Concept - Cache arbitrary objects

Draft for 5.3

Allowing usage of cache module for arbitrary objects (not just pages) ~~MGNLCACHE-55~~ - Caching arbitrary objects CLOSED

ARCHI-13 - Jira project doesn't exist or you don't have permission to view it.

- Rationale
- Current situation
- Proposal
- Task-ification and stories
- Further

Rationale

The current cache module is **very** biased towards caching pages (or request/responses in general).

We've already abused it to cache uuid mappings, and it'd be beneficial to do this for other things too (personalization could use it, forum, ...)

With our current API, one can get an arbitrary cache object by name, but that comes with FlushStrategy, etc, which are very likely irrelevant. Besides this unnecessary API leak, it is currently impossible to configurable such an arbitrary cache's underlying cache (i.e, for example, use different sizes and eviction mechanisms) - we can currently only configure the "defaultCacheFactory". (however, one can get an arbitrary named cache from the factory, those will (currently, with ehCache) all share the same configuration (expiration times, eviction strategies etc))

One could **argue** that implementing support for caching arbitrary objects isn't necessary. And indeed, one could easily sneak in calls to, for example, Guava's `Cachebuilder`, or even work around our API, or use EhCache directly. In the first case, we're introducing another cache library (even though we already use Guava), and we're loosing EhCache's persistence capabilities. In the second case, well, we're simply being redundant.

Current situation

The current APIs look like this

Current APIs

```
public class CacheModule implements ModuleLifecycle {

        /* Gets *the* cache factory */
        public CacheFactory getCacheFactory() { .. }

        /* Gets *a* page-cache-config by name */
        public CacheConfiguration getConfiguration(String name) { .. }

}

public class CacheConfiguration {
    CachePolicy cachePolicy;
    FlushPolicy flushPolicy;
    BrowserCachePolicy browserCachePolicy;
}

public interface CacheFactory {
    /**
     * Retrieves a named cache. Implementations should take care of initializing the cache properly
     * on first call or whenever necessary.
     */
    Cache getCache(String name);
    void start();
    void stop();
}
```

The `EhCacheFactory` lets one configure a `defaultCacheConfiguration` (which as instance of `net.sf.ehcache.config.CacheConfiguration` , *not* our (page) `CacheConfiguration`.

Proposal

- Optional: Rename `CacheConfiguration` to `PageCacheConfiguration`? This would make the below much less confusing.
    - this seems to be not-so trivial, after all. The `uuid-key-mapping` cache currently HAS a PageCacheConfiguration entry, for the sole purpose of having a `FlushPolicy`
    - at the same time, this specific cache has leaked all over the place and is managed, for instance, by `i.m.c.cachepolicy.Default`
    - We could look into this and possibly "merge" the two - i.e stop trying to consider `uuid-key-mapping` as "another" cache, and have that be an integral part of the Page Caching
        - Jan and Greg will look into this a bit closer
        - PageCacheConfig maybe needs a "cache name" to know where the **pages** are cached, and one uuid-key-mapping per PCC ?
        - PCC's FlushPolicy would flush both caches
        - Do we still need to register workspaces in FP especially the FlushAllPolicy ? Was using observation really the good way ?
        - Maybe we need to call this MagnoliaContentCacheConfiguration, since it's used for other stuff ? (but really it's a request /response cache)
- Make it possible to have different (ehCache) cache configurations: keep `CacheModule` as-is, so to cache arbitrary objects, one would do `cacheModule.getCacheFactory().getCache("myCache")` ...
    - If someone wants to use a different library for different caches, they could implement a delegating CacheFactory which would dispatch to the appropriate library based on cache name, I suppose.

        - info.magnolia.module.cache.ehcache.EhCacheFactory#start could call `net.sf.ehcache.config.Configuration#addCache` (which adds `net.sf.ehcache.config.CacheConfiguration` instances), like it already does to handle the defaultConfiguration object
            - TBD: the `cfg` object we use in `EhCacheFactory#start` could have perhaps been exposed to node2bean (i.e not directly, but by delegating from the get/set methods of EhCacheFactory) - perhaps this was a c2b limitation of sorts at the time ? Not sure...
            - it appears that Configurations' map of configurations (...) already supports the idea that an entry could be called "default", so perhaps our `defaultConfiguration` node and handling could be merged
            -
- OR, have `CacheModule.getCache(name)` hide the factory - but the factory would still be needed to **create** the cache when needed...
    - this could be implemented as a convenience method on `CacheModule`, and/or hidden from the factory ?

Task-ification and stories

- As a developer, I can cache arbitrary objects, and the APIs are clear enough to let me know that i can.
    - Configured caches should be lazily created (which doesn't seem to be ehcache's default behavior)
        - OR, anyway, we should make sure we still create (all?) caches with the blocking wrapper we currently use (not sure if that should be the default for everything)
- As an administrator, I can fine-tune how each and every cache is configured (amount of cached items, eviction policies, etc) or just use defaults
- Update EHCache version
- Fix-up PageCacheConfiguration misnomer and "hide" uuid-key-mappings into page caching (as opposed to having it be an individual cache that somehow leaked out in many places)
    - Update tasks required (check that uuid-key-mapping PCC is default, remove it if so, or adapt it somehow ?)

Further

This opens up the doors to implement MAGNOLIA-5550 - Allow dedicated caching of areas OPEN , aka partial caching, etc.