Concept - Cache Improvements

Your Rating: Results: 102 rates

Draft for 5.2, 5.3

Proposal for cache improvements.

The locking was solved in 4.3, other improvements are pending

- Separate client caching from server-side caching
- Memory consumption optimization
- Cache locking
- Remove boilerplate and hide locking concerns
- Caching other objects

There are 4 main areas to study/develop in order to have a simplified and faster caching module:

1. separate client caching from server-side caching
2. remove byte arrays and use stream to write to e read from cache elements
3. synchronize read / write operations at cache element level, not at global cache level
4. add a global voter

Separate client caching from server-side caching

Split cache filter in two filters (cachable resources are the resources not bypassed):

- Headers filter, with a manager (the simple implementation is an in-memory (concurrenthashmap) table) to
  - store response headers
  - apply max-age and Expires (or no-cache), or whatever else (for example ETags)
  - check request headers in order to send back to client SC_NOT_MODIFIED
- Content filter, with a manager (the simple implementation is a filesystem based manager) to
  - cache resources by streaming response (multiplex streams) to an outputstream taken from cache element
  - check for SC_NOT_MODIFIED using cache element creation date
  - stream from cache using an inputstream

Memory consumption optimization

Optimize memory usage by removing the use of byte arrays both in writing to cache and reading from cache

Cache locking

Use the java.util.concurrent.locks.ReentrantReadWriteLock ReadLock and WriteLock to do per-element resource locking

Remove boilerplate and hide locking concerns

We could think of adding a "`getOrCache`" (unconvincing name to be debated) method on the `Cache` interface whose implementation could look something like the following (not taking any locking/synchronizing issue in account, so this code might not be accurate)

```
Object getOrCache(Object key, Callback c) {
  Object cached = get(key);
  if (cached == null) {
    Object value = generateCacheValue();
    put(key, value);
    return value;
  } else {
    return cached;
  }
}

interface Callback {
  Object generateCacheValue();
}
```

... where the `Callback` interface would thus be responsible for "generating" the cache value; this new method could thus be called as such:

```
cache.getOrCache(key, new Callback() {
  public Object generateCacheValue() {
    return retrieveValueFromSomeRemoteService();
  }
});
```

See this diff (this class) for an example of an implementation.

TBH, I wouldn't be surprised if more recent versions of EhCache and other cache libraries had such a construct natively. Seems clean and elegant enough to be used in many cases. Not sure it would work for our page caching, but most likely for many other situations where we want to cache "stuff" (I'm using this in the external-indexing module, for example)

> edit: looking at the EhCache 2.5 API, it could perhaps indeed be implemented with `Cache.putIfAbsent`, by passing a subclass of `Element` whose `getValue` (or `getObjectValue`, not sure which) would be lazy. It also has a `SelfPopulatingCache` class, which might be interesting looking into.

Additionally, if feasible, in some cases, using generics for key and values in the cache would avoid casting.
(there's possibly going to be an unchecked cast at some point when retrieving the cache instance, but eh)

Caching other objects

See Concept - Cache arbitrary objects.

Global cache voter

MGNLCACHE-37 - Getting issue details... STATUS

I.e.

```
public class AllInOneCacheVoter extends AbstractBoolVoter
{

    private String allowedExtensions;

    private String deniedExtensions;

    private String allowedRequestContentTypes;

    private String deniedRequestContentTypes;

    private String allowedResponseContentTypes;

    private String deniedResponseContentTypes;

    private boolean allowRequestWithParameters = false;

    private boolean allowAdmin = false;

    private boolean allowAuthenticated = false;

    private boolean allowDocroot = true;

    private boolean allowDotResources = true;

    private boolean allowDotMagnolia = false;

    private VoterSet voters = new VoterSet();

    // called by Content2Bean
    public void init()
    {
       if (StringUtils.isNotBlank(allowedExtensions) || StringUtils.isNotBlank(deniedExtensions))
        {
            ExtensionVoter voter = new ExtensionVoter();
            voter.setAllow(allowedExtensions);
            voter.setDeny(deniedExtensions);
            voter.setNot(true);
            voters.addVoter(voter);
        }
       ... create voters and add them to voters
    }

    /**
     * {@inheritDoc}
     */
    @Override
    protected boolean boolVote(Object value)
    {
        return voters.vote(value) == 0;
    }

    ... getters and setters ...
}
```