



IBM Software Group

IBM WebSphere Application Server V6

Dynamic Cache



@business on demand.

© 2006 IBM Corporation
Converted to video May 12, 2015

This presentation will focus on Dynamic Caching in WebSphere® Application Server.

Goals

- Introduce dynamic caching feature that is a part of WebSphere Application Server
- Describe how to enable dynamic caching service
- Introduce cache monitoring tools that are a part of WebSphere Application Server



The goals for this presentation are to introduce you to the dynamic caching feature that is a part of the WebSphere Application Server. The presentation will also introduce you to monitoring tools that can be used to tune and monitor your Dynamic Cache. The goal is also to point out any differences between Version 5 and Version 6 along the way.

Agenda

- Overview
- Additional Features
- Enabling Dynamic Caching
 - ▶ Administrative Console settings
 - ▶ cachespec.xml file
- Cache Monitoring Tools
 - ▶ Performance Monitoring Infrastructure
 - ▶ Dynamic cache monitor



The agenda for this presentation is to discuss the Dynamic Caching functionality, Administrative Console settings, and monitoring tools that can be used to monitor and tune dynamic cache.

Section

Overview



This section will discuss what the Dynamic Caching functionality provides in WebSphere Application Server.

Dynamic Caching

- Application Server service used to temporarily store dynamically created Java™ 2 Enterprise Edition (J2EE) application content
 - ▶ Relieves backend servers from repeatedly creating data therefore increasing performance
- Non-intrusive solution – no/limited amount of code modification needed to implement
 - ▶ Policy files used to store caching rules (cachespec.xml)
 - ▶ API available to programmatically work with cache
- Built-in invalidation support – rule-, time-, group, and programmatic-based methods



Dynamic caching is a built-in WebSphere service that stores generated application content to memory or disk so that subsequent requests can be served by using the already generated content. This increases the performance of the Application Server, because it does not have to go through the steps of reprocessing each step over and over again. Instead it can just grab the already generated result from cache and return that data. Ideal contents to cache are those that are expensive to reproduce, but do not change frequently. For example, you may want to cache mutual fund price lookups for a brokerage application.

Dynamic caching was designed to be a non-intrusive caching solution. Generally, using dynamic caching is as simple as enabling the dynamic caching service and defining cache policy files, named cachespec.xml, within your application. You will need to write code within your application when you want to cache WebSphere command objects or Java™ objects. Enabling and configuring your application for caching will be discussed later on in the presentation.

Dynamic caching has built-in invalidation support. Cache invalidation ensures that your cache stays fresh and consistent with minimal effort on your part. To enable invalidation within your cache you can define invalidation rules within your cache policy files, or it can be done programmatically within your application through the provided API.

Cacheable Items

- Servlets, JSPs, Struts, and Tiles (ex: stock quote data, account information)
 - ▶ Stores whole or fragments of generated pages
 - ▶ Struts caching – Support for multiple cache policies per servlet
 - ▶ Tiles caching – Use `<tiles:insert...flush=true>` in order for fragments to be cached correctly
- WebSphere command objects (ex: database queries, file system lookups)
- Java objects (ex: EJBs)
- Web Services responses



Dynamic cache is capable of storing many different types of content. In Version 6, the ability to additionally cache Tiles and Struts pages and fragments has been added.

