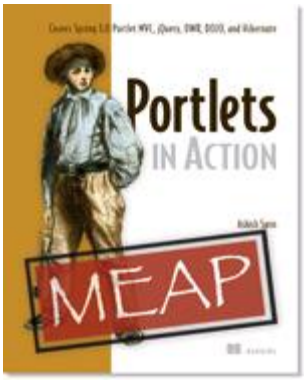
MANNING PUBLICATIONS

Caching generated content

Article from



Portlets in Action
EARLY ACCESS EDITION

Ashish Sarin
MEAP Release: October 2009
Softbound print: Fall 2010 | 550 pages
ISBN: 9781935182542

This article is taken from the book Portlets in Action. *The author discusses expiration and validation-based caching strategies for content generated by a portlet.*

Content generated by a portlet may be cached by the portlet container to allow faster access to the portlet content by the web portal. The content is generated by the portlet when it services a render request. In most cases, portlets retrieve the data from one or more databases, web services, or other data sources and then display it using JSPs (or any other view technology). In cases when a portlet delegates the content generation responsibility to a servlet, the servlet is responsible for retrieving the data from data sources. In any case, the portlet or the servlet component is responsible for retrieving the data every time a render request is sent to the portlet, resulting in the increased load time of the portal page. One way to reduce this performance overhead is by caching the data using caching libraries like `ehcache`.

When using *data caching* to improve portal performance, the portlet retrieves the data from the cache instead of firing a query against the database or invoking web services. Data caching improves performance but it may still take substantial time because the data needs to be obtained from the cache, transformed, and ultimately displayed by a portlet. If request processing is computationally intensive or resource intensive, you may still not achieve the desired performance benchmarks for your web portal.

The data caching strategy can have a substantial negative impact on performance if there are many portlets on your portal page (which is usually the case in most web portals) and each portlet uses the data cache to obtain the data every time a render request is received by the portlet. For instance, in a typical portal, you'll normally have five to 10 portlets on a portal page. Each portlet may be getting its data from a data source. Let's consider a portal page with five portlets: A, B, C, D, and E. If a user sends an action request to portlet A, while rendering the portal page the render methods of portlets B, C, D, and E are also invoked. Even if each portlet had its data cached, the cumulative effect of rendering five portlets can have a substantial effect on the portal's performance.

NOTE

Portlets that show real-time data should get data from the data sources every time render a request is received by the portlet container. It would be a bad design to use data caching in such scenarios unless the data cache is *actively* synched with the data source holding the real-time data.

If you are convinced that data caching is not good enough to boost your portal performance, you can use *content caching*. Content caching refers to the caching of the *markup* generated by the portlet. If the markup is cached, your portlet doesn't even need to hit the data cache to generate content. Portlet containers are responsible for caching the markup generated by the portlet and sending this markup to the portal server when the render request is received for the portlet.

A portlet is responsible for specifying whether it wants to use the content caching feature or not. You can't specify at portlet application level that the content of all the portlets must be cached by the portlet container.

NOTE

Portlet containers cache *portlet content* and not the portal's *page content*.

You may have multiple portlets on your portal page, with some portlets showing fresh content and some cached content. Portlets that are intended to show real-time data should not use the content caching feature and generate content every time they receive a render request.

A portlet may also specify whether its cached content can be shared by the portlet container with different users of the portal or the content should be cached on per user basis. If a portlet shows content that is not specific to a user, the portlet's cached content can be shared with different users of the portal. For instance, if a portlet displays announcements, such content is not user-specific and it can be shared with different users.

Portlet specification defines two types of content caching strategies: *expiration-based* and *validation-based* caching strategies.

Expiration-based caching strategy

Expiration-based caching strategy refers to the caching strategy in which the cached content of the portlet is valid only for a *predefined* duration of time. Once that duration expires, the portlet is asked to generate fresh content, which is again cached by the portlet container for a predefined duration of time.

NOTE

Expiration-based caching strategy must not be used in portlets that show content-based on real-time data. It is ideally suited for portlets that generate content based on computationally intensive or resource intensive request-processing operation.

