IBM WebSphere Application Server - Express for Distributed Platforms, Version 8.5

Tuning various types of applications

Note

Before using this information, be sure to read the general information under “Notices” on page 99.

Compilation date: June 1, 2012

Contents

How to send your comments

Your feedback is important in helping to provide the most accurate and highest quality information.

- To send comments on articles in the WebSphere Application Server Information Center
  1. Display the article in your Web browser and scroll to the end of the article.
  2. Click on the **Feedback** link at the bottom of the article, and a separate window containing an email form appears.
  3. Fill out the email form as instructed, and submit your feedback.
- To send comments on PDF books, you can email your comments to: **wasdoc@us.ibm.com**.

  Your comment should pertain to specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. Be sure to include the document name and number, the WebSphere Application Server version you are using, and, if applicable, the specific page, table, or figure number on which you are commenting.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer. When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about your comments.

Using this PDF

Links

Because the content within this PDF is designed for an online information center deliverable, you might experience broken links. You can expect the following link behavior within this PDF:

- Links to Web addresses beginning with http:// work.
- Links that refer to specific page numbers within the same PDF book work.
- The remaining links will *not* work. You receive an error message when you click them.

Print sections directly from the information center navigation

PDF books are provided as a convenience format for easy printing, reading, and offline use. The information center is the official delivery format for IBM WebSphere Application Server documentation. If you use the PDF books primarily for convenient printing, it is now easier to print various parts of the information center as needed, quickly and directly from the information center navigation tree.

To print a section of the information center navigation:

1. Hover your cursor over an entry in the information center navigation until the **Open Quick Menu** icon is displayed beside the entry.
2. Right-click the icon to display a menu for printing or searching your selected section of the navigation tree.
3. If you select **Print this topic and subtopics** from the menu, the selected section is launched in a separate browser window as one HTML file. The HTML file includes each of the topics in the section, with a table of contents at the top.
4. Print the HTML file.

For performance reasons, the number of topics you can print at one time is limited. You are notified if your selection contains too many topics. If the current limit is too restrictive, use the feedback link to suggest a preferable limit. The feedback link is available at the end of most information center pages.

Chapter 1. Tuning Application profiling

This page provides a starting point for finding information about application profiling, a WebSphere extension for defining strategies to dynamically control concurrency, prefetch, and read-ahead.

Application profiling and access intent provide a flexible method to fine-tune application performance for enterprise beans without impacting source code. Different enterprise beans, and even different methods in one enterprise bean, can have their own intent to access resources. Profiling the components based on their access intent increases performance in the application server run time.

Application profiling performance considerations

Application profiling enables assembly configuration techniques that improve your application run time, performance and scalability. You can configure tasks that identify incoming requests, identify access intents determining concurrency and other data access characteristics, and profiles that map the tasks to the access intents.

The capability to configure the application server can improve performance, efficiency and scalability, while reducing development and maintenance costs. The application profiling service has no tuning parameters, other than a checkbox for disabling the service if the service is not necessary. However, the overhead for the application profile service is small and should not be disabled, or unpredictable results can occur.

Access intents enable you to specify data access characteristics. The WebSphere runtime environment uses these hints to optimize the access to the data, by setting the appropriate isolation level and concurrency. Various access intent hints can be grouped together in an access intent policy.

In the product, it is recommended that you configure bean level access intent for loading a given bean. Application profiling enables you to configure multiple access intent policies on the entity bean, if desired. Some callers can load a bean with the intent to read data, while others can load the bean for update. The capability to configure the application server can improve performance, efficiency, and scalability, while reducing development and maintenance costs.

Access intents enable the EJB container to be configured providing optimal performance based on the specific type of enterprise bean used. Various access intent hints can be specified declaratively at deployment time to indicate to WebSphere resources, such as the container and persistence manager, to provide the appropriate access intent services for every EJB request.

The application profiling service improves overall entity bean performance and throughput by fine tuning the run time behavior. The application profiling service enables EJB optimizations to be customized for multiple user access patterns without resorting to "worst case" choices, such as pessimistic update on a bean accessed with the findByPrimaryKey method, regardless of whether the client needs it for read or for an update.

Application profiling provides the capability to define the following hierarchy: **Container-Managed Tasks** > **Application Profiles** > **Access Intent Policies** > **Access Intent Overrides**. Container-managed tasks identify units of work (UOW) and are associated with a method or a set of methods. When a method associated with the task is invoked, the task name is propagated with the request. For example, a UOW refers to a unique path within the application that can correspond to a transaction or ActivitySession. The name of the task is assigned declaratively to a Java EE client or servlet, or to the method of an enterprise bean. The task name identifies the starting point of a call graph or subgraph; the task name flows from the starting point of the graph downstream on all subsequent IIOP requests, identifying each subsequent invocation along the graph as belonging to the configured task. As a best practice, wherever a UOW starts, for example, a transaction or an ActivitySession, assign a task to that starting point.

The application profile service associates the propagated tasks with access intent policies. When a bean is loaded and data is retrieved, the characteristics used for the retrieval of the data are dictated by the application profile. The application profile configures the access intent policy and the overrides that should be used to access data for a specific task.

Access intent policies determine how beans are loaded for specific tasks and how data is accessed during the transaction. The access intent policy is a named group of access intent hints. The hints can be used, depending on the characteristics of the database and resource manager. Various access intent hints applied to the data access operation govern data integrity. The general rule is, the more data integrity, the more overhead. More overhead causes lower throughput and the opportunity for simultaneous data access from multiple clients.

If specified, access intent overrides provide further configuration for the access intent policy.

Best practices

Application profiling is effective in a variety of different scenarios including:

- **The same bean is loaded with different data access patterns**

  The same bean or set of beans can be reused across applications, but each of those applications has differing requirements for the bean or for beans within the invocation graph. One application can require that beans be loaded for update, while another application requires beans be loaded for read only. Application profiling enables deploy time configuration for beans to distinguish between EJB loading requirements.
- **Different clients have different data access requirements**

  The same bean or set of beans can be used for different types of client requests. When those clients have different requirements for the bean, or for beans within the invocation graph, application profiling can be used to tailor the bean loading characteristics to the requirements of the client. One client can require beans be loaded for update, while another client requires beans be loaded for read only. Application profiling enables deploy time configuration for beans to distinguish between EJB loading requirements.

Monitoring tools

You can use the Tivoli Performance Viewer, database and logs as monitoring tools.

You can use the Tivoli Performance Viewer to monitor various metrics associated with beans in an application profiling configuration. The following sections describe at a high level the Tivoli Performance Viewer metrics that reflect changes when access intents and application profiling are used:

- Collection scope:

  The enterprise beans group contains EJB life cycle information, either a cumulative value for a group of beans, or for specific beans. You can monitor this information to determine the difference between using the ActivitySession scope versus the transaction scope. For the transaction scope, depending on how the container transactions are defined, activates and passivates can be associated with method invocations. The application could use the ActivitySession scope to reduce the frequency of activates and passivates. For more information, see the topic, Using the ActivitySession service.
- Collection increment:

  The enterprise beans group contains EJB life cycle information, either a cumulative value for a group of beans, or for specific beans. You can monitor `Num Activates` to watch the number of enterprise beans activated for a particular findByPrimaryKey operation. For example, if the collection increment is set to 10, rather than the default 25, the `Num Activates` value shows 10 for the initial findByPrimaryKey, prior to any result set iterator runs. If the number of activates rarely exceeds the collection increment, consider reducing the collection increment setting.
- Resource manager prefetch increment:

The resource manager prefetch increment is a hint acted upon by the database engine to depend upon the database. The Tivoli Performance Viewer does not have a metric available to show the effect of the resource manager prefetch increment setting.

- Read ahead hint:

  The enterprise beans group contains EJB life cycle information, either a cumulative value for a group of beans, or for specific beans. You can monitor `Num Activates` to watch the number of enterprise beans activated for a particular request. If a read ahead association is not in use, the `Num Activates` value shows a lower initial number. If a read ahead association is in use, the `Num Activates` value represents the number of activates for the entire call graph.

Database tools are helpful in monitoring the different bean loading characteristics that introduce contention and concurrency issues. These issues can be solved by application profiling, or can be made worse by the misapplication of access intent policies.

Database tools are useful for monitoring locking and contention characteristics, such as locks, deadlocks and connections open. For example, for locks the DB2 Snapshot Monitor can show statistics for lock waits, lock time-outs and lock escalations. If excessive lock waits and time-outs are occurring, application profiling can define specific client tasks that require a more string level of locking, and other client tasks that do not require locking. Or, a different access intent policy with less restrictive locking could be applied. Once applying this configuration change, the snapshot monitor shows less locking behavior. Refer to information about the database you are using on how to monitor for locking and contention.

The application server logs can be monitored for information about rollbacks, deadlocks, and other data access or transaction characteristics that can degrade performance or cause the application to fail.

Chapter 2. Tuning Client applications

This page provides a starting point for finding information about application clients and client applications. Application clients provide a framework on which application code runs, so that your client applications can access information on the application server.

For example, an insurance company can use application clients to help offload work on the server and to perform specific tasks. Suppose an insurance agent wants to access and compile daily reports. The reports are based on insurance rates that are located on the server. The agent can use application clients to access the application server where the insurance rates are located.

Adding tracing and logging for stand-alone clients

You can add tracing and logging to help analyze performance and diagnose problems with stand-alone clients.

About this task

This information applies to the following WebSphere® Application Server stand-alone clients:

- Thin Client for JMS with WebSphere Application Server
- Thin Client for EJB with WebSphere Application Server
- Thin Client for JAX-WS with WebSphere Application Server
- Thin Client for JAX-RPC with WebSphere Application Server

Procedure

- To enable trace, use either a long form or short form system property.

  Note: Trace settings are determined from the system property values the first time that a WebSphere Application Server client is called. The trace settings are then fixed. Therefore, any subsequent changes to the system property settings do not change the trace settings that the WebSphere Application Server client uses.

  For further information, see the information on the trace user interface for stand-alone clients.
- To enable First Failure Data Capture (FFDC), use either a long or short form system property.

  Note: FFDC settings are determined from the system property values the first time that a WebSphere Application Server client performs an FFDC. The FFDC settings are then fixed. Therefore, any subsequent changes to the system property settings do not change the FFDC settings that the WebSphere Application Server client uses.

  For further information, see the information on the First Failure Data Capture user interface for stand-alone clients.

Chapter 3. Tuning Data access resources

This page provides a starting point for finding information about data access. Various enterprise information systems (EIS) use different methods for storing data. These backend data stores might be relational databases, procedural transaction programs, or object-oriented databases.

The flexible IBM® WebSphere Application Server provides several options for accessing an information system backend data store:

- Programming directly to the database through the JDBC 4.0 API, JDBC 3.0 API, or JDBC 2.0 optional package API.
- Programming to the procedural backend transaction through various J2EE Connector Architecture (JCA) 1.0 or 1.5 compliant connectors.
- Programming in the bean-managed persistence (BMP) bean or servlets indirectly accessing the backend store through either the JDBC API or JCA-compliant connectors.
- Using container-managed persistence (CMP) beans.
- Using the IBM data access beans, which also use the JDBC API, but give you a rich set of features and function that hide much of the complexity associated with accessing relational databases.

Service Data Objects (SDO) simplify the programmer experience with a universal abstraction for messages and data, whether the programmer thinks of data in terms of XML documents or Java objects. For programmers, SDOs eliminate the complexity of the underlying data access technology, such as, JDBC, RMI/IIOP, JAX-RPC and JMS, and message transport technology, such as, (java.io.Serializable, DOM Objects, SOAP and JMS).

Tuning data

Tuning connection pools

Using connection pools helps to both alleviate connection management overhead and decrease development tasks for data access. Each time an application attempts to access a backend store (such as a database), it requires resources to create, maintain, and release a connection to that datastore. To mitigate the strain this process can place on overall application resources, the application server enables administrators to establish a pool of backend connections that applications can share on an application server. Connection pooling spreads the connection overhead across several user requests, thereby conserving application resources for future requests.

About this task

yuConnection pooling can improve the response time of any application that requires connections, especially Web-based applications. When you make a request over the web to a resource, the resource accesses a data source. Because you connect and disconnect frequently with applications on the Internet, the application requests for data access can surge to considerable volume. Consequently, the total overhead for a datastore can quickly become high for Web-based applications, causing performance to deteriorate. When connection pooling capabilities are used, however, web applications can realize performance improvements of up to 20 times the normal results.

Note: Connection pooling is not supported in an application client. The application client calls the database directly and does not go through a data source. If you want to use the getConnection() request from the application client, configure the JDBC provider in the application client deployment descriptors, using Rational® Application Developer or an assembly tool. The connection is established between application client and the database. Application clients do not have a connection pool, but you can configure JDBC provider settings in the client deployment descriptors.

Procedure

- Prevent a connection deadlock. Deadlock can occur if the application requires more than one concurrent connection per thread, and the database connection pool is not large enough for the number of threads. Suppose each of the application threads requires two concurrent database connections and the number of threads is equal to the maximum connection pool size. Deadlock can occur when both of the following are true:
  - Each thread has its first database connection, and all are in use.
  - Each thread is waiting for a second database connection, and none would become available since all threads are blocked.

  To prevent the deadlock in this case, increase the maximum connections value for the database connection pool by at least one. This ensures that at least one of the waiting threads obtains a second database connection and avoids a deadlock scenario.