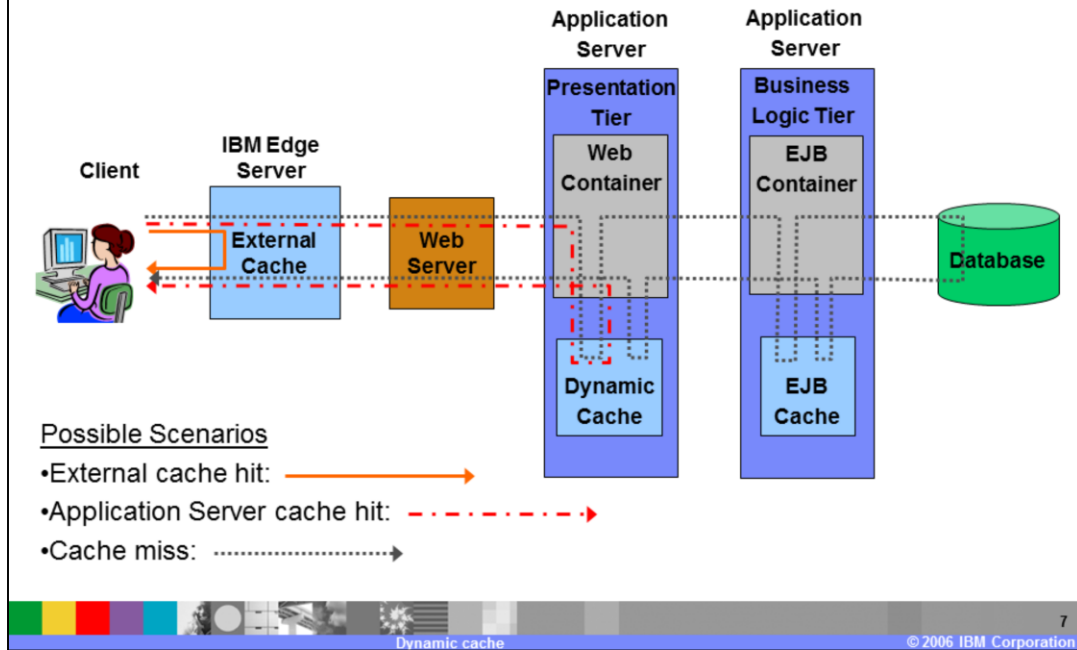
Dynamic cache can be used to store either whole pages or fragments of generated pages from Servlets, JSPs, Struts or Tiles. When storing Tiles to cache it is important to note that the “flush=true” attribute needs to be used along with the “tiles:insert” tag. This is the same behavior as caching JSP fragments. The cache policy file `cachespec.xml` is used to cache servlets, JSPs, Struts, and Tiles. An example of when you may want to cache a JSP response could be a customer’s account for a shopping application or a brokerage application. The customer’s account information in this example would be something that stays fairly static and would not require a lot of updating.

Dynamic caching can be used to store WebSphere command objects. To cache WebSphere command object you will need to extend your command objects using the `CacheableCommandImpl` abstract class. After you have written your command object to be cacheable, you will need to add entries to your application’s cache policy file that describe how to cache the command object. An example of a command object that could be cached could be a file system lookup. If the lookup is one that is commonly used in your application you can avoid the expense of having to recreate the lookup and just reuse it throughout your application.

Dynamic caching can be used to store Java objects. Caching Java objects within your application requires that you use the provided cache APIs provided with WebSphere Application Server. The cache policy file is not involved with caching Java objects. An example of a type of Java object you may cache would be EJB Entity Beans.

Dynamic caching can be used to store Web Services responses. Web Services responses can be cached in several ways, ranging from using just the SOAP action or the entire SOAP envelope as the cache identifier. The cache policy file is used to cache Web Services responses. An example of when you may want to cache a Web Service response would be if you had a Web Service that just returned your company’s current stock value.

Dynamic Caching Example



The diagram in the slide attempts to outline three different caching scenarios. In the first scenario, the request is served from the external cache. This saves the HTTP server and Application Servers from having to process the request. In the second scenario, the appropriate response for the request can not be served from the external cache. The request then passes through the HTTP server in route to the Application Server's Web container. Before the Web container begins processing the request, dynamic cache is checked for a matching response. In this case a matching response is found and returned. This saves the Application Server from having to send the request to EJB container for further processing. In the third scenario, the request can not be served from either the external cache or from the cache residing on the first Application Server. In this case, the request needs to move throughout the entire system. It should be noted, however, that as the response leaves the system it may be stored in cache somewhere along the way so that subsequent requests will benefit from the generated response.

Section

Additional Functionality



The next section will discuss additional functionality that is available when using the Dynamic Caching service.

