BEA WebLogic Server R
Performance Enhancements

Version: 10.3 Tech Preview

Document Date: October 2007

Table of Contents

Overview of Performance Enhancements

The following sections provide information on performance improvements for this release:

Clustering

The following sections provide detailed information on cluster performance improvements for this release:

Asynchronous HTTP Session replication - Description

This section describes the asynchronous HTTP Session replication (“AsyncRep”).

AsyncRep provides the option of choosing asynchronous session replication to the secondary server. It also provides the capability of throttling the maximum size of the queue that batches up session objects before the batched replication takes place.

Definition of New Terms, Acronyms and Abbreviations

Term	Definition
AsyncRep	The asynchronous replication of http sessions in general
WLS	WebLogic Server
MAN	Cluster topology in a metropolitan area network
WAN	Cluster topology in a wide area network
AsyncJDBC	The asynchronous persistence of session objects to a database
LAN	Cluster topology in a local area network (default topology).

AsyncRep. is used to specify asynchronous replication of data between a primary server and a secondary server. In addition, this option enables asynchronous replication of data between a primary server and a remote secondary server located in a different cluster according to a cluster topology of MAN.

Asynchronous HTTP Session replication - Usage

When you specify “async-replicated” or “async-replicated-if-clustered” as the PersistentStoreType for your application, session replication occurs asynchronously.

You can configure this parameter in WeblogicApplicationBean and the WeblogicWebAppBean or application.xml and weblogic.xml respectively. The application level value takes precedence over all webapp values. This joins other allowable values of replicated, jdbc, file, memory, replicated-if-clustered, and cookie.

You can also fine tune the batched replication by adjusting the SessionFlushThreshold on the ClusterMBean. This parameter exists on the ClusterMBean and is used for the WAN persistence in the same manner.

When a developer specifies a cluster type of either MAN, WAN, or LAN, session replication behavior can change slightly. The table below describes where the asynchronous replication occurs in each instance. Again, use the ClusterMBean parameters mentioned above for tuning this functionality.

Topology	**Replication occurring**
LAN	Replication to a secondary server within the same cluster occurs asynchronously with the “async-replication” setting in the webapp.
MAN	Replication to a secondary server in a remote cluster. This happens asynchronously with the “async-replication” setting in the webapp.
WAN	Replication to a secondary server within the cluster happens asynchronously with the “async-replication” setting in the webapp. Persistence to a database through a remote cluster happens asynchronously regardless of whether “async-replication” or “replication” is chosen.

When an administrator chooses to undeploy or redeploy the application, the sessions are as they are in the current replication system. The session is unregistered and removed from the queue of updates in the AsyncRep case. The session is also unregistered on the secondary server.

In the case where the administrator has performed a graceful operation to move the application to admin mode, the sessions are flushed causing replication to the secondary server. This flush causes the sessions to be flushed even if the secondary server is down, picking a new secondary if necessary.

Server shutdown or state change indicating failure triggers any batched up sessions to be replicated in order to prevent as much session loss as possible.

Asynchronous HTTP Session replication - Asynchronous Persistence to a database

When you specify “async-jdbc” as the PersistentStoreType for your application, session persistence occurs asynchronously. This can be configured using the WeblogicApplicationBean and the WeblogicWebAppBean or application.xml and weblogic.xml respectively.

You can also fine tune the batched replication by adjusting the SessionFlushThreshold and/or SessionFlushInterval on the ClusterMBean. These values already exist on the ClusterMBean and are currently used for the WAN persistence in the same manner. The SessionFlushThreshold determines how many session objects are put into the queue before the entire queue is batched up and replicated to the secondary server. By default this value is 100. The SessionFlushInterval is the defined time interval (in seconds) between batched replications of the session queue. The default value is 180 seconds.

When an administrator chooses to undeploy or redeploy the application, the sessions are handled as they are in the current replication system. The session is unregistered and removed from the queue of updates in the AsyncJDBC case. The session is also removed from the database. In the case where the administrator has performed a graceful operation to move the application to admin mode, the sessions are flushed to the database.

