过程中可能会出现较大的磁盘I/O和网络拥堵。这被称之为 **写入放大 (write amplification)**。

Write amplification可被设置为自动运行。由于写入放大，major compactions往往被安排在周末或者晚上进行。Major compaction能够使得那些由于服务器故障或者负载均衡而较远的数据，变得相对RegionServer处于本地。

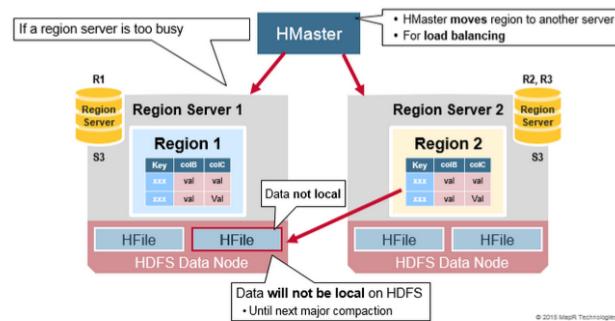


6.12. Region 拆分 (split)

起初，每个表只有一个region。当一个region逐渐变得过大时，它会分裂为两个子region。这两个子region分别代表了原region的二分之一，它们在RegionServer上被并行地打开，之后这个拆分会告知HMaster。

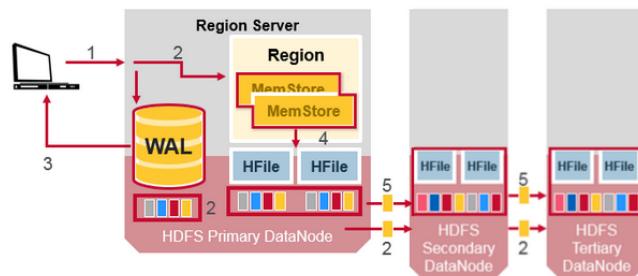


由于负载均衡的因素，HMaster可能会将新的region移到其他服务器。这导致新的在RegionServer需要从远处的HDFS节点中获取数据，除非major compactions将这些数据移动到RegionServer的本地节点。



6.13. HDFS数据备份

所有读或写操作的对象都是 **主节点 (primary node)**。HDFS备份了WAL和HFile块，HFile块的备份是自动进行的。HBases通过依赖HDFS来在存储文件时保障数据安全。当数据被写入到HDFS时，一份数据拷贝会被写在本地，接着它会被备份到二级节点，第三份拷贝会被写入叔节点。

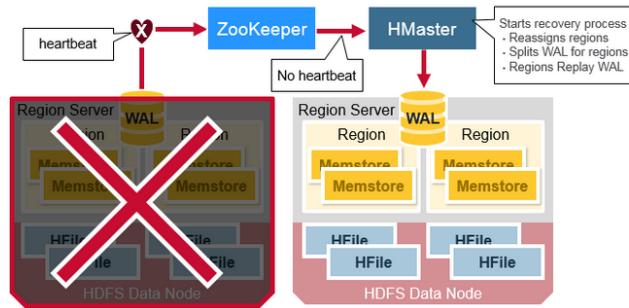


6.14. Transwarp HBase崩溃修复

当一个RegionServer故障时，在采取探测和修复操作前，崩坏的region不可用。Zookeeper会在它失

去RegionServer的heartbeats时，确定故障节点。接着HMaster将会被告知RegionServer已经故障。

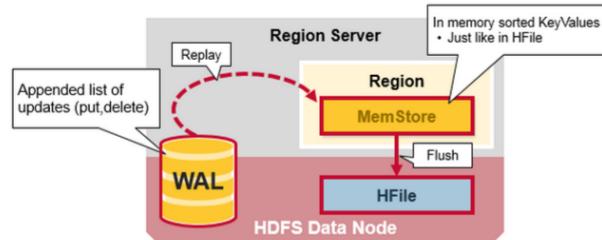
当HMaster检测到某个RegionServer出现了故障，HMaster会重新分配从故障服务器到活跃服务器的region。为了修复RegionServer的MemStore中还没保存到磁盘上的更改，HMaster会把故障服务器的WAL分成单独的文件，并将这些文件保存至新的RegionServer数据节点上。每个RegionServer从相分割出的WAL中回放（replay）WAL，来为损坏的region重建MemStore。



6.15. 数据修复

WAL文件包含有一个更改列表，每个更改（edit）表示一个单独的put或者delete操作。更改按照时间顺序记录，为了保持持久性，新增加的操作被放在磁盘上WAL文件的尾端。

如果当数据在内存中却不在HFiles中时出现了故障会怎样呢？WAL文件会回放。回放WAL通过读WAL，添加并排序当前MemStore中的更改实现。最后，MemStore会将写入更改flush进HFile。



6.16. Transwarp HBase架构评测

6.16.1. 优势

- 强一致模型
 - 当写操作返回时，所有的读者都能看到相同的值。
- 自动化规模
 - 当数据过大时，会自动分割Region
 - 利用HDFS来传播和备份数据

- 内置恢复
 - Write Ahead Log的使用
- 由Hadoop集成
 - 在HBase上运用MapReduce是容易的

6.16.2. 不足

- WAL回放较慢
- 复杂的故障修复较慢
- Major Compaction的I/O风暴

7. Transwarp HBase安全

安全模式下Transwarp HBase和Transwarp Data Hub集群中其他服务相同，由Kerberos提供安全认证。本章介绍Transwarp HBase的认证和权限管理。更多和Kerberos以及Transwarp Data Hub中的安全相关内容请参考《Transwarp Data Hub 安全手册》。

7.1. Transwarp HBase Shell交互的认证和授权

7.1.1. 认证：获取Kerberos Ticket

在集群服务器上使用Transwarp HBase Shell之前，您登陆的服务器上需要有有效的Kerberos Ticket，查看方式为在服务器命令行中执行 **klist**。如果你的机器上没有有效的Ticket，您需要进行获取。获取指令为：

```
kinit <principal>
```

您需要有能够通过Kerberos认证的用户名（principal）和密码才能获取ticket。这些信息需要向您的集群管理员索取。

7.1.2. 权限管理

Transwarp HBase通过用户的principal判断用户身份并使用Access Control Labels（ACLs）在客户端进行授权管理。Transwarp HBase的超级用户为hbase，可以向其他用户授权。经hbase授权之前，一个普通用户在Transwarp HBase没有任何权限，它只能 **list** Transwarp HBase中的表，而不能进行诸如查看表内容、建表、修改表等操作，这些操作权限需要hbase授予。

在Transwarp HBase中一个用户可以拥有 **RWCA** 权限，即：

- **R (READ)**: 读权限，用来进行 **get**, **scan**, **exists** 等读操作；
- **W (WRITE)**: 写权限，用来进行 **put**, **delete** 等写操作；
- **X (EXEC)**: 执行权限，用来执行coprocessor endpoint（高级操作）；
- **C (CREATE)**: 建表权限，用来进行 **create**, **alter** 等操作；
- **A (ADMIN)**: 管理员权限，用来进行 **enable**, **disable**, **grant**, **revoke** 等操作。

默认情况下，一个用户对自己建的表有全部 **RWCA** 权限。

对这些权限的赋予、收回和查看的语法分别如下：

- 授予权限：

```
grant '<user|@group>', '<permissions>', ['<scope_specification>']
```

- 一次性收回所有权限:

```
revoke '<user|@group>', ['<scope_specification>']
```

revoke用于批量地收回权限——将一个用户或用户组的 **RWXCA** 权限一次性收回。如果要单独收回某些权限，使用grant重新授予权限，新授的权限中不要包括要收回的权限即可（细节请参见后文）。



指定组时需要在组名前加“@”字符，如 **@groupx**。

- 查看（指定表的）权限:

```
user_permission ['<table>']
```

7.1.3. Transwarp HBase中权限作用的级别

Transwarp HBase提供不同粒度的访问控制，权限作用的 级别 由 **<scope_specification>** 指定。Transwarp HBase支持的权限作用级别如下所列（由高到低排列）：

1. Global: 全局
2. Namespace (NS): 命名空间
3. Table: 表
4. Column Family (CF): 列族
5. Column Qualifier (CQ): 列

权限从高到低继承，比如如果一个用户对一张表有R权限，那么它对表中的所有列族和列都有R权限。拥有global级别的权限意味着对Transwarp HBase中所有的命名空间、表、列族、列都有该权限。

<scope_specification> 需要按照级别顺序 从左向右 指定。

```
@<namespace> <table> <column family> <column qualifier>
```

7.2. Transwarp HBase中的“角色”

Transwarp HBase中没有数据库中常见的“角色”概念，而是通过对用户组权限的管理来实现批量授权。Transwarp HBase没有自己管理的用户组信息，而是将用户的身份映射到 登陆Transwarp HBase命令行的服务器 所在的操作系统上的用户并通过服务器操作系统上的用户组信息判断用户所在的组。映射方式如下：alice/instance@realm和alice@realm都会被映射到操作系统上的alice用户。此时，如果alice在操作系统中没有对应用户，那么alice对Transwarp HBase来说则没有组信息。从 TDH4.5开始，Guardian会将集群(KRB5LDAP)类型的用户和用户组以及它们之间的关系映射到集群中每台服务器的操作系统。对于任何通过Transwarp Manager添加的集群(KRB5LDAP)类型用户，Transwarp HBase读取的操作系统用户组配置将和Guardian中的用户组配置一致，也就是说Transwarp HBase可以使用Guardian的组信息做认证。通过Transwarp Manager添加集群用户的细节请参考《Transwarp Data Hub运维手册》或者《Transwarp Data Hub安全手册》。

7.3. hbase:acl表

Transwarp HBase将权限控制信息存放在hbase:acl表中。表中的记录是用户对namespace、表、column family, column qualifier的权限。该表的结构如下：

表 5. hbase:acl 结构

Row	CF	CQ	Value
table/@namespace	1	user/@group	permissions
table	1	user/@group, cf	permissions
table	1	user/@group, cf, cq	permissions

其中：

- Row: Transwarp HBase中的表名或namespace名
- Column Family (CF): 只有一个列族，名为“1”
- Column Qualifier (CQ): 如果row是表名，那么CQ由用户名/用户组名, row对应的表的column family(cf), column qualifier(cq)组成；如果row是namespace名，那么CQ是用户名。
- Value: 权限

我们可以用scan指令查看一张实际的hbase:acl中的内容（注意，只有有全局权限的用户才可以看到hbase:acl中的内容）：

```
hbase(main):017:0> scan 'hbase:acl'
ROW                                              COLUMN+CELL
@ns1
  bi
  bi1
  hbase:acl
    hbase:acl
      ns1:tb1
      ns1:tb1
      ss
      stuff_infor
      tb
      yy
9 row(s) in 0.1090 seconds
                                         COLUMN+CELL
                                         column=l:alice, timestamp=1449080847180, value=R
                                         column=l:alice,ct,cp, timestamp=1448999267124, value=R
                                         column=l:usera, timestamp=1446067150397, value=RWXCA
                                         column=l:hive, timestamp=1445441376184, value=RWC
                                         column=l:usera, timestamp=1446567820203, value=RWC
                                         column=l:alice, timestamp=1449079911226, value=R
                                         column=l:hbase, timestamp=1448998136037, value=RWXCA
                                         column=l:hbase, timestamp=1445442094860, value=RWXCA
                                         column=l:hive, timestamp=1445357734352, value=RWXCA
                                         column=l:usera, timestamp=1445524596994, value=RWXCA
                                         column=l:usera, timestamp=1446054553613, value=RWXCA
```

注意表中ROW为hbase:acl的记录（如上图中红框里的记录），拥有对这张表的某个权限意味这个权限的级别为global。

7.3.1. 通过Transwarp HBase Shell授予权限

下面我们介绍如何在Transwarp HBase中向用户和用户组授权。



- 指定用户组时需要在组名前加“@”字符来标识它是组名而不是用户名，如`@groupx`；
- 单独指定namespace时需要在namespace名前加“@”来标识它是namespace名而不是表名，如 `@ns1`；
- 指定namespace下的某张表用namespace:table，如 `ns1:tb1`。

7.3.1.1. Global (全局) 权限的授予



作为管理员，您可以通过查看hbase:acl来查看一个用户的全局权限。hbase:acl表中的全局权限的row就是hbase:acl，意思为如果对hbase:acl有某个权限，则对全局都有该权限。

语法

```
grant <user|@group> <permissions>
```

例 1. 授予用户alice全局R权限

```
grant 'alice', 'R'
```

例 2. 授予用户alice全局RW权限

```
grant 'alice', 'RW'
```

例 3. 授予用户组groupx全局W权限

```
grant '@groupx', 'W'
```

7.3.1.2. Namespace (命名空间) 权限的授予

语法

```
grant '<user|@group>' '<permissions>' '<@namespace>'
```

例 4. 授予用户alice对命名空间ns1的R权限

```
grant 'alice', 'R', '@ns1'
```

例 5. 授予用户组groupx对命名空间ns1的C权限

```
grant '@groupx', 'C', '@ns1'
```

7.3.1.3. Table (表) 权限的授予

语法

```
grant '<user|@group>' '<permissions>' '<table>'
```

例 6. 授予用户 alice 对表 bi 的 R 权限

```
grant 'alice', 'R', 'bi'
```

例 7. 授予用户 alice 对命名空间 ns1 中的表 tb1 的 WC 权限

```
grant 'alice', 'WC', 'ns1:tb1'
```

例 8. 授予用户组 groupx 对表 bi 的 R 权限

```
grant '@groupx', 'R', 'bi'
```

7.3.1.4. 列族权限的授予

语法

```
grant '<user|@group>' '<permissions>' '<table>', '<column_family>'
```

例 9. 授予用户 alice 对表 t4 中列族 f1 的 R 权限

```
grant 'alice', 'R', 't4', 'f1'
```

7.3.1.5. 列权限的授予

语法

```
grant '<user|@group>' '<permissions>' '<table>', '<column_family>',  
'<columnQualifier>'
```

例 10. 授予用户alice对表t4中f2:q1列的R权限

```
grant 'alice', 'R', 't4', 'f2', 'q1'
```

7.3.2. 通过Transwarp HBase Shell收回权限

权限的收回有两种方式：

- 收回 **RWXCA** 中的指定权限：使用 **grant** 命令。
- 一次性收回所有 **RWXCA** 权限：使用 **revoke** 命令。

7.3.2.1. 收回指定权限

当您需要收回用户或用户组的指定权限，您只需要使用 **grant** 命令重新授权，在授权语句中不包含您需要收回的权限即可。

例 11. 从用户alice处收回表t4的 R 权限

我们先向用户alice授予表t4的所有 **RWXCA** 权限：

```
grant 'alice', 'RWXA', 't4'
```

现在执行 **user_permission 't4'** 命令，描述alice对表t4权限的输出为：

```
alice          t4,, : [Permission: actions=READ, WRITE, EXEC, CREATE, ADMIN]
```

现在要从alice处收回t4上的 **R** 权限，我们需要执行：

```
grant 'alice', 'WXCA', 't4'
```

再一次执行 **user_permission 't4'** 命令，描述alice对表t4权限的输出会变为：

```
alice          t4,, : [Permission: actions=WRITE, EXEC, CREATE, ADMIN]
```

7.3.2.2. 一次性收回权限

使用 **revoke** 权限则可以一次性收回所有 **RWXCA** 权限。

语法

```
revoke '<user|@group>', ['<scope_specification>']
```

执行 `revoke` 后，指定用户或用户组将在 `<scope_specification>` 指定的级别上没有任何权限。

例 12. 收回用户alice对表bi的所有权限

```
revoke 'alice','t4'
```

7.3.3. 通过Transwarp HBase Shell查看权限

查看权限的方式有两种：

- 查看`hbase:acl`表中的信息
- 使用 `user_permission` 命令（只能查看表级和表级以下权限）

两种方式都需要有全局 `R` 权限才能操作。

7.3.3.1. 查看`hbase:acl`表

我们来查看一张样例`hbase:acl`表，并解释其中的一些记录：

ROW	COLUMN+CELL	
@ns1	column=l:alice, timestamp=1457636462495, value=R	①
bi	column=l:hbase, timestamp=1453427488388, value=RWXCA	②
bl	column=l:hive, timestamp=1454513614929, value=RWXCA	
hbase:acl	column=l:hive, timestamp=1457629405417, value=R	③
t1	column=l:hbase, timestamp=1457528928129, value=RWXCA	

① alice用户对命名空间ns1有 `R` 权限

② hbase用户对表bi有 `RWXCA` 权限

③ hive用户有全局 `R` 权限

7.3.3.2. `user_permission` 命令

语法

```
user_permission ['<table>']
```

`user_permission` 查看default命名空间中所有表的表级和表级以下权限。加上 `<table>` 可以只查看指定表的权限。

例 13. 查看命名空间ns1中的表tb1上的权限

```
user_permission 'ns1:tb1'
```

7.4. API交互的认证和权限管理

之前的章节介绍了有关权限授予、收回以及查看操作的相应命令，并简单介绍了Kerberos认证。如果需使用API来进行权限的管理，请参考本节给出的示例。

环境准备

1. 配置Kerberos客户端（下面两个选项二选一）
 - 配置默认路径: /etc/krb5.conf (Linux) 或者 C://Windows/krb5.ini (Windows)
 - 添加Java程序启动参数 **-Djava.security.krb5.conf=<your_krb5_conf_path>** 来指定Kerberos客户端配置文件路径
2. 客户端时间与集群时间同步，注意：确保客户端时间和集群时间的时间差在5分钟之内。

7.4.1. 通过认证访问Transwarp HBase

API访问时使用 **UserGroupInformation.loginUserFromKeytab** 登录。取决于程序的classpath中是否加入了配置文件 (hbase-site.xml, core-site.xml)，通过Kerberos认证访问Transwarp HBase的方式略有不同：

通过认证访问Transwarp HBase示例:classpath中有配置文件目录

```
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.HTableDescriptor;
import org.apache.hadoop.hbase.client.HBaseAdmin;
import org.apache.hadoop.security.UserGroupInformation;

public class HBaseSecureTest {
    public static void main(String[] args) throws IOException {
        UserGroupInformation.loginUserFromKeytab(
            "hbase/tw-node2125", ①
            "/etc/hyperbase1/hbase.keytab"); ②
        Configuration conf = HBaseConfiguration.create();
        HBaseAdmin hBaseAdmin = new HBaseAdmin(conf);
        hBaseAdmin.createTable(new HTableDescriptor("t1"));
    }
}
```

① **/etc/hyperbase1/hbase.keytab** 处为客户端的keytab文件所在地址，而非服务端的，所以需要根据具体的客户端地址加以修改。

- ② “`hbase/tw-node2125`” 为访问的Transwarp HBase服务的principal，`tw-node2125` 是Transwarp HBase 服务所在的节点之一。这里可以使用集群中任意一个节点上的Transwarp HBase principal，也可以直接将 `/tw-node2125` 部分省略。

[通过认证访问Transwarp HBase示例：classpath中没有配置文件目录](#)

```

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.HTableDescriptor;
import org.apache.hadoop.hbase.client.HBaseAdmin;
import org.apache.hadoop.security.UserGroupInformation;

public class HBaseSecureTest {
    public static void main(String[] args) throws IOException {
        Configuration HBASE_CONFIG = new Configuration();
        HBASE_CONFIG.set("hbase.zookeeper.quorum",
        "172.16.2.125,172.16.2.126,172.16.2.127");
        ①
        HBASE_CONFIG.set("hbase.master.kerberos.principal",
        "hbase/_HOST@TDH");
        HBASE_CONFIG.set("hbase.regionserver.kerberos.principal",
        "hbase/_HOST@TDH");
        HBASE_CONFIG.set("hbase.security.authentication", "kerberos");
        HBASE_CONFIG.set("hadoop.security.authentication", "kerberos");

        HBASE_CONFIG.set("zookeeper.znode.parent", "/hyperbase1");

        Configuration conf = HBaseConfiguration.create(HBASE_CONFIG);

        UserGroupInformation.setConfiguration(conf);
        UserGroupInformation.loginUserFromKeytab("hbase/tw-node2125",
        "/etc/hyperbase1/hbase.keytab");      ②
        HBaseAdmin hBaseAdmin = new HBaseAdmin(conf);
        hBaseAdmin.createTable(new HTableDescriptor("t1"));
    }
}

```

① 以下四条命令中的参数都是固定写法。

② 和[通过认证访问Transwarp HBase示例：classpath中有配置文件目录](#)中使用方法一致。

API交互中可以使用不同的 UserGroupInformation 对象登录不同的用户，并让这些用户执行不同的代码段：

```
UserGroupInformation ugi1 =
UserGroupInformation.loginUserFromKeytabAndReturnUGI(principal1, keytab1);
UserGroupInformation ugi2 =
UserGroupInformation.loginUserFromKeytabAndReturnUGI(principal2, keytab2);

ugi1.doAs(new PrivilegeExceptionAction<T> {
    override T run() {
        action1;
        return T;
    }
});

ugi2.doAs(new PrivilegeExceptionAction<T> {
    override T run() {
        action2;
        return T;
    }
});
```

7.4.2. 安全模式下的基本操作

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.*;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.util.Bytes;
import org.apache.hadoop.security.UserGroupInformation;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

/**
 * Created by gordon on 2015/6/16.
 */
public class HyperbaseSecureTest {

    private static Configuration conf = null;

    static {
        Configuration HBASE_CONFIG = new Configuration();
        HBASE_CONFIG.set("hbase.zookeeper.quorum", "transwarp-
node5,transwarp-node6,transwarp-node7");
        HBASE_CONFIG.set("hbase.master.kerberos.principal",
"hbase/_HOST@TDH");
        HBASE_CONFIG.set("hbase.regionserver.kerberos.principal",
"hbase/_HOST@TDH");
        HBASE_CONFIG.set("hbase.security.authentication", "kerberos");
        HBASE_CONFIG.set("zookeeper.znode.parent", "/hyperbase1");
        HBASE_CONFIG.set("hadoop.security.authentication", "kerberos");

        conf = HBaseConfiguration.create(HBASE_CONFIG);
    }

    /**

```

```

-----+
 * 创建一张表
 */
public static void creatTable(String tableName, String[] familys)
throws Exception {
    HBaseAdmin admin = new HBaseAdmin(conf);
    if (admin.tableExists(tableName)) {
        System.out.println("table already exists!");
    } else {
        HTableDescriptor tableDesc = new HTableDescriptor(tableName);
        for(int i=0; i<familys.length; i++){
            tableDesc.addFamily(new HColumnDescriptor(familys[i]));
        }
        admin.createTable(tableDesc);
        System.out.println("create table " + tableName + " ok.");
    }
}

/**
 * 删除表
 */
public static void deleteTable(String tableName) throws Exception {
    try {
        HBaseAdmin admin = new HBaseAdmin(conf);
        admin.disableTable(tableName);
        admin.deleteTable(tableName);
        System.out.println("delete table " + tableName + " ok.");
    } catch (MasterNotRunningException e) {
        e.printStackTrace();
    } catch (ZooKeeperConnectionException e) {
        e.printStackTrace();
    }
}

/**
 * 插入一行记录
 */
public static void addRecord (String tableName, String rowKey, String
family, String qualifier, String value)
throws Exception{
    try {
        HTable table = new HTable(conf, tableName);
        Put put = new Put(Bytes.toBytes(rowKey));
        put.add(Bytes.toBytes(family), Bytes.toBytes
(qualifier), Bytes.toBytes(value));
        table.put(put);
        System.out.println("insert record " + rowKey + " to table " +
tableName + " ok.");
    } catch (IOException e) {
        e.printStackTrace();
    }
}

/**
 * 删除一行记录
 */
public static void delRecord (String tableName, String rowKey) throws
IOException{
    HTable table = new HTable(conf, tableName);
    List list = new ArrayList();

```

```

-----Delete del = new Delete(rowKey.getBytes());
list.add(del);
table.delete(list);
System.out.println("del recorded " + rowKey + " ok.");
}

/**
 * 查找一行记录
 */
public static void getOneRecord (String tableName, String rowKey)
throws IOException{
    HTable table = new HTable(conf, tableName);
    Get get = new Get(rowKey.getBytes());
    Result rs = table.get(get);
    for(KeyValue kv : rs.raw()){
        System.out.print(new String(kv.getRow()) + " ");
        System.out.print(new String(kv.getFamily()) + ":" );
        System.out.print(new String(kv.getQualifier()) + " ");
        System.out.print(kv.getTimestamp() + " ");
        System.out.println(new String(kv.getValue()));
    }
}

/**
 * 显示所有数据
 */
public static void getAllRecord (String tableName) {
    try{
        HTable table = new HTable(conf, tableName);
        Scan s = new Scan();
        ResultScanner ss = table.getScanner(s);
        for(Result r:ss){
            for(KeyValue kv : r.raw()){
                System.out.print(new String(kv.getRow()) + " ");
                System.out.print(new String(kv.getFamily()) + ":" );
                System.out.print(new String(kv.getQualifier()) + " ");
                System.out.print(kv.getTimestamp() + " ");
                System.out.println(new String(kv.getValue()));
            }
        }
    } catch (IOException e){
        e.printStackTrace();
    }
}

public static void main(String[] args) throws IOException {
    try {
        UserGroupInformation.setConfiguration(conf);
        UserGroupInformation.loginUserFromKeytab("hbase/transwarp-
node5", "/etc/hyperbase1/hbase.keytab");
//        HBaseAdmin hBaseAdmin = new HBaseAdmin(conf);
//        hBaseAdmin.createTable(new HTableDescriptor("gordon2"));

        String tablename = "scores";
        String[] familys = {"grade", "course"};
        HyperbaseSecureTest.createTable(tablename, familys);

        //add record zkb
        HyperbaseSecureTest.addRecord(tablename, "zkb", "grade", "", "5");
    }
}
-----
```

```
        HyperbaseSecureTest.addRecord(tablename, "zkb", "course", "",  
        "90");  
        HyperbaseSecureTest.addRecord(tablename, "zkb", "course", "math"  
        , "97");  
        HyperbaseSecureTest.addRecord(tablename, "zkb", "course", "art"  
        , "87"));  
        //add record baoniu  
        HyperbaseSecureTest.addRecord(tablename, "baoniu", "grade", ""  
        , "4");  
        HyperbaseSecureTest.addRecord(tablename, "baoniu", "course"  
        , "math", "89");  
  
        System.out.println("=====get one record=====");  
        HyperbaseSecureTest.getOneRecord(tablename, "zkb");  
  
        System.out.println("=====show all record=====");  
        HyperbaseSecureTest.getAllRecord(tablename);  
  
        System.out.println("=====del one record=====");  
        HyperbaseSecureTest.delRecord(tablename, "baoniu");  
        HyperbaseSecureTest.getAllRecord(tablename);  
  
        System.out.println("=====show all record=====");  
        HyperbaseSecureTest.getAllRecord(tablename);  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

7.4.3. 权限管理API示例

Transwarp HBase提供了 `AccessControlClient` 这个类用来操作Transwarp HBase的权限管理。

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.protobuf.generated.AccessControlProtos;
import org.apache.hadoop.hbase.security.access.AccessControlClient;
import org.apache.hadoop.hbase.security.access.UserPermission;
import org.apache.hadoop.security.UserGroupInformation;

import java.security.PrivilegedAction;
import java.util.List;

public class HyperbaseAuthorization {
    public static void main(String[] args) throws Exception {
        String zkquorum = "baogang1";
        String znode = "/hyperbase1";
        final String targetTableName = "zhaoliu:zhaoliu_table";

        Configuration config = new Configuration();
        config.set("hbase.zookeeper.quorum", zkquorum);
        config.set("zookeeper.znode.parent", znode);
```

```

config.set("hbase.master.kerberos.principal", "hbase/_HOST@TDH");
config.set("hbase.regionserver.kerberos.principal", "hbase/_HOST@TDH");
config.set("hbase.security.authentication", "kerberos");
config.set("hadoop.security.authentication", "kerberos");

final Configuration conf = HBaseConfiguration.create(config);

UserGroupInformation.setConfiguration(conf);
// hbase user has global permissions
UserGroupInformation ugi =
UserGroupInformation.loginUserFromKeytabAndReturnUGI("hbase",
"/home/zhaoliu/data/baogang/hbase-baogang.keytab");

ugi.doAs(new PrivilegedAction<Void>() {
    @Override
    public Void run() {
        try {
            if (AccessControlClient.isAccessControllerRunning(conf)) {
                List<UserPermission> permissions =
AccessControlClient.getUserPermissions(conf, targetTableName);
                AccessControlProtos.GrantResponse grantResponse =
AccessControlClient.grant(conf, TableName.valueOf(targetTableName), "zhu",
"f".getBytes(), null,
AccessControlProtos.Permission.Action.READ,
AccessControlProtos.Permission.Action.WRITE);
                permissions = AccessControlClient.getUserPermissions(conf,
targetTableName);
                AccessControlProtos.RevokeResponse revokeResponse =
AccessControlClient.revoke(conf, "zhu", TableName.
valueOf(targetTableName), "f".getBytes(), null,
AccessControlProtos.Permission.Action.READ,
AccessControlProtos.Permission.Action.WRITE);
                permissions = AccessControlClient.getUserPermissions(conf,
targetTableName);

// Global permissions
String global = "hbase:acl";
grantResponse = AccessControlClient.grant(conf, TableName.
valueOf(global), "zhu", null, null,
AccessControlProtos.Permission.Action.READ,
AccessControlProtos.Permission.Action.WRITE);
permissions = AccessControlClient.getUserPermissions(conf, global);
revokeResponse = AccessControlClient.revoke(conf, "zhu",
TableName.valueOf(global), null, null,
AccessControlProtos.Permission.Action.READ,
AccessControlProtos.Permission.Action.WRITE);
permissions = AccessControlClient.getUserPermissions(conf, global);

} else {
            System.out.println("Hbase access controller is not running.");
}
        } catch (Throwable e) {
            e.printStackTrace();
}
        return null;
}

```

```
});  
}  
}
```

8. Transwarp HBase Shell命令

登陆集群中的服务器，执行 `hbase shell` 操作可以进入命令行和Transwarp HBase进行简单交互。本章介绍Transwarp HBase Shell指令，您需要注意以下几点：

- 所有名字如表名和列名都必须使用单引号进行引用：如 `'table1', 'key1'`。
- 创建和修改表的配置时使用的是Ruby Hashes，如 `{'key1' 'value1', 'key2' 'value2', ...}`。它需用 `{` 和 `}` 表明整个对象的开始和结尾，每对key, value之间通过逗号分隔，key和value之间通过 `=>` 分隔。
- 如果想输入二进制的数值，需用 双引号 进行引用，并且使用16进制表示法，如：

```
get 't1', "key\x03\x3f\xcd"
get 't1', "key\003\023\011"
put 't1', "test\xef\xff", 'f1:', "\x01\x33\x40"
```

Transwarp HBase Shell命令大致可以分成以下七类：

- 表管理命令
- 数据操作命令
- namespace相关命令
- 通用命令
- 索引命令
- Hyperdrive表相关命令
- 授权命令

本章会详细解释前六类命令，授权指令将在[\[hyperbase-security-chapter\]](#)中介绍。如需要了解某个命令的细节可以输入 `help '<command>'`，如 `help 'create'`。

8.1. Transwarp HBase Shell表管理命令



可以在Transwarp HBase Shell中执行 `help 'ddl'` 来查看帮助。

8.1.1. list

语法

```
list [<regex>]
```

列出Transwarp HBase中所有的表，可以选择列出和正则表达式 `<regex>` 匹配的表。

例 14. list

```
list
list 'abc.*'
```

8.1.2. create

语法

```
create '<table>', {NAME => '<column_family>' [, ...]} [, {...}, ...]
```

建表时至少要指定一个列族，列族名通过 `NAME => '<column_family>'` 指定，在建表同时还可以设置列族的其他元数据。如果建表时要创建多个列族，不同列族的元信息要用 `{}` 隔开。

例 15. 建表

```
create 't4', {NAME => 'f1'}, {NAME => 'f2'}
```

该建表语句可以简写为：

```
create 't4', 'f1', 'f2'
```

例 16. 建表同时划分region

```
create 't1', 'f1', 'q1', {SPLITS => ['3', '5', '7']}
```

这里 `['3', '5', '7']` 为表 `t1` 的split key。

8.1.3. describe

语法

```
describe '<table>'
```

查看 `<table>` 的元数据。

例 17. `describe`

```
describe 'bi'
```

8.1.4. `exists`

语法

```
exists '<table>'
```

查看 `<table>` 是否存在。

例 18. `exists`

```
exists 'bi'
```

8.1.5. `show_filters`

语法

```
show_filters
```

列出Transwarp HBase中所有的filter。

8.1.6. `enable`

语法

```
enable '<table>'
```

将指定表上线。

8.1.7. `disable`

语法

```
disable '<table>'
```

将指定表下线。

8.1.8. enable_all

语法

```
enable_all
```

将所有表上线。

8.1.9. is_enabled

语法

```
is_enabled '<table>'
```

查看 `<table>` 是否上线。

8.1.10. disable_all

语法

```
disable_all
```

将所有表下线。

8.1.11. is_disabled

语法

```
is_disabled '<table>'
```

查看 `<table>` 是否下线。

8.1.12. delete_table

语法

```
delete_table '<my_table>'
```

`delete_table` 删除指定表，不需要先将表下线就可以删除表。我们推荐使用 `delete_table` 而不是 `disable` 和 `drop` 来删除表。

例 19. delete_table

```
delete_table 'htable'
```

8.1.13. drop_all

语法

```
drop_all '<regex>'
```

将所有表名和指定正则表达式 **<regex>** 匹配的表删除。

例 20. drop_all

```
drop_all 't.*'
```

8.1.14. alter

语法

```
alter '<table>', {<PROPERTY_1> => <value_1>, <PROPERTY_2> => <value_2>,
..., {...}, ...
```

alter 可以用于添加和修改表或表中列族的元数据，可以同时修改多个列簇。使用 **<PROPERTY> => <value>** 的方式设置表或列族的属性。如果只修改一组元数据，可以不加 **{}**；如果同时修改多组元数据（例如修改多个列族元数据），每组属性需要放在不同的 **{}** 中。

例 21. 用 alter 修改列族属性

将表t1中的列簇f1改为常驻内存，并将列簇f2的版本改为5：

```
alter 't1', {NAME => 'f1', IN_MEMORY => true}, {NAME => 'f2', VERSION
=> 5}
```

例 22. 用 `alter` 删除列族

删除表中某个列族:

```
alter 't1', NAME => 'f1', METHOD => 'delete'
```

上面的指令可以简写为:

```
alter 't1', 'delete' => 'f1'
```

8.1.15. `alter_async` 和 `alter_status`

`alter_async` 和 `alter` 语法相同，但是语义和 `alter` 略有不同。`alter_async` 指令可以立即返回，而 `alter` 需要等到所有的region都更新完成后才会返回。而要查看更新期间所有regions的更新进度，可以使用 `alter_status` 命令。

8.2. Transwarp HBase Shell数据操作命令



可以在Transwarp HBase Shell中执行 `help 'dml'` 来获取帮助。

8.2.1. `count`

语法

```
count '<table>' [, CACHE => <n>, INTERVAL => <m>]
```

返回指定表中的行数。默认情况下该命令每次只获取一行，因此对于一张大表来说该命令会执行得很慢。可以通过设置 `CACHE` 参数来增加每次获取的行数，从而加速该命令的执行。还可以指定查询到多少行显示一次 `count` 结果，默认值是1000行，可以通过 `INTERVAL` 参数进行修改。

例 23. `count`

```
count 't2'  
count 't2', CACHE => 1000, INTERVAL => 10000
```

8.2.2. `put`

语法

```
put '<table>', '<row_key>', '<column_family:column_qualifier>',
'<cell_value>', [<timestamp>]
```

将值 `<cell_value>` 填入指定的表 (`<table>`)、行 (`<row_key>`) 和列 (`<column_family:column_qualifier>`) 对应的单元格中。时间戳 `<timestamp>` 为可选项。

例 24. put

```
put 't4', '1', 'f1:q1', 'a'
```

8.2.3. get

语法

```
get '<table>', '<row_key>' [{<PROPERTY> => <value>, ...}]
```

获取指定的表和行中的数据。可以通过 `<PROPERTY>` `<value>` 指定某些属性来过滤获取的结果。例如 `COLUMN` `'column_family:column_qualifier'` 指定只获取某一列中的数据。可指定的属性有: `COLUMN`, `TIMTranswarp ESTAMP`, `TIMERANGE`, `VERSIONS` 和 `FILTER`。

例 25. get

```
get 't4', '1'
get 't4', '1', {COLUMN => 'f1:q1'}
```

8.2.4. scan

语法

```
scan '<table>', [{<PROPERTY> => <value>, ...}]
```

扫描指定的表，批量获取表中数据。可以通过 `<PROPERTY>` `<value>` 指定某些属性来过滤获取的结果。可指定的属性有: `TIMERANGE`, `FILTER`, `LIMIT`, `STARTROW`, `STOPROW`, `TIMTranswarp ESTAMP`, `MAXLENGTH`, `COLUMNS` 和 `CACHE`。

例 26. scan

```
scan 't4'
scan 't4', {COLUMNS => 'f1:q1'}
```

8.2.5. delete

语法

```
delete '<table>', '<row_key>', '<column_family:column_qualifier>' [, <timestamp>]
```

删除指定的表（`<table>`）、行（`<row_key>`）和列（`<column_family:column_qualifier>`）对应的单元格。可以加上 `<timestamp>` 选项指定删除某个时间戳对应的数据。

例 27. delete

```
delete 't4', '1', 'f1:q1'
```

8.2.6. deleteall

语法

```
deleteall '<table>', '<row_key>' [, '<column_family:column_qualifier>', <timestamp>]
```

删除指定表、指定行中全部的数据。可以加上 '`<column_family:column_qualifier>`' 和 `<timestamp>` 选项指定删除某个列或时间戳对应的数据。

例 28. deleteall

```
deleteall 't4', '2'
```

8.2.7. truncate 和 truncate_preserve

语法

```
truncate '<table>'  
truncnate_preserve '<table>'
```

`truncate` 类似 `delete`, 但该命令会立即删除表中所有的数据以及region的划分. 它的内部实现是将指定的表下线, 删除, 并重建. 如果只想立即删除表中所有的数据而不想丢掉原来的region划分, 需要使用 `truncate_preserve`.

8.3. Transwarp HBase Shell Namespace相关命令



可以在Transwarp HBase Shell中执行 `help 'namespace'` 来获取帮助。

8.3.1. alter_namespace

语法

```
alter_namespace '<namespace>', {METHOD => 'set|unset', <PROPERTY> => <value>, ...}
```

修改namespace属性。METHOD 选择 `set` 为添加或设置属性；选择 `unset` 为删除属性。

8.3.2. create_namespace

语法

```
create_namespace '<namespace>' [, {<PROPERTY_1> => <value_1>, ...}]
```

创建一个namespace，并可以通过 `<PROPERTY_1> => '<value>'` 额外指定namespace的属性。

8.3.3. describe_namespace

语法

```
describe_namespace '<namespace>'
```

查看 `<namespace>` 的元数据。

8.3.4. drop_namespace

语法

```
drop_namespace '<namespace>'
```

删除指定的namespace。要删除的namespace必须是一个空的namespace，不能存在表。

8.3.5. list_namespace

语法

```
list_namespace [<regex>]
```

列出Transwarp HBase中所有的namespace，可以加上正则表达式 <regex> 来对结果进行匹配。

例 29. `list_namespace`

```
list_namespace
list_namespace 'abc.*'
```

8.3.6. `list_namespace_tables`

语法

```
list_namespace_tables '<namespace>'
```

列出指定的namespace下所有的表。

8.4. Transwarp HBase Shell索引命令

Transwarp HBase表支持全局索引、LOB索引和全文索引。在Transwarp HBase Shell中的索引命令包括：

- 创建、生成和删除全局索引：`add_index`,
`rebuild_global_index/rebuild_global_index_with_range`, `delete_global_index`
- 生成全文索引：`rebuild_fulltext_index/rebuild_fulltext_index_with_range`

本节我们将介绍如何使用Transwarp HBase Shell指令为Transwarp HBase表创建、生成和删除全局索引。LOB索引的操作需要使用API（请参考[Object Store使用方法](#)）或者JSON配置（请参考[lob模块](#)）。全文索引的创建需要使用JSON配置（请参考[fulltextindex模块](#)）。

8.4.1. 全局索引指令

8.4.1.1. 创建全局索引

语法

```
add_index '<table>', '<global_index_name>', '<index_definition>'
```

该语句为指定表 `<table>` 添加全局索引，生成一张索引表，索引表的表名将是 `table_global_index_name`（注意区分索引名和索引表名）。`<index_definition>` 为索引的定义，用于指示Transwarp HBase如何建索引。

索引的定义形式如下：

**COMBINE_INDEX|INDEXED= <cf>:<cq>:<n>[|<cf>:<cq>:<n>| ...] | rowKey :
rowKey:<m>, [UPDATE=true]**

说明

- **<cf>:<cq>:<n>** 指定生成索引所用的列以及索引长度: **<cf>** 是该列所在的列族, **<cq>** 是该列的列限定符, **<n>** 是索引字段在索引词条中的长度。例如 **f1:q1:n1** 为表中每一行记录用 **f1:q1** 列的数据生成的一段长度为n1字段放在索引词条中。如果指定的 **f1:q1** 列的值长度不足n1, Transwarp HBase将用0表示空格将长度补足。如果长度超过n1, 超过的部分将在索引词条的末尾添上。
- 如果需要使用多列生成索引, 可以在定义中添加多组 **<cf>:<cq>:<n>**, 组之间用 **|** 隔开。
- 每个生成的索引词条中都会包含一段原表Row Key生成的长度为m的字段, 由 **rowKey:rowKey:<m>** 指定。
- **[UPDATE=true]** 为可选项, 设为true代表索引词条会随着对应原表数据的更新自动更新, 设为false则不会。

例 30. 用单列创建全局索引

对bi表用 **ps:nm** 列创建名为 **psnmindex** 的索引。索引词条中 **ps:nm** 对应字段长度为8, **rowKey** 对应字段长度为10。创建的索引表表名为 **bi_psnmindex**。

```
add_index 'bi', 'psnmindex'  
, 'COMBINE_INDEX|INDEXED=ps:nm:8|rowKey:rowKey:10'
```

例 31. 用多列创建全局索引

对bi表用 **ps:nm** 和 **ps:pw** 两列创建名为 **psnmpwindex** 的索引。索引词条中 **ps:nm** 对应字段长度为8, **ps:pw** 对应字段长度为9, **rowKey** 对应字段长度为10。创建的索引表表名为 **bi_psnmpwindex**。

```
add_index 'bi', 'psnmpwindex'  
, 'COMBINE_INDEX|INDEXED=ps:nm:8|ps:pw:9|rowKey:rowKey:10, UPDATE=true'
```

8.4.1.2. 生成全局索引

[创建全局索引](#) 创建一张空的索引表, 没有索引词条, 索引词条在两种情况下生成:

1. 当原表中有新的数据插入时, Transwarp HBase会 **自动** 为新增数据生成索引词条, 写入索引表中。
2. 用户执行 **rebuild_global_index/rebuild_global_index_with_range** 让Transwarp HBase为原表中数据生成索引词条, 写入索引表中。



在使用 **rebuild_global_index/rebuild_global_index_with_range** 前请先预分好 索引表 的region, 以保证索引词条插入索引表的性能。

Transwarp HBase中可以为全表生成全局索引, 也可以按表的Row Key分段生成索引。

为全表生成全局索引

为全表生成全局索引的指令为 `rebuild_global_index`，使用时要提供表名和索引名。

语法

```
rebuild_global_index '<table>', '<index_name>'
```



在TDH4.5以前的版本中，生成全局索引的指令为 `rebuild_index`，TDH 4.5兼容该指令，但是我们建议新的应用都采用 `rebuild_global_index`。

例 32. 为全表生成全局索引

生成用单列创建全局索引中创建的全局索引。

```
rebuild_global_index 'bi', 'psnmindex'
```

分段生成全局索引

在Transwarp HBase表非常大时，可以分段生成全局索引，指令为 `rebuild_global_index_with_range`，为指定的Row Key区间中的记录生成索引。

语法

```
rebuild_global_index_with_range '<table>', '<index_name>', '<start_key>',  
'<end_key>', '<encoder>'
```

`start_key` 为区间的下限，`end_key` 为区间的上限，该指令为`[start_key, end_key)` 左闭右开 区间生成索引；Encoder为编码方式，表示'`<start_key>`' 和 '`<end_key>`' 的输入类型，目前支持' string'、'hex' 和' binary' 三种，默认使用' string' 来编码。

十六进制和二进制编解码分别由如下package实现：



```
org.apache.hadoop.hbase.util.Bytes.fromHex(string)  
org.apache.hadoop.hbase.util.Bytes.toHex(byte[])  
org.apache.hadoop.hbase.util.Bytes.toBytesBinary(string)  
org.apache.hadoop.hbase.util.Bytes.toStringBinary(byte[])
```

例 33. 默认使用string来编码start_key和end_key示例

```
rebuild_global_index_with_range 't1', 'index_name', 'rowkey1',
'rowkey2'

rebuild_global_index_with_range 't1', 'index_name', 'rowkey1',
'rowkey2', 'string'
```

例 34. 使用二进制编码start_key和end_key示例

可从60010界面中的user tables列表里，选取region的start_key和end_key，直接拷贝下来即可。

```
rebuild_global_index_with_range 't1', 'index_name',
'\x00\x00\x00\x00\x1D', '\x00\xC1\xF2\xF1\x8C', 'binary'

rebuild_global_index_with_range 't1', 'index_name',
'\x00.07593071128814799', '\x02.4329335438689883', 'binary'
```

例 35. 使用十六进制编码start_key和end_key示例

如start_key='abc',end_key='xyz',会按照十六进制对应编码为 abc → 616263, xyz → 78797a

```
rebuild_global_index_with_range 't1', 'index_name', '616263',
'78797a', 'hex'
```



- hex方式：**对于byte[]数组，hex encoder会为其除去为0的前缀，如 \x00ab 会hex转码为 ab。可以看出该转码不可逆，因而此方式不适用于hyperdrive表。hex方式一般可用在非hyperdrive的表上，且表中元素含有中文的或特殊字符，并需保证byte[]前缀非0。
- binary方式：**基本对所有表都可以进行转码。可从hbase shell scan得到binary值，如 \x00abc；也可以在60010页面上拷贝得到binary值，如 \x12\x23。当rowkey有不可见字符时，推荐使用这种方式。

8.4.1.3. 查看全局索引

全局索引在一张表中，我们可以用普通查看表的命令 **scan** 来查看全局索引所在的索引表。

例 36. 查看全局索引

通过 **scan** 查看**为全表生成全局索引**中生成的全局索引。

```
scan 'bi_psnmindex'
```

8.4.1.4. 删除全局索引

虽然全局索引存在一张表中，我们不能像删除普通Transwarp HBase表一样（先 `disable` 然后 `drop`）删除它，因为这样Zookeeper还会保留索引表的信息，之后创建同名索引可能会出现问题。要彻底删除索引，必须使用删除全局索引专用的命令 `delete_global_index`。

语法

```
delete_global_index '<table>', '<index_name>'
```

例 37. 删除全局索引

删除用单列创建全局索引中建的全局索引：

```
delete_global_index 'bi', 'psnmindex'
```

8.4.2. 本地索引指令

仅Hyperdrive表支持本地索引，HBase表不支持本地索引。Hyperdrive本地索引的创建需要使用[SQL](#)。

8.4.2.1. 生成本地索引

语法：分段生成本地索引

```
rebuild_local_index_with_range '<t1>', '<index_family>', '<start_key>',  
'<end_key>', '<encoder>'
```

`start_key` 为区间的下限，`end_key` 为区间的上限，该指令为`[start_key, end_key)` 左闭右开 区间生成索引；Encoder为编码方式，表示'`<start_key>`'和'`<end_key>`'的输入类型，目前支持' string'、'hex' 和' binary' 三种，默认使用' string' 来编码。

例 38. 默认使用string来编码start_key和end_key示例

```
rebuild_local_index_with_range 't1', 'index_family', 'rowkey1',  
'rowkey2'  
rebuild_local_index_with_range 't1', 'index_family', 'rowkey1',  
'rowkey2', 'string'
```

例 39. 使用二进制编码start_key和end_key示例

可从60010界面中的user tables列表里，选取region的start_key和end_key，直接拷贝下来即可。

```
rebuild_local_index_with_range 't1', 'index_family'
, '\x00\x00\x00\x00\x1D', '\x00\xC1\xF2\xF1\x8C', 'binary'

rebuild_local_index_with_range 't1', 'index_family',
'\x000.07593071128814799', '\x002.4329335438689883', 'binary'
```

例 40. 使用十六进制编码start_key和end_key示例

如start_key='abc',end_key='xyz',会按照十六进制对应编码为 abc → 616263, xyz → 78797a

```
rebuild_local_index_with_range 't1', 'index_family', '616263',
'78797a', 'hex'
```

8.4.3. 全文索引指令

Transwarp HBase Shell中可以生成全文索引，指令分为

- 为全表生成索引: `rebuild_fulltext_index`
- 按Row Key分段生成索引: `rebuild_fulltext_index_with_range`



全文索引的创建需要通过[JSON配置](#)。或者[SQL](#)。

8.4.3.1. 生成全文索引

语法：为全表生成全文索引

```
rebuild_fulltext_index '<table>'
```

因为一张表只能有一个全文索引，生成时只需指定表名即可。

语法：分段生成全文索引

```
rebuild_fulltext_index_with_range '<table>', '<start_key>', '<end_key>'
```

`start_key` 为区间的下限，`end_key` 为区间的上限，该指令为`[start_key, end_key)` 左闭右开 区间生成索引；Encoder为编码方式，表示'`<start_key>`' 和 '`<end_key>`' 的输入类型，目前支持' string'、' hex' 和' binary' 三种，默认使用' string' 来编码。

例 41. 默认使用string来编码start_key和end_key示例

```
rebuild_fulltext_index_with_range 't1', 'rowkey1', 'rowkey2'
rebuild_fulltext_index_with_range 't1', 'rowkey1', 'rowkey2', 'string'
```

例 42. 使用二进制编码start_key和end_key示例

可从60010界面中的user tables列表里，选取region的start_key和end_key，直接拷贝下来即可。

```
rebuild_fulltext_index_with_range 't1', '\x00\x00\x00\x00\x1D',
'\x00\xC1\xF2\xF1\x8C', 'binary'
rebuild_fulltext_index_with_range 't1', '\x000.07593071128814799',
'\x002.4329335438689883', 'binary'
```

例 43. 使用十六进制编码start_key和end_key示例

如start_key='abc',end_key='xyz',会按照十六进制对应编码为 abc → 616263, xyz → 78797a

```
rebuild_fulltext_index_with_range 't1', '616263', '78797a', 'hex'
```

8.5. Transwarp HBase Shell快照相关命令

8.5.1. snapshot

语法

```
snapshot '<table_name>', '<my_snapshot>'
```

为 `<my_table>` 表生成快照，您需要指定一个快照名 `<my_snapshot>`。

8.5.2. list_snapshots

语法

```
list_snapshots
```

列出当前Transwarp HBase中所有的快照。

8.5.3. delete_snapshot

语法

```
delete_snapshot '<my_snapshot>'
```

删除 `<my_snapshot>` 快照。

8.5.4. clone_snapshot

语法

```
clone_snapshot '<my_snapshot>', '<new_table>'
```

用 `<my_snapshot>` 快照建一个新表 `<new_table>`。建成的新表 `<new_table>` 将和 `<my_snapshot>` 的原表在 生成快照时 相同。对 `<new_table>` 的改动不会影响 `<my_snapshot>` 或它的原表。

8.5.5. restore_snapshot

语法

```
disable '<my_table>'  
restore_snapshot '<my_snapshot>'
```

`restore_snapshot` 用一张表的快照将其还原成快照生成时的状态。如果必要会对表的数据和元数据都进行修改。
*执行 `restore_snapshot` 前需要先 `disable` 原表再执行。

9. Transwarp HBase Shell常规指令

9.1. status

语法

```
status [ 'options' ]
```

查看集群状态。

Options

- summary (default): 总结状态
- simple: 简单状态
- detailed: 详细状态

举例

```
status
3 servers, 0 dead, 3.6667 average load
```

9.2. version

语法

```
version
```

输出Transwarp HBase版本。

举例

```
version
0.98.6-transwarp, r, Wed Sep 23 09:55:27 EDT 2015
```

9.3. whoami

语法

```
whoami
```

输出当前Transwarp HBase用户。Transwarp HBase根据认证方式识别用户身份。如果Transwarp HBase使用简单认证模式（除服务器操作系统认证以外没有认证系统），那么当前Transwarp HBase用户身份和用户在操作系统上的身份一致。如果Transwarp HBase使用Kerberos认证，用户需要先持Kerberos principal+密码通过Kerberos认证，那么当前Transwarp HBase用户身份和她的Kerberos principal中的第一个字段一致。例如alice@TDH和alice/admin@TDH都会被Transwarp HBase识别为用户alice。

举例：简单认证模式