Edge of Network Caching

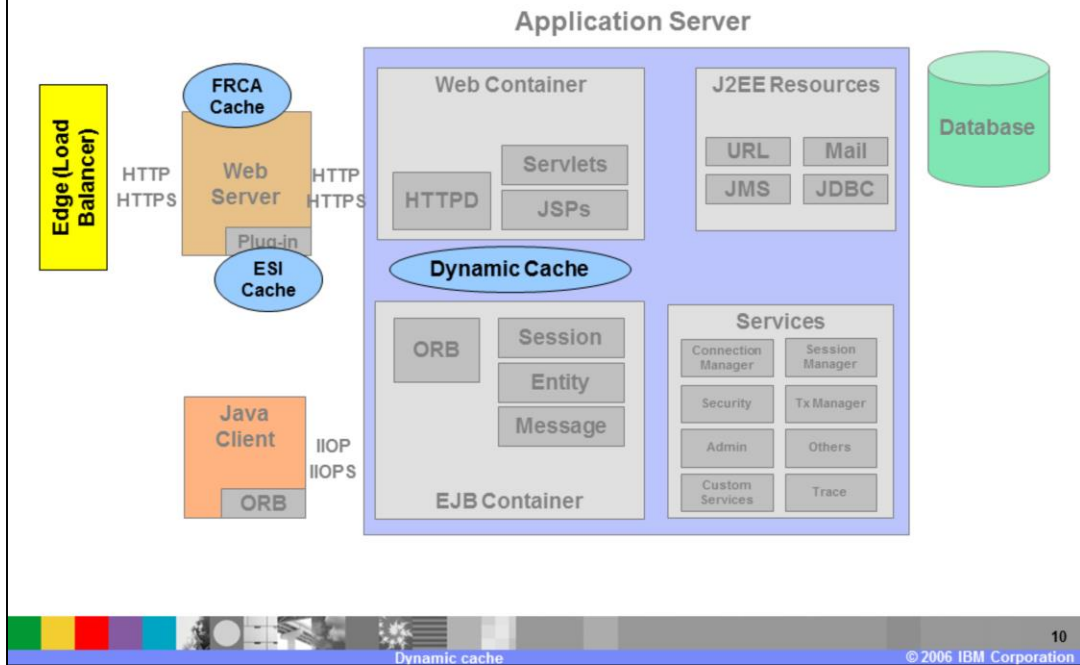
- Allows Application Server to extend caching support and control to network-based caches
- Supported external caching solutions
 - ▶ IBM HTTP Server for distributed platforms' Fast Response Cache Accelerator (FRCA) cache
 - ▶ WebSphere HTTP Server plug-in for distributed platforms' Edge Side Include (ESI) Fragment Processor
 - ▶ IBM Edge Server
 - ▶ Third party "Edge of Network" caching solutions (ex: Akamai)



WebSphere Application Server includes support for controlling edge of network caches. Edge of network caches are those caches which exist outside of the Application Server. Edge of network caches allow responses to be served to incoming requests before they ever reach your Application Server. This results in less network traffic and load on your Application Server. When the Application Server is properly configured it is capable of invalidating entries within the supported edge of network solutions.

WebSphere Application Server supports several different edge of network solutions. It currently has support for the IBM HTTP server for distributed platform's Fast Response Cache Accelerator or FRCA cache, WebSphere HTTP server plug-in for distributed platforms' Edge Side Include Fragment Processor and the IBM Edge Server. In addition to the IBM supported solutions, WebSphere Application Server also supports other third party edge of network caching solutions.

Architecture: Dynamic Caching Components



The diagram on this slide shows where the edge of network solutions discussed in the previous slide fit into a WebSphere Application Server environment.

Dynamic Content Provider



- Allows fragments that contain dynamic content to be cached
 - ▶ Use DynamicContentProvider interface
- When servlet response is generated – dynamic content provider is added to response object
 - ▶ Servlet response cached for subsequent requests
- When servlet response is returned from cache – dynamic content provider called to output dynamic portion of page



The dynamic content provider allows fragments that contain dynamic content to be cached. An example of where this can be used is helpful in this situation. Pretend that you have a home page that contains all static content except for a date and time stamp that is updated each time a user accesses the home page. In previous versions of WebSphere Application Server you would not be able to cache this page because the response would need to be recreated each time the page is requested. In Version 6, you are now able to cache the static portion of the page and have the Application Server generate the dynamic content when the page is returned.

