Geomajas caching plug-in guide

Geomajas Developers and Geosparc

Geomajas caching plug-in guide

by Geomajas Developers and Geosparc

2.0.0

Table of Contents

List of Figures

List of Examples

Chapter 1. Introduction

The caching plug-in allows you to speed-up the performance of the Geomajas system. By deep integration into the functioning of the system, this can be done in a way which prevents stale data from being presented to the user.

Thanks to the use of pipelines when rendering tiles and doing updates of data, and thanks to the power of the configuration system, Geomajas is very flexible with regards to choices of when and how to apply caching.

The system is sufficiently powerful to provide many use cases, including

- keeping data cached for a pre-configured period

- limiting the size of the cache according to a strategy

- evicting cached data to disk according to a strategy to reduce cache memory consumption

- invalidation of cached data when it overlaps with a certain area, avoiding stale data

- smart invalidation based on an area, only invalidating when objects where the cache may be visible (based on zoom level)

- clustered cache

- ...

1. Overview

The caching is enabled by adding steps and interceptors to the default pipelines at some of the hooks. Two of the example pipelines are the GetVectorTile and SaveOrUpdate pipelines.

The GetVectorTile pipeline can be enhanced to support caching as in figure Figure 1.1, "Cache interceptors in the GetVectorTile pipeline". Interceptors are added to get items from the cache and or store them in the cache. This is specifically the cache for caching the entire tile and for getting the features (strictly speaking feature caching is done in the get features pipeline).

Figure 1.1. Cache interceptors in the GetVectorTile pipeline
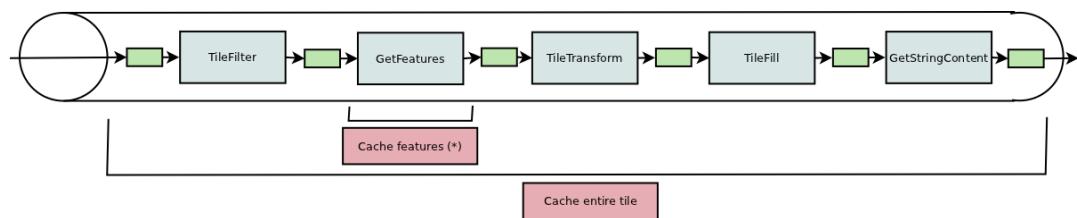
This is a pipeline only reads data and thus only shows how to get data from the cache. In figure Figure 1.2, "Cache hooks in (a part of) the SaveOrUpdate pipeline" below a part of the SaveOrUpdate pipeline is shown. As all changes to data occur through this layer, it is useful to handle invalidation of cached data.

Figure 1.2. Cache hooks in (a part of) the SaveOrUpdate pipeline
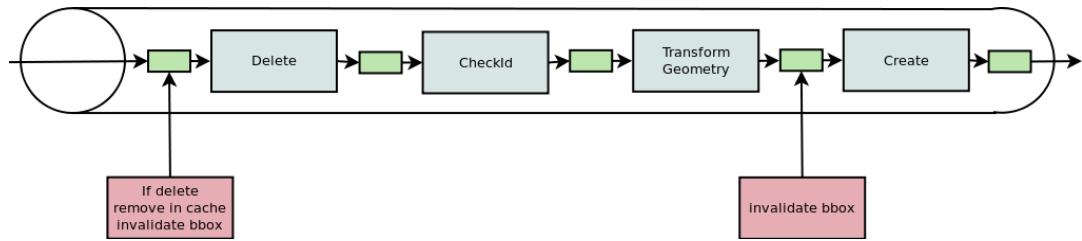
For each category of objects, a different cache configuration can be used. Each combination of layer and category can use a different cache implementation. The type of spatial index can also be chosen per layer and category combination. This is all managed by the CacheManagerService.