As long as the portlet content is cached, the portlet container doesn't send a render request to the portlet (this is not always the case, as discussed later in this section); that is, the portlet container *doesn't* invoke the render method of the portlet. The portlet container returns the cached content when it receives the render request for the portlet. After the content expires, if the portlet container receives the render request for the portlet, it invokes the portlet's render method. If, at any time, the portlet container receives an *action* or *event* request for the portlet, the portlet container immediately expires the cached content for the portlet and sends the action or event request to the portlet instance.

CODE REFERENCE

At this time, I recommend that you import ch3_BookCatalog eclipse project from http://portletsinaction.googlecode.com/svn/trunk/ into your eclipse workspace to understand the code references in the rest of this article.

You can specify expiration caching in the portlet deployment descriptor or set it programmatically. The following XML fragment shows how the Book Catalog portlet from our example defines expiration caching in `portlet.xml`:

```
<portlet>
   ....
   <expiration-cache>60</expiration-cache>
   <cache-scope>private</cache-scope>
   <supports>
    ....
   </supports>
   ....
</portlet>
```

Where the `expiration-cache` element defines the duration (in seconds) in which the portlet content is cached by the portlet container. The value `60` indicates that the Book Catalog portlet defines 30 seconds as its cache expiration time.

`cache-scope` defines the scope of a cache; that is, it's shared across portal users (public) or it's user specific (private). The value `private` indicates that the Book Catalog portlet's cache is user specific and not shared across different users of the portal.

You can also set the expiration caching time and scope programmatically in one of the following ways:

- Use the `setExpirationTime` and `setScope` methods of the `CacheControl` object (obtained from `MimeResponse`).
- Set the `EXPIRATION_CACHE` and `CACHE_SCOPE` properties (constants defined in `MimeResponse`) in `RenderResponse`. The `MimeResponse` defines `PRIVATE_SCOPE` and `PUBLIC_SCOPE` constants, which can be specified as the value of the `CACHE_SCOPE` property.
- You can override the expiration cache settings in the portlet deployment descriptor by setting the expiration cache information programmatically, as defined above.

Portlet behavior

Figure 1 highlights some important elements of the Book Catalog portlet that we'll be referring to in this article.

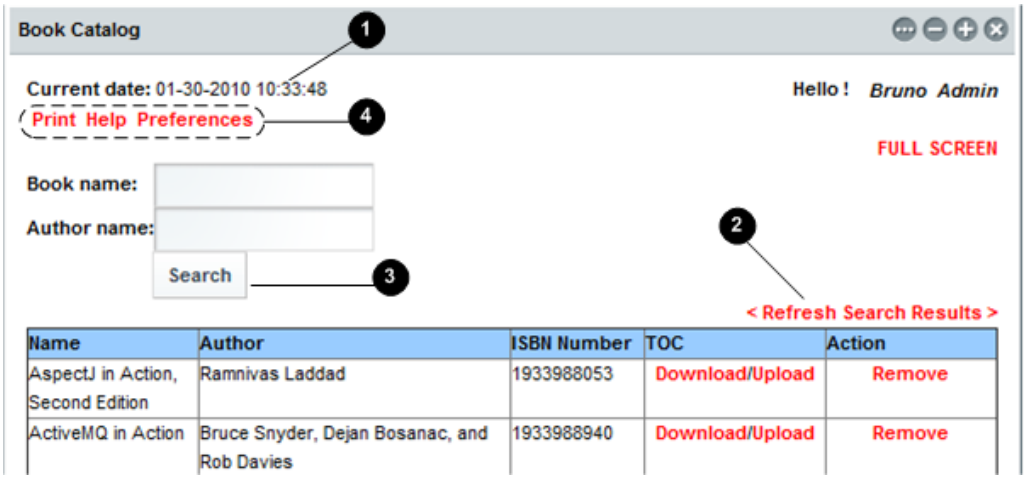


Figure 1 The Book Catalog portlet's initial content, which shows the current date/time, books in the catalog, and options to search for books and add books to the catalog

At #1, the system's current date/time is displayed in MM-dd-yyyy HH:mm:ss format. At #2, if a user selects the Refresh Search Results hyperlink, it sends a render request to the portlet. At #3, if a user clicks the Search button, it sends an action request to the portlet. At #4, if a user selects the Print, Help, or Preferences hyperlink, it sends a render request to the portlet.