```
whoami
alice (auth:SIMPLE)
groups: alice
```

举例：Kerberos认证模式

```
whoami
alice@TDH (auth:KERBEROS)
groups: alice
```

9.4. Hyperdrive相关命令

本节介绍的命令用于对Hyperdrive表进行操作。

9.4.1. `create_table`

语法

```
create_table '<my_table>', ':key:string, f:<cq1>:<datatype1>, ...',
[ {<PROPERTY> => <value>, ...}]'
```

'`:key:string, f:<cq1>:<datatype1>, ...`' 定义表 `<my_table>` 的schema，`f:<cq1>` 为第一列的列名，`<datatype1>` 为 `f:<cq1>` 列的数据类型。`{<PROPERTY> => <value>, ...}` 用于指定一些可选参数，和建Transwarp HBase表的指令`create`相比，`create_table` 的第二个参数用于定义表的schema（而 `create` 的第二个参数用于指定列族），其他参数（例如`SPLITS`）和 `create` 相同，用法也相同。

例 44. 用 `create_table` 建Hyperdrive表

```
create_table 'htable', ':key:string, f:c1:int, f:c2:double,
f:c3:varchar(10), f:c4:boolean, f:c5:binary, f:c6:tinyint,
f:c7:smallint, f:c8:bigint,
f:c9:date, f:c10:timestamp, f:c11:float, f:c12:decimal(5,22),
f:c13:struct<string, int,varchar(10), decimal(9,33), boolean, float,
double, bigint, tinyint, smallint, date, timestamp>'
```

例 45. 用 `create_table` 建带有splitkey的Hyperdrive表

```
create_table 'htable1', ':key:string, f:c1:int, f:c2:double,
f:c3:varchar(10), f:c4:boolean, f:c5:binary, f:c6:tinyint,
f:c7:smallint, f:c8:bigint, f:c9:date, f:c10:timestamp, f:c11:float,
f:c12:decimal(5,22), f:c13:struct<string, int,varchar(10),
decimal(9,33), boolean, float, double, bigint, tinyint, smallint, date,
timestamp>',
SPLITS => ['10', '20', '30']
```

9.4.2. hput

语法

```
hput '<my_table>', '<row_key>', 'f:<cq>', '<value>'
```

`hput` 向Hyperdrive表中插入记录。使用时需要给定目标表 `<my_table>`、RowKey `<row_key>`、列名 `f:<cq>` 以及对应的值 `<value>`。

例 46. hput

向用 `create_table` 建Hyperdrive表中建的表 `htable` 插入数据:

```
hput 'htable', '0001', 'f:c1', '100'
hput 'htable', '0001', 'f:c2', '100.0'
hput 'htable', '0001', 'f:c3', 'varchar'
hput 'htable', '0001', 'f:c4', 'true'
hput 'htable', '0001', 'f:c5', '100011'
hput 'htable', '0001', 'f:c6', '5'
hput 'htable', '0001', 'f:c7', '10'
hput 'htable', '0001', 'f:c8', '1000000'
hput 'htable', '0001', 'f:c9', '1343243253'
hput 'htable', '0001', 'f:c10', '1343243253'
hput 'htable', '0001', 'f:c11', '1.34'
hput 'htable', '0001', 'f:c12', '0.3434'
hput 'htable', '0001', 'f:c13', '[string, 10, varchar, 1234.5E-4, true,
1.0, 1.00, 1000, 10, 15, 1111111111,1111111111]'
```

9.4.3. hget

语法

```
hget '<my_table>', '<row_key>', {<PROPERTY> => <value>, ...}
```

`hget` 从表中获取指定RowKey的记录，可选参数及它们的用法和`get`相同。

例 47. hget

获取`hput`中插入的数据:

```
hget 'htable', '0001'
```

只获取指定列中的数据:

```
hget 'htable', 'row', {COLUMN => ['f:c1', 'f:c3']}
```

9.4.4. hscan

语法

```
hscan '<my_table>', [ {<PROPERTY> => <value>, ...} ]
```

hscan 扫描指定表，用于批量获取Hyperdrive表中的数据。可选参数及它们的用法和**scan**相同。但是，**hscan**支持的 FILTER 只有 **TimestampsFilter** 和 **HSingleColumnValueFilter**，不支持其他 FILTER。

例 48. hscan

扫描整张表：

```
hscan 'htable'
```

扫描指定列：

```
hscan 'htable',{COLUMN => [ 'f:c1', 'f:c4', 'f:c6', 'f:c9' ]}
```

例 49. 在 hscan 时使用 HSingleColumnValueFilter

```
import org.apache.hadoop.hbase.filter.CompareFilter
import org.apache.hadoop.hbase.filter.HSingleColumnValueFilter
import org.apache.hadoop.hbase.filter.SubstringComparator
import org.apache.hadoop.hbase.util.Bytes

hscan 'htable', { COLUMNS => 'f:c6', FILTER =>
HSingleColumnValueFilter.newBuilder('htable', Bytes.toBytes('f'),
Bytes.toBytes('c6'), CompareFilter::CompareOp.valueOf('EQUAL'), '10')}
```

例 50. 在 hscan 时使用 TimestampsFilter

```
import org.apache.hadoop.hbase.filter.TimestampsFilter
import java.util.ArrayList

timestamps = ArrayList.new
timestamps.add(100)
timestamps.add(1000)
hscan 'htable', {COLUMNS => 'f:c6', FILTER =>
TimestampsFilter.newBuilder(timestamps)}
```

注意，不支持下面的使用方式：

```
hscan 'htable', {FILTER => "(TimestampsFilter ( 123, 456 ))"}
```

9.4.5. hdelete

语法

```
hdelete '<my_table>', '<row_key>', 'f:<cq>'
```

hdelete 删除表中某个单元格的数据，需要指定RowKey **<row_key>** 和列名 **f:<cq>**。

例 51. **hdelete**

```
hdelete 'htable', 'row', 'f:c13'
```

10. Transwarp HBase JSON配置使用说明

10.1. JSON配置操作简介

10.1.1. 表数据 vs 表的扩展数据

索引是Transwarp HBase的核心功能之一，我们在使用Transwarp HBase时，常常会为表建各类索引，包括全局索引、局部索引、全文索引和LOB索引，利用索引中的数据提高查询效率。索引中的数据不属于表数据，但是从表数据而来，和表密不可分，所以我们将表数据和它所有索引中的数据合称为 **表的扩展数据**，也就是说，我们做如下定义：

表的扩展数据 = 表数据 + 全局索引数据 + 局部索引数据 + 全文索引数据 + LOB索引数据

10.1.2. 表的元数据 vs 表的扩展元数据

Transwarp HBase表的元数据包括表名、列族名、**DATA_BLOCK_ENCODING**、**TTL**、**BLOCKSIZE** 等等。一张Transwarp HBase表的各个索引也有自己的元数据，和索引数据一样，索引的元数据和表的关系也十分紧密，所以我们将表的元数据和它所有索引的元数据合称为 **表的扩展元数据**：

表的扩展元数据 = 表的元数据 + 全局索引元数据 + 局部索引元数据 + 全文索引元数据 + LOB索引元数据

我们有时也会将表的元数据称为 **基础元数据** 或者 **base元数据**。

10.1.3. JSON配置的命令行指令

为操作表的扩展数据和扩展元数据服务，Transwarp HBase提供了 扩展的命令行指令：**describeInJson**、**alterUseJson** 和 **truncate_all**。

语法：**describeInJson**

```
describeInJson '<table>', ['true|false'] ,['/<target_dir>/<target_file>'], ['true|false']
```

说明：打印 **<table>** 的扩展元数据（以JSON串形式输出）。第二个参数为可选参数，选择true会将表的扩展元数据以格式化的JSON串打印，默认值为false。第三个参数为可选参数，指定 **/<target_dir>/<target_file>** 可以将表的扩展元数据打印到本地 **<target_dir>** 目录下的 **<target_file>** 文件中，不指定则默认将结果打印到console。如果指定打印到文件，请注意操作的用户需要有本地 **<target_dir>** 的写权限，所以保险起见可以选择使用/tmp目录。第四个参数为可选参数，指定是否将Split Keys以十六进制打印。

例 52. `describeInJson`

我们在Transwarp HBase中的jsondoc namespace下有一张空表jtable，暂时没有任何索引。

将jsondoc:jtable的扩展元数据以未格式化的JSON串形式打印到console:

```
describeInJson 'jsondoc:jtable'
```

将json:jtable的扩展元数据以格式化的JSON串形式打印到console:

```
describeInJson 'jsondoc:jtable', 'true'
```

将json:jtable的扩展元数据以格式化的JSON串形式打印到本地/tmp/jtable.txt文件中:

```
describeInJson 'jsondoc:jtable', 'true', '/tmp/jtable.txt'
```

将json:jtable的扩展元数据以格式化的JSON串形式打印到本地/tmp/jtable.txt文件中，以十六进制打印Split Keys:

```
describeInJson 'jsondoc:jtable', 'true', '/tmp/jtable.txt', 'true'
```

语法: `alterUseJson`

```
alterUseJson '<table>', '<json_string>|<dir>/<json_file>', '[true|false]'
```

说明: 如果 `<table>` 存在，根据提供的JSON串配置它的扩展元数据；如果 `<table>` 不存在，则先建表，并根据提供的JSON串配置它的扩展元数据。有两种提供JSON串的方式：直接提供一个JSON串 `<json_string>`，或者提供JSON串所在文件的路径 `/<dir>/<json_file>`。第三个参数为可选参数，指定是否以将JSON串中指定的Split Keys当做十六进制的字符读取。

例 53. `alterUseJson`

```
alterUseJson 'jsondoc:jtable', '/tmp/jtable2.txt'
```

将Split Keys当做十六进制字符打印/读取

`describeInJson` 和 `alterUseJson` 两个指令的最后一个参数指定是否将Split Keys以十六进制输出/输入。在Split Keys以十六进制字符表示的情况下，执行 `describeInJson` 和 `alterUseJson` 时必须总是将这个参数设为true，以保证在每次扩展元数据输出/输入时，Transwarp HBase可以正常地识别Split Keys。如果不进行设置，会发生Split Keys乱码，导致JSON串中的Split Key和表实际的Split Key不同。

语法: `truncate_all`

```
truncate_all '<table>'
```

说明: 清空 `<table>` 以及所有 `<table>` 的各类索引中的数据, 也就是清除表的扩展数据, 保留表的扩展元数据。



- 表的扩展元数据全部都包含在传给Transwarp HBase的JSON串中, `alterUseJson` 会根据JSON描述来创建和修改表的扩展元数据, 将表配置成输入的JSON串描述相同的结构。在[JSON配置详解](#)中, 我们会详细介绍怎样写JSON串来配置扩展元数据。

我们可以和命令行中的基础命令 `describe`、`alter` 和 `truncate` 大致地做下面的类比:

普通指令	扩展指令
<code>describe</code> (打印元数据)	<code>describeInJson</code> (打印扩展元数据)
<code>truncate</code> (清空表数据, 保留基础元数据)	<code>truncate_all</code> (清空扩展数据, 保留扩展元数据)
<code>alter</code> (配置元数据)	<code>alterUseJson</code> (配置扩展元数据)



- `alter` 指令不能建表, 只能修改表的元数据; 而 `alterUseJson` 可以建表。
- 扩展指令可以对表的扩展元数据进行统一操作和管理, 而普通指令只能修改表的基础元数据, 容易导致表(元)数据和表的扩展(元)数据不统一的情况(譬如, 某表被执行`truncate`后, 其索引数据并不会被清空)。所以我们建议尽量使用扩展指令。

10.2. JSON配置详解

10.2.1. 扩展元数据JSON串的基本格式

`describeInJson`中的`jsondoc:jtable`的扩展元数据如下:

```
{
  "tableName" : "jsondoc:jtable", ①
  "base" : {
    "families" : [ {
      "FAMILY" : "f1",
      "DATA_BLOCK_ENCODING" : "NONE",
      "BLOOMFILTER" : "ROW",
      "REPLICATION_SCOPE" : "0",
      "VERSIONS" : "1",
      "COMPRESSION" : "NONE",
      "MIN VERSIONS" : "0",
      "TTL" : "2147483647",
      "KEEP_DELETED_CELLS" : "FALSE",
      "BLOCKSIZE" : "65536",
      "IN_MEMORY" : "false",
      "BLOCKCACHE" : "true"
    } ],
    "THEMIS_ENABLE" : false
  },
  "fulltextindex" : { ③
  },
  "globalindex" : { ④
    "indexs" : [ ]
  },
  "lob" : { ⑤
    "indexs" : [ ]
  }
}
```

① 表名，可选项。Transwarp HBase会根据`alterUseJson` 指令后提供的表名来判断应该为哪张表配置扩展元数据。

② 基础（base）模块，必填项，包含了表的基础元数据。

③ fulltextindex模块，选填项，包含了表fulltext index的元数据。

④ globalindex模块，选填项，包含了表global index的元数据。

⑤ lob模块，选填项，包含了表lob的元数据。

这也是任何一张Transwarp HBase表的扩展元数据的JSON串都具有的基本格式，概括总结为以下几个模块：

```
{
  "tableName" : "<table_name>"
  "base" : { ... },
  "fulltextindex" : { ... },
  "globalindex" : { ... },
  "lob" : { ... }
}
```

当执行 `alterUseJson` 来根据输入的JSON串来配置表时，Transwarp HBase会直接将表的扩展元数据修改为和输入的JSON串一致。也就是说：

- 如果输入的JSON串相对于表当前的扩展元数据减少了一部分信息，Transwarp HBase会删除这部分信息。注意，扩展元数据中的所有可选配置项都有默认值，所以“将某配置项删除”等同于“将某配置项的值恢复为

默认值”。

- 如果输入的JSON串相对于表当前的扩展元数据增加了一部分信息，Transwarp HBase会添加这部分信息；
- 如果输入的JSON串相对于表当前的扩展元数据有一部分信息发生了变化，Transwarp HBase会更新这部分信息。

JSON串的编写规则

关于拼写：

您在编写JSON串时，需要保证配置项的名称和值的拼写和我们提供的 完全一致。其中尤其要注意的几点为：

- 除了fulltextindex模块和部分索引模块的配置项，大多数配置项都要放在双引号中，我们建议您使用我们提供的模板编写JSON串。同时您还可以尽量 复用 JSON串——让 `describeInJson` 打印到文件，在文件基础上修改，减少不必要的错误。
- 除了fulltextindex模块以外，所有模块中的配置项名称都需要 大写，除非另外指出，配置项的值都需要放在双引号中。同样，我们建议您使用我们提供的模板，或者复用您自己的JSON串。
- 因为fulltextindex的信息需要和ElasticSearch共享，fulltextindex模块的拼写和其他模块不同，有专门的规定，请参考[fulltextindex模块](#)来了解细节。

关于JSON串中的顺序：

- JSON串中的模块顺序没有规定。
- JSON串中配置项的先后顺序没有规定。
- `"SPLIT_KEYS"` 组中元素出现顺序没有规定。
- 有的配置项组内元素的顺序有规定，包括：
 - `globalindex`模块中的 `"indexColumnInfo"` 配置项

下面我们分别介绍base、fulltextindex、globalindex和lob模块分别的配置细节。

10.2.2. base模块

base模块是表的扩展元数据中必须的模块，任何一个 `alterUseJson` 传递给Transwarp HBase的JSON串都必须有该模块。一张Transwarp HBase表的base模块具有如下格式：

普通Transwarp HBase表的base模块格式

```
"base" : {
  "SPLIT_KEYS" : [ "<split_key1>", "<split_key2>", ... ], ①
  "families" : [ {<cf_meta>} , {<cf_meta>} , ... ] , ②
  "THEMIS_ENABLE" : false
}
```

① `"SPLIT_KEYS"` 配置项指定该表的Split Key，该配置项为 选填项。Split Key只能在建表时指定，建表后无法修改，所以建表后传给Transwarp HBase的JSON串中这一配置项都无效。

- ② "families" 配置项配置表中列族的元数据，该配置项为 必填项。它由一组列族元数据组成：每个元素 {<cf_meta>} 各包含一个列族的元数据，组中至少要有一个元素。列族的元数据配置 {<cf_meta>} 有固定的格式，在 {<cf_meta>} 的配置中会详细介绍。

{<cf_meta>} 的配置

```
{
    "FAMILY": "<column_family>", // 必选项，指定列族名。
    "DATA_BLOCK_ENCODING": "<encoding_scheme>", // 可选项，默认为
    NONE, 无编码方式
    "BLOOMFILTER": "<bloomfilter>", // 可选项，默认为NONE
    "REPLICATION_SCOPE": "<int>", // 可选项，默认为0
    "VERSIONS": "<num_versions>", // 可选项，默认为1
    "COMPRESSION": "<compression_scheme>", // 可选项，默认为NONE
    "MIN VERSIONS": "<num_minversions>", // 可选项，默认为0
    "TTL": "<ttl>", // 可选项，默认为Integer.Max
    "KEEP_DELETED_CELLS": "<boolean>", // 可选项，请指定为FALSE
    "BLOCKSIZE": "<blocksize>", // 可选项，默认65535
    "IN_MEMORY": "<boolean>", // 可选项，默认为false
    "BLOCKCACHE": "<boolean>" // 可选项，默认为true
}
```

表 6. <cf_meta> 中的可选配置项信息

配置项	选项	默认值	推荐值
DATA_BLOCK_ENCODING	NONE, PREFIX, DIFF, FAST_DIFF, PREFIX_TREE, INDEXED_PREFIX	NONE	PREFIX 或 PREFIX_TREE
BLOOMFILTER	NONE, ROW, ROWCOL	NONE	ROW
REPLICATION_SCOPE	0 (表示local scope), 1 (表示global scope)	0	0
VERSIONS	1或更多	1	1
COMPRESSION	LZO, GZ, NONE, SNAPPY, LZ4	NONE	SNAPPY
MIN VERSIONS	0或更多	0	0
TTL	正整数 (表示秒数)	INT最大值2147483647, 表示永久	按业务确定表的 TTL, 如对于LOB这样的特大列族, 设置合理的 TTL 是一个删除过期旧数据的好方法。
KEEP_DELETED_CELLS	true或false	false	false
BLOCKSIZE	正整数 (表示一个Hfile中的byte数量)	65536	65536
IN_MEMORY	true或false	false	false
BLOCKCACHE	true或false	false	false

例 54. 有两个列族的Transwarp HBase表的base模块

下面是一张有两个列族 (f1, f2) 的Transwarp HBase表的base模块:

```
"base" : {
    "families" : [ {
        "FAMILY" : "f1", // 列族f1
        "DATA_BLOCK_ENCODING" : "NONE",
        "BLOOMFILTER" : "ROW",
        "REPLICATION_SCOPE" : "0",
        "VERSIONS" : "1",
        "COMPRESSION" : "NONE",
        "MIN VERSIONS" : "0",
        "TTL" : "2147483647",
        "KEEP_DELETED_CELLS" : "FALSE",
        "BLOCKSIZE" : "65536",
        "IN_MEMORY" : "false",
        "BLOCKCACHE" : "true"
    }, {
        "FAMILY" : "f2", // 列族f2
        "DATA_BLOCK_ENCODING" : "NONE",
        "BLOOMFILTER" : "ROW",
        "REPLICATION_SCOPE" : "0",
        "VERSIONS" : "1",
        "COMPRESSION" : "NONE",
        "MIN VERSIONS" : "0",
        "TTL" : "2147483647",
        "KEEP_DELETED_CELLS" : "FALSE",
        "BLOCKSIZE" : "65536",
        "IN_MEMORY" : "false",
        "BLOCKCACHE" : "true"
    } ],
    "THEMIS_ENABLE" : false
}
```

10.2.3. fulltextindex模块

fulltextindex模块是表的扩展元数据中的可选模块，您只需在为表建全文索引时在JSON串中填写该模块。

fulltextindex模块

```
"fulltextindex" : {
    "tableName" : "<affiliated_table>", ①
    "allowUpdate" : <boolean>, //可选项，是否允许对建索引列进行update
操作，默认为true
    "ttl" : <int>, //可选项，单位为毫秒，默认为0
    "source" : <boolean>, //可选项，是否在ES中存储_source信息，默认为true
    "all" : <boolean>, //可选项，是否在ES中存储_all信息，默认为false
    "fields" : [{<field_meta>}, {<field_meta>}, ... ] ②
}
```

① 必填项，指定全文索引所属表的名称。注意和base模块的同名配置项的意义区分。

② "fields" 配置项用于配置建全文索引的各个字段，为必选项。它由一组字段配置信息组成：每个元素 {`<field_meta>`} 各包含一个字段的配置，组中至少要有一个元素。字段的配置 `<field_meta>` 有固定的格式，在 {`<field_meta>`} 的配置中会详细介绍。

表 7. fulltextindex模块中的配置项

配置项	选项	默认值	推荐值
<code>allowUpdate</code>	true或false	true	true
<code>ttl</code>	正整数（表示毫秒数）	0	如果没有超期限制，使用默认值。
<code>source</code>	true或false	true	true
<code>all</code>	true或false	false	false



和其他模块不同，fulltextindex模块中的配置项都 必须小写，并且，配置项的值的拼写，包括大小写和是否放在双引号之间，必须和上面提供的完全一致。为了减少配置出错的情况，我们建议使用我们提供的[创建/修改fulltext索引](#)和[创建/修改fulltext分索引](#)这两个模板。

{`<field_meta>`} 的配置

```
{
  "family" : "<column_family_name>", // 必填项，字段所在列族的名称
  "qualifier" : "<column_qualifier>", // 必填项，字段的column qualifier
  "encode_as_string" : <boolean>, ①
  "attributes" : {
    "index" : "<index_option>", // 可选项，索引方式。
    "store" : "<boolean>", // 可选项，指定是否存储。
    "doc_values" : "<boolean>", // 可选项，指定是否使用doc_values优化。
    "type" : "<data_type>" ②
  }
}
```

① 可选项，这个配置项 非常重要，它指定了是否将该字段作为STRING类型转换为byte[]——如果它为true，则将该字段作为STRING类型转换；如果它为false，则根据该字段的 实际类型 转换。该配置项的默认值为 false，即按字段的实际类型转换。但是如果您从Inceptor Engine映射表向Transwarp HBase导入数据，Transwarp HBase会默认将数据当做STRING转换（除非使用 #b 关键字显示指定）。所以，您需要格外注意该字段的数据是如何转换的，如果使用默认方式从Inceptor Engine中导入数据（不根据数据实际类型转换），那么您需要将 `encode_as_string` 值设为true。

② 必填项，指定该字段的数据类型。目前支持的数据类型有：string, float, double, byte, short, integer, long和boolean。

表 8. `<field_meta>` 中的可选配置项信息

配置项	选项	默认值	推荐值
<code>store</code>	true或false	true	true
<code>index</code>	analyzed, not_analyzed, no	not_analyzed	not_analyzed

配置项	选项	默认值	推荐值
doc_values	true或false	true	true

例 55. 用两个字段创建全文索引的表的fulltextindex模块

下面是一张名为 `hyper:test` 的表的fulltextindex模块。该全文索引使用了两个字段创建。

```
"fulltextindex" : {
  "tableName" : "hyper:test",
  "allowUpdate" : true,
  "ttl" : 0,
  "source" : true,
  "all" : false,
  "storeAsSource" : false, ①
  "storeFamily" : "", ①
  "writeConsistencyLevel" : "default", ①
  "fields" : [ {
    "family" : "f", // 字段1的field_meta
    "qualifier" : "q1",
    "encode_as_string" : true,
    "attributes" : {
      "index" : "not_analyzed",
      "store" : "true",
      "doc_values" : "true",
      "type" : "string"
    }
  }, { // 字段2的field_meta
    "family" : "f",
    "qualifier" : "q3",
    "encode_as_string" : true,
    "attributes" : {
      "index" : "not_analyzed",
      "store" : "true",
      "doc_values" : "true",
      "type" : "double"
    }
  } ]
}
```

① 1 Transwarp HBase会自动生成这三项，您应当永远使用系统默认值，无需您自己配置。

10.2.4. globalindex模块

globalindex模块是表的扩展元数据中的可选模块，您只需在为表建全局索引时在JSON串中填写该模块，它的格式如下：

globalindex模块格式

```
"globalindex": {
    "indexes": [{<global_index_meta>}, {<global_index_meta>}, ...]
}
```

globalindex模块下只有一个配置项 "indexes"，为必填项。它是由表的所有全局索引元数据构成的组，组中的每个元素 {<global_index_meta>} 对应表的一个全局索引的元数据，组中至少要有一个元素。全局索引的元数据 <global_index_meta> 有固定的格式，在{<global_index_meta>} 的配置中会详细介绍。

{<global_index_meta>} 的配置

```
{
    "INDEX_NAME": "<global_index_name>", //必填项，指定索引名称
    "SPLIT_KEYS": [ "<split_key1>", "<split_key2>" ], //可选项，指定Split Key。默认无Split key，即只有一个region。只能在建索引时指定，建好后不能修改。
    "UPDATE": "<boolean>", //可选项，默认为true
    "DCOP": "<boolean>", //可选项，默认为true
    "INDEX_CLASS": "COMBINE_INDEX", //必填项，值填为COMBINE_INDEX
    "indexColumnInfos": [{<index_column_info>}, {<index_column_info>},
    ..., {<rowkey_info>}] ①
}
```

- ① 必填项。它是一个组，组中的元素 {<index_column_info>} 是各个构成该全局索引的列的信息，包括该列的列族，qualifier和在索引词条中所占的长度。组中的最后一个元素 {<rowkey_info>} 是Row Key信息。组中至少要有两个元素：一个属于构成索引的列，另一个属于Row Key。列信息 {<index_column_info>} 如{<index_column_info>} 的配置中所示，Row Key信息 {<rowkey_info>} 如{<rowkey_info>} 的配置中所示。注意，列信息的先后顺序决定了各列在索引词条中出现的顺序，所以这里组中元素的先后顺序是有意义的，需要您根据自己对索引的设计排列。

表 9. {<global_index_meta>} 中的可选配置项

配置项	选项	默认值	推荐值
UPDATE	true或false (指定是否更新)	true	true
DCOP	true或false	true	true

{<index_column_info>} 的配置

```
{
    "FAMILY" : "<column_family>", // 必填项，该列所属的列族名
    "QUALIFY" : "<column_qualifier>", //必填项，该列的column qualifier
    "SEGMENT_LENGTH" : <int> // 必填项，该列在全局索引词条中所占长度
}
```

{<rowkey_info>} 的配置

```
{  
    "FAMILY" : "rowKey", // 必填项, rowKey为固定用法  
    "QUALIFY" : "rowKey", //必填项, rowKey为固定用法  
    "SEGMENT_LENGTH" : <int> //必填项, 该列在全局索引词条中所占长度  
}
```

注意, **SEGMENT_LENGTH** 的值不放在引号中。

SEGMENT_LENGTH 的选择

总得来说，列的 **SEGMENT_LENGTH** 和列值的平均长度相近即可。但是列值的长度如何计算呢？Transwarp HBase表中的数据以byte[]形式存储，所有类型的数据在存入Transwarp HBase时都会被转换成byte[]，列值的长度和 [转换成byte\[\]](#) 的方式直接相关。将数据转换成byte[]有两种方式：

方法一

将数据的值当做STRING类型转换成byte[]

方法二

根据数据的实际类型转换成对应byte[]

如果用[方法一](#)转换，那么 **SEGMENT_LENGTH** 应当设置成列值作为STRING类型的平均长度；如果用[方法二](#)转换，那么根据数据类型的不同，**SEGMENT_LENGTH** 也不同，具体如下表：

表 10. 数据类型和对应的byte[]长度

数据类型	byte[]长度
BOOLEAN	1
TINYINT	1
SMALLINT	2
INTEGER	4
BIGINT	8
FLOAT	4
DOUBLE	8
STRING	字符串长度
VARCHAR	字符串长度

所以，您不仅需要对建索引列的值有大致的了解，还需要知道建索引列的数据存入Transwarp HBase时是按照哪种方法转换的。

如果您选择通过Inceptor Engine的映射表向Transwarp HBase插入数据，可以在建表时指定数据是否按类型转换：

指定数据是否按类型转换

```
CREATE TABLE ... STORED BY
  'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
  WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key, f:q1, f:q2#b");
```

在 "**hbase.columns.mapping**" 中，列名后添加 **#b** 为指定按照数据实际类型转换为byte[] ([方法二](#))，如果没有添加 **#b** 则默认将数据当做STRING转换为byte[] ([方法一](#))。

例 56. 一张有两个全局索引的表的globalindex模块

下面的globalindex模块属于一张有两个全局索引的表。两个全局索引分别名为 `name_balance_global_index` 和 `name_global_index`。其中，`name_balance_global_index` 由两列建成，`name_global_index` 由一列建成。

```
"globalindex" : {
  "indexes" : [ {
    //第一个全局索引
    "INDEX_NAME" : "name_balance_global_index",
    "UPDATE" : "true",
    "DCOP" : "true",
    "INDEX_CLASS" : "COMBINE_INDEX",
    "indexColumnInfos" : [ { //建name_balance_global_index的列之一
      "FAMILY" : "f",
      "QUALIFY" : "q1",
      "SEGMENT_LENGTH" : 8
    }, { // 建name_balance_global_index的列之二
      "FAMILY" : "f",
      "QUALIFY" : "q3",
      "SEGMENT_LENGTH" : 9
    }, {
      "FAMILY" : "rowKey",
      "QUALIFY" : "rowKey",
      "SEGMENT_LENGTH" : 16
    } ],
    //第二个全局索引
    "INDEX_NAME" : "name_global_index",
    "UPDATE" : "true",
    "DCOP" : "true",
    "INDEX_CLASS" : "COMBINE_INDEX",
    "indexColumnInfos" : [ { //建name_global_index的列
      "FAMILY" : "f",
      "QUALIFY" : "q1",
      "SEGMENT_LENGTH" : 8
    }, {
      "FAMILY" : "rowKey",
      "QUALIFY" : "rowKey",
      "SEGMENT_LENGTH" : 16
    } ]
  } ]
}
```

10.2.5. lob模块

lob模块是表中LOB列族的lob索引元数据。当表中有LOB列族时，表的base模块中LOB列族的元数据需要多一个配置项，同时表的扩展元数据中 必须要包含lob模块。

LOB列族的列族元数据 (`{<cf_meta>}`) 配置

```
{
    "FAMILY": "<column_family>", // 必选项，指定列族名。
    "DATA_BLOCK_ENCODING": "<encoding_scheme>", // 可选项，默认为
    NONE, 无编码方式
    "BLOOMFILTER": "<bloomfilter>", // 可选项，默认为NONE
    "REPLICATION_SCOPE": "<int>", // 可选项，默认为0
    "VERSIONS": "<num_versions>", // 可选项，默认为1
    "COMPRESSION": "<compression_scheme>", // 可选项，默认为NONE
    "MIN VERSIONS": "<num_minversions>", // 可选项，默认为0
    "TTL": "<ttl>", // 可选项，默认为Integer.Max
    "KEEP_DELETED_CELLS": "<boolean>", // 必选项，请指定为FALSE
    "BLOCKSIZE": "<blocksize>", // 可选项，默认65535
    "IN_MEMORY": "<boolean>", // 可选项，默认为false
    "BLOCKCACHE": "<boolean>" // 可选项，默认为true
}
```

注，LOB列族的 `{<cf_meta>}` 中的可选配置项和普通列族的 `{<cf_meta>}` 相同，请参考 `<cf_meta>` 中的可选配置项信息。

lob模块的格式

```
"lob" : {
    "indexes" : [ {<lob_index_meta>}, {<lob_index_meta>}, ... ]}
```

lob模块下只有一个配置项 `"indexes"`，为必填项。它是由表的所有LOB索引元数据构成的组，组中的每个元素 `{<lob_index_meta>}` 对应表的一个LOB索引的元数据，组中至少要有一个元素。：LOB索引的元数据 `{<lob_index_meta>}` 有固定的格式，在 `{<lob_index_meta>}` 格式中会详细介绍。

{<lob_index_meta>} 格式

```
{  
    "INDEX_NAME" : "<lob_index_name>", // 必填项, LOB索引名  
    "DCOP" : "<boolean>",  
    "INDEX_CLASS" : "LOB_INDEX", //必填项, 必须填LOB_INDEX  
    "DATA_BLOCK_ENCODING" : "<encoding_scheme>",  
    "BLOOMFILTER" : "<bloomfilter>",  
    "REPLICATION_SCOPE" : "<int>",  
    "COMPRESSION" : "<compression_scheme>",  
    "VERSIONS" : "<int>",  
    "TTL" : "<int>",  
    "MIN VERSIONS" : "<int>",  
    "KEEP_DELETED_CELLS" : "<boolean>",  
    "BLOCKSIZE" : "<int>",  
    "SPLIT_FAMILY" : "<boolean>",  
    "IN_MEMORY" : "<boolean>",  
    "BLOCKCACHE" : "<boolean>",  
    "indexColumnInfos" : [ {  
        "FAMILY" : "<lob_column_family>", // 必填项, LOB列的列族名  
        "QUALIFY" : "" //留为空  
    } ]  
}
```

注: {<lob_index_meta>} 中的可选配置项和 {<cf_meta>} 以及 {<global_index_meta>} 中的可选配置项重叠, 请参考<cf_meta> 中的可选配置项信息以及{<global_index_meta>} 中的可选配置项。

例 57. 一张有LOB列族的表的扩展元数据

```
{
  "tableName" : "test_json",
  "base" : {
    "families" : [ {
      "FAMILY" : "f",
      "DATA_BLOCK_ENCODING" : "PREFIX",
      "BLOOMFILTER" : "ROW",
      "REPLICATION_SCOPE" : "0",
      "COMPRESSION" : "SNAPPY",
      "VERSIONS" : "1",
      "MIN VERSIONS" : "0",
      "TTL" : "2147483647",
      "hbase.hstore.defaultengine.compactionpolicy.class" :
        "org.apache.hadoop.hbase.regionserver.compactions.ExploringWithLargeHFi
        leCompactionPolicy", ①
      "KEEP_DELETED_CELLS" : "false",
      "BLOCKSIZE" : "65536",
      "IN_MEMORY" : "false",
      "SPLIT_FAMILY" : "false",
      "BLOCKCACHE" : "true"
    } ]
  },
  "lob" : {
    "indexs" : [ {
      "INDEX_NAME" : "LOBP",
      "DCOP" : "true",
      "INDEX_CLASS" : "LOB_INDEX",
      "DATA_BLOCK_ENCODING" : "PREFIX",
      "BLOOMFILTER" : "ROW",
      "REPLICATION_SCOPE" : "0",
      "COMPRESSION" : "NONE",
      "VERSIONS" : "1",
      "TTL" : "2147483647",
      "MIN VERSIONS" : "0",
      "KEEP_DELETED_CELLS" : "false",
      "BLOCKSIZE" : "65536",
      "SPLIT_FAMILY" : "true",
      "IN_MEMORY" : "false",
      "BLOCKCACHE" : "true",
      "indexColumnInfos" : [
        {
          "FAMILY" : "f",
          "QUALIFY" : ""
        }
      ]
    } ]
  }
}
```

① Transwarp HBase会根据lob模块下的 `indexColumnInfos` 下的 `FAMILY` 值判断哪一个列族为LOB列族，并自动为该列族加上这个配置项和它的值。您无需在JSON串中设置。

10.3. JSON配置操作模板

下面提供的模板在TDH4.2及以后的版本中都适用。

创建/修改一张最基础的表

```
{  
    "tableName" : "test_json",  
    "base" : {  
        "families" : [ {  
            "FAMILY" : "f",  
            "DATA_BLOCK_ENCODING" : "PREFIX",  
            "BLOOMFILTER" : "ROW",  
            "REPLICATION_SCOPE" : "0",  
            "VERSIONS" : "1",  
            "COMPRESSION" : "SNAPPY",  
            "MIN VERSIONS" : "0",  
            "TTL" : "2147483647",  
            "KEEP_DELETED_CELLS" : "false",  
            "BLOCKSIZE" : "65536",  
            "IN_MEMORY" : "false",  
            "BLOCKCACHE" : "true"  
        } ]  
    }  
}
```

创建/修改global索引

```
{  
    "tableName" : "test_json",  
    "base" : {  
        "families" : [ {  
            "FAMILY" : "f",  
            "DATA_BLOCK_ENCODING" : "PREFIX",  
            "BLOOMFILTER" : "ROW",  
            "REPLICATION_SCOPE" : "0",  
            "VERSIONS" : "1",  
            "COMPRESSION" : "SNAPPY",  
            "MIN VERSIONS" : "0",  
            "TTL" : "2147483647",  
            "KEEP_DELETED_CELLS" : "false",  
            "BLOCKSIZE" : "65536",  
            "IN_MEMORY" : "false",  
            "BLOCKCACHE" : "true"  
        } ]  
    },  
    "globalindex" : {  
        "indexs" : [ {  
            "INDEX_NAME" : "index_q",  
            "UPDATE" : "true",  
            "DCOP" : "true",  
            "INDEX_CLASS" : "COMBINE_INDEX",  
            "indexColumnInfo" : [ {  
                "FAMILY" : "f",  
                "QUALIFY" : "q",  
                "SEGMENT_LENGTH" : 16  
            }, {  
                "FAMILY" : "rowKey",  
                "QUALIFY" : "rowKey",  
                "SEGMENT_LENGTH" : 16  
            } ]  
        } ]  
    }  
}
```

注意，`SEGMENT_LENGTH` 的值不放在双引号中。

创建/修改LOB索引

```
{
  "tableName" : "test_json",
  "base" : {
    "families" : [ {
      "FAMILY" : "f",
      "DATA_BLOCK_ENCODING" : "PREFIX",
      "BLOOMFILTER" : "ROW",
      "REPLICATION_SCOPE" : "0",
      "COMPRESSION" : "SNAPPY",
      "VERSIONS" : "1",
      "MIN VERSIONS" : "0",
      "TTL" : "2147483647",
      "hbase.hstore.defaultengine.compactionpolicy.class" :
"org.apache.hadoop.hbase.regionserver.compactions.ExploringWithLargeHFileC
ompactionPolicy",
      "KEEP_DELETED_CELLS" : "false",
      "BLOCKSIZE" : "65536",
      "IN_MEMORY" : "false",
      "SPLIT_FAMILY" : "false",
      "BLOCKCACHE" : "true"
    } ]
  },
  "lob" : {
    "indexes" : [ {
      "INDEX_NAME" : "LOBP",
      "DCOP" : "true",
      "INDEX_CLASS" : "LOB_INDEX",
      "DATA_BLOCK_ENCODING" : "PREFIX",
      "BLOOMFILTER" : "ROW",
      "REPLICATION_SCOPE" : "0",
      "COMPRESSION" : "NONE",
      "VERSIONS" : "1",
      "TTL" : "2147483647",
      "MIN VERSIONS" : "0",
      "KEEP_DELETED_CELLS" : "false",
      "BLOCKSIZE" : "65536",
      "SPLIT_FAMILY" : "true",
      "IN_MEMORY" : "false",
      "BLOCKCACHE" : "true",
      "indexColumnInfos" : [ {
        "FAMILY" : "f",
        "QUALIFY" : ""
      } ]
    } ]
  }
}
```

创建/修改fulltext索引

```
{  
    "tableName" : "test_json",  
    "base" : {  
        "families" : [ {  
            "FAMILY" : "f",  
            "DATA_BLOCK_ENCODING" : "PREFIX",  
            "BLOOMFILTER" : "ROW",  
            "REPLICATION_SCOPE" : "0",  
            "VERSIONS" : "1",  
            "COMPRESSION" : "SNAPPY",  
            "MIN VERSIONS" : "0",  
            "TTL" : "2147483647",  
            "KEEP_DELETED_CELLS" : "false",  
            "BLOCKSIZE" : "65536",  
            "IN_MEMORY" : "false",  
            "BLOCKCACHE" : "true"  
        } ]  
    },  
    "fulltextindex" : {  
        "tableName" : "test_json",  
        "allowUpdate" : "true",  
        "ttl" : 0,  
        "source" : true,  
        "fields" : [ {  
            "family" : "f",  
            "qualifier" : "a",  
            "encode_as_string" : true,  
            "attributes" : {  
                "index" : "not_analyzed",  
                "store" : "true",  
                "doc_values" : "true",  
                "type" : "string"  
            }  
        }, {  
            "family" : "f",  
            "qualifier" : "b",  
            "encode_as_string" : true,  
            "attributes" : {  
                "index" : "not_analyzed",  
                "store" : "true",  
                "doc_values" : "true",  
                "type" : "string"  
            }  
        }]  
    }  
}
```

创建/修改fulltext分索引

```
{
  "tableName" : "test_json",
  "base" : {
    "families" : [ {
      "FAMILY" : "f",
      "DATA_BLOCK_ENCODING" : "PREFIX",
      "BLOOMFILTER" : "ROW",
      "REPLICATION_SCOPE" : "0",
      "VERSIONS" : "1",
      "COMPRESSION" : "SNAPPY",
      "MIN VERSIONS" : "0",
      "TTL" : "2147483647",
      "KEEP_DELETED_CELLS" : "false",
      "BLOCKSIZE" : "65536",
      "IN_MEMORY" : "false",
      "BLOCKCACHE" : "true"
    } ]
  },
  "fulltextindex" : {
    "tableName" : "test_json",
    "allowUpdate" : "true",
    "ttl" : 0,
    "source" : true,
    "fields" : [ {
      "family" : "f",
      "qualifier" : "a",
      "encode_as_string" : true,
      "attributes" : {
        "index" : "not_analyzed",
        "store" : "true",
        "doc_values" : "true",
        "type" : "string"
      }
    }, {
      "family" : "f",
      "qualifier" : "b",
      "encode_as_string" : true,
      "attributes" : {
        "index" : "not_analyzed",
        "store" : "true",
        "doc_values" : "true",
        "type" : "string"
      }
    }],
    "splits" : {
      "family" : "f",
      "qualifier" : "b",
      "splitKeys" : [ "9223370633623114807", "9223370647781835807",
      "9223370647781839807" ]
    }
  }
}
```

10.4. JSON配置简单使用实例

本章我们将用一些实际操作来演示怎样使用Transwarp HBase JSON配置操作来完成一些常见的简单任务。操作前，我们指出下面几个要点：

- 传给Transwarp HBase的JSON串必须有base模块。如果base模块中有LOB列族，那么JSON串中还必须有lob模块。
- 如果您需要使用可选配置项的默认值，请直接将该配置项在JSON串中省略。
- 我们建议使用JSON操作修改扩展元数据时，总是先将当前的扩展元数据输出到文件，在该文件的基础上修改，再将修改好的文件重新传给Transwarp HBase，以减少不必要的错误。在实际生产中，我们推荐您根据您的应用场景在建表时一次性建好表所需的所有索引，尽量避免建表后，尤其是表中有数据后，对元数据的修改。

为了表述的简洁，我们做出下面规定：

- 当我们说“对表jsondoc:jtable使用jtable.txt文件”，我们指的是执行：

```
alterUseJson 'jsondoc:jtable', '/tmp/jtable.txt'
```

我们将所有的JSON串文件存在本地的/tmp目录下。

- 当我们说“查看表test_table的扩展元数据”，我们指的是执行：

```
describeInJson 'jsondoc:jtable', 'true', '/tmp/file.txt'
```

然后查看文件file.txt中的内容。

例 58. 建表

假设Transwarp HBase中没有名为jsondoc:jtable的表。将下面的JSON串存在本地的/tmp/jtable.txt文件中（这个JSON串会建一张有一个列族f1的表，表的Split Keys为1, 2, 3。）：

```
{
  "tableName": "jsondoc:jtable",
  "base": {
    "SPLIT_KEYS": ["1", "2", "3"],
    "families": [
      {
        "FAMILY": "f1"
      }
    ]
  }
}
```

然后执行：

```
alterUseJson 'jsondoc:jtable', '/tmp/jtable.txt'
```

现在查看jsondoc:jtable的扩展元数据：

```
{
  "tableName" : "jsondoc:jtable",
  "base" : {
    "SPLIT_KEYS" : [ "1", "2", "3" ],
    "families" : [ {
      "FAMILY" : "f1",
      "DATA_BLOCK_ENCODING" : "NONE",
      "BLOOMFILTER" : "ROW",
      "REPLICATION_SCOPE" : "0",
      "VERSIONS" : "1",
      "COMPRESSION" : "NONE",
      "MIN_VERSIONS" : "0",
      "TTL" : "2147483647",
      "KEEP_DELETED_CELLS" : "FALSE",
      "BLOCKSIZE" : "65536",
      "IN_MEMORY" : "false",
      "BLOCKCACHE" : "true"
    } ],
    "THEMIS_ENABLE" : false
  },
  "fulltextindex" : {
  },
  "globalindex" : {
    "indexes" : [ ]
  },
  "lob" : {
    "indexes" : [ ]
  },
  "localindex" : {
    "indexes" : [ ]
  }
}
```

例 59. 添加和删除列族

对[建表](#)中建的表jsondoc:jtable使用下面的JSON串，将表中的f1列族删除，增加f2列族：

```
{  
    "tableName" : "jsondoc:jtable",  
    "base" : {  
        "families" : [ {  
            "FAMILY" : "f2",  
            "DATA_BLOCK_ENCODING" : "NONE",  
            "BLOOMFILTER" : "ROW",  
            "REPLICATION_SCOPE" : "0",  
            "VERSIONS" : "1",  
            "COMPRESSION" : "NONE",  
            "MIN VERSIONS" : "0",  
            "TTL" : "2147483647",  
            "KEEP_DELETED_CELLS" : "FALSE",  
            "BLOCKSIZE" : "65536",  
            "IN_MEMORY" : "false",  
            "BLOCKCACHE" : "true"  
        } ],  
        "THEMIS_ENABLE" : false  
    },  
    "fulltextindex" : {  
    },  
    "globalindex" : {  
        "indexes" : [ ]  
    },  
    "lob" : {  
        "indexes" : [ ]  
    }  
}
```

例 60. 修改列族属性

对[添加和删除列族](#)中的表jsondoc:jtable使用下面的JSON串，会将表中列族f2的 **COMPRESSION** 配置项设为 **SNAPPY**, **DATA_BLOCK_ENCODING** 设为 **PREFIX**:

```
{  
    "tableName" : "jsondoc:jtable",  
    "base" : {  
        "families" : [ {  
            "FAMILY" : "f2",  
            "DATA_BLOCK_ENCODING" : "PREFIX",  
            "BLOOMFILTER" : "ROW",  
            "REPLICATION_SCOPE" : "0",  
            "VERSIONS" : "1",  
            "COMPRESSION" : "SNAPPY",  
            "MIN VERSIONS" : "0",  
            "TTL" : "2147483647",  
            "KEEP_DELETED_CELLS" : "FALSE",  
            "BLOCKSIZE" : "65536",  
            "IN_MEMORY" : "false",  
            "BLOCKCACHE" : "true"  
        } ],  
        "THEMIS_ENABLE" : false  
    },  
    "fulltextindex" : {  
    },  
    "globalindex" : {  
        "indexes" : [ ]  
    },  
    "lob" : {  
        "indexes" : [ ]  
    }  
}
```

例 61. 建表的同时创建全局索引

假设Transwarp HBase中没有jsondoc:jtable_gi这张表，使用下面的JSON串会创建这张表，同时为表建全局索引|index_g:

```
{  
    "tableName": "jsondoc:jtable_gi",  
    "base": {  
        "families": [ {  
            "FAMILY": "f1",  
        } ]  
    },  
    "globalindex": {  
        "indexes": [  
            {  
                "INDEX_NAME": "index_g",  
                "SPLIT_KEYS": ["a", "b"],  
                "UPDATE": false,  
                "DCOP": true,  
                "INDEX_CLASS": "COMBINE_INDEX",  
                "indexColumnInfo": [  
                    {  
                        "FAMILY": "f1",  
                        "QUALIFY": "g",  
                        "SEGMENT_LENGTH": 8  
                    },  
                    {  
                        "FAMILY": "rowKey",  
                        "QUALIFY": "rowKey",  
                        "SEGMENT_LENGTH": 10  
                    }  
                ]  
            }  
        ]  
    }  
}
```

例 62. 建表的同时创建全文索引

假设Transwarp HBase中没有jsondoc:jtable_fu这张表，使用下面的JSON串会创建这张表，同时为表建全文索引：

```
{  
    "tableName": "jsondoc:jtable_fu",  
    "base" : {  
        "families" : [ {  
            "FAMILY" : "f1",  
        } ]  
    },  
    "fulltextindex": {  
        "tableName": "jsondoc:jtable_fu",  
        "fields": [{  
            "family": "f1",  
            "qualifier": "q1",  
            "encode_as_string" : true,  
            "attributes" : {  
                "type" : "string"  
            }  
        }  
    }  
}
```

例 63. 建表的同时创建LOB索引

假设Transwarp HBase中没有jsondoc:jtable_lob这张表，使用下面的JSON串会创建这张表，同时为表建LOB索引LOBP：

```
{
  "tableName": "jsondoc:jtable_lob",
  "base": {
    "families": [
      "FAMILY": "f1",
    ],
  },
  "lob": {
    "indexes": [
      {
        "INDEX_NAME": "LOBP",
        "DCOP": true,
        "INDEX_CLASS": "LOB_INDEX",
        "indexColumnInfos": [
          {
            "FAMILY": "f1",
            "QUALIFY": ""
          }
        ]
      }
    ]
  }
}
```

JSON串的复用可以帮助您轻松迁移表的扩展元信息，也就是使用一张已有表的JSON串来创建一张扩展元信息完全相同的表——您只需将 `describeInJson` 的结果输出到文件中，拷贝这个文件，再通过 `alterUseJson` 用这个文件建表，这在集群迁移中非常实用。

10.5. JSON配置迁移扩展元数据

在将表从一个集群迁移到另一个集群的时候，我们常常不仅希望能够迁移表的元数据，还希望迁移表索引的元数据。现在，通过复用表的JSON串就可以轻松地做到这一点——您只需将 `describeInJson` 的结果输出到文件中，拷贝这个文件，再通过 `alterUseJson` 用这个文件建表，便可以在新集群中建一张扩展元数据完全相同的表。下面我们演示如何迁移Transwarp HBase表的扩展元数据。

例 64. 迁移Transwarp HBase表的扩展元数据

假设我们要迁移一张名为 jsondoc:jtable 的表。

1. 将 jsondoc:jtable 的扩展元信息输出到本地的 /tmp/jtable.txt 文件中：

```
describeInJson 'jsondoc:jtable', 'true', '/tmp/jtable.txt', 'true'
```

请格外留意最后一个参数的设置，保证 Split Keys 的正常输出。

我们可以打开文件查看jtable.txt中的信息：

```
{
  "tableName" : "jsondoc:jtable",
  "base" : {
    "SPLIT_KEYS" : [ "31", "32", "33" ],
    "families" : [ {
      "FAMILY" : "f1",
      "DATA_BLOCK_ENCODING" : "NONE",
      "BLOOMFILTER" : "ROW",
      "REPLICATION_SCOPE" : "0",
      "VERSIONS" : "1",
      "COMPRESSION" : "NONE",
      "MIN VERSIONS" : "0",
      "TTL" : "2147483647",
      "KEEP_DELETED_CELLS" : "FALSE",
      "BLOCKSIZE" : "65536",
      "IN_MEMORY" : "false",
      "BLOCKCACHE" : "true"
    } ],
    "THEMIS_ENABLE" : false
  },
  "fulltextindex" : {
  },
  "globalindex" : {
    "indexs" : [ ]
  },
  "lob" : {
    "indexs" : [ ]
  },
  "localindex" : {
    "indexs" : [ ]
  }
}
```

2. 将jtable.txt文件拷贝到新集群的/tmp目录下。
3. 在新集群的Transwarp HBase命令行中执行下面指令建表：

```
alterUseJson 'jtable_copy', '/tmp/jtable.txt', 'true'
```

请再次留意最后一个参数的设置，保证Split Keys的正常输入。

4. 现在jtable_copy表已经在新集群中建好，它的扩展元数据和jtable完全相同。我们可以将它的扩展元数据打印到文件jtable_copy.txt：

```
describeInJson 'jtable_copy', '/tmp/jtable_copy.txt', 'true'
```

然后查看文件中的信息：

