Ehcache Web Cache User Guide

Version 2.10.1

October 2015

EHCACHE

Document ID: EHC-WCUG-2101-20151015

Table of Contents

1 About the Web Cache Module

Introduction

Ehcache provides a set of general purpose web caching filters in the `ehcache-web` module. Using these can make an amazing difference to web application performance. A typical server can deliver 5000+ pages per second from the page cache. With built-in gzipping, storage and network transmission is highly efficient. Cache pages and fragments make excellent candidates for `DiskStore` storage, because the object graphs are simple and the largest part is already a `byte[]`.

Components and Concepts

SimplePageCachingFilter

The SimplePageCachingFilter is a simple caching filter suitable for caching compressible HTTP responses such as HTML, XML or JSON. It uses a singleton CacheManager created with the default factory method. Override to use a different CacheManager.

It is suitable for:

- complete responses (i.e. not fragments).
- A content type suitable for gzipping. For example, text or text/html.

For fragments see "SimplePageFragmentCachingFilter" on page 8.

Keys

Pages are cached based on their key. The key for this cache is the URI followed by the query string. An example is `/admin/SomePage.jsp?id=1234&name=Beagle`. This key technique is suitable for a wide range of uses. It is independent of host name and port number, so will work well in situations where there are multiple domains which get the same content, or where users access based on different port numbers. A problem can occur with tracking software, where unique ids are inserted into request query strings. Because each request generates a unique key, there will never be a cache hit. For these situations it is better to parse the request parameters and override `calculateKey(javax.servlet.http.HttpServletRequest)` with an implementation that takes account of only the significant ones.

Configuring the Cache Name

A cache entry in ehcache.xml should be configured with the name of the filter. Names can be set using the init-param `cacheName`, or by sub-classing this class and overriding the name.

Concurrent Cache Misses

A cache miss will cause the filter chain, upstream of the caching filter to be processed. To avoid threads requesting the same key to do useless duplicate work, these threads block behind the first thread. The thread timeout can be set to fail after a certain wait by setting the init-param `blockingTimeoutMillis`. By default threads wait indefinitely. In the event upstream processing never returns, eventually the web server may get overwhelmed with connections it has not responded to. By setting a timeout, the waiting threads will only block for the set time, and then throw a net.sf.ehcache.constructs.blocking.LockTimeoutException. Under either scenario an upstream failure will still cause a failure.

Gzipping

Significant network efficiencies, and page loading speedups, can be gained by gzipping responses. Whether a response can be gzipped depends on:

- Whether the user agent can accept GZIP encoding. This feature is part of HTTP1.1.

  If a browser accepts GZIP encoding it will advertise this by including in its HTTP header: All common browsers except IE 5.2 on Macintosh are capable of accepting gzip encoding. Most search engine robots do not accept gzip encoding.

- Whether the user agent has advertised its acceptance of gzip encoding. This is on a per request basis. If they will accept a gzip response to their request they must include the following in the HTTP request header:

  ```
  Accept-Encoding: gzip
  ```

Responses are automatically gzipped and stored that way in the cache. For requests which do not accept gzip encoding the page is retrieved from the cache, ungzipped and returned to the user agent. The ungzipping is high performance.

Caching Headers

The `SimpleCachingHeadersPageCachingFilter` extends `SimplePageCachingFilter` to provide the HTTP cache headers: ETag, Last-Modified and Expires. It supports conditional GET. Because browsers and other HTTP clients have the expiry information returned in the response headers, they do not even need to request the page again. Even once the local browser copy has expired, the browser will do a conditional GET. So why would you ever want to use SimplePageCachingFilter, which does not set these headers?

The answer is that in some caching scenarios you may wish to remove a page before its natural expiry. Consider a scenario where a web page shows dynamic data. Under Ehcache the Element can be removed at any time. However if a browser is holding expiry information, those browsers will have to wait until the expiry time before getting updated. The caching in this scenario is more about defraying server load rather than minimizing browser calls.

Init-Params

The following init-params are supported:

- `cacheName` - the name in ehcache.xml used by the filter.
- `blockingTimeoutMillis` - the time, in milliseconds, to wait for the filter chain to return with a response on a cache miss. This is useful to fail fast in the event of an infrastructure failure.

