ControllingModCache

How to control the caching of pages by mod_cache via HTTP headers

Apache httpd mod_cache is a wonderful tool to dramatically enhance the performance of your Cocoon-based site.

For this we assume that httpd is configured to act as a reverse proxy, with mod_cache enabled (see ApacheModProxy).

This page explains how to generate the required HTTP headers for mod_cache to store the pages in its front-end cache, with some example code.

Mod_cache uses the standard HTTP protocol definitions to determine if and how long to cache pages, see the official mod_cache and HTTP protocol specs for more information.

HTTP headers

For mod_cache to put a page in cache, Cocoon must generate the following HTTP headers:

- Last-Modified
- Expires
- Cache-Control
- Content-length

Except for Content-length, these must be generated by your Cocoon application, for example using an Action as shown below.

To generate Content-length, you must currently create your own serializer, to set the buffering flag, as it is not yet configurable.

Simply inherit from HTMLSerializer (for example) and add the following method:
{{{public boolean shouldSetContentLength() {
return true;
}
}}}

Invalidating the cache

According to the HTTP caching specs, a POST on an URL must cause the cache to invalidate the page. This seems to work well with mod_cache, but I haven't used this function extensively.

Example code

Here's some java code to set the required HTTP headers (except Content-Length, see above):

```
    public static final String LAST_MOD_HEADER = "Last-Modified";
    public static final String EXPIRES_HEADER = "Expires";
    public static final String CACHE_CONTROL_HEADER = "Cache-Control";

    /** set headers so that the response is cached for nSeconds
     *  @param lastModified if null, now - 2 seconds is used */
    public void setCacheHeaders(Response resp,Date lastModified,int cacheForHowMaySeconds) {

        final long lastModTime = (lastModified == null ? System.currentTimeMillis() - 2000L : lastModified.
getTime());
        final long expires = System.currentTimeMillis() + (cacheForHowMaySeconds * 1000L);

        resp.addDateHeader(LAST_MOD_HEADER,lastModTime);
        resp.addDateHeader(EXPIRES_HEADER,expires);
        resp.addHeader(CACHE_CONTROL_HEADER,"max-age="+ cacheForHowMaySeconds);
    }
```

And here's code for a Cocoon Action through a helper which contains the above code:

```
    public Map act(Redirector redirector, SourceResolver resolver, Map objectModel, String source, Parameters
parameters)
            throws Exception {
        // parameters tell us how long to cache
        final int nSec = parameters.getParameterAsInteger("cache-validity-seconds",0);
        final String cacheInfo = parameters.getParameter("cache-info","NO-CACHE-INFO");
        final String pageInfo = parameters.getParameter("page-info","NO-PAGE-INFO");

        final HttpCacheHeadersHelper helper = new HttpCacheHeadersHelper(cacheInfo,getLogger());
        helper.setCacheHeaders(ObjectModelHelper.getResponse(objectModel),null,nSec,pageInfo);

        // don't execute what's inside this action, it's just here to set headers
        return null;
    }
```

Then, you only need to use the Action like this in a sitemap to setup page caching:
{{{<map:act type="http-cache-headers">
<map:parameter name="cache-validity-seconds" value="30"/>
<map:parameter name="cache-info" value="cache-name-for-logging"/>
<map:parameter name="page-info" value="page-info-for-logging"/>
</map:act>
}}}

Setting the Content-Length

Unfortunately the "set content length" option of most serializers is not configurable, in some cases you'll need to extend the appropriate Serializer class just to set the appropriate flag. This Needs Improvement (tm).

Here's an example for the HTMLSerializer class:

```
package yourpackage;

import org.apache.cocoon.serialization.HTMLSerializer;

public class BufferingHtmlSerializer extends HTMLSerializer {
    public boolean shouldSetContentLength() {
        return true;
    }
}
```

How to control mod_cache w/o touching Cocoon?

It is easy to create two virtual hosts in apache configuration.

```
<VirtualHost 127.0.0.1>
  ServerName localhost
  <Location /images>
    ExpiresActive On
    ExpiresDefault A3600
  </Location>
  <Location /user>
    ExpiresActive On
    ExpiresDefault A0
  </Location>
  ProxyPass / http://localhost:8080
  ProxyPassReverse / http://localhost:8080
</VirtualHost>
```

```
<VirtualHost 123.45.67.89>
  ServerName yourdomain.com
  CacheEnable disk /images
  CacheRoot /var/www/cache
  ProxyPass / http://localhost
  ProxyPassReverse / http://localhost
</VirtualHost>
```

Update regarding mod_cache under Apache httpd 2.2.x

The mod_cache bundled with versions 2.0.x of Apache httpd does not work fine with the "Vary:" header : when you set this header, mod_cache will detect it but will simply regenerate over and over again the cached content, resulting in zero cache efficiency.

The bug has been corrected in branch 2.2 of Apache, with version 2.2.0 being labeled as the new stable one (as of January 2006). However, a severe regression bug was also introduced when using mod_cache in connection with mod_proxy ! The bugzilla entry is at http://issues.apache.org/bugzilla/show_bug.cgi?id=38017 and the 2.2.0 patch is available at http://issues.apache.org/bugzilla/attachment.cgi?id=17342

So if you want to cache content responses based on http headers variations thanks to "Vary:", you have to run Cocoon under Apache 2.2 with this bugzilla patch. Happy caching 🙂