TextEngine

# 1. 系统构成

TextEngine（简称TE）是CIC Tech 开发的分布式存储系统，旨在为大量BBS文本存储以及数据处理提供一个全面的解决方案，包括容量的可扩充性，数据的冗余备份，以及访问时的缓存和负载平衡。

系统组件包括核心的存储部分（Text Repository，简称TR），支持数据处理业务流的ItemImporter，以及支持其他系统读取TE数据的TEClient。

# 2. 存储架构

## 存储系统架构图

Data Node

IDF

IDF Engine

Repository Engine

Data Node Daemon

IDF Engine

IDF

Name Node

Data Node

Name Node

Repository

Name Node

Daemon

## 基本概念

* + 1. **Item**

Item是一个帖子的内部表示，包含两部分信息，即帖子元信息(Site, Forum, Year, Month, Poster, DateOfPost)，以及正文(Subject, Content)。

* + 1. **Partition**

Partition目前表示某一个网站中的一个论坛，在某年某月中发表的帖子的集合，是TE中数据存储的最小逻辑划分，具体实现中对应于文件系统一个的目录。通过year, month, site\_id, forum\_id四个字段可以唯一的确定一个partition。于是我们用这四个字段产生一个字符串表示Partition Key。其中site\_id包括数据源信息，以及由数据源定义的site\_id。例如：“FF1”表示数据源为FixForum，1是由数据源来定义的site\_id，两个字段合起来作为一个site\_id字段存储于TE当中。forum\_id是由数据源确定的。这样，每个数据源自己定义的site\_id和forum\_id组合的唯一性，使得TE的Partition Key具有唯一性。

* + 1. **ItemID**

ItemID 是一个Item在Partition中的编号，按照item写入TE的顺序自增的。。注意，ItemID是by partition的。

* + 1. **ItemKey**

在Partition Key所需的四个字段之后再加上ItemID, 则构成ItemKey。Item Key能够在全局唯一确定一个item。

* 1. ***DNKey***

用来标识一个Data Node

## 组件介绍

* + 1. **Data Node**

Data Node 是一台提供数据服务的物理机器，包含数据的存储部分和提供数据访问的接口程序

* + 1. **IDF:**

Item Data File 的简称，一个IDF是物理存储的最小单元，包含Index 和 Data 两部分。

Index中包含数据Item的存储位置信息记录，每一个记录的结构为

ItemID: 4bytes

ItemAddress: 8bytes

ItemLen: 4bytes

给定 ItemID，可以取得该Item 在data块中的开始偏移地址和长度，从而获取内容。

每个IDF最多存储Item数为1M（1024＊1024）

* + 1. **IDF Engine:**

IDF Engine 为一个IDF 文件对外的访问接口，通过其可对IDF文件进行下列操作

* 管理内部ItemID
* 初始化 IDF
* 向IDF 添加信息
* 从IDF删除信息
* 给定ItemID，从IDF获取内容
* 清空IDF, 即将IDF内容清空，并且重新初始化
* 检查IDF的数据完整性

IDF Engine 没有实现线程安全性，也没有文件级别的锁机制，因此在使用时需确保任何时候没有两个IDF Engine实例在对同一个IDF文件进行操作，不论是JVM内还是垮JVM。

* + 1. **Partition**

目前TE的实现将Item按照Site, Forum, Year, Month来划分，每一个划分单元为一个Partition，对应文件系统中的一个或多个IDF文件，IDF文件名为Partitioin的信息按照一定规则编码得出。

* + 1. **Repository Engine**

Repository Engine 对一台服务器上所有IDF 形成的库(repository)进行管理，负责将外界提供的Item的Partition信息 (site, forum, year, month) 映射到该Item所在IDF的物理地址(文件系统目录和文件名)，并返回对应的IDF Engine。客户端进而可以通过IDF Engine进行后续操作。