Figure 1.3. CacheManager and cache per layer and category
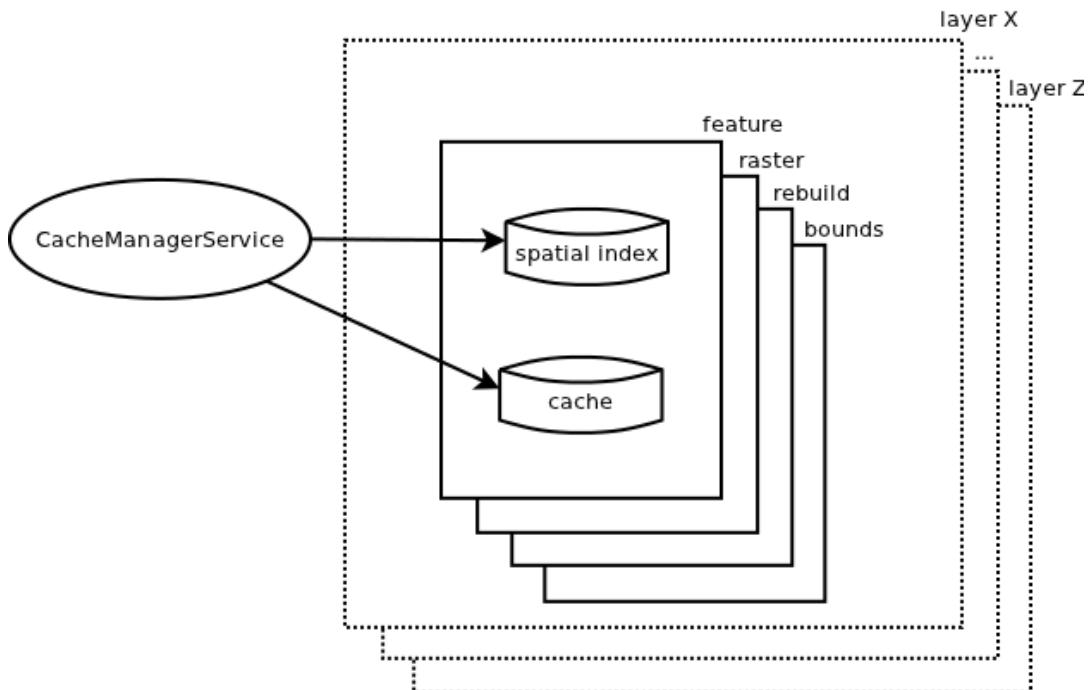
Each cache or index implementation has it's own configuration which may again depend on the layer and category.

The different caches for a layer work together as shown in figure Figure 1.4, "Handling a request for the image part of a rasterized tile". This displays the handling of a request for a raster image, for example the image for a rasterized tile. The request handler first check whether the image is available in the cache. If this is the case, the image is returned to the client. If not, the rebuild cache is used to convert the image name into a the context which is used to (re)start the pipeline to build the raster image. During the execution of the pipeline, the feature cache may be used to avoid queries on the data source for the image.

Figure 1.4. Handling a request for the image part of a rasterized tile
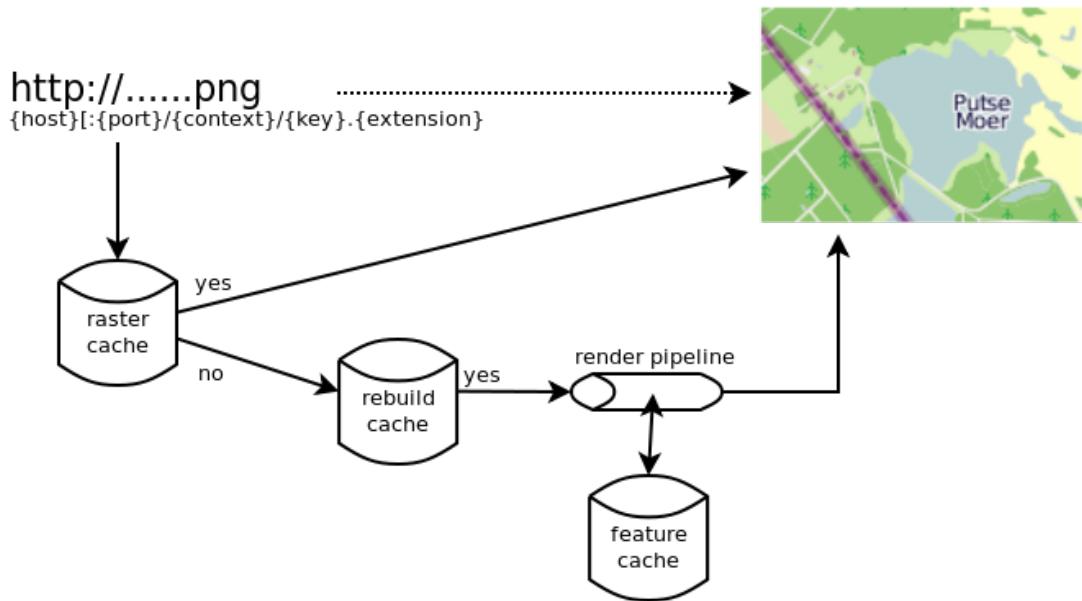
The reason for the use of different caches is the configuration of the cache.