  For general prevention of connection deadlock, code your applications to use only one connection per thread. If you code the application to require *C* concurrent database connections per thread, the connection pool must support at least the following number of connections, where *T* is the maximum number of threads:

  ```
  T * (C - 1) + 1
  ```

  The connection pool settings are directly related to the number of connections that the database server is configured to support. If you increase the maximum number of connections in the pool and the corresponding settings in the database are not increased accordingly, the application might fail. The resulting SQL exception errors are displayed in the following locations:

  - the `stderr.log` file

  One of the most common causes of connection deadlock is the use of the same connection pool by both servlets and by Enterprise JavaBeans (EJBs), and where the servlet directly or indirectly invokes the bean. For example, a servlet that obtains a JMS connection from the connection pool, sends a message to a Message Driven Bean (MDB) and waits for a reply. The MDB is configured to use the same connection pool as the servlet, therefore, another connection from the pool is required for the MDB to send a reply to the servlet. Servlets and enterprise beans do not share the same connection pool. This is a classic case of concurrent (C) threads, where C=2 and *T* is the maximum size of the servlet and EJB thread pools.
- Disable connection pooling.
  - For relational resource adapters (RRAs), add the `disableWASConnectionPooling` custom property for your data sources.
    1. Click **JDBC** > **Data sources**.
    2. Click on the name of the data source that you want to configure.
    3. Click **Custom properties** under the **Additional Properties** heading.
    4. Click **New**.
    5. Complete the required fields with the following information:
       - **Name**: disableWASConnectionPooling
       - **Value**: true
  - For other resource adapters, consult with the binding specifications for that resource adapter to configure your applications to disable connection pooling.
    1. Programmatically disable connection pooling through the resource adapter.
    2. The application server leverages the following code to detect the javax.resource.NotSupportedException exception and disable connection pooling:

       ```
       _managedFactory.matchManagedConnections(s,subject,cri); // 169059 174269     }
        catch(javax.resource.NotSupportedException e){
       ```
- Enable deferred enlistment. In the application server environment, deferred enlistment refers to the technique in which the application server waits until the connection is used before the connection is enlisted in the application's unit of work (UOW) scope.

Consider the following illustration of deferred enlistment:

- An application component that uses deferred enlistment calls the `getConnection` method from within a global transaction.
- The application component does not immediately use the connection.
- When the application issues the call for initial use of the connection, the transaction manager intercepts the call.
- The transaction manager enlists the XA resource for the connection and calls the `XAResource.start` method.
- The connection manager associated with the XA resource sends the call to the database.

Given the same scenario, but the application component does not use deferred enlistment, the component container immediately enlists the connection in the transaction. Thus the application server incurs, for no purpose, an additional load of all of the overhead associated with that transaction. For XA connections, this overhead includes the two phase commit (2PC) protocol to the resource manager.

Deferred enlistment offers better performance in the case where a connection is obtained, but not used, within the UOW scope. The technique saves the cost of transaction participation until the UOW in which participation must occur.

Check with your resource adapter provider if you need to know if the resource adapter provides this functionality. The application server relational resource adapter automatically supports deferred enlistment.

Incorporating deferred enlistment in your code:

The Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) Version 1.5 and later specification calls the deferred enlistment technique lazy transaction enlistment optimization. This support comes through a marker interface (LazyEnlistableManagedConnection) and a method on the connection manager (LazyEnlistableConnectionManager()):

```
package javax.resource.spi; import javax.resource.ResourceException; import
 javax.transaction.xa.Xid;  interface LazyEnlistableConnectionManager { // application server     void
 lazyEnlist(ManagedConnection) throws ResourceException; }  interface LazyEnlistableManagedConnection
{ // resource adapter }
```

- Control connection pool sharing. You can use the defaultConnectionTypeOverride, or globalConnectionTypeOverride connection pool custom property for a particular connection factory or data source to control connection sharing:
  - The defaultConnectionTypeOverride property changes the default sharing value for a connection pool. This property enables you to control connection sharing for direct queries. If resource references are configured for this data source or connection factory the resource reference's configurations take precedence over the defaultConnectionTypeOverride property settings. For example, if an application is doing direct queries and unshared connections are needed, set the defaultConnectionTypeOverride property to `unshared`.
  - The value specified for the globalConnectionTypeOverride custom property takes precedence over all of the other connection sharing settings. For example if you set this property to `unshared`, all connection requests are unshared for both direct queries and resource reference lookups. This property provides you with a quick way to test the consequences of moving all connections for a particular data source or connection factory to unshared or shared without changing any resource reference setting.

  If you specify values for both the defaultConnectionTypeOverride and the globalConnectionTypeOverride properties, only the values specified for the globalConnectionTypeOverride property are used to determine connection sharing type.

  To add these new custom properties to the settings for a data source or connection factory connection pool, a new connection pool custom property must be created. To add one of these properties to a data source, use the administrative console. Click **Resources** > **JDBC** > **Data sources**, select your data source from the list, and click **Additional properties** > **Connection pool properties** > **Connection pool custom properties** > **New**. For other J2C or JMS connection factories, navigate to the connection factory definition in the administrative console. Then select **Additional Properties** > **Connection pool**

> **Connection pool custom properties > New**. Now specify `defaultConnectionTypeOverride` or `globalConnectionTypeOverride` in the **Name** field and `shared` or `unshared` in the **Value** field.

Important: The properties must be set in the **Connection pool custom properties** and NOT the general **Custom propeties** on the data source or connection factory.

- Discard connections.

  Reap time and unused timeout settings do not cause the idle or unused connections to be discarded if the servant region is idle. This situation might cause some DB2 connections to be held longer than is necessary.

  If you prefer to have the connections discarded at the time specified by a combination of reaper time and unused timeout settings, even if this preference might cause an idle servant region to become active again, you can add the nondeferredreaper custom property to your JDBC driver provider data source settings. When you add this custom property, connections are discarded at the time specified by a combination of reaper time and unused timeout settings.

  To add this custom property to your JDBC driver provider data source settings, in the administrative console, click **Resources** > **JDBC providers** > **DB2 Universal JDBC Driver Provider** > **Data sources** > ***data_source*** > **Custom properties** > **New**. Then specify `nondeferredreaper` in the **Name** field, `true` in the **Value** field, and `java.lang.Boolean` in the **Type** field. This new setting does not go into effect until you restart the server that is using this data source.

  gotcha: Activating an idle servant region for the sole purpose of discarding unused connection, might cause additional and sometimes undesirable CPU usage. Also, the following warning message might be logged and should be ignored:

  ```
  DSRA8200W: DataSource Configuration:
  DSRA8020E: Warning: The property 'nondeferredreaper' does not exist on the DataSource
  classcom.ibm.db2.jcc.DB2ConnectionPoolDataSource.
  ```

- Purge connection pools based on the purge policy

  When the connection pool error detection model is configured to exception mapping, the stale connection exception indicates that the connection is no longer valid.

  Typically, when a StaleConnectionException results from the exception mapping process a connection error event is fired and subsequently the connection pool gets purged. However, in this case, the SCE is instantiated, the existing code does not have the functionality to fire the connection error event and the pool does not get purged When connections are terminated on the database side when requesting a new connection, the driver throws a XAException. If the errorDetectionModel=ExceptionMapping results, a ConnectionErrorEvent will be fired so that the connection pool gets purged based on the purge policy.

  To add this custom property to your JDBC driver provider data source settings, in the administrative console, click **Resources** > **JDBC providers** > **DB2 Universal JDBC Driver Provider** > **Data sources** > ***data_source*** > **Custom properties** > **New**. Then specify `fireCEEventOnSCE` in the **Name** field, `true` in the **Value** field, and `java.lang.Boolean` in the **Type** field. This new setting does not go into effect until you restart the server that is using this data source.

  The WebSphere RRA code has been changed so that the pool is purged properly upon a StaleConnectionException when using ExceptionMapping as the errorDetection model.

Connection pool custom properties

You can use the custom properties page to define the following connection pool custom properties:

- “defaultConnectionTypeOverride”
- “globalConnectionTypeOverride” on page 11

defaultConnectionTypeOverride

You can use the defaultConnectionTypeOverride connection pool custom property for a particular connection factory or data source to control connection sharing.

The defaultConnectionTypeOverride property changes the default sharing value for a connection pool. This property enables you to control connection sharing for direct look-ups. If resource references are configured for a data source or connection factory they take precedence over this property and the resource reference settings are used. For example, if an application is doing direct look-ups, and you do not want unshared connections, set this property to unshared.

Information	Value
Data Type	String
Value	unshared, shared

globalConnectionTypeOverride

You can use the globalConnectionTypeOverride connection pool custom property to globally control connection sharing for a particular connection factory or data source.

The value specified for the globalConnectionTypeOverride custom property takes precedence over all of the other connection sharing settings. For example, if you set this property to unshared, all connection requests are unshared for both direct look-ups and resource reference lookups.

This property provides you with a quick way to test the consequences of moving all connections for a particular data source or connection factory to unshared or shared without changing the resource reference settings.

If you specify values for both the defaultConnectionTypeOverride and the globalConnectionTypeOverride properties, only the value that is specified for the globalConnectionTypeOverride property is used to determine connection sharing type.

Information	Value
Data Type	String
Value	unshared, shared

For more information on how these properties are used, see the topic, Tuning connection pools.

Throttling inbound message flow for JCA 1.5 message-driven beans

This topic describes how to throttle message delivery for message-driven beans (MDB) which are deployed as message endpoints for Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) Version 1.5 inbound resource adapters.

Before you begin

The throttling of messages as described in this topic does not apply to the two JCA 1.5-compliant messaging providers that are supplied with WebSphere Application Server:

- The default messaging provider.
- The WebSphere MQ messaging provider.

For the default messaging provider, you configure message throttling as described in the related tasks. For the WebSphere MQ messaging provider, you configure message throttling by setting the **maximum server sessions** property on the WebSphere MQ messaging provider activation specifications panel, or the **maxPoolSize** property when using the createWMQActivationSpec or modifyWMQActivationSpec **wsadmin** commands.

If you have a third-party JCA 1.5-compliant JMS messaging provider, check with your supplier to see whether the method of message throttling described in this topic is appropriate for their messaging provider.

About this task

For installations that use resource adapters that implement the Java EE Connector Architecture (JCA) Version 1.5 message delivery support, the WebSphere Application Server provides message throttling support to control the delivery of messages to endpoint message-driven beans (MDB). You can use this support to avoid overloading the server with a flood of inbound messages.

Message delivery is throttled on a message-driven bean basis by limiting the maximum number of endpoint instances that can be created by the adapter that the MDB is bound to. When the adapter attempts to create an endpoint instance, a proxy for the MDB instance is created and returned as allowed by the JCA 1.5 architecture. There is a one-to-one correspondence between proxies and MDB instances, and like the MDB instances, the proxies are pooled based on the minimum and maximum pool size values associated with the message-driven bean. Throttling is performed through the management of the proxy pool.

At the time the adapter attempts to create an endpoint, if the number of endpoint proxies currently created is equal to the maximum size of the pool, adapter *createEndPoint* processing returns an `Unavailable Exception`. When this displays, the adapter does not issue any more createEndPoint() requests until it has released at least one endpoint back to the server for reuse. Thus, installations can control the throttling of message delivery to a JCA 1.5 MDB based on the setting of the maximum size of the pool associated with each JCA 1.5 message-driven bean.

You can specify the pool size by using the com.ibm.websphere.ejbcontainer.poolsize JVM system property to define the minimum and maximum pool size of stateless, message-driven, and entity beans. For a message-driven bean that supports JCA 1.5, the maximum pool size value specified limits how many message endpoint instances can be created for that message-driven bean. For example, if the installation sets the maximum size of a JCA 1.5 MDB pool to 5, then at most 5 messages can be concurrently delivered to 5 instances of the message-driven bean. This property can be specified using the **wsadmin** scripting tool or by specifying it under the administrative console as an environmental variable.

Procedure

1. Open the administrative console.
2. Select **Servers** > **Server Types** > **WebSphere application server** > ***server_name***.
3. Under **Server Infrastructure**, expand **Java and Process Management** >**Process Definition**.
4. Under Additional Properties, select **Java Virtual Machine**.
5. Under Additional Properties, select **Custom Properties**.
6. Select **New**. A panel with three **General Properties** fields displays. This is where you set the property.
7. In the **Name** field, enter `com.ibm.websphere.ejbcontainer.poolsize`.
8. To complete the **Value** field, see the EJB container system properties for values.
9. After defining the value of the property, click **OK**. You are now prompted to save the changes you have made.
10. Click **Save**.

Database performance tuning

Database performance tuning can dramatically affect the throughput of your application. For example, if your application requires high concurrency (multiple, simultaneous interactions with backend data), an improperly tuned database can result in a bottleneck. Database access threads accumulate in a backlog when the database is not configured to accept a sufficient number of incoming requests.

Because WebSphere Application Server supports the integration of many different database products, each one with unique tuning configurations, consult your database vendor documentation for comprehensive tuning information. This information center provides introductory material in the following topics:

- DB2® tuning parameters

DB2 tuning parameters

Read this topic for parameters that you can configure for better database performance.

For complete DB2 tuning information, refer to the *DB2 UDB Administration Guide: Performance* document.

AIX For more information about using AIX® with DB2 see the topic Tuning AIX systems.

DB2 logging:

DB2 has corresponding log files for each database that provides services to administrators, including viewing database access and the number of connections. For systems with multiple hard disk drives, you can gain large performance improvements by setting the log files for each database on a different hard drive from the database files.

- How to view or set: At a DB2 command prompt, issue the command: `db2 update db cfg for [database_name] using newlogpath [fully_qualified_path]`.
- Default value: Logs reside on the same disk as the database.
- Recommended value: Use a separate high-speed drive, preferably performance enhanced through a redundant array of independent disk (RAID) configuration.

DB2 configuration advisor:

Located in the DB2 Control Center, this advisor calculates and displays recommended values for the DB2 buffer pool size, the database, and the database manager configuration parameters, with the option of applying these values. See more information about the advisor in the online help facility within the Control Center.

Number of connections to DB2 - MaxAppls and MaxAgents:

When configuring the data source settings for the databases, confirm the DB2 MaxAppls setting is greater than the maximum number of connections for the data source. If you are planning to establish clones, set the MaxAppls value as the maximum number of connections multiplied by the number of clones. The same relationship applies to the session manager number of connections. The MaxAppls setting must be equal to or greater than the number of connections. If you are using the same database for session and data sources, set the MaxAppls value as the sum of the number of connection settings for the session manager and the data sources.

For example, MaxAppls = (number of connections set for the data source + number of connections in the session manager) multiplied by the number of clones.

After calculating the MaxAppls settings for the WebSphere Application Server database and each of the application databases, verify that the MaxAgents setting for DB2 is equal to or greater than the sum of all of the MaxAppls values. For example, MaxAgents = sum of MaxAppls for all databases.

DB2 buffpage:

Improves database system performance. Buffpage is a database configuration parameter. A buffer pool is a memory storage area where database pages containing table rows or index entries are temporarily read and changed. Data is accessed much faster from memory than from disk.

