Concept Import Handler

Your Rating: ☆☆☆☆☆ Results: ★★★☆☆ 94 rates

GREY Discussion points on refactoring/rewriting of the ImportHandlers in Data module. GREY

- Main Issues
    - Target Featurs

Main Issues

- the whole concept of import handlers as implemented is very sequential and single threaded. The process is divided in multiple steps:
    - tag all existing content
    - perform the import
    - retag new content
    - delete old content
    - (optionally) activate new/updated content
- dataItem doesn't allow multivalue properties by default
- same name sibling activation is troublesome
- the module enforces transfer of data between steps (list of UUIDs)

side note:

```
1 char == 16 bit == 2 bytes
1 uuid == 37 chars = 72 bytes (+2 bytes overhead per object envelope) = 74 bytes
30K items in the set ~ 2MB of data in memory for no good reason
```

- the tagging before and after the import is slow when dealing with too many items
- the tagging will be broken with hierarchies anyway (should a hierarchy be deleted because no child was updated? or should we traverse tree with all ids to determine parents of updated data to not delete them accidentally? ...)

General (probably) JR related issues:

- the `getChildren()` and search queries on imported content during import can get very slow (+10 secs persimple query or single `getChildren()` call). This is probably due to fact that indexing is done on the background and import is running ahead of the indexes.
- having more then few hundreds of nodes in one folder is also a performance killer ... the impact is even more pronounced visually when browsing tree

- attempt to delete folder with 30K children (throught hierarchy, not directly) leads to OOM (-Xmx512m)
  ... probably because JR tries to keep hold of all transient items before disposing of them ... `org.apache.jackrabbit.core.state.SessionItemStateManager.disposeAllTransientItemStates(SessionItemStateManager.java:738)`

Target Featurs

Features using this module could be:

- separate the steps and allow their parallelization
- use different concepts for when non existent items have to be determined dynamically from changes to reduce overhead
- Provide default implementation helping to create structures similar to version store to deal with distribution of big amounts of nodes