- The raster cache does not need to store much in memory. The data may very quickly be evicted to disk. On disk this cache may be allowed to grow very big (disk space is cheaper than memory). It can be chosen how long data is kept. Removing data from the cache is no problem as the image can be rendered at any time. Keeping data in memory is no real advantage as the only purpose is to stream the data to the network.

- The rebuild cache contains many small objects to allow rebuilding of rendered tiles. The data should not be removed for a reasonably long amount of time (the maximum duration of a session) as the tiles would otherwise be impossible to rebuild. Evicting to disk is no problem, the time required to read the pipeline context from disk is likely to be small compared to the evaluation of the pipeline.

- The feature cache is used to cache data store queries. Evicting to disk is less useful as the original query could also be repeated.

2. Security

The cache also has to consider security. All entries in the cache are stored based on keys. These keys are determined based on the context which is relevant when building the cached object, including the security context. When a key is determined, as the key is determined using a hashing algorithm, the chances of having two different contexts which may to the same key is very small. Still, when putting an object in the cache, if the key was already used in the cache, it is checked that the cache context (including security) matches and if not, the key is modified to make it unique.

The situation as displayed in figure Figure 1.4, "Handling a request for the image part of a rasterized tile" is a little bit different. When requesting (cached) images to be served, they key is used as (part of) the URL. The default behaviour in this case is that the security token is not included in the URL and this the matching of the security context can not be done. This is explicitly done to be able to serve the images faster. However, the URL can be considered secret as they cannot be predicted. As such you will normally see the tile as it was previously calculated. However, it is possible that the rights for a user changes which would result in changed data. It is possible that the cache represents information according to old rights. There are two possible solutions, one is to explicitly invalidate the caches when rights are changed. The other is to limit the life of cache entries (notably including the rebuild cache).

Chapter 2. Configuration

The configuration of the caching plug-in covers many aspects.

- Use pipelines which include caching

- Configure the cache implementations and configurations to use for each layer/category combination

- Configure the spatial index implementation and configuration for each layer/category combination

1. Dependencies

Make sure you include the correct version of the plug-in in your project. Use the following excerpt (with the correct version) in the dependencyManagement section of your project:

```
 <dependency>
     <groupId>org.geomajas.plugin</groupId>
     <artifactId>geomajas-plugin-caching-all</artifactId>
     <version>1.0.0</version>
     <type>pom</type>
     <scope>import</scope>
</dependency>
```

If you are using geomajas-dep, this includes the latest released version of the caching plug-in (at the time of publishing of that version). If you want to overwrite the caching plug-in version, make sure to include this excerpt *before* the geomajas-dep dependency.

You can now include the actual dependency without explicit version.

```
<dependency>
     <groupId>org.geomajas.plugin</groupId>
     <artifactId>geomajas-plugin-caching</artifactId>
</dependency>
```

2. Pipeline configuration

To make sure the caching is actually used, you have to assure that the cached pipelines are used in the cases where you want the cache to be enabled.