* + 1. **Data Node Daemon**

Data Node Daemon 是运行在Data Node服务器上的守护程序，负责监听来自Name Node以及数据访问客户端的指令。

|  |  |
| --- | --- |
| PING | 返回DN当前已运行的时间（毫秒） |
| SHUTDOWN | 关闭DNDaemon |
| ADD\_ITEMS | 添加Item数据 |
| REMOVE\_ITEMS | 删除多条数据 |
| QUERY\_ITEMS | 给定一组ItemID，返回所有内容 |
| ADD\_ITEMS\_STREAM | 以流的方式批量添加数据 |
| QUERY\_IDF\_META | 返回IDF的meta信息(未实现) |
| ASSIGN\_DN\_KEY | 赋予Data Node Key |
| ASSIGN\_NN\_DAEMON | 告知Name Node 所在地址(host, port) |
| ENUMERATE\_ITEMS | 给定起始ItemID和Item count，遍历并返回所有的Item |
| CHECK\_DN\_PARTITION\_WRITE\_LOCK | 检查DN上某Partition的Write Lock是否存在 |
| SYNC\_PARTITION | 同步partitioin |
| QUERY\_ONE\_ITEM | 给定一个ItemID返回Item数据 |
| PERSIST\_ADD\_ITEMS | 保持DN连接的方式添加多个Partition的数据（未实现） |
| DELETE\_ITEMS\_BYCONDITION | 按照条件删除Items |
| GET\_FREE\_SPACE | 获取当前DN所在硬盘的剩余空间 |
| QUERY\_ITEMS\_BYCONDITION | 按照条件获取Items |

* + 1. **Name Node**
    2. **Name Node Daemon**

Name Node Daemon 是运行在Nata Node服务器上的守护程序，监听来自客户端和Data Node的指令。

特别的，负责协调Data Node的注册，负载平衡，同步等操作

|  |  |
| --- | --- |
| SHUT\_DOWN | 关闭NameNode |
| PING | Ping |
| APPLY\_PARTITION\_WRITE\_LOCK | 给特定Partition加锁 |
| RELEASE\_PARTITION\_WRITE\_LOCK | 释放特定Partition的锁 |
| GET\_PARTITIOIN\_APPEND\_POINT | 要在特定Partition 上新增数据，起始的ItemID |
| GET\_DN\_CLIENT\_FOR\_APPENDING | 获取用作新增数据的DataNode Client |
| GET\_DN\_CLIENT\_FOR\_QUERY | 获取用作查询数据的DataNode Client |
| GET\_DN\_PARTITION\_ITEM\_COUNT | 获取特定Data Node特定Partition上已有的Item数量 |
| CLEAN\_PARTITION | 清空特定Partition |
| GET\_NEXT\_DN\_PARTITION\_OPERATION | 给定Data Node以及Partition，获取该Data Node 下一步的操作（用作Partition同步） |
| UPDATE\_DN\_PARTITION\_VERSION | 更新给定Data Node上指定Partition版本 |
| DEACTIVATE\_DATA\_NODE |  |
| GET\_DN\_ADDRESS\_FOR\_QUERY | 给定Partition信息，返回可以读取该Partition的一个Data Node |
| GET\_DN\_ADDRESS\_FOR\_APPEND | 给定Partition信息，返回可以写入信息的Data Node |
| CLEAN\_NN\_CACHE | 清空NameNode上的Partition Item count的缓存信息 |
| GET\_DN\_LIST\_FOR\_QUERY | 给定Partition信息，返回可以读取Partition的全部Data Node |

* + 1. **Name Node Repository** 数据库 信息

表名：T\_DATANODE 存放 Data Node 的相应信息

|  |  |
| --- | --- |
| DN\_KEY | 主键，唯一的表示每个DataNode。 |
| REGISTER\_DT | 该DN最早注册在NN上的时间。 |
| LASTACTIVATE\_DT | 该DN最新一次启动(active)的时间。 |
| DAEMON\_IP | 该DN的IP地址。 |
| DAEMON\_PORT | 该DN使用的端口地址。 |

表名：T\_DN\_PARTITION 存放Data Node 和 Partition 的关联信息

|  |  |
| --- | --- |
| ID | 自增的字段。 |
| DN\_KEY | 含义同T\_DATANODE中的DN\_KEY字段。 |
| PARTITION\_ID | 含义同T\_PARTITION中的PARTITION\_ID字段。 |
| PARTITIONI\_ITEM\_COUNT | 该DN上的这个Partition中的数据数目。 |
| TS\_LAST\_MODIFY | 上次修改时间。 |
| VERSION | 该DN上的这个Partition的版本信息。 |