Deployment

The following sections contain the descriptions of the enhancements for deployment sub-system:

Generic Overrides

The following sections provide information on how to override application specific property files without having to crack the jar.

Overview

This feature allows you to place application specific files to be overridden into a new optional subdirectory in the existing plan directory structure (named "AppFileOverrides"). The presence or absence of this new optional subdirectory controls whether file overrides are enabled for the deployment. If the new subdirectory is present, an internal ClassFinder is added to the front of the application and module ClassLoaders for the deployment. As a result, the file override hierarchy rules follow the existing ClassLoader and resource loading rules/behaviours for applications.

Note: This mechanism is for overriding resources only and does not override classes.

These are application specific files and the contents are opaque to WLS, so the entire file contents are overridden when an override file is supplied.

The files placed in the AppFileOverrides subdirectory are staged and distributed along with the rest of the plan directory contents and are available on all of the targets.

Applications are then be able to load these files as resources using the current ClassLoader (for example, using ClassLoader.getResourceAsStream). This either finds the overridden files or the files packaged in the application depending on the configuration and whether overridden files are supplied.

For web applications, application file overrides only apply to the classpath related resources (which are in WEB-INF/classes and WEB-INF/lib) and do not apply to the resource path for the webapp. So overrides are seen by web applications using classloader.getResourceAsStream() to lookup resources, but overrides do not affect web application calls to ServletContext.getResourceAsStream().

Directory structure

The contents of the AppFileOverrides subdirectory use the existing plan directory structure and directory naming conventions that already exist for descriptor overrides. For more info on the directory naming conventions, refer to the documentation at:

http://e-docs.bea.com/wls/docs100/deployment/deployunits.html#wp1045820

Enabling file overrides causes a directory ClassFinder to be added to the application and module level ClassLoaders which point to the appropriate root directories within the AppFileOverrides subdirectory (which is in the plan directory). The ClassFinder inserted into the front of the application's ClassLoader is given AppDeploymentMBean.getLocalPlanDir + separator + "AppFileOverrides". The ClassFinder inserted into the front of the module's ClassLoaders is given the AppDeploymentMBean.getLocalPlanDir + separator + "AppFileOverrides" + separator + moduleURI.

For Example:

install-root/plan/AppFileOverrides/...	(Directory put in front of main apps classloader's classpath)
install-root/plan/AppFileOverrides/WebApp1.war/...	(Directory put in front of WebApp1.war classloader's classpath)
install-root/plan/AppFileOverrides/WebApp2.war/...	(Directory put in front of WebApp2.war classloader's classpath)

In order for this feature to be used, you must:

- Specify a plan for the deployment
- Specify a config-root within in the plan
- Provide a config-root/AppFileOverrides subdirectory.

Application Usage

It is important to note that the application controls the file contents and format and controls if/when the contents of the files are accessed by the application code.

The expectation is that this feature is primarily used by application code which has environment specific properties files, and that is loading those properties files as resources using the application's classloader.

For example, the application code may do the following:

```
Properties myAppProps = new Properties();
InputStream iostream =
Thread.currentThread().getContextClassLoader().getResourceAsStream("myCfg/myApp.properties");
myAppProps.load(iostream);
```

On-Demand Deployment

The following sections provide information on how to reduce startup time and memory usage by deploying internal apps on demand (first use).

Overview

There are many internal applications that are deployed during startup. These internal applications consume memory and require CPU time during deployment. This contributes to the WLS startup time and base memory footprint. Since not all of these internal applications are needed by every customer, WLS can be configured to wait and deploy these applications on the first access (on-demand) instead of always deploying them during server startup. This reduces startup time and memory footprint.

There are two different types of internal applications:

- User interfaces: This group includes the console, UDDI explorer, and WLS test client.
- All others: This group includes UDDI, Web Service async response, deployment service servlet, and management file distribution/Bootstrap servlet.

