Concept - Dynamic Page Caching

This concept is implemented for Magnolia 5.4.

From evaluated solutions, the one that was finally implemented is that based on SiteMesh filter.

The implementation itself is part of Advanced Cache Module and is using SiteMesh module to perform it's functions.

The main use is to allow caching of parts of the page that are static even when parts of the page are dynamic.

JIRA ticket

MAGNOLIA-5992 - Getting issue details... STATUS

Related concepts

- Concept - partial caching
- EhCache upgrade
- Personalization and Cache
- Concept - Cache Improvements
- Concept - Configurable cache constraints on renderables
- Concept - Cache arbitrary objects

Prerequisites

Direct Area Rendering improvements

Current implementation of DAR is not sufficient. MAGNOLIA-5895 - Getting issue details... STATUS

Performance

Whole page needs to be rendered if we want to render a single area. This reduces the advantages of snippet caching. We need rendering listeners to be able to improve this. MAGNOLIA-5839 - Getting issue details... STATUS

- The rendering listeners are already implemented, see MAGNOLIA-4990 - Getting issue details... STATUS .

```
public interface RenderingListener {
    public void before(Node content, RenderableDefinition definition, Map<String, Object> contextObjects,
OutputProvider out);
    public void after(Node content, RenderableDefinition definition, Map<String, Object> contextObjects,
OutputProvider out);
}
```

- What is missing is configurable *OuputProvider* which would serve configurable wrapper for ouput. (currently only custom outputprovider for Direct Area Rendering, hereinafter referred as DAR, is provided) - easy to achieve.
- Every listener is called before/after every content node to render.
- We needs to be able send signals to rendering to be able e.g. skip rendering of current content or whole rest of a page after the requested area is rendered.

```
public interface RenderingListener {
    public String init(OutputProvider out);
    public String before(Node content, RenderableDefinition definition, Map<String, Object> contextObjects,
OutputProvider out);
    public String after(Node content, RenderableDefinition definition, Map<String, Object> contextObjects,
OutputProvider out);
}
```

- And return codes like:

```
public class RenderingListenerCode {
    SKIP_RENDERING,
        RENDERING_FINISHED,
        ...
}
```

Quick performance test

Time to render http://demopublic.magnolia-cms.com/demo-project~mgnlArea=stage~.html

Quick performance test

	wo MAGNOLIA-5895	w/ MAGNOLIA-5895
TTFB	0.08398	0.04024
Total	0.08444	0.04053
% Total	100	48

Limitations

- Current implementation of DAR is able to render only areas which exist in JCR as a node (The only reason for this limitation is to prevent blank page to be rendered). The request url for DAR looks like this: http://demopublic.magnolia-cms.com/demo-project~mgnlArea=stage~.html
- MAGNOLIA-5837 - Getting issue details... STATUS
    - We can improve this situation and be able to render also areas which are defined in a template definition: http://demopublic.magnolia-cms.com/demo-project~mgnlArea=main~.html (currently renders the whole page instead)
    - Since we support also nested areas, we would need to specify whole path to an area (to be able check existence of corresponding AreaDefinition): http://demopublic.magnolia-cms.com/demo-project~mgnlArea=main/content~.html
    - Unfortunately some areas don't exist in JCR nor in template definition: http://demopublic.magnolia-cms.com/demo-project~mgnlArea=htmlHeader~.html The solution to this would be make *AreaFilteringListener* configurable to be able skip check for area existence.

DAR for JSP not supported

See Direct Area Rendering, MAGNOLIA-5126 - Getting issue details... STATUS . The JSP uses its own output (*JspWriter*) (see *info.magnolia.templating.jsp.cms.AbstractTag.doTag()).* It's not possible to control output unless we implement wrapper for it.

⚠ We should decide if we want to invest time also to JSP or area caching for JSP won't be supported.

Investigated Options

A] Caching rendering Listener

Creation of cache items

- A caching listener would:
    - *before* method: Start caching a component or do nothing according to its cache configuration available from its *RenderableDefinitio* (to be provided by MAGNOLIA-3902 - Getting issue details... STATUS)
    - Every cacheable component would have its own *OutputStream.*
    - Everything writed in response output is saved also in:
        - Every seperate component output
            - A] duplicates - every component is saved also in its parent component and again for its grandparent....
            - B] no special logic for putting cached items together needed (just render whole content from the cache if it's in there, don't collect any cached children)
    - *OutputStream* is saved to cache in *after* method and removed from list of components which are in process of caching.
- ⊕ Straighforward, listeners in rendering already implemented, only some cosmetics improvement needed.
- ⊖ Cache requires depedency on rendering.
- ⊖ Not able to cache surrounding page.
- ⊖ Needs to go to RenderingEngine.