- How to view or set: To view the current value of buffpage for database x, issue the DB2 command `get db cfg for x` and look for the value BUFFPAGE. To set BUFFPAGE to a value of n, issue the DB2 command `update db cfg for x` using BUFFPAGE n and set NPAGES to -1 as follows:

```
db2   <-- go to DB2 command mode, otherwise the following "select" does not work as is
    connect to x    <-- (where x is the particular DB2 database name)
    select * from syscat.bufferpools
       (and note the name of the default, perhaps: IBMDEFAULTBP)
       (if NPAGES is already -1, there is no need to issue following command)
    alter bufferpool IBMDEFAULTBP size -1
    (re-issue the above "select" and NPAGES now equals -1)
```

  You can collect a snapshot of the database while the application is running and calculate the buffer pool hit ratio as follows:
  1. Collect the snapshot:
     a. Issue the update monitor switches using bufferpool on command.
     b. Make sure that bufferpool monitoring is on by issuing the get monitor switches command.
     c. Clear the monitor counters with the reset monitor all command.
  2. Run the application.
  3. Issue the get snapshot for all databases command prior to all applications disconnect from the database, otherwise statistics are lost.
  4. Issue the update monitor switches using bufferpool off command.
  5. Calculate the hit ratio by looking at the following database snapshot statistics:
     – Buffer pool data logical reads
     – Buffer pool data physical reads
     – Buffer pool index logical reads
     – Buffer pool index physical reads
- Default value: 250
- Recommended value: Continue increasing the value until the snapshot shows a satisfactory hit rate.

The buffer pool hit ratio indicates the percentage of time that the database manager did not need to load a page from disk to service a page request. That is, the page is already in the buffer pool. The greater the buffer pool hit ratio, the lower the frequency of disk input and output. Calculate the buffer pool hit ratio as follows:
- P = buffer pool data physical reads + buffer pool index physical reads
- L = buffer pool data logical reads + buffer pool index logical reads
- Hit ratio = (1-(P/L)) * 100%

DB2 query optimization level:

Sets the amount of work and resources that DB2 puts into optimizing the access plan. When a database query runs in DB2, various methods are used to calculate the most efficient access plan. The range is from 0 to 9. An optimization level of 9 causes DB2 to devote a lot of time and all of its available statistics to optimizing the access plan.
- How to view or set: The optimization level is set on individual databases and can be set with either the command line or with the DB2 Control Center. Static SQL statements use the optimization level that is specified on the `prep` and `bind` commands. If the optimization level is not specified, DB2 uses the default optimization as specified by the dft_queryopt setting. Dynamic SQL statements use the optimization class that is specified by the current query optimization special register, which is set using the SQL Set statement. For example, the following statement sets the optimization class to 1:

```
Set current query optimization = 1
```

  If the current query optimization register is not set, dynamic statements are bound using the default query optimization class.
- Default value: 5
- Recommended value: Set the optimization level for the needs of the application. Use high levels only when there are very complicated queries.

DB2 reorgchk:

Obtains the current statistics for data and rebinding. Use this parameter because SQL statement performance can deteriorate after many updates, deletes or inserts.

- How to view or set: Use the DB2 reorgchk update statistics on table all command to perform the `runstats` operation on all user and system tables for the database to which you are currently connected. Rebind packages using the `bind` command. If statistics are available, issue the db2 -v "select tbname, nleaf, nlevels, stats_time from sysibm.sysindexes" command on DB2 CLP. If no statistic updates exist, nleaf and nlevels are -1, and stats_time has an empty entry (for example: "-"). If the runstats command was previously run, the real-time stamp from completion of the runstats operation also displays under stats_time. If you think the time shown for the previous runstats operation is too old, run the runstats command again.
- Default value: None
- Recommended value: None

DB2 locktimeout:

Specifies the number of seconds that an application waits to obtain a lock. Setting this property helps avoid global deadlocks for applications.

- How to view or set: To view the current value of the lock timeout property for database `xxxxxx`, issue the DB2 `get db cfg for xxxxxx` command and look for the value, LOCKTIMEOUT. To set LOCKTIMEOUT to a value of `n`, issue the DB2 `update db cfg for xxxxxx` command using `LOCKTIMEOUT n`, where `xxxxxx` is the name of the application database and `n` is a value between 0 and 30 000 inclusive.
- Default value: -1, meaning lock timeout detection is turned off. In this situation, an application waits for a lock if one is not available at the time of the request, until either of the following events occurs:
  - The lock is granted
  - A deadlock occurs
- Recommended value: If your database access pattern tends toward a majority of writes, set this value so that it gives you early warning when a timeout occurs. A setting of 30 seconds suits this purpose. If your pattern tends toward a majority of reads, either accept the default lock timeout value, or set the property to a value greater than 30 seconds.

DB2 maxlocks:

Specifies the percentage of the lock list that is reached when the database manager performs escalation, from row to table, for the locks held by the application. Although the escalation process does not take much time, locking entire tables versus individual rows decreases concurrency, and potentially decreases overall database performance for subsequent attempts to access the affected tables.

- How to view or set: To view the current value of the maxlocks property for database `xxxxxx`, issue the DB2 `get db cfg for xxxxxx` command and look for the MAXLOCKS value. To set MAXLOCKS to a value of `n`, issue the DB2 `update db cfg for xxxxxx` command using `MAXLOCKS n`, where `xxxxxx` is the name of the application database and `n` is a value between 1 and 100 inclusive.
- Default value: Refer to the current database information for property default values per operating system.
- Recommended value: If lock escalations are causing performance concerns, you might need to increase the value of this parameter or the locklist parameter, which is described in the following paragraph. You can use the database system monitor to determine if lock escalations are occurring.

DB2 locklist:

Specifies the amount of storage that is allocated to the lock list.

- How to view or set: To view the current value of the locklist property for database `xxxxxx`, issue the DB2 `get db cfg for xxxxxx` command and look for the LOCKLIST value . To set LOCKLIST to a value of `n`, issue the DB2 `update db cfg for xxxxxx` command using `LOCKLIST n`, where `xxxxxx` is the name of the application database and `n` is a value between 4 and 60 000 inclusive.
- Default value: Refer to the current database information for property default values per operating system.

- Recommended value: If lock escalations are causing performance concerns, you might need to increase the value of this parameter or the maxlocks parameter, which is described in the previous paragraph. You can use the database system monitor to determine if lock escalations are occurring. Refer to the *DB2 Administration Guide: Performance* document for more details.

Data access tuning parameters

For better application performance, you can tune some data access resources through the WebSphere Application Server administrative console.

Tune these properties of data sources and connection pools to optimize the performance of transactions between your application and datastore. See the *Administering applications and their environment* PDF for more information.

Data source tuning

To view the administrative console page where you configure the following properties, click **Resources** > **JDBC Providers** > **JDBC_provider** > **Data sources** > **data_source** > **WebSphere Application Server connection properties**.

Enable JMS one phase optimization support

If your application does not use JMS messaging, do not select this option. Activating this support enables the Java Message Service (JMS) to get optimized connections from the data source. Activating this support also prevents JDBC applications from obtaining connections from the data source. For further explanation of JMS one phase support, refer to the article entitled “Sharing connections to benefit from one phase commit optimization” in this information center.

Statement cache size

Specifies the number of statements that can be cached per connection.

The WebSphere Application Server data source optimizes the processing of prepared statements and callable statements by caching those statements that are not being used in an active connection. Both statement types help reduce overhead for transactions with backend data.

- A prepared statement is a precompiled SQL statement that is stored in a `PreparedStatement` object. Application Server uses this object to run the SQL statement multiple times, as required by your application run time, with values that are determined by the run time.
- A callable statement is an SQL statement that contains a call to a stored procedure, which is a series of precompiled statements that perform a task and return a result. The statement is stored in the `CallableStatement` object. Application Server uses this object to run a stored procedure multiple times, as required by your application run time, with values that are determined by the run time.

In general, the more statements your application has, the larger the cache should be. Be aware, however, that specifying a larger statement cache size than needed wastes application memory and does not improve performance.

Determine the value for your cache size by adding the number of uniquely prepared statements and callable statements (as determined by the SQL string, concurrency, and the scroll type) for each application that uses this data source on a particular server. This value is the maximum number of possible statements that can be cached on a given connection over the life of the server. See the *Administering applications and their environment* PDF for more information.

Default: For most databases the default is 10. Zero means there is no cache statement.

Connection pool tuning

To view the administrative console page where you configure the following properties, click **Resources** > **JDBC Providers** > **JDBC_provider** > **Data sources** > **data_source** > **Connection pool settings**.

Maximum connections

Specifies the maximum number of physical connections that can be created in this pool. These are the physical connections to the backend datastore. When this number is reached, no new physical connections are created; requestors must wait until a physical connection that is currently in use is returned to the pool.

For optimal performance, set the value for the connection pool lower than the value for the web container threadpool size. Lower settings, such as 10 to 30 connections, might perform better than higher settings, such as 100. See the *Administering applications and their environment* PDF for more information.

Default: 10

Minimum connections

Specifies the minimum number of physical connections to maintain. Until this number is exceeded, the pool maintenance thread does not discard physical connections.

If you set this property for a higher number of connections than your application ultimately uses at run time, you do not waste application resources. WebSphere Application Server does not create additional connections to achieve your minimum setting. Of course, if your application requires more connections than the value you set for this property, application performance diminishes as connection requests wait for fulfillment. See the *Administering applications and their environment* PDF for more information.

Default: 1

Directory conventions

References in product information to *app_server_root*, *profile_root*, and other directories imply specific default directory locations. This article describes the conventions in use for WebSphere Application Server.

Default product locations (distributed)

The following file paths are default locations. You can install the product and other components or create profiles in any directory where you have write access. Multiple installations of WebSphere Application Server - Express products or components require multiple locations. Default values for installation actions by root and nonroot users are given. If no nonroot values are specified, then the default directory values are applicable to both root and nonroot users.

app_client_root

Table 1. Default installation root directories for the Application Client for IBM WebSphere Application Server.

This table shows the default installation root directories for the Application Client for IBM WebSphere Application Server.

User	Directory
Root	AIX `/usr/IBM/WebSphere/AppClient` (Java EE Application client only) HP-UX Linux Solaris `/opt/IBM/WebSphere/AppClient` (Java EE Application client only) Windows `C:\Program Files\IBM\WebSphere\AppClient`
Nonroot	AIX HP-UX Linux Solaris *user_home*`/IBM/WebSphere/AppClient` (Java EE Application client only) Windows `C:\IBM\WebSphere\AppClient`

app_server_root

Table 2. Default installation directories for WebSphere Application Server.

This table shows the default installation directories for WebSphere Application Server - Express.

User	Directory
Root	AIX /usr/IBM/WebSphere/AppServer HP-UX Linux Solaris /opt/IBM/WebSphere/AppServer Windows C:\Program Files\IBM\WebSphere\AppServer
Nonroot	AIX HP-UX Linux Solaris *user_home*/IBM/WebSphere/AppServer Windows *user_home*\IBM\WebSphere\AppServer

component_root

The component installation root directory is any installation root directory described in this article. Some programs are for use across multiple components—in particular, the Web Server Plug-ins, the Application Client, and the IBM HTTP Server. All of these components are part of the product package.

gskit_root

IBM Global Security Kit (GSKit) can now be installed by any user. GSKit is installed locally inside the installing product's directory structure and is no longer installed in a global location on the target system.

Table 3. Default installation directories for GSKit.

This table shows the default installation root directory for Version 8 of the GSKit, where product_root is the root directory of the product that is installing GSKit, for example IBM HTTP Server or the web server plug-in.

User	Directory
Root and nonroot	AIX HP-UX Linux Solaris *product_root*/gsk8 Windows *product_root*\gsk8

profile_root

Table 4. Default profile directories.

This table shows the default directories for a profile named profile_name on each distributed operating system.

User	Directory
Root	AIX /usr/IBM/WebSphere/AppServer/profiles/*profile_name* HP-UX Linux Solaris /opt/IBM/WebSphere/AppServer/profiles/*profile_name* Windows C:\Program Files\IBM\WebSphere\AppServer\profiles*profile_name*
Nonroot	AIX HP-UX Linux Solaris *user_home*/IBM/WebSphere/AppServer/profiles Windows *user_home*\IBM\WebSphere\AppServer\profiles

plugins_root

Table 5. Default installation root directories for the Web Server Plug-ins.

This table shows the default installation root directories for the Web Server Plug-ins for WebSphere Application Server.

User	Directory
Root	AIX /usr/IBM/WebSphere/Plugins HP-UX Linux Solaris /opt/IBM/WebSphere/Plugins Windows C:\Program Files\IBM\WebSphere\Plugins
Nonroot	AIX HP-UX Linux Solaris *user_home*/IBM/WebSphere/Plugins Windows C:\IBM\WebSphere\Plugins

wct_root

Table 6. Default installation root directories for the WebSphere Customization Toolbox.

This table shows the default installation root directories for the WebSphere Customization Toolbox.

User	Directory
Root	AIX /usr/IBM/WebSphere/Toolbox HP-UX Linux Solaris /opt/IBM/WebSphere/Toolbox Windows C:\Program Files\IBM\WebSphere\Toolbox
Nonroot	AIX HP-UX Linux Solaris *user_home*/IBM/WebSphere/Toolbox Windows C:\IBM\WebSphere\Toolbox

web_server_root

Table 7. Default installation root directories for the IBM HTTP Server.

This table shows the default installation root directories for the IBM HTTP Server.

User	Directory
Root	AIX /usr/IBM/HTTPServer HP-UX Linux Solaris /opt/IBM/HTTPServer Windows C:\Program Files\IBM\HTTPServer
Nonroot	AIX HP-UX Linux Solaris *user_home*/IBM/HTTPServer Windows C:\IBM\HTTPServer

Chapter 4. Tuning EJB applications

This page provides a starting point for finding information about enterprise beans.

Based on the Enterprise JavaBeans (EJB) specification, enterprise beans are Java components that typically implement the business logic of Java 2 Platform, Enterprise Edition (J2EE) applications as well as access data.

EJB 2.1 container tuning

This page provides a starting point for finding information about tuning the Enterprise JavaBeans 2.1 container.

About this task

The following topic listed here to get started with tuning your EJB 2.1 container.

EJB container tuning

If you use applications that affect the size of the Enterprise JavaBeans (EJB) container cache, it is possible that the performance of your applications can be affected by an incorrect size setting. Container managed persistence (CMP) is discussed in this topic; although it is important to know that entity beans are not supported in an EJB 3.x module.