```
{  
    "tableName" : "jtable_copy",  
    "base" : {  
        "SPLIT_KEYS" : [ "31", "32", "33" ],  
        "families" : [ {  
            "FAMILY" : "f1",  
            "DATA_BLOCK_ENCODING" : "NONE",  
            "BLOOMFILTER" : "ROW",  
            "REPLICATION_SCOPE" : "0",  
            "VERSIONS" : "1",  
            "COMPRESSION" : "NONE",  
            "MIN VERSIONS" : "0",  
            "TTL" : "2147483647",  
            "KEEP_DELETED_CELLS" : "FALSE",  
            "BLOCKSIZE" : "65536",  
            "IN_MEMORY" : "false",  
            "BLOCKCACHE" : "true"  
        } ],  
        "THEMIS_ENABLE" : false  
    },  
    "fulltextindex" : {  
    },  
    "globalindex" : {  
        "indexes" : [ ]  
    },  
    "lob" : {  
        "indexes" : [ ]  
    },  
    "localindex" : {  
        "indexes" : [ ]  
    }  
}
```

我们发现除了表名以外，其他所有的信息和jtable.txt完全一致。

5. 扩展元数据迁移成功。

11. Transwarp HBase API使用说明

本章介绍Transwarp HBase中的常用API。完整的Apache HBase原生API请参考
<http://hbase.apache.org/apidocs/index.html>

11.1. HBaseConfiguration

作用

通过此接口配置Transwarp HBase。

常用方法签名及描述

添加Transwarp HBase配置资源

```
static org.apache.hadoop.conf.Configuration addHbaseResources(org.apache.hadoop.conf.Configuration conf)
```

利用classpath中的HbaseResource创建配置对象

```
static org.apache.hadoop.conf.Configuration create()
```

暂无描述

```
static org.apache.hadoop.conf.Configuration create(org.apache.hadoop.conf.Configuration that)
```

获取属性值，并转换为Int作为返回值

```
static int getInt(org.apache.hadoop.conf.Configuration conf, String name, String deprecatedName, int defaultValue)
```

从配置实例中获取密码

```
static String getPassword(org.apache.hadoop.conf.Configuration conf, String alias, String defPass)
```

获取属性名对应的值

```
string get(String name)
```

获取boolean类型属性值,如果其属性值不是boolean类型的,则返回默认属性值

```
String getBoolean(String name, boolean defaultValue)
```

通过属性名来设置值

```
void set(String name, String value)
```

设置boolean类型的属性值

```
void setBoolean(String name, boolean value)
```

例 65. 用 HBaseConfiguration 设置ZooKeeper配置项

```
HBaseConfiguration hc = new HBaseConfiguration();
hc.set("hbase.zookeeper.property.clientPort", "2181");
```

11.2. HBaseAdmin

作用

提供了一个接口来管理Transwarp HBase数据库的表信息。包括：创建表,删除表,列出表 项,使表有效或无效,以及添加或删除表列族成员等。

常用方法签名及描述

修改一列

```
void modifyColumn(byte[] tableName, final HColumnDescriptor descriptor)
```

删除一列

```
void deleteColumn(byte[] tableName, final String columnName)
```

向一个已经存在的表添加列

```
void addColumn(byte[] tableName, HColumnDescriptor column)
```

静态函数,查看Transwarp HBase是否处于运行状态

```
void checkHBaseAvailable(HBaseConfiguration conf)
```

创建一个表,同步操作

```
void createTable(HTableDescriptor desc)
```

删除一个已经存在的表

```
void deleteTable(byte[] tableName)
```

将表上线

```
void enableTable(byte[] tableName)
```

将表下线

```
void disableTable(byte[] tableName)
```

列出所有用户控件表项

```
HTableDescriptor[] listTables()
```

修改表的模式 (异步的操作,可能需要花费一定的时间)

```
void modifyTable(byte[] tableName, HTableDescriptor htd)
```

检查表是否存在

```
boolean tableExists(String tableName)
```

关闭region

```
void closeRegion(final String regionname, final String serverName)
```

表或者region的压缩

```
void compact(final String tableNameOrRegionName)
```

表或者region的flush

```
void flush(final byte[] tableNameOrRegionName)
```

例 66. 用 HBaseAdmin 创建、 disable 和删除表

```
HBaseAdmin admin = HBaseAdmin(HBaseConfiguration.create());
HTableDescriptor desc = new HTableDescriptor(TableName.valueOf(
    "default_table"));
HColumnDescriptor colDesc = new HColumnDescriptor("cf1");
desc.addFamily(colDesc);
// 创建表
admin.createTable(desc);
// disable 表
admin.disableTable(desc.getTableName());
// 删除表
admin.deleteTable(desc.getTableName());
```

11.3. HyperbaseAdmin

作用

用于创建和删除全局索引、全文索引和LOB索引。注意，HyperbaseAdmin 用于 创建 索引，而不生成索引。生成索引使用的API请参考[IndexRebuilder](#)。

常用方法签名及描述

创建全局索引

```
void addGlobalIndex(
    org.apache.hadoop.hbase.TableName tableName,
    org.apache.hadoop.hyperbase.SecondaryIndex index,
    byte[] indexName,
    byte[][] splitKeys,
    boolean withCompression)
throws java.io.IOException
```

删除全局索引

```
public void deleteGlobalIndex(
    org.apache.hadoop.hbase.TableName tableName,
    byte[] indexName)
throws java.io.IOException
```

例 67. 添加全局索引示例

```

Configuration conf = HBaseConfiguration.create();
HyperbaseAdmin admin = new HyperbaseAdmin(conf);
TableName tableName = TableName.valueOf("test");
//split key used to index table
byte[][][] splitKeys = null;
boolean withCompression = false;

byte[] indexName = Bytes.toBytes("IDX");
byte[] family = Bytes.toBytes("f");
byte[] q = Bytes.toBytes("q3");
byte[] q1 = Bytes.toBytes("q4");

// build index, index column is q
HyperbaseProtos.SecondaryIndex.Builder builder =
HyperbaseProtos.SecondaryIndex.newBuilder();

//set index type and properties
builder.setClassName(CombineIndex.class.getName());
builder.setUpdate(false);
builder.setDcop(false);

// index column
builder.addColumns(HyperbaseProtos.IndexedColumn.newBuilder().setColumn(
HyperbaseProtos.Column.newBuilder().
    setFamily(HBaseZeroCopyByteString.wrap(family)).
    setQualifier(HBaseZeroCopyByteString.wrap(q))).setSegmentLength(6));

// rowkey column
builder.addColumns(HyperbaseProtos.IndexedColumn.newBuilder().setColumn(
HyperbaseProtos.Column.newBuilder().
    setFamily(HBaseZeroCopyByteString.wrap(IndexedColumn.
ROWKEY_FAMILY)).
    setQualifier(HBaseZeroCopyByteString.wrap
(IndexedColumn.ROWKEY_QUALIFIER))).setSegmentLength(7));

CombineIndex index =
(CombineIndex)SecondaryIndexUtil.constructSecondaryIndexFromPb(builder.
build());

//add global index
admin.addGlobalIndex(tableName, index, indexName, splitKeys,
withCompression);

//delete global index
admin.deleteGlobalIndex(tableName, indexName);

```

添加LOB索引

```

void addLob(org.apache.hadoop.hbase.TableName tableName, org.apache.
hadoop.hyperbase.secondaryindex.SecondaryIndex lob, byte[] lobName,
boolean useCompress, int hdfsReplica)

```

删除LOB索引

```
void deleteLob(org.apache.hadoop.hbase.TableName tableName, byte[] lobName)
```

添加全文索引

```
void addFulltextIndex(org.apache.hadoop.hbase.TableName tableName,
org.apache.hadoop.hyperbase.fulltextindex.metadata.Mapping mapping)
```

删除全文索引

```
void deleteFulltextIndex(org.apache.hadoop.hbase.TableName tableName)
```

例 68. 添加全文索引示例

```
Configuration conf = HBaseConfiguration.create();
HyperbaseAdmin admin = new HyperbaseAdmin(conf);
TableName tableName = TableName.valueOf("test");

//allow update index data when data of original table is modified
boolean allowUpdate = true;

Mapping mapping = new Mapping(tableName.getNameAsString(),
allowUpdate);

mapping.addField(new Mapping.Field("f", "q1").setOptField(Mapping.
Field.TYPE, Mapping.Field.INTEGER)
.setOptField(Mapping.Field.INDEX, "not_analyzed").
setOptField(Mapping.Field.STORE, "true"));
mapping.addField(new Mapping.Field("f", "q2").setOptField(Mapping.
Field.INDEX, "not_analyzed")
.setOptField(Mapping.Field.TYPE, Mapping.Field.STRING).
setOptField(Mapping.Field.STORE, "false"));
mapping.addField(new Mapping.Field("f", "q3").setOptField(Mapping.
Field.INDEX, "not_analyzed")
.setOptField(Mapping.Field.TYPE, Mapping.Field.BYTE).
setOptField(Mapping.Field.STORE, "false"));

//add fulltext index
//Index name will be automatically generated.
admin.addFulltextIndex(tableName, mapping);

//delete fulltext index
admin.deleteFulltextIndex(tableName);
```

11.4. IndexRebuilder

作用：重建索引

常用方法签名及描述:

重建全局索引

```
//为全表重建全局索引  
void rebuildGlobalIndex(indexedTabName, Bytes.toBytes(globalIndexName))  
//为Row Key在[startKey, endKey)之间的记录建全局索引  
void rebuildGlobalIndex(indexedTabName, Bytes.toBytes(localIndexName),  
startKey, endKey)
```

重建全文索引

```
//为全表建全文索引  
void rebuildFulltextIndex(indexedTabName)  
//为Row Key在[startKey, endKey)之间的记录建全文索引  
void rebuildFulltextIndex(indexedTabName, startKey, endKey)
```

例 69. IndexRebuilder 使用示例

```

package org.apache.hadoop.hyperbase;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.util.Bytes;
import org.apache.hadoop.hyperbase.mapreduce.IndexRebuilder;

import java.io.IOException;

public class IndexRebuilderDemo {
    public static void main(String[] args) throws IOException {
        Configuration conf = HBaseConfiguration.create();
        IndexRebuilder rebuilder = new IndexRebuilder(conf);

        TableName indexedTabName = TableName.valueOf("test");
        String localIndexName = "localindex";
        String globalIndexName = "globalindex";
        byte[] startKey = Bytes.toBytes(1);
        byte[] endKey = Bytes.toBytes(100);

        //rebuild fulltext
        rebuilder.rebuildFulltextIndex(indexedTabName);
        //rebuild fulltext with range
        rebuilder.rebuildFulltextIndex(indexedTabName, startKey, endKey);

        //rebuild local index
        rebuilder.rebuildLocalIndex(indexedTabName, Bytes.
toBytes(localIndexName));
        //rebuild local index with range
        rebuilder.rebuildLocalIndex(indexedTabName, Bytes.
toBytes(localIndexName), startKey, endKey);

        //rebuild global index
        rebuilder.rebuildGlobalIndex(indexedTabName, Bytes.
toBytes(globalIndexName));
        //rebuild global index with range
        rebuilder.rebuildGlobalIndex(indexedTabName, Bytes.
toBytes(localIndexName), startKey, endKey);
    }
}

```

11.5. HTableDescriptor

作用

描述一个Transwarp HBase表的所有细节，包括列族等信息。

常用方法签名及描述

获取所有列信息

```
Collection getFamilies()
```

添加一个列族

```
void addFamily(HColumnDescriptor)
```

移除一个列族

```
HColumnDescriptor removeFamily(byte[] column)
```

获取表的名字

```
byte[] getName()
```

获取属性的值

```
byte[] getValue(byte[] key)
```

设置属性的值

```
void setValue(String key, String value)
```

使用示例请参考用 [HBaseAdmin](#) 创建、disable和删除表。

11.6. HColumnDescriptor

作用

描述了一个Transwarp HBase表的列族信息, 包括: 版本, 压缩设置等。

常用方法签名及描述

获取最大版本

```
int getMaxVersions()
```

获取最小版本

```
int getMinVersions()
```

设置最大版本

```
HColumnDescriptor setMaxVersions(int maxVersions)
```

设置最小版本

```
HColumnDescriptor setMinVersions(int minVersions)
```

获取列族名称

```
byte[] getName()
```

获取对应属性的值

```
byte[] getValue(byte[] key)
```

设置对应属性的值

```
void setValue(String key, String value)
```

使用示例请参考用 [HBaseAdmin](#) 创建、disable和删除表。

11.7. HTable

作用

用于单个表的通信, 使用这个类对表进行读写是非线程安全的。

常用方法签名及描述

释放所有的资源或挂起内部缓冲区中的更新

```
void close()
```

检查Get实例所指定的值是否存在与HTable的列中

```
Boolean exists(Get get)
```

获取指定行的某些单元格所对应的值

```
Result get(Get get)
```

获取当前给定列族的scanner实例

```
ResultScanner getScanner(byte[] family)
```

获取当前表的HTableDescriptor实例

```
HTableDescriptor getTableDescriptor()
```

获取表名

```
byte[] getTableName()
```

检查表是否在线

```
static boolean isTableEnabled(HBaseConfiguration conf, String tableName)
```

向表中添加值

```
void put(Put put)
```

获取当前表中所有region的start keys

```
byte[][][] getStartKeys()
```

获取当前表中所有region的end keys

```
byte[][][] getEndKeys()
```

例 70. 使用 HTable

```
byte[] tableName = Bytes.toBytes("tableName");
HTable table = new HTable(conf, Bytes.toBytes(tableName));
byte[] tableName = table.getTableName();
```

11.8. Put

作用

写数据到一行中。

常用方法签名及描述

将指定的列和对应的值添加到Put实例中

```
Put add(byte[] family, byte[] qualifier, byte[] value)
```

将指定的列和对应的值及时间戳添加到Put实例中

```
Put add(byte[] family, byte[] qualifier, long ts, byte[] value)
```

获取Put实例的行

```
byte[] getRow()
```

获取Put实例的时间戳

```
long getTimeStamp()
```

检查familyMap是否为空

```
boolean isEmpty()
```

设置Put实例的时间戳

```
Put setTimeStamp(long timeStamp)
```

例 71. 使用 Put

```
byte[] row = new byte[] {'r'};
byte[] tableName = Bytes.toBytes("tableName");
HTable table = new HTable(conf, tableName);
Put p = new Put(row);
p.add(family, qualifier, value);
table.put(p);
```

11.9. Get

作用

获取单行的相关信息。

常用方法签名及描述

获取指定列族和qualifier对应的列

```
Get.addColumn(byte[] family, byte[] qualifier)
```

通过指定的列族获取其对应列的所有列

```
Get addFamily(byte[] family)
```

获取指定取件的列的版本号

```
Get setTimeRange(long minStamp, long maxStamp)
```

当执行Get操作时设置服务器端的过滤器

```
Get setFilter(Filter filter)
```

例 72. 使用 Get

```
byte[] row = new byte[] {'r'};
byte[] tableName = Bytes.toBytes("tableName");
HTable table = new HTable(conf, tableName);
Get g = new Get(row);
Result result = table.get(g);
```

11.10. Scan

作用

扫描指定row key范围内的行。

常用方法签名及描述

设置scan的开始rowkey

```
Scan setStartRow(byte[] startRow)
```

设置scan结束rowkey

```
Scan setStopRow(byte[] stopRow)
```

指定filter

```
Scan setFilter(Filter filter)
```

获取指定的列

```
Scan addColumn(byte[] family, byte[] qualifier)
```

获取所有列族下的列

Scan addFamily(byte[] family)

例 73. 使用 Scan

```
SingleColumnValueFilter filter = new SingleColumnValueFilter(
    cf,
    column,
    CompareOp.EQUAL,
    Bytes.toBytes("my value")
);
scan.setFilter(filter);

byte [] tableName = Bytes.toBytes("tableName");
Configuration conf = HBaseConfiguration.create();
HTable table = new HTable(conf, tableName);

try {
    rs = table.getScanner(scan);
    for (Result r : rs) {
        for (KeyValue kv : r.list()) {
            System.out.println("row:" + Bytes.toString(kv.getRow()));
        }
    }
} finally {
    rs.close();
}
```

11.11. Result

作用

存储Get或者Scan操作后获取表的单行值。

常用方法签名及描述

判断结果集是否存在

Boolean getExists()

获得最新版本的列值

byte[] getValue(byte[] family, byte[] qualifier)

判断结果集合是否为空

boolean isEmpty()

转换为Cell[] 数组返回

```
Cell[] rawCells()
```

扫描一个cell

```
boolean advance()
```

使用示例请参考[使用 Scan](#)。

11.12. Object Store使用方法

默认情况下，Transwarp HBase用相同的方法处理结构化和非结构化的数据。但是，由于图片等非结构化数据非常大，在向Transwarp HBase导入时会让Region大小增长迅速，频繁触发Region的Split和Compaction，在一定程度上卡住客户端的写入，影响Transwarp HBase的插入性能。因此，对非结构化的大对象采用不同方式处理，在插入时降低Region的Split和Compaction频率，提高插入性能，Object Store就是为了解决这个问题而设计的。我们提供了API以及JSON配置方法来使用Object Store。本章将展示Object Store API的示例代码。关于JSON配置操作请参考[Transwarp HBase JSON配置使用说明](#)。

注意

- 在使用Object Store建表时一定要预分Region，确保每个Region最多 **500G** 数据。
- TDH4.3之后的版本中，您需要在hbase-site.xml的配置项 **hbase.coprocessor.region.classes** 中额外添加一个类 **org.apache.hadoop.hyperbase.coprocessor.LobCoprocessor** 来保证Object store的正常使用。
- Object Store是对列族生效，请将Object Store放在同一个列族中，其他列放在另一个列族里面。
- 如果出现报错信息如“Keyvalue size too large”，可将 **hbase.client.keyvalue.maxsize** 的值调整到所需大小，具体解决方法和原因参见[Transwarp HBase常见问题](#)。

例 74. TDH4.2及之前版本的Object Store API

```
package io.transwarp;

import static org.junit.Assert.assertTrue;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.concurrent.TimeUnit;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.*;
import org.apache.hadoop.hbase.client.Get;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.protobuf.generated.HyperbaseProtos;
```

```

-----[import org.apache.hadoop.hbase.util.Bytes;
import org.apache.hadoop.hyperbase.client.HyperbaseAdmin;
import org.apache.hadoop.hyperbase.metadata.HyperbaseMetadata;
import org.apache.hadoop.hyperbase.secondaryindex.IndexedColumn;
import org.apache.hadoop.hyperbase.secondaryindex.LOBIndex;

public class TestLOB4_3 {

    protected static HyperbaseAdmin admin = null;
    protected static Configuration conf = null;
    static {
        conf = HBaseConfiguration.create();
        //改成对应的集群上面的配置!
        conf.set("hbase.zookeeper.quorum", "transwarp-perf2,transwarp-
perf1,transwarp-perf3");
        conf.set("zookeeper.znode.parent", "/hyperbase1");
        conf.set("hbase.zookeeper.property.clientPort", "2181");
        try {
            admin = new HyperbaseAdmin(conf);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    public static void main(String[] args) throws Exception {
        testLOBGet();
    }

    public static void testLOBGet() throws Exception {
        byte[] row = Bytes.toBytes("rowkey01");
        byte[] tableName = Bytes
            .toBytes("SIMPLE_TEST_PUT_SCAN" + System.nanoTime());
        byte[] indexName = Bytes.toBytes("IDX");
        byte[] family1 = Bytes.toBytes("f1");
        byte[] family2 = Bytes.toBytes("f2");
        String path = "/tmp/file1";
        createTable(TableName.valueOf(tableName), family1, family2);
        addLOB(tableName, family1, indexName);
        byte[] value01 = getFileBytes(path);

        HTable htable = new HTable(conf, tableName);

        Put put = new Put(row);
        put.add(family1, Bytes.toBytes("q1"), value01);
        htable.put(put);
        htable.flushCommits();
        TimeUnit.SECONDS.sleep(1);

        Get get = new Get(row);
        Result rs = htable.get(get);
        CellScanner cs = rs.cellScanner();
        while(cs.advance()){
            assertTrue(Bytes.equals(CellUtil.cloneValue(cs.current()), value01));
            System.out.println(Bytes.toString(CellUtil.cloneValue(cs.
current())));
        }
        htable.close();
    }
}
-----
```

```

----- admin.deleteTable(TableName.valueOf(tableName)); -----
}

public static byte[] getFileBytes(String path) throws IOException {
    FileInputStream fis = new FileInputStream(new File(path)); // 新建一个FileInputStream对象
    byte[] b = new byte[fis.available()]; // 新建一个字节数组
    fis.read(b); // 将文件中的内容读取到字节数组中
    fis.close();
    return b;
}

public static void createTable(TableName tableName, byte[]... families) throws Exception {
    HTableDescriptor tableDescriptor = new HTableDescriptor(tableName);
    for (byte[] family : families) {
        tableDescriptor.addFamily(new HColumnDescriptor(family));
    }
    // create table succ
    admin.createTable(tableDescriptor, null);
    HyperbaseMetadata metadata = admin.getTableMetadata(tableName);
    assertTrue(metadata != null);
    // check metadata
    assertTrue(metadata.getFulltextMetadata() == null);
    assertTrue(metadata.getGlobalIndexes().isEmpty());
    assertTrue(metadata.getLocalIndexes().isEmpty());
    assertTrue(metadata.getLobs().isEmpty());
    assertTrue(metadata.isTransactionTable() == false);
}

public static void addLOB(byte[] tableName, byte[] family, byte[] LOBFamily) throws IOException {
    HyperbaseProtos.SecondaryIndex.Builder LOBBuilder =
    HyperbaseProtos.SecondaryIndex
        .newBuilder();
    LOBBuilder.setClassName(LOBIndex.class.getName());
    LOBBuilder.setUpdate(true);
    LOBBuilder.setDcop(true);
    IndexedColumn column = new IndexedColumn(family, Bytes.toBytes("q1"));
    LOBBuilder.addColumns(column.toPb());
    admin.addLob(TableName.valueOf(tableName), new LOBIndex(LOBBuilder.build()), LOBFamily, false, 1);
}
}

```

例 75. TDH4.3 及之后版本的 Object Store API

```

package io.transwarp;

import static org.junit.Assert.assertTrue;

import java.io.File;
-----
```

```

-----[import java.io.FileInputStream;
import java.io.IOException;
import java.util.concurrent.TimeUnit;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.*;
import org.apache.hadoop.hbase.client.Get;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.protobuf.generated.HyperbaseProtos;
import org.apache.hadoop.hbase.util.Bytes;
import org.apache.hadoop.hyperbase.client.HyperbaseAdmin;
import org.apache.hadoop.hyperbase.metadata.HyperbaseMetadata;
import org.apache.hadoop.hyperbase.secondaryindex.IndexedColumn;
import org.apache.hadoop.hyperbase.secondaryindex.LOBIndex;

public class TestLOB4_3 {

    protected static HyperbaseAdmin admin = null;
    protected static Configuration conf = null;
    static {
        conf = HBaseConfiguration.create();
        //改成对应的集群上面的配置!
        conf.set("hbase.zookeeper.quorum", "transwarp-perf2,transwarp-
perf1,transwarp-perf3");
        conf.set("zookeeper.znode.parent", "/hyperbase1");
        conf.set("hbase.zookeeper.property.clientPort", "2181");
        try {
            admin = new HyperbaseAdmin(conf);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    public static void main(String[] args) throws Exception {
        testLOBGet();
    }

    public static void testLOBGet() throws Exception {
        byte[] row = Bytes.toBytes("rowkey01");
        byte[] tableName = Bytes
            .toBytes("SIMPLE_TEST_PUT_SCAN" + System.nanoTime());
        byte[] indexName = Bytes.toBytes("IDX");
        byte[] family1 = Bytes.toBytes("f1");
        byte[] family2 = Bytes.toBytes("f2");
        String path = "/tmp/file1";
        createTable(TableName.valueOf(tableName), family1, family2);
        addLOB(tableName, family1, indexName);
        byte[] value01 = getFileBytes(path);

        HTable htable = new HTable(conf, tableName);

        Put put = new Put(row);
        put.add(family1, Bytes.toBytes("q1"), value01);
        htable.put(put);
        htable.flushCommits();
        TimeUnit.SECONDS.sleep(1);
    }
}
-----
```

```

Get get = new Get(row);
Result rs = htable.get(get);
CellScanner cs = rs.cellScanner();
while(cs.advance()){
    assertTrue(Bytes.equals(CellUtil.cloneValue(cs.current()),
value01));
    System.out.println(Bytes.toString(CellUtil.cloneValue(cs.
current())));
}
htable.close();
admin.deleteTable(TableName.valueOf(tableName));
}

public static byte[] getFileBytes(String path) throws IOException {
    FileInputStream fis = new FileInputStream(new File(path));//
新建一个FileInputStream对象
    byte[] b = new byte[fis.available()];// 新建一个字节数组
    fis.read(b);// 将文件中的内容读取到字节数组中
    fis.close();
    return b;
}

public static void createTable(TableName tableName, byte[]...
families) throws Exception {
    HTableDescriptor tableDescriptor = new HTableDescriptor(tableName);
    for (byte[] family : families) {
        tableDescriptor.addFamily(new HColumnDescriptor(family));
    }
    // 注意, object store一定要预分配region, 每个region最好不要超过500G的数据。
    byte[][] splitKeys = new byte[10][];
    for (int i = 0; i < 100; i++) {
        splitKeys[i] = Bytes.toBytes("rowkey" + i);
    }
    // create table succ
    admin.createTable(tableDescriptor, null, splitKeys);
    HyperbaseMetadata metadata = admin.getTableMetadata(tableName);
    assertTrue(metadata != null);
    // check metadata
    assertTrue(metadata.getFulltextMetadata() == null);
    assertTrue(metadata.getGlobalIndexes().isEmpty());
    assertTrue(metadata.getLocalIndexes().isEmpty());
    assertTrue(metadata.getLobs().isEmpty());
    assertTrue(metadata.isTransactionTable() == false);
}

public static void addLOB(byte[] tableName, byte[] family, byte[]
LOBFamily) throws
    IOException {
    HyperbaseProtos.SecondaryIndex.Builder LOBBuilder =
HyperbaseProtos.SecondaryIndex
        .newBuilder();
    LOBBuilder.setClassName(LOBIndex.class.getName());
    LOBBuilder.setUpdate(true);
    LOBBuilder.setDcop(true);
    IndexedColumn column = new IndexedColumn(family, Bytes.toBytes(
    "q1"));
    LOBBuilder.addColumn(column.toPb());
    admin.addLob(TableName.valueOf(tableName), new LOBIndex

```

```
    - (LOBBuilder build()); LOBFamily, false, 1);  
}
```

12. Transwarp HBase SQL使用说明

本章介绍如何使用SQL与Transwarp HBase交互。我们将重点介绍Transwarp HBase SQL中的DDL（包括表DDL和索引DDL）以及DML中的 **INSERT/UPDATE/DELETE**。在DML的 **SELECT** 语句方面，Transwarp HBase SQL独有通过指定索引来加速查询的功能，除此之外，Transwarp HBase SQL的 **SELECT** 语句和Inceptor SQL的 **SELECT** 语句用法完全相同，如 **WHERE**、**GROUP BY**、**JOIN**、子查询和集合运算等，这里将不赘述，请参考《Transwarp Data Hub Inceptor 使用手册》。

12.1. DESC FORMATTED

DESC FORMATTED 可以用于查看映射表的元数据，包括列名、数据类型、索引信息等等。

语法：查看映射表的元数据

```
DESC FORMATTED <table>;
```

12.2. Transwarp HBase SQL DDL

Transwarp HBase SQL中的DDL(Data Definition Language)包含：

- 创建/编辑/删除/清空表：**CREATE/ALTER/DROP/TRUNCATE TABLE**

12.2.1. 建表：CREATE TABLE

建映射表语法

```
CREATE [EXTERNAL] TABLE <table> ( ①
    <key> <key_data_type>,
    <column1> <column1_data_type>,
    <column2> <column2_data_type>,
    ...
)
STORED BY
'org.apache.hadoop.hive.hbase.HBaseStorageHandler|io.transwarp.hyperdrive.
HyperdriveStorageHandler' ②
WITH SERDEPROPERTIES('hbase.columns.mapping'
=':key,<column1>,<column2>,...') ③
[TBLPROPERTIES('hbase.table.name'='<table_in_hbase>')]; ④
```

① **[EXTERNAL]** 为外表选项，加上 **EXTERNAL** 即建外表。Inceptor Engine对外表没有所有权，通过Inceptor Engine删除表时，仅仅删除表在Inceptor Engine中的元数据，不会删除原表在Transwarp HBase中的数据。对于非外表，Inceptor Engine有所有权，所以在删除时会将元数据和数据都删除。

② 指定表使用的Storage Handler：建HBase表，使用 '**org.apache.hadoop.hive.hbase.HBaseStorageHandler**'；建Hyperdrive表使用 '**io.transwarp.hyperdrive.HyperdriveStorageHandler**'。注意选择的Storage Handler要放在引号

中。

- ③ 指定映射表和实际Transwarp HBase表的列映射关系。
- ④ 可选项，指定映射表对应的Transwarp HBase表的名称。

SQL建表时发生了什么

如我们前面提到过的，SQL操作的是Transwarp HBase表的映射表。SQL建表时，会在Inceptor Engine建一张二维表，并且建立和Transwarp HBase中表的映射关系。SQL建的表可以是Transwarp HBase中一张已有表的映射表，也可以是一张全新的表。

- 当SQL建的表Transwarp HBase中已有原表，建表语句仅仅建立新建表和Transwarp HBase原表之间的列名以及表名映射关系。
- 如果SQL建的表在Transwarp HBase中没有原表，Inceptor Engine会在Transwarp HBase中按照建表语句的 **TBLPROPERTIES** 中指定的表名和 **SERDEPROPERTIES** 中指定的列名新建一张Transwarp HBase表，同时建立映射关系。

下面分别详细介绍该语法在建HBase映射表和建Hyperdrive映射表的具体使用。另外，SQL建Hyperdrive表有更为简洁的建表语法（注意，只有Hyperdrive表有这个简洁建表语法，HBase表必须使用**建映射表语法**建表。）。

12.2.1.1. 建HBase映射表

语法：建HBase映射表

```
CREATE [EXTERNAL] TABLE <table> (
    <row_key_column> <DATA_TYPE>,
    <column> <DATA_TYPE>,
    <column> <DATA_TYPE>,
    ...
)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler' ①
    WITH SERDEPROPERTIES (
        "hbase.columns.mapping" = ":key, <cf>:<cq>, <cf>:<cq>, ...") ②
    [TBLPROPERTIES ("hbase.table.name" = "<hbase_table>")]; ③
```

- ① HBase映射表使用HBaseStorageHandler，**org.apache.hadoop.hive.hbase.HBaseStorageHandler**为它的类名。
- ② **hbase.columns.mapping** 属性定义映射表和HBase表之间的列对应关系。映射表的第一列必须对应HBase表中的Row Key。映射表中剩余的列将是对应HBase表中的 **<cf>:<cq> (<column_family>:<columnQualifier>)** 组合。
- ③ 可选项，**TBLPROPERTIES** 中的 **hbase.table.name** 属性定义映射表对应的HBase表的表名。

例 76. 建HBase映射表

参考[创建bank_info的HBase映射表](#)。

12.2.1.2. 建Hyperdrive映射表

建Hyperdrive映射表有如下两种方式：

简化建表方式

```
CREATE TABLE <table> (
    <key> <key_data_type>,
    <column1> <column1_data_type>,
    <column2> <column2_data_type>,
    ...
)
STORED AS HYPERDRIVE;
```

普通建表方式

```
CREATE [EXTERNAL] TABLE <table> (
    <key> <key_data_type>,
    <column1> <column1_data_type>,
    <column2> <column2_data_type>,
    ...
)
STORED BY 'io.transwarp.hyperdrive.HyperdriveStorageHandler' ①
    WITH SERDEPROPERTIES('hbase.columns.mapping'
    ='<key>,<hcolumn1>,<hcolumn2>,...') ②
    [TBLPROPERTIES('hbase.table.name'='<hyperdrive_table>')]; ③
```

- ① 指定使用 'io.transwarp.hyperdrive.HyperdriveStorageHandler' 作为storage handler。
- ② 指定映射表和对应Hyperdrive表的列的映射关系。
- ③ 指定映射表对应的Hyperdrive表的表名。

[简化建表方式](#)步骤简单，易于记忆，不易出错，所以我们 鼓励用户尽量使用[简化建表方式](#)。但是[简化建表方式](#)会使用默认的方式建立映射表和Hyperdrive表之间的对应关系：映射表对应同名Hyperdrive表，映射表中的列对应列族为、column qualifier和映射表列名相同的列（例如，如果映射表中的列名为a，Hyperdrive表中的列名为f:a）。所以，需要自定义对应关系的情况下还是需要选择[普通建表方式](#)。例如，当和映射表同名的Hyperdrive表已经存在，那么就需要用 **TBLPROPERTIES('hbase.table.name'='<hyperdrive_table>')** 来指定另一个表名。



Hyperdrive中维护了自己的类型信息编码方式，只能通过特定的接口来对里面的数据进行处理。我们推荐只用SQL和Hyperdrive表进行交互，也最好不要创建Hyperdrive外表。如果您会有应用不使用SQL处理数据，请选择HBase表。

建Hyperdrive映射表时STRUCT列中STRING字段的长度指定

如果Hyperdrive映射表中有STRUCT类型的列，且该列中有STRING类型的字段，那么在建表时，您必须指定该STRING类型字段的长度。例如：

指定STRUCT列中STRING字段的长度

```
CREATE TABLE test(
    key STRING,
    name STRING,
    info STRUCT<
        add:STRING LENGTH 20, ①
        age:INT,
        height:DOUBLE
    >,
    balance DOUBLE
)
STORED AS HYPERDRIVE;
```

①用 LENGTH 20 指定info中add字段的长度。

如果执行了下面的命令，Inceptor Engine会报错。

不指定STRUCT列中STRING字段的长度

```
CREATE TABLE test(
    key STRING,
    name STRING,
    info STRUCT<
        add:STRING, ①
        age:INT,
        height:DOUBLE
    >,
    balance DOUBLE
)
STORED AS HYPERDRIVE;
```

①未指定info中add字段的长度。

例 77. 用简化方式建Hyperdrive表

```
CREATE TABLE test(
    key STRING,
    name STRING,
    info STRUCT<
        add:STRING LENGTH 20,
        age:INT,
        height:DOUBLE
    >,
    balance DOUBLE
)
STORED AS HYPERDRIVE;
```

例 78. 用普通方式建Hyperdrive表

```
CREATE TABLE test1(
    key STRING,
    name STRING,
    info STRUCT<
        add:STRING LENGTH 20,
        age:INT,
        height:DOUBLE
    >,
    balance DOUBLE
)
STORED BY 'io.transwarp.hyperdrive.HyperdriveStorageHandler'
    WITH SERDEPROPERTIES('hbase.columns.mapping'=':key,f:n,f:i,f:b')
    TBLPROPERTIES('hbase.table.name'='test1');
```

12.2.2. 编辑表: ALTER TABLE

HBase表 仅支持使用 **ALTER TABLE** 添加列: **ALTER TABLE ... ADD COLUMNS**。

Hyperdrive表则支持下列 **ALTER TABLE** 语法:

- 添加列: **ALTER TABLE ... ADD COLUMNS**
- 重命名列: **ALTER TABLE ... CHANGE COLUMN**
- 删除列: **ALTER TABLE ... DELETE COLUMNS**

本章我们对这些语法进行详细介绍。

添加列的语法

```
ALTER TABLE <table> ADD COLUMNS (<column> <datatype>, <column>
<datatype>, ...);
```

例 79. 添加列

```
ALTER TABLE test ADD COLUMNS (dept STRING, obd DATE);
```

表示对表test添加了两列信息dept、obd，分别为STRING、DATE类型。

语法：重命名列（不支持对HBase表使用）

```
ALTER TABLE <table> CHANGE COLUMN <old_column_name> <new_column_name> <data_type>;
```

注意，重命名列的时候必须要重新声明一次列的类型，并且您需要确保这次的声明和列原来的数据类型相同。

例 80. 重命名列

```
ALTER TABLE test CHANGE COLUMN obd on_board_date DATE;
```

表示将test表的obd列重命名为on_board_date。

语法：删除列（不支持对HBase表使用）

```
ALTER TABLE <table> DELETE COLUMNS (<column1>, <column2>, ...);
```

例 81. 删除列

```
ALTER TABLE test DELETE COLUMNS (dept, on_board_date);
```

表示删除test表的dept、on_board_date两列。

12.2.3. 删除表：DROP TABLE

HBase映射表和Hyperdrive映射表使用相同的语法删除：

语法：删除表

```
DROP TABLE <table>;
```

12.2.4. 清空表：TRUNCATE

HBase映射表和Hyperdrive映射表使用相同的语法清空：

清空表中数据的语法

```
TRUNCATE TABLE <table>;
```

将表中数据全部删除，将表变为空表（不删除元数据）。

例 82. 清空表中数据

```
TRUNCATE TABLE sub_test;
```

12.3. Transwarp HBase SQL索引DDL

HBase映射表和Hyperdrive映射表的索引都可以使用SQL创建和删除。

12.3.1. 建索引：CREATE INDEX

HBase映射表和Hyperdrive映射表的建索引方式不同，下面分别介绍。

12.3.1.1. 为HBase映射表建索引



HBase映射表不支持本地索引，仅支持全局索引和全文索引。

语法：为HBase映射表建全局索引

```
CREATE GLOBAL INDEX <index_name> ON <table> (
  <column1>(<length1>|SEGMENT LENGTH <length1>),
  [<column2>(<length2>|SEGMENT LENGTH <length2>)],
  ...
);
```

<column1>, <column2> ... 为建索引所用的列；<length1>, <length2> 为对应列在索引词条中所占字段的长度。

创建HBase映射表全局索引时要指定字段长度

创建全局索引时（无论是STRING类型还是其他类型），必须指定该列在索引词条中所占字段的长度。指定的方法为 <column> SEGMENT LENGTH <length> 或者 <column>(<length>)。

注意，建HBase映射表的全文索引时则无需指定字段长度。

例 83. 为HBase映射表创建全局索引

```
CREATE GLOBAL INDEX i_bi_global ON hbase_index_demo (i(4), bi(8));
CREATE GLOBAL INDEX str_i_dl_global ON hbase_index_demo (str(16), dl(8));
```

语法：为HBase映射表创建全文索引

```
CREATE FULLTEXT INDEX ON <table> ( ①
    <column1> [DOCVALUES[TRUE|FALSE]], ②
    [<column2> [DOCVALUES[TRUE|FALSE]]], ...
SHARD NUM <n>; ③
```

① 一张表只能有一个全文索引，所以建全文索引无须指定索引名。

② **DOCVALUES** 是一个优化查询的开关，默认是打开（TRUE）的。

③ **SHARD NUM** 指定全文索引的分片数。

全文索引的分片数是不可变的，只能在创建时指定，这里需要用户预估计索引数据的量，一个SHARD上的数据不要超过25GB。超过25GB可能会有比较严重的性能问题。DOCVALUES默认是TRUE，请使用默认值。

例 84. 为HBase映射表创建全文索引

```
CREATE FULLTEXT INDEX ON hbase_index_demo(i, bi, dl, str) SHARD NUM 1;
```

12.3.1.2. 为Hyperdrive映射表建索引

语法：为Hyperdrive映射表创建全局和本地索引

```
CREATE (GLOBAL|LOCAL) INDEX <index_name> ON <table> (
    <column1>(<length1>|SEGMENT LENGTH <length1>),
    [<column2>(<length2>|SEGMENT LENGTH <length2>)],
    ...);
```

这里，**<column1>, <column2> ...** 为建索引所用的列。

用STRING类型列创建全局和本地索引时要指定字段长度

当使用STRING类型的列建索引时，必须指定该列在索引词条中所占字段的长度。指定的方法为 **<column> SEGMENT LENGTH <length>** 或者 **<column>(<length>)**。

注意，建全文索引无需指定STRING类型的字段长度。

例 85. 为Hyperdrive映射表创建全局索引

```
CREATE GLOBAL INDEX name_global_index ON test(name SEGMENT LENGTH 8);
①
CREATE GLOBAL INDEX name_balance_global_index ON test (name(8),
balance); ②
```

① 对test表创建名为name_global_index的索引，涉及字段为name。SEGMENT LENGTH 8 指定name在索引词条中所占长度为8。

② name SEGMENT LENGTH 8 可以简写为 name(8)，两种写法是等价的。

全局索引的实现方式是在Transwarp HBase中创建一张索引表用来加快查询，一般适用于一些短平快的查询。

应该如何设置 SEGMENT LENGTH?

一个字段的 SEGMENT LENGTH 的设定最好大于等于字段的实际长度，这样设置可以最优化查询性能。SEGMENT LENGTH 小于字段的实际长度可能会影响查询性能，但是不会影响查询的正确性。

例如，如果我们用身份证号id建索引，那么id的 SEGMENT LENGTH 可以设置为18。

例 86. 为Hyperdrive映射表创建本地索引

```
CREATE LOCAL INDEX name_balance_local_index ON test(name(8), balance);
```

具体使用方式和全局索引类似，除了关键字LOCAL。本地索引的实现方式是在Transwarp HBase表中添加一个索引列，所有的数据都在各自的region内部，索引数据与原始数据的查询不需要网络开销。一般适合于一些过滤性很高但返回结果比较多的查询。

语法：为Hyperdrive映射表创建全文索引

```
CREATE FULLTEXT INDEX ON <table> ( ①
<column1> [DOCVALUES[TRUE|FALSE]], ②
[<column2> [DOCVALUES[TRUE|FALSE]]], ...
SHARD NUM <n>; ③
```

① 一张表只能有一个全文索引，所以建全文索引无须指定索引名。

② DOCVALUES 是一个优化查询的开关，默认是打开 (TRUE) 的。

③ SHARD NUM 指定全文索引的分片数。

全文索引的分片数是不可变的，只能在创建时指定，这里需要用户预估计索引数据的量，一个SHARD上的数据不要超过25GB。超过25GB可能会有比较严重的性能问题。DOCVALUES默认是TRUE，请使用默认值。

例 87. 为Hyperdrive映射表创建全文索引

```
CREATE FULLTEXT INDEX ON test (name , balance) SHARD NUM 1;
```

这里表示对test表的name、balance字段创建全文索引，1个分片。

12.3.2. 删除索引: `DROP INDEX`

语法: **删除全局索引和本地索引**

```
DROP INDEX [IF EXISTS] <index_name> ON <table>;
```

例 88. 删除全局索引

```
DROP INDEX name_balance_global_index ON test;
```

例 89. 删除本地索引

```
DROP INDEX name_balance_local_index ON test;
```

语法: **删除全文索引**

```
DROP FULLTEXT INDEX [IF EXISTS] ON <table>;
```

例 90. 删除全文索引

```
DROP FULLTEXT INDEX ON test;
```

12.3.3. 生成索引

目前Transwarp HBase不支持使用SQL生成索引，您可以从Transwarp HBase Shell中执行 `rebuild` 指令来生成索引，请参考：[为全表生成全局索引](#)，[生成本地索引](#)和[生成全文索引](#)。

12.4. Transwarp HBase SQL DML

Transwarp HBase SQL中的DML（Data Manipulation Language）包含插入数据（`INSERT`），更新数据（`UPDATE`）和删除表中记录（`DELETE`）。DML对HBase表和Hyperdrive表的使用方法相同，下面将两者统

称Transwarp HBase表，不做区分。

12.4.1. 插入数据：INSERT

Transwarp HBase SQL支持向表中单条插入数据或者批量插入查询结果。

语法：单条插入（`INSERT INTO TABLE ... VALUES...`）

```
INSERT INTO TABLE <table> [(<column1>, <column2>, ...) ] VALUES (<value1>,
<value2>, ...);
```

例 91. 单条插入

向用简化方式建Hyperdrive表中建的test表插入单条数据：

```
INSERT INTO TABLE test VALUES ('1', 'Alice', NAMED_STRUCT('add', '481
Guiping Road', 'age', 28, 'height', 1.57), 10000.0);
INSERT INTO TABLE test VALUES ('2', 'Bob', NAMED_STRUCT('add', '111 Pubei
Road', 'age', 25, 'height', 1.75), 20000.0);
INSERT INTO TABLE test VALUES ('3', 'Carol', NAMED_STRUCT('add', '52
Quanzhou Road', 'age', 29, 'height', 1.66), 30000.0);
INSERT INTO TABLE test (key, name, balance) VALUES ('4', 'Derek',
40000.0);
```

语法：批量插入查询结果（`INSERT INTO TABLE ... SELECT`）

```
INSERT INTO TABLE <table> SELECT <select_statement> FROM <src>;
```

例 92. 批量插入查询结果

建一张表sub_test：

```
CREATE TABLE sub_test(key STRING, name STRING) STORED AS HYPERDRIVE;
```

向sub_test中插入test中的key和name两列：

```
INSERT INTO TABLE sub_test SELECT t.key, t.name FROM test t;
```

12.4.2. 更新表：UPDATE

语法：更新表

```
UPDATE <table> SET <column> = <value> WHERE <filter_conditions>;
```

更新满足过滤条件 `filter_conditions` 的记录。

例 93. 更新表

```
UPDATE test SET balance = 50000.0 WHERE name = 'Carol';
```

例 94. UPDATE 条件中嵌套关联子查询

Transwarp HBase SQL支持在 `UPDATE` 条件中嵌套关联子查询这样的复杂语句:

```
UPDATE hd_table a SET (sv1, sv3, sv2) =
(SELECT 'updatesubquery', 0, b.sv2
FROM (SELECT * FROM hd_table) b WHERE a.key.kchar = b.key.kchar AND
b.key.kchar = 'clv');
```

12.4.3. 删除记录: DELETE

语法: 删除记录

```
DELETE FROM <table> WHERE <filter_conditions>;
```

删除满足过滤条件 `filter_conditions` 的记录。

例 95. 删除记录

```
DELETE FROM test WHERE key = '4';
```

12.4.4. 查询: SELECT

Transwarp HBase中的索引可以让带 `WHERE` 过滤的查询更加高效。Transwarp HBase SQL中提供了一系列独有的利用索引的语法，可以让用户在使用SQL和Transwarp HBase交互时能够利用索引。除此之外，Transwarp HBase SQL中的查询语法和Inceptor SQL相同，请参考《Transwarp Data Hub Inceptor 使用手册》。

使用SQL交互时，Inceptor Engine的优化器有一套自动选择索引的机制，可以为HBase表选择优化器认为合适的全局索引（注意，优化器不能为HBase表选择全文索引，HBase全文索引的利用必须通过 `CONTAINS` 函数）和为Hyperdrive表选择合适的全局、本地和全文索引。Transwarp HBase SQL中提供的索引指定语法则可以让用户自定义想要使用的索引。Transwarp HBase SQL中对HBase映射表和Hyperdrive映射表指定索引的方法有较大区别，总结如下：

- HBase映射表：
 - 使用提示指定全局索引，不指定的情况下Inceptor Engine的优化器会自动选择合适的全局索引。
 - 必须使用 `CONTAINS` 利用全文索引，不使用则 无法利用全文索引。

- Hyperdrive映射表：

使用提示指定全局、本地或者全文索引；不指定的情况下Inceptor Engine的优化器会自动选择合适的全局、本地或全文索引。

下面我们分别详细介绍对两种表指定索引的语法。

12.4.4.1. 指定HBase映射表的索引

指定HBase映射表的全局索引

语法：使用提示指定HBase映射表的全局索引

```
SELECT /*+USE_INDEX(<table_alias> USING <index_name>)*/ ... ①
  FROM <table> <table_alias> ②
 WHERE <filter_conditions>;
```

① `/*+USE_INDEX(<table_alias> USING <index_name>)*/` 为指定索引的提示，在提示内必须使用表的化名 `<table_alias>`。

② 必须为表起化名。

语法：不指定任何索引做查询（明确放弃在查询中使用任何索引）

```
SELECT /*+USE_INDEX(<table_alias> USING NOT_USE_INDEX)*/ ... FROM <table>
<table_alias> WHERE <filter_conditions>;
```

HBase表全文检索（利用HBase表的全文索引）

Transwarp HBase表的全文索引通过 `CONTAINS` 函数使用，目前支持精确匹配，前缀查询，模糊查询和范围查询四种语义，支持检索条件的逻辑组合。下面用一个例子解释其使用方法。



Hyperdrive表兼容 `CONTAINS` 语法，但是不建议这样使用，而且对Hyperdrive表使用 `CONTAINS` 时 `CONTAINS` 内不能有 `and`, `or` 等逻辑运算符。使用Hyperdrive表全文索引的方法请参考[语法：指定Hyperdrive映射表的全文索引](#)。

先建一张HBase表，并对其建立全文索引：

例 96. 建HBase表并建全文索引

```
DROP TABLE IF EXISTS hbase_index_demo;
--建表
CREATE TABLE hbase_index_demo(key INT, dt INT, hphm STRING)
  STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
  WITH SERDEPROPERTIES ('hbase.columns.mapping'
= ':key#b,dt:int#b,cf:val');

--建全文索引
CREATE FULLTEXT INDEX ON hbase_index_demo(dt, hphm) SHARD NUM 1;
```



更多关于HBase表全文索引操作，请参考[建索引：CREATE INDEX](#)。

语法：对HBase映射表指定全文索引（HBase表全文检索）

```
SELECT ... FROM <table>
  WHERE CONTAINS(<column>, "<fulltext_search_query>") ①
    [AND|OR CONTAINS(<column>, "<fulltext_search_query>") AND|OR CONTAINS
    (<column>, "<fulltext_search_query>") ...]; ②
```

① **CONTAINS** 函数需要两个参数：列名 **<column>** 和全文检索查询 "**<fulltext_search_query>**"（注意，检索条件要放在双引号中）。

② **CONTAINS** 中包含全文检索条件，一次查询中可以使用多个 **CONTAINS** 表，**CONTAINS** 之间用 **AND** 或者 **OR** 连接。

CONTAINS 中的全文检索条件

CONTAINS 函数中的全文检索条件 "**<fulltext_search_query>**" 需要有如下形式：

```
"<operator> '<search_contents>' [and|or <operator> '<search_contents>'
and|or <operator> '<search_contents>' ...]"
```

其中，**<operator>** 为全文检索运算符，**<search_contents>** 为检索内容（注意要放在单引号中）。一个全文检索条件可以由多个 **<operator> <search_contents>** 组成，之间用 **and** 或 **or** 连接。

全文检索操作符 **<OPERATOR>** 包括：

- **term**: 精确匹配)
- **prefix**: 前缀查询
- **wildcard**: 模糊匹配
- **range**: 范围查询)
- **regexp**: 正则表达式
- **in**: in查询
- **match**: 全文检索
- **>**: 范围查询，大于
- **<**: 范围查询，小于
- **>=**: 范围查询，大于等于
- **<=**: 范围查询，小于等于

例 97. 精确匹配(term)

选择满足hphm中数据为“鲁D528E8”的对应dt值：

```
SELECT dt FROM hbase_index_demo
WHERE CONTAINS(hphm, "term '鲁D528E8' ")
ORDER BY dt LIMIT 100;
```

例 98. 前缀匹配(prefix)

选择满足hphm中前缀为“鲁D528”的对应dt值，对后文无限制：

```
SELECT dt FROM hbase_index_demo
WHERE CONTAINS(hphm, "prefix '鲁D528' ")
ORDER BY dt LIMIT 100;
```

例 99. 模糊查询(wildcard)

选择满足hphm中数据格式为“鲁D*28E8”的对应dt值，**表示** 前字符出现任意次：

```
SELECT dt FROM hbase_index_demo
WHERE CONTAINS(hphm, "wildcard '鲁D*28E8' ")
ORDER BY dt LIMIT 100;
```

例 100. 多个精确匹配

需同时满足两个 **CONTAINS** 条件：

```
SELECT dt FROM hbase_index_demo
WHERE CONTAINS(dt, "term '9223370633649793807' ")
  AND CONTAINS(hphm, "term '鲁P56729' ")
ORDER BY dt LIMIT 100;
```



不要对非STRING类型进行模糊、前缀、正则等查询。如果是对数字类型的进行范围查询，则需保证该列的数据类型为#b才可以。

例 101. 范围条件

选择满足dt中数据范围不大于“9223370647735474807”的对应dt值：

```
SELECT dt FROM hbase_index_demo
WHERE CONTAINS(dt, "<= '9223370647735474807' ")
AND CONTAINS(hphm, "term'鲁D0H795'")
ORDER BY dt LIMIT 100;
```

例 102. 正则表达式(regexp)

选择满足hphm中数据格式满足正则表达式为 '**s.*y**' 的对应dt值，'*****' 表示有任意的某个字符，它在s和y中出现任意次：

```
SELECT dt FROM hbase_index_demo
WHERE CONTAINS(hphm, "regexp 's.*y' ");
```

例 103. in表达式(in)

选择满足hphm中数据为“鲁D0H795, 鲁D528E8, 鲁DT4, 鲁DTXXX, 鲁DT4058”中其一的对应dt值：

```
SELECT dt FROM hbase_index_demo
WHERE CONTAINS(hphm, "in '鲁D0H795,鲁D528E8,鲁DT4,鲁DTXXX,鲁DT4058' ");
```

例 104. 范围表达式(range)

数据范围是较特殊的search_content，支持开闭区间的组合：