Retrieving of cached items

1. A listener checks if a cache item for the current content is cached.

2. If so, writes the output and returns status "skip-rendering".
    - ⛔ Small changes in rendering required:
        - rendering listeners API change: listeners are currently not able to signal anything, there are just void *methods*
        - *RederingEngine.render()* has to react and skip redering.

B] RenderingTree

See also Concept - partial caching ("Current thoughts").

PageCacheFilter(PCF) -> RenderingTreeFilter(RTF) -> RenderingFilter(RF)

Creation of cache items

- First page request:
    1. -> *RTF*: cache item for this URL not found ->
    2. -> RF: create RenderingTree (tree containing all areas, components and plain text fragments).
    3. *RTF* <- save RenderingTree into cache.
- Next page request:
    1. -> *RTF*: cache item found.
    2. Process cached item (rendering tree), go trough root elements of the tree:
        a. If an element is cached, retrieve it from cache.
        b. Render it if its not in cache: call *RenderingEngine* (we would need new *RenderingEngine.renderArea(Node, areaName)* method if we don't want to have any rendering login in *RTF* filter and also to not have to modify selectors for every area).
            i. save element into cache if it's cacheable
            ii. OR save rendering subtree for this element if not
    3. <- PCF

Conclusion:

- ➕ Page around cached components could be cached.
- ⛔ More complex logic needed.

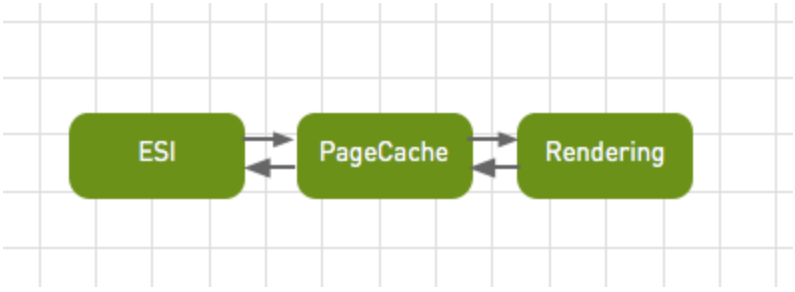
Retrieving of cached items

1. Check if a 'cache' template for the requested content is available.
2. If so, compose requested content:
    a. All subcomponents available from cache: just return composed output.
    b. A subcomponent not in cache: render only this component and put it together with cached components.
    c. A subarea not in cache: render only his area.

- ⛔ A logic for smart composing of components may be hard to implement.

C] SiteMesh filter

SiteMesh is filter which intercepts the request and is able to modify content. The main difference with previous solution would be the order of filters:



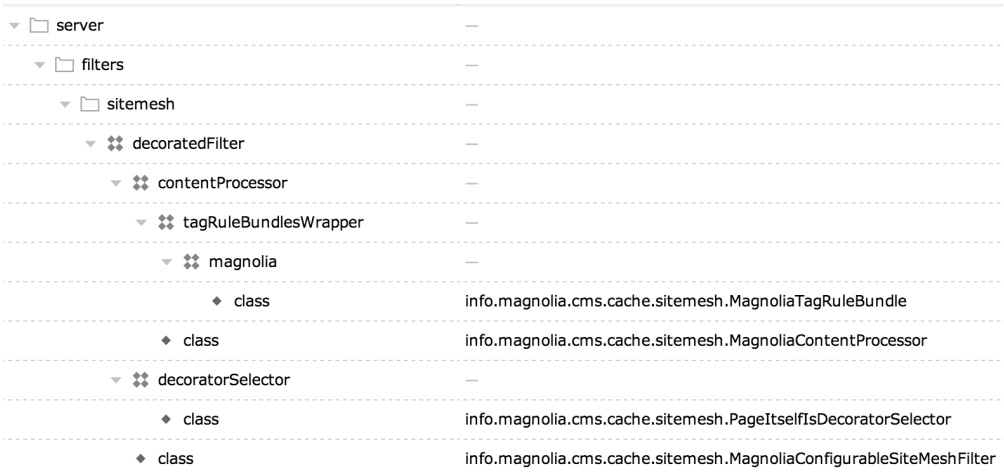
We can still use current page caching mechanism, SiteMesh filter sends request for non cached components and inserts them into the resulting page.

The original purpose of SiteMesh is to merge a page with so called 'decorators' which are a kind of template. SiteMesh extracts content from page and composes it according to decorator. We actually doesn't need a decorator for snippet caching but we can use useful implementations of content insertions, request dispatchers and other stuff.

