How does the Alfresco Node Lifecycle work?

Environment

Problem

How does the Alfresco Node Lifecycle work?

Overview

A node is a specialized object with properties and content. A node is an abstract identity that can represent several things, like a document or a content space. A node has hierarchical associations with other nodes to identify content and content spaces.

Typically, a single node is comprised of metadata and content. The metadata is stored in the database. The content is stored in the contentstore (on disk).

Every node also is stored in one *"contentstore"*. The "*contentstore(s)"* are on the filesystem. In Alfresco, there are a few types of *"contentstore(s)"* out of the box. Each "*contentstore*" represents and is used for a stage related to the life cycle of a node.

For indexing and searching purposes, the "indexes" generated against the metadata and content (full text indexing) are also stored in the filesystem. In the case of "Lucene" this would be at the location defined in the configurations for the "dir.indexes=${dir.root}/lucene-indexes","dir.indexes.backup=${dir.root}/backup-lucene-indexes","dir.indexes.lock=${dir.indexes}/locks". In the 4.0.x and newer version if you are using "SOLR" these located where you have defined in the "SOLR" application's configuration.

The principal difference with old versions ("lucene") and "SOLR" search subsystem is that "SOLR" does not use in transaction indexing anymore. This means that all the indexing operations are realized outside of the transaction, in a background style.

All nodes in the repository follow the same lifecycle from the creation through deletion. This includes several batch processes. As the node travels through its lifecycle it will move to different "contentstores" based on the stage of the lifecycle it is in.

The Node Service along with several background jobs manage a node's lifecycle.

The node service

All the node lifecycle operations are managed by the Node Service. Node Service is also the service responsible for all of the operations that can be applied to nodes.

The follow operations owned by the Node Service are the basic operations involved in the node life cycle:

- NodeRef createNode(NodeRef parentRef, String name, String nodeType, [Map properties]). Creates a new, non-abstract node as a primary child of the given parent node.
- void deleteNode(NodeRef nodeRef). Deletes a given node from the repository.

Cause

Solution

What happens when we create a node?

When we create a node there are several internal steps that happen inside the alfresco server.

Roughly, prior version 4, the server realizes the follow steps when we call *nodeService.createNode*:

1. **We start the transaction.** All the creation of the node is involved in one transaction. This transaction will take care of the consistency and atomicity of the whole operation. For this use, the server will use the *Transaction Service*.

2. **The server inserts into the database the node metadata.** In this step, the server will create the node in the database. For this purpose, the server will use the *Node Service*. The server will update several tables, like *alf_node, alf_node_properties, alf_node_assoc* or *alf_child_assoc*.

3. **The server writes the stream (eg. one image or document) into the content-store (disk).** The server will write the stream into the *content-store*. In this operation, the content will have a new content-url. Typically, the file will be renamed with a new name that looks like:*alf_data/contentstore/yyyy/mm/dd/hh/mm/uuid.bin*

   It's important to notice that this UUID is not the same that the well know node-ref.

4. **The server udpates the database with the new content-url.** The server will update the alf*_content_url* and *alf_content_data* tables with the new file location.

5. **The server will start the commit state.** In this step, the server will begin the commit of the transaction.

6. **The server will extract text and data from the content and update the Lucene index.** Depending of the configuration of the server and the type of the content, the server will extract the desired data. Also, the server will update the lucene indexes in the content-store, usually *alf_data/lucene-indexes/workspace/SpacesStore/,* with the content and the metadata extracted.

   If the transform and update step exceeds 20ms, this step will move outside from the transaction and will be realized at the background.

7. **The server will commit the transaction.** After this steps, the server will commit the transaction and will insert one record on the *alf_transaction* table, using the Transaction Service

8. **The server will update the L2 cache with the new node.** Usually, the server will update the L2 cache with the new node for improving performance. Cache is aware from the transaction always.

Since version 4, search has been moved into a sub-system with a SOLr and Lucene implementations. One of the main differences is that we don't use indexing in transaction anymore. The system works pretty identical that prior release 4, but the step 6 is always done in background.

What happens when we modify a node?

The process of modifying a node is pretty similar to the process that creates one node. There are there are several internal steps that happen inside the alfresco server. As the process that creates nodes, there are differences between versions, related with the launch of the version 4 and the way how version

4 manages the content indexing.

The step that shows the next list can change depending of the nature of the modification. The next list shows the basic common steps for modifying one node, for example, updating the content or updating the metadata.

Roughly, prior version 4, the server realize the follow steps:

1. **We start the transaction.** After the user upload new version, the system will create one transaction. As the creation of a node, this transaction will take care of the consistency and atomicity of the whole operation. This will ensure the atomicity required for being sure the data consistency.

2. **The server updates the database metadata.** In this step, the server will update the tables related with the node in the database. For this purpose, the server will use the *Node Service*. The server will update several tables, like *alf_node, alf_node_properties, alf_node_assoc* or *alf_child_assoc.*

3. **The server writes the stream (eg. one image or document) into the content-store (disk).** If the modification action involves changes in the stream, the server will write the stream into the *content-store* as a new file with a new content-url.
It's important to know that the server never changes the content in the content-store. The server always will create one new file.