The render request is dispatched to the Book Catalog portlet in a couple of scenarios, including:

- Users' refreshing the portal page, which contains the Book Catalog portlet, using the browser's refresh button.
- User interaction with the Book Catalog portlet that results in sending a render request for the portlet. For instance, clicking the Refresh Search Results and Print hyperlinks (refer to figure 1) sends render request to the portlet.
- User interaction with some other portlet on the same portal page. In this case render request is sent to all the portlets on the portal page, including Book Catalog portlet.

Because the Book Catalog portlet defines 60 seconds as its cache expiration time, render requests received within 30 seconds of content generation will always result in showing the cached content. The current date/time displayed by the Book Catalog portlet can be used to identify whether the content is fresh or cached.

Table 1 shows the behavior of the Book Catalog portlet when it is deployed on JetSpeed 2.2 and Liferay 5.2.3.

Table 1 The impact of expiration caching on the behavior of the Book Catalog portlet

User action	JetSpeed 2.2	Liferay 5.2.3
User presses the browser's Refresh button	Current date/time displayed by Book Catalog *doesn't* change.	Current date/time displayed by the Book Catalog *doesn't* change.
User clicks the hyperlink Refresh Search Results or Print or any other hyperlink that sends a render request to the portlet	Current date/time displayed by the Book Catalog changes, if the render URL is not the one that generated the content; that is, to generate fresh content there should be difference in render parameters or portlet mode of the URLs.	Current date/time displayed by the Book Catalog *changes* to reflect that the content is freshly generated.
A user takes action on any other portlet on the same portal page as the Book Catalog portlet	Current date/time displayed by the Book Catalog *changes* to reflect that the content is freshly generated.	Current date/time displayed by the Book Catalog *doesn't* change.

NOTE

Table 1 describes the behavior of the Book Catalog portlet deployed on Liferay 5.2.3 and JetSpeed 2.2, when the user is logged in. A user who is not logged into the Liferay portal will experience that the current date/time changes even when actions are taken on other portlets or when the page is refreshed using the browser's refresh button. This behavior can be attributed to the *private* caching-scope.

WARNING

In JetSpeed 2.2, some of the Book Catalog portlet's functionality will not work because JetSpeed 2.2 caches content based on the render URL that generated the content. Reinvoking the portlet using the same render URL will not have any effect until the content expires. To use the Book Catalog portlet on JetSpeed 2.2, set the expiration time for the cache to 0, which means that the content is always considered expired by the portlet container.

Expiration caching support

The expiration caching support is optional for portlet containers. The portlet container has the flexibility to disable portlet content caching at anytime to reclaim the memory held by the cached content.

Glassfish, Liferay 5.2.3 and JetSpeed 2.2 support expiration-based caching strategy.

NOTE

In multipage portlets like the Book Catalog portlet, it is sometimes preferred to cache content for specific page(s) instead of caching content for all portlet pages. Some portal servers, like WebSphere, provide an additional feature of allowing caching content for specific page(s) of a portlet.

Let's now look at validation-based caching strategy, which is an extension to the expiration-based caching strategy.

Validation-based caching strategy

Validation-based caching strategy extends the expiration caching strategy by allowing portlets to *validate* the cached content after the expiration time has passed. If a portlet finds that the cached content is still valid, then the portlet instructs the portlet container to use the cached content for another expiration period. If the cache becomes invalid, then the portlet generates fresh content.

In validation-based caching strategy, a validation token is used for validating cached content. The value of validation token is *unique* to the cached content. The validation token is set by the portlet and stored by the portlet container when the content is cached. The validation token is made available to the portlet once the cached content expires. Portlet then compares the value of validation token with that of current state of the system to find whether the cached content is still valid or not. If the cached content is still valid, then portlet instructs the portlet container to continue using it. If the cached content is no longer valid, then the portlet generates fresh content.

The steps followed in validation caching are shown in figure 2, in the context of the Book Catalog portlet. The Book Catalog portlet uses the number of books in the data store as the value of validation token. If the number of books changes in the database, the Book Catalog portlet generates a fresh list of books.