EJB cache size

Monitor Tivoli® Performance Viewer to diagnose whether the EJB container cache size setting is tuned correctly for your application.

If the application has filled the cache causing evictions to occur, Tivoli Performance Viewer shows a high rate of the ejbStores method being called and probably a lower than expected processor utilization on the workstation machine.

All applications that use enterprise beans must have this setting adjusted from the default value, if the following formula result equals more than `2000`:

```
EJB_Cache_Size = (Largest number of Option B or C entity beans enlisted in a
transaction * maximum number of concurrent transactions) +
(Largest number of unique Option A entity beans expected to be accessed during
typical application workload) +
(Number of stateful session beans active during typical workload) +
(Number of stateless session bean types used during typical workload)

Where:
Option B and C entity beans are only held in the EJB cache during the lifetime
of the transaction they are enlisted in. Therefore, the first term in the formula
computes the average EJB cache requirements for these types of beans.

Option A entity beans are held in the EJB cache indefinitely, and are only removed
 from the cache if there starts to become more beans in the cache than the cache
size has been set to. If your application uses Read Only Beans, consider them to be
Option A beans for this tuning calculation.

Stateful session beans are held in the EJB cache until they are removed by the
application, or until their session timeout value is reached.

Only a single stateless session bean instance for each EJB type is held in the
cache during the time any methods are running on that stateless session
bean. If two or more methods are being implemented simultaneously on the same
stateless session bean type, each method runs on its own bean instance, but
only one cache location is used for all of these instances.
```

This formula calculates the upper bound on the maximum number of enterprise beans active at one time inside the application server. Because the cache of the EJB container is built to contain all these beans for performance optimizations, best performance can be achieved by setting this cache size to be larger than the number resulting from the previous formula.

You can set the EJB cache size in the administrative console under **Servers** > **Application Servers** > ***server_name*** > **EJB Container** > **EJB Cache Settings**.

Also, while adjusting the EJB cache size, you can tune the EJB container management thread parameter to meet the needs of the application. The management thread is controlled through the **Clean Up Interval** setting. This setting controls how frequently a daemon thread inside of the product attempts to remove bean instances from the cache that have not been used recently, attempting to keep the number of bean instances at or below the cache size. This behavior ensures that the EJB container places and looks up items in the cache quickly. Leave this interval set to the default; however, in some cases, it might be worthwhile to see if there is a benefit to reducing this interval.

For information about tuning the EJB cache using the EJB cache trace service, read about tuning the EJB cache and using the trace service.

EJB stateful session bean tuning

Stateful session bean timeout is configured in different ways, with different scopes, depending on the version of WebSphere Application Server.

WebSphere Application Server Version 6.1 and earlier supports configuration of stateful session bean timeout, per bean, using the `ibm-ejb-jar-ext.xmi` file.

WebSphere Application Server Version 7.0 supports configuration of stateful session bean timeout, per bean, using the `ibm-ejb-jar-ext.xmi` file (for EJB 2.x modules), and the `ibm-ejb-jar-ext.xml` file (for EJB 3.x modules).

WebSphere Application Server Version 8.x supports configuration of stateful session bean timeout, per bean, using the `ibm-ejb-jar-ext.xmi` (for EJB 2.x modules) and the `ibm-ejb-jar-ext.xml` (for EJB 3.x modules) files, and the stateful-timeout XML element and the @StatefulTimeout annotation. Additionally, you can configure a server-wide (global) stateful session timeout value using the com.ibm.websphere.ejbcontainer.defaultStatefulSessionTimeout system property.

Note: For IBM extension and binding files, the .xmi or .xml file name extension is different depending on whether you are using a pre-Java EE 5 application or module or a Java EE 5 or later application or module. An IBM extension or binding file is named ibm-*-ext.xmi or ibm-*-bnd.xmi where * is the type of extension or binding file such as app, application, ejb-jar, or web. The following conditions apply:

- For an application or module that uses a Java EE version prior to version 5, the file extension must be .xmi.
- For an application or module that uses Java EE 5 or later, the file extension must be .xml. If .xmi files are included with the application or module, the product ignores the .xmi files.

However, a Java EE 5 or later module can exist within an application that includes pre-Java EE 5 files and uses the .xmi file name extension.

The `ibm-webservices-ext.xmi`, `ibm-webservices-bnd.xmi`, `ibm-webservicesclient-bnd.xmi`, `ibm-webservicesclient-ext.xmi`, and `ibm-portlet-ext.xmi` files continue to use the .xmi file extensions.

Stateless session beans do not have a timeout value because they have no conversational state and are not dedicated to any specific client.

You can use Rational Application Developer to update the ibm-ejb-jar-ext.xmi file, which is used to configure the stateful session timeout value for beans in an EJB 2.x module. For more information, read about defining session timeout settings for a bean in the Rational Application Developer Information Center.

For example, the generated XMI code to set a stateful session timeout value of 15 minutes is:

```
<ejbExtensions xmi:type="ejbext:SessionExtension" xmi:id="SessionExtension_1"
  timeout="15">
```

You can modify the ibm-ejb-jar-ext.xml file to set the stateful session timeout for beans in an EJB 3.x module. For example, the code to set a stateful session timeout value to 15 minutes for the myBean bean is:

```
<ejbExtensions xmi:type="ejbext:SessionExtension" xmi:id="SessionExtension"
 timeout="15">
    <enterpriseBean xmi:type="ejb:Session" href="META-INF/ejb-jar.xml#MyBean"/>
 </ejbExtensions>
```

You can configure stateful session bean timeout using the @StatefulTimeout annotation. The @StatefulTimeout annotation takes a required value parameter representing the duration of the timeout, and an optional unit parameter. If the optional unit parameter is not specified, the default unit is minutes. The @StatefulTimeout annotation is introduced as part of EJB 3.1.

For example, use the @StatefulTimeout annotation to declare a timeout value of 100 seconds:

```
@StatefulTimeout(value=100 unit=java.util.concurrent.TimeUnit.SECONDS)
```

You can configure stateful session bean timeout using the stateful-timeout XML element in the ejb-jar.xml deployment descriptor. The stateful-timeout element is introduced as part of EJB 3.1.

For example, to set a timeout value of 100 seconds:

```
<stateful-timeout>
     <timeout>100</timeout>
     <unit>Seconds</unit>
</stateful-timeout>
```

The @StatefulTimeout annotation and the stateful-timeout XML element are the specification-defined mechanisms for declaring timeout values per bean, beginning with EJB 3.1. Prior to EJB 3.1, there is no specification-defined way for declaring stateful session timeout per bean. When using the stateful-timeout XML element or @StatefulTimeout annotation, a value of -1 means that the bean never times out, and a value of 0 means that the bean is immediately eligible for removal.

You can configure stateful session bean timeout on a global (server-wide) basis, using the com.ibm.websphere.ejbcontainer.defaultStatefulSessionTimeout system property. The unit of time for the com.ibm.websphere.ejbcontainer.defaultStatefulSessionTimeout is minutes. The specified value must be 0 or greater, and if an invalid value is specified, the default value of 10 minutes is used instead. The global timeout value cannot be configured using XML or annotations. The global timeout value applies to all stateful session beans running in the server, including stateful session beans in EJB 2.x or EJB 3.x modules.

In WebSphere Application Server Version 8.x, bean-level stateful timeout settings take precedence over the server-wide timeout setting. The server-wide timeout setting takes precedence over the default (unspecified) timeout. The following order of precedence is used to determine the stateful session timeout value for a bean running in WebSphere Application Server Version 8.x:

1. stateful-timeout XML element
2. @StatefulTimeout annotation
3. ibm-ejb-jar-ext.xml or ibm-ejb-jar-ext.xmi

4. com.ibm.websphere.ejbcontainer.defaultStatefulSessionTimeout system property
5. If no bean-level or server-wide timeout value is explicitly specified, then the default timeout value of `10` minutes is applied.

Dcom.ibm.websphere.ejbcontainer.poolSize

If the application is using most bean instances in the pool, the Tivoli Performance Viewer indicates this. When the majority of the bean instances are used, increase the size of those bean pools that are being exhausted by adding this parameter in the custom properties tag of the JVM; for example:

```
-Dcom.ibm.websphere.ejbcontainer.poolSize=<application_name>#<module_name>#
<enterprisebean_name>=<minSize>,<maxSize>
```

Where:

The <*application_name*> element is the Java EE application name as defined in the enterprise archive (EAR) file deployment descriptor, for the bean whose pool size is being set.

The <*module_name*> element is the Java archive (JAR) file name of the EJB module, for the bean whose pool size is being set.

The <*bean_name*> element is the Java EE enterprise bean name as defined in the EJB module deployment descriptor, for the bean whose pool size is being set

The <*minSize*> element is the number of bean instances the container maintains in the pool, irrespective of how long the beans have been in the pool (beans greater than this number are cleared from the pool over time to optimize memory usage)

The <*maxSize*> element is the number of bean instances in the pool where no more bean instances are placed in the pool once they are used (that is, when the pool is at this size, any additional beans are discarded rather than added into the pool, which ensures the number of beans in the pool have an upper limit so that memory usage does not grow in an unbounded way).

To keep the number of instances in the pool at a fixed size, the minSize and maxSize elements can be set to the same number. A separate instance pool for every EJB type running in the application server exists, and that every pool starts out with no instances; the number of instances grow as beans are used and then placed in the pool.

When a bean instance is needed by the container and no beans are available in the pool, the container creates a bean instance, uses it, then places that instance in the pool, unless there are already maxSize instances in the pool. For example, the statement -

```
Dcom.ibm.websphere.ejbcontainer.poolSize=ivtApp#ivtEJB.jar#ivtEJBObject=125,1327
```

would set a minSize of `125` and a maxSize of `1327` on the bean named ivtEJBObject within the `ivtEJB.jar` file, in the application ivtApp.

The application, ivtApp, is replaced by the actual application name, the `ivtEJB.jar` file is replaced by the JAR file that contains the bean that must have its pool size increased, and ivtEJBObject is the bean name of the enterprise bean whose pool size must be increased. The minimum number of beans that are held in the pool is `125`. The maximum number of beans that are held in the pool is `1327`. Set these so that no more evictions occur from the pool. In most cases these must be set equal if memory is plentiful because no growth and shrinkage of the pool occurs.

Dcom.ibm.websphere.ejbcontainer.noPrimaryKeyMutation

You must understand how your application handles the creation of primary key objects for use by CMP beans and bean-managed persistence (BMP) beans inside of the product.

The EJB container uses the primary key of an entity bean as an identifier inside of internal data structures to optimize performance. However, the EJB container must copy these primary key objects upon the first access to the bean to ensure that the objects stored in the internal caches are separate from the ones used in an application. This process occurs to keep the internal structures consistent in case the application changes or mutates the primary key. If the application does not mutate any of the primary keys used to create and access entity beans once they are created, a special flag can be used that ensures that the EJB container skips the copy of the primary key object, saving processor cycles, and increasing performance. This mechanism can be enabled at your own risk by adding the **–D** property to the JVM custom property field.

```
-Dcom.ibm.websphere.ejbcontainer.noPrimaryKeyMutation=true
```

Note: Entity beans are not supported in EJB 3.x and later modules.

The performance benefit of this optimization depends on the application. If the application uses primitive types for the primary keys of enterprise beans, there is no gain because these objects are already immutable and the copy mechanism takes this into account. If, however, the application uses many complex primary keys, that is, an object for a primary key or multiple fields, this parameter can yield significant improvements.

Dcom.ibm.ws.pm.deferredcreate

The persistence manager is used by the EJB container to persist data to the database from CMP entity beans.

When creating entity beans by calling the ejbCreate method, by default the persistence manager immediately inserts the empty row with only the primary key in the database. In most cases, once creating the bean you must modify fields in the bean created or in other beans inside of the same transaction. If you want to postpone the insert into the database until the end of the transaction to eliminate one trip to the database, set the **–D** flag inside of the JVM custom properties field. The data is inserted into the database and consistency is maintained.

Note: Entity beans are not supported in EJB 3.x and later modules.

```
-Dcom.ibm.ws.pm.deferredcreate=true
```

The performance benefit of this optimization depends on the application. If the EJB applications transactions are insert intensive, the application can benefit from this optimization. If the application performs few inserts, the benefit of this optimization is less.

Dcom.ibm.ws.pm.batch

When an EJB application accesses multiple CMP beans inside of a single transaction, depending on the operations performed on the beans, such as updates, inserts, and reads, the number of operations issued to the database corresponds directly to the operations performed on the CMP beans. If the database system you are using supports batching of update statements, you can enable this flag and increase performance on all interactions with the database that involve more than two updates in a single transaction.

Note: Entity beans are not supported in EJB 3.x and later modules.

Use this flag, which supports the persistence manager adding all the update statements into one single batch statement that is issued to the database. This process saves round trips to the database, which increases performance. If you know that your application exhibits the behavior of updating multiple CMP beans in a single transaction, and the database supports batch updates, you can set the **–D** flag inside of the JVM custom properties field; for example:

```
-Dcom.ibm.ws.pm.batch=true
```

The performance benefit of this optimization depends on the application. If the application never or infrequently updates CMP beans, or updates only a single bean per transaction, there is no performance gain. If the application updates multiple beans per transaction, this parameter benefits the applications performance.

The following table lists the backend databases that support batch update.

Table 8. Backend databases that support batch update. The following table lists the backend databases that support batch update.

Database	Supports batch update	Supports batch update with Optimistic Concurrency Control
DB2	yes	no
Oracle	yes	no
DB2 Universal Driver	yes	yes
Informix®	yes	yes
SQLServer	yes	yes
Apache Derby	yes	yes

Note: Batch update with OCC cannot be performed for databases that do not support it, even if specified by the access intent.

com.ibm.ws.pm.useLegacyCache

Specifies the name of the Java class that the product uses to implement the javax.rmi.CORBA.UtilDelegate interface.

Note: Entity beans are not supported in EJB 3.x and later modules.

Persistence manager has two types of caching mechanisms, *legacy cache* and *two-level cache*. Typically, two-level cache is more efficient than legacy cache because of optimizations in this mode. The default is legacy cache, although two-level cache is recommended. Set this configuration through the system property as follows:

```
com.ibm.ws.pm.useLegacyCache=false
```

com.ibm.ws.pm.grouppartialupdate and com.ibm.ws.pm.batch

The partial updates feature enhances the performance of applications with enterprise beans in certain scenarios. Persistence manager has two caching mechanisms available, *legacy cache* and *two-level cache*. Typically, two-level cache performs better than legacy cache because of the optimizations in this mode.