   It's important to know that the server never changes the content in the content-store. The server always will create one new file.

4. **The server udpates the database with the new content-url.** If the modification action involves changes in the stream, the server will update the alf*_content_url* and *alf_content_data* tables with the new file location on the content-store.

5. **The server will start the commit state.** In this step, the server will begin the commit of the transaction.

6. **The server will extract text and data from the content and update the Lucene index.** Depending of the configuration of the server and the type of the content, the server will extract the desired data. Like in the indexes we store information related with the stream and the metadata, this step will happens always in versions prior 4.

   Also, the server will update the lucene indexes in the content-store, usually *alf_data/lucene-indexes/workspace/SpacesStore/,* with the content and the metadata extracted.

   If the extract of data and the update step exceeds 20ms, this step will move outside from the transaction and will be realized at the background.

   Also, the server will update the lucene indexes in the content-store, usually *alf_data/lucene-indexes/workspace/SpacesStore/,* with the content and the metadata extracted.

   If the extract of data and the update step exceeds 20ms, this step will move outside from the transaction and will be realized at the background.

7. **The server will commit the transaction.** After this steps, the server will commit the transaction and will insert one record on the *alf_transaction* table, using the Transaction Service

8. **The server will update the L2 cache with the new node.** Usually, the server will update the L2 cache with the new node for improving performance.

Since version 4, as we don't use indexing in transaction anymore, the step 6 is always done in background.

We must notice that any change in any node of the system will generate one record on the *alf_transaction* table.

What happens when we delete a node?

The process of node deletion is a bit more complex. For deleting completely one node from the system, the server must realize 3 high level steps:

- **Delete the node.** This step is typically fired by the user when delete one content from the explorer browser, share or any other user interface. Roughly, the content doesn't disappear from the system. The content is moved to the trash.
- **Empty the trashcan.** In the Alfresco trash, we will find all the deleted nodes. This step is

typically fired by the user or can be scheduled via the TrashCanCleaner job on the forge (http://forge.alfresco.com/gf/project/trashcancleaner/).
Manually, you can purge the trashcan directly from the Alfresco Explorer. You will find this option under *User Profile >> Manage Deleted Items >> Purge*

- **Content Store Cleaner and Deleted node cleaner Jobs.** The last steps are done by scheduled jobs. These scheduled jobs will erase definitely the nodes from file system and database.

When a user deletes one node from any interface, internally the system will call the *nodeService.deleteNode()*. This method will perform the follow tasks:

1. **The server will start the transaction.** For ensuring the atomicity of the operation, the delete process needs to be enclosed into a transaction.

2. **The server updates the node store in the database.** After create the transaction, the *.bin* file will be unaltered into the content-store. But the server will update the table *alf_node* and change the ID of the field *store_id.* In fact, internally, Alfresco will change the node store from the *workspace* to the *archive,*changing the content-url even if the file is in the same location. We can look this behavior easily looking at the follow urls:

   Live Node: *workspace/Spaces/Store/nodeuuid*

   DeletedNode: *archive/SpacesStore/nodeuuid*

3. **The server will start the commit state.** In this step, the server will begin the commit of the transaction.

4. **The server will move the IDX to the archive.** The system stills having the indexes of the node, but also the server will move these indexes to a different space: *alf_data/lucene-indexes/archive/SpacesStore.* The original file will be removed from the location */alf_data/lucene-indexes/workspace/SpacesStore.*

5. **The server will commit the transaction.** After this steps, the server will commit the transaction and will insert one record on the *alf_transaction* table, using the Transaction Service

6. **The server will update the L2 cache with the new node.** The server will update the L2 cache and remove it for improve performance.

If the user decides to undelete the content, the server will change another time the *store_id* in the *alf_node* database and restore the lucene index to its original location in*/alf_data/lucene-indexes/workspace/SpacesStore.*

What happens when we empty the trashcan?

The next step for removing the node permanently in the alfresco system is to empty the trashcan. The trashcan is where the documents and another kind of contents live after the user delete them.

Roughly, this step will call the method a*rchiveService.purge.* The steps involved in this process are the follow:

1. **The server will start the transaction.** For ensuring the atomicity of the operation, the process needs to be enclosed into a transaction.

2. **The server updates the *node_deleted property*.** After create the transaction, the *.bin* file will be unaltered into the content-store, the same way as when the node is deleted.

   The server will update several tables in the database. The first step is to update the table *alf_node* and set the *node_deleted property* to true (1).

3. **The server will set the current timestamp as *orphan_time.*** The server will set the property *orphan_time* in the *alf_content*_url table to the current time.

4. **The server will delete the node from related tables.** After setting the *node_deleted* property, the system will delete the records from other tables, like*alf_node_properties, alf_node_assoc or alf_chil_assoc*.