表名：T\_PARTITION，存放Partition的信息

|  |  |
| --- | --- |
| PARTITION\_ID | 自增的字段，唯一的表示一个partition。 |
| THE\_YEAR | partition所在的年份。 |
| THE\_MONTH | partition所在的月份。 |
| SITE\_ID | partition key中定义的site\_id，包括数据源信息以及由数据源定义的siteid。 |
| FORUM\_ID | 由数据源定义的forumid。 |
| CREATE\_DT | 这个partition的建立时间。 |
| VERSION | 这个partition的最新版本。 |
| TS\_LASTM | 这个partition的最近更改时间。 |
| ITEM\_COUNT | 这个partition的最新版本中包括的item数目 |

表名：T\_PARTITION\_OPRERATION\_LOG，存放Partition操作的log

|  |  |
| --- | --- |
| ID | 唯一表示某个操作的标识。 |
| PARTITION\_ID | 含义同T\_PARTITION中的PARTITION\_ID |
| VERSION | 版本ID |
| OPERATION\_TYPE | 该版本对应的操作类型。1添加；2 清除；3删除 |
| OPERATION\_DATA | 该操作对应的数据。添加或者删除操作对应的item\_id |
| TS | 该操作的时间 |

## 系统内部行为

* + 1. **Data Node 注册 (DataNode Registration)**

Data Node 启动后，先尝试读取存放本地的DNKey，如果没有读到，则在稍后向Name Node发送心跳信息时通过注册来得到一个DNKey

* + 1. **Data Node 心跳检查(Data Node Heartbeat)**

1. 每10 秒钟(参数可调) 广播一次心跳信息。
2. 每一次心跳会通知DNRegistryTable中相应DNRegistry 来restTTL, 以免 expire。
3. NameNode 轮询检查DataNode的心跳信息，如果发现超过一定时间没有该DataNode的心跳，则 判定 expire, 并取消该Data Node的注册。如果DataNode 刚刚启动（判断其是否已经包含Name Node 信息), 则通知DataNode进行Partition同步

**数据写入（包括删除）**

1. 获取可以写入数据的DN

NN先判断该partition是否存在：若存在则选择写入次数最少的DN；若不存在则选择当前剩余空间最大的DN

1. 向DN数据写入：

DN检查本地是否存在对该partition的write lock，若存在则拒绝写入；若不存在，则向NN获取partition的write lock以及写入开始的item id，然后开始写入数据。

1. 写入完成

写入完成后，释放NN和DN本地的write lock，并返回写入的数据量。NN在处理释放partition write lock之后，会记录写入操作，并向其他存储该partition数据的DN发起数据同步请求

* + 1. **Partition Lock**

为了保证TE添加数据时的事务性，在NNDaemon和DNDaemon的程序中都加入了对partition的内存锁PartitionLock。在NNDaemon上锁保证不会有对同一个partition在多个DN进行不同的写操作，而在DNDaemon上锁保证不会同时写同一个文件。添加数据的时候，首先在需要添加数据的DN上锁，成功之后再向NN申请上锁，如果上锁失败，则将DN上的锁释放掉。数据添加完成之后，先释放掉NN上的锁，在释放掉DN上的锁。一旦中途操作失败，需要将已经上的锁释放掉。

* + 1. **Partition 同步(PartitionSync)**

基本思想是通过版本和Operation Log来实现Data Node上Partition的递增式同步

1. Data Node 收到Partition同步指令时，将自身的DNKey, 以及分区信息发送给Name Node
2. Name Node 通过 T\_DN\_PARTITION 表查询该Data Node当前Partition的Version, 跟 T\_PARTITION\_OPERATION\_LOG中的相同Partiton的Version对比，得到相邻下一个版本的Operation Log 项(通过OLogItem实现)， 返回给Data Node
3. Data Node 根据 OlogItem包含的指令信息获取自己还没有执行的操作。特别的，当指令为添加Item时，OLogItem会提供另一台已经存储相应Item的Data Node(称为SeedDN)和相应的Partition, Start ItemID, Item Count 信息，因此原Data Node可以将Item从SeedDN读取过来并存储。

**NameNode的缓存机制**

在访问TE中的数据的时候，需要查询要从哪个DataNode上读取或者写入数据以及DataNode上某个partition的item count，在负载比较大的时候对数据库频繁操作导致查询操作很慢，为了解决这个问题，目前对Data Node Partition Item count进行了缓存。