Note: Entity beans are not supported in EJB 3.x and later modules.

In certain applications where you must perform both batch updates and partial updates, you must configure the following system properties to gain the benefits of both:

```
'com.ibm.ws.pm.grouppartialupdate=true' and 'com.ibm.ws.pm.batch=true'
```

Tuning Enterprise JavaBeans applications

Tuning EJB cache with trace service

The size of your Enterprise JavaBeans (EJB) cache can affect the performance of the application server. One of the steps in tuning your EJB container to optimum performance levels is to fine-tune the EJB cache.

Before you begin

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

About this task

The following procedure describes how to use the diagnostic trace service to help determine the best cache size.

Procedure

1. Enable the EJB cache trace. To learn about working with the trace service, see the topic, Working with trace. For information about the trace service settings, see the topic Diagnostic trace service settings.

   Set up your trace to use this trace string:

   ```
   com.ibm.ejs.util.cache.BackgroundLruEvictionStrategy=all=enabled:com.ibm.ejs.util.cache.CacheElementEnumerator=
   all=enabled
   ```

   Set **Maximum File Size** to `200MB` or more. If you leave the default value of `20MB`, you could fill up the single 20 MB trace log and lose some data because of trace wrapping.

   Set **Maximum Number of Historical Files** to `5`. Five files should be sufficient, but if you see that all five files are full and trace wrapping occurs, increase this value.
2. Stop your server, delete existing logs, then start your server.
3. Run typical scenarios to capture cache trace data. By running a typical scenario with the trace enabled, you get the EJB cache trace data to analyze in the following steps.
4. View and analyze the trace output.
   a. Open your trace log. Look for either or both of the following trace strings to display:

      ```
      BackgroundLru 3  EJB Cache: Sweep (1,40) - Cache limit not reached : 489/2053
      BackgroundLru >  EJB Cache: Sweep (16,40) - Cache limit exceeded : 3997/2053 Entry
      ```

      In the trace strings that include the words **Cache limit** you find a ratio. For example, 3997/2053. The first number is the number of enterprise beans currently in the EJB cache (called the *capacity*). The second number is the EJB cache setting (more about this in later steps). Use this ratio, particularly the capacity, in your analysis.

      Also look for the statements *Cache limit not reached* and *Cache limit exceeded*.

      Cache limit not reached
      Your cache is equal to or larger than what is appropriate. If it is larger, you are wasting memory, and should reduce the cache size to a more appropriate value.

      Cache limit exceeded
      The number of beans currently being used is greater than the capacity you have specified, indicating that your cache is not properly tuned. The capacity can exceed the EJB Cache setting because the setting is not a hard limit. The EJB Container does not stop adding beans to the cache when the limit is reached. Doing so could mean that when the cache is full, a request for a bean would not be fulfilled, or would at least be delayed until the cache fell below the limit. Instead, the cache limit can be exceeded, but the EJB Container attempts to clean up the cache and keep it below the EJB Cache size.

      In the case where the cache limit is exceeded, you might see a trace point similar to this:

      ```
      BackgroundLru <  EJB Cache: Sweep (64,38) - Evicted = 50 : 3589/2053 Exit
      ```

Notice the *Evicted =* string. If you see this string, you are using either Stateful Session Beans or Entity Beans configured for Option A or B caching. Evicted objects mean that you are not taking full advantage of the caching option that you have chosen. Your first step is to try increasing the EJB Cache size. If continued running of your application results in more evictions, it means that the application is accessing or creating more new beans between EJB Cache sweeps than the cache can hold, and *NOT* reusing existing beans.

You might want to consider using Option C caching for your entity beans, or checking your application to see if it is not removing Stateful Session Beans after they are no longer needed.

Note: Entity beans configured with Option C caching are only in the cache while in a transaction, and are required to be held in the cache for the entire transaction. Therefore, they are never evicted during a cache sweep, but are removed from the cache when the transaction completes. In addition, if you are using only Stateless Session Beans or Entity Beans with Option C caching (or both), then you might want to increase your EJB Cache *cleanup interval* to a larger number. The cleanup interval can be set as described in EJB cache settings. Stateless Session Beans are **NOT** in the EJB Cache, and since Entity Beans using Option C caching are never evicted by the caching (LRU) strategy, there is really no need to sweep often. When using only Stateless Session Beans or Option C caching, you should only see "Evicted = 0" in the trace example shown above.

b. Analyze your trace log. Look for the trace string *Cache limit exceeded*.
   - You might find more than one instance of this string. Examine them all to find the highest capacity value of beans in the EJB Cache. Reset your EJB Cache size to about 110% of this number. Setting the EJB Cache size is explained in a later step.
   - You might find no instances of this string. This means that you have not exceeded the capacity of the EJB Cache (which is your end goal), but not seeing it during your initial analysis could also mean that your cache is too large and using unnecessary memory. In this case, you still must tune your cache by reducing the cache size until your cache limit is not exceeded, then increasing it to the optimum value. Setting the EJB Cache size is explained in a later step.

   Your ultimate goal is to set the cache limit to a value that does not waste resources, but also does not get exceeded. A good set-up gives you a trace with only the *Cache limit not reached* message, and a ratio where the capacity number can be near, but below, 100% of the EJB Cache setting.

   Note: It is recommended that you do not set your cache size to anything less than the default of `2053`.

5. Modify the cache settings based on your analysis. See EJB cache settings for information about how to do this.
6. Stop your server, delete all logs, and restart your server.
7. Repeat the previous steps until you are satisfied with your settings.
8. Disable the EJB Cache trace. With the cache properly tuned, you can remove the trace, remove old logs, and restart your server.

What to do next

From your analysis, it is possible to set the EJB cache optimally from an EJB Container perspective, but perhaps not optimally from a WebSphere Application Server perspective. A larger cache size provides more hits and better EJB cache performance, but uses more memory. Memory used by the cache is not available to other areas of the product, potentially causing the overall performance to suffer. In a system with ample memory, this might not be an issue and properly tuning the EJB cache might increase overall performance. However, you should take into account this system performance versus EJB cache performance when configuring the cache.

EJB method Invocation Queuing

Method invocations to enterprise beans are only queued for remote clients making the method call. An example of a remote client is an Enterprise JavaBeans (EJB) client running in a separate Java virtual machine (JVM) (another address space) from the enterprise bean. In contrast, no queuing occurs if the EJB client, either a servlet or another enterprise bean, is installed in the same JVM on which the EJB method runs, and on the same thread of execution as the EJB client.

Remote enterprise beans communicate by using the Remote Method Invocation over Internet Inter-ORB Protocol (RMI-IIOP). Method invocations initiated over RMI-IIOP are processed by a server-side object request broker (ORB). The thread pool acts as a queue for incoming requests. However, if a remote method request is issued and there are no more available threads in the thread pool, a new thread is created. Once the method request completes the thread is destroyed. Therefore, when the ORB is used to process remote method requests, the EJB container is an open or closed queue, due to the use of unbounded threads.

The following illustration depicts the two queuing options of enterprise beans.

The following are two tips for queueing enterprise beans:

- Analyze the calling patterns of the EJB client.

  When configuring the thread pool, it is important to understand the calling patterns of the EJB client. If a servlet is making a small number of calls to remote enterprise beans and each method call is relatively quick, consider setting the number of threads in the ORB thread pool to a value lower than the web container thread pool size value.

  The degree to which the ORB thread pool value needs increasing is a function of the number of simultaneous servlets, that is, clients, calling enterprise beans and the duration of each method call. If the method calls are longer or the applications spend a lot of time in the ORB, consider making the ORB thread pool size equal to the web container size. If the servlet makes only short-lived or quick calls to the ORB, servlets can potentially reuse the same ORB thread. In this case, the ORB thread pool can be small, perhaps even one-half of the thread pool size setting of the web container.
- Monitor the percentage of configured threads in use.

  Tivoli Performance Viewer shows a metric called `percent maxed`, which is used to determine how often the configured threads are used. A value that is consistently in the double-digits, indicates a possible bottleneck a the ORB. Increase the number of threads.

Tuning applications that use the Java Persistence API

Configuring OpenJPA caching to improve performance

The OpenJPA implementation gives you the option of storing frequently used data in the memory to improve performance. OpenJPA provides concurrent data and concurrent query caches that support applications to save persistent object data and query results in memory to share among threads and for use in future queries.

About this task

The OpenJPA data cache is a cache of persistent object data that operates at the EntityManagerFactory level. This optional-use cache is designed to increase performance while remaining in full compliance with the Java Persistence API (JPA) standard. This means that enabling the caching option can increase the performance of your application, with no changes to your code. The OpenJPA data cache is designed to provide significant performance increases over cacheless operations and ensures that behavior is identical in both cache-enabled and cacheless operations.

When enabled, the cache is examined before accessing the data store. The cache stores data when objects are committed and when persistent objects are loaded from the data store. If operating in a single Java virtual machine (JVM) environment, the JVM maintains and shares a data cache across all EntityManager instances obtained from a particular EntityManagerFactory. The OpenJPA data cache cannot do this in a distributed environment because caches in different JVMs, created from different EntityManagerFactory objects are not synchronized.

Using the OpenJPA cache in a multi-JVM environment can be done by configuring the OpenJPA second level (L2) cache provider plug-in. See the topic, Dynamic cache provider for the JPA 2.0 second level cache, and the section “Using the dynamic cache L2 Cache Provider in a clustered environment”, for more information. Configuring the DynaCache plug-in allows for the Data and Query cache content to be replicated and consistent across JVMs. Other alternatives include setting up an event notification framework or using a third-party distributed cache such as IBM WebSphere eXtreme Scale.

You can enable the OpenJPA data cache for a single or a multiple JVM environment, set its default element size, including soft references, and specify **timeout** values.

To set up and configure the OpenJPA data cache, do the following:

1. To enable the cache for a single JVM, set the openjpa.DataCache property to `true`, and set the openjpa.RemoteCommitProvider property to `sjvm`:

   ```
   <property name="openjpa.DataCache" value="true"/>
   <property name="openjpa.RemoteCommitProvider" value="sjvm"/>
   ```

   To enable the data cache in a distributed environment, the openjpa.RemoteCommitProvider must be configured specifically for the environment, or a third-party cache management utility can be used.
2. The maximum cache size can be adjusted by setting the CacheSize property:

   ```
   <property name="openjpa.DataCache" value="true(CacheSize=5000...
   ```

   By default, the OpenJPA data cache holds 1000 elements. Objects that are pinned into the cache are not counted when determining if the cache size exceeds its maximum size. If the cache overflows, it evicts random elements. You can preserve evicted elements longer with the SoftReferenceSize property. By default, soft references are unlimited. If you must, you can limit the number of soft references or set to 0 to disable soft references completely:

   ```
   <property name="openjpa.DataCache" value="true(CacheSize=5000 SoftReferenceSize=0 ...
   ```
3. You can specify that a cache is cleared at certain times. The EvictionSchedule property of the OpenJPA cache implementation accepts a cron style eviction schedule. The cron format specifies the minute, hour of day, day of month, day of month, and day of the week beginning with 1 for Sunday; the * symbol (asterisk), indicates match all. To schedule a cache to evict at 45 minutes past 3 PM on Sunday every month you would add this property:

   ```
   <property name="openjpa.DataCache" value="true(CacheSize=5000 SoftReferenceSize=0 EvictionSchedule='15,45 * * 1'")/>
   ```
4. You also can specify a cache timeout value for a single class by setting the timeout metadata extension to the amount of time in milliseconds that the data of the class is valid; for example:

   ```
   @Entity
   @DataCache(timeout=10000)
   public class Employee {
       ...
   }
   ```

   Refer to the org.apache.openjpa.persistence.DataCache Javadoc for more information.

After configuring your data cache, you can use it after you restart your application.

Refreshing an entity may lead to different behavior with or without a DataCache when a separate process or part of the same application are updated or even deleted the corresponding record in the database. By default, entities are refreshed from the database even when DataCache is active. Therefore, with the default configuration the refresh behaves identically with or without a DataCache. However, a persistence

unit can be configured to refresh entities from DataCache with the property openjpa.RefreshFromDataCache for improved performance. Under this configuration, any out-of-band changes that occur in the database record do not appear in the refreshed state of the entity.

Note: Regardless of the **openjpa.RefreshFromDataCache** setting, the DataCache is always bypassed for refresh when locks are active, such as for a pessimistic transaction, in a persistence context. An application may activate **openjpa.RefreshFromDataCache** but can still bypass the DataCache while refreshing an entity by explicitly evicting the entity from DataCache before refresh.

OpenJPA provides a concurrent query cache that supports applications to save persistent object data and query results in memory to share among threads and for use in future queries. The query cache stores the object IDs returned by query executions. When you run a query, OpenJPA assembles a key based on the query properties and the parameters used at execution time, and checks for a cached query result. If one is found, the object IDs in the cached result are looked up, and the resultant persistence-capable objects are returned. Otherwise, the query is executed against the database, and the object IDs loaded by the query are put into the cache.

You can configure the query cache settings in a similar way to the data cache. The interface provided to the query cache is the org.apache.openjpa.persistence.QueryResultCache class. You can access this class through the OpenJPAEntityManagerFactory.

The default query cache implementation caches 100 query executions in a *least-recently-used* cache. This can be changed by setting the cache size in the CacheSize plug-in property. Like the data cache, the query cache also has a backing soft reference map that can be changed using the SoftReferenceSize property. To keep queries in the cache at all times, you can pin them to a cache. To change the query cache properties do the following:

1. Modify the CacheSize property of the openjpa.QueryCache:
   ```
   <property name="openjpa.QueryCache" value="true("CacheSize=1000, ...
   ```
2. Change the SoftReferenceSize property to enable and control the size of this map:
   ```
   <property name="openjpa.QueryCache" value="true(CacheSize=1000, SoftReferenceSize=100")/>
   ```

   The SoftReferenceSize table is disabled by default. Setting the size enables it.
3. Pin or unpin queries in the cache through the QueryResultCache with this syntax:
   ```
   public void pin(Query q);
   public void unpin(Query q);
   ```

Modifying these properties allows you to make better use of the query cache.

OpenJPA provides classes that may be extended for further functionality.