Again how the dynamic content provider works outside of the example is as follows: When the servlet response is generated it adds a dynamic content provider to the response object. The response object is then in turn cached and returned. When another user requests the same response, cache loads the response and the appropriate dynamic content providers are called to generate the remainder of the content for the response. The response is then forwarded to the user.

Dynamic Content Provider: Example

```

public class DCPServlet extends CacheTestCase {
    public void performTest(HttpServletRequest request, HttpServletResponse response) throws IOException,
        ServletException {
        out.println("Testing the DCP feature begin "+System.currentTimeMillis());
        DynamicContentProvider dcp = new DynamicContentProviderImpl();
        ServletCacheResponse scr = (ServletCacheResponse)(response);
        scr.addDynamicContentProvider(dcp);
        out.println("Testing the DCP feature end"+System.currentTimeMillis());
    }

    class DynamicContentProviderImpl implements DynamicContentProvider {
        DynamicContentProviderImpl() {}

        public void provideDynamicContent(HttpServletRequest request, OutputStream streamWriter) throws IOException {
            String dynamicContent = System.currentTimeMillis();
            streamWriter.write(dynamicContent.getBytes());
        }

        public void provideDynamicContent(HttpServletRequest request, Writer streamWriter) throws IOException {
            String dynamicContent = System.currentTimeMillis();
            streamWriter.write(dynamicContent.toCharArray());
        }
    }
}

```


Adds dynamic content provider to the servlet response

When the servlet response is generated the appropriate method from the dynamic content provider will be called to generate the time stamp for the page



This slide further expands on the example given in the previous slide. Notice that in the performTest method a dynamic content provider is being added to the servlet's response between the two System.out.println lines. The DynamicContentProviderImpl class will generate the dynamic portion of the servlet response – in this case the class will output a time stamp to the servlet response object. It is important to note that the two lines generated in the performTest method will be cached and will not change each time the response is generated. The time generated by the DynamicContentProviderImpl class, however, will change each time the servlet is requested.

Cache Instances

- Provides additional locations where dynamic cache can store, distribute and share data
 - ▶ In addition to default shared dynamic cache
 - ▶ Increases configurability/performance of dynamic cache
- Servlet cache instances 
 - ▶ New in V6
 - ▶ Extends existing cache instance support
- Object cache instances
 - ▶ Known as “Cache Instances” in previous releases
 - ▶ Functionality has been extended



Cache instances allow you to define additional locations that can be used to store, distribute and share cached objects. These cache instances are defined in addition to the default dynamic cache that is part of the WebSphere Application Server. The ability to define additional cache instances allows WebSphere Application Server to become a more configurable solution that is capable of adapting to more customers environments. For example, in your brokerage application you may want to store your cached stock quote lookups in a separate cache instance than WebSphere Application Server's default dynamic cache. In addition to being more configurable, the ability to create separate caches also leads to a performance gain. Instead of storing all of your cached items to one large cache, you are now able to create several smaller caches which will result in faster cache lookups.

There are two different types of cache instances available in WebSphere Application Server. Both will be mentioned on this slide and then covered in detail in later slides. The first type of cache instance is the servlet cache instance. Servlet cache instances were not available in WebSphere Application Server Version 5. It is important to note that servlet cache instances are an extension of the cache instance support that was available in WebSphere Application Server Version 5. The second type of cache instance is the object cache instance. Object cache instances were available in WebSphere Application Server Version 5 but were know as “cache instances” in that release. In Version 6 that same functionality has been renamed “object cache instances” and further enhanced. The upcoming slides will discuss the enhancements made to object cache instances.