在NNDaemon程序的内存中记录了已经查询data node partition item count的数字，最多记录10000条。当记录数超过10000条的时候，会按照记录添加的时间排序，删除最早添加的5000条记录，并且在日志中记录下缓存的命中率。

# 3. 外围组件

## 数据导入

ItemImporter是目前数据批量导入TE的方法，主要包括数据文件隔离，解析，上传，PostTrend更新，数据固化，数据删除，XML移除7个步骤。

1. 数据文件隔离(ProcessIsolationXML.java):是将要处理的文件移动到一个指定的文件夹当中，等待后续处理。如果isolation文件夹中已经有数据则不会移动新数据进来。在这个过程中会查询数据库中是否有等待执行的删除操作，如果有的话，则不会将最后修改时间在数据库中未执行的删除操作提交时间之后的数据文件移动过来。这样可以保证不会删除正确的数据。如果所有在未执行的删除操作提交之前的数据都已经被移动过来的话，则更改删除操作的状态为1（未执行为0，等待执行为1，执行完毕为0），在后续过程中执行，并且将删除操作的条件写成“TE\_Item\_Delete+时间戳+.log“的文件记录下来。
2. 数据文件解析(ProcessParseXML.java):将处于隔离文件夹中的数据文件(xml或者gz结尾的文件)，进行解析，在本地的TERepo目录中生成IDF文件，相同partition的数据会group到一起。解析时会用到bloomfilter进行去重。如果这个过程中途打断的话，对于已经被解析的item而言再重新开始这个过程的时候，可能会被bloomfilter判断为重复数据（看是否已经写入bloomfilter的文件）。
3. 数据上传(ProcessUploadItem.java): 开启一个线程池将本地的IDF文件上传到DataNode当中。上传的时候，加入了TEFilter的功能，可以过滤出符合某些条件的数据。目前是将这些数据的item key存在数据库当中，等待添加filter的应用从数据库中将数据取走。
4. PostTrend更新(ProcessUpdatePosttrend.java):根据上传的数据量更新T\_POSTTREND中的数据。
5. 数据固化(ProcessDataConsolidate.java):固化是指在数据进入分析系统之前打标签，保证进入分析系统的数据不会被删除。目前是在TEPostTrend数据库当中的T\_POSTTREND表中的checkpoint\_post\_count, checkpoint\_item\_id以及checkpoint\_date中记录相应信息作为标签来判断是否已经固化。固化是按照partition来进行的。固化完成时，会出发调用leoworkflow的api，表示数据固化完成，可以进行数据交付。
6. 数据删除(ProcessDeleteItem.java): 删除过程又分成两个子过程：
   1. 先从数据库中获得删除操作的条件，然后从索引库中查询得到相应的数据，然后按照partition group起来，并计算出删除操作后partition的post trend。
   2. 将group的结果依次从TE和bloomfilter中将数据删除，更新posttrend。
7. 数据导入完成(ProcessFinishXML.java):将isolation文件夹中的数据文件包括删除操作的记录，移到finished目录中按照当前时间戳建立的文件夹当中。

## 数据读取

TE为了explorer数据处理流程中的Data Delivery（Kettle），CIC Store，Index Engine， Rules Engine以及Leo Analytics读取TE中的数据，提供了TE的“客户端”调用方法TEClient。

为了支持这些应用基于TE ItemKey的快速随机Item的读取，TEClient里缓存了Partition 与DN的对照关系，并且对当前从每个DN读取数据的速度进行了统计，在一个TEClient实例中，每次通过TEClient获取数据时，都会在该数据Partition所在的DN中，选择活跃并且响应时间最短的DN来进行数据读取。