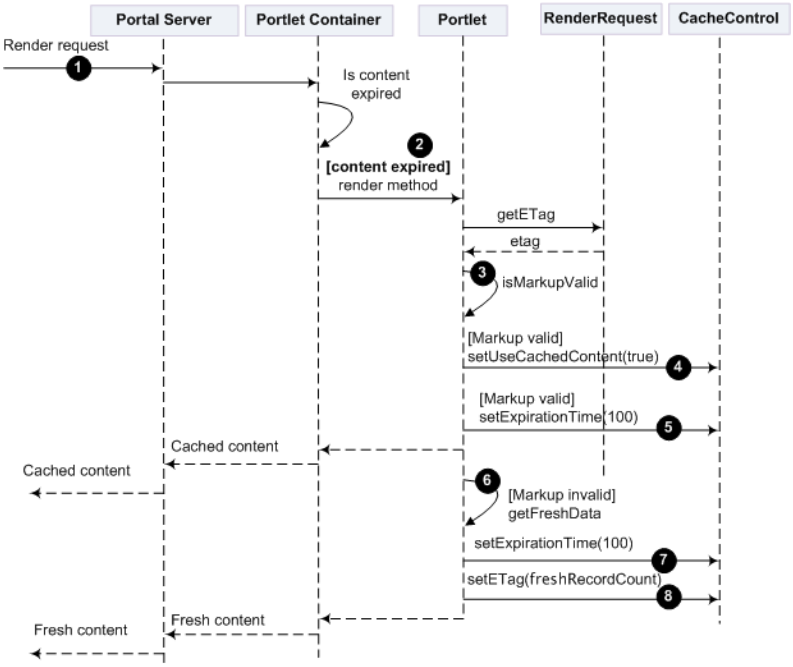


Figure 2 Validation caching in the Book Catalog portlet. The portlet obtains the value of a validation token, which is record count, from RenderRequest using getETag method. The portlet compares the current record count in the database/datastore with that of the record count value obtained using getETag method.

At #1, a render request is sent to the Book Catalog portlet. The portlet container checks if the cached content's expiration time has passed or not. If the cached content's expiration time has passed, then the render method of the portlet is invoked at #2. In the Book Catalog portlet, the number of books stored in the data source is used as a validation token value. In the `isMarkupValid` method at #3, the portlet uses the `getETag` method of

RenderRequest to obtain the *validation token value*, which is the number of books at the time when the content was last cached. Let's say the value of the validation token is 10. The portlet checks the number of books that are currently in the data store and compares it with the validation token value. If the validation token value is the same as the number of books currently stored in the database, the portlet instructs the portlet container to use the already cached content by calling setUseCachedContent method of the CacheControl object at #4. The CacheControl object of Portlet 2.0 allows programmatically setting scope, expiration time, and the validation token value and instructing the portlet container to use the cached content. If the expiration time isn't set at #5, the content will be considered *always* expired, which means that the portlet container will invoke the portlet's render method again when it receives a render request for the portlet. If the validation token value and the current number of books in the data store don't match, then the cached content is no longer valid. Therefore, the portlet retrieves fresh data at #6 from the data source and generates fresh content for the portlet. The portlet uses the setETag method of CacheControl at #7 to set the current number of books in the data store as the new value of validation token. The Book Catalog portlet sets the expiration time to 100 for the newly generated content at #8 so that the content is not considered expired for next 100 seconds by the portlet container when the render request is received for the portlet.

Listing 1 shows how the Book Catalog portlet implements the isMarkupValid method shown in figure 1. The isMarkupValid method of BookCatalogPortlet class is responsible for validating that the content cached by the portlet container is still valid.

Listing 1 isMarkupValid method of BookCatalogPortlet class

```
private boolean isMarkupValid(RenderRequest request,
     RenderResponse response) {
  boolean isMarkupValid = false;
  BookDataObject catalog = (BookDataObject)                          #1
    getPortletContext().getAttribute("bookCatalog");                 #1
  int currentCountInDatastore =                                      #1
      catalog.getBooks().size();                                     #1

  String earlierCount =                                              #2
    response.getCacheControl().getETag();                            #2

  if (String.valueOf(currentCountInDatastore).                       #3
           equals(earlierCount)) {                                   #3
    isMarkupValid = true;                                            #3
  }                                                                  #3
  return isMarkupValid;
}
```

#1 Get count of books in the catalog
#2 Retrieve the book count from the validation token
#3 Compare the current count with the validation token

At #1, isMarkupValid method finds the number of books that are currently in the catalog. At #2, validation token value is obtained from request. The validation token value represents the number of books that were there when the content was last cached by the portlet container. At #3, check is made if the book count stored in the validation token is same as the number of books that are currently in the catalog. If the values match, then it is assumed that the Book Catalog hasn't changed since the last time content was cached.