Servlet Cache Instances

- Allows Portal, Commerce, and web applications to store data in separate caches
- Stores servlets, JSPs, Struts, Tiles, command objects, and Web Services responses
- Create as resource in Administrative Console
- Specify the cache instance in the cache policy file by using the `<cache-instance...>` tag
 - ▶ Multiple instances can be specified in a cache policy file



Servlet cache instances allow applications running on WebSphere Application Server to store their cached objects in a cache other than the default dynamic cache. The servlet cache instance support was derived with the Portal and Commerce servers in mind, but can also be utilized by web applications running on the Application Server. Servlet cache instances are used to store whole pages or fragments of pages of servlets, JSPs, Struts, or Tiles; WebSphere command objects; or Web Services responses.

To utilize servlet cache instances in your application, you must first create a servlet cache instance within the Administrative Console. After the cache instance has been created, you need to add the cache-instance tag to your cache policy file. The cache-instance tag allows you to specify the JNDI name of the servlet cache instance where cached objects should be stored. The cache-instance tag can appear in the same cache policy files multiple times. The cache-instance tag is optional – the absence of the tag means that cached objects should be stored to the default dynamic cache.

Object Cache Instances


- Stores cached Java objects
- Provided APIs allow applications to interact with cache
- Cache replication used to share cached objects between clustered servers
- Create using the following methods
 - ▶ Create as resource in Administrative Console
 - ▶ Define in `cacheinstances.properties` file
 - Allows developer to define and package cache instance with application
 - `distributedmap.properties` file has been deprecated in V6





Object cache instances also allow applications running on the WebSphere Application Server to store cached objects to locations other than the default dynamic cache. Object cache instances are used to store Java objects. WebSphere Application Server provides APIs that allow you to work with the dynamic cache within your application. WebSphere Application Server also provides mechanisms for sharing cached objects between multiple servers within a cluster.

Object cache instances can be created in two different ways. The first method involves creating the object cache instance within your Administrative Console. The second method involves using the `cacheinstances.properties` file to store the definition of object cache instance with the application. Then when the application is installed on your Application Server, the object cache instances are created as part of the installation process. The second method is the preferred method of creating object cache instances because it allows you to define the object cache instances along with the application. An important thing to note here is that `cacheinstances.properties` file replaces `distributedmap.properties` files from earlier versions of WebSphere Application Server. The `distributedmap.properties` file is being deprecated in Version 6 – which means that it will only be supported for the three next major releases of WebSphere Application Server.

Object Cache Instances: Types and APIs

Types		V5	V6
 Servlet Cache Instance		No	Yes
Object Cache Instance ¹		Yes ¹	Yes

APIs		V5	V6
 DistributedObjectCache		No	Yes
 DistributedNioMap		No	Yes
DistributedMap		Yes	Yes

1 Called "Cache Instance" in V5



This table further clarifies what was discussed on earlier slides and also introduces the new object cache APIs. As you can see from the table, the servlet cache instances functionality was introduced in Version 6. Also from the table note that the object cache instance functionality existed in both WebSphere Application Server Version 5 and Version 6, although it was named cache instances in Version 5. Also note that two new APIs have been added for Version 6. Both of the APIs will be introduced on an upcoming slide.

Object Cache Instances: APIs

- **DistributedObjectCache**
 - ▶ Implements DistributedMap and DistributedNioMap
 - ▶ Programming interface for object cache instances
- **DistributedNioMap**
 - ▶ High performance map
 - ▶ Designed to store java.nio.Buffer objects
 - ▶ Release() method is called when a cached object implementing DistributedNioMapObject interface is removed from cache



Both of the APIs listed on the slide are new for Version 6. The DistributedObjectCache interface implements both the DistributedMap and DistributedNioMap interfaces and should be used when writing applications that access cache instances. The DistributedNioMap interface was also added. The DistributedNioMap interface provides a high performance map that was designed to store java.nio.Buffer objects.

Cache Replication



- Cache replication allows cached objects to be shared across multiple servers in a cluster
- Cache replication settings are more configurable
 - ▶ V5 – All instances shared the same replication settings
 - ▶ V6 – Replication settings are per instance

cacheinstance.properties

```
cache.instance.1=services/cache/map2
cache.instance.1.cacheSize = 1000
enableCacheReplication = true
replicationDomain = domainName
```

Administrative Console

Consistency settings

Use listener context

Enable cache replication

Replication domain
DynaCacheCluster

Replication type
Push only

Push frequency
1



Cache replication allows WebSphere Application Server to share cached objects across multiple servers within the same cluster. What this means is that cached data is generated one time on one server and copied to the rest of the servers in the replication domain or cluster. This results in savings of time and work for the cluster as a whole.