TEClient初始化时需要指定TE NameNode的IP和port即可。

# 4. 运维支持

## 程序部署，启动

由于数据同步的关系，目前DN和NN以及ItemImporter都没有配置开机自启动。

最新代码的SVN版本为2.4.0，SVN路径为 http://192.168.1.8/svn/TextEngine/branch/TE2.4.0beta

SVN CO代码，并编译后，需要根据机器的环境修改配置文件，其中：

NN需要修改的配置文件和配置项包括：

1）log4j文件里日志文件的路径

2）NameNode.properties配置文件中的NameNode的IP，端口，以及广播的配置；NameNode Repository的数据库配置

DN需要修改的配置文件和配置项包括：

1）log4j文件里的日志文件的路径

2）DataNode.properties 配置文件中的DataNode的端口以及广播的配置；IDFRepository文件夹的路径。

ItemImporter需要修改的配置文件和配置项包括：

1）log4j文件里日志文件的路径

2）ItemImporter.properties 里NameNode的IP和端口，BloomFilter文件的路径，IWMWorkflow的IP和端口；去重开关。

3）GlobalDB.properties IWM\_Global数据库的配置

4）PostTrendDB.properties 里PostTrend数据库的配置

5）DCMISdb.properties里DCMIS数据库的配置

NN启动方法：

1) 使用CICDATA\\opr\_te账号登录192.168.2.2，进入~/TextEngine 目录下

2) 执行 nohup ./NNDaemon.sh 2>./err.out &

DN启动方法：

1) 使用 te\_opr 账号登录到每台Data Node，进入~/TextEngine 目录下

2) 执行 nohup ./DNDaemon.sh & （启动前需确认TE的数据盘已经挂载到/te\_data目录下)

ItemImproter启动方法

1) 使用 te\_opr 账号登录到 192.168.2.11

2) 检查DC的存放XML的文件夹是否已经挂载好了；如果没有，则切换到root账号后，进入 /home/te\_opr/TE\_item\_remote下，执行 mount\_remote.sh 脚本

3) 使用te\_opr 账号进入 ~/TextEngine 目录下

4) 执行 nohup ./ItemImporter.sh ~/TE\_item\_remote/ &

## 数据解固化

数据解固化操作，在确认解固化对项目的影响后，如果要执行解固化操作，可以：

1. te\_opr账号登录到192.168.2.11上，在TextEngine目录下执行 Unconsolidate.sh命令

2. Unconsolidate 命令支持的参数包括:

a) ProjectID，必填，解固化的项目编号，数字格式

b) Month，必填，解固化的月份，格式为年＋月，比如“201108”，可输入多个月份以半角逗号分隔，比如：“201107,201108“

c) Site ID，可选，解固化的网站ID（TE site ID），仅可输入一个网站，比如："FF1"，或者"SERA1"

d) Forum ID，可选，解固化的论坛ID（TE forum ID），输入Forum ID则必须输入Site ID，仅可输入一个论坛，比如："FID1FID"

3. 示例：

a) 解固化452项目，2011年7月，8月数据：./Unconsolidate.sh 452 201107,201108

b) 解固化452项目，2011年8月，site：FF1094下的数据： ./Unconsolidate.sh 452 201108 FF1094

c) 解固化452项目，2011年8月，site：FF1094，forum：fid1fid的数据：./Unconsolidate.sh 452 201108 FF1094 fid1fid

## 关闭数据去重

修改ItemImporter配置文件中IsBloomOn的值，true表示开启去重，false表示关闭去重。配置文件修改后，需要重启ItemImporter才能生效。

关闭ItemImporter的方法 ，在192.168.2.11上，TextEngine目录下执行

./StopItemImporter.sh 192.168.2.11 ItemImporter会在当前步骤执行完成后退出。在确认程序退出后，再启动ItemImporter。

## 新数据写入报错

数据写入Data Node时（新数据写入/同步）如果由于故障重启，可能会出现本地index文件中的item count并未写入，但是数据库里的记录数已更新的情况。这个bug一直没找到原因。

错误日志示例如下：

com.cic.textengine.repository.datanode.repository.exception.RepositoryEngineException: Can not add items to partition from the position which exceed the current item count in the partition.[startItemIdx:4846,IDFItemCount:4842 for PK[y:2010,m:12,s:SERA26,f:356]

解决方法：

1)在NameNodeRepository数据库的T\_PARTITION表里找到对应的Partition记下Partition ID；在T\_DATANODE中找到对应的Data Node

2）T\_DN\_PARTITION中找到对应的DN的Partition记录，将该记录的VERSION－1，并修改ITEM COUNT为错误日志中的本地的Item Count

## 其他错误排查

在数据交付计算流程中如果发现报错，可以通过TEClient读取出错数据的ItemKey来分析错误原因。