- `varyHeader` - set to true to set Vary:Accept-Encoding in the response when doing Gzip. This header is needed to support HTTP proxies however it is off by default.

```
<init-param>
  <param-name>varyHeader</param-name>
  <param-value>true</param-value>
</init-param>
```

Re-entrance

Care should be taken not to define a filter chain such that the same `CachingFilter` class is reentered. The `CachingFilter` uses the `BlockingCache`. It blocks until the thread which did a get which results in a null does a put. If reentry happens a second get happens before the first put. The second get could wait indefinitely. This situation is monitored and if it happens, an IllegalStateException will be thrown.

SimplePageFragmentCachingFilter

The SimplePageFragmentCachingFilter does everything that SimplePageCachingFilter does, except it never gzips, so the fragments can be combined. There is a variant of this filter which sets browser caching headers, because that is only applicable to the entire page.

2 Caching Web Pages and Page Fragments

Sample Use Case

The following describes an example of how to use the included filters to cache web pages and web page fragments.

Problem

You'd like to improve the time it takes to return a page from your web application. Many of the pages in your application are not time- or user-specific and can be cached for a period of time.

Solution

Cache the entirety of the web page, or a fragment of the web page for a period of time. Rather than having to generate the page on each page hit, it will be served out of the cache.

Modern application hardware is able to serve as many as 5,000 pages per second, affording a significant speedup in your application for pages that are frequently read but infrequently change.

Steps for Caching Web Pages

Caching web pages does not require any code changes. Your application server should support servlet filtering already. Simply update your web.xml file, re-deploy. and you should see the speedup right away.

The basic steps you'll need to follow to configure Ehcache for web-page caching are as follows (these steps assume you already have Ehcache installed in your application):

1. Configure a servlet page filter in web.xml.
2. Configure an appropriate cache in ehcache.xml.
3. Start (or re-start) your application.

The following settings should help you setup web caching for your application.

Step 1 - Add a filter to your web.xml

The first thing you'll need to do is add a filter to enable page caching.

The following web.xml settings will enable a servlet filter for page caching:

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
     http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd "
     version="2.5">
  <filter>
    <filter-name>SimplePageCachingFilter</filter-name>
    <filter-class>net.sf.ehcache.constructs.web.filter.SimplePageCachingFilter
    </filter-class>
  </filter>
  <!-- This is a filter chain. They are executed in the order below.
  Do not change the order. -->
  <filter-mapping>
    <filter-name>SimplePageCachingFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
</web-app>
```

Step 2 - Configure an ehcache.xml

The second step to enabling web page caching is to configure Ehcache with an appropriate ehcache.xml.

The following ehcache.xml file should configure a reasonable default cache:

```
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="../../main/config/ehcache.xsd">
    <cache name="SimplePageCachingFilter"
           maxEntriesLocalHeap="10000"
           maxEntriesLocalDisk="1000"
           eternal="false"
           overflowToDisk="true"
           timeToIdleSeconds="300"
           timeToLiveSeconds="600"
           memoryStoreEvictionPolicy="LFU"
            />
</ehcache>
```

Step 3 - Start your application server

Now start your application server. Pages should be cached.

3 Sample web.xml Configuration

Example web.xml Configuration

Following is a sample web.xml configuration file.

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
version="2.5">
 <filter>
<filter-name>CachePage1CachingFilter</filter-name>
<filter-class>net.sf.ehcache.constructs.web.filter.SimplePageCachingFilter
</filter-class>
<init-param>
 <param-name>suppressStackTrace</param-name>
 <param-value>false</param-value>
</init-param>
<init-param>
 <param-name>cacheName</param-name>
 <param-value>CachePage1CachingFilter</param-value>
</init-param>
 </filter>
 <filter>
<filter-name>SimplePageFragmentCachingFilter</filter-name>
<filter-class>
net.sf.ehcache.constructs.web.filter.SimplePageFragmentCachingFilter
</filter-class>
<init-param>
 <param-name>suppressStackTrace</param-name>
 <param-value>false</param-value>
</init-param>
<init-param>
 <param-name>cacheName</param-name>
 <param-value>SimplePageFragmentCachingFilter</param-value>
</init-param>
 </filter>
 <filter>
<filter-name>SimpleCachingHeadersPageCachingFilter</filter-name>
<filter-class>
net.sf.ehcache.constructs.web.filter.SimpleCachingHeadersPageCachingFilter
</filter-class>
<init-param>
 <param-name>suppressStackTrace</param-name>
 <param-value>false</param-value>
</init-param>
<init-param>
 <param-name>cacheName</param-name>
 <param-value>CachedPage2Cache</param-value>
</init-param>
 </filter>
 <!-- This is a filter chain. They are executed in the order below.
      Do not change the order. -->
 <filter-mapping>
<filter-name>CachePage1CachingFilter</filter-name>
<url-pattern>/CachedPage.jsp</url-pattern>
<dispatcher>REQUEST</dispatcher>
<dispatcher>INCLUDE</dispatcher>
<dispatcher>FORWARD</dispatcher>
 </filter-mapping>
 <filter-mapping>
<filter-name>SimplePageFragmentCachingFilter</filter-name>
```

```
<url-pattern>/include/Footer.jsp</url-pattern>
 </filter-mapping>
 <filter-mapping>
<filter-name>SimplePageFragmentCachingFilter</filter-name>
<url-pattern>/fragment/CachedFragment.jsp</url-pattern>
 </filter-mapping>
 <filter-mapping>
<filter-name>SimpleCachingHeadersPageCachingFilter</filter-name>
<url-pattern>/CachedPage2.jsp</url-pattern>
 </filter-mapping>
```

An ehcache.xml configuration file, matching the above, would then be:

```
<Ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../main/config/ehcache.xsd">
<diskStore path="auto/default/path"/>
 <defaultCache
   maxEntriesLocalHeap="10"
   eternal="false"
   timeToIdleSeconds="5"
   timeToLiveSeconds="10">
   <persistence strategy="localTempSwap"/>
   />
  <!-- Page and Page Fragment Caches -->
<cache name="CachePage1CachingFilter"
  maxEntriesLocalHeap="10"
  eternal="false"
  timeToIdleSeconds="10000"
  timeToLiveSeconds="10000">
  <persistence strategy="localTempSwap"/>
</cache>
<cache name="CachedPage2Cache"
  maxEntriesLocalHeap="10"
  eternal="false"
  timeToLiveSeconds="3600">
  <persistence strategy="localTempSwap"/>
</cache>
<cache name="SimplePageFragmentCachingFilter"
  maxEntriesLocalHeap="10"
  eternal="false"
  timeToIdleSeconds="10000"
  timeToLiveSeconds="10000">
  <persistence strategy="localTempSwap"/>
</cache>
<cache name="SimpleCachingHeadersTimeoutPageCachingFilter"
  maxEntriesLocalHeap="10"
  eternal="false"
  timeToIdleSeconds="10000"
  timeToLiveSeconds="10000">
  <persistence strategy="localTempSwap"/>
</cache>
</ehcache>
```

4 CachingFilter Exceptions

CachingFilter Exceptions

Additional exception types have been added to the Caching Filter.

FilterNonReentrantException

Thrown when it is detected that a caching filter's doFilter method is reentered by the same thread. Reentrant calls will block indefinitely because the first request has not yet unblocked the cache.

ResponseHeadersNotModifiableException

Same as "FilterNonReentrantException" on page .

AlreadyGzippedException

This exception is thrown when a gzip is attempted on already gzipped content.

The web package performs gzipping operations. One cause of problems on web browsers is getting content that is double or triple gzipped. They will either get unreadable content or a blank page.

ResponseHeadersNotModifiableException

A gzip encoding header needs to be added for gzipped content. The `HttpServletResponse#setHeader()` method is used for that purpose. If the header had already been set, the new value normally overwrites the previous one. In some cases according to the servlet specification, setHeader silently fails. Two scenarios where this happens are:

- The response is committed.
- `RequestDispatcher#include` method caused the request.