Listing 2 shows how the isMarkupValid method is used by the render method of BookCatalogPortlet class.

Listing 2 showBooks render method of BookCatalogPortlet class

```
@RenderMode(name = "VIEW")
public void showBooks(RenderRequest request,
   RenderResponse response) throws ....{
 if(isMarkupValid(request, response)) {
   response.getCacheControl().setUseCachedContent(true);             #1
   response.getCacheControl().setExpirationTime(60);                 #2
   return;
 } else {
```

```
    BookDataObject catalog = (BookDataObject)                          #3
      getPortletContext().getAttribute("bookCatalog");                 #3
    int currentCountInDatastore = catalog.getBooks().size();           #3
    response.getCacheControl().setETag("" +                            #4
      currentCountInDatastore);                                        #4
  }
  ....
}
```

#1 Instruct for using cached content
#2 Set new expiration time for content
#3 Obtain current book count in catalog
#4 Set validation token value to book count

At #1, if the cached content is found valid, that is, `isMarkupValid` method returns `true`, the render method of the Book Catalog portlet instructs the portlet container to use the cached content. At #2, a new expiration time is set for the cached content. If the expiration time is not set for the cached content, the content is treated as always expired. At #3, if the cached content is not found valid, we obtain the current book count from the data store. At #4, we set the validation token value to the current book count.

TIP

The Book Catalog portlet makes use of the record count as a validation token value to introduce you to the concept of validation tokens. In real-world portlets, you will need to figure out what can act as a validation token value for the cached content of the portlet. Some examples of what you can possibly use is: last modified timestamp of the record and hash code value of an object. The `setETag` method accepts `String` as the value of validation token, which may not be sufficient for your portlets to identify whether the cached content is valid or not. To address scenarios where `String` value for validation token doesn't meet your requirements, store your *actual* validation token value (a complex object) in `PortletSession` and store a *dummy* validation token value using `setETag` method. When the render method of your portlet is invoked, you obtain the actual validation token from `PortletSession` and validate the cached content.

Validation caching support

The validation caching support is optional for portlet containers. Glassfish (with OpenPortal Portlet Container 2.1), JetSpeed 2.2, and Liferay 5.2.3 (with default porltet container) don't support validation caching, that is, the `getETag` method of `CacheControl` object always returns `null`.

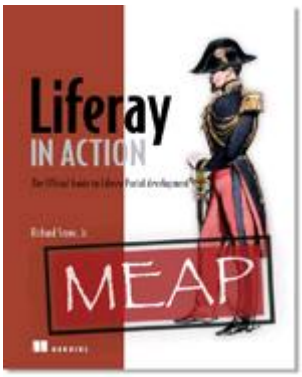
Summary

This article focused on Portlet 2.0 API and two different caching strategies. We used Liferay 5.2.3, JetSpeed 2.2 (configured with Pluto 2.0 portlet container) and Glassfish v2.1 (configured with OpenPortal Portlet Container 2.1) to describe the support for the caching strategies.

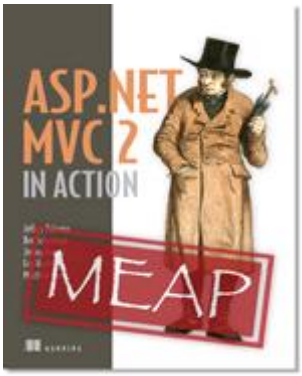
Here are some other Manning titles you might be interested in:

[Spring in Action, Third Edition](#)
EARLY ACCESS EDITION

Craig Walls
MEAP Release: June 2009
Softbound print: Fall 2010 | 700 pages
ISBN: 9781935182351



[Liferay in Action](#)
EARLY ACCESS EDITION
The Official Guide to Liferay Portal development
Richard Sezov, Jr
MEAP Release: February 2010
Softbound print: Fall 2010 | 375 pages
ISBN: 9781935182825



[ASP.NET MVC 2 in Action](#)
EARLY ACCESS EDITION

Jeffrey Palermo, Ben Scheirman, Matthew Hinze, Jimmy Bogard, and Eric Hexter
MEAP Release: February 2010
Softbound print: Summer 2010 | 450 pages
ISBN: 9781935182795