```
SELECT dt FROM hbase_index_demo
WHERE CONTAINS(hphm, "range
'[9223370647735474807,9223370647735478807]'");
SELECT dt FROM hbase_index_demo
WHERE CONTAINS(hphm, "range
'(9223370647735474807,9223370647735478807)'");
SELECT dt FROM hbase_index_demo
WHERE CONTAINS(hphm, "range
'(9223370647735474807,9223370647735478807)'");
SELECT dt FROM hbase_index_demo
WHERE CONTAINS(hphm, "range
'[9223370647735474807,9223370647735478807]'");
```

使用 `CONTAINS` 时的已知问题

1. 只支持Inceptor Engine在 `local mode`，在非 `local mode` 下运行直接报错。
2. 只支持正序排序。
3. 对OR操作符，不支持与全文检索条件的混用，如不能写 `CONTAINS(hphm, "= 'xxx'") OR yyyy = "yyyy"`；对AND条件无此限制，可以写 `CONTAINS(hphm, "= 'xxx'") AND yyyy = "yyyy"`，会使用全文检索条件。
4. 如果全文检索条件涉及的列不在全文索引中，会直接报错。
5. 使用 `range` 操作符比 `<=`, `\<`, `>=` 和 `>` 好，如推荐使用 `(dt, "range '[1234, 5678]'"` 而cankao不推荐使用 `(dt, "<= '5678' and >='1234'"`。
6. 使用 `in` 操作符比使用多个or表达式好，如推荐使用 `(hphm, "in '1234, 5678, 9011'"`），不推荐使用 `(hphm, "term '1234' or term'5678' or term'9011'"`。

12.4.4.2. 指定Hyperdrive映射表的索引

语法：指定Hyperdrive映射表的全局或本地索引

```
SELECT /*+USE_INDEX(<table_alias> USING <index_name>)*/ ... FROM <table>
<table_alias> WHERE <filter_conditions>;
```

语法：指定Hyperdrive映射表的全文索引

```
SELECT /*+USE_INDEX(<table_alias> USING FULLTEXT)*/ ... FROM <table>
<table_alias> WHERE <filter_conditions>;
```

语法：不指定任何索引做查询（明确放弃在查询中使用任何索引）

```
SELECT /*+USE_INDEX(<table_alias> USING NOT_USE_INDEX)*/ ... FROM <table>
<table_alias> WHERE <filter_conditions>;
```



使用索引查询时必须要给表起化名，并在指定索引时使用表化名。

例 105. 指定Hyperdrive表的本地索引

```
SELECT /*+USE_INDEX(t USING name_balance_local_index)*/ * FROM test t
WHERE t.name = 'Alice';
```

例 106. 指定Hyperdrive表的全局索引

```
SELECT /*+USE_INDEX(t USING name_global_index)*/ * FROM test t WHERE t.name = 'Alice';
```

例 107. 指定Hyperdrive表的全文索引

```
SELECT /*+USE_INDEX(t USING FULLTEXT)*/ * FROM test t WHERE t.name = 'Alice';
```

例 108. 不使用索引做查询

```
SELECT /*+USE_INDEX(t using NOT_USE_INDEX)*/ * FROM test t WHERE t.name = 'Alice';
```

例 109. 在 JOIN 中使用两张表中的索引做查询

```
SELECT /*+USE_INDEX(t USING name_global_index, d USING absence_global_index)*/ name, absence FROM test t JOIN demo d ON t.key=d.key WHERE t.name = 'Alice' AND d.absence = '2015-02-14';
```

注意，过滤条件中两个索引的条件都要包含。

13. Transwarp HBase故障诊断

13.1. 一般准则

故障诊断总是从Master的日志（log）开始。通常情况下，它总是一行一行地重复信息。如果不是这样，说明有问题，可以Google或是用 <http://search-hadoop.com> 来搜索遇到的异常。

错误很少仅仅单独出现在Transwarp HBase中，通常是某一个地方出了问题，引起各处大量异常和调用栈跟踪信息。遇到这样的错误，最好的办法是根据日志向上回溯，找到最初的异常。例如 **RegionServer**（区域服务器）会在退出的时候打印一些metrics。过滤搜索Dump文件，就应该可以找到最初的异常信息。

RegionServer的自杀是很“正常”的。当一些事情发生错误的，它们就会自杀。如果 **ulimit** 和 **xcievers** 没有修改，在某些地方DataNode无法创建新线程，HDFS将无法运转正常。因而在Transwarp HBase看来，HDFS死掉了。假想一下，你的MySQL突然无法访问它的本地文件系统，它会怎么做。同样的事情会发生在Transwarp HBase和HDFS上。还有一个造成RegionServer自杀的常见的原因是，它们执行了一个长时间的GC（**garbage collection**，垃圾收集）停顿操作，这个时间超过了ZooKeeper的会话时长。

13.2. 日志

关键构件的日志位置（<user>是启动服务的用户，<hostname> 是机器的名字）

NameNode: /var/logs/hdfs1/hadoop-<user>-namenode-<hostname>.log

DataNode: /var/logs/hdfs1/hadoop-<user>-datanode-<hostname>.log

NodeManager: /var/logs/yarn1/yarn-<user>-nodemanager-<hostname>.log

ResourceManager: /var/logs/yarn1/yarn-<user>-resourcemanager-<hostname>.log

HMaster: /var/logs/hyperbase1/hbase-<user>-master-<hostname>.log

RegionServer: /var/logs/hyperbase1/hbase-<user>-regionserver-<hostname>.log

ZooKeeper: /var/logs/zookeeper1/zookeeper.log

13.2.1. 日志位置

对于独立部署的日志，它们自然都会在一台机器上，但这仅仅是一个开发配置。对于生产部署，它们都会运行在一个集群上。

- NameNode

NameNode的日志在NameNode server上。Transwarp HBase Master通常也运行在NameNode server上，ZooKeeper通常也是这样。

对于小一点的集群，ResourceManager也通常运行在NameNode server上面。

- DataNode

每一台DataNode server有一个HDFS的日志，Region有一个Transwarp HBase日志。

每个DataNode server还有一份NodeManager日志，来记录MapReduce的Task执行情况。

13.2.2. 日志级别

启用RPC级别日志

启用RPC级别日志通常需要关注服务器的时间控制。一旦启用，日志将会大量喷涌。不推荐让该日志启用状态保持过长时间。为了启用RPC级别日志，浏览 **RegionServer UI**，点击 **Log Level**。在 **package org.apache.hadoop.ipc**（是hadoop. ipc，而非hbase. ipc）中，将日志级别设置为 **Debug**。接着跟踪（tail）RegionServers的日志并分析。

要禁用日志，可将日志级别调回 **INFO** 级别。

13.2.3. JVM Garbage Collection Log（垃圾收集日志）

Transwarp HBase的是内存密集型（**memory intensive**），并使用默认的GC可以看到长时间停顿中的所有线程，包括 **Juliet Pause aka "GC of Death"**。为了调试和确认这种问题的发生，可以在JVM（Java virtual machine）中开启GC logging。

要在 **hbase-env.sh** 启用GC logging，可将下面某行的注释去掉。

```
# This enables basic gc logging to the .out file.
# export SERVER_GC_OPTS="-verbose:gc -XX:+PrintGCDetails
-XX:+PrintGCDateStamps"

# This enables basic gc logging to its own file.
# export SERVER_GC_OPTS="-verbose:gc -XX:+PrintGCDetails
-XX:+PrintGCDateStamps -Xloggc:<FILE-PATH>

# This enables basic GC logging to its own file with automatic log
rolling. Only applies to jdk 1.6.0_34+ and 1.7.0_2+.
# export SERVER_GC_OPTS="-verbose:gc -XX:+PrintGCDetails
-XX:+PrintGCDateStamps -Xloggc:<FILE-PATH> -XX:+UseGCLLogFileRotation
-XX:NumberOfGCLogFiles=1 -XX:GCLLogFileSize=512M"

# If <FILE-PATH> is not replaced, the log file(.gc) would be generated in
the HBASE_LOG_DIR.
```

这时，你可以看到如下日志：

```

64898.952: [GC [1 CMS-initial-mark: 2811538K(3055704K)]
2812179K(3061272K), 0.0007360 secs] [Times: user=0.00 sys=0.00, real=0.00
secs]
64898.953: [CMS-concurrent-mark-start]
64898.971: [GC 64898.971: [ParNew: 5567K->576K(5568K), 0.0101110 secs]
2817105K->2812715K(3061272K), 0.0102200 secs] [Times: user=0.07 sys=0.00,
real=0.01 secs]

```

在这个部分，第一行表示CMS最初标记需要0.0007360秒的停顿。它暂停了整个虚拟机在那个时段的所有线程。

第三行意味着有一个 **minor GC**，它将虚拟机暂停了0.0101110秒，也即10毫秒。它将 **ParNew** 从5.5M减少到576K。之后我们可以看到：

```

64901.445: [CMS-concurrent-mark: 1.542/2.492 secs] [Times: user=10.49
sys=0.33, real=2.49 secs]
64901.445: [CMS-concurrent-preclean-start]
64901.453: [GC 64901.453: [ParNew: 5505K->573K(5568K), 0.0062440 secs]
2868746K->2864292K(3061272K), 0.0063360 secs] [Times: user=0.05 sys=0.00,
real=0.01 secs]
64901.476: [GC 64901.476: [ParNew: 5563K->575K(5568K), 0.0072510 secs]
2869283K->2864837K(3061272K), 0.0073320 secs] [Times: user=0.05 sys=0.01,
real=0.01 secs]
64901.500: [GC 64901.500: [ParNew: 5517K->573K(5568K), 0.0120390 secs]
2869780K->2865267K(3061272K), 0.0121150 secs] [Times: user=0.09 sys=0.00,
real=0.01 secs]
64901.529: [GC 64901.529: [ParNew: 5507K->569K(5568K), 0.0086240 secs]
2870200K->2865742K(3061272K), 0.0087180 secs] [Times: user=0.05 sys=0.00,
real=0.01 secs]
64901.554: [GC 64901.555: [ParNew: 5516K->575K(5568K), 0.0107130 secs]
2870689K->2866291K(3061272K), 0.0107820 secs] [Times: user=0.06 sys=0.00,
real=0.01 secs]
64901.578: [CMS-concurrent-preclean: 0.070/0.133 secs] [Times: user=0.48
sys=0.01, real=0.14 secs]
64901.578: [CMS-concurrent-abortable-preclean-start]
64901.584: [GC 64901.584: [ParNew: 5504K->571K(5568K), 0.0087270 secs]
2871220K->2866830K(3061272K), 0.0088220 secs] [Times: user=0.05 sys=0.00,
real=0.01 secs]
64901.609: [GC 64901.609: [ParNew: 5512K->569K(5568K), 0.0063370 secs]
2871771K->2867322K(3061272K), 0.0064230 secs] [Times: user=0.06 sys=0.00,
real=0.01 secs]
64901.615: [CMS-concurrent-abortable-preclean: 0.007/0.037 secs] [Times:
user=0.13 sys=0.00, real=0.03 secs]
64901.616: [GC[YG occupancy: 645 K (5568 K)]64901.616: [Rescan (parallel),
0.0020210 secs]64901.618: [weak refs processing, 0.0027950 secs] [1 CMS-
remark: 2866753K(3055704K)] 2867399K(3061272K), 0.0049380 secs] [Times:
user=0.00 sys=0.01, real=0.01 secs]
64901.621: [CMS-concurrent-sweep-start]

```

第一行，表示CMS当前标记（寻找垃圾）消耗了2.4秒。但是这2.4秒是“并发的”，Java在时间上没有暂停。

这里有一些**minor GC**，在倒数第二行有一个停顿。

```
64901.616: [GC[YG occupancy: 645 K (5568 K)]64901.616: [Rescan (parallel)
, 0.0020210 secs]64901.618: [weak refs processing, 0.0027950 secs] [1 CMS-
remark: 2866753K(3055704K)] 2867399K(3061272K), 0.0049380 secs] [Times:
user=0.00 sys=0.01, real=0.01 secs]
```

这里停顿了0.0049380秒（即4.9毫秒）来“标记”堆。

此时扫描开始，你可以看到堆的规模正在变小：

```
64901.637: [GC 64901.637: [ParNew: 5501K->569K(5568K), 0.0097350 secs]
2871958K->2867441K(3061272K), 0.0098370 secs] [Times: user=0.05 sys=0.00,
real=0.01 secs]
... lines removed ...
64904.936: [GC 64904.936: [ParNew: 5532K->568K(5568K), 0.0070720 secs]
1365024K->1360689K(3061272K), 0.0071930 secs] [Times: user=0.05 sys=0.00,
real=0.01 secs]
64904.953: [CMS-concurrent-sweep: 2.030/3.332 secs] [Times: user=9.57
sys=0.26, real=3.33 secs]
```

这里CMS扫描消耗了3.332秒，堆由约2.8GB变小到约1.3GB。

关键要保持所有的暂停都在一个比较低的水平。CMS停顿往往很短，但如果ParNew开始增长，你可以看到minor GC停顿的时候渐渐达到100ms，然后超过100ms，甚至最后会高达400ms。

这可能是由于ParNew的规模大小引起的，该规模应该相对较小。如果ParNew在Transwarp HBase运行一段时间后非常大，例如ParNew约为150MB，这时就该限制ParNew大小了。（ParNew越大，collection需要的时间就越长；但如果太小，question：对象会过快被提升至old gen）。下面，我们将new gen的规模限制在64M。

在 `hbase-env.sh` 文件中加入下面这行：

```
export SERVER_GC_OPTS="$SERVER_GC_OPTS -XX:NewSize=64m -XX:MaxNewSize=64m"
```

同样的，要为客户端进程启用GC logging，在 `hbase-env.sh` 文件中下面某行的注释取消掉：

```

# This enables basic gc logging to the .out file.
# export CLIENT_GC_OPTS="-verbose:gc -XX:+PrintGCDetails
-XX:+PrintGCDateStamps"

# This enables basic gc logging to its own file.
# export CLIENT_GC_OPTS="-verbose:gc -XX:+PrintGCDetails
-XX:+PrintGCDateStamps -Xloggc:<FILE-PATH>

# This enables basic GC logging to its own file with automatic log
rolling. Only applies to jdk 1.6.0_34+ and 1.7.0_2+.
# export CLIENT_GC_OPTS="-verbose:gc -XX:+PrintGCDetails
-XX:+PrintGCDateStamps -Xloggc:<FILE-PATH> -XX:+UseGCLogFileRotation
-XX:NumberOfGCLogFiles=1 -XX:GCLogFileSize=512M"

# If <FILE-PATH> is not replaced, the log file(.gc) would be generated in
the HBASE_LOG_DIR .

```

13.3. 资源

13.3.1. IRC

#hbase on irc.freenode.net

13.3.2. JIRA

JIRA 在处理Hadoop/HBase相关问题时也很有帮助。

13.4. 工具

13.4.1. 内置工具

- Master Web接口

Master启动的缺省web接口为端口60010。

Master web UI 列表创建了表和它们的定义（如列族，块大小等）。另外，集群中的可用RegionServers与选定的高层次指标（requests，region数量，usedHeap，maxHeap）列在一起。Master web UI允许导航到每个RegionServer Web UI。

- RegionServer Web接口

RegionServer启动的缺省web接口为端口60030。

RegionServer web UI 列出了在线region和它们的开始/结束键，以及即时点（**point-in-time**）的RegionServer的指标（requests，region数量，storeFileIndexSize，compactionQueueSize等）。

- zkcli

zkcli 是一个研究ZooKeeper相关问题的有用工具。调用：

```
./hbase zkcli -server host:port <cmd> <args>
```

命令（和参数）：

```
connect host:port
get path [watch]
ls path [watch]
set path data [version]
delquota [-n|-b] path
quit
printwatches on|off
create [-s] [-e] path data acl
stat path [watch]
close
ls2 path [watch]
history
listquota path
setAcl path acl
getAcl path
sync path
redo cmdno
addauth scheme auth
delete path [version]
setquota -n|-b val path
```

13.4.2. 外部工具

- tail

tail 是一个命令行工具，可以用来看日志的结尾。加入的 **-f** 参数后，就会在数据更新的时候自己刷新。用它来看日志很方便。例如，一个集群需要花很多时间来启动或关闭，你可以开启一个新的终端来跟踪它的Master log(或是RegionServer log)。

- top

top 是一个很重要的工具，它用来看机器各个进程的资源占用情况。下面是一个生产系统的例子：

例 110. top示例

```
top - 14:46:59 up 39 days, 11:55, 1 user, load average: 3.75,
3.57, 3.84
Tasks: 309 total, 1 running, 308 sleeping, 0 stopped, 0 zombie
Cpu(s): 4.5%us, 1.6%sy, 0.0%ni, 91.7%id, 1.4%wa, 0.1%hi,
0.6%si, 0.0%st
Mem: 24414432k total, 24296956k used, 117476k free, 7196k
buffers
Swap: 16008732k total, 14348k used, 15994384k free, 11106908k
cached

 PID USER      PR  NI    VIRT    RES    SHR   S %CPU %MEM   TIME+ COMMAND
15558 hadoop    18  -2 3292m  2.4g  3556 S    79 10.4    6523:52 java
13268 hadoop    18  -2 8967m  8.2g  4104 S    21 35.1    5170:30 java
 8895 hadoop    18  -2 1581m 497m  3420 S    11  2.1    4002:32 java
...
...
```

这里你可以看到系统最近5分钟的load average是3.75，意思就是说这5分钟里面平均有3.75个线程在CPU时间的等待队列里面。通常来说，最完美的情况是这个值和CPU的核数相等，比这个值低意味着资源闲置，比这个值高意味着过载。这是一个重要的概念，要想理解得更多，可以看这篇文章
<http://www.linuxjournal.com/article/9001>.

除了负载，我们可以看到系统已经几乎使用了它的全部RAM，其中大部分都是用于OS cache(这是一件好事)。Swap只使用了一点点KB, 这正是我们期望的，如果数值很高的话，就意味着在进行交换，这对Java程序的性能是致命的。另一种检测交换的方法是看 **load average** 是否过高(load average过高还可能是磁盘损坏或者其他什么原因导致的)。

默认情况下进程列表不是很有用，我们可以看到3个Java进程使用了111%的CPU。要想知道哪个进程是哪个，可以输入 **c**，每一行就会扩展信息。输入 **1** 可以显示CPU的每个核的具体状况。

- jps

jps 是JDK集成的一个工具，可以用来看当前用户的Java进程id。(如果是root, 可以看到所有用户的id)，例如：

例 111. jps示例

```

hadoop@sv4borg12:~$ jps
1322 TaskTracker
17789 HRegionServer
27862 Child
1158 DataNode
25115 HQuorumPeer
2950 Jps
19750 ThriftServer
18776 jmx

```

按顺序看：

- **Hadoop TaskTracker** , 管理本地的Task
- **HBase RegionServer** , 提供region的服务
- **Child** , 一个MapReduce task, 无法看出详细类型
- **Hadoop DataNode** , 管理blocks
- **HQuorumPeer** , ZooKeeper集群的成员
- **Jps** , 当前进程
- **ThriftServer** , 当thrift启动后, 就会有这个进程
- **jmx** , 这个是本地监控平台的进程。可以不用这个。

可以看到这个进程启动是全部命令行信息：

```

hadoop@sv4borg12:~$ ps aux | grep HRegionServer
hadoop 17789 155 35.2 9067824 8604364 ? S<1 Mar04 9855:48
/usr/java/jdk1.6.0_14/bin/java -Xmx8000m -XX:+DoEscapeAnalysis
-XX:+AggressiveOpts -XX:+UseConcMarkSweepGC -XX:NewSize=64m
-XX:MaxNewSize=64m -XX:CMSInitiatingOccupancyFraction=88 -verbose:gc
-XX:+PrintGCDetails -XX:+PrintGCTimeStamps -Xloggc:/export1/hadoop/logs/gc
-hbase.log -Dcom.sun.management.jmxremote.port=10102
-Dcom.sun.management.jmxremote.authenticate=true
-Dcom.sun.management.jmxremote.ssl=false
-Dcom.sun.management.jmxremote.password.file=/home/hadoop/hbase/conf/jmxre
mote.password -Dcom.sun.management.jmxremote
-Dhbase.log.dir=/export1/hadoop/logs -Dhbase.log.file=hbase-hadoop
-regionserver-sv4borg12.log -Dhbase.home.dir=/home/hadoop/hbase
-Dhbase.id.str=hadoop -Dhbase.root.logger=INFO,DRFA
-Djava.library.path=/home/hadoop/hbase/lib/native/Linux-amd64-64
-classpath /home/hadoop/hbase/bin/../conf:[many
jars]:/home/hadoop/hadoop/conf
org.apache.hadoop.hbase.regionserver.HRegionServer start

```

- jstack

jstack

是很重要(除了看Log)的java工具，可以看到具体的Java进程的在做什么。可以先用Jps看到进程的Id, 然后就可以用jstack。它会按线程的创建顺序显示线程的列表，还有这个线程在做什么。下面是例子：

这个主线程是关于一个RegionServer正在等master返回什么信息：

```
"regionserver60020" prio=10 tid=0x0000000040ab4000 nid=0x45cf waiting on
condition [0x00007f16b6a96000..0x00007f16b6a96a70]
java.lang.Thread.State: TIMED_WAITING (parking)
    at sun.misc.Unsafe.park(Native Method)
    - parking to wait for  <0x00007f16cd5c2f30> (a
java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject)
    at
java.util.concurrent.locks.LockSupport.parkNanos(LockSupport.java:198)
    at
java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.awaitNanos(AbstractQueuedSynchronizer.java:1963)
    at
java.util.concurrent.LinkedBlockingQueue.poll(LinkedBlockingQueue.java:395)
    at
org.apache.hadoop.hbase.regionserver.HRegionServer.run(HRegionServer.java:647)
    at java.lang.Thread.run(Thread.java:619)
```

MemStore flusher进程正flush到一个文件中：

```
"regionserver60020.cacheFlusher" daemon prio=10 tid=0x0000000040f4e000
nid=0x45eb in Object.wait() [0x00007f16b5b86000..0x00007f16b5b87af0]
java.lang.Thread.State: WAITING (on object monitor)
    at java.lang.Object.wait(Native Method)
    at java.lang.Object.wait(Object.java:485)
    at org.apache.hadoop.ipc.Client.call(Client.java:803)
    - locked <0x00007f16cb14b3a8> (a org.apache.hadoop.ipc.Client$Call)
    at org.apache.hadoop.ipc.RPC$Invoker.invoke(RPC.java:221)
    at $Proxy1.complete(Unknown Source)
    at sun.reflect.GeneratedMethodAccessor38.invoke(Unknown Source)
    at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessor
Impl.java:25)
    at java.lang.reflect.Method.invoke(Method.java:597)
    at
org.apache.hadoop.io.retry.RetryInvocationHandler.invokeMethod(RetryInv
ocationHandler.java:82)
    at
org.apache.hadoop.io.retry.RetryInvocationHandler.invoke(RetryInvocation
Handler.java:59)
    at $Proxy1.complete(Unknown Source)
    at
org.apache.hadoop.hdfs.DFSClient$DFSOutputStream.closeInternal(DFSClient
.java:3390)
    - locked <0x00007f16cb14b470> (a
org.apache.hadoop.hdfs.DFSClient$DFSOutputStream)
    at
org.apache.hadoop.hdfs.DFSClient$DFSOutputStream.close(DFSClient.java:33
```

```
-04)
    at
org.apache.hadoop.fs.FSDataOutputStream$PositionCache.close(FSDataOutputStream.java:61)
    at
org.apache.hadoop.fs.FSDataOutputStream.close(FSDataOutputStream.java:86)
)
    at
org.apache.hadoop.hbase.io.hfile.HFile$Writer.close(HFile.java:650)
    at
org.apache.hadoop.hbase.regionserver.StoreFile$Writer.close(StoreFile.java:853)
    at
org.apache.hadoop.hbase.regionserver.Store.internalFlushCache(Store.java:467)
    - locked <0x00007f16d00e6f08> (a java.lang.Object)
    at
org.apache.hadoop.hbase.regionserver.Store.flushCache(Store.java:427)
    at
org.apache.hadoop.hbase.regionserver.Store.access$100(Store.java:80)
    at
org.apache.hadoop.hbase.regionserver.Store$StoreFlusherImpl.flushCache(Store.java:1359)
    at
org.apache.hadoop.hbase.regionserver.HRegion.internalFlushcache(HRegion.java:907)
    at
org.apache.hadoop.hbase.regionserver.HRegion.internalFlushcache(HRegion.java:834)
    at
org.apache.hadoop.hbase.regionserver.HRegion.flushcache(HRegion.java:786)
)
    at
org.apache.hadoop.hbase.regionserver.MemStoreFlusher.flushRegion(MemStoreFlusher.java:250)
    at
org.apache.hadoop.hbase.regionserver.MemStoreFlusher.flushRegion(MemStoreFlusher.java:224)
    at
org.apache.hadoop.hbase.regionserver.MemStoreFlusher.run(MemStoreFlusher.java:146)
```

处理线程正在等待着什么(例如put, delete, scan...)：

```
"IPC Server handler 16 on 60020" daemon prio=10 tid=0x00007f16b011d800
nid=0x4a5e waiting on condition [0x00007f16afefd000..0x00007f16afefd9f0]
    java.lang.Thread.State: WAITING (parking)
        at sun.misc.Unsafe.park(Native Method)
        - parking to wait for  <0x00007f16cd3f8dd8> (a
java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject)
        at
java.util.concurrent.locks.LockSupport.park(LockSupport.java:158)
        at
java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.aw
ait(AbstractQueuedSynchronizer.java:1925)
        at
java.util.concurrent.LinkedBlockingQueue.take(LinkedBlockingQueue.java:3
58)
        at
org.apache.hadoop.hbase.ipc.HBaseServer$Handler.run(HBaseServer.java:101
3)
```

有一个线程正在忙，它在递增一个counter(这个阶段是在创建一个scanner，用来读最新的值)：

```
"IPC Server handler 66 on 60020" daemon prio=10 tid=0x00007f16b006e800
nid=0x4a90 runnable [0x00007f16acb77000..0x00007f16acb77cf0]
    java.lang.Thread.State: RUNNABLE
        at
org.apache.hadoop.hbase.regionserver.KeyValueHeap.<init>(KeyValueHeap.java:56)
        at
org.apache.hadoop.hbase.regionserver.StoreScanner.<init>(StoreScanner.java:79)
        at
org.apache.hadoop.hbase.regionserver.Store.getScanner(Store.java:1202)
        at
org.apache.hadoop.hbase.regionserver.HRegion$RegionScanner.<init>(HRegion.java:2209)
        at
org.apache.hadoop.hbase.regionserver.HRegion.instantiateInternalScanner(HRegion.java:1063)
        at
org.apache.hadoop.hbase.regionserver.HRegion.getScanner(HRegion.java:1055)
        at
org.apache.hadoop.hbase.regionserver.HRegion.getScanner(HRegion.java:1039)
        at
org.apache.hadoop.hbase.regionserver.HRegion.getLastIncrement(HRegion.java:2875)
        at
org.apache.hadoop.hbase.regionserver.HRegion.incrementColumnValue(HRegion.java:2978)
        at
org.apache.hadoop.hbase.regionserver.HRegionServer.incrementColumnValue(HRegionServer.java:2433)
        at sun.reflect.GeneratedMethodAccessor20.invoke(Unknown Source)
        at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
        at java.lang.reflect.Method.invoke(Method.java:597)
        at
org.apache.hadoop.hbase.ipc.HBaseRPC$Server.call(HBaseRPC.java:560)
        at
org.apache.hadoop.hbase.ipc.HBaseServer$Handler.run(HBaseServer.java:1027)
```

还有一个线程在从HDFS获取数据:

```

"IPC Client (47) connection to sv4borg9/10.4.24.40:9000 from hadoop"
daemon prio=10 tid=0x00007f16a02d0000 nid=0x4fa3 runnable
[0x00007f16b517d000..0x00007f16b517dbf0]
    java.lang.Thread.State: RUNNABLE
        at sun.nio.ch.EPollArrayWrapper.epollWait(Native Method)
        at
sun.nio.ch.EPollArrayWrapper.poll(EPollArrayWrapper.java:215)
        at
sun.nio.ch.EPollSelectorImpl.doSelect(EPollSelectorImpl.java:65)
        at
sun.nio.ch.SelectorImpl.lockAndDoSelect(SelectorImpl.java:69)
        - locked <0x00007f17d5b68c00> (a sun.nio.ch.Util$1)
        - locked <0x00007f17d5b68be8> (a
java.util.Collections$UnmodifiableSet)
        - locked <0x00007f1877959b50> (a sun.nio.ch.EPollSelectorImpl)
        at sun.nio.ch.SelectorImpl.select(SelectorImpl.java:80)
        at
org.apache.hadoop.net.SocketIOWithTimeout$SelectorPool.select(SocketIOWi
thTimeout.java:332)
        at
org.apache.hadoop.net.SocketIOWithTimeout.doIO(SocketIOWithTimeout.java:
157)
        at
org.apache.hadoop.net.SocketInputStream.read(SocketInputStream.java:155)
        at
org.apache.hadoop.net.SocketInputStream.read(SocketInputStream.java:128)
        at java.io.FilterInputStream.read(FilterInputStream.java:116)
        at
org.apache.hadoop.ipc.Client$Connection$PingInputStream.read(Client.java
:304)
        at
java.io.BufferedInputStream.fill(BufferedInputStream.java:218)
        at
java.io.BufferedInputStream.read(BufferedInputStream.java:237)
        - locked <0x00007f1808539178> (a java.io.BufferedInputStream)
        at java.io.DataInputStream.readInt(DataInputStream.java:370)
        at
org.apache.hadoop.ipc.Client$Connection.receiveResponse(Client.java:569)
        at
org.apache.hadoop.ipc.Client$Connection.run(Client.java:477)

```

这里是一个RegionServer故障， master正尝试着恢复：

```

"LeaseChecker" daemon prio=10 tid=0x00000000407ef800 nid=0x76cd waiting
on condition [0x00007f6d0eae2000..0x00007f6d0eae2a70]
-->
    java.lang.Thread.State: WAITING (on object monitor)
        at java.lang.Object.wait(Native Method)
        at java.lang.Object.wait(Object.java:485)
        at org.apache.hadoop.ipc.Client.call(Client.java:726)
        - locked <0x00007f6d1cd28f80> (a
org.apache.hadoop.ipc.Client$Call)
        at org.apache.hadoop.ipc.RPC$Invoker.invoke(RPC.java:220)
        at $Proxy1.recoverBlock(Unknown Source)
        at
org.apache.hadoop.hdfs.DFSClient$DFSOutputStream.processDatanodeError(DF
SClient.java:2636)
        at
org.apache.hadoop.hdfs.DFSClient$DFSOutputStream.<init>(DFSClient.java:2
832)
        at org.apache.hadoop.hdfs.DFSClient.append(DFSClient.java:529)
        at
org.apache.hadoop.hdfs.DistributedFileSystem.append(DistributedFileSyste
m.java:186)
        at org.apache.hadoop.fs.FileSystem.append(FileSystem.java:530)
        at
org.apache.hadoop.hbase.util.FSUtils.recoverFileLease(FSUtils.java:619)
        at
org.apache.hadoop.hbase.regionserver.wal.HLog.splitLog(HLog.java:1322)
        at
org.apache.hadoop.hbase.regionserver.wal.HLog.splitLog(HLog.java:1210)
        at
org.apache.hadoop.hbase.master.HMaster.splitLogAfterStartup(HMaster.java
:648)
        at
org.apache.hadoop.hbase.master.HMaster.joinCluster(HMaster.java:572)
        at
org.apache.hadoop.hbase.master.HMaster.run(HMaster.java:503)

```

- OpenTSDB

OpenTSDB 是一个Ganglia的很好的替代品，因为它使用Transwarp HBase来存储所有的时序而不需要采样。使用OpenTSDB来监控Transwarp HBase是一个很好的实践。

这里有一个例子，集群正在同时进行上百个compaction，严重影响了IO性能。

给集群构建一个图表监控是一个很好的实践。包括集群和每台机器。这样就可以快速定位到问题。例如，在StumbleUpon，每个机器有一个图表监控，包括OS和Transwarp HBase，涵盖所有的重要的信息。你也可以登录到机器上，获取更多的信息。

- clusterssh+top

clusterssh+top，感觉是一个穷人用的监控系统，但当你只有几台机器时，它确实很有效、很好设置。启动clusterssh后，每台机器都会得到一个终端。另外还有一个终端，你在这个终端的操作都会反应到其他的每一个终端上。这就意味着，你执行一次 **top**，所有机器都会同时给你全部的集群状态信息。你还可以这样同时跟踪全部的log，编辑文件，等等。

13.5. 客户端

13.5.1. 因 `hbase.client.scanner.max.result.size` 不匹配使得扫描结果缺失

如果客户端或者服务器版本低于0.98.11/1.0.0，以及服务器在 `hbase.client.scanner.max.result.size` 上的值比客户端小，那么到达服务器 `hbase.client.scanner.max.result.size` 的扫描请求就有可能丢失数据。尤其，0.98.11的 `hbase.client.scanner.max.result.size` 的默认值为2MB，但是其他版本的默认值更大。由于这个原因，在使用0.98.11时要格外小心。

13.5.2. ScannerTimeoutException/UnknownScannerException

当从客户端到RegionServer的RPC请求超时，这个错误会被抛出。例如，如果 `Scan.setCaching` 的值设置为500，RPC请求就要去获取500行的数据，每500次 `.next()` 操作获取一次。因为数据是以大块的形式传到客户端的，就可能造成超时。将 `setCaching` 的值调小是一个解决办法，但是这个值要是设的太小就会影响性能。

13.5.3. Thrift 和 Java APIs的性能偏差

如果 `Scan.setCaching` 过高，性能会变得很糟，甚至会出现 `ScannerTimeoutExceptions`。question: 如果Thrift client针对给定的工作量，使用了错误的缓存设定，性相比于Java API就会变得很糟。针对给定的扫描，为了在Thrift client中设定缓存，可采用 `scannerGetList(scannerId, numRows)` 这种方法。这里，`numRows` 是表示缓存中行数的一个整数。在某种情况下，人们发现将Thrift扫描的缓存从1000减少到100，和Java API比起来，能在邻近奇偶校验上取得更好的性能。

13.5.4. Scanner.next 时出现 LeaseException

在某些情况下，客户端从RegionServer获取数据时，可能会收到 `LeaseException` 而非通常的 `ScannerTimeoutException` 或 `UnknownScannerException`。通常这类报错是来源于 `org.apache.hadoop.hbase.regionserver.Leases.removeLease(Leases.java:230)`（行号可能不同）。它往往在一次缓慢/冻结的 `RegionServer#next` 调用下产生。通过 `hbase.rpc.timeout > hbase.regionserver.lease.period` 可避免这一问题。

13.5.5. Shell或客户端应用抛出很多不太重要的异常

由于0.20.0在 `org.apache.hadoop.hbase.*` 中的默认log level是DEBUG，在你的客户端上，编辑 `$HBASE_HOME/conf/log4j.properties` 并改变如下：

```
log4j.logger.org.apache.hadoop.hbase=DEBUG to this:  
log4j.logger.org.apache.hadoop.hbase=INFO, or even  
log4j.logger.org.apache.hadoop.hbase=WARN.
```

13.5.6. 压缩时客户端长时停顿

这是一个在Transwarp HBase区列表（`dist-list`）中经常被问的问题。它一般发生在是客户端正插入大量数据到相对未优化的Transwarp HBase集群中时。压缩加重停顿时间，尽管这不是问题源头。

对于为什么会停顿解释如下：

当给 **compactor** 太多很小的文件让它去压缩时，**compactor**会不得不把同样的数据反复压缩。这样会导致 **flusher** 线程被锁住，从而导致 **MemStores** 也被锁住，最后Put操作就会被锁在**MemStores**上。即使执行的是 **minor compactations** 也可能发生这种情况。为了调和这种情况，Transwarp HBase不在内存中压缩数据。在**MemStore**中的64MB的数据可能在压缩后变为6MB，存放为一个更小的 **StoreFile** 。有利的一面是，越来越多的数据被包装到同一region，但性能能通过编写更大的文件来实现 ——这就是为什么Transwarp HBase会等到 flushsize才继续下一步，flushsize在写入新的**StoreFile**前。更小的**StoreFiles**成为**compaction**的对象。没有压缩的话，这些文件会大许多也不需要这样多的**compaction**，然而，这样会导致I/O开销很大。

13.5.7. 安全客户端不能连接

你可能遇到如下错误：

```
Secure Client Connect ([Caused by GSSEException: No valid credentials provided
(Mechanism level: Request is a replay (34) V PROCESS_TGS)])
```

这种情况是由于 **MIT Kerberos replay_cache** 组件中的bug造成的。这些bug导致了旧版本的krb5服务器错误地阻止了 **Principal** 发送的后续请求。进而使得krb5服务器阻止了某个客户端（即某个HTable实例，它对每个RegionServer都有多线程连接）发送的连接；一些消息，如 **Request is a replay (34)**，会登入客户端日志中。你可以忽略这些消息，因为HTable默认为失败的连接重试5*10（50）次。如果重试之后与RegionServer的任何连接都失败了，HTable将抛出 **IOException** 异常，这就能让HTable的客户端能进一步处理这些问题。

还可以更新 **krb5-server** 到一个解决了这些问题的版本，如 **krb5-server-1.10.3**。

13.5.8. ZooKeeper 客户端连接错误

错误类似于…

```
11/07/05 11:26:41 WARN zookeeper.ClientCnxn: Session 0x0 for server null,
unexpected error, closing socket connection and attempting reconnect
java.net.ConnectException: Connection refused: no further information
        at sun.nio.ch.SocketChannelImpl.checkConnect(Native Method)
        at sun.nio.ch.SocketChannelImpl.finishConnect(Unknown Source)
        at
org.apache.zookeeper.ClientCnxn$SendThread.run(ClientCnxn.java:1078)
11/07/05 11:26:43 INFO zookeeper.ClientCnxn: Opening socket connection to
server localhost/127.0.0.1:2181
11/07/05 11:26:44 WARN zookeeper.ClientCnxn: Session 0x0 for server null,
unexpected error, closing socket connection and attempting reconnect
java.net.ConnectException: Connection refused: no further information
        at sun.nio.ch.SocketChannelImpl.checkConnect(Native Method)
        at sun.nio.ch.SocketChannelImpl.finishConnect(Unknown Source)
        at
org.apache.zookeeper.ClientCnxn$SendThread.run(ClientCnxn.java:1078)
11/07/05 11:26:45 INFO zookeeper.ClientCnxn: Opening socket connection to
server localhost/127.0.0.1:2181
```

…要么是 ZooKeeper 不在了，或网络不可达问题。

工具`zkcli`可以帮助调查 ZooKeeper 问题。

13.5.9. 客户端内存耗尽，但堆大小看起来不太变化

客户端内存耗尽，但堆大小看起来不太变化(`off-heap / direct heap` 在增长)。一个可行的解决方案是通过你的客户端，给JVM一个合理的属性值 `-XX:MaxDirectMemorySize`。在默认情况下，`MaxDirectMemorySize` 和 `-Xmx` (最大堆大小) 相等，我们可以将该值调小(例如，当客户端堆大小为12g时，`MaxDirectMemorySize` 调为1g)。但如果你将这个值调得过小，又可能造成 `FullGCs`。

13.5.10. 安全客户端不能连接

安全客户端不能连接，提示由 `GSSEexception` 引起: `No valid credentials provided (Mechanism level: Failed to find any Kerberos tgt)`。该问题可能是由多个原因造成了：

首先，检查你是否有一个 `Kerberos Ticket`，这是用来和一个安全的Transwarp HBase集群建立连接的。查看方式为在服务器命令行中执行 `klist`。如果你的机器上没有有效的Ticket，您需要进行获取。更多Kerberos Ticket资料，可参考[认证：获取Kerberos Ticket](#)

接着，咨询 [Java安全故障排查](#)。很多问题可以通过将 `javax.security.auth.useSubjectCredsOnly` 的系统值设为 `false`。

由于MIT Kerberos在写高速缓存时的格式发生过变化，所有在Oracle JDK 6 Update 26以前的版本都有个bug，那就是会使得Java无法读取MIT Kerberos 1.8.1以及更高版本的内容。如果你出现了这种问题，可以采用如下方式解决：首先登陆进 `kinit`，然后用 `kinit -R` 指令刷新高速缓存。这次刷新会重写高速缓存从而解决格式问题。

最后，依照您的Kerberos配置，您可能需要安装 `Java Cryptography Extension` 或者 `JCE`。确保JCE的jar文件被包含在了服务器和客户端系统的classpath路径中。

您还可能需要下载 `unlimited strength JCE policy files`。解压文件，然后将这些jar文件安装到 `<java-home>/lib/security` 中。

13.6. MapReduce

13.6.1. 你认为自己在用集群，实际上你在用本地(Local)

如下的调用栈在使用 `ImportTsv` 时发生，但同样的事可以在错误配置的任何任务中发生。

```

WARN mapred.LocalJobRunner: job_local_0001
java.lang.IllegalArgumentException: Can't read partitions file
    at
org.apache.hadoop.hbase.mapreduce.hadoopbackport.TotalOrderPartitioner.set
Conf(TotalOrderPartitioner.java:111)
    at
org.apache.hadoop.util.ReflectionUtils.setConf(ReflectionUtils.java:62)
    at
org.apache.hadoop.util.ReflectionUtils.newInstance(ReflectionUtils.java:11
7)
    at
org.apache.hadoop.mapred.MapTask$NewOutputCollector.<init>(MapTask.java:56
0)
        at org.apache.hadoop.mapred.MapTask.runNewMapper(MapTask.java:639)
        at org.apache.hadoop.mapred.MapTask.run(MapTask.java:323)
        at
org.apache.hadoop.mapred.LocalJobRunner$Job.run(LocalJobRunner.java:210)
Caused by: java.io.FileNotFoundException: File _partition.lst does not
exist.
    at
org.apache.hadoop.fs.RawLocalFileSystem.getFileStatus(RawLocalFileSystem.j
ava:383)
    at
org.apache.hadoop.fs.FilterFileSystem.getFileStatus(FilterFileSystem.java:
251)
        at org.apache.hadoop.fs.FileSystem.getLength(FileSystem.java:776)
        at
org.apache.hadoop.io.SequenceFile$Reader.<init>(SequenceFile.java:1424)
        at
org.apache.hadoop.io.SequenceFile$Reader.<init>(SequenceFile.java:1419)
        at
org.apache.hadoop.hbase.mapreduce.hadoopbackport.TotalOrderPartitioner.rea
dPartitions(TotalOrderPartitioner.java:296)

```

看到调用栈的关键部分了吗？就是…

```

at
org.apache.hadoop.mapred.LocalJobRunner$Job.run(LocalJobRunner.java:210)

```

LocalJobRunner 意思就是任务跑在本地，不在集群。

为了解决这个问题，你应该将 **HADOOP_CLASSPATH** 设置为包括Transwarp HBase的依赖性，来运行MapReduce工程。“hbase classpath”工具可以用来轻松地做到这一点。例如（用VERSION替代Transwarp HBase的版本）：

```

HADOOP_CLASSPATH=`hbase classpath` hadoop jar $HBASE_HOME/hbase-server-
VERSION.jar rowcounter usertable

```

13.7. NameNode

13.7.1. 表和region的HDFS 工具

要确定Transwarp HBase用了HDFS多大空间，可在NameNode使用 hadoop shell命令，例如…

```
hadoop fs -dus /hyperbase1
```

…返回全部Transwarp HBase对象磁盘占用的情况。

```
hadoop fs -dus /hyperbase1/data/default/myTable
```

…返回Transwarp HBase表’ myTable’ 磁盘占用的情况。

```
hadoop fs -du /hyperbase1/data/default/myTable
```

…返回Transwarp HBase的’ myTable’ 表的各区域列表的磁盘占用情况。

13.7.2. 浏览 HDFS，查看Transwarp HBase对象

有时需要浏览HDFS上的Transwarp HBase对象。对象包括WALs (Write Ahead Logs)、表、region、StoreFiles等。最简易的方法是在NameNode web应用中查看，端口50070。NameNode web 应用提供到集群中所有DataNode的链接，可以无缝浏览。

存储在HDFS集群中的HBase表的目录结构是…

```
/hyperbase1
 /data ①
   /<Namespace> ②
     /<Table> ③
       /<Region> ④
         /<ColumnFamily> ⑤
           /<StoreFile> ⑥
```

① 用户表数据所在目录

② 表按照namespace分组

③ 集群包含的表

④ 上面选定的表中的region

⑤ 上面选定的region中的列族

⑥ 上面选定的列族的存储文件

HDFS 中的Transwarp HBase WAL目录结构是…

```
/hyperbase1
/WALs
/<RegionServer>
/<HLog> ①
```

① 选定的RegionServer的WAL HLog文件

- 有数据但大小显示为0的WAL

这是HDFS的一个怪癖，当前正在被写入的文件会显示大小为0，但是一旦它关掉后又会显示出正确的尺寸。

- 用例([use-cases](#))

查询HDFS，获取Transwarp HBase对象的两个通常用例是研究表的 [uncompaction](#) 程度。如果每个列族有大量存储文件([StoreFile](#))，这表示需要 [major compaction](#) 。另外，在主紧缩之后，如果存储文件的比较小，意味着表的列族要减少。

13.7.3. 意想不到的文件系统峰值

如果您看到了一个意想不到的文件系统使用的峰值，可能是由下面两个原因造成的：

- 快照 (Snapshots)

当您创建一个快照时，Transwarp HBase会保持需要重建表状态的所有东西，包括删除单元和过期版本。这样，您的快照使用模式应该是精心策划的，并且您应该删去您不再需要的快照。快照被存放在 [/hyperbase1/data/namespace/tablename/.snapshots](#) 下，用于恢复快照的存档在 [/hyperbase1/archive/data/namspace/<_tablename_>/<_region_>/<_column_family_>/](#) 下。



不要手动在HDFS中操作快照和存档。 Transwarp HBase提供了API和Transwarp HBase Shell命令来管理它们。

- WAL

WAL([Write-ahead logs](#))被存放在Transwarp HBase根目录下的子目录中，如 [/hyperbase1/](#)。已经被处理了的WAL放在 [/hyperbase1/oldWALs/](#) 下，错误的WAL被放在 [/hyperbase1/corrupt/](#) 下以便检查。如果这几个子目录的规模变大了，那么检查Transwarp HBase日志来找到WAL没有被正确处理的根源。



不要手动在HDFS中操作WAL。

13.8. 网络

13.8.1. 网络峰值(Network Spikes)

如果看到周期性网络峰值，你可能需要检查 [compactionQueues](#) ，来看 [major compaction](#) 是否正在进行。

13.8.2. 回环IP(Loopback IP)

Transwarp HBase希望回环 IP 地址是 127.0.0.1.

13.8.3. 网络接口

所有网络接口是否正常？确定吗？

13.9. RegionServer

13.9.1. 启动错误

- 主服务器启动了，但RegionServer没有启动

主服务器认为RegionServer的IP地址是127.0.0.1——这是localhost，而且它被解析到master自己的localhost。

RegionServer错误的通知主服务器它们的IP地址是127.0.0.1。

修改RegionServer的 /etc/hosts 从…

```
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1      fully.qualified.regionservername
regionservername  localhost.localdomain localhost
::1            localhost6.localdomain6 localhost6
```

…改到(将主名称从localhost中移掉)…

```
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1      localhost.localdomain localhost
::1            localhost6.localdomain6 localhost6
```

- 压缩链接报错

因为LZO压缩算法需要在集群中的每台机器都要安装，这是一个启动失败的常见错误。如果你获得了如下信息

```
11/02/20 01:32:15 ERROR lzo.GPLNativeCodeLoader: Could not load native
gpl library
java.lang.UnsatisfiedLinkError: no gplcompression in java.library.path
        at java.lang.ClassLoader.loadLibrary(ClassLoader.java:1734)
        at java.lang.Runtime.loadLibrary0(Runtime.java:823)
        at java.lang.System.loadLibrary(System.java:1028)
```

就意味着你的压缩库出现了问题。

13.9.2. 运行时错误

- `java.io.IOException...` 打开太多文件(Too many open files) 如果看到如下消息…

```
2010-09-13 01:24:17,336 WARN
org.apache.hadoop.hdfs.server.datanode.DataNode:
Disk-related IOException in BlockReceiver constructor. Cause is
java.io.IOException: Too many open files
    at java.io.UnixFileSystem.createFileExclusively(Native Method)
    at java.io.File.createNewFile(File.java:883)
```

- `xeceiverCount 258 exceeds the limit of concurrent xcievers 256`

这个时常会出现在DataNode的日志中。

- 系统不稳定, DataNode或者其他系统进程有 “`java.lang.OutOfMemoryError : unable to create new native thread in exceptions`”的错误

Linux发行版缺省值是1024——这对Transwarp HBase实在太小了。

- DFS不稳定或者RegionServer租期超时

如果你收到了如下的消息：

```
2009-02-24 10:01:33,516 WARN org.apache.hadoop.hbase.util.Sleeper: We
slept xxx ms, ten times longer than scheduled: 10000
2009-02-24 10:01:33,516 WARN org.apache.hadoop.hbase.util.Sleeper: We
slept xxx ms, ten times longer than scheduled: 15000
2009-02-24 10:01:36,472 WARN
org.apache.hadoop.hbase.regionserver.HRegionServer: unable to report to
master for xxx milliseconds - retrying
```

…或者看到了全GC压缩操作，你可能正在执行一个全GC。

"No live nodes contain current block" and/or YouAreDeadException 这个错误有可能是OS的文件句柄溢出，也可能是网络故障导致节点无法访问。

- ZooKeeper会话超时 (`timeout`)

Master或RegionServers关闭时在日志中出现如下提示信息：

```

WARN org.apache.zookeeper.ClientCnxn: Exception
closing session 0x278bd16a96000f to sun.nio.ch.SelectionKeyImpl@355811ec
java.io.IOException: TIMED OUT
    at
org.apache.zookeeper.ClientCnxn$SendThread.run(ClientCnxn.java:906)
WARN org.apache.hadoop.hbase.util.Sleeper: We slept 79410ms, ten times
longer than scheduled: 5000
INFO org.apache.zookeeper.ClientCnxn: Attempting connection to server
hostname/IP:PORT
INFO org.apache.zookeeper.ClientCnxn: Priming connection to
java.nio.channels.SocketChannel[connected local=/IP:PORT
remote=hostname/IP:PORT]
INFO org.apache.zookeeper.ClientCnxn: Server connection successful
WARN org.apache.zookeeper.ClientCnxn: Exception closing session
0x278bd16a96000d to sun.nio.ch.SelectionKeyImpl@3544d65e
java.io.IOException: Session Expired
    at
org.apache.zookeeper.ClientCnxn$SendThread.readConnectResult(ClientCnxn.
java:589)
    at
org.apache.zookeeper.ClientCnxn$SendThread.doIO(ClientCnxn.java:709)
    at
org.apache.zookeeper.ClientCnxn$SendThread.run(ClientCnxn.java:945)
ERROR org.apache.hadoop.hbase.regionserver.HRegionServer: ZooKeeper
session expired

```

JVM正在做一个耗时较长的GC过程，从而中止了所有的线程。由于RegionServer的本地ZooKeeper客户端不能发送heartbeats，因而会话超时了。

- 保证您提供了足够的RAM（在 `hbase-env.sh` 中），默认的1GB的大小会不足以保证长时的输入。
- 保证您没有swap，JVM在swap下表现很差。
- 保证RegionServer线程不会被饿死。例如，当你用6个CPU密集型任务在一台四核机器上运行一个MapReduce工程，RegionServer线程可能已经没有资源去创建更长的GC暂停了。
- 延长ZooKeeper的会话超时时限

如果你希望延长ZooKeeper的会话超时，在 `hbase-site.xml` 中将默认的60s超时设定，延长为120s。

```

<property>
  <name>zookeeper.session.timeout</name>
  <value>1200000</value>
</property>
<property>
  <name>hbase.zookeeper.property.tickTime</name>
  <value>6000</value>
</property>

```

注意，将超时调到一个较高的水平意味着，由一个故障的RegionServer管理的region需要至少这么长的时间来转移到另一个RegionServer上。对于现场服务请求的生产系统，我们反而建议将它设为低于1分钟，以及调整集群以减少每台机器上的内存负载（这样每台机器上会有很少的垃圾需要收集）。

如果这种情况只在某次上传过程中发生了一次，那么可以考虑是不是批量加载的问题。

- 无服务区域异常(**NotServingRegionException**)

在DEBUG级别的RegionServer日志发现这个报错是很常见的。这个异常被返回到客户端，然后在客户端回到 **hbase:meta** 来找到移动过的region的新位置。

然而，如果 **NotServingRegionException** 在日志中记录下来，那么客户端可能会耗尽重试次数，还可能出现其他问题。

- 区域列示先是域名，然后IP

修复DNS。在HBase0.92.x以前的版本，反向DNS需要和正向查询相同答案。

- 大量类似 **2011-01-10 12:40:48,407 INFO org.apache.hadoop.io.compress.CodecPool: Got brand-new compressor** 日志消息

没有采用本地压缩库版本。从hadoop的HBase库目录复制本地库或建立链接到正确位置，该消息将消失。

- Server handler X on 60020 caught: **java.nio.channels.ClosedChannelException**

13.9.3. 由反向DNS引起的快照问题

Transwarp HBase内的一些操作，包括快照等，依赖于被合理配置的反向DNS。在某些环境下，如Amazon EC2，会在反向DNS上出出现一些问题。如果你看到RegionServer上有如下的报错，可以检查您的反向DNS配置：

