Large number of documents HOWTO

Introduction

- It is possible to include **30 000+** documents in a Lenya publication on even a modest server with 512MB of RAM.
- Yet some constructs make the default Lenya publication slow down enormously with that number of documents. This is because the hierarchy of documents is kept in 1 single XML file: `sitetree.xml`. And because the default Lenya publication is reading this file a lot of times, it gets incredibly slow, even with 10 000 docs.
- In this HOWTO, we do not touch at the sitetree.xml concept, because changing it requires some more modifcations then the ones proposed here. However, with some smaller adjustments (used cocoon profiling to see the bottlenecks)I I was able to speed up the Lenya default pub enormously. I'm running it now with 30 000 documents very smoothly.

HOWTO (Lenya 1.2.4)

1. Java XML processing supports XML documents of 10's of thousands of nodes, yet therefore the **default heap space** needs to be enlarged.

1.
    - In Linux/tomcat (jetty probably also) set the environment variable : JAVA_OPTS="-Xmx256m"
    - In Windows/tomcat, there is a GUI tool called "tomcat5w" where you can set this option for the tomcat service (Maximum Memory Pool option) .
2. The main slow-down problem is that the whole sitetree is passed through 4 navigation pipes (breadcrumb, menu, tabs and search). This is not really necessary, because generally, one only wants to see the root folders, the current document, its parents and its direct descendants. This is why navigation in "Site" mode goes fast: it uses the 'SiteTreeFragmentGenerator'. A similar solution can be applied for the left navigation menu in authoring mode. I created a Cocoon generator ('SelectiveSitetreeGenerator') which only generates the minimal tree to navigate from the currently selected document (see Attachment SelectiveSitetreeGenerator.java).
    - You need to include the SelectiveSitetreeGenerator.java file into the right Lenya src folder: `src/java/org/apache/lenya/cms/cocoon/generation/`
    - Now we declare our generator in `src/webapp/sitemap.xmap`. Add to the generators list in front:

    ```
    <map:generator name="sitetree-selective" label="content,data" logger="sitemap.generator.sitetree-
    selective" pool-grow="2" pool-max="16" pool-min="2" src="org.apache.lenya.cms.cocoon.generation.
    SelectiveSitetreeGenerator"/>
    ```

    - Now we can reference it in our navigation xmap (`src/webapp/lenya/navigation.xmap`) replace our normal sitetree generator:

    ```
    <map:generate type="sitetree">
       <map:parameter name="area" value="{2}"/>
    </map:generate>
    ```

    with our selective one:

    ```
    <map:generate type="sitetree-selective">
         <map:parameter name="area" value="{page-envelope:area}"/>
         <map:parameter name="documentid" value="{page-envelope:document-id}"/>
    </map:generate>
    ```

3. OK, we now only generate the sitetree that we need. Still, the sitetreeselective generator takes on a Pentium 4 / 2.4 Ghz about 200ms to fetch the right components out of the sitetree.xml (30 000 nodes). Unfortunatly, in the current sitemap, this fetch occurs 4 times, which results in slowing down every click to a new document which wasn't cached to about 1 second. This is because the "map:aggregate" construct does not cache the first fetch of this sitetree-selective. The "cinclude" does, which speeds up the rendering of a page with about 600ms for every page you visit (when sitetree was not cached after eg. move)!
    - Therefore, in your `publicaton-sitemap.xml` replace the part:

```
<map:aggregate element="cmsbody">
          <map:part src="cocoon://navigation/{2}/{3}/breadcrumb/{5}.xml"/>
          <map:part src="cocoon://navigation/{2}/{3}/tabs/{5}.xml"/>
          <map:part src="cocoon://navigation/{2}/{3}/menu/{5}.xml"/>
          <map:part src="cocoon://navigation/{2}/{3}/search/{5}.xml"/>
          <map:part src="cocoon:/lenya-document-{1}/{3}/{4}/{page-envelope:document-path}"/>
</map:aggregate>
```

- with:

```
<map:generate src="../../content/util/empty.xml" />
 <map:transform src="xslt/custom/lenyaBodyCincludes.xsl">
        <map:parameter name="rendertype" value="{1}" />
        <map:parameter name="publication-id" value="{2}" />
        <map:parameter name="area" value="{3}" />
        <map:parameter name="doctype" value="{4}" />
        <map:parameter name="url" value="{5}" />
        <map:parameter name="document-path" value="{page-envelope:document-path}" />
 </map:transform>
<map:transform type="cinclude" />
```

  - You should copy the attached XSL file lenyaBodyCincludes.xsl in "yourPUB/xslt/custom/"
  - Note that for this caching to take effect you should make sure that the sitetree-selective pipe (in navigation.xmap) is cached ! On a normal build this should be ok since the default pipe was defined to be caching (in sitemap.xmap).

4. Speeding up delete usecase. the DocumentReferencesHelper looks through all the documents to find links to a document. If don't want to slow down the delete usecase (which looks for broken links in all documents !),
   - You should uncomment the part with the DocumentReferencesHelper statements in `src/webapp/lenya/content/info/delete.xsp`
     See also the attached delete.xsp delete.xsp
     (close the xsp:logic block after <parent-url> and comment out the rest of original logic:block)

     Other xsp's that use the reference helper and which you may want to disable (called via usecase.xmap).

```
src/webapp/lenya/content/info/archive.xsp
src/webapp/lenya/content/info/deactivate.xsp
src/webapp/lenya/content/info/delete.xsp
src/webapp/lenya/content/publishing/referenced-documents.xsp
src/webapp/lenya/content/publishing/screen.xsp
src/webapp/lenya/content/scheduler/screen.xsp
```

Attached files

<<AttachList>>