Latest version is SiteMesh 3.0.0

- was rewritten from SiteMesh2 and promises 3x gain in throughput and half the memory usage.
- configurable by XML, filter properties and of course by code
- extendable
- Apache Software License v2.0
- Frontend Proxy didn't make it into SiteMesh 3.0
  - parallelized loading of sources
  - JCache
  - external request forwarder
- Server side includes not out of the box (vs. SiteMesh2)
- poor documentation

SiteMesh & Magnolia



Page script

```
...
<mgnl:include name="stage" url="/demo-project~mgnlArea=stage~" ttl="0" />
...
```

Filter position

Requirements

1. In front of *CacheFilter* since we need cached pages
   - We can't use gzipped items from cache:
     - **Solution:**
       a. Ungzip and gzip again.
       b. Prevent Gzipping: Signal to GZip/Cache Filter to prevent encoding (we could modify the header of our request wrapper and remove gzip from "Accept-Encoding" - SiteMesh has similar functionallily to filter"If-Modified-Since" header already). Implemented in the prototype
2. In front of *ContentTypeFilter* and following to be able dispatch requests for snippets.
3. Behind *GzipFilter* to be able gzip processed pages coming from SiteMesh. This brings the little problem since *GzipFilter* expect non null content type to be able vote properly.

   - **Solution:**

a. Adjust *ResponseContentTypeVoter* to be able to vote on null content type (use default mime type as *ContentTypeFilter* does).
b. Split *ContentTypeFilter* into two filters since current implementation sets except content type also whole *AggregationState*.

Conclusion

- ⊕ Page around cached components could be cached.
- ⊕ If separate module, cache remains independent from rendering.
- ⊕ Can be used also as original SiteMesh filter.
  - any post-rendering modifications
- ⊖ GZipping needs to be handled.

To be done

- snippet caching module
  - introduce SiteMesh as separate module (concept to module) MSITEMESH-8 - Getting issue details... STATUS
  - inner caching (implement internal cache for snippet discovery)
- ce cache
  - backend engine upgrade/migration
    i. ehcache 2.8
    ii. Hazelcast / JCache
- introduce configuration for snippet caching MSITEMESH-7 - Getting issue details... STATUS
- redering listeners for automatic snippet inclusion MAGNOLIA-6000 - Getting issue details... STATUS
- integration tests MGNLEE-376 - Getting issue details... STATUS

Simple performance tests

Table 1

	Page=cached SiteMesh=false	Page=cached SiteMesh=true Snippets=0	Page=cached SiteMesh=true Snippets: 1cached	Page=cached SiteMesh=true Snippets: 1uncached	Page=uncached SiteMesh=false	Page=uncached SiteMesh=true
TTFB	0.00233	0.00311	0.004119	0.04095	0.04864	0.06377
Total	0.00275	0.00315	0.004142	0.04143	0.06443	0.06418
% TTFB	5	6	8	84	100	131
% Total	4	5	6	64	100	100

TTFB = Time to first byte, Total = time to get whole page, Tested page: demo-project home page, number of requests per value: 100. Note that only our current PageCache was used without any optimisations like caching of pre-parsed pages or any caching mechanism directly inside of SiteMesh filter.

- Column 1: values for our current page cache with SiteMesh filter disabled.
- Column 2: cached page with SiteMesh filter enabled, but no snippets included.
- Column 3: processing of page with one snippet (stage), both page and snippet are cached.
- Column 4: processing of page with one snippet (stage), page cached, the stage has to be rendered.
- Column 5: redering of whole page with SiteMesh disabled (taken as base = 100%).
- Column 6: redering of whole page with SiteMesh enabled.

Questions

- *Part of cache of separate module?*

- If we remain the original purpose of SiteMesh (so it could be used not only for snippet caching), should we move SiteMesh into separate module? Secondly if we implement a rendering listener to automatic snippet tags insertion, we would need to add dependency to rendering. By moving it to a separate module cache remains independent on rendering module.

Questions from architecture meetings

- *Is it possible to use SiteMesh for non HTML / XML pages?*
  - SiteMesh filter processes only text/html by default, but can be configured for other mime-types as well and is able to include snippets if an include tag is present (✅ experimentally checked with CSS).