```
2013-05-01 00:04:56,356 DEBUG
org.apache.hadoop.hbase.procedure.Subprocedure: Subprocedure 'backup1'
coordinator notified of 'acquire', waiting on 'reached' or 'abort' from
coordinator.
```

通常来看，RegionServer报出的主机名（ **hostname** ）应该和它要连接的Master主机名相同。您可以通过寻找在启动时在该RegionServer记录下的以下消息来发现主机名不匹配的问题：

```
2013-05-01 00:03:00,614 INFO
org.apache.hadoop.hbase.regionserver.HRegionServer: Master passed us
hostname
to use. Was=myhost-1234, Now=ip-10-55-88-99.ec2.internal
```

13.9.4. 终止错误

13.10. Master

13.10.1. 启动错误

- Master提示你需要运行Transwarp HBase迁移脚本

在运行脚本之前，Transwarp HBase迁移脚本提示说根目录下没有文件。Transwarp HBase期望根目录要么不存在，要么在上一次Transwarp HBase运行的时候被初始化好了。如果您想要用 Hadoop DFS 来为Transwarp HBase创建一个新的目录，那么就会报这个错。确保根目录要么不存在，要么在上一次Transwarp HBase运行

的时候被初始化好了。确保只是用Hadoop DFS来删除Transwarp HBase根目录，或是让Transwarp HBase自己来创建或者初始化目录。

13.10.2. 终止错误

13.11. ZooKeeper

13.11.1. 启动错误

- 找不到地址: `xyz in list of ZooKeeper quorum servers`

这是一个名称查找问题。Transwarp HBase想要在某个机器上启动ZooKeeper，但是这台机器找不到自己的 `hbase.zookeeper.quorum` 配置值。

用报错信息中出现的主机名，来替代您之前用的。如果你有一个DNS服务器，你可以在 `hbase-site.xml` 中设定好 `hbase.zookeeper.dns.interface` 以及 `hbase.zookeeper.dns.nameserver` 来确保它能被正确解析为FQDN。

工具`zkcli`可以帮助调查 ZooKeeper 问题。

14. Transwarp HBase工具

前面章节介绍了Transwarp HBase的基础知识与基本操作，下面给出几种当Transwarp HBase出现异常或要执行特殊功能时所需的一些工具：

14.1. 分布式存储运维工具（DSTools）

两个工具都在DSTools-1.0.0-transwarp项目中，目录结构如下：

```
DSTools-1.0.0-transwarp
├── conf
│   ├── core-site.xml
│   ├── hbase-site.xml
│   ├── hdfs-site.xml
│   └── log4j.properties
├── DSTools-1.0.0-transwarp.jar
├── runDSTools.sh
└── runHFileCheck.sh
```

14.1.1. 工具修复常见异常情况：

- 从TDH管理界面中找到hbase active master角色，点击link进入hbase active master 60010页面。下拉到页面最下面“Regions in Transition”项目中，如果有region出现在此，并且“RIT time (ms)”这一项显示时间已经很久了，比如1到两个小时的，此时可以运行此runDSTools.sh工具或者runHFileCheck.sh工具进行修复。详细操作步骤见下一节。



The screenshot shows a table titled "Regions in Transition". It has three columns: "Region", "State", and "RIT time (ms)". There is one row highlighted in red, which corresponds to the first item in the list above. The "Region" column contains the value "63f0759ae4f4731de5b059a0783df3e11". The "State" column contains the value "state=FAILED_OPEN, ts=Tue Apr 19 15:51:12 CST 2016 (21735s ago) server=jzsd-dm". The "RIT time (ms)" column contains the value "21735023". A note at the bottom of the table says "Total number of Regions in Transition for more than 60000 milliseconds" followed by the number "1".

Region	State	RIT time (ms)
1. region即是region的name	state=FAILED_OPEN, ts=Tue Apr 19 15:51:12 CST 2016 (21735s ago) server=jzsd-dm	21735023
Total number of Regions in Transition for more than 60000 milliseconds		
1		
Total number of Regions in Transition		
1		

- 当Transwarp HBase集群因meta表损坏启动不了时，运行runDSTools.sh工具进行Transwarp HBase彻底的元信息重建更新，然后再启动集群。详细操作步骤见下一节。
- 当出现Transwarp HBase一张表查询不了时，且hbase active master的60010页面中的“Regions in Transition”也没有显示任务region卡住的现象时，运行runDSTools.sh工具。详细操作步骤见下一节。

在工具检查前，需要确保hdfs正常，可用如下命令：

1. 从TDH管理界面中找到hdfs中active namenode角色，点击link进入检查节点是否处于safemode；保证safemode处于关闭状态，即“Safemode is off”。
2. sudo -u hdfs hdfs dfsadmin -report //make sure all datanode is health
3. sudo -u hbase hdfs fsck /hyperbase1 -files -locations -blocks //make sure no hdfs block was missing or corrupt
4. sudo -u hbase hdfs fsck /hyperbase1 -list-corruptfileblocks //also you can use this to find out corruptfile. If there are files corrupted, check it out before running this tools.

14.1.2. 工具操作步骤说明：

首先将此工具拷到集群任一节点中的tmp文件夹下，若拷贝到其他文件夹下可能出现权限问题，也可更改其他文件夹的权限来将工具拷贝进去；下图为因权限限制而报错：

```
[root@transwarp-perf2 DSTools-1.0.0-transwarp]# sh runHFileCheck.sh /hyperbase1/data/default/yifeitest >
Exception in thread "main" java.lang.NoClassDefFoundError: io/transwarp/hbase/util/HyperbaseRegionFsck
Caused by: java.lang.ClassNotFoundException: io.transwarp.hbase.util.HyperbaseRegionFsck
        at java.net.URLClassLoader$1.run(URLClassLoader.java:217)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java:205)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:323)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:294)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:268)
Could not find the main class: io.transwarp.hbase.util.HyperbaseRegionFsck. Program will exit.
[root@transwarp-perf2 DSTools-1.0.0-transwarp]#
```

然后找到当前节点的/etc/hyperbase1/conf, /etc/hdfs1/conf 下的对应hbase-site.xml/core-site.xml/hdfs-site.xml，用它们替换工具目录下的conf目录下的对应文件；这样即可让工具知道它作用的对象和环境。

接着：

14.1.3. runDSTools.sh工具操作步骤：

- 运行前提：

1. 如果Transwarp HBase集群本身起动不了，则保证所有Transwarp HBase所有角色服务都关闭。

图为启动不了集群，需将所有角色关闭：

HYPERBASE | ● DOWN (Details)

More Actions ·

Home > Hyperbase

Search Role...

+

Role Name	Node Name	Rack Name	Service Link	Health	Operations
region server (transwarp-perf1)	transwarp-perf1	/1	Link	● Not Running	▶ ■ ×
region server (transwarp-perf2)	transwarp-perf2	/1	Link	● Not Running	▶ ■ ×
region server (transwarp-perf3)	transwarp-perf3	/2	Link	● Not Running	▶ ■ ×
region server (transwarp-perf4)	transwarp-perf4	/2	Link	● Not Running	▶ ■ ×
master (transwarp-perf2)	transwarp-perf2	/1	Link	● Not Running	▶ ■ ×
master (transwarp-perf3)	transwarp-perf3	/2	Link	● Not Running	▶ ■ ×
master (transwarp-perf4)	transwarp-perf4	/2	Link	● Not Running	▶ ■ ×
chronos server (transwarp-perf4)	transwarp-perf4	N/A		● Not Running	▶ ■ ×

2. 如果Transwarp HBase集群已经启动正在运行，则保证所有regionserver都是在线存活状态，并且至少有两个及以上的master角色在线存活状态。Transwarp HBase集群角色的运行状态可以从TDH管理界面中的Transwarp HBase组件角色一项中查看到。

HYPERBASE | ● HEALTHY

More Actions ·

Home > Hyperbase

Search Role...

+

Role Name	Node Name	Rack Name	Service Link	Health	Operations
region server (transwarp-perf1)	transwarp-perf1	/1	Link	● Running	▶ ■ ×
region server (transwarp-perf2)	transwarp-perf2	/1	Link	● Running	▶ ■ ×
region server (transwarp-perf3)	transwarp-perf3	/2	Link	● Running	▶ ■ ×
region server (transwarp-perf4)	transwarp-perf4	/2	Link	● Running	▶ ■ ×
master (transwarp-perf2)	transwarp-perf2	/1	Link	● Running	▶ ■ ×
master (transwarp-perf3)	transwarp-perf3	/2	Link	● Running	▶ ■ ×
master (transwarp-perf4)	transwarp-perf4	/2	Link	● Running	▶ ■ ×
chronos server (transwarp-perf4)	transwarp-perf4	N/A		● Running	▶ ■ ×

- 运行操作：

- 在工具所在的目录下，直接 `sh runDSTools.sh 1>&2 2> /tmp/dstool.log.1` 运行即可；
- 查看工具运行输出的日志信息 `vi /tmp/dstool.log.1`，查看最后的输出日志提示。如果运行时，集群是在线模式，正确的日志应该显示 `inconsistance` 为0。

下图后两行显示修复完成：

```
16/07/06 15:15:07 INFO zookeeper.ClientCnxn: Opening socket connection to server transwarp-perf2/172.16.1.62:2181. Will not attempt to authenticate using SASL (unknown error)
16/07/06 15:15:07 INFO zookeeper.ClientCnxn: Socket connection established to transwarp-perf2/172.16.1.62:2181, initiating session
16/07/06 15:15:07 INFO zookeeper.ClientCnxn: Session establishment complete on server transwarp-perf2/172.16.1.62:2181, sessionid = 0x155018437a2bf80, negotiated timeout = 180000
16/07/06 15:15:07 INFO client.HConnectionManager$HConnectionImplementation: Closing master protocol: MasterService
16/07/06 15:15:07 INFO client.HConnectionManager$HConnectionImplementation: Closing zookeeper sessionid=0x155018437a2bf71
16/07/06 15:15:07 INFO zookeeper.ZooKeeper: Session: 0x155018437a2bf71 closed
16/07/06 15:15:07 INFO zookeeper.ClientCnxn: EventThread shut down
16/07/06 15:15:07 WARN util.HyperbaseFsck: step last fix meta & deploy finished !
16/07/06 15:15:07 WARN util.HyperbaseFsck: enable balance true !
"/tmp/loa_5" 12927l 3611163C 12927.1 Bot
```

下图后两行表示此为正确状态下日志：

```

sl_ifs_crbs_corp_cons is okay.
  Number of regions: 2
    Deployed on: transwarp-perf1,60020,1467783446694 transwarp-perf3,60020,1467783438684
first_input_batch_detail is okay.
  Number of regions: 8
    Deployed on: transwarp-perf1,60020,1467783446694 transwarp-perf2,60020,1467783438327 transwarp-perf3,60020,1467783438684 transwarp-perf4,60020,1467783438434
hyper_simple_table_global_single is okay.
  Number of regions: 1
    Deployed on: transwarp-perf3,60020,1467783438684
yifeitmp is okay.
  Number of regions: 1
    Deployed on: transwarp-perf2,60020,1467783438327
0 inconsistencies detected.
Status: OK

```

如果集群本身是启动不了的情况，运行后，它会提示你 可以在TDH管理界面中启动Transwarp HBase集群的信息。

```

16/07/07 10:43:21 INFO regionserver.HStore: Closed info
16/07/07 10:43:21 INFO regionserver.HRegion: Closed hbase:meta,,1.1588230740
16/07/07 10:43:21 INFO wal.FSHLog: Thread-0-WAL.AsyncNotifier exiting
16/07/07 10:43:21 INFO wal.FSHLog: Thread-0-WAL.AsyncSyncer0 exiting
16/07/07 10:43:21 INFO wal.FSHLog: Thread-0-WAL.AsyncSyncer1 exiting
16/07/07 10:43:21 INFO wal.FSHLog: Thread-0-WAL.AsyncSyncer2 exiting
16/07/07 10:43:21 INFO wal.FSHLog: Thread-0-WAL.AsyncSyncer3 exiting
16/07/07 10:43:21 INFO wal.FSHLog: Thread-0-WAL.AsyncSyncer4 exiting
16/07/07 10:43:21 INFO wal.FSHLog: Thread-0-WAL.AsyncWriter exiting
16/07/07 10:43:21 INFO util.HBaseFsck: Success! hbase:meta table rebuilt.
16/07/07 10:43:21 INFO util.HBaseFsck: Old hbase:meta is moved into /tmp
16/07/07 10:43:21 INFO util.HyperbaseFsck: SUCCESS !!! all offline fix are finished ! now you can restart hyperbase cluster by using web manager!

```

如果日志不是以上两种，则反馈给客服，把 `dstool.log.1` 文件发给客服，并描述观察到的现象即可。如：

```

16/07/06 14:12:02 INFO zookeeper.RecoverableZooKeeper: Process identifier=hbase Fsck connecting to ZooKeeper ensemble=transwarp-perf1:2181,transwarp-perf2:2181,transwarp-perf3:2181
16/07/06 14:12:02 INFO zookeeper.ZooKeeper: Initiating client connection, connectString=transwarp-perf2:2181,transwarp-perf3:2181,baseZNode=/hyperbase1
16/07/06 14:12:02 INFO zookeeper.ClientCnxn: Opening socket connection to server transwarp-perf1/172.16.1.61:2181. Will attempt to negotiate version 3
16/07/06 14:12:02 INFO zookeeper.ClientCnxn: Socket connection established to transwarp-perf1/172.16.1.61:2181, initiating session
16/07/06 14:12:02 INFO zookeeper.ClientCnxn: Session establishment complete on server transwarp-perf1/172.16.1.61:2181, session id: 0x550184490fa9b6
16/07/06 14:12:02 INFO client.HConnectionManager$HConnectionImplementation: Closing master protocol: MasterService
16/07/06 14:12:02 INFO client.HConnectionManager$HConnectionImplementation: Closing zookeeper sessionid=0x550184490fa9b6
16/07/06 14:12:02 INFO zookeeper.ZooKeeper: Session: 0x550184490fa9b6 closed
16/07/06 14:12:02 INFO zookeeper.ClientCnxn: EventThread shut down
16/07/06 14:12:02 WARN util.HyperbaseFsck: step NO.4 fix overlaps failed!
16/07/06 14:12:02 WARN util.HyperbaseFsck: Hyperbase HDFS Integrity check failed ! all fix action will not go on "/tmp/Log.2" 6962L, 907218C

```

14.1.4. runHFileCheck.sh工具操作步骤：

- 运行前提：

Transwarp HBase集群为在线模式，即至少有一个master存活，至少有一个regionserver存活。这些角色是否存活可以在TDH管理界面Transwarp HBase组件的角色栏中查看到。

HYPERBASE | ● HEALTHY

Home > Hyperbase1

Search Role...

Role Name	Node Name	Rack Name	Service Link	Health	Operations
region server (transwarp-perf1)	transwarp-perf1	/1	Link	● Running	▶ ■ ×
region server (transwarp-perf2)	transwarp-perf2	/1	Link	● Running	▶ ■ ×
region server (transwarp-perf3)	transwarp-perf3	/2	Link	● Running	▶ ■ ×
region server (transwarp-perf4)	transwarp-perf4	/2	Link	● Running	▶ ■ ×
master (transwarp-perf2)	transwarp-perf2	/1	Link	● Running	▶ ■ ×
master (transwarp-perf3)	transwarp-perf3	/2	Link	● Running	▶ ■ ×
master (transwarp-perf4)	transwarp-perf4	/2	Link	● Running	▶ ■ ×
chronos server (transwarp-perf4)	transwarp-perf4	/2	N/A	● Running	▶ ■ ×

- 运行操作:

- 先运行一次命令 `sh runHFileCheck.sh tableName 1>&2 2> /tmp/hfilecheck.log.1` 检查看是否有hfile有损坏或无效引用存在，此信息会在工具运行的日志中输出。

若无损坏:

```
***** 1. Start checking all hfiles for corruption *****
*****
***** Finish corrupted hfiles checking *****
Checked 2 hfile for corruption
    HFiles corrupted:          0
    HFiles moved while checking: 0
Summary: OK
#####
##### 2. Start detecting all invalid hfilelinks #####
#####
##### Finish invalid reference checking #####
##### total invalid reference 0
Summary: OK
#####
[root@transwarp-perf2 DSTools-1.0.0-transwarp]#
```

否则:

```
Caused by: java.lang.IllegalArgumentException: Invalid HFile version: 6976266 (expected to be between 2 and 3)
    at org.apache.hadoop.hbase.io.hfile.HFile.checkFormatVersion(HFile.java:824)
    at org.apache.hadoop.hbase.io.hfile.FixedFileTrailer.readFromStream(FixedFileTrailer.java:402)
    at org.apache.hadoop.hbase.io.hfile.HFile.pickReaderVersion(HFile.java:462)
    ... 16 more
***** Finish corrupted hfiles checking *****
Checked 2 hfile for corruption
    HFiles corrupted:          1
    HFiles moved while checking: 0
Summary: CORRUPTED
#####
##### 2. Start detecting all invalid hfilelinks #####
#####
##### Finish invalid reference checking #####
##### total invalid reference 0
Summary: OK
#####
```

- 运行命令修复它 `sh runHFileCheck.sh -fixReferenceFiles -sidelineCorruptHFiles tableName 1>&2 2> /tmp/hfilecheck.log.2`

修复完成:

```

16/07/06 15:40:25 WARN hbck.HFileCorruptionChecker: Quarantining corrupt HFile hdfs://service/hyperbase1/data/def
1f135520 into hdfs://service/hyperbase1/corrupt/test/4284a41b078db95b124f1d1ac61aa228/f/b717ed4d60bd4cc29c4cb61b1
***** Finish corrupted hfiles checking *****
Checked 2 hfile for corruption
HFiles corrupted:      1
    HFiles successfully quarantined: 1
        hdfs://service/hyperbase1/corrupt/test/4284a41b078db95b124f1d1ac61aa228/f/b717ed4d60bd4cc29c4cb61b1f135520
    HFiles failed quarantine:      0
    HFiles moved while checking:   0
Summary: CORRUPTED => OK
*****
#####
##### 2. Start detecting all invalid hfilelinks #####
#####
##### Finish invalid reference checking #####
#####
##### total invalid reference 0
##### sideline reference successful:
##### failed to sideline reference:
Summary: OK
#####

```

3. 再次运行命令验证是否已经修复 `sh runHFileCheck.sh tableName 1>&2 2>/tmp/hfilecheck.log.3`。

文件正常：

```

***** 1. Start checking all hfiles for corruption *****
16/07/06 15:40:58 INFO hfile.CacheConfig: Allocating LruBlockCache with maximum size 1.4 G
16/07/06 15:40:58 INFO util.ChecksumType: Checksum using org.apache.hadoop.util.PureJavaCrc32
16/07/06 15:40:58 INFO util.ChecksumType: Checksum can use org.apache.hadoop.util.PureJavaCrc32C
***** Finish corrupted hfiles checking *****
Checked 1 hfile for corruption
HFiles corrupted:      0
    HFiles moved while checking:   0
Summary: OK
#####
##### 2. Start detecting all invalid hfilelinks #####
#####
##### Finish invalid reference checking #####
#####
##### total invalid reference 0
Summary: OK
#####

```

如果发现输出日志信息还显示有没被fix的hfile，请联系客服，并把此 `hfilecheck.log.1`、`hfilecheck.log.2`、`hfilecheck.log.3` 发给客服，并描述观察到的现象即可。



runHFile/runDSTools工具运行结束后，可能会由于master cache的清空而导致master被关闭。此时需要在TDH管理界面，把被工具关闭的master再手动启动起来。

14.2. 在Transwarp HBase集群之间迁移数据

跨Transwarp HBase集群迁移数据有三种方法：`import/export` 数据方法、快照方法和 `CopyTable` 方法。假设我们有两个Transwarp HBase集群：集群h1和集群h2，它们分别使用的HDFS服务为ha1和ha2。我们想要将集群h1上的表t1迁移到集群h2上。下面我们介绍如何分别用这三种迁移方法完成这个操作。

14.2.1. 用 `import/export` 迁移数据

import/export 操作需要以hbase用户身份执行。

- 如果您的集群不处于安全模式下（没有开启Kerberos），那么您需要以操作系统中hbase用户的身份执行指令，方法为在指令前加上 **sudo -u hbase**。如下面第一步的 **export** 操作指令需要写为：

```
sudo -u hbase hbase
org.apache.hadoop.hbase.mapreduce.Driver export t1
hdfs://ha1/tmp/t1
```



- 如果您的集群处于安全模式下（开启了Kerberos），那么您需要在执行指令前作为hbase用户取票，方法为执行：

```
kinit -kt /etc/<hbase_service_name>/hbase.keytab
hbase<host>
```

这里 **<hbase_service_name>** 为对应Transwarp HBase服务的服务名。**<host>** 为执行操作节点的hostname。

- 将t1中的数据导出（**export**）到ha1上的/tmp下的一个目录，如/tmp/t1：

```
hbase org.apache.hadoop.hbase.mapreduce.Driver export t1
hdfs://ha1/tmp/t1
```

- 将上一步拷贝到hdfs1上的数据copy到ha2上的/tmp下的一个目录，假设还是/tmp/t1。
- 在h2上创建t1_new，保证和h1中的t1结构一致（可以参考[JSON配置迁移扩展元数据](#)来快速迁移表的（扩展）元数据。）
- 将ha1的数据导入（**import**）到h2中的t1_new下面：

```
hbase org.apache.hadoop.hbase.mapreduce.Driver import t1_new
hdfs://ha2/tmp/t1
```

如果涉及到不同版本的Transwarp HBase之间迁移，在 **import** 时需要提供集群1的HBase版本号：

```
hbase -Dhbase.import.version=0.94
org.apache.hadoop.hbase.mapreduce.Driver import t1_new hdfs://ha2/tmp/t1
```

- 迁移结束，h2中的t1_new为迁移后的表。

14.2.2. 用快照迁移数据



更多快照命令细节请参考[Transwarp HBase Shell快照相关命令](#)。

- 使表t1下线

```
disable 't1'
```

- 在h1的Transwarp HBase Shell中为表t1生成快照:

```
snapshot 't1', 'snapshot-t1'
```

- 将快照 **snapshot-t1** 导出到ha2的hbase根目录中:

```
hbase org.apache.hadoop.hbase.snapshot.ExportSnapshot -snapshot
snapshot-t1 -copy-to hdfs://ha2:8082/hbase -mappers <n>
```

您需要将 **<n>** 设置为导出快照使用的mapper数量, 比如:

```
hbase org.apache.hadoop.hbase.snapshot.ExportSnapshot -snapshot
snapshot-t1 -copy-to hdfs://ha2:8082/hbase -mappers 16
```

- 在h2的Transwarp HBase Shell中使用 **snapshot-t1** 建一张新表t1_new:

```
clone_snapshot 'snapshot-t1', 't1_new'
```

- 迁移结束, h2中的t1_new为迁移后的新表。

14.2.3. 直接 **CopyTable** 迁移数据

CopyTable是一种实用工具, 它能用来拷贝部分或者全部的表, 无论这些表是在同一集群还是不同集群。要求目标操作的表必须存在, 具体用法如下:

```
/bin/hbase org.apache.hadoop.hbase.mapreduce.CopyTable --help
Usage: CopyTable [general options] [--starttime=X] [--endtime=Y] [--
new.name=NEW] [--peer.adr=ADR] <tablename>
```

选项 :

- startrow** 起始行;
- stoprow** 结束行;
- starttime** 起始时间 (毫秒级unixtime), 没有设定结束时间意味着永远执行;
- endtime** 结束时间, 若没有明确指定起始时间, 可忽略该选项;

- **versions** 指定拷贝的单元版本数;
- **new.name** 新表名;
- **peer.adr** 目标对等集群地址（实为Zookeeper地址），采用如下格式：

hbase.zookeeper.quorum:hbase.zookeeper.client.port:zookeeper.znode.parent ;

- **families** 需拷贝的列族列表，用逗号分隔；

若要从cf1拷贝到cf2，应设为 sourceCfName:destCfName；

若要用同一个表名，直接设为 “cfName”；

- **all.cells** 另拷贝带删除标记的数据和未回收的删除单元（高级选项）。

参数：

- 待拷贝的表的表名。

从优化性能的角度，还可考虑如下选项的设置：

- 建议将下值设为 ≥ 100 的值。较大的值虽然会占用更多的内存，但减少了往返时延（round trip time），从而提升了性能：

`-Dhbase.client.scanner.caching=100`

- 下面这条语句应该总被设置为false，以防数据被多次写入而产生错误结果：

`-Dmapred.map.tasks.speculative.execution=false`

将 **TestTable** 拷贝到某个集群，该集群是1小时时间窗口区间段的备份：

```
hbase org.apache.hadoop.hbase.mapreduce.CopyTable
-Dhbase.client.scanner.caching=100
-Dmapred.map.tasks.speculative.execution=false --startrow=rk1
--stoprow=rk4 --starttime=1265875194289 --endtime=1265878794289
--peer.adr=transwarp-perf1,transwarp-perf2,transwarp
-perf3:2181:/hyperbase1 --new.name=TestTableNew --families=cf1:cf2
TestTable
```

上例中，transwarp-perf1-3表示hbase.zookeeper.quorum，即配置有Zookeeper的服务器；2181表示hbase.zookeeper.client.port，即Zookeeper的端口；/hyperbase1表示zookeeper.znode.parent，即Zookeeper根节点位置。TestTableNew从TestTable拷贝cf1列族中一定时间和行键范围的数据，并保存在自己的cf2列族。



- 在指定startrow和stoprow时要注意，选择的范围是[startrow, stoprow) **左闭右开** 区间；
- 默认情况下copy的数据是最新版本的数据，如果想要拷贝被打上删除标记的数据，则需要加上 **--all.cells**； 若要指定版本，可加上 **--versions=n**。

14.3. SQL BulkLoad

本章介绍如何使用我们提供的SQL BulkLoad工具向HBase表中批量导入大量数据。SQL BulkLoad绕过 **put** API直接将数据写入表下的HFile中，避免了用 **put** API写数据的性能问题。SQL BulkLoad的适用场景为：

- 初次原始数据导入。
- 增量数据导入。

假设我们有一个数据集，SQL BulkLoad工具将其导入HBase表的流程包括以下几步：

1. 将数据集上传至HDFS（该步骤可选）。
2. 在Inceptor Engine中建外表指向HDFS上的数据集。
3. 目标HBase表需要预分Region。如果您已知将要建的HBase表的Split Key，可以直接将其提供给SQL BulkLoad工具。如果您没有已知Split Key，您可以通过设置让SQL BulkLoad工具为您生成Split Key，分为两步：
 - a. 对外表进行取样；
 - b. 利用取样结果生成一张Split Key表。
4. 在Inceptor Engine中建HBase表的映射表，使用上一步计算出的Split Key预分Region。
5. 将Inceptor Engine中使用SQL BulkLoad语句将数据从第2步中的表中插入HBase映射表。
6. 操作结束。

我们针对向HBase表导数据提供了一套SQL BulkLoad工具来执行上述步骤，这套工具是一个包含了一系列脚本的目录，工具的大致使用步骤为：

1. 根据您的使用需求修改目录中的config.txt文件，对上述的SQL BulkLoad流程进行设置，例如配置目标HBase表的名称、Region数量、数据文件地址等。
2. 运行目录下的run.sh脚本，脚本会自动生成一系列SQL BulkLoad自动生成上述任务的SQL并依次执行它们，run.sh脚本运行结束后您的数据也就成功导入HBase表中了。

在使用SQL BulkLoad工具前，您需要准备好您的数据集。您可以选择将数据集放在HDFS上，也可以放在本地（您可以设置让SQL BulkLoad工具帮您将数据放到HDFS上）。您还需要根据您的需求计算好目标HBase表的Region数量。

14.3.1. HBase SQL BulkLoad



HBase SQL BulkLoad工具暂时不支持decimal和struct类型的数据的导入。

用一个文本编辑器打开目录下的config.txt文件，修改或检查下列参数：

- **TranswarCommand:** 根据您使用Hive Server 1还是Hive Server 2修改:

- 使用Hive Server 1: `transwarp -t -h <host> -f`
- 使用Hive Server 2: `beeline -u jdbc:hive2://<host>:10000/default -f`

- **Hdfsdir** 设置为您的数据在HDFS上的目录, 例如 `/user/xiefeng/data/udft/data`

如果您的数据不在HDFS上, 您可以手动先将数据 `put` 上去, 也可以靠设置**LocalInputDir**来让SQL BulkLoad工具替您将数据放到HDFS上。

- **LocalInputDir:** 数据集在本地的路径, 例如: `./data/*`。该参数为选填参数, 提供该参数后, SQL BulkLoad会将该路径中的数据放到HDFS上。如果您不需要SQL BulkLoad执行这项操作, 只需将这个参数值留为空, SQL BulkLoad会直接使用**Hdfsdir**对应目录中的数据。
- **ExternalTableDatabase:** SQL BulkLoad在Inceptor Engine中使用的数据库。
- **ExternalTableName:** Inceptor Engine中指向HDFS上的数据集的外表的名称。
- **ExternaltableParameter:** 指向HDFS上的数据集的外表的列名和数据类型, 例如 `id int, name string, users string`。
- **HbaseTableName:** 目标HBase表的表名, 同时也是它在Inceptor Engine中的映射表的表名。
- **NameSpace:** 目标HBase表所在的Namespace。
- **TableParameter:** 目标HBase表的列名和数据类型, 例如 `id int, name string, users string`。
- **KeyFields:** 生成Split Key所用的列以及它的数据类型, 例如 `id:int`
- **HbaseSelectedColumn:** 指定从HDFS外表中导哪几列数据进入目标HBase表中。
- **SplitKeys:** 如果您已知用于预分目标HBase表的Split Key, 在这里提供, 那么SQL BulkLoad工具将会直接使用您提供的Split Key, 而不会生成Split Key。
- **RegionNum:** 目标HBase表的Region数目。
- 下面几个参数指定HDFS外表的字段分隔符, 不指定则使用Inceptor Engine的默认值:
 - **ExternalTableFieldsTerminatedBy:** 列分隔符, 例如 `,`
 - **ExternalTableCollectionItemsTerminatedBy:** array类型元素分隔符和map类型键值对之间的分隔符, 例如 `\001`
 - **ExternalTableMapKeysTerminatedBy:** map类型的键和值之间的分隔符, 例如 `\003`
 - **ExternalTableLinesTerminatedBy:** 换行符, 例如 `\n`。
- **ExternalTableStoredAs:** HDFS上数据文件的格式, 例如 `TEXTFILE`。
- 下面几个参数指定HBase映射表的字段分隔符(注意, 目标HBase映射表和HDFS外表使用的是同一份数据文件, HBase映射表和HDFS外表使用的对应分隔符应该相同):
 - **HbaseTableFieldsTerminatedBy:** 列分隔符, 例如 `,`
 - **HbaseTableCollectionItemsTerminatedBy:** array类型元素分隔符和map类型键值对之间的分隔符, 例如 `\001`
 - **HbaseTableMapKeysTerminatedBy:** map类型的键和值之间的分隔符, 例如 `\003`
 - **HbaseTableLinesTerminatedBy:** 换行符, 例如 `\n`。

- **HbaseFormat:** 映射表的Storage Handler。HBase映射表的Storage Handler为 `org.apache.hadoop.hive.hbase.HBaseStorageHandler`。HBase SQL BulkLoad中的config.txt应该已经设置好该值，您无需修改。

例 112. HBase SQL BulkLoad的示例config.txt

```

#-----
#You must specify these variables

#command used to send sql file to hive server1 or hive server 2
TranswarpCommand:=beeline -u jdbc:hive2://172.16.1.200:10000/default -f

#Local directory of files you want to send to HDFS, if null, it will
use the HDFS files
LocalInputDir:=./data/*

#The hdfs directory you want to load data into Hbase table
Hdfsdir:=/user/xiefeng/data/udft/data

#database
ExternalTableDatabase:=sqlbulkload
HbaseTableDatabase:=sqlbulkload

#ExternalTableDatabase:=default
#HbaseTableDatabase:=default

#external table name
ExternalTableName:=externalTable

#external table column fields
ExternaltableParameter:=id int,name string,users string

#hbase mapped table name, also being the name of hbase table
HbaseTableName:=transwarp

#hbase namespace
NameSpace:=default

#hbase table column fields
TableParameter:=id int,name string,users string

#the columns you used to generate split keys. If you have give the
split
#keys, you also need to specify this parameter
KeyFields:=id:int

#extra udf attach to rowkey
Udf:=uniq()

#which columns you want to select from external table to hbase table
HbaseSelectedColumn:=id,name,users

#-----
#You'd better specify these variables
#If you give split keys, it will not generate by our UDTF
SplitKeys:=
-----
```

```

#the number of regions of the hbase table
RegionNum:=3

#External table separator
ExternalTableFieldsTerminatedBy:|,
ExternalTableCollectionItemsTerminatedBy:=\001
ExternalTableMapKeysTerminatedBy:=\003
ExternalTableLinesTerminatedBy:=\n
ExternalTableStoredAs:=TEXTFILE

#hbase table separator
HbaseTableFieldsTerminatedBy:|,
HbaseTableCollectionItemsTerminatedBy:=\001
HbaseTableMapKeysTerminatedBy:=\003
HbaseTableLinesTerminatedBy:=\n
HbaseFormat:=org.apache.hadoop.hive.hbase.HBaseStorageHandler

#-----
#If not necessary, you do not need to specify these variables
SampleTableName:=sampleTable
SampleTableDatabase:=sqlbulkload

```

完成对config.txt的修改以后，运行HBase SQL BulkLoad中的run.sh脚本：

sh run.sh

该脚本运行完成后，您的数据集会根据您在config.txt中的设置导入目标HBase表中，操作结束。

14.3.2. Hyperdrive SQL Bulkload



Hyperdrive SQL BulkLoad工具支持这decimal和struct这两种数据类型的导入。

用一个文本编辑器打开目录下的config.txt文件，修改或检查下列参数：

- **TranswarCommand:** 根据您使用Hive Server 1还是Hive Server 2修改：
 - 使用Hive Server 1: `transwarp -t -h <host> -f`
 - 使用Hive Server 2: `beeline -u jdbc:hive2://<host>:10000/default -f`
 - **Hdfsdir** 设置为您的数据在HDFS上的目录，例如 `/user/zhangyun/testcase_struct/`
- 如果您的数据不在HDFS上，您可以手动先将数据 `put` 上去，也可以靠设置**LocalInputDir**来让SQL BulkLoad工具替您将数据放到HDFS上。
- **LocalInputDir:** 数据集在本地的路径，例如：`./data/*`。该参数为选填参数，提供该参数后，SQL BulkLoad会将该路径中的数据放到HDFS上。如果您不需要SQL BulkLoad执行这项操作，只需将这个参数值留为空，SQL BulkLoad会直接使用**Hdfsdir**对应目录中的数据。
 - **ExternalTableDatabase:** HDFS外表在Inceptor Engine中所在的数据库。

- **HbaseTableDatabase**: Hyperdrive映射表在Inceptor Engine中所在的数据库。
- **SplitTmpTbl**: 用于存放SQL BulkLoad
- **ExternalTableName**: Inceptor Engine中指向HDFS上的数据集的外表的名称。
- **SplitTmpTbl**: 用于存放Hyperdrive SQL BulkLoad工具生成的Split Key的临时表，这张表只会被SQL BulkLoad工具本身用到，数据导入完成后将不再需要这张表。
- **ExternaltableParameter**: 指向HDFS上的数据集的外表的列名和数据类型，例如：

```
ti tinyint,si smallint,i int,bi bigint,f float,d1 double, dc
decimal(20,10),str varchar(30),ts string,tsgp string,strChina string,dt
string,dtgp string, bl boolean,st
struct<ti:tinyint,si:smallint,i:int,bi:bigint,f:float,d1:double,dc:decim
al(20,10),
str:varchar(30),ts:string,tsgp:string,strChina:string,dt:string,dtgp:str
ing, bl:boolean>
```

- **HbaseTableName**: 目标Hyperdrive表的表名，同时也是它在Inceptor Engine中的映射表的表名。
- **NameSpace**: 目标Transwarp HBase表所在的Namespace。
- **Hyperdrive_structstring_length_default**: 目标Hyperdrive表的struct列中string类型字段的默认Segment Length。
- **TableParameter**: 目标Hyperdrive表的schema（您无需在这里设置Row Key列的schema，Row Key的schema用**KeyFields**设置。），例如：

```
ti tinyint,si smallint,i int,bi bigint,f float,d1 double, dc
decimal(20,10),str varchar(30),ts string,tsgp string,strChina string,dt
string,dtgp string, bl boolean,st
struct<ti:tinyint,si:smallint,i:int,bi:bigint,f:float,d1:double,dc:decim
al(20,10),
str:varchar(30),ts:string,tsgp:string,strChina:string,dt:string,dtgp:str
ing, bl:boolean>
```

- **HbaseTableStructSegmentLen**: 目标Hyperdrive表中各个string类型字段的Segment length。指定struct类型列中的string字段的segment length时用 **<struct_name>.<field_name>** 来指代名称。例如：

st.tsgp:10,st.strChina:15,st.dtgp:9,st.ts:15,st.dt:18,tsgp:10

String类型列的segment length可以选择指定，不指定则使用默认值。Struct类型列中的string字段则必须指定segment length。

- 下面两个参数对Row Key进行设置：
 - **KeyFields**: Hyperdrive表的Row Key的schema，例如 **i:int**。如果Row Key是struct类型，只需指定多个字段，例如：

```
i:int,bi:bigint,
dl:double,dc:decimal(20,10),tsgp:string,strChina:string
```

- **FieldsSegmentLen**: 设置Row Key中string类型字段的Segment Length, 例如 `tsgp:14,strChina:12`。
- **HbaseSelectedColumn**: 指定从HDFS外表中导哪几列数据进入目标Transwarp HBase表中。
- **RegionNum**: 目标Hyperdrive表的Region数目。
- 下面几个参数指定HDFS外表的字段分隔符, 不指定则使用Inceptor Engine的默认值:
 - **ExternalTableFieldsTerminatedBy**: 列分隔符, 例如 `,`
 - **ExternalTableCollectionItemsTerminatedBy**: array类型元素分隔符和map类型键值对之间的分隔符, 例如 `\001`
 - **ExternalTableMapKeysTerminatedBy**: map类型的键和值之间的分隔符, 例如 `\003`
 - **ExternalTableLinesTerminatedBy**: 换行符, 例如 `\n`。
- **ExternalTableStoredAs**: HDFS上数据文件的格式, 例如 `TEXTFILE`。
- 下面几个参数指定Hyperdrive映射表的字段分隔符, 不指定则使用Inceptor Engine的默认值(注意, 目标Hyperdrive映射表和HDFS外表使用的是同一份数据文件, Hyperdrive映射表和HDFS外表使用的对应分隔符应该相同):
 - **HbaseTableFieldsTerminatedBy**: 列分隔符, 例如 `,`
 - **HbaseTableCollectionItemsTerminatedBy**: array类型元素分隔符和map类型键值对之间的分隔符, 例如 `\001`
 - **HbaseTableMapKeysTerminatedBy**: map类型的键和值之间的分隔符, 例如 `\003`
 - **HbaseTableLinesTerminatedBy**: 换行符, 例如 `\n`。
 - **HbaseFormat**: 映射表的Storage Handler。Hyperdrive映射表的Storage Handler为 `io.transwarp.hyperdrive.HyperdriveStorageHandler`。Hyperdrive SQL BulkLoad中的config.txt应该已经设置好该值, 您无需修改。

例 113. Hydrive SQL BulkLoad的示例config.txt

```
-----
#You must specify these variables
#command used to send sql file to hive server1 or hive server 2
#TranswarpCommand:=beeline -u jdbc:hive://transwarp-perf2:10000/default
-f
TranswarpCommand:=transwarp -t -h transwarp-perf2 -f
#Local directory of files you want to send to HDFS, if null, it will
use the HDFS files
LocalInputDir:=./data/*
#The hdfs directory you want to load data into Hbase table
Hdfsdir:=user/zhangyun/testcase_struct/
#database
ExternalTableDatabase:=sqlbulkload_test
HbaseTableDatabase:=sqlbulkload_test
-----
```

```

#ExternalTableDatabase:=default
#HbaseTableDatabase:=default
#temporary table that is used store split key, this table just be used
by this script
SplitTmpTbl:=tmp_split_keys_hb_bulkload
#external table name
ExternalTableName:=externalTable_test
#external table column fields
ExternaltableParameter:= ti tinyint,si smallint,i int,bi bigint,f
float,d1 double, dc decimal(20,10),str varchar(30),ts string,tsgp
string,strChina string,dt string,dtgp string, bl boolean,st
struct<ti:tinyint,si:smallint,i:int,bi:bigint,f:float,d1:double,
dc:decimal(20,10),str:varchar(30),ts:string,tsgp:string,strChina:string
, dt:string,dtgp:string,bl:boolean>
#hbase mapped table name, also being the name of hbase table
HbaseTableName:=transwarp_bulkload_test
#hbase namespace
NameSpace:=default
#defualt lenght of struct string for hyperdrive
Hyperdrive_structstring_length_default:=10
#hyperdrive table column fields(except for key field that is configured
with KeyFields parameter and FieldsSegmentLen parameter)
TableParameter:= ti tinyint,si smallint,i int,bi bigint,f float,d1
double, dc decimal(20,10),str varchar(30),ts string,tsgp
string,strChina string, dt string,dtgp string, bl boolean,st
struct<ti:tinyint,si:smallint,i:int,bi:bigint,f:float,d1:double,
dc:decimal(20,10),str:varchar(30),ts:string,tsgp:string,strChina:string
, dt:string,dtgp:string,bl:boolean>
#used strurct files of hbase talbe, to set segment lenght of fields of
string type
HbaseTableStructSegmentLen:=st.tsgp:10,st.strChina:15,st.dtgp:9,st.ts:1
5,st.dt:18,tsgp:10
#the columns you used to generate split keys. If you have give the
split
#keys, you also need to specify this parameter
KeyFields:=i:int,bi:bigint,d1:double,dc:decimal(20,10),tsgp:string,strC
hina:string
#set segment length of string fields in keyFieds, used for generating
split keys
FieldsSegmentLen:=tsgp:14,strChina:12
#extra udf attach to rowkey
Udf:=uniq()
#which columns you want to select from external table to hbase table
HbaseSelectedColumn:=ti,si,i,bi,f,d1,dc,str,ts,tsgp,strChina,dt,dtgp,
bl,st
#-----
#You'd better specify these variables
#the number of regions of the hbase table
RegionNum:=10
#External table separator
ExternalTableFieldsTerminatedBy:=,
ExternalTableCollectionItemsTerminatedBy:=
ExternalTableMapKeysTerminatedBy:=
ExternalTableLinesTerminatedBy:=\n
ExternalTableStoredAs:=TEXTFILE
#hbase table separator
HbaseTableFieldsTerminatedBy:=,
HbaseTableCollectionItemsTerminatedBy:=
HbaseTableMapKeysTerminatedBy:=
-----
```

```
-HbaseTableLinesTerminatedBy:=\n-----\nHbaseFormat:=io.transwarp.hyperdrive.HyperdriveStorageHandler\n#-----\n#If not necessary, you do not need to specify these variables\nSampleTableName:=sampleTable\nSampleTableDatabase:=sqlbulkload_test
```

完成对config.txt的修改以后，运行Hyperdrive SQL BulkLoad中的run.sh脚本：

```
sh run.sh
```

该脚本运行完成后，您的数据集会根据您在config.txt中的设置导入目标Hyperdrive表中，操作结束。

14.4. 运维管理工具

Transwarp HBase运维管理工具清单详见[\[operational-management-tools-list\]](#)。

15. Transwarp HBase常见问题

Inceptor Engine插入HBase表时报错

- 报错信息：

```
org.apache.hadoop.hive.q1.metadata.HiveException: java.lang.IllegalArgumentException:  
Row length is 0
```

```
| row selected (320.193 seconds)  
0: jdbc:hive2://localhost:10000  
0: jdbc:hive2://localhost:10000  
0: jdbc:hive2://localhost:10000 use nx;  
0: jdbc:hive2://localhost:10000 insert into table invm_tb  
. . . . > select /*+USE_BULKLOAD*/ acct_no,curr_status,acct_type,int_cat,customer_no,curr_bal,term_basis,trn_hold,trn_frm_dt,trn_to_dt,mat_dt,term_max_roll_no,trn_day,term_value,t  
rn_int_accr,rate_intd,trn_mat_rqd,trn_allow_ty,itype_int_accr,prev_int_rate,prev_to_date,trn_reason_cd from nx.invm  
. . . . > order by acct_no;  
Error: Error while processing statement: FAILED: Execution Error, return code 1 from io.transwarp.inceptor.execution.SparkTask. Job aborted due to stage failure: Task 0 in stage 248.0 failed 4 times, most recent failure: Lost task 0.3 in stage 248.0 (ID 25823, tdh102): org.apache.hadoop.hive.q1.metadata.HiveException: java.lang.IllegalArgumentException: Row length is 0 (state=08S01,code=1)  
0: jdbc:hive2://localhost:10000  
0: jdbc:hive2://localhost:10000
```

- 环境

TDH产品

- 问题原因

HBase的Rowkey长度必须 > 0；当Inceptor Engine插入到HBase时，若Rowkey为NULL或长度为0，就会抛出此异常。

- 解决方法

当key是string类型时：

```
Insert into table select * from table  
where key is not null and key != '';
```

当key不是string类型时：

```
Insert into table select * from table  
where key is not null;
```

其中，key是Inceptor Engine中对应Transwarp HBase表Rowkey的列名。

Inceptor Engine插入HBase表时报错

- 报错信息：

org.apache.hadoop.hive.ql.metadata.HiveException: java.lang.IllegalArgumentException:
No columns to insert

```
2016-08-04 16:17:03,017 WARN  thrift.ThriftCLIService: (ThriftCLIService.java:ExecuteStatement(549)) [HiveServer2-Handler-Pool:1]
nt:
org.apache.hive.service.cli.HiveSQLException: EXECUTION FAILED: Task MAPRED-SPARK error SparkException: [Error 1] Job aborted due
.0 failed 4 times, most recent failure: Lost task 0.3 in stage 2030.0 (TID 6089, node3): org.apache.hadoop.hive.ql.metadata.Hive
ption: No columns to insert
        at io.transwarp.inceptor.server.InceptorSQLOperation.runInternal(InceptorSQLOperation.scala:84)
        at org.apache.hive.service.cli.operation.Operation.run(Operation.java:279)
        at org.apache.hive.service.cli.session.HiveSessionImpl.executeStatementInternal(HiveSessionImpl.java:423)
        at org.apache.hive.service.cli.session.HiveSessionImpl.executeStatementWithParamsAndPropertiesAsync(HiveSessionImpl.java
        at org.apache.hive.service.cli.CLIService.executeStatementWithParamsAndPropertiesAsync(CLIService.java:320)
        at io.transwarp.inceptor.server.InceptorCLIService.executeStatementWithParamsAndPropertiesAsync(InceptorCLIService.scala
        at org.apache.hive.service.cli.thrift.ThriftCLIService.ExecuteStatement(ThriftCLIService.java:538)
        at org.apache.hive.service.cli.thrift.TCLIService$Processor$ExecuteStatement.getResult(TCLIService.java:1737)
        at org.apache.hive.service.cli.thrift.TCLIService$Processor$ExecuteStatement.getResult(TCLIService.java:1722)
        at org.apache.thrift.ProcessFunction.process(ProcessFunction.java:39)
        at org.apache.thrift.TBaseProcessor.process(TBaseProcessor.java:39)
        at org.apache.hive.service.auth.TSetIpAddressProcessor.process(TSetIpAddressProcessor.java:56)
        at org.apache.thrift.server.TThreadPoolServer$WorkerProcess.run(TThreadPoolServer.java:285)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1145)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615)
        at java.lang.Thread.run(Thread.java:745)
2016-08-04 16:17:03,017 INFO  scheduler.StatsReportListener: (Logging.scala:logInfo(59)) [SparkListenerBus()] -          0.0 B
0.0 B  0.0 B  0.0 B
2016-08-04 16:17:03,018 INFO  scheduler.StatsReportListener: (Logging.scala:logInfo(59)) [SparkListenerBus()] - executor (non-f
6, stdev: 3.337990, max: 94.565217, min: 85.950413)
2016-08-04 16:17:03,018 INFO  scheduler.StatsReportListener: (Logging.scala:logInfo(59)) [SparkListenerBus()] -          0%
```

- 环境

TDH产品

- 问题原因

数据插入到HBase中时，除了Rowkey这列以外，其余所有的列都指定为Null值，HBase不支持这种情况。

- 解决方法

执行SQL之前执行 `set hyperbase.fill.null.enable=true`

HBase插入数据时提示 Keyvalue 过大

- 报错信息：

`java.lang.IllegalArgumentException: Keyvalue size too large`