On-demand deployment works differently depending upon the type of internal application. For applications with a user interface, WLS traps the first access to the context path for the internal application (/console or /uddiexplorer), and displays a status page indicating that on-demand deployment is in progress. This page refreshs every 2 seconds and when the application completes deployment, the user is redirected to the internal application. This status page is displayed only on the first access of each application. Subsequent invokes do not deploy the application and go directly to the user interface for the internal application.

For applications without a user interface, WLS the first access to the context path for the internal application and deploys the internal application. The caller (a server or application) experiences a slight delay as the internal application is deployed. When the application is completely deployed, the servlet request proceeds and is handled by the internal application servlet.

Configuration

In a development domain, the default is set to use on-demand deployment for internal applications. In a production-mode domain, the default is to deploy internal applications as part of startup.

Use the **InternalAppsDeployOnDemandEnabled** attribute (added to the Domain MBean) to set the behavior of your WLS instance. The default is true for development domains and false in production-mode.

Support for JDK1.6

This release supports Sun JDK 1.6 which delivers superior performance compared to previous versions and leverages JDK6 delivered features such as:

- Thread synchronization improvements thus delivering increased scaling in the number of concurrent users supported and more reliable/stable code.
- Web container supports multiple scripting languages such as PHP, Groovy, and Ruby: Refer to http://jcp.org/en/jsr/detail?id=223.
- Allows hotswap of classes (without bouncing the class loaders in development mode) assuming that the class profile (new methods, method names) have not changed (only code in the method changed). The benefit is that it reduces redeployment effort (for iterative development, redeployment and restart).
- Clients for for this release require JDK5 or later for client JVMs and a server JVM on JDK6. WLS apps built using the development tools (eclipse, intelliJ, Workshop etc.) running in a separate JVM (such as JDK5) should work with this release.
- Compiler support. In this release, the JSP compiler uses the Java Compiler API instead of Javelin (the JSP compiler will no longer depend on Javelin framework java class bytecode generation feature) to generate byte code. This replacement does not expose any new public features and should not impact JSP files in existing applications. The rationale was to deliver a highly performing and reliable JSP compiler. See [JSP199], Java compiler API http://jcp.org/en/jsr/detail?id=199.

JSP Compiler

This section contains the descriptions of the JSP performance enhancements for this release:

Compress Html Template

The following sections provide information on how the Html Template in JSP files can be compressed to reduce the network traffic.

Overview

Whitespaces in template text of a JSP page are preserved by default. This preserves extraneous whitespaces in the response output although they do not impact the presentation result in the browser (except those in `<pre>` tag). In JSP 2.1 specification, a new JSP configuration element `trim-directive-whitespaces` is introduced to remove template text which only contains whitlespaces (see JSP.3.3.8 Removing whitespaces from template text). This feature is limited to remove whitespaces that can only removed from template text which contains only whitespaces. For the template text which is mixed, containing whitespaces and other characters, no whitespaces are removed. In Weblogic Server 10.3, Weblogic JSP compiler introduces an useful feature which can remove whitespaces in template text of a JSP page which does not impact the presentation result. By removing these whitespaces, network traffic can be reduced and performance inproved. For example:

```
<html>
```

```
        <body>
           <text>
           </text>
        </body>
    </html>
```

Can be compressed as:

```
    <html><body><text></text></body></html>
```

Configuration

The compress html template feature of JSP compiler can be turned on by deploy descriptor in the weblogic.xml as shown below:

```
    <weblogic-web-app>
        <jsp-descriptor>
            <compress-html-template>
                true
            </compress-html-template>
        </jsp-descriptor>
    </weblogic-web-app>
```

Note:

1. Once the `compress-html-template` feature is turned on, the `trim-directive-whitespaces` feature of JSP 2.1 is automatically turned on.
2. If there are any `<pre>` tags in the JSP page, do not use this feature. You may not get the expected presentation result

Using a Static String as a Template

The following sections provide information on how to use a static string inside a JSP expression as a template.