Cache replication is more configurable in WebSphere Application Server Version 6. Object cache instances can now define their own replication settings. In Version 5, all object cache instances on a server had to share the same set of replication settings.

Section

Enabling Dynamic Cache



The next section will discuss how to enable the Dynamic Caching functionality within WebSphere Application Server.

Administrative Console

- Enable Dynamic cache service (Servers > Application servers > \${SERVER} > Container Services > Dynamic Cache Service)
 - ▶ Enabled by default
 - ▶ Optionally configure extended network caching solutions from same panel
- Optionally enable servlet caching (Servers > Application servers > \${SERVER}> > Web Container Settings)
 - ▶ Disabled by default
 - ▶ Create cachespec.xml files to configure cacheable data
- Optionally create servlet and object cache instance(s) (Resources > Cache Instances)



20

Dynamic cache

© 2006 IBM Corporation

The first step in using dynamic caching in WebSphere Application Server is validating that it is enabled on your server. By default the dynamic caching service is enabled on a new install of WebSphere Application Server. To access the dynamic caching service, navigate to the Dynamic Cache Service panel within the Administrative Console. From this panel you can configure options such as the number of items that can be stored in dynamic cache or whether or not cache overflow should be pushed to disk. You can also configure any edge-of-network solutions from this panel. After you have made your changes to the dynamic cache service, you will need to restart the server in order for the changes to take effect.

If you want to enable the ability to cache servlets, JSPs, Tiles, Struts, WebSphere command objects, or Web Services responses you will need to enable servlet caching. By default the servlet caching feature or dynamic caching is disabled on a new installation of WebSphere Application Server. To enable servlet caching, navigate to the Web Container Settings panel within the Administrative Console. After you have enabled servlet caching you will need to restart the Application Server in order for changes to take effect.

The next step in enabling and configuring dynamic cache is to optionally create any needed servlet and object cache instances for your application. Remember that cache instances provide additional locations where cache objects can be stored outside of the default dynamic cache.

After you have configured dynamic caching, the last step is incorporating the correct cache policy files and code into your application to the dynamic cache service.

The demonstration that is available on this slide takes you through the different dynamic cache configuration options and where they are located within the Administrative Console. To view the demonstration, pause this presentation and click the Show Me icon.

cachspec.xml

- Defines how an item should be cached, what it is dependent on and when it should be invalidated
- File located in either or both
 - ▶ `${PATH_TO_PROFILE}\properties` directory – Server-wide settings
 - ▶ Web module `WEB-INF` directory or enterprise bean `META-INF` directory – Individual application settings (recommended way)
- Configured per application, server or both



The `cachspec.xml` file, defines the rules used to cache servlets, JSPs, Tiles, Struts, WebSphere command objects, and Web Services responses. The file defines which items should be cached, how their cache identifiers should be generated, which objects they are dependent on (if any) and when a cached object should be invalidated or removed from cache.

The `cachspec.xml` file can be stored either on your server in the profile's property directory, or with your application in the `WEB-INF` directory of your Web application or the `META-INF` directory of your enterprise bean. When the `cachspec.xml` file is stored within your application, no additional administrative configuration is needed. Storing the `cachspec.xml` file within the application is the recommended way of storing cache policy file.