```
[root@tdh1 test]# java -jar hbaseos.jar test 10
log4j:WARN No appenders could be found for logger (org.apache.hadoop.metrics2.lib.MutableMetricsFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
pic1 is exists
Finished
times:0, cost:0, avgcost:0
times:0, cost:0, avgcost:1000
times:0, cost:0, avgcost:2000
times:0, cost:0, avgcost:3001
Exception in thread "Thread-3" java.lang.IllegalArgumentException: KeyValue size too large
        at org.apache.hadoop.hbase.client.HTable.validatePut(HTable.java:1341)
        at org.apache.hadoop.hbase.client.HTable.doPut(HTable.java:970)
        at org.apache.hadoop.hbase.client.HTable.put(HTable.java:937)
        at org.apache.hadoop.hbase.client.BaseIndexHTable.put(BaseIndexHTable.java:88)
        at io.transwarp.test.TestConcurrencyAPI.insert(TestConcurrencyAPI.java:191)
        at io.transwarp.test.TestConcurrencyAPI.insert(TestConcurrencyAPI.java:178)
        at io.transwarp.test.TestConcurrencyAPI$Worker.run(TestConcurrencyAPI.java:59)
times:0 cost:0 avgcost:1001
```

- 环境

TDH产品

- 问题原因

HBase默认支持的最大KeyValue size为10MB；可适当调大，如100MB。

- 解决方法

在客户端依赖的conf.set(hbase.client.keyvalue.maxsize,'102400000')中，将hbase.client.keyvalue.maxsize调整到所需大小。



主要出现在Object Store，详见[Object Store使用方法](#)。

Transwarp ES使用手册

16. Transwarp ES入门

16.1. Transwarp ES是什么

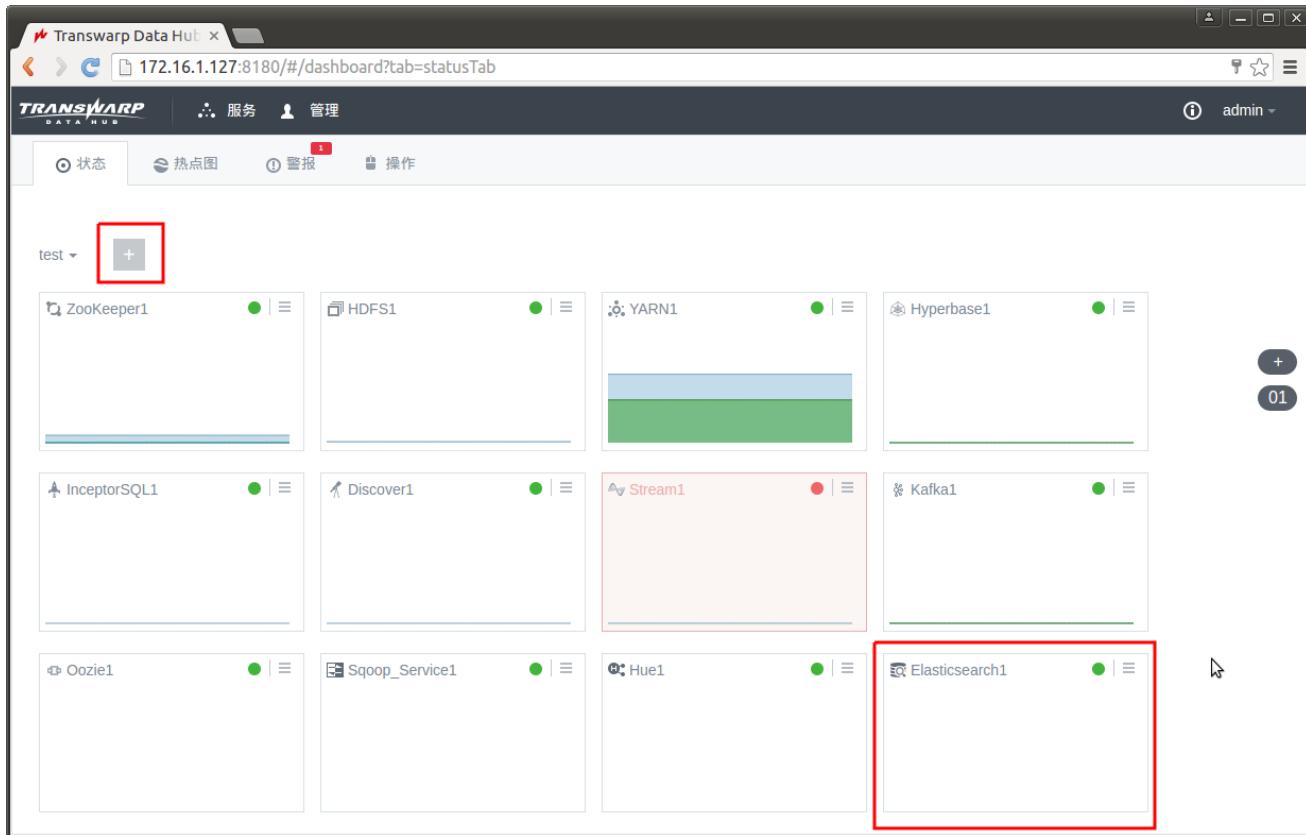
Transwarp ES基于开源的Elasticsearch并对其进行了优化。Transwarp ES是一个可扩展的分布式全文搜索和分析引擎。在Transwarp Data Hub中，Transwarp ES扮演两个角色：

- 作为Transwarp HBase全文索引的底层实现。
- 作为一个单独的服务，它是：
 - 分布式文件存储（Distributed Document Store）；
 - 强大的搜索引擎。常见应用场景有海量数据的存储和搜索、日志分析等。

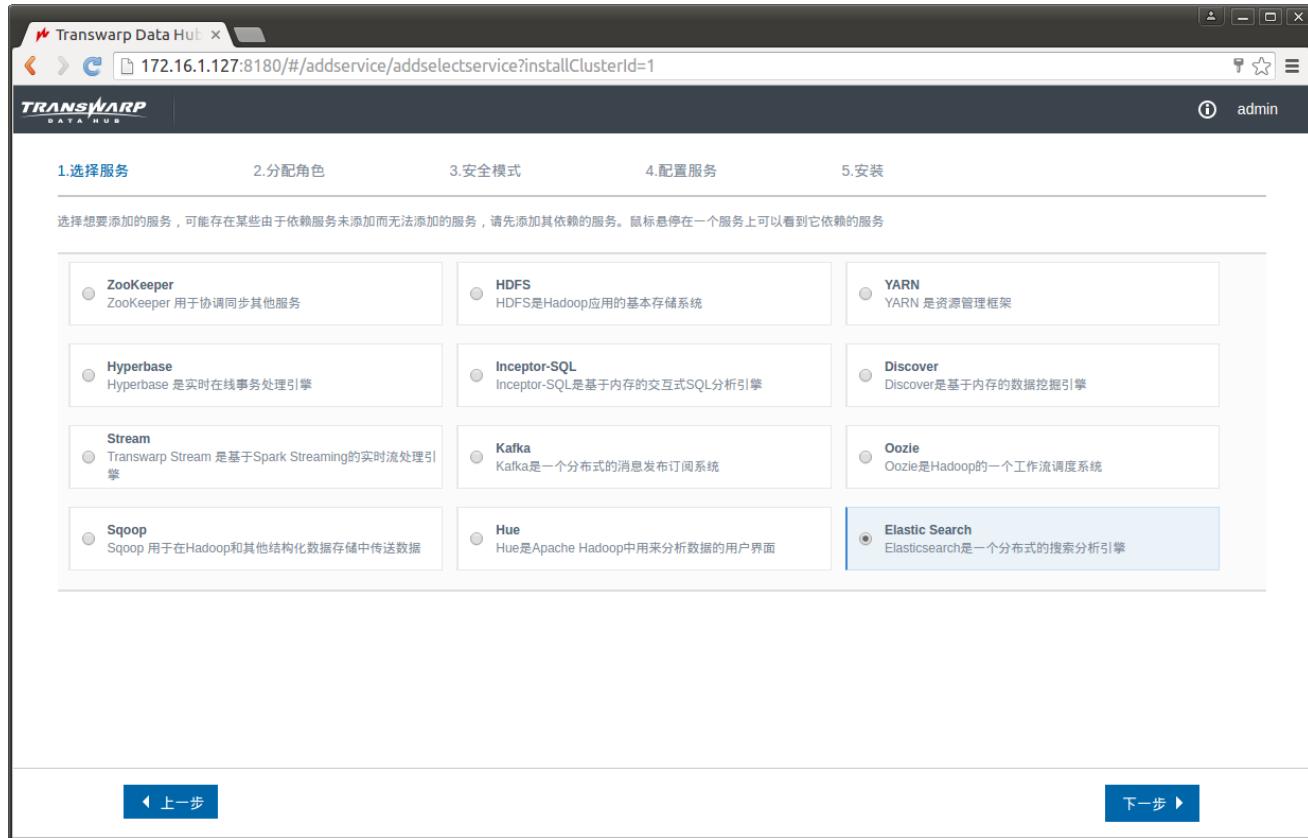
本手册将介绍如何在Transwarp Data Hub平台上将Transwarp ES作为一个单独的服务来使用。

16.2. 安装Transwarp ES

要查看您的集群是否已经安装了Transwarp ES，请登陆集群的Transwarp Manager（用浏览器访问`http://<manager_node_ip>:8180`）。Transwarp Manager首页如果显示有Elasticsearch，那么您的集群已经安装了一个Transwarp ES服务：



您也可以点击页面左上的“+”来安装一个Transwarp ES服务。Transwarp Manager会打开一个安装向导，您需要在向导中选择Elasticsearch：



接下来，安装向导会全程帮助您安装服务。更多细节请参考《Transwarp Data Hub安装手册》。

16.3. 了解您的Transwarp ES服务

Transwarp Manager提供了大量的服务信息，您可以方便地在其中监控和配置您的Transwarp ES服务。在Transwarp Manager首页点击Transwarp ES服务，进入服务主页（同时也是角色页面）：

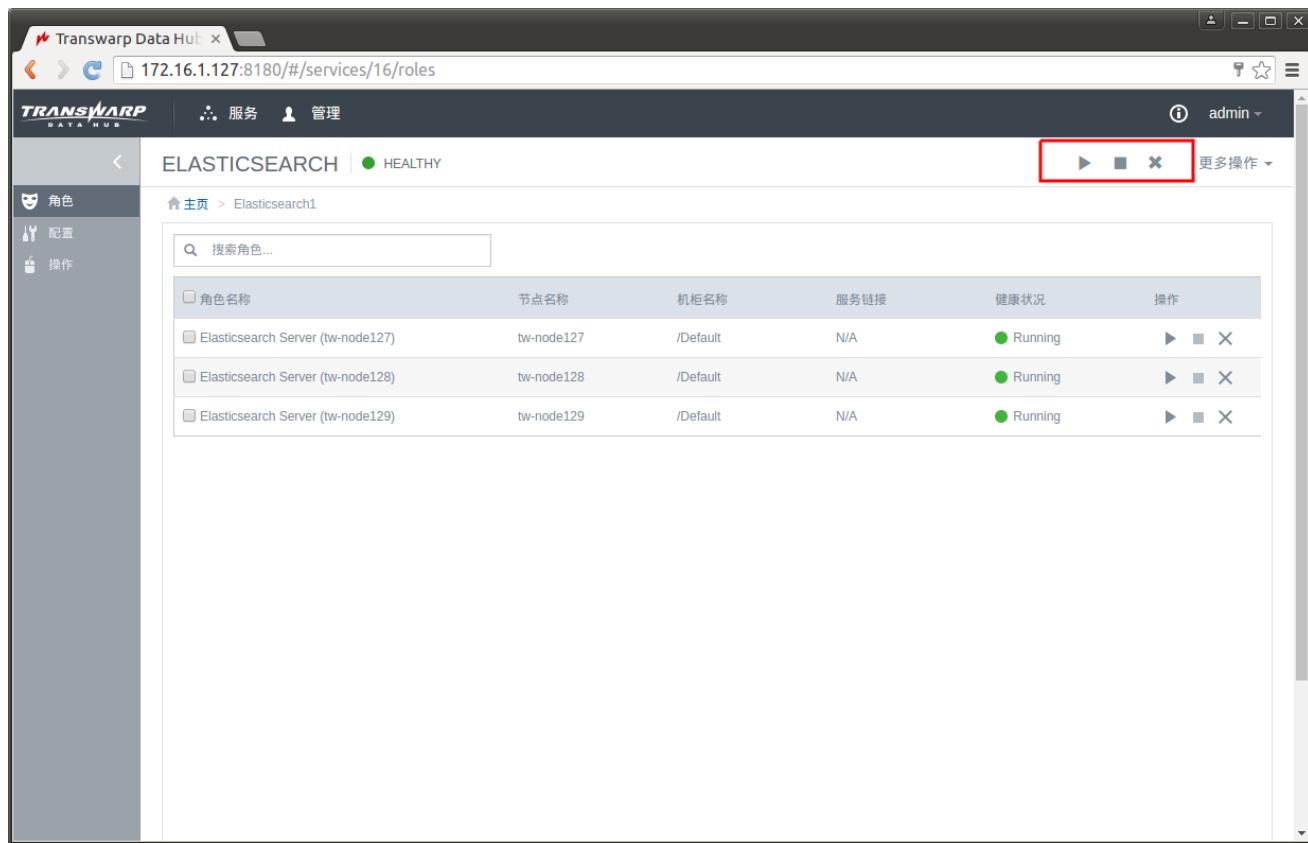


图 4. Transwarp ES服务主页/角色页面

这里您可以看到整个Transwarp ES服务以及各个节点的健康状态。您可以点击右上角的：

- ▶ 来启动服务；
- 来停止服务；
- ✖ 来删除服务。

点击页面左侧的菜单可以在服务的“角色”、“配置”和“操作”页面之间切换。服务主页同时也是服务的角色页面。在该页，您可以看到Transwarp ES集群中的节点（也就是安装了Transwarp ES的节点）以及它们的健康状况。我们可以看到，这个Transwarp ES服务中有三个节点，分别在tw-node127, tw-node128和tw-node129上。您可以点击节点对应的▶, ■ 和 ✖ 来启动、停止和删除节点。

16.4. 配置Transwarp ES

在Transwarp ES的配置页面，您可以查看和配置Transwarp ES的参数。



Transwarp Manager已经为所有参数配置了我们推荐的值，您无须自行配置便可以开始使用Transwarp ES。只有在当前配置无法满足您的业务需求时，您才需要按需修改这些参数的值。如果目前您还不了解这些参数的具体意义，您可以跳过这段直接进入下一章开始使用。

配置项	值	描述
elasticsearch.healthcheck.daemon.level	basic	frequency level of elasticsearch daemon status check
rack.name	(...)	The rack name of elasticsearch node
elasticsearch.healthcheck.vitalsign.level	basic	frequency level of elasticsearch vital sign check
es_heap_size	(...)	Set both the minimum and maximum memory to allocate to jvm
path.data	(...)	The data files path of elasticsearch
index.number_of_shards	10	Set the number of shards (splits) of an index
cluster.name	(...)	The cluster name of elasticsearch node
http.port	(...)	The http port of elasticsearch
path.conf	(...)	The config files path of elasticsearch
node.master	(...)	Allow this node to be eligible as a master node
discovery.zen.minimum_master_nodes	2	
gateway.recover_after_nodes	2	
node.data	(...)	Allow this node to store data
elasticsearch.healthcheck.operation.level	extended	frequency level of elasticsearch operation check
gateway.expected_nodes	3	
path.logs	(...)	The log files path of elasticsearch
elasticsearch.healthcheck.basic.interval	5	The interval between two batches of basic health checks

图 5. Transwarp ES的配置页面

要配置任何一个参数，您都必须进行下面三步：

1. 点击它的值进行修改；
2. 修改完成后，点击配置参数右上方的 **保存更改** 保存修改；
- 3.

最后点击页面右上角的 **更多操作**，然后点击弹出的 **配置服务** 使配置生效。

下面我们介绍一些重要的配置参数，一般情况下使用系统提供的默认值即可。

- **cluster.name:** Transwarp ES的集群名称

点击cluster.name的值，Transwarp Manager会弹出下面的小窗，供您修改集群中每个节点的名称：

编辑

配置项: cluster.name

搜索配置项...

批量修改

节点名称	值
tw-node127	test
tw-node128	test
tw-node129	test

保存

- **node.master:** 是否为master node

点击node.master的值, Transwarp Manager会弹出下面的小窗, 您可以查看并修改当前Transwarp ES集群中各个节点是否为master:

编辑

配置项: node.master

搜索配置项...

批量修改

节点名称	值
tw-node127	true
tw-node128	true
tw-node129	true

保存

- **node.data:** 是否为data node, 即是否用于存储数据

点击node.data的值, Transwarp Manager会弹出下面的小窗, 供您查看并修改当前Transwarp ES集群中各个节点是否为data node:

编辑 ×

配置项: node.data

搜索配置项...

批量修改

节点名称	值
tw-node127	true
tw-node128	true
tw-node129	true

保存

- **discovery.zen.ping.unicast.hosts:** 使用unicast来发现节点时的host。该参数须列出该Transwarp ES服务的所有master node。这里列出了tw-node127, tw-node128和tw-node129:

```
discovery.zen.ping.unicast.hosts      [ "tw-node127", "tw-node128", "tw-node129" ]
```

- **discovery.zen.minimum_master_nodes:** 节点能加入集群需要连接的最小master数量，为了防止脑裂，推荐配置为 $n/2+1$ ，n为master数量。例如一个3节点的Transwarp ES集群推荐配置为2。
- **discovery.zen.ping.multicast.enabled:** 是否使用multicast来发现节点。推荐配置为false。
- **index.number_of_shards:** Transwarp ES index的默认主分片（Primary Shard）数。默认配置为10。
- **index.number_of_replicas:** Transwarp ES index的每个主分片的副本分片数（Replica Shard）。默认配置为2。
- **path.data:** 数据存储路径。

点击该配置项的值，Transwarp Manager会弹出下面的小窗：

The screenshot shows a configuration interface for Transwarp ES. At the top, there's a search bar labeled '搜索配置项...' and a blue button labeled '批量修改'. Below is a table with columns '节点名称' (Node Name) and '值' (Value). The table contains three rows, each representing a node and its corresponding data path:

节点名称	值
tw-node127	/elasticsearch/data
tw-node128	/elasticsearch/data
tw-node129	/elasticsearch/data

At the bottom right of the interface is a blue '保存' (Save) button.

在这里，您可以为每个节点配置用于存储数据的目录，或者挂载存储数据使用的硬盘。

- **gateway.recover_after_nodes:** 触发recovery的节点数量阈值。该配置项需要根据具体的集群规模设置。
- **gateway.expected_nodes:** 集群规划的节点数，按具体情况设置。
- **gateway.recover_after_time:** 当集群节点数达不到expected_nodes时，可以设置延迟一段时间进行recovery。该参数配置延迟的时间。

16.5. 和Transwarp ES交互

和Transwarp ES交互有两种方式：REST API和ESDrive SQL。

16.5.1. REST API

Transwarp ES提供的丰富的REST API用于交互。默认设置下，Transwarp ES使用9200端口提供REST API访问。我们将在命令行中通过 `curl` 来向REST API提出请求，您只需进入一个Transwarp ES节点的Shell执行我们介绍的指令便可和Transwarp ES集群交互，这些指令的通用格式为：

通过 `curl` 使用REST API的通用格式

```
curl -X<VERB> '

- <VERB> 为HTTP方法，可以为：GET, POST, PUT, HEAD 和 DELETE。
- <HOST> 为一个运行着Transwarp ES的服务器的IP或者hostname。
- <PATH> 用于指定Index和Type，形式一般为 <INDEX>/<TYPE>
- <API> 为可选项，用于指定接受请求的REST API。
- <PARAMETERS> 为API可以接受的参数，API可以接收多个参数，参数之间用 & 隔开：例如 pretty&q=age:26&size=5。

```

- <BODY> 为可选项，是一个JSON格式的“请求体”（request body），包含检索请求的细节。

我们可以在实际操作中省去 `http://` 部分，将指令直接写为：

```
curl -X<VERB> '<HOST>:9200/<PATH>/[<API>]/[?<PARAMETERS>]' [-d '{<BODY>}' ]
```

另外，您也可以按需使用 `curl` 自带的选项。

了解您的Transwarp ES服务中我们提到了通过Transwarp Manager查看集群状态，通过REST API也可查看集群状态：

查看集群的health

```
curl -XGET 'localhost:9200/_cat/health?v'
```

输出类似如下：

```
[root@tw-node127 ~]# curl -XGET localhost:9200/_cat/health?v
epoch      timestamp cluster status node.total node.data shards pri relo init unassign
1459347413 22:16:53  test    green          3         3     0   0   0   0       0
```

- `cluster` 下对应的是Transwarp ES集群名称；
- `status` 下对应的是集群健康状况（`green` 代表健康）；
- `node.total` 下对应的是节点个数；
- `node.data` 下对应的是存储数据的节点的个数。

查看集群节点信息

```
curl -XGET 'localhost:9200/_cat/nodes?v'
```

输出类似如下：

```
[root@tw-node127 ~]# curl -XGET localhost:9200/_cat/nodes?v
host      ip      heap.percent ram.percent load node.role master name
tw-node127 172.16.1.127        20              d      m  tw-node127
tw-node128 172.16.1.128        21              d      m  tw-node128
tw-node129 172.16.1.129        10              d      *  tw-node129
```



为了描述的简洁，除非另外指出，后文我们将直接用HTTP方法名指代执行的指令。例如“`PUT ...`”指代“`curl -XPUT ...`”。

16.5.2. ESDrive SQL

4.6版本以前，Transwarp Data Hub(TDH)上还不能直接将SQL跑在Transwarp ES上，用户只能使用Transwarp ES的API或HBase全文检索来使用Transwarp ES。从TDH4.6开始，Hyperbase中引入了全新的ESDrive使得用户可以通过SQL的方式使用Transwarp ES，大大降低了全文检索相关的使用门槛。ESDrive SQL可以完全兼容所有的SQL语法，包括PLSQL，同时也有自己特定的分词检索等特殊语法。相比较API使用Transwarp ES和HBase全文检索的

方式，采用ESDrive有如下优势：

- 极高的易用性：

以往的Transwarp ES使用有较高的入门门槛，必须对Transwarp ES提供的各种REST API、query的写法、甚至是Apache Lucene底层技术比较熟悉的情况下，才能写出高效的查询条件，使用成本比较高，而使用ESDrive SQL，用户只要有编写SQL的经验，就可以简单的使用Transwarp ES。

- 性能提升：

大部分情况下，使用ESDrive可以获得比使用API有更高的性能提升，这是因为ESDrive内部做了一些特殊优化。比如，多条件查询的时候，不同条件的先后顺序会对查询有极大的影响，ESDrive内部对查询有一套自己的优化接口。

- 迁移成本低：

用户的业务逻辑从Oracle或DB2上迁移到ESDrive上时，迁移成本非常低，如果直接用API的方式迁移到Transwarp ES上，代价会非常高。有了ESDrive之后，用户只需要修改少量的SQL即可。

关于ESDrive SQL的具体使用，请参考[Transwarp ESDrive SQL使用说明](#)。

17. Transwarp ES数据模型

17.1. Transwarp ES中的数据对象

17.1.1. Transwarp ES Index（索引）

Transwarp ES以Index为单位来组织数据（[Transwarp ES Document](#)），一个Index下的数据常常有相似的特征。例如您可以为员工信息建一个Index，或者为商品信息建一个Index。每个Index都有一个名字用于在操作中指代，Index名必须都为英文小写。



注意，这里的Index（索引）和Transwarp HBase中的索引（本地索引、局部索引、全文索引等）不是一个概念。这里的索引是Transwarp ES中的数据对象。

17.1.2. Transwarp ES Type

在一个Transwarp ES Index下，您可以定义一个或多个Transwarp ES Type。Transwarp ES Type是Transwarp ES Index的逻辑上的分类，分类逻辑完全由用户决定。例如存储员工信息的Index可以按部门分类（分为财务部Type、销售部Type、研发部Type等）也可以按办公地点分类（分为北京办公室Type、上海办公室Type、广州办公室Type等）。

17.1.3. Transwarp ES Document

Transwarp ES Document 是Transwarp ES中最基础的数据单元。例如，员工信息Index中一名员工的信息可以作为一个Document保存；商品信息Index中一件商品的信息也可以作为一个Document保存。Transwarp ES Document以JSON格式存储，例如：

```
{
  "name": "Zhang San",
  "age": 26,
  "on_board_date": "2015-10-31",
  "school": "Nanjing University"
}
```

17.1.4. Field

Document中的信息存储在字段（Field）中。上面的Document中，“`name`”，“`age`”，“`on_board_date`”和“`school`”都是Document的字段（注意字段名称周围需要有双引号）。“`Zhang San`”，`26`，“`2015-10-31`”和“`Nanjing University`”则分别为这些字段的值。

17.2. 将Index映射为二维表

使用Transwarp ESDrive SQL和Transwarp ES交互时，Transwarp ESDrive会将Index映射成一张二维表，Index中的Document映射成表中的行，Index中的Field映射为表中的列。

Elasticsearch	Transwarp ESDrive
Index	表(Table)
Document	行(Row)
Field	列(Column)



用于映射为二维表（以便使用ESDrive SQL操作）的Transwarp ES Index只能有一个Type: **default_type_**。

Transwarp ESDrive支持数据类型

Transwarp ESDrive支持数据类型及其映射到Transwarp ES中的实际类型:

Transwarp ESDrive SQL支持数据类型	Transwarp ES中的实际类型
string	string
tinyint	byte
smallint	short
int	integer
bigint	long
float	float
double	double
boolean	boolean

为了更具体地说明Index和二维表的映射关系，我们来看下面的例子。

首先我们来看Transwarp ES是如何将处理在Transwarp ESDrive中插入的二维表的。在Transwarp ESDrive中创建一张Transwarp ES内表，并插入一些数据:

创建Transwarp ES内表

```
CREATE TABLE es_start(
    key STRING,
    content STRING,
    tint INT,
    tfloat FLOAT,
    tbool BOOLEAN)
STORED AS ES;
```

插入数据

```
INSERT INTO TABLE es_start(key, content, tint, tfloat, tbool) VALUES ("1",
"oracle", 1 ,1.1, true);
INSERT INTO TABLE es_start(key, content, tint, tfloat, tbool) VALUES ("2",
"oracle is database",2,2.2, false);
```

查看Transwarp ES表中内容

```
SELECT * FROM es_start;
1 oracle 1 1.1 true
2 oracle is database 2 2.2 false
```

接着我们在Transwarp ES中，查看 `default.es_start` 的Index的值：

查看Indexes

```
curl 'localhost:9200/default.es_start/_mget' -d '{
  "ids" : ["1", "2"]
}'

{
  "docs": [
    {
      "_index": "default.es_start",
      "_type": "default_type_",
      "_id": "1",
      "_version": 1,
      "_timestamp": "1470888259144",
      "found": true,
      "_source": {
        "content": "oracle",
        "tint": 1,
        "tfloat": 1.1,
        "tbool": true
      }
    },
    {
      "_index": "default.es_start",
      "_type": "default_type_",
      "_id": "2",
      "_version": 1,
      "_timestamp": "1470888262181",
      "found": true,
      "_source": {
        "content": "oracle is database",
        "tint": 2,
        "tfloat": 2.2,
        "tbool": false
      }
    }
  ]
}
```

可以看到，二维表对应了Transwarp
ES中的Index，二维表的每一行对应了每个Document，每个列族对应了Index中的每个Field。

在了解到Transwarp ES是如何处理二维表之后，下面我们显式地来创建Transwarp ES中Index到二维表的映射：
先将两段段员工信息信息编入 `default.employee` 下的 `default_type_` 中，可参考[编入Document](#)：

编入员工的信息

```
curl -XPUT 'localhost:9200/default.employee/default_type_1?pretty' -d '{  
    "firstname": "San",  
    "lastname": "Zhang",  
    "age": 26,  
    "on_board_date": "2015-10-31",  
    "hometown": "Beijing",  
    "school": "Nanjing University",  
    "married": false,  
    "about": "I love Beijing Opera"  
}'  
  
curl -XPUT 'localhost:9200/default.employee/default_type_2' -d '{  
    "firstname": "Si",  
    "lastname": "Li",  
    "age": 28,  
    "on_board_date": "2014-09-16",  
    "hometown": "Nanjing",  
    "school": "Beijing University",  
    "married": true,  
    "about": "cooking, mountain climbing"  
}'
```

我们来看Transwarp ES是如何映射这些信息的：

```
{
  "state": "open",
  "settings": {
    "index": {
      "creation_date": "1470820757137",
      "number_of_shards": "10",
      "number_of_replicas": "0",
      "uuid": "MLaEq5MeQWqa8bcWfZTAnw",
      "version": {
        "created": "2000099"
      }
    }
  },
  "mappings": {
    "default_type_": {
      "properties": {
        "firstname": {
          "type": "string"
        },
        "hometown": {
          "type": "string"
        },
        "on_board_date": {
          "format": "strict_date_optional_time||epoch_millis",
          "type": "date"
        },
        "school": {
          "type": "string"
        },
        "about": {
          "type": "string"
        },
        "married": {
          "type": "boolean"
        },
        "age": {
          "type": "long"
        },
        "lastname": {
          "type": "string"
        }
      }
    }
  },
  "aliases": [ ]
}
}
```

利用Transwarp ESDrive SQL建立二维表 `employee`，并建立 `default.employee` 和二维表 `employee` 的映射关系：

```

CREATE EXTERNAL TABLE employee(
    key string,
    firstname string,
    lastname string,
    age bigint,
    on_board_date string,
    hometown string,
    school string,
    married boolean,
    about string
)
STORED BY 'io.transwarp.esdrive.ElasticSearchStorageHandler'
WITH SERDEPROPERTIES(
    'elasticsearch.columns.mapping'='_id,firstname,lastname,age,on_board_date,
    hometown,school,married,about',
    'elasticsearch.indextype'='default_type_')
TBLPROPERTIES('elasticsearch.tablename'='default.employee');

```

在 `elasticsearch.columns.mapping` 指定的映射中，二维表 `employee` 将 Document 的 `_id` 作为行键，即 Document 被映射成表中的行。而 Index 中的 Field，如 `firstname` 等，分别对应着二维表的列族。下面用 `select` 语句来查看建表结果：

```

SELECT * FROM employee;

1 San Zhang 26 2015-10-31 Beijing Nanjing University false I love
Beijing Opera
2 Si Li 28 2014-09-16 Nanjing Beijing University true cooking,
mountain climbing

```



在 Transwarp ESDrive 中建表时，需要将 `date` 类型改为 `string` 类型；也要将 `long` 类型写作 `bigint` 类型。

18. Transwarp ES API简明教程



本章介绍一些基础的Transwarp ES API操作，使您可以快速熟悉如何使用Transwarp ES API 和Transwarp ES交互。如果您想要快速熟悉如何使用Transwarp ESDrive SQL和Transwarp ES交互，请参考[Transwarp ESDrive SQL 快速入门](#)。

本章我们搭建一个员工信息Index（Index名为 **employee**），并通过这个Index来演示一些常见操作。对这个Index我们将进行如下设计：

- Index下的每个Document对应一名员工的信息。
- Index下的Type按部门分类，分为 **dev**（研发部）、**finance**（财务部）和 **sales**（销售部）。

18.1. 编入Document



在Transwarp ES术语中，“向Index新增Document”的操作称为将Document“编入”Index，本手册中我们将使用该术语。

目前，employee Index还不存在，但是我们无需专门创建它，只需直接将Document放入（PUT）对应的路径下，路径的形式为 /<index>/<type>/<id>，Transwarp ES会在 PUT 时自动创建employee Index。例如，下面我们将员工Zhang San的Document编入 employee 下的 dev：

编入员工Zhang San的信息

```
curl -XPUT 'localhost:9200/employee/dev/1?pretty' -d '{
  "firstname": "San",
  "lastname": "Zhang",
  "age": 26,
  "on_board_date": "2015-10-31",
  "hometown": "Beijing",
  "school": "Nanjing University",
  "married": false,
  "about": "I love Beijing Opera"
}'
```

路径 **/employee/dev/1** 包含了三条信息：

- **employee** 为Index名字
- **dev** 为Type名字
- **1** 为这条Document的ID。

Transwarp ES会输出下面信息，说明Document插入成功：

```
{
  "_index": "employee",
  "_type": "dev",
  "_id": "1",
  "_version": 1,
  "created": true
}
```

这条信息包含了一个Document的身份元数据 (identity metadata)：`_index`、`_type`、`_id` 和 `_version` 分别对应该Document的Index、Type、ID和版本号。Transwarp ES中每个Document都有一个版本号，而每次对该Document进行修改（包括更新、插入和删除）都会让版本号加1。`created` 项的值为 `true` 说明该Document是第一次创建。

下面我们再插入两个Document:

```
curl -XPUT 'localhost:9200/employee/dev/2' -d '{
  "firstname": "Si",
  "lastname": "Li",
  "age": 28,
  "on_board_date": "2014-09-16",
  "hometown": "Nanjing",
  "school": "Beijing University",
  "married": true,
  "about": "cooking, mountain climbing"
}'

curl -XPUT 'localhost:9200/employee/dev/3' -d '{
  "firstname": "Wu",
  "lastname": "Wang",
  "age": 24,
  "on_board_date": "2016-01-05",
  "hometown": "Shanghai",
  "school": "Fudan University",
  "married": false,
  "about": "My favorite writer is Mo Yan."
}'

curl -XPUT 'localhost:9200/employee/sales/4' -d '{
  "firstname": "Qui",
  "lastname": "Li",
  "age": 35,
  "on_board_date": "1205-09-16",
  "hometown": "unknown",
  "school": "Home Schooled",
  "married": false,
  "about": "I am very strong."
}'
```

18.2. 获取整个Document

要获取一个完整的Document，使用 `GET` 并指定Document的路径，路径的形式为 `/<index>/<type>/<id>`:

获取/employee/dev/1下的Document

```
curl -XGET 'localhost:9200/employee/dev/1?pretty'
```

这里，我们用 `?pretty` 指定了JSON格式化输出。输出如下：

```
{
  "_index" : "employee",
  "_type" : "dev",
  "_id" : "1",
  "_version" : 1,
  "found" : true,
  "_source":{
    "firstname" : "San",
    "lastname" : "Zhang",
    "age" : 26,
    "on_board_date" : "2015-10-31",
    "hometown" : "Beijing",
    "school" : "Nanjing University",
    "married" : false,
    "about" : "I love Beijing Opera"
  }
}
```

我们看到，`GET` 的结果包含了/employee/dev/1的元数据以及一个 `_source` 字段，`_source` 字段中存储的是Document内部的信息。

18.3. 获取部分Document

默认情况下，`GET` 会打印Document完整的 `_source`。在 `GET` 时加上 `_source` 参数可以获取Document中的指定字段，如果一次指定多个字段，字段名称之间用“,” 隔开：

获取/employee/dev/1中的 `name` 和 `age` 字段

```
curl -XGET
'localhost:9200/employee/dev/1?_source=firstname,lastname,age&pretty'
```

Transwarp ES会只输出name和age字段：

```
{
  "_index" : "employee",
  "_type" : "dev",
  "_id" : "1",
  "_version" : 1,
  "found" : true,
  "_source":{"age":26, "lastname":"Zhang", "firstname":"San"}
```

如果不希望Transwarp ES输出元数据，可以将 `_source` 作为一个endpoint使用：

只输出 `_source`

```
curl -XGET 'localhost:9200/employee/dev/1/_source?pretty'
```

Transwarp ES将只输出 `_source` 中的数据:

```
{
  "firstname": "San",
  "lastname": "Zhang",
  "age": 26,
  "on_board_date": "2015-10-31",
  "hometown": "Beijing",
  "school": "Nanjing University",
  "married": false,
  "about": "I love Beijing Opera"
}
```

18.4. 查看Document是否存在

查看某个路径下的Document是否存在使用 `HEAD`, `curl` 要加上 `-i` 选项打印HTTP header:

查看`/employee/dev/1`下是否存在Document

```
curl -i -XHEAD 'localhost:9200/employee/dev/1'
```

输出为:

```
HTTP/1.1 200 OK ①
Content-Type: text/plain; charset=UTF-8
Content-Length: 0
```

① `200 OK` 说明该Document存在。

18.5. 删除Document

删除指定路径下的Document使用 `XDELETE`:

删除`/employee/dev/1`下的Document

```
curl -XDELETE 'localhost:9200/employee/dev/1?pretty'
```

Transwarp ES会输出下面信息, 说明删除成功:

```
{
  "found":true,
  "_index":"employee",
  "_type":"dev",
  "_id":"1",
  "_version":2
}
```

注意，删除操作让版本号变为了2。

我们可以再一次用 **XHEAD** 查看/employee/dev/1下是否存在Document，Transwarp ES会返回 **404 Not Found**:

```
curl -i -XHEAD 'localhost:9200/employee/dev/1'
HTTP/1.1 404 Not Found
Content-Type: text/plain; charset=UTF-8
Content-Length: 0
```

18.6. 更新Document

Transwarp ES中的Document不可更改，只能 **更新**。操作和编入Document相同——对想要更新的Document路径执行 **PUT**，Transwarp ES会将路径下的Document更新并对版本号加1。下面我们将对/employee/dev/2下的Document进行更新（将age从原来的28改为30）：

更新/employee/dev/2下的Document

```
curl -XPUT 'localhost:9200/employee/dev/2' -d '{
  "firstname": "Si",
  "lastname": "Li",
  "age": 30,
  "on_board_date": "2014-09-16",
  "hometown": "Nanjing",
  "school": "Beijing University",
  "married": true,
  "about": "cooking, mountain climbing"
}'
```

Transwarp ES会输出下面信息，注意版本号变成了2。另外，**created** 值为false说明这次 **PUT** 并没有新建该Document。

```
{
  "_index" : "employee",
  "_type" : "dev",
  "_id" : "2",
  "_version" : 2,
  "created" : false
}
```

18.7. 新建一个Document

我们看到 **PUT** 可能新建一个Document（例如[编入Document](#)中介绍的），也可能覆盖原有的Document（例如[更新Document](#)中介绍的）。那么如何确保我们在 **PUT** 的时候不会覆盖已有的Document呢？特定的/`<index>/<type>`下的Document靠它们的ID区分，所以我们需要确保新增的Document的ID是唯一的即可，使用 **POST** 可以让Transwarp ES为新增的Document自动生成一个唯一的ID：

向`/employee/sales`下编入一个新Document

```
curl -XPOST 'localhost:9200/employee/sales?pretty' -d '{
  "firstname": "Lei",
  "lastname": "Li",
  "age": 28,
  "on_board_date": "2013-10-03",
  "hometown": "Hangzhou",
  "school": "Zhejiang University",
  "married": true,
  "about": "I appear in your English textbook."
}'
```

注意，使用 **POST** 只需给出`<index>/<type>/`。Transwarp ES的输出如下：

```
{
  "_index": "employee",
  "_type": "sales",
  "_id": "aKWyjab5Se-nt7gyYNJsGg", ①
  "_version": 1,
  "created": true
}
```

① Transwarp ES自动生成的Document ID。

18.8. 轻量检索

在[获取整个Document](#)和[获取部分Document](#)中，我们看到 **GET** 可以获取Document信息。使用 **GET** 是将 `_search` 作为endpoint可以进行检索。下面我们执行一个最简单的检索：

```
curl -XGET 'localhost:9200/employee/dev/_search?pretty'
```

这个检索指令会将`/employee/dev/`下所有的Document返回。

轻量检索 指通过在 **GET** 请求的URI中直接提供检索串（query-string）来进行检索，可以用于快速简单的检索请求。例如：

在`employee` Index 中查找`firstname`为`Si`的Document

```
curl -XGET 'localhost:9200/employee/_search?pretty&q=firstname:Si'
```

Transwarp ES的输出为：

```
{  
    "took" : 13,  
    "timed_out" : false,  
    "_shards" : {  
        "total" : 10,  
        "successful" : 10,  
        "failed" : 0  
    },  
    "hits" : {  
        "total" : 1,  
        "max_score" : 0.30685282,  
        "hits" : [ {  
            "_index" : "employee",  
            "_type" : "dev",  
            "_id" : "2",  
            "_score" : 0.30685282,  
            "_source":{  
                "firstname" : "Si",  
                "lastname" : "Li",  
                "age" : 30,  
                "on_board_date" : "2014-09-16",  
                "hometown" : "Nanjing",  
                "school" : "Beijing University",  
                "married" : true,  
                "about" : "cooking, mountain climbing"  
            }  
        } ]  
    }  
}
```

用类似的请求我们可以检索/employee/dev下age为26的员工信息：

检索age为26的员工信息

```
curl -XGET 'localhost:9200/employee/dev/_search?pretty&q=age:26'
```

Transwarp ES的输出为：

```
{  
    "took" : 17,  
    "timed_out" : false,  
    "_shards" : {  
        "total" : 10,  
        "successful" : 10,  
        "failed" : 0  
    },  
    "hits" : {  
        "total" : 1,  
        "max_score" : 1.0,  
        "hits" : [ {  
            "_index" : "employee",  
            "_type" : "dev",  
            "_id" : "1",  
            "_score" : 1.0,  
            "_source":{  
                "firstname" : "San",  
                "lastname" : "Zhang",  
                "age" : 26,  
                "on_board_date" : "2015-10-31",  
                "hometown" : "Beijing",  
                "school" : "Nanjing University",  
                "married" : false,  
                "about" : "I love Beijing Opera"  
            }  
        } ]  
    }  
}
```

18.9. 总结

希望上述操作对您快速熟悉Transwarp ES的功能和操作有帮助。这些操作仅仅是Transwarp ES强大功能的冰山一角。我们会在下面的内容中介绍更多的Transwarp ES REST API的使用。

19. Transwarp ES架构

19.1. 分片（Shard）和副本（Replica）

一个Transwarp ES Index下可能会有大量的数据，超过硬件的存储能力。Transwarp ES中，您可以将Index分成多个 分片（shard），将Index分片有两个作用：

- 横向扩展一个Index的容量；
- 提高计算的并行度从而提升性能。

Transwarp ES中的分片分为两种：主分片（Primary Shard）和副本分片（Replica Shard，或简称Replica）。Index中的每个Document都属于一个唯一的Primary Shard，所以Primary Shard数量决定了一个Index的容量。Replica Shard则是Primary Shard的拷贝，不仅用于提供数据冗余，也提供数据读取服务（比如检索请求、Document获取请求等）。Primary Shard数量需要在Index创建时指定，创建后不可修改。一个Index中每个Primary Shard的Replica数则既可以在Index创建时指定，也可以在Index创建后动态修改。默认设置下，一个Transwarp ES Index有10个Primary Shard，每个Primary Shard有2个Replica，所以默认情况下一个Transwarp ES Index会总共有30个分片。

分片的分布和计算的并行化完全由Transwarp ES来管理，您作为用户完全无须费心。Transwarp ES会保证一个节点上相同数据只有一份，也就是说Primary Shard和它自己的Replica永远不会存在一个节点上。所以，如果一个Index的Replica数大于或等于Transwarp ES集群中节点数量，这个Index中将会有分片无法分配到节点上。

20. Transwarp ES检索简介

到目前为止，我们只介绍了Transwarp ES作为分布式文件存储的功能。然而，Transwarp ES真正的强大之处在于它的检索功能。Transwarp ES为Document中的每一个字段都建索引，让Document中的每一个字段都可以被检索。Transwarp ES提供 `_search` API用于接受检索请求，本章将简单概括 `_search` API的一部分使用方法，后面的章节还会涉及更多内容。

20.1. 空检索

我们从Transwarp ES中最简单的检索开始。Transwarp ES中最简单的检索是空检索（empty search），即不指定任何检索条件，要求Transwarp ES返回所有Index下的所有Document。我们利用空检索解释一些Transwarp ES对检索返回的信息。下面的检索输出为了文档的简洁，进行了删减：

空检索

```
curl -XGET 'localhost:9200/_search?pretty'
```

输出为：

```
{
  "took" : 118, ①
  "timed_out" : false, ②
  "_shards" : {
    "total" : 50,
    "successful" : 50,
    "failed" : 0
  },
  "hits" : {
    "total" : 1020,
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "bank",
        "_type" : "account",
        "_id" : "1",
        "_score" : 1.0,
        "_source": {"account_number":1,"balance":39225,"firstname":"Amber"
        , "lastname":"Duke","age":32,"gender":"M","address":"880 Holmes Lane"
        , "employer":"Pyrami","email":"amberduke@pyrami.com","city":"Brogan","state"
        :"IL"}
      },
      ... 为简洁删减了其他9条返回记录 ...
    ]
  }
}
```

① `took`: 本次检索所花时间（单位毫秒）。

② `timed_out`: 本次检索是否超时。

③ **_shards**: 所有本次检索涉及的分片数，分为所有涉及的（**total**）、成功的（**successful**）和失败的（**failed**）。保存相同数据的分片（主分片和它的副本分片）算作同一个分片。所以只有当某一主分片和它所有的副本分片都无法响应检索才会出现 **failed** 的情况。

④ **hits**: 检索的匹配结果，是返回内容中最重要的信息。我们将在[下面](#)讨论。

⑤ 默认情况下，一次查询只会返回前10条结果。这里为了行文简洁，我们删减了其他9条。

下面，我们只看上面输出中的 **hits** 部分。

hits 字段

```
"hits" : {
  "total" : 1020, ①
  "max_score" : 1.0, ②
  "hits" : [ { ③
    "_index" : "bank",
    "_type" : "account",
    "_id" : "1",
    "_score" : 1.0,
    "_source": {"account_number":1,"balance":39225,"firstname":"Amber"
, "lastname":"Duke","age":32,"gender":"M","address":"880 Holmes Lane"
, "employer":"Pyrami","email":"amberduke@pyrami.com","city":"Brogan","state
":"IL"}
  },
  ... 为简洁删减了其他9条返回记录 ...
]
}
```

① **total**: 检索出的Document数量，我们将把一个检索出的Document称为一个“hit”。所以该查询有1020个hit。

② **max_score**: 所有检索出的Document的 **_score** 中的最大值。

③ **hits**: 这里的 **hits** 是一个数组，数组中的元素是匹配的Document的信息，包括Document的Index, Type, ID, 关联分数 (**_score**) 和 Document 内数据 (**_source**)。为了和包含这个数组的 **hits** 字段做区分，我们称该数组为“**hits** 数组”，而包含它的字段为 **hits** 字段。

20.2. 检索请求的格式

更多时候，我们需要指定检索条件。通常，检索请求将和下面相似：

查询的基本格式

```
curl (-XGET|-XPOST) '<HOST>:9200/<PATH>/_search?<PARAMETERS>' -d
'<BODY>'
```

其中，**<PATH>** 用于指定检索的Index和Type。Transwarp ES支持同时跨Index和跨Type检索。例如：

在employee Index下的所有Type中检索

```
curl -XGET 'localhost:9200/employee/_search'
```

在employee Index下的dev和sales Type中检索

```
curl -XGET 'localhost:9200/employee/dev,sales/_search'
```

在employee, new_employee这两个Index下的dev和sales Type中检索

```
curl -XGET 'localhost:9200/new_employee,employee/dev,sales/_search'
```

在所有Index中的dev和sales Type中检索

```
curl -XGET 'localhost:9200/_all/dev,sales/_search'
```

不指定 <PATH> : 在整个Transwarp ES集群的所有Document中检索

```
curl -XGET 'localhost:9200/_search'
```

Transwarp ES提供两种使用 `_search` API的方法：URI检索和请求体（Request Body）检索。`GET` 和 `POST` 都可以用于检索请求，两者的返回信息没有区别。有的客户端不支持在 `GET` 时使用请求体，这时就必须使用 `POST`。

- **URI检索** 将查询放在 `<PARAMETERS>` 中，省掉 `-d '{<BODY>}'` 部分。这种形式的检索又称为“轻量检索（Search Lite）”，适合短小的即席查询。例如：

```
curl -XGET 'localhost:9200/employee/dev/_search?pretty&q=lastname:Li'
```

URI检索的细节将在[URI检索](#)中展开介绍。

- **请求体检索** 将查询放在 `<BODY>` 中，例如：

```
curl -XPOST 'localhost:9200/employee/dev/_search?pretty' -d '
{
  "query" : {
    "match_phrase" : {
      "lastname" : "Li"
    }
  }
}'
```

请求体检索的细节将在[Request Body检索（Elasticsearch 1.3.1）](#) 或[Request Body检索（Elasticsearch 2.0.0）](#) 中展开介绍。

21. URI检索

在URI检索中，我们通过直接将检索参数包含在URI中发出检索请求。URI检索的功能较少，适用于快速的即席检索。它的格式为：

URI检索格式

```
curl -XGET
'<HOST>:9200/<PATH>/_search?q=<QUERY_STRING> [&<PARAMTERS>=<VALUE>&<PARAMTER
S>=<VALUE>... ]'
```

其中，`q` 参数用于接收查询（`<QUERY_STRING>`），`<QUERY_STRING>` 需要符合一定的书写规范。URI中还可以包含一些其他参数用于控制Transwarp ES输出的形式。其中较为常见的有：

- `df`: `<QUERY_STRING>` 中不指定查询字段时默认使用的查询字段。它的默认值为 `_all` 字段。
- `analyzer`: 用于指定为 `<QUERY_STRING>` 分词的分词器。
- `explain`: 为每一个匹配的Document解释该Document的关联分 `_score` 是如何计算的。
- `default_operator`: 默认的运算符，可以设为 `AND` 或 `OR`。它的默认值为 `OR`。
- `_source`: 可以通过将值设为 `false` 做到不获取Document的 `_source` 字段。
 - `_source_include`: 指定获取某个或某几个字段
 - `_source_exclude`: 指定不获取某个或某几个字段
- `sort`: 指定排序方式。写法为 `sort=<fieldName>:<asc|desc>`，`<fieldName>` 可以是Document中实际的字段名，也可以是 `_score` 字段。`sort` 可以接受多个参数，参数间用“,” 隔开。排序键出现的顺序会影响性能。
- `timeout`: 超时时间。默认为没有超时时间。设置后，Transwarp ES将在 `timeout` 内返回已查询到的结果。
- `from`: 指定从第几个结果开始输出，默认值为0。
- `size`: 指定输出多少个匹配结果，默认值为10。
- `lowercase_expanded_terms`: 设置是否将查询中的词项（term）全部变为小写，默认为 `true`。
- `analyze_wildcard`: 是否为wildcard query和prefix query分词，默认为 `false`。

例 114. 用 `df` 指定默认查询字段

```
curl -XGET 'localhost:9200/employee/_search?q=Beijing&df=school'
```

例 115. 将返回结果排序

```
curl -XGET
'localhost:9200/employee/_search?q=*&sort=lastname:asc&pretty'
```

21.1. 输出结果分页

默认情况下，一次查询返回前十条匹配结果。使用 `from` 和 `size` 参数可以用于对返回结果分页。

例 116. 输出前2条结果

```
curl -XGET 'localhost:9200/employee/_search?q=*&size=2'
```

例 117. 从第3条起输出结果

```
curl -XGET 'localhost:9200/employee/_search?q=*&from=3'
```

例 118. 从第3条起输出2条结果

```
curl -XGET 'localhost:9200/employee/_search?q=*&from=3&size=2'
```

21.2. Query String语法

本节介绍URI检索格式中 `<QUERY_STRING>` 的书写规范。



URI中的 `<QUERY_STRING>` 需要使用 [百分号编码](#)。一些特殊字符需要用 % 开头的字符串表示。下面会用到的用 %20 表示空格，例如 `a AND b` 将表示为 `a%20AND%20b`。

下面我们会在实际的指令中使用百分号编码，其他地方使用字符本身。

21.2.1. 检索字段的指定

检索时，可以用 `<fieldName>:<value>` 的方式指定在哪个字段内搜索 `value`。

例 119. 在指定字段中检索

下面的查询返回employee Index中name字段包含 Li 的Document:

```
curl -XGET 'localhost:9200/employee/_search?pretty&q=name:li'
```

21.2.2. 逻辑运算符

在同一个字段中指定多个 value 时, value 之间可以用 AND 或 OR 运算符连接。一个Document的指定字段中必须包含 所有 AND 连接的 value 才能算匹配; 一个Document的字段中只要包含 OR 连接的 value 中的一个就能算匹配。另外, AND 和 OR 也可以用于连接多个 <fieldName>:<value> 条件。条件之间用 AND 连接时, Document必须满足全部条件才能算匹配; 条件之间用 OR 连接时, Document只需满足条件之一便能算匹配。不指定运算符则代表 value 或 <fieldName>:<value> 条件之间用默认运算符连接。例如:

- hometown:(Nanjing OR Beijing)
- school:(Beijing AND University)
- school:(Beijing University)
- lastname:li AND school:beijing

例 120. 使用 OR 运算符连接 value

```
curl -XGET  
'localhost:9200/employee/_search?q=hometown:(Nanjing%20OR%20Beijing)'
```

例 121. 使用 AND 运算符连接 value

```
curl -XGET  
'localhost:9200/employee/_search?q=school:(Beijing%20AND%20University)'
```

例 122. 不指定运算符连接 value

```
curl -XGET  
'localhost:9200/employee/_search?q=school:(Beijing%20University)'
```

例 123. 使用 AND 运算符连接 <fieldName>:<value> 条件

```
curl -XGET
'localhost:9200/employee/_search?q=(lastname:li)%20AND%20(school:beijing)'
```

21.2.3. 偏好运算符

默认情况下，`fieldName` 对应的所有 `value` 都是可选的，例如 `school` 字段只要包含 `beijing` 或 `university` 便可匹配 `school:beijing university`。我们在逻辑运算符中看到，用 `AND` 运算符可以指定哪些 `value` 是必选的。Transwarp ES还提供两个偏好运算符：`+`（必须匹配）和 `-`（必须不匹配）。前面没有偏好运算符的 `value` 则为可选项——它们为Document的关联性加分。

- `school:(-university)`
- `school:+nanjing+beijing`

例 124. - 运算符

```
curl -XGET 'localhost:9200/employee/_search?q=school:(-university)'
```

例 125. + 运算符

```
curl -XGET 'localhost:9200/employee/_search?q=school:+nanjing+beijing'
```

21.2.4. 括号的使用

可以通过使用括号为 `<QUERY_STRING>` 分组，使表达更清晰。括号还可以用于写子查询，例如 `school=(beijing AND university) OR (nanjing AND university)`。

例 126. 用括号写子查询

```
curl -XGET
'localhost:9200/employee/_search?q=school=(beijing%20AND%20university)%20OR%20(nanjing%20AND%20university)'
```

21.2.5. 通配符

`<QUERY_STRING>` 中支持使用通配符：`?` 匹配任意一个字符，`*` 匹配任意多个字符。例如：

例 127. 在URI检索中使用通配符

```
curl -XGET 'localhost:9200/employee/_search?q=??d*' 
```

21.2.6. 正则表达式

<QUERY_STRING> 中支持使用正则表达式。正则表达式需要放在 / 中，例如：/.*y/。

例 128. 在 <QUERY_STRING> 中使用正则表达式

```
curl -XGET 'localhost:9200/employee/_search?q=/.*y/' 
```

21.2.7. 范围查询

对date类型、数值类型和string类型的字段查询时可以用 >, >=, <, <= 来在指定的范围内进行匹配，例如 `on_board_date:>2014-01-01 AND <2015-12-31`。

例 129. 范围查询

```
curl -XGET 'localhost:9200/employee/_search?q=on_board_date:(>2014-01-01%20AND%20<2015-12-31)' 
```

21.2.8. 保留字符

如果您需要在 <QUERY_STRING> 中包含以下Transwarp ES保留字符，您需要使用 \ 进行转译：

+ - = && || > < ! () { } [] ^ " ~ * ? : \ /

21.2.9. 复杂查询

当我们想要进行一些复杂的查询时，<QUERY_STRING> 会变得非常难以理解和维护，同时会非常容易出错。所以 我们建议只用URI检索进行非常简单的查询，尽量使用[\[requestb-body-search-chapter\]](#)中介绍的方法。

22. Request Body检索（Elasticsearch 1.3.1）



本章描述适用于TDH4.5及之前的版本（Elasticsearch 版本为1.3.1）。如果您使用4.6及之后的TDH版本（Elasticsearch版本为2.0.0），请参考[Request Body检索（Elasticsearch 2.0.0）](#)。

使用Request Body检索时，检索参数会通过HTTP Request Body传递：

RequestBody检索格式