Overview

In the JSP specifiaction, an expression element in a JSP page is a scripting language expression that is evaluted and the result is coerced to a String. The result is subsequently emitted into the current out JspWiter object (see JSP 1.12.3 Expressions). Unfortunately this means that the JSP compiler will not parse/analyze the expression and just output them as the parameter of JspWriter.print() method. This limitation prevents JSP compiler from optimizing the expression. In Weblogic Server 10.3, Weblogic JSP compiler introduces a new feature which can optimize the static string inside JSP expression. For example:

```
    <%= "<INPUT type=\"hidden\" name=\"name\" value=\"" +
    userBean.getName() + "\">" %>
```

The old generated source code is like this:

```
out.print((new StringBuilder()).append("<INPUT
type=\"hidden\" name=\"itemId\"
value=\"").append(userBean.getName()).append("\">").toStrin
g());
```

It's not efficient to use a StringBuilder to concatenate static string especially for the JSP page which is under heavy load requests.

With the new expression optimization feature, the generated source code is like this:

```
{_writeText(response, out, _wl_block9, _wl_block9Bytes);}
out.print(userBean.getName());
{_writeText(response, out, _wl_block10, _wl_block10Bytes);}
```

The static String is outputted directly instead of concatenating with `userBean.getName()`.

Note: If the Java expression contains brackets other than the ones for methods, or contains operator having lower priority than '+', then optimization on Java expression does not happen.

Configuration

This feature is enabled by default. If you encounter issues with this configuration, set system property 'weblogic.jsp.expression.optimizeDisable' to 'true'.

Entity Bean Load with Relationship Caching

This feature allows customers to enable relationship-caching in ejbLoad. In the preceding release, for container-managed entity beans, relationship-caching is only supported in query. But for the ejbLoad, there is no way to support relationship-caching. It only loads the bean itself when DB is queried. The feature provides a way to get the benefit of relation-ship caching. If enabling this feature, the ejbLoad uses the relationship-caching defined in the primary finder.

The feature is only for container-managed entity bean. For many cases, when the owner bean reloads, the related beans are also required to load in the same transaction. This leads to multiple database queries. For these cases, enabling this feature loads the owner bean and related beans in one database query. Since the database query is always the bottle-neck in entity application, most situations will experience a performance improvement. In some situations, where the feature is used excessively, performance may degrade.

To enable this feature:

1. Applies only to a container-managed entity bean
2. Set finder-load-bean to true
3. The relationship-caching to be supported in the ejbLoad must be specified in the primary finder.

For this feature, add a new element `load-by-finder` in weblogic-cmp-jar.xml descriptor. The element is described as the following table:

Load-by-finder

Range of values	true | false
Default value	false
Parent elements	weblogic-rdbms-bean

For example: A student and course have one-many relationship. To enable the feature for this case:

- Specify the relationship for these beans in the `ejb-jar.xml`
- Specify a relationship-caching in `weblogic-cmp-jar.xml`
- Specify the relationship-caching in the method "`findByPrimaryKey`" in `weblogic-cmp-jar.xml`
- Setting the element "`load-by-finder`" to `true` in `weblogic-cmp-jar.xml`.

The following is an example `weblogic-cmp-jar.xml` file:

```
<weblogic-rdbms-bean>
    …
    <!— specifying relationship-caching name for cmr field: courses  -->
    <relationship-caching>
        <caching-name>courseCache</caching-name>
        <caching-element>
            <cmr-field>courses</cmr-field>
        </caching-element>
    </relationship-caching>

    <!—specifying relationship-caching for primary finder -->
    <weblogic-query>
        <query-method>
            <method-name>findByPrimaryKey</method-name>
            <method-params>
                <method-param>
                    java.lang.String
                </method-param>
            </method-params>
        </query-method>
        <ejb-ql-query>
            <caching-name>courseCache</caching-name>
        </ejb-ql-query>
    </weblogic-query>

    <!—enabling the relaship-caching defined in primary finder  in ejbLoad -->
    <load-by-finder>true</load-by-finder>
    …
</weblogic-rdbms-bean>
```