- As previously mentioned, if you want to implement a distributed cache that uses an unsupported method for communications, create an implementation of org.apache.openjpa.event.RemoteCommitProvider.
- If you are adding new behavior, extend org.apache.openjpa.datacache.DataCacheImpl.
- To use your own storage mechanism, extend org.apache.openjpa.datacache.AbstractDataCache.
- To add query functionality, you can extend the default org.apache.openjpa.datacache.QueryCacheImpl.
- Implement your own storage mechanism for query results by extending org.apache.openjpa.datacache.AbstractQueryCache

OpenJPA provides a cache that provides caching of SQL strings used by find operations performed on the entity manager and some queries to manage eagerly fetched relationships. When this cache is enabled, SQL queries used by these operations are generated one time per entity manager factory and can be reused. This cache is enabled by default but can also be configured through the openjpa.jdbc.QuerySQLCache configuration property.

The query SQL cache can be configured or disabled through the openjpa.jdbc.QuerySQLCache property. By default, this property is set to `true`. When the property is set to `true`, the cache is enabled and uses the org.apache.openjpa.util.CacheMap class for its cache store. The CacheMap is a managed cache, meaning that it limits the number of cache entries and has a cache eviction scheme to manage memory usage. If the cache is set to all the org.apache.openjpa.lib.util.ConcurrentHashMap class is used as a cache store. The ConcurrentHashMap is not a managed cache so entries remain in the cache for the lifetime of an entity manager factory. This caching mechanism can provide better performance at the expense of increased memory usage. A custom cache store class can also be specified if it implements the java.util.Map interface. To disable the cache, specify the value `false`. See the following examples on how to configure or disable the SQL query cache:

- To use an unmanaged cache:

  ```
  <property name="openjpa.jdbc.QuerySQLCache" value="all"/>
  ```

- To specify a custom cache class:

  ```
  <property name="openjpa.jdbc.QuerySQLCache" value="com.mycompany.MyCustomCache"/>
  ```

- To use an unmanaged cache:

  ```
  <property name="openjpa.jdbc.QuerySQLCache" value="false"/>
  ```

What to do next

You can read more about Caching in the OpenJPA for all caching extensions in the Apache OpenJPA User Guide.

JPA system properties

In addition to the settings that are accessible from the administrative console, you can set Java Persistence API (JPA) system properties using command-line scripting.

You can use the properties page to define the com.ibm.websphere.jpa.entitymanager.poolcapacity JPA system property. Add the system property directly to the `server.xml` file or as a generic Java virtual machine (JVM) argument using the administrative console.

com.ibm.websphere.jpa.entitymanager.poolcapacity

Use this property to specify the default pool capacity for all JPA EntityManager instances on a server.

Information	Value
Data type	Integer
Range	`0` to `500`
Default	`10`

There is a separate pool of EntityManager instances for every persistence context reference that is defined in an application. A persistence context reference can be @PersistenceContext or a persistence-context XML element.

This property setting affects all pools in the application server process. Each pool uses the capacity defined by this property. Although the JPA specification supports pooling of EntityManager instances, some JPA providers might not support the pooling. By default, pooling of EntityManager instances is only enabled for the JPA providers supplied with WebSphere Application Server. The default pool capacity is 10.

When this property is enabled, EntityManager pooling is enabled for all JPA providers. If a JPA provider stops functioning normally when this property is enabled, disable the property and contact the JPA provider for support.

Increasing the integer value of this setting might improve performance by reducing the number of EntityManager instances that must be created. However, increasing the value affects the amount of consumed memory.

Configuring WSJPA ObjectCache to improve performance

The WebSphere Java Persistence API (WSJPA) extension to OpenJPA provides a read-only object cache that can improve performance in certain use cases.

About this task

Implementing a WSJPA ObjectCache object can improve the performance of an application that has a set of data that is used in a static, read-only method, like accessing basic persistent fields and persisting unidirectional relationships to a read-only type. WSJPA ObjectCache is a non-distributed cache of read-only entities that operates at the EntityManagerFactory object level. These cached instances are shared by all EntityManager objects in the Java virtual machine (JVM), but the instances are not managed by any. When the feature is enabled, the ObjectCache object is examined before the application accesses the OpenJPA DataCache object and database, and persistent objects are loaded from the database and stored in the OpenJPA object cache. In addition, the ObjectCache implementation can be used with OpenJPA DataCache and QueryCache features for even greater performance.

Be aware of the following conditions and limitations:

- Include types that are strictly **read-only** from the application point-of-view
  - Passing a read-only type into the following operations results in an UnsupportedOperationException error message:
    - Passing a read-only entity into EntityManager.merge(...).
    - Passing a read-only entity into EntityManager.persist(...).
    - Passing a read-only entity into EntityManager.remove(...).
  - Calling a setter method on a read-only type that was returned by the WebSphere JPA runtime will result in an UnsupportedOperationException error message.
- Types that are included in the ObjectCache must not be eligible to be cached in the OpenJPA DataCache. Do not intersect types that are cacheable by the OpenJPA L2 cache (openjpa.DataCache); the ObjectCache should not be confused with the second-level cache that is defined by the JPA 2.0 specification. If types are intersected, an exception will occur when the EntityManager object is created.
- Include only basic fields, or an exception can occur when the EntityManager object is created.
- Passing a read-only entity into EntityManager.contains(...) always returns false, even if it was just returned from a find/query operation.

You can enable the object cache for a single JVM environment, specify the types that are included in this cache, set its maximum element size, and specify timeout values.

Note: The preferred property name is wsjpa.ObjectCache, but openjpa.ObjectCache is also a valid configuration.

Procedure

- Enable the cache for a single JVM. Set the wsjpa.ObjectCache property to `true` and specify a list of object types to be cached. For example, use the following property:

```
<property name="wsjpa.ObjectCache"
          value="true(Types=com.ibm.wsjpa.Foo; com.ibm.wsjpa.Bar)"/>
```

- Adjust the maximum cache size by setting the MaxSize property. For example:

```
<property name="wsjpa.ObjectCache"
          value="true(Types=com.ibm.wsjpa.Foo; com.ibm.wsjpa.Bar,
          MaxSize=5000)"/>
```

By default an ObjectCache object holds 1000 elements. If the cache overflows, it evicts random elements that are preserved in a soft reference map. The size of the soft reference map is unlimited and cannot be configured.

- Specify to clear the object cache at certain times. The EvictionSchedule property of the ObjectCache implementation accepts a cron-style or interval-style eviction schedule:
  - The cron format specifies the time in the following order:
    1. Minute
    2. Hour of day
    3. Day of month
    4. Month of the year
    5. Day of the week, beginning with 1 for Sunday.

    Use an * (asterisk) for any value to match all for that field. To schedule a cache to evict at 3:15 and 3:45 PM on Sunday every month, add this property:

    ```
    <property name="wsjpa.ObjectCache"
              value="true(Types=com.ibm.wsjpa.Foo; com.ibm.wsjpa.Bar,
                          MaxSize=5000,
                          EvictionSchedule='15,45 15 * * 1')"/>
    ```
  - The interval style eviction schedule specifies the number of minutes between each instance that the cache should be evicted. The format of this property is a plus sign (+) followed by the number of minutes between each instance that the cache should be evicted. To schedule a cache to evict every 20 minutes, for example, add this property:

    ```
    <property name="wsjpa.ObjectCache"
              value="true(Types=com.ibm.wsjpa.Foo; com.ibm.wsjpa.Bar,
                          MaxSize=5000,
                          EvictionSchedule='+20')"/>
    ```
- Specify named queries, which will return read-only types, with the NamedQueries attribute. In addition to the types cached, the maximum number of cached queries can be specified by the QuerySize property. This cache is cleared anytime data is removed from the main object cache. Use the following example to configure the object cache with one type, two named queries, and 5000 cached named queries:

  ```
  <property name="wsjpa.ObjectCache"
            value="true(Types=com.ibm.wsjpa.Foo;com.ibm.wsjpa.Bar,
                        NamedQueries='foo.byid,bar.byid',
                        QuerySize=5000)"/>
  ```

  The default value for the QuerySize property is 1000.

What to do next

To enable automatic loading of the entire ObjectCache when the first EntityManager object is created, follow the steps in the “Pre-loading the WSJPA ObjectCache automatically” topic. In addition, you can read more about caching in the OpenJPA User Guide for information about all caching extensions.

Pre-loading the WSJPA ObjectCache automatically

The WebSphere Java Persistence API (WSJPA) extension to OpenJPA provides a read-only ObjectCache that can improve performance in certain use cases. By default, the data in the cache is loaded in a lazy method, which means that individual entities are loaded into memory when they are requested by an application. If you want to load all the entities from the beginning, though, you can configure the application server to preload all of the entities from the database that are configured in the ObjectCache. Pre-loading the ObjectCache will allow you to cache entities that would otherwise be restricted when if you load the ObjectCache through the lazy method.

About this task

When you enable automatic loading of the ObjectCache, the JPA environment will automatically size and preload the ObjectCache instead of waiting on single entities to be requested by applications. When the application server creates the first EntityManager, the pre-loading process will start, and the application server will preload all entities from the database that are configured in the ObjectCache. This pre-loading process will happen asynchronously in a separate processing thread.

Note: Be aware of the following information:

- For entities that are configured to be in the ObjectCache, you also need to configure all eager relationships in the ObjectCache; any lazy relationships will not be available.
- While the application server is pre-loading the ObjectCache, entities will be fetched from the database. The application server will not add any entities to the ObjectCache that are loaded by other methods.
- Auto loading the cache can take a very long time if your object graph is complex.
- Be careful when you enable this feature, because it could consume all of the available memory. For this function to work, the JPA environment needs to have the complete set of data in memory.

When the pre-loading process is complete, you will see an informational message similar to this:

```
Successfully loaded the ObjectCache with [...] Entities in [...] seconds.
```

Procedure

1. Set the MaxSize property to `auto` for the ObjectCache. For example, include the following entry in your .properties file:

   ```
   <property name="wsjpa.ObjectCache" value="true(Types=com.ibm.wsjpa.Foo; com.ibm.wsjpa.Bar, MaxSize=auto)"/>
   ```

   The default value for the MaxSize property is 1000, but when you set the MaxSize property to `auto` the pre-loading mechanism will be enabled.
2. Optional: Review the value you set for the EvictionSchedule property, which is explained in the “Configuring WSJPA ObjectCache to improve performance” on page 33 topic. When you use the EvictionSchedule property in conjunction with the MaxSize=auto setting, the EvictionSchedule property is implemented differently from the default behavior: instead of clearing the ObjectCache on the configured schedule, the ObjectCache will be automatically reloaded.

Example

The following examples show how to configure the ObjectCache to be loaded automatically:

- The following property specifies that the ObjectCache will be automatically loaded when the application server creates the first EntityManager:

  ```
  <property name=" wsjpa.ObjectCache" value="true(Types=com.ibm.wsjpa.Foo; com.ibm.wsjpa.Bar, MaxSize=auto)"/>
  ```
- The following property specifies that the ObjectCache will be automatically loaded when the application server creates the first EntityManager, and the ObjectCache will be reloaded every 20 minutes:

  ```
  <property name=" wsjpa.ObjectCache" value="true(Types=com.ibm.wsjpa.Foo; com.ibm.wsjpa.Bar,
   MaxSize=auto, EvictionSchedule=+20)"/>
  ```

Chapter 5. Tuning Messaging resources

This page provides a starting point for finding information about the use of asynchronous messaging resources for enterprise applications with WebSphere Application Server.

WebSphere Application Server supports asynchronous messaging based on the Java Message Service (JMS) and the Java EE Connector Architecture (JCA) specifications, which provide a common way for Java programs (clients and Java EE applications) to create, send, receive, and read asynchronous requests, as messages.

JMS support enables applications to exchange messages asynchronously with other JMS clients by using JMS destinations (queues or topics). Some messaging providers also allow WebSphere Application Server applications to use JMS support to exchange messages asynchronously with non-JMS applications; for example, WebSphere Application Server applications often need to exchange messages with WebSphere MQ applications. Applications can explicitly poll for messages from JMS destinations, or they can use message-driven beans to automatically retrieve messages from JMS destinations without explicitly polling for messages.

WebSphere Application Server supports the following messaging providers:

- The WebSphere Application Server default messaging provider (which uses service integration as the provider).
- The WebSphere MQ messaging provider (which uses your WebSphere MQ system as the provider).
- Third-party messaging providers that implement either a JCA Version 1.5 resource adapter or the ASF component of the JMS Version 1.0.2 specification.

Tuning messaging

To tune asynchronous messaging, you can, for example, configure message-driven bean (MDB) throttling for a JCA 1.5-compliant messaging provider.

About this task

To tune your asynchronous messaging, complete one or more of the following steps.

Procedure

- Configure MDB throttling for a JCA 1.5-compliant messaging provider in one of the following ways, depending on your choice of messaging provider:
  - For the default messaging provider, complete the task “Configuring MDB or SCA throttling for the default messaging provider” on page 39.
  - For the WebSphere MQ messaging provider, either set the **maximum server sessions** property on the “WebSphere MQ messaging provider activation specifications advanced properties” page, or the **maxPoolSize** property when using the **createWMQActivationSpec** or **modifyWMQActivationSpec** wsadmin commands.
  - For a third-party JCA 1.5-compliant JMS messaging provider, refer to the generic method of message throttling described in “Throttling inbound message flow for JCA 1.5 message-driven beans” on page 11.
- Tune messaging destinations for the WebSphere MQ.

Tuning messaging performance for the default messaging provider

To help optimize performance, you can set tuning properties that control the performance of message-driven beans and other messaging applications.

About this task

To optimize the performance of messaging with the default messaging provider, you can use the administrative console to set various parameters as described in the steps below. You can also set these parameters by using the wsadmin tool.

Procedure

- Monitor MDB thread pool size for the default messaging provider.

  You might experience a performance bottleneck if there are insufficient threads available for the message-driven beans. There is a trade-off between providing sufficient threads to maximize the throughput of messages and configuring too many threads, which can lead to CPU starvation of the threads in the application server. If you notice that the throughput for express nonpersistent, reliable nonpersistent, or reliable persistent messaging has fallen as a result of increasing the size of the default thread pool, then decrease the size of the thread pool and reassess the message throughput.