```
curl -XPOST '<HOST>:9200/[<PATH>]/_search?<PARAMETERS>' -d '{
  "query" : {<BODY>} ②
  "<SEARCH_PARAMETER>" : {<VALUE>} ③
}'
```

- ① 这里的HTTP方法可以是 **POST** 也可以是 **GET**，使用方法没有区别。有些客户端不支持 **GET** 和Request Body合用，这时就要使用 **POST**。
- ② 查询通过 **<BODY>** 来表达。**<BODY>** 的书写使用Transwarp ES的[Query DSL语法](#)。
- ③ Request Body中还可以包含其他的检索参数，例如排序参数 **sort**。

RequestBody检索相对URI检索表达能力、可读性都更强。URI检索能做到的查询RequestBody检索都能做到。

22.1. Query DSL

Query DSL是用于构造查询体**<BODY>**的语言。本节我们介绍Query DSL查询体的结构。为了表述简洁，除非另外指出，本节给出的语法和示例都默认包含在 **BODY** 中。在实际操作中，您应当将 **curl** 指令以及Request Body的其他部分补齐。例如，当我们下面查询时：

```
{
  "match": {
    "lastname": "li"
  }
}
```

实际操作中需要执行：

```
curl -XPOST '<HOST>:9200/[<PATH>]/_search?<PARAMETERS>' -d '{
  "query" : {
    "match": {
      "lastname": "li"
    }
  }
}'
```

22.1.1. 查询语句

查询体由 **查询语句** 构成，Query DSL中的语句一般有如下形式：

语法：Query DSL查询语句

```
{
  "<QUERY_NAME>": { ①
    "<ARGUMENT>": <VALUE>, ②
    "<ARGUMENT>": <VALUE>,
    ...
  }
}
```

① **<QUERY_NAME>** 为查询语句的种类，见[常见Transwarp ES查询语句](#)。

② **<ARGUMENT>** 为该语句接受的一些参数。

如果查询语句使用字段名，那么参数要放在字段名下一级：

语法：Query DSL查询语句（使用字段名）

```
{
  "<QUERY_NAME>": {
    "<FIELD_NAME>": {
      "<ARGUMENT>": <VALUE>,
      "<ARGUMENT>": <VALUE>, ...
    }
  }
}
```

22.1.2. 查询语句的组合和嵌套

一个查询中可以包含多个查询语句，语句还可以互相嵌套。如果把一个查询看做一个抽象语法树（AST），那么一个查询中有两类语句：

- **叶语句：**语句中将一个或多个字段和一个查询串做比较。叶语句中不嵌套其他语句，例如：

```
{
  "match": {
    "lastname": "li"
  }
}
```

- 复合语句：嵌套了其他语句的语句。例如：

```
{
  "bool": {
    "must": {
      "term": { "lastname": "li" }
    },
    "must_not": {
      "range": {
        "on_board_date": { "from": 2014-01-01, "to": 2015-12-
31 }
      }
    }
}
```

Transwarp ES的语句可以多层嵌套，用于构造非常复杂的查询。

22.1.3. Query和Filter

Transwarp ES中的查询语句分两种：query和filter。一些同名查询语句既是query也是filter，例如 **term**，**terms** 等等。同一查询语句作为query和filter时使用形式相似，但是表现不同：

- 作为query，查询语句回答的问题是“查询串和字段的匹配程度如何”，例如找出和 **I love Beijing Opera** 最匹配的Document，所以 **query**会计算关联分 **_score** 来衡量Document的关联度。query常用于做全文检索。
- 作为filter，查询语句回答的问题是“查询串是否和字段匹配”，所以filter的回答是二元的：是或不是。filter不计算关联分。下面是两个filter回答的问题：
 - 是否在2014-01-01和2015-12-31之间入职（**on_board_date** 字段是否在 **2014-01-01** 和`2015-12-31` 之间）？
 - Status是否是active（**status_active** 字段是否为 **true**）？

filter用于做精确匹配。

书写查询时，“**query**”对应的查询语句为query：

语法：query

```
"query": {
  查询语句在这里为query
}
```

filter 对应的查询语句为filter

语法：filter

```
"filter": {  
    查询语句在这里为filter  
}
```

同一个查询中，filter和query可以互相结合。要结合filter和query，filter和query都要放在 **filtered** 查询语句中：

语法：结合filter和query

```
{  
    "filtered": {  
        "query": {...  
        "filter": {...  
    }  
}  
}
```

例 130. 结合filter和query

```
{  
    "filtered": {  
        "query": {  
            "match": {"lastname": "li"}  
        },  
        "filter": {  
            "term": {"married": true}  
        }  
    }  
}
```

23. Request Body检索（Elasticsearch 2.0.0）



本章描述适用于TDH4.6及以后（Elasticsearch版本为2.0.0）。如果您使用4.5及之前的TDH版本（Elasticsearch版本为1.3.1），请参考[Request Body检索（Elasticsearch 1.3.1）](#)。

使用Request Body检索时，检索参数会通过HTTP Request Body传递：

Request Body检索格式

```
curl -XPOST '<HOST>:9200/[<PATH>]/_search?<PARAMETERS>' -d '{
  "query" : {<BODY>} ②
  "filter" : {<BODY>} ③
  "<SEARCH_PARAMETER>" : {<VALUE>} ④
}'
```

- ① 这里的HTTP方法可以是 **POST** 也可以是 **GET**，使用方法没有区别。有些客户端不支持 **GET** 和Request Body合用，这时就要使用 **POST**。
- ② 查询通过**query** 或 **filter** 中的 **查询体 <BODY>** 来表达。**<BODY>** 的书写使用Transwarp ES的**Query DSL语法**。
- ③ 同上。
- ④ Request Body中还可以包含其他的检索参数，例如排序参数 **sort**。

Request body检索相对URI检索表达能力、可读性都更强。URI检索能做到的查询Request body检索都能做到。

23.1. Query DSL

Query DSL是用于构造查询体**<BODY>**的语言。本节我们介绍Query DSL查询体的结构。为了表述简洁，除非另外指出，本节给出的语法和示例都默认包含在 **BODY** 中。在实际操作中，您应当将 **curl** 指令以及Request Body的其他部分补齐。例如，当我们下面查询时：

```
{
  "match": {
    "lastname": "li"
  }
}
```

实际操作中需要执行：

```
curl -XPOST '<HOST>:9200/[<PATH>]/_search?<PARAMETERS>' -d '{
  "query" : {
    "match": {
      "lastname": "li"
    }
  }
}'
```

23.1.1. 查询语句

查询体由 **查询语句** 构成，Query DSL中的语句一般有如下形式：

语法：Query DSL查询语句

```
{
  "<QUERY_NAME>": { ①
    "<ARGUMENT>": <VALUE>, ②
    "<ARGUMENT>": <VALUE>,
    ...
  }
}
```

① **<QUERY_NAME>** 为查询语句的种类，见[常见Transwarp ES查询语句](#)。

② **<ARGUMENT>** 为该语句接受的一些参数。

如果查询语句使用字段名，那么参数要放在字段名下一级：

语法：Query DSL查询语句（使用字段名）

```
{
  "<QUERY_NAME>": {
    "<FIELD_NAME>": {
      "<ARGUMENT>": <VALUE>,
      "<ARGUMENT>": <VALUE>, ...
    }
  }
}
```

23.1.2. 查询语句的组合和嵌套

一个查询中可以包含多个查询语句，语句还可以互相嵌套。如果把一个查询看做一个抽象语法树（AST），那么一个查询中有两类语句：

- **叶语句：**语句中将一个或多个字段和一个查询串做比较。叶语句中不嵌套其他语句，例如：

```
{
  "match": {
    "lastname": "li"
  }
}
```

- 复合语句：嵌套了其他语句的语句。例如：

```
{
  "bool" : {
    "must" : {
      "term" : { "lastname" : "li" }
    },
    "must_not" : {
      "range" : {
        "on_board_date" : { "from" : 2014-01-01, "to" : 2015-12-
31 }
      }
    }
}
```

Transwarp ES的语句可以多层嵌套，用于构造非常复杂的查询。

23.1.3. Query Context和Filter Context

Transwarp ES中有两种Context: query context和filter context。查询语句在不同context中的用法相同，但是表现不同：

- query context中，查询回答的问题是“查询串和字段的匹配程度如何”，例如找出和 **I love Beijing Opera** 最匹配的Document，所以 **查询会计算关联分 score** 来衡量Document的关联度。query context常用于做全文检索。
- filter context中，查询回答的问题是“查询串是否和字段匹配”，所以回答是二元的：是或不是。**过滤不计算关联分**。下面是两个filter context中的查询回答的问题：
 - 是否在2014-01-01和2015-12-31之间入职 (**on_board_date** 字段是否在 **2014-01-01** 和`**2015-12-31`** 之间)？
 - Status是否是active (**status_active** 字段是否为 **true**)？

query context常用于做精确匹配。

书写时，**query** 字段中的语句处在query context下：

语法：query context

```
"query": {
  这里为query context
}
```

filter 字段中的语句则处在filter context下:

语法: filter context

```
"filter":{  
    这里为filter context  
}
```

filter和query context可以结合使用。在Elasticsearch 1.3.1中，结合query和filter的方式为将它们放在**filtered**语句中。在Transwarp ES 2.0.0中，不推荐再使用 **filtered** 语句，而是建议使用 query context中的**bool** 查询语句来结合filter和query context: 将原query中的查询语句放在 **must** 子句中，将filter中的查询语句放在filter context中。例如结合filter和query可以改写为:

例 131. 用 **bool** 语句结合filter和query context

```
{  
    "query": { ①  
        "bool": {  
            "must": {  
                "match": {"lastname": "li"} ②  
            },  
            "filter": { ③  
                "term": {"married": true} ④  
            }  
        }  
    }  
}
```

① **query** 标志着内部是query context。

② **match** 查询在query context中。

③ **filter** 标志着内部是filter context。

④ **term** 查询在filter context中。

24. 映射 (mapping) 与分词 (analysis) 基础

映射 (mapping) 机制用于进行字段类型确认，将每个字段匹配为一种确定的数据类型 (string, number, booleans, date 等)。

分词 (analysis) 机制用于进行全文文本 (Full Text) 的分词，以建立供搜索用的反向索引。

24.1. 数据类型差异

当在索引中处理数据时，我们注意到一些奇怪的事，有些东西似乎被破坏了。下面Index中有12个 tweets，只有一个包含日期2014-09-15，但是我们看看查询中的total hits:

```
GET /_search?q=2014          # 12 results
GET /_search?q=2014-09-15    # 12 results !
GET /_search?q=date:2014-09-15 # 1 result
GET /_search?q=date:2014      # 0 results !
```

为什么全日期的查询返回所有的 tweets，而针对 date 字段进行年度查询却什么都不返回？为什么我们的结果会因为查询 _all 字段(默认所有字段中进行查询)或 date 字段而变得不同？

可以想到，这是因为我们的数据在 _all 字段的索引方式和在 date 字段的索引方式不同而导致。让我们看看Elasticsearch在对 gb 索引中的 tweet 类型进行映射后是如何解读我们的文档结构的：

```
GET /gb/_mapping/tweet
```

返回:

```
{
  "gb": {
    "mappings": {
      "tweet": {
        "properties": {
          "date": {
            "type": "date",
            "format": "strict_date_optional_time||epoch_millis"
          },
          "name": {
            "type": "string"
          },
          "tweet": {
            "type": "string"
          },
          "user_id": {
            "type": "long"
          }
        }
      }
    }
  }
}
```

Elasticsearch为对字段类型进行猜测，动态生成了字段和类型的映射（mapping）关系。返回的信息显示了 `date` 字段被识别为 `date` 类型。 `_all` 因为是默认字段所以没有在此显示，不过我们知道它是 `string` 类型。

`date` 类型的字段和 `string` 类型的字段的索引方式是不同的，因此导致查询结果的不同。你可能会期望每一种核心数据类型(strings, numbers, booleans及dates)以不同的方式进行索引，事实也正是这样：在Elasticsearch中他们是被区别对待的。

但是更大的区别在于确切值(`exact values`) (比如 `string` 类型)及全文文本(`full text`)之间。这两者的区别才真的很重要——这是区分搜索引擎和其他数据库的关键所在。

24.2. 确切值(Exact values) vs. 全文文本(Full text)

Elasticsearch中的数据可以大致分为两种类型：确切值(`exact values`)及全文文本(`full text`)

确切值 是确定的，正如它的名字一样。比如一个date或用户ID，也可以包含更多的字符串比如username或email地址。确切值“Foo”和“foo”就并不相同。确切值2014和2014-09-15也不相同。

全文文本，从另一个角度来说是文本化的数据(常常以人类的语言书写)，比如一篇Twitter文章或邮件正文。

全文文本常常被称为非结构化数据，其实是一种用词不当的称谓，实际上自然语言是高度结构化的。问题是自然语言的语法规则是如此的复杂，计算机难以正确解析。例如这个句子：



May is fun but June bores me.

到底是说的月份还是人呢？

确切值是很容易查询的，因为结果是二进制的——要么匹配，要么不匹配。这种查询很容易以SQL表达：

```
WHERE name      = "John Smith"
  AND user_id = 2
  AND date     > "2014-09-15"
```

而对于全文数据的查询来说，却有些微妙。我们不会去询问 **这篇文档是否匹配查询要求**，而会询问 **这篇文档和查询的匹配程度如何**？也就是说，对于查询条件，这篇文档的相关性有多高？

我们很少确切地匹配整个全文文本。我们想在全文中查询包含查询文本的部分。不仅如此，我们还期望搜索引擎能理解我们的意图：

- 一个针对 **UK** 的查询将返回涉及 **United Kingdom** 的文档
- 一个针对 **jump** 的查询同时能够匹配 **jumped**，**jumps**，**jumping** 甚至 **leap**
- **johnny walker** 也能匹配 **Johnnie Walker**，**johnnie depp** 及 **Johnny Depp**
- **fox news hunting** 能返回有关**hunting on Fox News**的故事，而 **fox hunting news** 也能返回关于**fox hunting**的新闻故事。

为了方便在全文文本字段中进行这些类型的查询，Elasticsearch首先对文本分词(**analyze**)，然后使用结果建立一个倒排索引(**inverted index**)。我们下面讨论倒排索引及分词过程。

24.3. 倒排索引(inverted index)

Elasticsearch使用一种叫做倒排索引(**inverted index**)的结构来做快速的全文搜索。倒排索引由在文档中出现的唯一的单词列表，以及对于每个单词在文档中的位置组成。

例如，我们有两个文档，每个文档 **content** 字段包含：

1. The quick brown fox jumped over the lazy dog
2. Quick brown foxes leap over lazy dogs in summer

为了创建倒排索引，我们首先切分每个文档的**content**字段为单独的单词（我们把它们叫做 **terms** 或者 **tokens**），把所有的唯一词放入列表并排序，结果是这个样子的：

Term	Doc_1	Doc_2
Quick		X
The	X	
brown	X	X
dog	X	
dogs		X
fox	X	
foxes		X
in		X
jumped	X	
lazy	X	X
leap		X
over	X	X
quick	X	
summer		X
the	X	

现在，如果我们想搜索 **quick brown**，我们只需要找到每个词在哪个文档中出现即可：

Term	Doc_1	Doc_2
brown	X	X
quick	X	
Total	2	1

两个文档都匹配，但是第一个比第二个有更多的匹配项。如果我们加入简单的相似度算法(**similarity algorithm**)，计算匹配单词的数目，这样我们就可以说第一个文档比第二个匹配度更高——对于我们的查询具有更多相关性。

但是在我们的倒排索引中还有些问题：

- **Quick** 和 **quick** 被认为是不同的单词，但是用户可能认为它们是相同的。
- **fox** 和 **foxes** 很相似——它们都是同根词。
- **jumped** 和 **leap** 不是同根词，但意思相似——它们是同义词。

上面的索引中，搜索 **+Quick +fox** 不会匹配任何文档（记住，前缀 **+** 表示单词必须匹配到）。只有 **Quick** 和 **fox** 都在同一文档中才可以匹配查询。如何才能合理地让两个文档都匹配查询？

如果我们将词统一为标准格式，这样就可以找到不是确切匹配查询，但是足以相似，从而可以关联的文档。例如：

- **Quick** 可以转为小写成为 **quick**。
- **foxes** 可以被转为根形式 **fox**。同理 **dogs** 可以被转为 **dog**。
- **jumped** 和 **leap** 同义就可以只索引为单个词 **jump**

现在的索引：

Term	Doc_1	Doc_2
brown	X	X
dog	X	X
fox	X	X
in		X
jump	X	X
lazy	X	X
over	X	X
quick	X	X
summer		X
the	X	X

但这样还不够。我们的搜索 `+Quick +fox` 依旧失败，因为 `Quick` 的确切值已经不在索引里。如果我们使用相同的标准化规则处理查询字符串的 `content` 字段，查询将变成 `+quick +fox`，这样就可以匹配到两个文档。



这很重要：你只可以找到 确实 存在于索引中的词，所以索引文本和查询字符串都要标准化为相同的形式！

24.4. 分词与分词器

分词是这样一个过程：首先，将一个文本块拆分为单独的词（`term`），然后标准化这些词为标准形式，提高它们的“可搜索性”或“查全率”，而后构建倒排索引。而这些工作就是由分词器完成的。

24.4.1. 内建的分词器

Elasticsearch附带了一些预装的分词器，你可以直接使用它们。下面我们列出了重要的几个分词器，来演示这个字符串分词后的表现差异：

`Set the shape to semi-transparent by calling set_trans(5)`

- 标准分词器(Standard analyzer)

标准分词器是Elasticsearch默认使用的分词器。对于文本分词，如果没有什么特殊要求，它是够用的。它根据Unicode Consortium的定义的单词边界(word boundaries)来切分文本，然后去掉大部分标点符号。最后，把所有词转为小写。产生的结果为：

`set, the, shape, to, semi, transparent, by, calling, set_trans, 5`

- 简单分词器(Simple analyzer)

简单分词器将非单个字母的文本切分，然后把每个词转为小写。产生的结果为：

`set, the, shape, to, semi, transparent, by, calling, set, trans`

- 空格分词器(Whitespace analyzer)

空格分词器依据空格切分文本。它不转换小写。产生结果为：

```
Set, the, shape, to, semi-transparent, by, calling, set_trans(5)
```

- 语言分词器(Language analyzer)

特定语言分词器适用于很多语言。它们能够考虑到特定语言的特性。例如，**english** 分词器自带一套英语停用词库——像 **and** 或 **the** 这些与语义无关的通用词。这些词被移除后，因为语法规则的存在，英语单词的主体含义依旧能被理解。

english 分词器将会产生以下结果：

```
set, shape, semi, transpar, call, set_tran, 5
```

注意 **transparent**、**calling** 和 **set_trans** 是如何转为词干的。

24.4.2. 分词器的使用

当我们索引(index)一个文档，全文字段会被分词为单独的词来创建倒排索引。不过，当我们在全文字段搜索(search)时，我们要让查询字符串经过同样的分词流程处理，以确保这些词在索引中存在。

全文查询我们将在稍后讨论。只有理解每个字段是如何定义的，这样才可以得到想要的结果：

- 当你查询全文(**full text**)字段，查询将使用相同的分词器来将查询字符串分词，以产生正确的词列表。
- 当你查询一个确切值(**exact value**)字段，查询不将查询字符串分词，但是你可以自己指定。

现在你可以明白为什么**映射和分词**的开头会产生那种结果：

- **date** 字段包含一个确切值：单独的一个词 **2014-09-15**。
- **_all** 字段是一个全文字段，所以分词过程将日期转为三个词：**2014**、**09** 和 **15**。

当我们在 **_all** 字段查询 **2014**，它一个匹配到12条推文，因为这些推文都包含词 **2014**：

```
GET /_search?q=2014 # 12 results
```

当我们在 **_all** 字段中查询 **2014-09-15**，首先将查询字符串分词，产生匹配任一词 **2014**、**09** 或 **15** 的查询语句，它依旧匹配12个推文，因为它们都包含词 **2014**。

```
GET /_search?q=2014-09-15 # 12 results !
```

当我们在 **date** 字段中查询 **2014-09-15**，它查询一个确切的日期，然后只找到一条推文：

```
GET /_search?q=date:2014-09-15 # 1 result
```

当我们在 `date` 字段中查询 `2014`，没有找到文档，因为没有文档包含那个确切的日期：

```
GET /_search?q=date:2014 # 0 results !
```

24.4.3. 测试分词器

尤其当你是Elasticsearch新手时，对于如何分词以及存储到索引中理解起来比较困难。为了更好的理解如何进行，你可以使用 `analyze` API来查看文本是如何被分词的。在查询字符串参数中指定要使用的分词器，被分词的文本作为请求体：

```
GET /_analyze
{
  "analyzer": "standard",
  "text": "Text to analyze"
}
```

结果中每个节点在代表一个词：

```
{
  "tokens": [
    {
      "token": "Text",
      "start_offset": 0,
      "end_offset": 4,
      "type": "<ALPHANUM>",
      "position": 1
    },
    {
      "token": "to",
      "start_offset": 5,
      "end_offset": 7,
      "type": "<ALPHANUM>",
      "position": 2
    },
    {
      "token": "analyze",
      "start_offset": 8,
      "end_offset": 15,
      "type": "<ALPHANUM>",
      "position": 3
    }
  ]
}
```

`token` 是一个实际被存储在索引中的词； `position` 指明词在原文本中是第几个出现的； `start_offset` 和 `end_offset` 表示词在原文本中占据的位置。

24.4.4. 指定分词器

当Elasticsearch在你的文档中探测到一个新的字符串字段，它将自动设置它为全文 `string` 字段并用标准分词器分词。

你不可能总是想要这样做。也许你想使用一个更适合这个数据的语言分词器。或者，你只想把字符串字段当作一个普通的字段——不做任何分词，只存储确切值，就像字符串类型的用户ID或者内部状态字段或者标签。

为了达到这种效果，我们必须通过mapping人工设置这些字段。

24.5. 映射(mapping)

为了能够把日期字段处理成日期，把数字字段处理成数字，把字符串字段处理成全文本（`Full-text`）或精确的字符串值，Elasticsearch需要知道每个字段里面都包含了什么类型。这些类型和字段的信息存储（包含）在mapping中。

索引中每个文档都从属于一个类型（`type`），每个 `type` 拥有自己的mapping。一个mapping在 `type` 中定义了字段，每个字段的数据类型，以及字段被Elasticsearch处理的方式。Mapping还用于设置关联到类型上的元数据。

24.5.1. 核心简单字段类型

Elasticsearch支持以下简单字段类型：

- String: `string`
- Whole number: `byte`, `short`, `integer`, `long`
- Floating-point: `float`, `double`
- Boolean: `boolean`
- Date: `date`

当你索引一个包含新字段的文档——一个之前没有的字段——Elasticsearch将使用动态映射猜测字段类型，这类型来自于JSON的基本数据类型，使用以下规则：

JSON type	Field type
Boolean: <code>true</code> or <code>false</code>	<code>boolean</code>
Whole number: <code>123</code>	<code>long</code>
Floating point: <code>123.45</code>	<code>double</code>
String, valid date: <code>2014-09-15</code>	<code>date</code>
String: <code>foo bar</code>	<code>string</code>



这意味着，如果你索引一个带引号的数字——`123`，它将被映射为 `string` 类型，而不是 `long` 类型。然而，如果字段已经被映射为 `long` 类型，Elasticsearch将尝试转换字符串为 `long`，并在转换失败时会抛出异常。

24.5.2. 查看mapping

我们可以使用 `_mapping` 后缀来查看Elasticsearch中的mapping。在本章开始我们已经找到索引 `gb` 类型 `tweet` 中的mapping:

```
GET /gb/_mapping/tweet
```

这展示给了我们字段的mapping（叫做属性 `properties`），这些mapping是Elasticsearch在创建索引时动态生成的：

```
{
  "gb": {
    "mappings": {
      "tweet": {
        "properties": {
          "date": {
            "type": "date",
            "format": "strict_date_optional_time||epoch_millis"
          },
          "name": {
            "type": "string"
          },
          "tweet": {
            "type": "string"
          },
          "user_id": {
            "type": "long"
          }
        }
      }
    }
  }
}
```



错误的mapping会造成查询结果混乱，例如把 `age` 字段映射为 `string` 类型而不是 `integer` 类型。请检查mapping类型。

24.5.3. 自定义字段mapping

虽然大多数情况下基本数据类型已经能够满足，但你也会经常需要自定义一些特殊类型（fields），特别是字符串字段类型。自定义类型可以使你完成一下几点：

- 区分 **全文字符串字段** 和 **准确字符串字段**（即分词与不分词，全文的一般要分词，准确的就不需要分词）。
- 使用特定语言的分词器（译者注：例如中文、英文、阿拉伯语，不同文字的断字、断词方式的差异）
- 优化部分匹配字段
- 指定自定义日期格式（例如英文的 `Feb, 12, 2016` 和中文的 `2016年2月12日`）
- 以及更多

mapping中最重要的字段参数是 `type`。除了 `string` 类型的字段，你可能很少需要指定除 `type` 外的其他值：

```
{
  "number_of_clicks": {
    "type": "integer"
  }
}
```

`string` 类型的字段，默认的，考虑到包含全文本，它们的值在索引前要经过分词器分词，并且在全文搜索此字段前要把查询语句做分词处理。

对于 `string` 字段，两个最重要的mapping参数是 `index` 和 `analyzer`。

- `index`

`index` 参数控制字符串以何种方式被索引。它包含以下三个值当中的一个：

`analyzed`

首先将字符串分词，然后索引。换言之，以全文形式索引此字段。

`not_analyzed`

索引这个字段，使之可以被搜索，但是索引内容和指定值一样。不对此字段分词。

`no`

不索引这个字段。这个字段不能为搜索到。

`string` 类型字段默认值是 `analyzed`。如果我们想映射字段为确切值，我们需要设置它为 `not_analyzed`：

```
{
  "tag": {
    "type": "string",
    "index": "not_analyzed"
  }
}
```



其他简单类型（`long`、`double`、`date` 等等）也接受 `index` 参数，但相应的值只能是 `no` 和 `not_analyzed`，它们的值不能被分词。

- `analyzer`

对于 `analyzed` 类型的字符串字段，使用 `analyzer` 参数来指定在搜索和索引的时候使用哪一种分词器。默认的是Elasticsearch使用标准分词器，但是你可以通过指定一个内建的分词器来更改它，例如 `whitespace`、`simple` 或 `english`。

```
{
  "tweet": {
    "type": "string",
    "analyzer": "english"
  }
}
```

24.5.4. 更新mapping

你可以在第一次创建索引的时候指定mapping的类型。此外，你也可以之后为新类型添加mapping（或者为已有的类型更新mapping）。



你可以向已有mapping中增加字段，但你 **不能修改** 它。如果一个字段在mapping中已经存在，这可能意味着那个字段的数据已经被索引。如果你改变了字段mapping，那已经被索引的数据将错误并且不能被正确的搜索到。

我们可以更新一个mapping来增加一个新字段，但是不能把已有字段的类型从 **analyzed** 改到 **not_analyzed**。

为了演示两个指定的mapping方法，让我们首先删除索引 **gb**：

```
DELETE /gb
```

然后创建一个新索引，指定 **tweet** 字段的分词器为 **english**：

```
PUT /gb ①
{
  "mappings": {
    "tweet": {
      "properties": {
        "tweet": {
          "type": "string",
          "analyzer": "english"
        },
        "date": {
          "type": "date"
        },
        "name": {
          "type": "string"
        },
        "user_id": {
          "type": "long"
        }
      }
    }
  }
}
```

① 这将创建包含 **mappings** 的索引，mapping在请求体中指定。

再后来，我们决定在 `tweet` 的mapping中增加一个新的 `not_analyzed` 类型的文本字段，叫做 `tag`，使用 `_mapping` 后缀：

```
PUT /gb/_mapping/tweet
{
  "properties" : {
    "tag" : {
      "type" : "string",
      "index": "not_analyzed"
    }
  }
}
```

注意到我们不再需要列出所有的已经存在的字段，因为我们没法修改他们。我们的新字段已经被合并至存在的那个mapping中。

24.5.5. 测试mapping

你可以通过名字使用 `analyze` API测试字符串字段的mapping。对比这两个请求的输出：

```
GET /gb/_analyze
{
  "field": "tweet",
  "text": "Black-cats" ①
}

GET /gb/_analyze
{
  "field": "tag",
  "text": "Black-cats" ①
}
```

① 我们想要分词的文本被放在请求体中。

`tweet` 字段产生两个词， `black` 和 `cat`， `tag` 字段产生单独的一个词 `Black-cats`。换言之，我们的mapping工作正常。

24.6. 复合核心字段类型

除了之前提到的简单的标量类型，JSON还有null值，数组和对象，所有这些Elasticsearch都支持：

24.6.1. 多值字段

我们很可能想让 `tag` 字段包含多个字段。我们可以索引一个标签数组来代替单一字符串：

```
{ "tag": [ "search", "nosql" ]}
```

对于数组不需要特殊的mapping。任何一个字段可以包含零个、一个或多个值，同样对于全文字段将被分词处理

并产生多个词。

言外之意，这意味着数组中所有值必须为同一类型。你不能把日期和字符串混合。如果你创建一个新字段，这个字段索引了一个数组，Elasticsearch将使用第一个值的类型来确定这个新字段的类型。



当你从Elasticsearch中取回一个文档，任何一个数组的顺序和你索引它们的顺序一致。你取回的 `_source` 字段的顺序同样与索引它们的顺序相同。

然而，数组是作为多值字段被索引的，它们没有顺序。在搜索阶段你不能指定“第一个值”或者“最后一个值”。倒不如把数组当作一个值集合(bag of values)

24.6.2. 空字段

当然数组可以是空的。这等价于有零个值。事实上，Lucene没法存放 `null` 值，所以一个 `null` 值的字段被认为是空字段。

这三个字段将被识别为空字段而不被索引：

```
"null_value": null,
"empty_array": [],
"array_with_null_value": [ null ]
```

24.6.3. 多层对象

我们需要讨论的最后一个自然JSON数据类型是对象(`object`)——在其它语言中叫做hash、hashmap、dictionary 或者 associative array。

内部对象(`inner objects`)经常用于在另一个对象中嵌入一个实体或对象。例如，作为在 `tweet` 文档中 `user_name` 和 `user_id` 的替代，我们可以这样写：

```
{
  "tweet": "Elasticsearch is very flexible",
  "user": {
    "id": "@johnsmith",
    "gender": "male",
    "age": 26,
    "name": {
      "full": "John Smith",
      "first": "John",
      "last": "Smith"
    }
  }
}
```

24.6.4. 内部对象的mapping

Elasticsearch 会动态的检测新对象的字段，并且映射它们为 `object` 类型，将每个字段加到 `properties` 字段下：

```
{
  "gb": {
    "tweet": { ①
      "properties": {
        "tweet": { "type": "string" },
        "user": { ②
          "type": "object",
          "properties": {
            "id": { "type": "string" },
            "gender": { "type": "string" },
            "age": { "type": "long" },
            "name": { ②
              "type": "object",
              "properties": {
                "full": { "type": "string" },
                "first": { "type": "string" },
                "last": { "type": "string" }
              }
            }
          }
        }
      }
    }
  }
}
```

① 根对象

② 内部对象

对 `user` 和 `name` 字段的mapping与 `tweet` 类型自己很相似。事实上，`type` mapping只是 `object` mapping的一种特殊类型，我们将 `object` 称为根对象。它与其他对象一模一样，除非它有一些特殊的顶层字段，比如 `_source`，`_all` 等等。

24.6.5. 内部对象是怎样被索引的

Lucene并不了解内部对象。一个Lucene文件包含一个键-值对应的扁平表单。为了让Elasticsearch可以有效的索引内部对象，将文件转换为以下格式：

```
{
  "tweet": [elasticsearch, flexible, very],
  "user.id": [@johnsmith],
  "user.gender": [male],
  "user.age": [26],
  "user.name.full": [john, smith],
  "user.name.first": [john],
  "user.name.last": [smith]
}
```

内部栏位可被归类至name，例如 `first`。为了区别两个拥有相同名字的栏位，我们可以使用完整路径，例如 `user.name.first` 或甚至类型名称加上路径： `tweet.user.name.first`。



在以上扁平化文件中，并没有栏位叫作 `user` 也没有栏位叫作 `user.name`。Lucene 只索引阶层或简单的值，而不会索引复杂的资料结构。

24.6.6. 内部对象数组

最后，一个包含内部对象的数组如何索引。我们有个数组如下所示：

```
{
  "followers": [
    { "age": 35, "name": "Mary White" },
    { "age": 26, "name": "Alex Jones" },
    { "age": 19, "name": "Lisa Smith" }
  ]
}
```

此文件会如我们以上所说的被扁平化，但其结果会像如此：

```
{
  "followers.age": [19, 26, 35],
  "followers.name": [alex, jones, lisa, smith, mary, white]
}
```

`{age: 35}` 与 `{name: Mary White}` 之间的关联会消失，因每个多值的栏位会变成一个值集合，而非有序的阵列。这让我们可以知道：

- 是否有26岁的追随者？

但我们无法取得准确的资料如：

- 是否有26岁的追随者且名字叫Alex Jones？

关联内部对象可解决此类问题，我们称之为嵌套对象。

25. Mapping操作

Mapping定义了Transwarp ES如何对一个Index中的Document进行保存并建倒排索引。Mapping可以定义的内容包括：

- 各个字段的数据类型。
- String类型字段是否应该做为全文本（full text）还是精确值（exact values）处理。
- 是否保存某字段的原始数据。
- 日期值的格式。

Mapping可以在创建Index时显式定义，Mapping也可以不显式定义。如果您不显式定义Mapping，Transwarp ES的动态Mapping功能会根据编入Index的Document的数据自动分辨各自段数据类型，并按照Transwarp ES的默认配置定义Mapping。然而Transwarp ES的默认配置并不总能满足您的检索需求，这时您需要对Mapping进行显式定义。Mapping的显式定义需要在编入Transwarp ES Index中的第一个Document之前完成，而且Transwarp ES Index中有数据后，星环信息科技（上海）有限公司强烈建议不要修改Mapping。由于Mapping直接影响检索结果，无论是显式定义还是动态定义，我们都建议您在检索前了解您Index的Mapping。

25.1. 如何显式定义Mapping

显式定义Mapping需要在创建Index时进行，格式如下：

语法：创建Index时显式定义Mapping

```
curl -XPUT '<HOST>:9200/<INDEX>?pretty' -d '①
{
  "mappings" : {
    "default_type_" : {
      <META-FIELD_MAPPING>, ②
      <META-FIELD_MAPPING>,
      ...
      "properties" : { ③
        <FIELD_MAPPING>,
        <FIELD_MAPPING>,
        ...
      }
    }
  }
}
```

① 创建名为 `<INDEX>` 的Index。

② Mapping中可以包含零到多个元数据字段的Mapping（Meta-field Mapping）。元数据字段都以下划线开头，例如 `_source`, `_all` 等。用于定义Index元数据字段的表现，详情请参考[定义元数据字段Mapping \(META-FIELD_MAPPING\)](#)。

③ `properties` 描述Index中的各个Document字段的Mapping（Field Mapping），每个 `<FIELD_MAPPING>` 对应一个字段的Mapping。字段Mapping包含字段的数据类型、分词器等信息，详情请参考[定义字段Mapping \(FIELD_MAPPING\)](#)。

下面是一个显式定义Mapping的示例:

例 132. 显式定义Mapping

```
curl -XPUT 'localhost:9200/sample_index?pretty' -d '
{
  "mappings" : {
    "default_type_" : {
      "_all" : { ①
        "enabled" : false
      },
      "_source" : { ②
        "enabled" : true
      },
      "properties" : {
        "d:C1" : { ③
          "type" : "string",
          "index" : "not_analyzed",
          "store" : true
        },
        "d:C2" : { ④
          "type" : "long",
          "store" : true
        }
      }
    }
  }
}'
```

① `_all` 元数据字段。

② `_source` 元数据字段。

③ 名为 `d:C1` 的字段的Mapping。该字段类型为string (`"type" : "string"`)，不分词 (`"index" : "not_analyzed"`) 并且存储原始数据 (`"store" : true`)。

④ 名为 `d:C2` 的字段的Mapping。该字段类型为long (`"type" : "long"`) 并且存储原始数据 (`"store" : true`)。

25.2. 定义元数据字段Mapping (META-FIELD_MAPPING)

Document不仅有字段 (Field)，还有一系列元数据字段 (Meta-Field)。Mapping可以控制Document元数据字段的行为。本节选取几个最重要的Document元数据字段，介绍它们的概念以及如何使用Mapping来修改其行为。

Document中较常用的Meta-Field包括（但不限于）：

- `_id`: Document的ID字符串。
- `_type`: Document所属Type。
- `_index`: Document所属的Index。
- `_uid`: `_type` 和 `_id` 相连：“type#id”

- **_source**: Document中的原始数据。默认状况下，**_source** 字段是存储的。但是 **_source** 字段的存储会占用磁盘空间，所以在有需求的情况下，您可以通过下面的方式关闭 **_source** 的存储：

```
{
  "mapping" : {
    "default_type_" : {
      "_source" : { "enabled" : false }
    }
  }
}
```

- **_all**: Document所有字段中数据首尾相连形成的字段。例如：

```
{
  "firstname" : "Si",
  "lastname" : "Li",
  "age" : 30,
  "on_board_date" : "2014-09-16",
  "hometown" : "Nanjing",
  "school" : "Beijing University",
  "married" : true,
  "about" : "cooking, mountain climbing"
}
```

的 **_all** 字段为：

```
Si Li 30 2014-09-16 Nanjing Beijing University true cooking mountain
climbing
```

在检索时如果不指定检索字段，Transwarp ES会在 **_all** 字段中进行匹配。通过Mapping可以设置关闭 **_all** 字段（Transwarp ES将不为Document生成 **_all** 字段）：

```
{
  "mapping" : {
    "default_type_" : {
      "_all" : { "enabled" : false }
    }
  }
}
```

Mapping还可以设置 **_all** 字段的分词器：

```
{
  "mapping" : {
    "default_type_" : {
      "_all" : { "analyzer" : "<analyzerName>" }
    }
  }
}
```

25.3. 定义字段Mapping (FIELD_MAPPING)

我们已经在[显式定义Mapping](#)中见过字段Mapping。字段的Mapping定义一般具有如下格式：

字段Mapping的格式

```
"<fieldName>" : {
  "<parameter>" : <value>,
  ...
}
```

其中比较重要的参数有

- **type**: 字段的数据类型。Transwarp ES支持的数据类型有：string, date, byte, short, integer, long, float, double和boolean。不显式定义Index Mapping，动态Mapping中会做根据Document字段中的内容自动做判断。
- **index**: 可以是analyzed、not_analyzed或no。如果设为no，则该字段不能被查询。analyzed/not_analyzed指定是否分词（仅对string类型字段有效，默认为analyzed）。
- **analyzer**: 字段使用的分词器（仅对string类型字段有效）。
- **store**: 可以是true或false。是否存储字段元数据。
- **doc_values**: 可以是true或false。在Elasticsearch 1.3.1中默认值为false，在Elasticsearch 2.0.0中默认值为true。如果该字段被用于排序或者聚合，我们推荐将 **doc_values** 设置为true。

例 133. 字段Mapping

```
"d:C5" : {
  "type" : "string", ①
  "store" : true, ②
  "index" : "analyzed", ③
  "analyzer" : "ik" ④
}
```

① 字段类型为string。

② 存储字段原始数据。

③ 为该字段分词。

④ 使用ik分词器。

26. 常见Transwarp ES查询语句

Query DSL提供一系列查询语句，用于完成不同类型的检索。例如下面的 `match_all` 查询匹配所有的Document，也就是做空检索：

`match_all` 查询

```
{
  "match_all": {}
}
```

Transwarp ES中的查询语句分为三个大类：[Full Text查询语句](#)，[Term-level查询语句](#)和[复合查询语句](#)。每个大类下又分别有多种不同的查询语句。

为了文档简洁明了，除非另外指出，本章的所有JSON文本都默认包含在[查询体](#)中。在实际操作中，您需要补上检索的其他元素，例如上面的 `match_all` 查询对应的实际命令行操作应当为：

```
curl -XPOST '<HOST>:9200/<PATH>/_search' -d '{
  "query": {
    "match_all": {}
  }
}'
```

26.1. Full Text查询语句

全文检索查询语句用于文本字段，例如邮件内容。全文检索查询语句会使用 被查询字段的分词器对查询串进行分词 后再执行查询。全文检索查询语句中最常见的是 `match` 查询语句。



本手册介绍Transwarp ES中最常用的Full Text查询语句，Elasticsearch中的Full Text查询不限于本手册的介绍范围，请参考Elasticsearch官方文档来了解其他Full Text查询语句。

26.1.1. `match` 查询语句

`match` 查询用于在指定的检索字段中检索文本、数值和日期类型的查询串。

语法: **match** 查询

```
{
  "match" : {
    "<FIELD_NAME>" : { ①
      "query" : "<QUERY_STRING>", ②
      "<PARAMETER>" : <VALUE>, ③
      ...
    }
  }
}
```

① 指定检索字段 `FIELD_NAME`。

② 提供检索串 `QUERY_STRING`。

③ 设置一些可选参数。

如果不设置可选参数, **match** 查询可以简写成:

语法: **match** 查询 (简写)

```
{
  "match" : {
    "<FIELD_NAME>" : "<QUERY_STRING>"
  }
}
```

match 查询是布尔型的, 意思是Transwarp ES会先将 `<queryString>` 分词, 然后用分词得到的词项构造一个 **bool** 查询, 该 **bool** 查询的默认逻辑运算符为 `or`, 要修改逻辑运算符可以使用 `operator` 参数, 如下:

语法: **match** 查询, 使用 `and` 作为逻辑运算符

```
{
  "match" : {
    "<FIELD_NAME>" : {
      "query" : "<QUERY_STRING>",
      "operator" : "and"
    }
  }
}
```

match 查询对 `queryString` 分词默认使用的分词器是 `fieldName` 的分词器。您也可以通过 `analyzer` 手动设置分词器:

语法: **match** 查询, 手动设置分词器:

```
{
  "match" : {
    "<FIELD_NAME>" : {
      "query" : "<QUERY_STRING>",
      "analyzer" : "<ANALYZER_NAME>"
    }
  }
}
```

例 134. **match** 查询

查询 **about** 字段中有

```
{
  "match": {
    "about": "beijing is great"
  }
}
```

26.2. Term-level查询语句

不像[全文检索查询语句](#)会对查询串分词, Term级别查询语句不对查询串分词, 而是将其和倒排索引中的数据进行精确匹配。Term级别的查询语句通常用于检索结构化数据, 例如数字、日期、布尔型数据等等。



本手册介绍Transwarp ES中最常用的Term-level查询语句, Elasticsearch中的Term-level查询不限于本手册的介绍范围, 请参考Elasticsearch官方文档来了解其他Term-level查询语句。

26.2.1. **term** 查询语句

term 查询语句的形式为:

语法: **term** 查询

```
{
  "term" : { "<FIELD_NAME>" : <TERM> }
}
```

term 查询返回 **FIELD_NAME** 字段包含和 **TERM** 完全相同 词项的Document。

例 135. term 查询

```
{
  "term" : {
    "married" : true
  }
}
```

26.2.2. terms 查询语句

terms 查询可以在指定字段中精确匹配多个词项，形式如下：

语法： **terms** 查询

```
{
  "terms" : {
    "<FIELD_NAME>" : [ <TERM1>, <TERM2>, ... ] ①
  }
}
```

① 用数组指定要匹配的多个词项。

例 136. terms 查询

```
{
  "terms" : {
    "school" : [ "nanjing", "beijing" ]
  }
}
```

26.2.3. range 查询语句

range 查询返回字段中词项在指定范围内的Document。

语法： **range** 查询

```
{
  "range" : {
    "<FIELD_NAME>" : {
      "gt"|"gte" : <LOWER_BOUND>, ①
      "lt"|"lte" : <UPPER_BOUND> ②
    }
  }
}
```

① 指定范围的下限，**gt** 表示大于，**gte** 表示大于等于。

② 指定范围的上限，**lt** 表示小于，**lte** 表示小于等于。

例 137. range 查询

```
{
  "range" : {
    "on_board_date" : {
      "gt" : "2013-12-31",
      "lte" : "2015-12-31"
    }
  }
}
```

26.2.4. wildcard 查询语句

wildcard 查询允许在查询串中使用通配符 * 和 ?。 * 用于匹配任意多个任意字符（包括0个字符）； ? 用于匹配任意的单个字符。



wildcard 查询因为需要比较非常多的Document，所以会比较慢。为了避免过慢的查询，查询串最好不要以 * 或 ? 开头。

例 138. wildcard 查询

```
{
  "wildcard" : { "school" : "bei*"}
```

26.3. 复合查询语句

复合查询语句有其他的查询语句嵌套在内。复合查询语句可以用于叠加不同语句产生的关联分、改变语句的表现或者在查询语境和过滤语境之间转换。



本手册介绍Transwarp ES中最常用的复合查询语句，Elasticsearch中的复合查询不限于本手册的介绍范围，请参考Elasticsearch官方文档来了解其他复合查询语句。

26.3.1. bool 查询语句 (Elasticsearch 1.3.1)



本节介绍的是TDH4.5及之前的版本 (Elasticsearch 版本为1.3.1) 中的 **bool** 查询，它的用法在4.6及之后的TDH版本 (Elasticsearch版本为2.0.0) 中略有不同。如果您使用的 是TDH4.6及以后的版本，请参考[bool 查询语句 \(Elasticsearch 2.0.0\)](#)。

bool 查询由逻辑运算符连接的 **bool** 子句构成，每个子句都有 成立类型 (occurrence type)，包括以下几种：

- **must**: 该子句必须成立。
- **should**: 该子句应该成立。如果一个 **bool** 查询没有 **must** 子句，那么至少一个 **should** 子句必须有匹配的Document。您也可以通过 **minimum_should_match** 参数来设置至少有几个 **should** 子句应该有

匹配的Document。

- **must_not**: 该子句必须不成立。

语法: **bool** 查询 (Elasticsearch 1.3.1)

```
{
  "bool" : {
    "must" : {
      <QUERY>
    },
    "should" : {
      <QUERY>
    },
    "must_not" : {
      <QUERY>
    },
    "minimum_should_match" : <n>
  }
}
```

must、**should** 和 **must_not** 中有多个查询语句时，用数组表示：

语法: **bool** 查询 (Elasticsearch 1.3.1)

```
{
  "bool" : {
    "must" : [ {<QUERY>} , {<QUERY>} , ... ],
    "should" : [ {<QUERY>} , {<QUERY>} , ... ],
    "must_not" : [ {<QUERY>} , {<QUERY>} , ... ],
    "minimum_should_match" : <n>
  }
}
```

对于 **bool** 查询来说，匹配越多越好，所以 **should** 和 **must** 子句为同一个Document贡献的 **_score** 会被相加算进Document的 **_score**。

例 139. `bool` 查询 (Elasticsearch 1.3.1)

```
{
  "bool": {
    "must": {
      "term": {"_all": "beijing"}
    },
    "should": {
      "term": {"about": "opera"}
    },
    "must_not": {
      "range": {
        "on_board_date": {"gte": "2013-01-01", "lte": "2014-12-31" }
      }
    }
  }
}
```

26.3.2. `bool` 查询语句 (Elasticsearch 2.0.0)



本节介绍的是TDH4.6及之后的版本 (Elasticsearch版本为2.0.0) 中的 `bool` 查询，它的用法在4.5及之前的TDH版本 (Elasticsearch版本为1.3.1) 中略有不同。如果您使用的 是TDH4.5及以前的版本，请参考[bool 查询语句 \(Elasticsearch 1.3.1\)](#)。

`bool` 查询由逻辑运算符连接的 `bool` 子句构成，每个子句都有 成立类型 (occurrence type)，包括以下几种：

- `must`: 该子句必须成立，并且它的成立会为 `_score` 加分。
- `filter`: 该子句必须成立，但是它的成立不会为 `_score` 加分。
- `should`: 该子句应该成立，并且它的成立会为 `_score` 加分。如果一个 `bool` 查询中没有 `must` 或 `filter` 子句，至少一个 `should` 子句应该有匹配的Document，您也可以通过 `minimum_should_match` 参数来设置至少有几个 `should` 子句应该有匹配的Document。
- `must_not`: 该子句必须不成立。

`bool` 查询的一般形式如下：

语法: **bool** 查询 (Elasticsearch 2.0.0)

```
{
  "bool" : {
    "must" : {
      <QUERY>
    },
    "filter" : {
      <QUERY>
    },
    "should" : {
      <QUERY>
    },
    "must_not" : {
      <QUERY>
    },
    "minimum_should_match" : <n>
  }
}
```

must、**should**、**filter** 和 **must_not** 中有多个查询语句时，用数组表示：

语法: **bool** 查询 (Elasticsearch 2.0.0)

```
{
  "bool" : {
    "must" : [{<QUERY>}, {<QUERY>}, ...],
    "filter" : [{<QUERY>}, {<QUERY>}, ...],
    "should" : [{<QUERY>}, {<QUERY>}, ...],
    "must_not" : [{<QUERY>}, {<QUERY>}, ...],
    "minimum_should_match" : <n>
  }
}
```

对于 **bool** 查询来说，匹配越多越好，所以 **should** 和 **must** 子句为同一个Document贡献的关联分会被相加算进Document的 **_score**。

例 140. **bool** 查询 (Elasticsearch 2.0.0)

```
{  
  "bool": {  
    "must": {  
      "term": {"_all": "beijing"}  
    },  
    "should": {  
      "term": {"about": "opera"}  
    },  
    "must_not": {  
      "range": {  
        "on_board_date": {"gte": "2013-01-01", "lte": "2014-12-  
31"}  
      }  
    },  
    "filter": {  
      "married": true  
    }  
  }  
}
```

27. 常见Transwarp ES API列表

27.1. API使用约定

除非特别指出，本章都将采用如下约定。

27.1.1. 多Index的使用

大多数使用Index参数的API都可以跨Index执行，指定Index时只需依次列出Index并将它们用逗号隔开，例如：

index1, index2, index3

指定Index时也可以使用通配符“*”和“.”，例如：

foo*, fooba.



Document API不支持跨Index执行。

27.1.2. 输出格式

在通过 curl 使用REST API的通用格式中的 <PARAMETERS> 部分加上 pretty 或者 pretty=true，Transwarp ES将以可读性较高的方式输出结果。

27.2. Cluster级别API

本章介绍Transwarp ES的集群（Cluster）级别的API，Cluster级别API包括Cluster API和Nodes API。这些API的使用格式为：

语法：Cluster API使用格式

```
curl -X<VERB> '<HOST>/_cluster/<API>?[<PARAMETERS>]' [-d '{<BODY>}']
```

或者

语法：Nodes API使用格式

```
curl -X<VERB> '<HOST>/_nodes/<API>?[<PARAMETERS>]' [-d '{<BODY>}']
```

27.2.1. Cluster health API

Cluster **health** API用于查看集群和Index的健康状况。Transwarp ES中健康状况分为三种：green, yellow 和red。在Shard级别：

- 如果健康状况为red，说明无法在集群中找到该Shard。
- 如果健康状况为yellow，说明该Shard可以在集群中找到，但是它的Replica找不到。
- 如果健康状况为green，则Shard和它的Replica都能找到。

Index的健康状况由Index下最差的Shard健康状况决定；集群的健康状况由集群下最差的Index健康状况决定。

语法：查看整个集群的**health**

```
curl -XGET '<HOST>/_cluster/health?pretty'
```

输出类似于：

```
{
  "cluster_name" : "test",
  "status" : "green",
  "timed_out" : false,
  "number_of_nodes" : 3,
  "number_of_data_nodes" : 3,
  "active_primary_shards" : 30,
  "active_shards" : 90,
  "relocating_shards" : 0,
  "initializing_shards" : 0,
  "unassigned_shards" : 0
}
```

health 后可以添加Index名字来查看指定一个或多个Index的健康状况。Index之间用“，”隔开：

查看指定Index的健康状况

```
curl -XGET '<HOST>/_cluster/health/<index>,<index>, ...?pretty'
```

例 141. 查看名为employee和inventory的Index的健康状况

```
curl -XGET '<HOST>/_cluster/health/employee,inventory?pretty'
```

Query String选项

Cluster **health** API在 **<PARAMETERS>** 部分可以使用的选项有：

- level**: 值可以为 **cluster** (集群级别)、**indices** (Index级别) 和 **shards** (Shard级别)，控制返回的健康状况的级别。默认值为 **cluster**。例如：

语法：查看集群中所有Index的健康状况

```
curl -XGET '<HOST>/_cluster/health?level=indices&pretty'
```

27.2.2. Cluster stats API

Cluster **stats** API用于获取整个集群的统计数据，包括基础的Index指标（Shard数量、存储大小、内存使用量）和节点指标（节点数量、角色、计算资源使用状况等等）。

语法：查看Cluster Stats

```
curl -XGET '<HOST>/_cluster/stats?pretty'
```

27.2.3. Cluster settings API

Cluster **settings** API可以用于查看（**GET**）或进行（**PUT**）集群的设置。

语法：查看整个集群的Settings