The easiest configuration is to enable the caches to be used by default. This can be done by including the file

`classpath:org/geomajas/plugin/caching/DefaultCachedPipelines.xml`

in your web.xml file.

Alternatively you can configure for each layer individually which pipelines should be used.

The following cached pipelines exist (bean names):

- PIPELINE_SAVE_OR_UPDATE_ONE_CACHED: pipeline for the saveOrUpdate() on a vector layer.

- PIPELINE_GET_FEATURES_CACHED: pipeline for getting features from a vector layer.

- PIPELINE_GET_BOUNDS_CACHED: pipeline to determine layer bounds.

- PIPELINE_GET_VECTOR_TILE_CACHED: pipeline to get a tile for a vector layer.

- PIPELINE_GET_VECTOR_TILE_NON_CACHED: pipeline to get a tile for a vector layer without caching, so the original definition before caching.

You can set a specific pipeline using a configuration like this:

Example 2.1. Applying the cached pipeline for getVectorTile() on myLayer

```
<bean class="org.geomajas.service.pipeline.PipelineInfo">
    <property name="pipelineName">
        <util:constant static-field="org.geomajas.service.pipeline.PipelineCode
    </property>
    <property name="layerId" value="myLayer" />
    <property name="delegatePipeline" ref="PIPELINE_GET_VECTOR_TILE_CACHED" />
</bean>
```

3. Cache configuration

Cache configuration can be configured individually for each layer and cache category (which more or less matches type of object). This is done using the CacheServiceInfo object. As you can see in listing Example 2.2, "Example cache configuration" you can define three properties

- *layerId*: the layer for which this configuration applies.

- *category*: the cache category for which this configuration applies. This is always in instance of the CacheCategory class.

- *cacheFactory*: the factory which should be used to create cache instances. This is a bean of type CacheFactory which is responsible for creating cache instances.

Example 2.2. Example cache configuration

```
<bean class="org.geomajas.plugin.caching.service.CacheServiceInfo">
    <property name="layerId" value="test" />
    <property name="category">
        <util:constant static-field="org.geomajas.plugin.caching.service.CacheCa
    </property>
    <property name="cacheFactory">
        <bean class="org.geomajas.plugin.caching.service.DummyCacheFactory">
            <property name="test" value="full" />
        </bean>
    </property>
</bean>
```

The layerId and category properties are not required to be set. This can be used to set default caches to be used. When searching the configuration for a layer/category combination, the following search order is used.

1. matching layerId and matching category

2. matching layerId and null category

3. matching category and null layerId

4. null layerId and null category

The properties are null when no value is specified. If the application context contains more than one bean matching bean (for each of the lines above) the last definition is used.

As an example you could use this mechanism to disable the bounds caching using the following configuration:

Example 2.3. Disable caching for SVG and VML.

```
<bean class="org.geomajas.plugin.caching.service.CacheServiceInfo">
    <property name="category">
        <util:constant static-field="org.geomajas.plugin.caching.service.CacheCa
    </property>
    <property name="cacheFactory">
        <bean class="org.geomajas.plugin.caching.cache.NoCacheCacheFactory" />
    </property>
</bean>
```

The configuration which applies still depends on the CacheFactory which is in use.

3.1. Default caches

The default cache configuration is show in listing Example 2.4, "Default Infinispan configuration". It uses Infinispan for the caching and builds individual caches for each of the categories. It has the following behaviour:

- SVG/VML cache: store 512 object in each.

- bounds cache: store up to 512 bounds.

- feature cache: store up to 512 query results.

- raster cache: store 64 items in memory based on a LRU (least recently used) algorithm. Additional items are passivated to disk. Items stay cached for a week after last use.

- rebuild cache: store up to 2048 items in memory based on a LRU (least recently used) algorithm. Additional items are passivated to disk. Items stay cached for 48 hours after last use.