  1. View or change the number of threads in the default thread pool for an application server. By default, message-driven beans use the default thread pool.
     a. Click **Servers -> Server Types -> WebSphere application servers -> *server_name* -> [Additional Properties] Thread Pools** > **Default**. By default the Minimum size value is set to 5 and the Maximum size value is set to 20. The best performance is obtained by setting the Maximum size value to the expected maximum concurrency for all message-driven beans. For high throughput using a single message bean, 41 was found to be the optimal Maximum size value.
     b. Change the Maximum size value, then click **OK**.
  2. Optional: Create your own thread pool. The default thread pool is also used by other WebSphere Application Server components, so you might want to define a separate thread pool for the message-driven beans. This reduces thread contention for the default thread pool.
     a. Click **Servers -> Server Types -> WebSphere application servers -> *server_name* -> [Additional Properties] Thread Pools**.
     b. Create a new thread pool.
     c. Create sufficient threads to support the maximum amount of concurrent work for the message-driven beans.
     d. Change the SIB JMS Resource Adapter to use the new thread pool:
        1) Click **Resources -> Resource Adapters -> Resource adapters**.
        2) If you cannot see any SIB JMS Resource Adapter instances in the list, expand **Preferences** and enable **Show built-in resources**.
        3) Select the **SIB JMS Resource Adapter** with the appropriate scope depending upon the scope of the connection factories.
        4) Add the name of the new thread pool in the **Thread pool alias** box.
        5) Click **Apply** .
  3. Save your changes to the master configuration.

- Tune MDB performance with the default messaging provider.
  1. Click **Resources -> JMS -> Activation specifications -> *activation_specification_name***.
  2. Set the maximum batch size for this activation specification.

     Delivering batches of messages to each MDB endpoint can improve performance, particularly when used with Acknowledge mode set to `Duplicates-ok auto-acknowledge`. However, if message ordering must be retained across failed deliveries, set this parameter to 1.
  3. Set the maximum number of concurrent endpoints for this activation specification.

     The maximum concurrent endpoints parameter controls the amount of concurrent work that can be processed by a message bean. The parameter is used with message-driven beans. Increasing the number of concurrent endpoints can improve performance but can increase the number of threads

in use at one time. To benefit from a change in this parameter, there should be sufficient threads available in the MDB thread pool to support the concurrent work. However, if message ordering must be retained across failed deliveries, set this parameter to 1.

4. Save your changes to the master configuration.

For additional information about tuning the throttling of message-driven beans, including controlling the maximum number of instances of each message bean and the message batch size for serial delivery, see "Configuring MDB or SCA throttling for the default messaging provider."

- Change the maximum connections in a connection factory for the default messaging provider.

  The maximum connections parameter limits the number of local connections. The default is 10. This parameter should be set to a number equal to or greater than the number of threads (enterprise beans) concurrently sending messages.

  1. Click **Resources -> JMS -> Topic connection factories ->** ***factory_name*** > **[Additional Properties] Connection pool properties**.
  2. Enter the required value in the **Maximum connections** field.
  3. Click **Apply**.
  4. Save your changes to the master configuration.

Configuring MDB or SCA throttling for the default messaging provider

Use this task to configure the throttling of messages for message-driven beans or Service Component Architecture (SCA) composites that you have deployed as JCA 1.5 resources on the default messaging provider.

Before you begin

The throttling support described in this topic only applies to the default messaging provider (the service integration JMS Resource Adapter).

For the WebSphere MQ messaging provider, you configure message throttling by setting the **maximum server sessions** property on the WebSphere MQ messaging provider activation specifications panel, or the **maxPoolSize** property when using the createWMQActivationSpec or modifyWMQActivationSpec wsadmin commands.

If you have a third-party JCA 1.5-compliant JMS messaging provider, refer to the generic method of message throttling described in "Throttling inbound message flow for JCA 1.5 message-driven beans" on page 11.

About this task

Use this task if you want to throttle messages for a message-driven bean or SCA composite deployed as a J2EE Connector Architecture (JCA) 1.5 resource on the default messaging JMS provider.

The default messaging provider (the service integration JMS Resource Adapter) uses a special type of message throttling. You can leave the message-driven bean pools to the default size of 500.

The default messaging provider enables the throttling of message delivery to a message-driven bean or SCA composite through the **Maximum concurrent endpoints** configuration option on the JMS activation specification used to deploy the bean or composite.

- The maximum number of instances of each message-driven bean or SCA composite is controlled by the Maximum concurrent endpoint setting in the activation specification used to deploy the message-driven bean or SCA composite. This maximum concurrency limit helps prevent a temporary build up of messages from starting too many MDB or SCA instances. By default, the maximum number of concurrent MDB or SCA instances is set to 10.

The Maximum concurrent endpoints field limits the number of endpoints (instances of a given message-driven bean or SCA composite) that process messages concurrently. If the maximum has been reached, new messages are not accepted from the messaging engine for delivery until an endpoint finishes its current processing.

If the available message count (queue depth) associated with a message-driven bean or SCA composite is frequently high, and if your server can handle more concurrent work, you can benefit from increasing the maximum concurrency setting.

If you set the maximum concurrency for a message-driven bean or SCA composite, be sure that you specify a value smaller than the maximum number of endpoint instances that can be created by the adapter that the message-driven bean or SCA composite is bound to. If necessary, increase the endpoint instance limit.

- An activation specification also has a **Maximum batch size** that refers to how many messages can be allocated to an endpoint in one batch for serial delivery. So, for example, if you have set the Maximum concurrent endpoints property to 10 and the Maximum batch Size property to 3, then there can be up to 10 endpoints each processing up to 3 messages giving a total of 30 messages allocated to that message-driven bean or SCA composite. If there are multiple message-driven beans or SCA composites deployed against a single activation specification then these maximum values apply to each message-driven bean or SCA composite individually.
- Take care to ensure that you always set the Maximum concurrent endpoints property is always less than the JCA pool size.

Note: You might want to tune the throttling of your message-driven beans or SCA composites, which is especially important on z/OS®. Workload arriving on the destination the message-driven bean or SCA composite is consuming from might use up more server resource and therefore obstruct other activities. An example of this is when restarting MDB or SCA applications you find a backlog of messages. The number of messages can be throttled so that the message-driven bean or SCA composite can process them in the most efficient manner.

To configure the message throttling support of the default messaging provider (the service integration bus JMS Resource Adapter), use the administrative console to complete the following steps.

Procedure

- Tune the maximum number of instances of a message-driven bean or SCA composite.

  The maximum concurrency is set in the activation specification used to deploy the message-driven bean or SCA composite.

  1. Click **Resources -> Resource Adapters -> J2C activation specifications -> *activation_specification_name* -> [Additional Properties] J2C activation specification custom properties**.
  2. View the maxConcurrency custom property. The default is value is 10. For high throughput primitive MDB or SCA tests, 40 was found to be an optimal value.
  3. Optional: To change the maxConcurrency setting, click the value field. This displays a panel for you to type a new value. In the Value field, type the new value then click **OK**. Save your changes to the master configuration.
- Tune the maximum batch size for a message-driven bean or SCA composite.

  By default, only a single message is delivered to a message-driven bean or SCA instance at one time. You can improve performance by batching messages to the message-driven bean or SCA composite. Each message-driven bean or SCA instance then receives a number (between 1 and the batch size) of messages at a time. A change in the maximum concurrency is likely to be beneficial if the available message count (queue depth) associated with the message-driven bean or SCA composite is frequently high. For more information about the available message count, see View the Available Message Count on a destination. The maximum batch size is set in the activation specification used to deploy the message-driven bean or SCA composite.

1. Click **Resources -> Resource Adapters -> J2C activation specifications -> *activation_specification_name* -> [Additional Properties] J2C activation specification custom properties**.
2. View the maxBatchSize custom property. The default value is 1. For high throughput primitive MDB or SCA tests, 5 was found to be optimal value (providing a 20 per cent gain over batch size 1).
3. Optional: To change the maxBatchSize setting, click the value field. This displays a panel for you to type a new value. In the Value field, type the new value then click **OK**. Save your changes to the master configuration.

Tuning messaging destinations for the WebSphere MQ messaging provider

You can optimize performance of message-driven bean (MDB) or other applications that use the WebSphere MQ messaging provider by configuring the properties of WebSphere MQ messaging provider destinations.

About this task

To optimize performance, configure queue destination properties in WebSphere Application Server and WebSphere MQ to best fit your applications.

Procedure

To optimize performance for WebSphere MQ queues, configure the queue destination properties to best fit your message-driven bean (MDB) or other applications that use the queue destinations. For example:

- When MDB applications are configured to queues on WebSphere MQ for z/OS, INDEX by MSGID, where the queue manager maintains an index of message identifiers, is very important. For further information, see “Type of index” in the *Developing Applications* section of the WebSphere MQ information center.
- Setting the Expiry property to SPECIFIED and the Specified Expiry property to 30000 milliseconds for the expiry timeout for the WebSphere MQ messaging provider queue in WebSphere Application Server reduces the number of messages that can be queued.
- To ensure that there are enough underlying WebSphere MQ resources available for the queue, ensure that you configure the queue destination properties in WebSphere Application Server adequately for use by your message-driven beans or other applications that use the queue.
- Consider the queue attributes associated with the queue name you created with WebSphere MQ. Inappropriate queue attributes can reduce the performance of WebSphere operations. You can use WebSphere MQ commands to change queue attributes for the queue name:

  BOQNAME
  The excessive backout requeue name. This attribute can be set to a local queue name that can hold the messages that were rolled back by the WebSphere applications. This queue name can be a system dead letter queue.

  BOTHRESH
  The backout threshold and can be set to a number when the threshold is reached, the message is moved to the queue name specified in BOQNAME.

  INDXTYPE
  Set this attribute to MSGID to cause an index of message identifiers to be maintained, which can improve WebSphere MQ retrieval of messages.

  DEFSOPT
  Set this attribute to SHARED (for shared input from the queue).

  SHARE
  This attribute must be specified (so that multiple applications can get messages from this queue).

For more information about using these properties, see the *Script (MQSC) Command Reference* section of the WebSphere MQ information center. For more information specifically about BOQNAME and BOTHRESH, see “Handling poison messages” in the *Using Java* section of the WebSphere MQ information center.

Throttling inbound message flow for JCA 1.5 message-driven beans

This topic describes how to throttle message delivery for message-driven beans (MDB) which are deployed as message endpoints for Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) Version 1.5 inbound resource adapters.

Before you begin

The throttling of messages as described in this topic does not apply to the two JCA 1.5-compliant messaging providers that are supplied with WebSphere Application Server:

- The default messaging provider.
- The WebSphere MQ messaging provider.

For the default messaging provider, you configure message throttling as described in the related tasks. For the WebSphere MQ messaging provider, you configure message throttling by setting the **maximum server sessions** property on the WebSphere MQ messaging provider activation specifications panel, or the **maxPoolSize** property when using the createWMQActivationSpec or modifyWMQActivationSpec **wsadmin** commands.

If you have a third-party JCA 1.5-compliant JMS messaging provider, check with your supplier to see whether the method of message throttling described in this topic is appropriate for their messaging provider.

About this task

For installations that use resource adapters that implement the Java EE Connector Architecture (JCA) Version 1.5 message delivery support, the WebSphere Application Server provides message throttling support to control the delivery of messages to endpoint message-driven beans (MDB). You can use this support to avoid overloading the server with a flood of inbound messages.

Message delivery is throttled on a message-driven bean basis by limiting the maximum number of endpoint instances that can be created by the adapter that the MDB is bound to. When the adapter attempts to create an endpoint instance, a proxy for the MDB instance is created and returned as allowed by the JCA 1.5 architecture. There is a one-to-one correspondence between proxies and MDB instances, and like the MDB instances, the proxies are pooled based on the minimum and maximum pool size values associated with the message-driven bean. Throttling is performed through the management of the proxy pool.

At the time the adapter attempts to create an endpoint, if the number of endpoint proxies currently created is equal to the maximum size of the pool, adapter *createEndPoint* processing returns an `Unavailable Exception`. When this displays, the adapter does not issue any more createEndPoint() requests until it has released at least one endpoint back to the server for reuse. Thus, installations can control the throttling of message delivery to a JCA 1.5 MDB based on the setting of the maximum size of the pool associated with each JCA 1.5 message-driven bean.

You can specify the pool size by using the com.ibm.websphere.ejbcontainer.poolsize JVM system property to define the minimum and maximum pool size of stateless, message-driven, and entity beans. For a message-driven bean that supports JCA 1.5, the maximum pool size value specified limits how many message endpoint instances can be created for that message-driven bean. For example, if the installation sets the maximum size of a JCA 1.5 MDB pool to 5, then at most 5 messages can be concurrently delivered to 5 instances of the message-driven bean. This property can be specified using the **wsadmin**

scripting tool or by specifying it under the administrative console as an environmental variable.

Procedure

1. Open the administrative console.
2. Select **Servers** > **Server Types** > **WebSphere application server** > ***server_name***.
3. Under **Server Infrastructure**, expand **Java and Process Management** >**Process Definition**.
4. Under Additional Properties, select **Java Virtual Machine**.
5. Under Additional Properties, select **Custom Properties**.
6. Select **New**. A panel with three **General Properties** fields displays. This is where you set the property.
7. In the **Name** field, enter `com.ibm.websphere.ejbcontainer.poolsize`.
8. To complete the **Value** field, see the EJB container system properties for values.
9. After defining the value of the property, click **OK**. You are now prompted to save the changes you have made.
10. Click **Save**.

Monitoring server session pools for listener ports

You can minimize the number of resources that server sessions use by enabling server session pool monitoring and defining the timeout value to be applied to a server session.

About this task

Each listener port uses one or more server sessions, which are held in a server session pool. Each server session is associated with a JMS session, which is taken from the JMS session pool that is associated with the JMS connection factory that the listener port is configured to use.

By default, server session pool monitoring is disabled. When a listener port uses a server session the listener port does not release the server session from the server session pool until the listener port is shut down. This means that the associated JMS session is not released into the JMS session pool until the listener port is shut down, even if the listener port is not processing any messages. Consequently the resources that the JMS session uses, for example TCP/IP connections, can be held for a long time, and this can cause problems for resource-constrained systems.

To minimize the number of resources that server sessions use, you must monitor the server session pools. When you enable server session pool monitoring each server session in each server session pool that a listener port uses is monitored to determine how much time has elapsed since the server session was last used. If the elapsed time is greater than the timeout value that you have configured, the server session is removed from the server session pool and its associated JMS session is returned to the JMS session pool. The returned JMS session can be either reused by another application or closed, depending on your JMS session pool settings. You can also configure additional pooling mechanisms, depending on your JMS provider.