5. **The server will start the commit state.** In this step, the server will start the commit of the transaction.

6. **The server will remove the indexes from the lucene archive store.** The server will remove any entry related with this node in the search indexes. This means to remove the IDX from the last location: *alf_data/lucene-indexes/archive/SpacesStore.*

7. **The server will commit the transaction.** After these steps, the server will commit the transaction and will insert one record on the *alf_transaction* table, using the Transaction Service.

It's important notice that the server will no delete the content from the file system. This file (the *.bin* file) will be considered by the system as an *orphan* node.

There are several reasons that can get an orphan node like a result. For example, if we look at the renditions, a thumbnail could become an orphan node if we try to move or copy it directly instead of copy its image parent.

If we notice that we have orphan nodes in our system, could be useful to launch the follow query for understanding if this orphan node is related with the node life cycle or is related with other cause:

SELECT * FROM alf_content_url where orphan_time is not null

This query is based on the assumption that when a node is deleted from the trashcan, the system will set on the *orphan_time* the current timestamp. This query will let to know which the legal orphan nodes on our system are.

Content Store Cleaner job

As we see, in the previous steps the content is not deleted from the file-system. Also we still have the node in the *alf_node* table. There is a background job that will delete the content in the file-system. This measure is done because performance and safety, but the principal reason is looking at the backup – restore scenarios.

If we think in one scenario while we must recover our system, we will need to restore the database, file-system and the indexes. As we see, the content in the file-system is never changed. In recovery situations, we can take advantage of this feature because it's not necessary to copy the orphan content within the file-system. The system recovery will be faster in huge systems, minimizing the downtime of the server.

But if the server never cleans the orphan nodes, the file-system will grow without control. Alfresco implements one solution , the Content Store Cleaner job, that will let us to have a protective period for taking advantage of the orphan nodes in recovery scenario, but that will erase them when a reasonable time has taken.

Roughly, this job will identify and delete the orphaned content from the file system. The trigger *contentStoreCleanerTrigger* (defined in <*configRoot*> *omcatwebappsalfrescoWEB-INFclassesalfrescoscheduled-jobs-context.xml)* is the responsible of launch the *contentStoreCleaner* bean (*org.alfresco.repo.content.cleanup.ContentStoreCleaner.java)*.

By default, this job is triggered at 4:00 am each day (0 0 4 * * ?). In a clustered environment, this job could be enabled on a headless (non-public) node only.

You can look the default values of your installation in the path <*configRoot*> *omcatwebappsalfrescoWEB-INFclassesalfrescocontent-services-context.xml*.

As we can see in the configuration of the bean, we can tune the configuration in relation our needs:

- **protectDays.** By default, this property is set to 14 days. Use this property to dictate the minimum time that content binaries should be kept in the *contentStore*. In the default configuration, if a file is created and immediately deleted, it will not be cleaned from the *contentStore* for at least 14 days. The value should be adjusted to account for backup strategies, average content size and available disk space. Setting this value to zero will result in a system warning as it breaks the transaction model and it is possible to lose content if the orphaned content cleaner runs whilst content is being loaded into the system. If the system backup strategy is just to make regular copies, then this value should also be greater than the number of days between successive backup runs.
- **store.** This is a list of *ContentStore* beans to scour for orphaned content.
- **listeners.** When orphaned content is located, these listeners are notified. In this example, the *deletedContentBackupListener* copies the orphaned content to a separate *deletedContentStore*.

By default, this configuration will not actually remove the files from the file system but rather moves them

to the designated *deletedContentStore*, usually*contentstore.deleted*. The files can be removed from the *deletedContentStore* via script or cron job once an appropriate backup has been performed.

For scheduling the job, we must look at the file *<configRoot> omcatwebappsalfrescoWEB-INFclassesalfresco epository.properties* file the property*system.content.orphanCleanup.cronExpression.*

Node Cleaner job

As we see, the Content Store Cleaner Job will erase the content in the file-system. But we still have the node record in the *alf_node* table. For completely deletion of the node in the database, the server will use another job, fired by the *nodeServiceCleanupTrigger*.

This job performs cleanup operations on DM node data, including old deleted nodes and old transactions. In a clustered environment, this job could be enabled on a headless (non-public) node only, which will improve efficiently. By default, this job is triggered at 21:00 am each day (0 0 21 * * ?).

Roughly, this job performs the follow operations:

- This job will delete the *alf_node* database record for the deleted record. By default, the job is configured to delete the nodes after 30 days from their deletion. This means, after 30 days from the *node_deleted* was set to 1. The default amount of days is configured in the property *index.tracking.minRecordPurgeAgeDays*
- This job will remove all the transactions older than 30 days from the *alf_transaction* table.

For scheduling the job, we must override the value *cronExpression* in the *nodeServiceCleanupTrigger* bean. The recommended way to do this configuration is to copy the file *<configRoot>/alfresco/scheduled-jobs-context.xml* to *<extension>/custom-scheduled-jobs-context.xml* and change the value in the custom extension file.

Upload

Related Information