See the WebSphere Application Server Information Center for more information on the `cachspec.xml` file.

cacheSpec.xml Example

```

<cache-instance name="services/cache/HRWebAppCache">
  <cache-entry>
    <class>Servlet</class>
    <name>/HRServlet</name>
    <cache-id>
      <component id="hr_user" type="parameter">
        <required>true</required>
      </component>
      <timeout>0</timeout>
      <priority>1</priority>
    </cache-id>
  </cache-entry>
</cache-instance>

```

Servlet cache instance that will be used to store cached items (optional)
If omitted the default dynamic cache is used

Type of item that is being cached

Fully qualified class name of cached item

Rules for generating unique cache identifiers

Time to live value for the amount of time cached value remains in cache (optional)

Used by dynamic cache to determine whether a cache entry should be removed when cache runs out of storage space (optional)

For complete documentation see the WebSphere Application Server Information Center



This slide displays an example taken from a simple cacheSpec.xml file. The class and name tags in the example indicate that the cache policy file was written to cache responses from the HRServlet servlet. The cache-id tag is used to define which responses (in this case) to cache and how to generate their cache identifiers. The cache-id tag, in this case, also contains an invalidation rule and a priority value. The component tag is used to generate the cache identifier for the cached object. The cache identifier should be defined in a way that generates a unique identifier for the cached object. Rules for cache identifier generation should be defined in a way such that duplicate cache IDs are not generated. This is because the cache ID is used to access the cached object, and duplicate cache entries would result in the loss of data integrity in your application. In this example, the cache identifier is generated from the value of the hr_user parameter passed to the servlet. The timeout tag defines the invalidation rule for this cache entry. In this example, the value of zero indicates that the cached object will live in cache until it is removed by the least recently used algorithm. The priority tag defines the priority of the cached object in the cache. The priority value is used by the cache's least recently used algorithm to determine which items should be removed from cache when cache is full.

Section

Monitoring Tools



The next section will discuss the monitoring tools that you can use to monitor and analyze the performance and contents of your caches.

Performance Tools

- Dynamic Cache PMI module provides counters on key areas of the dynamic cache service
 - ▶ Total number of in-memory cache entries
 - ▶ Number of in-memory hits/misses
 - ▶ Number of invalidations
 - ▶ Cache replication statistics
- Use PMI clients to view collected performance data



The Performance Monitoring Infrastructure, or PMI, provides a Dynamic Cache module that allows you to monitor cache performance through your PMI clients. The list provided in this slide is an incomplete list of the performance counters that are available for dynamic caching. The complete list of performance counters can be obtained from WebSphere Application Server Information Center. The dynamic cache performance data can then be viewed using a PMI client, such as the Tivoli Performance Viewer.

Dynamic Cache Monitor

- Web application that allows you to view and invalidate data currently in cache
- Install cachemonitor.ear from `${WAS_INST}/installableApps` directory
- Access using `http://${HOST_NAME}:${PORT_NUMBER}/cache monitor`



The dynamic cache monitor is a Web application. The cache monitor allows you to view and control the servlet cache instances installed on your Application Server. Using the cache monitor you can view basic usage statistics, the contents in the cache, and the policies loaded for each cache instance available on the Application Server. Additionally it is capable of looking at both the caches that are located on your hard disk and edge-of-network caches. The cache monitor is useful in debugging which of your defined policies were loaded and whether the correct objects are being cached. It is an integral part in debugging problems that occur with dynamic cache.

By default, the cache monitor application is not installed on a new WebSphere Application Server installation. The application can be installed from the `installableApps` directory at the root of your WebSphere Application Server installation. After the application is installed, it can be accessed from the URL shown here.

Dynamic Cache Monitor: Cache Statistics

WebSphere Application Server Cache Monitor

Cache instance
Refresh Instances
baseCache OK

Cache Statistics
Reset Statistics Clear Cache

Statistic	Value
Cache Size	2000
Used Entries	0
Cache Hits	0
Cache Misses	0
LRU Evictions	0
Explicit Removals	0
Default Priority	1
Servlet Caching Enabled	Yes
Disk Offload Enabled	No

Use to select cache instance that you want to view

Displays general statistics for selected cache instance

Dynamic cache © 2006 IBM Corporation 26

This slide shows the Cache Statistics view. This view displays the general usage statistics for the selected cache instance. Notice that the cache instance that is selected is the “baseCache” instance - this is dynamic cache’s default cache instance. A majority of the statistics in this view are also captured by the dynamic cache PMI module discussed earlier.