- *Why the above example uses two tags for snippet insertion? (*obsolete*)*
  - It's only example to demonstrate two rules (tags) process (<mgnl:include> for snippet retrieval, <sitemesh:write> for snippet inclusion). It possible to merge these two into one tag below which also renders area within the page and shows it in the resulting HTML if SiteMesh filter is disabled (✅checked by implementation):

    ```
    <mgnl:include url="/demo-project~mgnlArea=stage~">
            [@cms.area name="stage"/]
    </mgnl:include>
    ```

- *Is it possible to include these tags into HTML comments (<!-- <mgnl:include...) to prevent being rendered when SiteMesh filter is disabled?*
  - The question is if we wanna to do that, because:
    1. We would need to use different comment tags for HTML/CSS/... . Maybe it would be better to just check if SiteMesh filter is disabled and don't include SiteMesh tags if so.
    2. We can't request for direct rendering of an area on the same page if it would be commented.
  - But in case we'll really want to:
    - *org.sitemesh.tagprocessor.TagTokenizer* doesn't search comments for tags, but it would be probably possible to use a custom lexer.flex to do that (❌not checked)
    - OR use conditional comment start tag as workaround and fool SiteMesh (SiteMesh searches for tags inside of these comments)
      - check in non IE browsers ✅
      - check in IE since it's not complete conditional tag ❓

```
<!--[mgnl:include url="/demo-project~mgnlArea=stage~"/><!---->
```

D] Our own ESI filter

Could be build on modify stream module. Although all weaknesses can be implemented by us, it's hard to estimate required time for that.

Comparison with SiteMesh approach

- ➕ No need of 3th party project.
- ➖ Needs to be implemented:
  - snippet request dispatcher
- ➖ Never used in production, no guarantee of stability.
- Modify stream vs. SiteMesh
  - ➖ processing of multiple tags (currently only one init/close tag can be specified)
  - ➖ wide XML/java configuration out of the box (processed mime-types)

Decision

We decided to go with option C] SiteMesh Filter.

Estimate

Minimal implementation

- integrate SiteMesh filter ✅
- move it to separate module, polish the code (2 weeks)
- create samples from SiteMesh and test (1 day)
- introduce snippet caching configuration (2 days)
- intergration tests (2 days)
- This minimal implementation is completely cache-independent, uses only current (Page)CacheFilter

Optimal implementation

- Cache results of parsing for future use (2 days)
- Parellel requests for snippets (2 days)
- Testing/polishing (up to 1 week)

Upgrade to JCache

- split cache to submodules (1 week)
- integrate Hazelcast implementation (2 weeks)

Cache keys

- **Workspace:UUID** as mentioned in Concept - Configurable cache constraints on renderables
  ➖ is probably not enough since a area doesn't have to have its own content node

- **Workspace:UUID:definitionName**
  ➕ ability to store also items which don't exist in JCR
  ➕ simple to compose, *RenderableDefinition* goes into listeners as well
  ➖ not able to cache personalised content

- **More complex cache key,** see Personalization and Cache
  ➕ ability to store personalized items
  ➖ probably too much for start
  ❓ ~~a class for cache key composition as part of *CacheConfiguration*?~~
    - *ContentIdentifier* configurable per *Renderable* (with reasonable default).
    - cross site / per site / unique ?

Snippets in page cache keys

We should be able to store parent page cache key into snippet key or better the other way around: store all snippet keys into page cache key (One tag could be used for multiple 'parent' pages). We should store tag key when adding tag (e.g. in rendering listener which would insert esi/ssi tags instead of a component). Where to store snippet keys?:

a. *AggregationState*
b. *MgnlContext*
c. *?*

We would add snippets key into page cache key on the way back when storing page into the cache so we could delete all related pages when removing a snippet from the cache.

Configurable caching per renderable

- enabled
- TTL
- ?

Open questions

- "A more advanced implementation would probably do something similar to Edge Side Includes or Varnish, i.e still cache the surrounding page." Concept+-+partial+caching
- "We shouldn't introduce cache-specific logic in the rendering module" Configurable cache constraints

- *info.magnolia.module.cache.AbstractListeningFlushPolicy.flushByUUID()* has to flush all subnodes?

Additional Reading

- Cache module documentation
- Cache-your-jee-application
- Varnish
    - http://symfony.com/doc/current/cookbook/cache/varnish.html
    - https://www.varnish-cache.org/trac/wiki/ESIfeatures
    - https://github.com/cadement/AmplifyDemo
    - https://s3-eu-west-1.amazonaws.com/uploads-eu.hipchat.com/20450/94494/XtblGLkrMfnolVf/esi-Amplify-2014.pdf
    - https://www.varnish-cache.org/docs/3.0/tutorial/esi.html
- Sitemesh
    - SiteMesh3
        - http://wiki.sitemesh.org/wiki/display/sitemesh3/SiteMesh+3+Overview
    - SiteMesh2
        - http://pratinas.net/wiki/SiteMesh
        - https://today.java.net/pub/a/today/2004/03/11/sitemesh.html
        - Further+Reading

Git prototypes

- Rendering
- Cache - rendering listener solution
- SiteMesh solution