Example 2.4. Default Infinispan configuration

```
<bean class="org.geomajas.plugin.caching.service.CacheServiceInfo">
    <property name="cacheFactory">
        <bean class="org.geomajas.plugin.caching.infinispan.cache.Infinispa
            <property name="defaultConfiguration" ref="defaultInfinispanCac
        </bean>
    </property>
</bean>

<bean name="defaultInfinispanCacheConfig" class="org.geomajas.plugin.caching
    <property name="configuration">
        <map>
            <entry>
                <key><util:constant static-field="org.geomajas.plugin.cachin
                    <bean class="org.geomajas.plugin.caching.infinispan.con
                        <property name="maxEntries" value="512" />
                    </bean>
            </entry>
            <entry>
                <key><util:constant static-field="org.geomajas.plugin.cachin
                    <bean class="org.geomajas.plugin.caching.infinispan.con
                        <property name="maxEntries" value="512" />
                    </bean>
            </entry>
            <entry>
                <key><util:constant static-field="org.geomajas.plugin.cachin
                    <bean class="org.geomajas.plugin.caching.infinispan.con
                        <property name="maxEntries" value="2048" />
                    </bean>
            </entry>
            <entry>
                <key><util:constant static-field="org.geomajas.plugin.cachin
                    <bean class="org.geomajas.plugin.caching.infinispan.con
                        <property name="maxEntries" value="64" />
                        <property name="evictionStrategy">
                            <util:constant static-field="org.infinispan.evi
                        </property>
                        <property name="expiration" value="10080" /> <!-- 6
                        <property name="level2CacheLocation" value="${geoma
                    </bean>
            </entry>
            <entry>
                <key><util:constant static-field="org.geomajas.plugin.cachin
                    <bean class="org.geomajas.plugin.caching.infinispan.con
                        <property name="maxEntries" value="2048" />
                        <property name="evictionStrategy">
                            <util:constant static-field="org.infinispan.evi
                        </property>
                        <property name="expiration" value="2880" /> <!-- 60
                        <property name="level2CacheLocation" value="${geoma
                    </bean>
            </entry>
        </map>
    </property>
</bean>
```

You can configure the directory where the passivated cache objects are stored using the geomajasCacheLocation property. This could for example be done by passing something like -DgeomajasCacheLocation=/var/geomajas/cache as startup parameter to your web container or application server.

When the variable is not set, the ${geomajasCacheLocation}will be interpreted as ${java.io.tmpdir}/ geomajas/cache. So the files will be placed in your systems temporary directory. Note that this can cause conflicts when starting more than one Geomajas instance on a system.

3.2. Infinispan caches

One of the caching services which is provided out of the box uses Infinispan as the caching library.

When using the Infinispan cache factory, the factory itself contains exactly one Infinispan CacheManager. Each manager can itself contain multiple caches. You can configure a default set of caches (one for each category) using a CacheInfo bean. Additional sets of caches can also be defined using further CacheInfo beans which are then referred to in the layer configuration.

To use Infinispan for the caches, you have to set the Infinispan cache factory. As mentioned earlier, this can be done specifically per layer and/or cache category. The example below shows a configuration using an Infinispan configuration file (infinispanConfiguration.xml) and a configuration which is used when no layer specific configuration exists (in this case, the actual default configuration).

Example 2.5. Configure Infinispan cache factory

```
<bean class="org.geomajas.plugin.caching.service.CacheServiceInfo">
    <property name="cacheFactory">
        <bean class="org.geomajas.plugin.caching.infinispan.cache.InfinispanCac
            <property name="configurationFile" value="infinispanConfiguration.x
            <property name="defaultConfiguration" ref="defaultInfinispanCacheCo
        </bean>
    </property>
</bean>
```

The Infinispan configuration file is searched on the class path (and if not found there, on the file system). This file can contain both a default configuration and named configurations. To use a named configuration, use a configuration as mentioned below. In all other cases, the default configuration is extended with the additional settings provided by you.

Example 2.6. Choose Infinispan configuration by name

```
<entry>
    <key><util:constant static-field="org.geomajas.plugin.caching.service.Cache
        <bean class="org.geomajas.plugin.caching.infinispan.configuration.Simpl
            <property name="configurationName" value="test" />
        </bean>
</entry>
```

In the CacheInfo beans contains a cache configuration per category. For straightforward configurations, you can use the SimpleInfinispanCacheInfo bean. This allows you to set the following cache parameters:

- maxEntries: maximum number of entries which are allowed in the cache. This should be a power of two (it is rounded up to the next power of two if it isn't).