```
curl -XGET '<HOST>/_cluster/settings?pretty'
```

语法：设置批量插入的队列大小

```
curl -XPUT '<HOST>/_cluster/settings' -d '{
  "persistent": {
    "threadpool.bulk.queue_size" : 1000
  }
}'
```

27.2.4. Nodes stats API

Nodes **stats** API用于查看节点的统计数据。

语法：查看所有节点的统计数据

```
curl -XGET '<HOST>/_nodes/stats?pretty'
```

还可以通过nodeID来查看指定节点的统计数据：

语法：查看指定节点的统计数据

```
curl -XGET '<HOST>/_nodes/<nodeID>,<nodeID>,.../stats?pretty'
```

例 142. 查看tw-node127和tw-node128节点的统计数据

```
curl -XGET 'localhost:9200/_nodes/tw-node127,tw-node128/stats?pretty'
```

27.3. Index级别API

Index级别API用于管理一个或多个Index。Index级别的API使用格式为：

语法：Index级别API使用格式

```
curl -X<VERB> '<HOST>/<index>[ , <index>] /[_<API>]?[<PARAMETERS>]' [-d '{<BODY>}' ]
```

其中，`<index>[, <index>]` 用于指定一个或多个Index，Index之间用“，”隔开。

27.3.1. 创建Index

创建Index执行 **PUT**。

语法：创建Index

```
curl -XPUT '<HOST>/<index>/?pretty' [-d '{<BODY>}' ]
```

在BODY中可以进行Index Settings的设置、显式定义Mapping等。

例 143. 创建名为test的Index

```
curl -XPUT 'localhost:9200/test/?pretty'
```

输出：

```
{
  "acknowledged" : true
}
```

27.3.2. 删除Index

删除Index执行 **DELETE**。

语法：删除指定Index

```
curl -XDELETE '<HOST>/<index>, <index>, ... /?pretty'
```

例 144. 删除名为test的Index

```
curl -XDELETE 'localhost:9200/test/?pretty'
```

27.3.3. 查看Index是否存在

查看Index是否存在使用 **HEAD**:

语法：查看指定Index是否存在

```
curl -i -XHEAD '<HOST>/<index>'
```

27.3.4. Index **settings** API

通过Index **settings** API可以查看（**GET**）或进行（**PUT**）一个或多个Index的设置。

语法：查看指定Index的设置

```
curl -XGET '<HOST>/<index>, <index>, .../_settings?pretty'
```

例 145. 查看名为employee的Index的设置

```
curl -XGET 'localhost:9200/employee/_settings?pretty'
```

语法：设置副本数

```
curl -XPUT '<HOST>/<index>, <index>, .../_settings?pretty' -d '{
  "number_of_replicas": <integer>
}'
```

语法：设置更新的interval

```
curl -XPUT '<HOST>/<index>, <index>, .../_settings?pretty' -d '{
  "index" : {
    "refresh_interval" : <integer>
  }
}'
```

27.3.5. Index **optimize** API

通过 **optimize** API可以优化一个或多个指定的Index。对Index优化可以加快检索操作。**optimize** API通过合并segments来减少segment数量。

语法：合并segments

```
curl -XPOST '<HOST>/<index>, <index>, .../_optimize'
```

接受的参数

optimize API接收的请求参数包括：

- **max_num_segments**: 指定最大segment数量，这将是优化的目标segment数量。

例 146. 将名为test的Index的 **max_num_segments** 设为3

```
curl -XPOST
'localhost:9200/test/_optimize?max_num_segments=3&pretty'
```

27.3.6. Index mapping API

通过 **mapping** API可以进行Index Mapping的获取（**GET**）和设置（**PUT**）。

语法：获取Index Mapping

```
curl -XGET '<HOST>/<index>, <index>, .../_mapping?pretty'
```

语法：为Index下指定Type设置Mapping

```
curl -XPUT '<HOST>/<index>/_mapping/<type>' -d '{<BODY>}'
```



Mapping必须在Type创建时就指定。如果不指定Mapping，Transwarp ES会自动根据Type下Document中的数据来判断并设置好Mapping，这个过程称为动态Mapping（Dynamic Mapping）。Mapping一旦设置就不能再更改。

27.4. Document API

通过Document API我们可以对Transwarp ES Document进行读写操作。每个Transwarp ES Document由唯一的<index>/<type>/<id>组合确定。所以单个Document API的使用格式为：

单个Document API的使用格式

```
curl -X<VERB> '<HOST>/<index>/<type>/<id>[_<API>][?<PARAMETERS>]' [-d
'{<BODY>}']
```

27.4.1. 编入API

编入Document使用 **PUT**（用户指定Document ID）或者 **POST**（Transwarp ES自动生成ID）。

语法：编入Document，用户指定Document ID

```
curl -XPUT '<HOST>/<index>/<type>/<id>' -d '{<document_body>}'
```

注意，该语法也可以用于更新整个Document。举例：见[编入员工Zhang San的信息和更新/employee/dev/2下的Document](#)。

语法：编入Document，Transwarp ES自动生成ID

```
curl -XPOST '<HOST>/<index>/<type>' -d '{<document_body>}'
```

举例：见[向/employee/sales下编入一个新Document](#)。

27.4.2. 获取API

获取Document使用 **GET**。

语法：获取整个Document

```
curl -XGET '<HOST>/<index>/<type>/<id>?pretty'
```

举例：见[获取/employee/dev/1下的Document](#)。

语法：获取Document中的某个字段

```
curl -XGET '<HOST>/<index>/<type>/<id>?_source=<field>,<field>,...&pretty'
```

举例：见[获取/employee/dev/1中的 name 和 age 字段](#)。

27.4.3. 删除API

删除Document使用 **DELETE**。

语法：删除一个Document

```
curl -XDELETE '<HOST>/<index>/<type>/<id>'
```

举例：见[删除/employee/dev/1下的Document](#)。

28. Transwarp ESDrive SQL使用说明

4.6版本以前，Transwarp Data Hub (TDH) 上还不能直接将SQL跑在Transwarp ES上，用户只能使用Transwarp ES的API或HBase全文检索来使用Transwarp ES。从TDH4.6开始，Hyperbase中引入了全新的ESDrive使得用户可以通过SQL的方式使用Transwarp ES，大大降低了全文检索相关的使用门槛。ESDrive SQL可以完全兼容所有的SQL语法，包括PLSQL，同时也有自己特定的分词检索等特殊语法。相比较API使用Transwarp ES和HBase全文检索的方式，采用ESDrive有如下优势：

- 极高的易用性：

以往的Transwarp ES使用有较高的入门门槛，必须对Transwarp ES提供的各种REST API、query的写法、甚至是Apache Lucene底层技术比较熟悉的情况下，才能写出高效的查询条件，使用成本比较高，而使用ESDrive SQL，用户只要有编写SQL的经验，就可以简单的使用Transwarp ES。

- 性能提升：

大部分情况下，使用ESDrive可以获得比使用API有更高的性能提升，这是因为ESDrive内部做了一些特殊优化。比如，多条件查询的时候，不同条件的先后顺序会对查询有极大的影响，ESDrive内部对查询有一套自己的优化接口。

- 迁移成本低：

用户的业务逻辑从Oracle或DB2上迁移到ESDrive上时，迁移成本非常低，如果直接用API的方式迁移到Transwarp ES上，代价会非常高。有了ESDrive之后，用户只需要修改少量的SQL即可。

本章介绍如何使用SQL与Transwarp ES交互。我们将重点介绍Transwarp ESDrive SQL中的DDL以及DML中的`INSERT/UPDATE/DELETE`。在DML的`SELECT`语句方面，Transwarp ESDrive SQL独有的`CONTAINS`和`MATCH`函数可以利用Transwarp ES强大的全文检索能力。除此之外，Transwarp ESDrive SQL的`SELECT`语句和Inceptor SQL的`SELECT`语句用法完全相同，如`WHERE`、`GROUP BY`、`JOIN`、子查询和集合运算等，这里将不赘述，请参考《Transwarp Data Hub Inceptor 使用手册》。

28.1. Transwarp ESDrive SQL 快速入门

我们先回忆一下Transwarp ES中的数据对象如何映射为Transwarp ESDrive二维表（更多关于Transwarp ES数据模型的内容请参考[Transwarp ES数据模型](#)。）：

使用Transwarp ESDrive SQL和Transwarp ES交互时，Transwarp ESDrive会将Index映射成一张二维表，Index中的Document映射成表中的行，Index中的Field映射为表中的列：

Elasticsearch	Transwarp ESDrive
Index	表(Table)
Document	行(Row)
Field	列(Column)

在详细介绍Transwarp ESDrive之前，我们先演示一个简单的Transwarp ESDrive例子。在该例子中，我们生成了

一张内表(es_start)和一张外表(es_start_ex)，而后以外表es_start_ex为例，对表进行了添加、查询和删除数据等操作。

1. 建表：CREATE

例 147. 创建一张Transwarp ES内表

```
CREATE TABLE es_start(
    key STRING,
    content STRING,
    tint INT,
    tfloat FLOAT,
    tbool BOOLEAN)
STORED AS ES;
```

例 148. 创建一张Transwarp ES外表

Transwarp ES表列名和源表列名相同：

```
CREATE EXTERNAL TABLE es_start_ex(
    key STRING,
    content STRING,
    tint INT,
    tfloat FLOAT,
    tbool BOOLEAN)
STORED AS ES
TBLPROPERTIES('elasticsearch.tablename'='default.es_start');
```

2. 插入数据：INSERT

例 149. 向外表es_start_ex插入数据：

```
INSERT INTO TABLE es_start_ex(key, content, tint, tfloat, tbool)
VALUES ("1", "oracle", 1, 1.1, true);
INSERT INTO TABLE es_start_ex(key, content, tint, tfloat, tbool)
VALUES ("2", "oracle is database", 2, 2.2, false);
INSERT INTO TABLE es_start_ex(key, content, tint, tfloat, tbool)
VALUES ("3", "db2 is database", 3, 3.3, true);
INSERT INTO TABLE es_start_ex(key, content, tint, tfloat, tbool)
VALUES ("4", "oracle and mysql are databases", 4, 4.4, false);
```

3. 查询数据：SELECT

例 150. 选择Transwarp ES表中key为1的行:

```
select * from es_start_ex WHERE key = '1';
1 oracle 1 1.1 true
```

4. 删除表: **DROP**

例 151. 删除Transwarp ES表

```
DROP TABLE es_start_ex;
DROP TABLE es_start;
```

28.2. Transwarp ESDrive SQL DDL

Transwarp ESDrive SQL中的DDL包括创建(**CREATE**) /编辑(**ALTER**) /删除(**DROP**) /清空(**TRUNCATE**)表。

28.2.1. 建表: **CREATE**

简化建Transwarp ES表语法

```
CREATE [EXTERNAL] TABLE <table> (
    <id> STRING, ①
    <column> <data_type>,
    <column> <data_type>,
    ...
)
STORED AS ES ②
[WITH SHARD NUMBER <m>] ③
[REPLICATION <n>] ④
[TBLPROPERTIES('elasticsearch.tablename'='<esdrive_table>')]; ⑤
```

① Transwarp ES表的第一列为Transwarp ES表的id，必须是STRING类型的。

② 指定表的格式为 **Transwarp ES**。

③ 可选项，指定Transwarp ES表的分片数m。如果不指定，使用默认值5。一旦建表即不能更改。这里需要用户预估索引数据的量，一个SHARD上的数据不要超过25GB。超过25GB可能会有比较严重的性能问题。

④ 可选项，指定每个分片的副本数n，如果不指定，使用默认值1。建表后还可以使用 **ALTER** 来更改。

⑤ 可选项，指定新建的表在ElasticSearch中的索引。建外表时必须指定新建的表在ElasticSearch中对应的表名。创建内表时可以指定，也可以忽略，当忽略时，默认的ElasticSearch表名是elasticsearch_<dbName.tableName>。

Transwarp ESdrive支持数据类型

Transwarp ESdrive支持数据类型及其映射到Transwarp ES中的实际类型:

Transwarp ESdrive支持数据类型	Transwarp ES中的实际类型
string	string
tinyint	byte
smallint	short
int	integer
bigint	long
float	float
double	double
boolean	boolean

普通建Transwarp ES表语法

```
CREATE [ EXTERNAL ] TABLE <table> (
    <id> STRING,
    <column> <data_type>,
    <column> <data_type>,
    ...
)
STORED BY 'io.transwarp.esdrive.ElasticSearchStorageHandler' ①
[WITH SERDEPROPERTIES('elasticsearch.columns.mapping'='_id,<c1>,<c2>,
...')] ②
[WITH SHARD NUMBER <m>]
[REPLICATION <n>]
[TBLPROPERTIES('elasticsearch.tablename'='<esdrive_table>')];
```

① 指定使用 '`io.transwarp.esdrive.ElasticSearchStorageHandler`' 作为storage handler。

② 可选项，指定新建的表和ElasticSearch中的对应索引的字段的映射关系。如果使用这个选项指定列的映射关系，等号右边的 `_id` 为固定用法，不能改为其他名字。`elasticsearch.columns.mapping` 的值中，`_id` 字段与inceptor表中 `key` 有映射关系，表示Transwarp ES索引表中 `key` 的字段名，`content` 也是Transwarp ES索引表中字段名与inceptor表中 `content` 字段相互映射，根据顺序依次形成所有字段映射，映射字段之间的类型必须相同

建表语法的选择

显然，[简化建Transwarp ES表语法](#)步骤简单，易于记忆，所以我们 鼓励用户在建内表时使用[简化建Transwarp ES表语法](#)。如果新建的表和ElasticSearch中已经有的表重名，需要使用 `TBLPROPERTIES ('elasticsearch.tablename'='<esdrive_table>')` 来指定新建表在ElasticSearch中的索引。建外表时，Transwarp ES表必须映射到ElasticSearch中一张已经存在的源表。如果Transwarp ES表和它的源表在ElasticSearch中的对应列列名相同，可以使用[简化建Transwarp ES表语法](#)建表；如果Transwarp ES表和它的源表在ElasticSearch中的对应列列名不同，需要使用[普通建Transwarp ES表语法](#)来建表，通过建表语句中的 `WITH SERDEPROPERTIES ('elasticsearch.columns.mapping'='_id,<c1>,<c2>, ...')` 来指定Transwarp ES表和ElasticSearch中源表之间的列名映射。注意，ElasticSearch中索引的字段名 区分大小写。

例 152. 建Transwarp ES内表

```
CREATE TABLE employee (
    eid STRING,
    name STRING,
    age TINYINT,
    height DOUBLE,
    about STRING
)
STORED AS ES;
```

例 153. 建Transwarp ES外表，Transwarp ES表列名和源表列名相同

```
CREATE EXTERNAL TABLE esdrive_test(
    key STRING,
    content STRING,
    tint INT,
    tfloat FLOAT,
    tbool BOOLEAN)
STORED AS ES
TBLPROPERTIES('elasticsearch.tablename'='esdrive_test');
```

例 154. 建Transwarp ES外表，Transwarp ES表列名和源表列名不同

```
CREATE EXTERNAL TABLE esdrive_test3(
    key3 STRING,
    content3 STRING,
    tint3 INT,
    tfloat3 FLOAT,
    tbool3 BOOLEAN)
    STORED BY 'io.transwarp.esdrive.ElasticSearchStorageHandler'
    WITH SERDEPROPERTIES('elasticsearch.columns.mapping'='_id, content,
    tint, tfloat, tbool')
    TBLPROPERTIES('elasticsearch.tablename'='esdrive_test');
```

建表后可以用 **DESCRIBE** 查看表的元数据：

```
DESCRIBE FORMATTED employee;
```

28.2.2. 修改表：ALTER

使用 **ALTER** 可以修改Transwarp ES表的TBLPROPERTIES。目前只能用于修改ElasticSearch表settings中可动态改变的配置项，例如replication。关于ElasticSearch表settings的细节请参考《ElasticSearch使用手册》。

修改Transwarp ES表TBLPROPERTIES的语法

```
ALTER TABLE <table> SET TBLPROPERTIES ('<property_name>' = '<property_value>', '<property_name>' = '<property_value>', ...);
```

其中，**property_name** 为属性名，必须 使用ElasticSearch表settings中配置项的名字。**property_value** 为属性值，它们都需要放在引号中间。该语法可以用于修改Transwarp ES表的副本数。

修改Transwarp ES表副本数的语法

```
ALTER TABLE <table> SET TBLPROPERTIES ('number_of_replicas'='<n>');
```

例 155. 修改Transwarp ES表副本数

```
ALTER TABLE employee SET TBLPROPERTIES ('number_of_replicas'='1');
```

TBLPROPERTIES 可以使用 **DESCRIBE FORMATTED** 来查看：

```
DESCRIBE FORMATTED employee;
```

还可用 **ALTER** 来为Transwarp ES表增加列：

为Transwarp ES表增加列的语法

```
ALTER TABLE <table> ADD COLUMNS (<column> <data_type>, <column> <data_type>, ...);
```

例 156. 为Transwarp ES表增加列

```
ALTER TABLE employee ADD COLUMNS (col1 string);
ALTER TABLE employee ADD COLUMNS (col2 int , col3 float);
```



目前在Transwarp ESdrive中，对列的 **ALTER** 操作只支持 **ADD COLUMNS**，用法同Hyperdrive表，可参考[添加列的语法](#)。

28.2.3. 删除表: **DROP**

删除Transwarp ES表的语法

```
DROP TABLE <table>;
```

28.2.4. 清空表: **TRUNCATE**

清空Transwarp ES表的语法

```
TRUNCATE TABLE <table>;
```

注意，不能 **TRUNCATE** 外表。

28.3. Transwarp ESdrive SQL DML

Transwarp ESdrive SQL中的DML（Data Manipulation Language）包含插入数据（**INSERT**），和删除表中记录（**DELETE**）。Transwarp ESdrive SQL目前不支持 **UPDATE**。

28.3.1. 插入: **INSERT**

Transwarp ESdrive SQL支持向Transwarp ES表中单条插入数据或者批量插入查询结果。

单条插入的语法

```
INSERT INTO TABLE <table> [(<column1>, <column2>, ...)] VALUES (<value1>, <value2>, ...);
```

例 157. 单条插入Transwarp ES表

向建Transwarp ES外表，Transwarp ES表列名和源表列名相同中创建的表esdrive_test插入数据：

```
INSERT INTO TABLE esdrive_test(key, content, tint, tfloat, tbool)
VALUES ("1", "mysql is database", 1, 1.1, true);
INSERT INTO TABLE esdrive_test(key, content, tint, tfloat, tbool)
VALUES ("2", "oracle is database", 2, 2.2, false);
INSERT INTO TABLE esdrive_test(key, content, tint, tfloat, tbool)
VALUES ("3", "db2 is database", 3, 3.3, true);
INSERT INTO TABLE esdrive_test(key, content, tint, tfloat, tbool)
VALUES ("4", "oracle and mysql are databases", 4, 4.4, false);
INSERT INTO TABLE esdrive_test(key, content, tint, tfloat, tbool)
VALUES ("5", "contains test !!!", 5, 5.5, false);
INSERT INTO TABLE esdrive_test(key, content, tint, tfloat, tbool)
VALUES ("6", "test", 6, 6.6, true);
INSERT INTO TABLE esdrive_test(key, content, tint, tfloat, tbool)
VALUES ("7", "first second third", 7, 7.7, false);
```

例 158. 向Transwarp ES表批量插入查询记录

向建Transwarp ES外表中创建的表employee插入数据：

```
INSERT INTO TABLE employee SELECT * FROM employee_2;
```

28.3.2. 删除表中记录：DELETE

删除Transwarp ES表中记录的语法

```
DELETE FROM <table> WHERE <filter_conditions>;
```

例 159. 删除Transwarp ES表中记录

```
DELETE FROM employee WHERE name = 'Alice';
```

28.3.3. 查询：SELECT

Transwarp ESdrive SQL支持所有Hyperdrive SQL中的DQL（除 **USE_INDEX** 以外），包括 **WHERE**、**GROUP BY**、**JOIN**、子查询和集合运算等，具体细节请参考《Hyperdrive SQL使用手册》。本章我们将详细介绍Transwarp ESdrive SQL中特有的DQL：使用 **CONTAINS** 和 **MATCHES** 这两个UDF做查询。

Transwarp ESDrive SQL独有的 **CONTAINS** 和 **MATCHES** 函数可以在 **SELECT** 语句中利用Transwarp ES强大的全文检索能力。除此之外，Transwarp ESDrive SQL的 **SELECT** 语句和Inceptor SQL的 **SELECT** 语句用法完全相同，如 **WHERE**、**GROUP BY**、**JOIN**、子查询和集合运算等，这里将不赘述，请参考《Transwarp Data

Hub Inceptor 使用手册》。

Transwarp ES中的分词

下面即将介绍的 **CONTAINS** 和 **MATCHES** 函数的必须Transwarp ES表的 **分词列** 使用。目前，Transwarp ESdrive SQL还没有分词语法，对表的分词必须通过Transwarp ES Query DSL在建表时设置，例如：

建esdrive_test表的Transwarp ES Query DSL Query Body

```
{
  "settings": {
    "refresh_interval": "5s",
    "number_of_shards": 4,
    "number_of_replicas": 1
  },
  "mappings": {
    "default_type_": {
      "_all": { "enabled": true }
    },
    "resource": {
      "dynamic": false,
      "properties": {
        "key": {
          "type": "string",
          "index": "analyzed" ①
        },
        "content": {
          "type": "string",
          "analyzer": "english" ②
        },
        "tint": {
          "type": "integer"
        },
        "tfloat": {
          "type": "float"
        },
        "tbool": {
          "type": "boolean"
        }
      }
    }
  }
}
```

① 表示对该列分词，使用Transwarp ES默认的分词插件standard。

② 显式指定对该列用分词插件english分词。Transwarp ES对不同语言提供不同的分词插件，分词插件的安装和配置请参考《Transwarp ES使用手册》。

在Transwarp ES中建表完成以后，用户可以通过Transwarp ESdrive SQL建外表映射到它，然后进行查询。目前在Transwarp ESdrive SQL中还不支持对内表使用 **CONTAINS** 和 **MATCHES** 函数（不会报错，但是不会得到预期结果）。

其他Transwarp ES Query DSL的使用请参考[Transwarp ES检索](#)。

28.3.3.1. CONTAINS

CONTAINS 运算符简介

CONTAINS 操作符号用在SELECT查询的WHERE子句后边，用于指定文本查询的表达式。

Transwarp EDrive **CONTAINS** 语法

CONTAINS 函数使用语法

```
CONTAINS(
    [schema.]column, ①
    (text_query ②
        [VARCHAR|STRING]) ③
    RETURN STRING;
```

① [schema.]column : 索引字段名

② text_query : 检索表达式

③ RETURN : 返回符合条件的文本

ORACLE CONTAINS语法:

```
CONTAINS(
    [schema.]column, ①
    text_query ②
    [VARCHAR2|CLOB]
    [,label NUMBER]) ③
    RETURN NUMBER; ④
```

① [schema.]column : 索引字段名。

② text_query : 检索表达式。

③ label : 可选参数，标识检索符号条件行的SCORE值，同SCORE运算符(返回CONTAINS的SCORE值)一起用。

④ RETURN : 返回符合条件行的SCORE值。

例 160. ORACLE CONTAINS示例

```
SELECT SCORE(1)
      ,col_name
  FROM table_name
 WHERE CONTAINS(col_name, 'oracle', 1) > 0;
```

检索包含单词 **oracle** 且SCORE大于0的行，利用SCORE操作符号显示当期行的SCORE值



ORACLE CONTAINS查询返回的是检索结果的 **SCORE值**，但是Transwarp ESdrive中查询返回之检索到的结果。

CONTAINS 中支持的运算符

表 11. **CONTAINS** 中支持的运算符

Transwarp ESdrive中的表示形式	ORACLE中的表示形式	描述
AND, &	AND, &	逻辑操作符“与”
OR,	OR,	逻辑操作符“或”
()	()	括号，用于提高表达式的优先级
NEAR	NEAR	间隔词匹配函数 near (具体见下)

表 12. **CONTAINS** 中支持的模糊查询符号

Transwarp ESdrive中的表示形式	ORACLE中的表示形式	描述
%	%	匹配0个或多个
-	-	匹配1个

CONTAINS 操作符使用说明

例 161. 在 **CONTAINS** 中使用“与”

```
SELECT col_name
FROM table_name
WHERE CONTAINS(col_name, 'oracle and mysql and db2');

SELECT col_name
FROM table_name
WHERE CONTAINS(col_name, 'oracle & mysql & db2');
```

例 162. 在 **CONTAINS** 中使用“或”

```
SELECT col_name
FROM table_name
WHERE CONTAINS(col_name, 'oracle or mysql or db2');

SELECT col_name
FROM table_name
WHERE CONTAINS(col_name, 'oracle | mysql | db2');
```

例 163. 在 CONTAINS 中使用多个运算符

```
SELECT col_name
FROM table_name
WHERE CONTAINS(col_name, 'QQ OR (UU AND (AAA OR TEST)) OR XX');

SELECT col_name
FROM table_name
WHERE CONTAINS(col_name, '((term | test) & sep)');
```

例 164. 在 CONTAINS 中使用 near

```
SELECT col_name
FROM table_name
WHERE CONTAINS(col_name, '(UU AND NEAR((XXX, tt, nsd), 10, TRUE) AND
TEST) OR XX');

SELECT col_name
FROM table_name
WHERE CONTAINS(col_name, 'NEAR((XXX, tt, nsd), 10) AND TEST');
```

使用Transwarp ES接口 **SpanNearQuery** 实现精确查询。比如某个term之后是另一个term，term之间的距离也可以自己设定，从而来实现精确搜索：

```
{
  "span_near" : {
    "clauses" : [ ①
      { "span_term" : { "field" : "value1" } },
      { "span_term" : { "field" : "value2" } },
      { "span_term" : { "field" : "value3" } }
    ],
    "slop" : 12, ②
    "in_order" : false, ③
    "collect_payloads" : false
  }
}
```

① 匹配项，可以有多个

② 间隔数

③ 表示是否排序

NEAR操作符

NEAR 操作符在Transwarp ESDrive下层执行的是Transwarp ES的 **span_near**，语意上同Oracle相似，但参数使用上稍微有所不同。

它们默认的 **in_order** 参数都是 **false**，但是对于 **slop** 参数，在Oracle中表示的间隔不包括查询项本身，该间隔可以表示一个单词的间隔也可以表示一个字符的间隔。例如 **NEAR((DOG, CAT), 1)**，**dog cat** 和 **dog ate cat** 能够被匹配，而 **dog sat on cat** 不会被匹配。但在Transwarp ESDrive中NEAR **slop** 参数的大小表示允许两个查询项之间允许的最大间隔数量。

例 165. 模糊查询符 %

```
SELECT col_name
FROM table_name
WHERE CONTAINS(col_name, '%oracle%');

SELECT col_name
FROM table_name
WHERE CONTAINS(col_name, '0%');
```

例 166. 模糊查询符 _

```
SELECT col_name
FROM table_name
WHERE CONTAINS(col_name, '_ing');
```



模糊查询目前只有在使用方法上和Oracle一致，但是Transwarp ESDrive底层的模糊查询只是简单的模糊查询，对于后缀查询有着较低的性能。在Oracle中的模糊查询可以有可以自适应的选择较优的查询方式。

CONTAINS 操作符使用说明

1. 创建ES索引表

在终端使用 **curl** 发送创建Transwarp ES索引请求到 **9200** 端口：

例 167. 创建Transwarp ES索引

```
curl -XPOST "http://shiva01:9200/esdrive_test" -d '
{
    "settings": { ①
        "refresh_interval": "5s",
        "number_of_shards": 4,
        "number_of_replicas": 1
    },
    "mappings": {
        "default_type_": { ②
            "_all": {
                "enabled": true
            },
            "properties": {
                "_id": { ③
                    "type": "string",
                    "index": "not_analyzed"
                },
                "content": {
                    "type": "string",
                    "analyzer": "english" ④ ⑤
                },
                "tint": {
                    "type": "integer"
                },
                "tfloat": {
                    "type": "float"
                },
                "tbool": {
                    "type": "boolean"
                }
            }
        }
    }
}'
```

① **settings** 表示当前表的配置信息，可以根据需要配置。可根据实际需求查询Transwarp ES使用文档

② **default_type_** 表示索引 **type** 的名字，在这里我们只能有一个 **type**。我们将用这个 **type** 中的 **properties** 字段与inceptor表中的字段生成映射关系。

③ **_id** 字段与inceptor表 **key** 字段形成映射。在建表时使用 **elasticsearch.columns.mapping** 指定 **_id** 和inceptor表中 **key** 字段的映射。

④ **content** 字段使用分词插件 **english**，所有存储在这个字段的英文将被此分词器作用。

⑤ 分词插件推荐：中文分词插件 **ik**，英文分词插件 **english**。

2. 创建Transwarp ESdrive外表

例 168. 创建Transwarp ESdrive外表

```

CREATE EXTERNAL TABLE esdrive_test(
    key string,
    content string,
    tint int,
    tfloat float,
    tbool boolean
)
STORED BY 'io.transwarp.esdrive.ElasticSearchStorageHandler'
WITH SERDEPROPERTIES(
    'elasticsearch.columns.mapping'='_id,content,tint,tfloat,tbool',
    'elasticsearch.indextype'='default_type_')
TBLPROPERTIES('elasticsearch.tablename'='esdrive_test');

```

Transwarp ESdrive创建外表注意事项

1. 如果我们需要指定 `elasticsearch.columns.mapping` 属性时，不能使用 `stored as es`，必须使用 `stored by 'io.transwarp.esdrive.ElasticSearchStorageHandler'`
 2. `elasticsearch.columns.mapping` 的值中，`_id` 字段与inceptor表中 `key` 有映射关系，表示Transwarp ES索引表中 `key` 的字段名，`content` 也是Transwarp ES索引表中字段名与inceptor表中 `content` 字段相互映射，根据顺序依次形成所有字段映射，映射字段之间的类型必须相同
 3. `elasticsearch.indextype` 的值为Transwarp ES索引表 `type` 的名字，在这里是上一步骤中创建的 `type: default_type_`
 4. `elasticsearch.tablename` 的值为Transwarp ES索引名字，在这里是步骤一中创建的Transwarp ES索引名字
3. 插入测试数据

例 169. 插入测试数据

```
INSERT INTO TABLE esdrive_test(key, content,tint, tfloat, tbool)
VALUES("1", "mysql is database", 1 ,1.1, true);
INSERT INTO TABLE esdrive_test(key, content,tint, tfloat, tbool)
VALUES("2", "oracle is database",2,2.2, false);
INSERT INTO TABLE esdrive_test(key, content,tint, tfloat, tbool)
VALUES("3", "db2 is database",3,3.3, true);
INSERT INTO TABLE esdrive_test(key, content,tint, tfloat, tbool)
VALUES("4", "oracle and mysql are databases",4,4.4, false);
INSERT INTO TABLE esdrive_test(key, content,tint, tfloat, tbool)
VALUES("5", "contains test !!!", 5,5.5, false);
INSERT INTO TABLE esdrive_test(key, content,tint, tfloat, tbool)
VALUES("6", "test", 6,6.6, true);
INSERT INTO TABLE esdrive_test(key, content,tint, tfloat, tbool)
VALUES("7", "The following are settings that can be set for a
standard analyzer type",7,7.7, false);
```

4. CONTAINS 查询示例

例 170. 一般 CONTAINS 查询

```
SELECT *
FROM esdrive_test
WHERE CONTAINS(content, 'oracle');

2 oracle is database 2 2.2 false
4 oracle and mysql are databases 4 4.4 false
```

例 171. 含有“或”的 CONTAINS 查询

```
SELECT *
FROM esdrive_test
WHERE CONTAINS(content, 'oracle | mysql | db2');
```

例 172. 组合 CONTAINS 查询

```
SELECT *
FROM esdrive_test
WHERE CONTAINS(content, '(oracle & mysql) | db2');

4 oracle and mysql are databases 4 4.4 false
3 db2 is database 3 3.3 true
```

例 173. 查询包含 follow 的子句

```
SELECT *
FROM esdrive_test
WHERE CONTAINS(content, 'follow');

7 The following are settings that can be set for a standard
analyzer type 7 7.7 false
```



需要注意的是，上例的查询语句中查询了包含 `follow` 关键词的子句，然而我们的短语中只有 `following` 这个单词。之所以能得到对应的结果是因为 `english` 分词插件的作用，它把 `following` 分词时转换成了词根 `follow`。

例 174. 查询包含 is 的子句

```
SELECT *
FROM esdrive_test
WHERE CONTAINS(content, 'is');
```

为什么查询不到包含 is 的结果？

我们先看看 `english` 分词器做了什么，以 `mysql is database` 为例：

```
curl "shiva01:9200/_analyze?analyzer=english&pretty" -d 'mysql is
database'
{
  "tokens" : [
    {
      "token" : "mysql",
      "start_offset" : 0,
      "end_offset" : 5,
      "type" : "<ALPHANUM>",
      "position" : 0
    },
    {
      "token" : "databas",
      "start_offset" : 9,
      "end_offset" : 17,
      "type" : "<ALPHANUM>",
      "position" : 2
    }
  ]
}
```

我们可以看到 `english` 分词器把 `mysql is database` 分成了两个token：`mysql` 和 `databas`，把 `is` 过滤掉了，所以就查不到结果了。

28.3.3.2. MATCHES

MATCHES 函数使用语法

```
MATCHES([<table>.]<column>, '<text_query>')
```

MATCHES 进行 不区分大小写, 匹配固定位置的字符和空格 的精确匹配, 它由Transwarp ES中的 `match_phrase` 查询实现。 **MATCHES** 在 `<column>` 中匹配 `<text_query>`, 如果匹配成功, 返回TRUE, 不成功则返回FALSE。第二个参数 `<text_query>` 必须是字符类型, 并且要放在引号之间。

MATCHES 按照token出现的位置 (position) 顺序匹配, 比如查询短语 `quick brown fox` , 只有符合以下查询条件的词条才能被查到:

- `quick` , `brown` 和 `fox` 必须全部出现;
- `brown` 的位置必须比 `quick` 的位置 (position) 大1;
- `fox` 的位置必须比 `quick` 的位置 (position) 大2。

任何一个条件不符合, 词条都不算匹配。

token的位置可以通过如下命令查看:

例 175. 查看token位置

```
curl "shiva01:9200/_analyze?analyzer=standard&pretty" -d 'Quick brown  
fox' ①  
  
{  
  "tokens" : [ {  
    "token" : "quick",  
    "start_offset" : 0,  
    "end_offset" : 5,  
    "type" : "<ALPHANUM>",  
    "position" : 0 ②  
  }, {  
    "token" : "brown",  
    "start_offset" : 6,  
    "end_offset" : 11,  
    "type" : "<ALPHANUM>",  
    "position" : 1  
  }, {  
    "token" : "fox",  
    "start_offset" : 12,  
    "end_offset" : 15,  
    "type" : "<ALPHANUM>",  
    "position" : 2  
  } ]  
}
```

① shiva01:9200 可替换为对应集群的端口； analyzer=standard 表示指定分词器为standard，也可替换为其他分词器。

② position 表示当前token的位置。

使用 slop 参数放宽 MATCHES 匹配条件

通过 `quick brown fox` 的例子，我们看到 `MATCHES` 有非常严格的匹配条件——一个词条要能被匹配上，它不仅要包含所有 `<text_query>` 中的词，这些词的顺序和position之差还要和 `<text_query>` 一致。有时，我们不需要匹配条件如此严格——在一定的范围内，我们想让有顺序或者位置偏差的词条都算作匹配。这时，我们可以通过设置 `slop` 参数来放宽匹配条件，设置方法如下：

```
SET esdrive.matches.slop.size = <n>; ①
```

① n可为任意正整数。该参数的默认值为0。

`slop` 设置了词条和 `<text_query>` 之间的 词距上限。词距 的定义为：为了使 `<text_query>` 和词条能 严格匹配（词序和position之差一致），需要将 `<text_query>` 中的词移动的 步数。设置 `slop` 为n以后，和 `<text_query>` 词距不超过n的词条将被 `MATCHES` 返回。

为了理解 词距 的具体含义，看下面这个例子：要使得 `fox quick` 能够和文本 `quick brown fox` 严 格匹配（词序和position之差一致），需要将 `fox quick` 中的词移动3步。所以 `fox quick` 和 `quick brown fox` 之间的词距为3。

	Pos 1	Pos 2	Pos 3
Doc:	quick	brown	fox
Query:	fox	quick	
第1步:	fox quick <- ①		
第2步:	quick	-> fox	
第3步:	quick		-> fox

① 此时 `fox` 和 `quick` 都在position 1。

例 176. 在查询中使用 MATCHES

```
SELECT * FROM esdrive_test WHERE MATCHES(content, 'mysql are
databases');
4 oracle and mysql are databases 4 4.4 false
```

因为 **MATCHES** 不区分大小写，所以下面查询返回和上面相同的结果：

```
SELECT * FROM esdrive_test WHERE MATCHES(content, 'mysql ARe
DaTabases');
4 oracle and mysql are databases 4 4.4 false
```

因为 **MATCHES** 匹配固定位置的字符和空格，词序会影响匹配结果。例如下面的查询没有输出：

```
SELECT * FROM esdrive_test WHERE MATCHES(content, 'databases are
mysql');
```

28.4. Transwarp ESdrive 实战

前面章节介绍了Transwarp ESdrive的相关操作，现在我们展示在实际场景中Transwarp ESdrive的应用。

28.4.1. 创建内表和外表

Fiona是售后服务的技术人员，她最近接到的任务是：收集外国客户的反馈邮件，并整理统计出相关问题。Fiona首先要做的是创建一张内表，表中数据包括有客户ID编号以及邮件内容。由于邮件语言为英文，她指定分词器为 **english**：

例 177. 创建源表

```
curl -XPOST "http://localhost:9200/service" -d '
{
    "settings": {
        "refresh_interval": "5s",
        "number_of_shards": 4, ①
        "number_of_replicas": 1 ②
    },
    "mappings": {
        "default_type_": {
            "_all": {
                "enabled": true ③
            },
            "properties": {
                "_id": {
                    "type": "string",
                    "index": "not_analyzed" ④
                },
                "content": {
                    "type": "string",
                    "analyzer": "english" ⑤
                }
            }
        }
    }
}'
```

- ① shard数为 4；
- ② 默认备份数和推荐备份数均为 1；
- ③ 对于默认字段只可指定是否 `enabled`, 详见[Mapping操作](#)；
- ④ 对字段 `id` 不分词；
- ⑤ 选用 `english` 分词器。

在目前的Transwarp ESDrive中，当需要对表中数据进行有关分词的检索时，需要指定一张外表来指向上面创建的内表 `service`。于是Fiona又创建了外表 `service_ex`：

例 178. 创建外表

```
CREATE EXTERNAL TABLE service_ex(
    key string,
    content string
)
STORED BY 'io.transwarp.esdrive.ElasticSearchStorageHandler'
WITH SERDEPROPERTIES(
    'elasticsearch.columns.mapping'=' _id,content',
    'elasticsearch.indextype'=' default_type_ ')
TBLPROPERTIES('elasticsearch.tablename'='service');
```

28.4.2. 插入数据

现在, Fiona需要将已有的邮件数据存放到表 `service_ex` 中:

例 179. 插入数据

```
INSERT INTO TABLE service_ex(key, content)
VALUES("01", "Transwarp TDH is an outstanding product in database
domain, and I will choose TDH as my first choice.");
INSERT INTO TABLE service_ex(key, content)
VALUES("23", "I met IllegalArgumentException but I do not know why.");
INSERT INTO TABLE service_ex(key, content)
VALUES("57", "transwarp tdh is attractive with reasonable price in the
same performance situation compared with teradata");
INSERT INTO TABLE service_ex(key, content)
VALUES("19", "oracle is an outstanding product so is transwarp tdh");
```

例 180. 查看表中内容

```
SELECT *
FROM service_ex;
23 I met IllegalArgumentException but I do not know why.
01 Transwarp TDH is an outstanding product in database domain, and I
will choose TDH as my first choice.
57 transwarp tdh is attractive with reasonable price in the same
performance situation compared with teradata
19 oracle is an outstanding product so is transwarp tdh
```

28.4.3. 数据查询

邮件数据已经保存好了, 现在老板让Fiona查看下用户是怎么评价TDH和其他数据库的。Fiona采用含有“或”的 `CONTAINS` 命令, 来查找包含TDH或Oracle或Teradata的邮件信息:

例 181. 含有“或”的 `CONTAINS` 查询

```
SELECT *
FROM service_ex
WHERE CONTAINS(content, 'tdh OR oracle OR teradata');
01 Transwarp TDH is an outstanding product in database domain, and I
will choose TDH as my first choice.
57 transwarp tdh is attractive with reasonable price in the same
performance situation compared with teradata
19 oracle is an outstanding product so is transwarp tdh
```

接着, Fiona单独将TDH和Oracle之间的对比信息也一并汇报给了老板:

例 182. 含有"和"的 CONTAINS 查询

```
SELECT *
FROM service_ex
WHERE CONTAINS(content, 'TDH AND Oracle');

19 oracle is an outstanding product so is transwarp tdh
```

这时销售部门需要Fiona提供有关Transwarp TDH售价的相关评价，Fiona采用了 **NEAR** 来有效查找：

例 183. 含有 NEAR 的 CONTAINS 查询

```
SELECT *
FROM service_ex
WHERE CONTAINS(content, 'NEAR((tdh, price), 10, TRUE)');

57 transwarp tdh is attractive with reasonable price in the same
performance situation compared with teradata
```

销售部门进一步需要Fiona将对售价的正面评价单独提交过去，Fiona采用了 **MATCHES** 语句：

例 184. MATCHES 选择语句

```
SELECT *
FROM service_ex
WHERE MATCHES(content, 'Reason Price');

57 transwarp tdh is attractive with reasonable price in the same
performance situation compared with teradata
```

29. Transwarp ES常见问题

在Inceptor Engine上创建Transwarp ES表时报错

- 报错信息：

```
java.lang.IllegalArgumentException: Conf missing for elasticsearch,
check your hive-site.xml
```

```
[localhost:10000] transwarp> create table test_es(key string, value1 string) sto
red as es;
[Hive Error]: EXECUTION FAILED: Task DDL error HiveException: [Error 1] java.lan
g.IllegalArgumentException: Conf missing for elasticsearch, check your hive-site
.xml
[localhost:10000] transwarp> █
```

- 环境

TDH产品

- 问题原因

Inceptor没有对应依赖的Transwarp ES配置。

- 解决方法

确保集群上已安装Transwarp ES。接着需要让Inceptor依赖你集群上的Transwarp ES配置，在 `hive-site.xml` 中加上如下语句：

```
<property>
    <name>discovery.zen.ping.unicast.hosts</name> ①
    <value>transwarp-perf2,transwarp-perf3,transwarp-perf4</value>
</property>
<property>
    <name>discovery.zen.ping.multicast.enabled</name> ②
    <value>false</value>
</property>
<property>
    <name>discovery.zen.minimum_master_nodes</name>③
    <value>2</value>
</property>
<property>
    <name>cluster.name</name>④
    <value>es-upgrade</value>
</property>
```

① Transwarp ES的所有节点。注意，一定要所有的Transwarp ES节点都放在里面。

- ② 默认false即可
 - ③ 小于集群Transwarp ES Master的个数
 - ④ Transwarp ES的集群名字

在Inceptor Engine上创建Transwarp ES表时报错

- 报错信息:

Error: EXECUTION FAILED: Task DDL error HiveException: [Error 1] MasterNotDiscoveredException[waited for [30s]] (state=08S01, code=1)

图 6. esdriver建表报错

```
2016-08-23 11:43:15,258 ERROR esdrive.ElasticSearchStorageHandler: {ElasticSearchStorageHandler.java:preCreateTable(199)} [HiveServer2-Handler-Pool: Thread-213(SessionHandle=1026794d-d55c-4179-a6a0-8721ce0ebe5d)] - MasterNotDiscoveredException[waited for [30s]]
2016-08-23 11:43:15,262 ERROR exec.DDLTask: {DDLTask.java:execute(569)} [HiveServer2-Handler-Pool: Thread-213(SessionHandle=1026794d-d55c-4179-a6a0-8721ce0ebe5d)] - org.apache.hadoop.hive.ql.metadata.HiveException: MasterNotDiscoveredException[waited for [30s]]
    at org.apache.hadoop.hive.ql.metadata.Hive.createTable(Hive.java:1008)
    at org.apache.hadoop.hive.ql.exec.DDLTask.createTable(DDLTask.java:508)
    at org.apache.hadoop.hive.ql.exec.DDLTask.execute(DDLTask.java:280)
    at org.apache.hadoop.hive.ql.exec.Task.executeTask(Task.java:164)
    at org.apache.hadoop.hive.ql.exec.TaskRunner.runSequential(TaskRunner.java:89)
    at org.apache.hadoop.hive.ql.Driver.launchTask(Driver.java:2337)
    at org.apache.hadoop.hive.ql.Driver.execute(Driver.java:2072)
    at org.apache.hadoop.hive.ql.Driver.runInternal(Driver.java:1526)
    at org.apache.hadoop.hive.ql.Driver.run(Driver.java:1382)
    at org.apache.hadoop.hive.ql.Driver.run(Driver.java:1352)
    at io.transwarp.inceptor.server.InceptorSQLOperation.runInternal(InceptorSQLOperation.scala:66)
    at org.apache.hive.service.cli.operation.Operation.run(Operation.java:279)
    at org.apache.hive.service.cli.session.HiveSessionImpl.executeStatementInternal(HiveSessionImpl.java:423)
    at org.apache.hive.service.cli.session.HiveSessionImpl.executeStatementWithParamsAndPropertiesAsync(HiveSessionImpl.java:390)
    at org.apache.hive.service.cli.CLIService.executeStatementWithParamsAndPropertiesAsync(CLIService.java:320)
    at io.transwarp.inceptor.server.InceptorCLIService.executeStatementWithParamsAndPropertiesAsync(InceptorCLIService.scala:130)
    at org.apache.hive.service.cli.thrift.TCLIService.ExecuteStatement(TCLIService.java:538)
    at org.apache.hive.service.cli.thrift.TCLIServiceProcessor$ExecuteStatement.getResult(TCLIService.java:1737)
    at org.apache.hive.service.cli.thrift.TCLIServiceProcessor$ExecuteStatement.getResult(TCLIService.java:1722)
    at org.apache.thrift.ProcessFunction.process(ProcessFunction.java:39)
    at org.apache.thrift.TBaseProcessor.process(TBaseProcessor.java:39)
    at org.apache.hive.service.auth.TSetIpAddressProcessor.process(TSetIpAddressProcessor.java:56)
    at org.apache.thrift.server.TThreadPoolServerWorkerProcess.run(TThreadPoolServer.java:285)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1145)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615)
    at java.lang.Thread.run(Thread.java:45)
Caused by: MetaException(message:MasterNotDiscoveredException[waited for [30s]])
    at io.transwarp.esdrive.ElasticSearchStorageHandler.preCreateTable(ElasticSearchStorageHandler.java:200)
    at io.transwarp.esdrive.ElasticSearchStorageHandler.preCreateTable(ElasticSearchStorageHandler.java:133)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
```

图 7. hive-server2.log 报错信息

```
[2016-08-23 10:48:33,269][WARN ][discovery.zen           ] [hkbank03] failed to connect to master [[hkbank02]{OKonj_FoTpCyyly6c9_OuA}{fe80::5cf3:fcff:fe51:7d43}{{fe80::5cf3:fcff:fe51:7d43}}{[fe80::5cf3:fcff:fe51:7d43]:9300}{{fe80::5cf3:fcff:fe51:7d43}:9300} {rack=Default, master=true}], retrying...  
ConnectTransportException[[hkbank02][[fe80::5cf3:fcff:fe51:7d43]:9300] connect_timeout[30s]]; nested: ConnectTimeoutException[connection timed out: /fe80:0:0:0:5cf3:fcff:fe51:7d43];  
Caused by: org.elasticsearch.transport.netty.NettyTransport.connectToChannels(NettyTransport.java:922)  
at org.elasticsearch.transport.netty.NettyTransport.connectToNode(NettyTransport.java:855)  
at org.elasticsearch.transport.netty.NettyTransport.connectToNode(NettyTransport.java:828)  
at org.elasticsearch.transport.TransportService.connectToNode(TransportService.java:243)  
at org.elasticsearch.discovery.zen.ZenDiscovery.joinElectedMaster(ZenDiscovery.java:428)  
at org.elasticsearch.discovery.zen.ZenDiscovery.innerJoinCluster(ZenDiscovery.java:360)  
at org.elasticsearch.discovery.zen.ZenDiscovery.access$5000(ZenDiscovery.java:84)  
at org.elasticsearch.discovery.zen.ZenDiscovery$JoinThreadControl$1.run(ZenDiscovery.java:1245)  
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)  
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)  
at java.lang.Thread.run(Thread.java:745)  
Caused by: org.jboss.netty.channel.ConnectTimeoutException: connection timed out: /fe80:0:0:0:5cf3:fcff:fe51:7d43:9300  
at org.jboss.netty.channel.socket.nio.NioClientBoss.processConnectTimeout(NioClientBoss.java:139)  
at org.jboss.netty.channel.socket.nio.NioClientBoss.process(NioClientBoss.java:83)  
at org.jboss.netty.channel.socket.nio.AbstractNioSelector.run(AbstractNioSelector.java:337)  
at org.jboss.netty.channel.socket.nio.NioClientBoss.run(NioClientBoss.java:42)  
at org.jboss.netty.util.ThreadRenamingRunnable.run(ThreadRenamingRunnable.java:108)  
at org.jboss.netty.util.internal.DeadLockProofWorker$1.run(DeadLockProofWorker.java:42)  
... 3 more  
[2016-08-23 10:49:07,878][WARN ][discovery.zen           ] [hkbank03] failed to connect to master [[hkbank02]{OKonj_FoTpCyyly6c9_OuA}{fe80::5cf3:fcff:fe51:7d43}{{fe80::5cf3:fcff:fe51:7d43}}{[fe80::5cf3:fcff:fe51:7d43]:9300}{{fe80::5cf3:fcff:fe51:7d43}:9300} {rack=Default, master=true}], retrying...  
ConnectTransportException[[hkbank02][[fe80::5cf3:fcff:fe51:7d43]:9300] connect_timeout[30s]]; nested: ConnectTimeoutException[connection timed out: /fe80:0:0:0:5cf3:fcff:fe51:7d43:9300];  
Caused by: org.elasticsearch.transport.netty.NettyTransport.connectToChannels(NettyTransport.java:922)  
at org.elasticsearch.transport.netty.NettyTransport.connectToNode(NettyTransport.java:855)  
at org.elasticsearch.transport.netty.NettyTransport.connectToNode(NettyTransport.java:828)  
at org.elasticsearch.transport.TransportService.connectToNode(TransportService.java:243)  
at org.elasticsearch.discovery.zen.ZenDiscovery.joinElectedMaster(ZenDiscovery.java:428)  
at org.elasticsearch.discovery.zen.ZenDiscovery.innerJoinCluster(ZenDiscovery.java:360)  
at org.elasticsearch.discovery.zen.ZenDiscovery.access$5000(ZenDiscovery.java:84)  
at org.elasticsearch.discovery.zen.ZenDiscovery$JoinThreadControl$1.run(ZenDiscovery.java:1245)  
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)  
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
```

图 8. ElasticSearch的log报错信息

- 环境

TDH4.6 final

- 问题原因

报错前的配置，是因为Transwarp ES启动时的 `publish address` 是 IPv6，而 `hosts` 解析是 IPv4，造成 `master` 监听的网段和 `data` 连接的网段不一致，因此会报错。

- 解决方法

将 `elasticsearch.yml` 中的 `network.host` 参数和 `discovery.zen.ping.unicast.hosts` 两个参数全部改成 IP 的形式，而非 `hostname` 的形式。



界面已经禁止掉了 `discovery.zen.ping.unicast.hosts` 参数的修改，TDH4.6 final 的当前版本只能在底层修改配置文件。

- 报错前的配置

```
network.host: "[::]"  
discovery.zen.ping.unicast.hosts:  
["hkbank01", "hkbank02", "hkbank03"]
```

- 修改后的配置

```
network.host: "192.168.41.103"  
discovery.zen.ping.unicast.hosts:  
["192.168.41.101", "192.168.41.102", "192.168.41.103"]
```

客户服务

技术支持

感谢你使用星环信息科技（上海）有限公司的产品和服务。如您在产品使用或服务中有任何技术问题，可以通过以下途径找到我们的技术人员给予解答。

Email: support@transwarp.io

技术支持热线电话: 4008 079 976

技术支持QQ专线: 3221723229, 3344341586

官方网址: www.transwarp.io

意见反馈

如果你在系统安装，配置和使用中发现任何产品问题，可以通过以下方式反馈:

Email: support@transwarp.io

感谢你的支持和反馈，我们一直在努力！