Dynamic Cache Monitor: Cache Contents

WebSphere Application Server Cache Monitor

Cache instance
Refresh Instances
baseCache OK

Cache Statistics
Edge Statistics
Cache Contents
Dependency IDs
Disk Offload
Cache Policies

Current Cache Contents
Entries 1 through 39 of 39
< 0 >
Clear Cache

Template	Cache ID
/PlantsByWebSphere/servlet/ImageServlet	/PlantsByWebSphere/servlet/ImageServlet:action=
/PlantsByWebSphere/servlet/ImageServlet	/PlantsByWebSphere/servlet/ImageServlet:action=
/PlantsByWebSphere/servlet/ImageServlet	/PlantsByWebSphere/servlet/ImageServlet:action=
/PlantsByWebSphere/servlet/ImageServlet	/PlantsByWebSphere/servlet/ImageServlet:action=
/PlantsByWebSphere/servlet/ImageServlet	/PlantsByWebSphere/servlet/ImageServlet:action=
/PlantsByWebSphere/servlet/ImageServlet	/PlantsByWebSphere/servlet/ImageServlet:action=
/PlantsByWebSphere/servlet/ImageServlet	/PlantsByWebSphere/servlet/ImageServlet:action=

Local intranet

Dynamic cache © 2006 IBM Corporation 27

Displays data that is currently in the select cache instance

Useful for troubleshooting whether your cache policies are caching the correct objects

This slide shows the Cache Contents view. This view displays the cached objects that are currently in the selected cache instance. If you scroll to the right of the window you can see additional information such as the cache policy that was used to create the cached object and amount of time before the object is removed from cache. You can also click on each one of the entries to see the object that is being cached. For example if you click on a cached HTML page, you will be able to see the HTML page that has been cached. The information provided in this view makes it very useful in determining whether the correct content from your application is being cached.

Dynamic Cache Monitor: Cache Policies

WebSphere Application Server Cache Monitor

Cache instance
Refresh Instances
baseCache OK

Cache Statistics
Edge Statistics
Cache Contents
Dependency IDs
Disk Offload
Cache Policies

Current Cache Policies

Template	Class
/trade/app	servlet
/trade/register.jsp	servlet
/trade/marketSummary.jsp	servlet
/trade/displayQuote.jsp	servlet
com.ibm.websphere.samples.trade.command.MarketSummaryCommand.class	command
com.ibm.websphere.samples.trade.command.QuoteCommand.class	command
com.ibm.websphere.samples.trade.command.UpdateQuotePriceCommand.class	command
com.ibm.websphere.samples.trade.command.AccountCommand.class	command
com.ibm.websphere.samples.trade.command.AccountProfileCommand.class	command

Local intranet

Dynamic cache © 2006 IBM Corporation 28

This slide shows the Cache Policies view. This view displays the cache policy files or cachespec.xml files that are being used for the selected cache instance. You can click on the links in this page to see details on generating each one of the cache entries. The information provided in this view is useful in determining whether the policies that you have defined on your server or within your application are being implemented correctly.

Section

Summary and Reference



The next section will summarize the points discussed in this module.

Summary

- Application Server service that provides caching functionality for J2EE applications
- Dynamic caching can be enabled and setup with minimal changes to your application
- The following data can be cached
 - ▶ Servlets/JSPs/Struts/Tiles
 - ▶ WebSphere command objects
 - ▶ Web Services responses
 - ▶ Java objects



In summary, the dynamic cache service is a WebSphere Application Server built-in service that provides caching functionality for J2EE applications. By using dynamic cache, you can enhance application performance by not having to regenerate dynamically created content each time that it is accessed. The dynamic caching solution provided by WebSphere Application Server has been designed to be as non-intrusive as possible. The service can often be utilized with only minimal changes to your application. Finally, dynamic caching can be used to store whole pages or fragments of pages for servlets, JSPs, Struts or Tiles, WebSphere command objects, Java objects, and Web Services responses.

Reference

- WebSphere Dynamic Cache: Improving J2EE application performance
 - ▶ <http://www.research.ibm.com/journal/sj/432/bakalova.pdf>
- WebSphere Application Server Information Center
 - ▶ <http://www.ibm.com/software/webservers/appserv/infocenter.html>