- evictionStrategy: the eviction strategy being the algorithm used to remove items from the cache memory.

- evictionWakeUpInterval: interval in milliseconds to invoke the eviction thread. Use -1 if you don't want the eviction thread to be used.

- expiration: period (in minutes) that a cache entry is considered valid since last use. When a cache entry has not been used longer, it will be removed from the cache. This applies to both in-memory items and passivated items and is the way to limit the amount of disk space used for the second level cache. Using -1 means that items never expire.

- isolationLevel: transaction isolation level for this cache.

- level2CacheLocation: location on disk for the second level cache. Should point to a directory. The directory is created if it did not exist. The location can start with a property reference. If a property reference is used but the property does not exist, the temporary directory followed by "/geomajas/cache" is used as property value.

More details about the behaviour of these settings can be found in the Infinispan documentation.

Example 2.7. Simplified Infinispan configuration

```
<entry>
    <key><util:constant static-field="org.geomajas.plugin.caching.service.Cache
        <bean class="org.geomajas.plugin.caching.infinispan.configuration.Simpl
            <property name="maxEntries" value="512" />
            <property name="evictionStrategy"><util:constant static-field="org.
            <property name="evictionWakeUpInterval" value="5000" />
            <property name="expiration" value="600" />
            <property name="isolationLevel"><util:constant static-field="org.in
            <property name="level2CacheLocation" value="${geomajasCacheLocation
        </bean>
</entry>
```

You can explicitly turn off caching for a category as in the example below. Alternatively, not mentioning a category in the configuration will also switch off caching for that category.

Example 2.8. Switch off caching

```
<entry>
    <key><util:constant static-field="org.geomajas.plugin.caching.service.Cache
        <bean class="org.geomajas.plugin.caching.infinispan.configuration.Simpl
            <property name="cacheEnabled" value="false" />
        </bean>
</entry>
```

You can also determine at layer level to use a specific set of caches for each category.

This can be configured inside the LayerInfo object as shown in the example below.

Example 2.9. Add cache configuration for a layer

```
<bean name="infiniLayerInfo" class="org.geomajas.configuration.VectorLayerInfo":
    <property name="extraInfo">
        <map>
            <entry>
                <key><util:constant static-field="org.geomajas.plugin.caching.co
                <ref bean="infiniLayerCacheConfig" />
            </entry>
        </map>
    </property>
</bean>
```

4. Spatial index configuration

The spatial index to be used while caching can be configured individually for each layer and cache category. This is done using `CacheIndexInfo` objects. As you can see in listing Example 2.10, "Example spatial index configuration" you can define three properties

- *layerId*: the layer for which this configuration applies.

- *category*: the cache category for which this configuration applies. This is always in instance of the `CacheCategory` class.

- *cacheIndexFactory*: the factory which should be used to create spatial index instances. This is a bean of type CacheIndexFactory which is responsible for creating cache instances.

Example 2.10. Example spatial index configuration

```
<bean class="org.geomajas.plugin.caching.service.CacheIndexInfo">
    <property name="layerId" value="test" />
    <property name="category">
        <util:constant static-field="org.geomajas.plugin.caching.service.CacheC
    </property>
    <property name="cacheIndexFactory">
        <bean class="org.geomajas.plugin.caching.service.DummyCacheIndexFactory
            <property name="test" value="full" />
        </bean>
    </property>
</bean>
```

The layerId and category properties are not required to be set. This can be used to set default caches to be used. When searching the configuration for a layer/category combination, the following search order is used.

1. matching layerId and matching category

2. matching layerId and null category

3. matching category and null layerId

4. null layerId and null category

The properties are null when no value is specified. If the application context contains more than one bean matching bean (for each of the lines above) the last definition is used.

4.1. Default spatial indices

??? @TODO

Chapter 3. How-to