Note: Server session pool monitoring cannot be used if the message listener service is operating in non-Application Server Facilities (non-ASF) mode, that is if the NON.ASF.RECEIVE.TIMEOUT message listener service custom property is set to a non-zero value.

Procedure

To enable server session pool monitoring, configure the following message listener service custom properties on each application server as required.

SERVER.SESSION.POOL.REAP.TIME
To enable server session pool monitoring, set this property to the time in seconds between checks on server session pools (this must be a non-negative value).

SERVER.SESSION.POOL.UNUSED.TIMEOUT
To specify the default server session pool timeout, set this property to the required number of seconds for the timeout. When this property is set to a non-negative value, it is compared with the time that has elapsed since a server session was used. If the timeout value is less than the elapsed time, the server session is removed from the server session pool and its JMS session is returned to the JMS session pool. For example, if the timeout value is one second and the time that has elapsed since a particular server session was used is two seconds, that server session is removed from the server session pool and its JMS session is returned to the JMS session pool.

SERVER.SESSION.POOL.UNUSED.TIMEOUT.*lpname*
To override the default SERVER.SESSION.POOL.UNUSED.TIMEOUT value for the listener port with the name *lpname*, set this property to the appropriate value:

- To override the SERVER.SESSION.POOL.UNUSED.TIMEOUT for the specified listener port, set this property to a non-negative value defining the required number of seconds for the server session timeout for this listener port.
- To disable server session pool monitoring for the specified listener port, set this property to a negative value.

The value that you set for this property applies to all message-driven beans that are using the specified listener port.

Example

For example, consider an application server that is configured with listener ports lp1, and lp2.

The following rules apply:

No properties set
If none of the properties are set, server session pool monitoring is disabled and JMS sessions used by server sessions are not returned to the JMS session pool until the listener port (lp1 or lp2), or its associated message-driven bean, is shut down.

SERVER.SESSION.POOL.REAP.TIME and SERVER.SESSION.POOL.UNUSED.TIMEOUT set
Consider, for example, the following settings:

SERVER.SESSION.POOL.REAP.TIME=60

SERVER.SESSION.POOL.UNUSED.TIMEOUT=120

The server session pool of both listener ports (lp1 and lp2) is checked for inactive server sessions every 60 seconds. If a server session is detected as being inactive for more than 120 seconds, it is removed from the server session pool and its JMS session is returned to the JMS session pool. Taking into account the SERVER.SESSION.POOL.REAP.TIME value, the server session pool could be removed from the session pool between two and three minutes after the server session was last used.

SERVER.SESSION.POOL.REAP.TIME and SERVER.SESSION.POOL.UNUSED.TIMEOUT set, and overrides set for SERVER.SESSION.POOL.UNUSED.TIMEOUT.*lpname*
Consider, for example, the following settings:

SERVER.SESSION.POOL.REAP.TIME=60

SERVER.SESSION.POOL.UNUSED.TIMEOUT=120

SERVER.SESSION.POOL.UNUSED.TIMEOUT.lp2=-1

SERVER.SESSION.POOL.UNUSED.TIMEOUT.lp1=60

The server session pool for listener port lp2 is not checked because it has a negative timeout value. In the server session pool for listener port lp1, any server sessions that are inactive for more than 60 seconds are removed from the server session pool.

Chapter 6. Tuning Object Request Broker (ORB)

This page provides a starting point for finding information about the Object Request Broker (ORB). The product uses an ORB to manage communication between client applications and server applications as well as among product components. These Java Platform, Enterprise Edition (Java EE) standard services are relevant to the ORB: Remote Method Invocation/Internet Inter-ORB Protocol (RMI/IIOP) and Java Interface Definition Language (Java IDL).

The ORB provides a framework for clients to locate objects in the network and call operations on those objects as though the remote objects were located in the same running process as the client, providing location transparency.

Tuning Object Request Brokers

Object Request Broker tuning guidelines

Use the guidelines in this document any time the Object Request Broker (ORB) is used in a workload.

The ORB is used whenever enterprise beans are accessed through a remote interface. If you experience particularly high or low CPU consumption, you might have a problem with the value of one of the following parameters. Examine these core tuning parameters for every application deployment.

Thread pool adjustments

Size

Tune the size of the ORB thread pool according to your workload. Avoid suspending threads because they have no work ready to process. If threads do not have work ready to process, CPU time is consumed by calling the Object.wait method, performing a context switch. Tune the thread pool size such that the length of time that the threads wait is short enough to prevent them from being destroyed because they are idle too long.

The thread pool size is dependent on your workload and system. In typical configurations, applications need 10 or fewer threads per processor.

However, if your application is performing a very slow backend request, like a request to a database system, a server thread blocks waiting for the backend request to complete. With backend requests, CPU use is fairly low. In this case, increasing the load does not increase CPU use or throughput. Your thread dumps indicate that nearly all the threads are in a call out to the backend resource. In this case, consider increasing the number of threads per processor until throughput improves and thread dumps show that the threads are in other areas of the run time besides the backend call. You should adjust the number of threads only if your backend resource is tuned correctly.

The **Allow thread allocation beyond maximum thread size** parameter also affects thread pool size, but do not use this parameter unless your back end stops for long periods of time, causing the blocking of all the run-time threads waiting for the backend system instead of processing other work that does not involve the backend system.

You can adjust the thread pool size settings in the administrative console. Click **Servers > Server Types > Application servers >** *server_name* **> Container services > ORB service > Thread pool**. You can adjust the minimum and maximum number of threads.

Thread pool timeout

Each inbound and outbound request through the ORB requires a thread from the ORB thread pool. In heavy load scenarios or scenarios where ORB requests nest deeply, it is possible for a Java virtual machine (JVM) to have all threads from the ORB thread pool attempting to send requests. Meanwhile, the

remote JVM ORB that process these requests has all threads from its ORB thread pool attempting to send requests. As a result, progress is never made, threads are not released back to the ORB thread pool, and the ORB is unable to process requests. As a result, there is a potential deadlock. Using the administrative console, you can adjust this behavior through the ORB com.ibm.websphere.orb.threadPoolTimeout custom property. For more information, see the documentation about the Object Request Broker custom properties.

Fragment size

The ORB separates messages into fragments to send over the ORB connection. You can configure this fragment size through the com.ibm.CORBA.FragmentSize parameter.

To determine and change the size of the messages that transfer over the ORB and the number of required fragments, perform the following steps:

1. In the administrative console, enable ORB tracing in the ORB Properties page.
2. Enable ORBRas tracing from the logging and tracing page.
3. Increase the trace file sizes because tracing can generate a lot of data.
4. Restart the server and run at least one iteration (preferably several) of the case that you are measuring.
5. Look at the traceable file and do a search for `Fragment to follow: Yes`.

   This message indicates that the ORB transmitted a fragment, but it still has at least one remaining fragment to send prior to the entire message arrives. A `Fragment to follow: No` value indicates that the particular fragment is the last in the entire message. This fragment can also be the first, if the message fit entirely into one fragment.

   If you go to the spot where `Fragment to follow: Yes` is located, you find a block that looks similar to the following example:

   ```
   Fragment to follow: Yes
   Message size:       4988 (0x137C)
    --
   Request ID:         1411
   ```

   This example indicates that the amount of data in the fragment is 4988 bytes and the Request ID is 1411. If you search for all occurrences of `Request ID: 1411`, you can see the number of fragments that are used to send that particular message. If you add all the associated message sizes, you have the total size of the message that is being sent through the ORB.
6. You can configure the fragment size by setting the com.ibm.CORBA.FragmentSize ORB custom property.

Interceptors

Interceptors are ORB extensions that can set up the context prior to the ORB runs a request. For example, the context might include transactions or activity sessions to import. If the client creates a transaction, and then flows the transaction context to the server, then the server imports the transaction context onto the server request through the interceptors.

Most clients do not start transactions or activity sessions, so most systems can benefit from removing the interceptors that are not required.

To remove the interceptors, manually edit the `server.xml` file and remove the interceptor lines that are not needed from the ORB section.

Connection Cache Adjustments

Depending on an application server's workload, and throughput or response-time requirements, you might need to adjust the size of the ORB's connection cache. Each entry in the connection cache is an object that represents a distinct TCP/IP socket endpoint, identified by the hostname or TCP/IP address, and the port number used by the ORB to send a GIOP request or a GIOP reply to the remote target endpoint. The

purpose of the connection cache is to minimize the time required to establish a connection by reusing ORB connection objects for subsequent requests or replies. (The same TCP/IP socket is used for the request and corresponding reply.)

For each application server, the number of entries in the connection cache relates directly to the number of concurrent ORB connections. These connections consist of both the inbound requests made from remote clients and outbound requests made by the application server. When the server-side ORB receives a connection request, it uses an existing connection from an entry in the cache, or establishes a new connection and adds an entry for that connection to the cache.

The ORB Connection cache maximum and Connection cache minimum properties are used to control the maximum and minimum number of entries in the connection cache at a given time. When the number of entries reaches the value specified for the Connection cache maximum property, and a new connection is needed, the ORB creates the requested connection, adds an entry to the cache and searches for and attempts to remove up to five inactive connection entries from the cache. Because the new connection is added prior to the inactive entries are removed, it is possible for the number of cache entries to temporarily exceed the value specified for the Connection cache maximum property.

An ORB connection is considered inactive if the TCP/IP socket stream is not in use and there are no GIOP replies pending for any requests made on that connection. As the application workload diminishes, the ORB closes the connections and removes the entries for these connections from the cache. The ORB continues to remove entries from the cache until the number of remaining entries is at or less than the value specified for the Connection cache maximum property. The number of cache entries is never less then the value specified for the Connection cache minimum property, which must be at least five connections less than the value specified for the Connection cache maximum property.

Adjustments to the connection cache in the client-side ORB are usually not necessary because only a small number of connections are made on that side.

JNI Reader Threads

By default, the ORB uses a Java thread for processing each inbound connection request it receives. As the number of concurrent requests increases, the storage consumed by a large number of reader threads increases and can become a bottleneck in resource-constrained environments. Eventually, the number of Java threads created can cause out-of-memory exceptions if the number of concurrent requests exceeds the system's available resources.

To help address this potential problem, you can configure the ORB to use JNI reader threads where a finite number of reader threads, implemented using native OS threads instead of Java threads, are created during ORB initialization. JNI reader threads rely on the native OS TCP/IP asynchronous mechanism that enables a single native OS thread to handle I/O events from multiple sockets at the same time. The ORB manages the use of the JNI reader threads and assigns one of the available threads to handle the connection request, using a round-robin algorithm. Ordinarily, JNI reader threads should only be configured when using Java threads is too memory-intensive for your application environment.

The number of JNI reader threads you should allocate for an ORB depends on many factors and varies significantly from one environment to another, depending on available system resources and workload requirements. The following potential benefits might be achieved if you use JNI threads:

- Because a fixed number of threads is allocated, memory usage is reduced. This reduction provides significant benefit in environments with unusually large and sustained client-request workloads.
- The time needed to dynamically create and destroy Java threads is eliminated because a fixed number of JNI threads is created and allocated during ORB initialization.
- Each JNI thread can handle up to 1024 socket connections and interacts directly with the asynchronous I/O native OS mechanism, which might provide enhanced performance of network I/O processing.

Chapter 7. Tuning Service integration

This page provides a starting point for finding information about service integration.

Service integration provides asynchronous messaging services. In asynchronous messaging, producing applications do not send messages directly to consuming applications. Instead, they send messages to destinations. Consuming applications receive messages from these destinations. A producing application can send a message and then continue processing without waiting until a consuming application receives the message. If necessary, the destination stores the message until the consuming application is ready to receive it.

Tuning messaging engines

Use this task to set tuning properties for the service integration environment.

About this task

The service integration environment includes properties that you can set to improve the performance of a messaging engine or the component of the messaging engine that manages the data store. These properties are known collectively as tuning properties. You can set these properties either with the WebSphere Application Server administrative console or by editing the `sib.properties` file.

Tip: Properties set with the administrative console take precedence over properties set in the `sib.properties` file.

Procedure

- Set tuning properties by using the administrative console:
  - Set the tuning properties of a messaging engine.
  - Control the memory buffers used by a messaging engine.
- Use the administrative console to tune the data source.
- Set tuning properties for any of the components mentioned above by editing the sib.properties file.

Setting tuning properties of a messaging engine

You can set the tuning properties for a messaging engine to improve its performance.

About this task

You can set the following tuning property for a messaging engine:

`sib.trm.retry`
The messaging engine to messaging engine connection retry interval, in seconds. The retry interval is the time delay left between attempts to contact neighboring messaging engines with which communications exist. The default retry interval is `30` seconds.

To set the tuning properties for a messaging engine, use the administrative console to complete the following steps.

Procedure

1. In the navigation pane, click **Service integration -> Buses -> *bus_name* -> [Topology] Messaging engines -> *engine_name* -> [Additional Properties] Custom properties**.
2. Type the name of the property that you want to set.
3. Type the value that you want to set for that property.
4. Click **OK**.

5. Save your changes to the master configuration.
6. Restart the messaging engine for the changes to take effect.

Controlling the memory buffers used by a messaging engine

Every messaging engine manages two memory buffers that contain messages and message-related data. You can improve the interaction of a messaging engine with its data store by tuning the properties that set the sizes of the two buffers.

About this task

You can set the following properties to improve the interaction of a messaging engine with its data store:

sib.msgstore.discardableDataBufferSize
The size in bytes of the data buffer that the messaging engine uses to contain data for which the quality of service attribute is best effort nonpersistent. The default value is `320000`, which is approximately 320 kilobytes.

The discardable data buffer contains all data for which the quality of service attribute is best effort nonpersistent. That data comprises both data that is involved in active transactions, and any other best effort nonpersistent data that the messaging engine has neither discarded nor consumed. The messaging engine holds this data entirely within this memory buffer and never writes the data to the data store. When the messaging engine adds data to the discardable data buffer, for example when the messaging engine receives a best effort nonpersistent message from a client, the messaging engine might discard data already in the buffer to make space. The messaging engine can discard only data that is not involved in active transactions. This behavior enables the messaging engine to discard best effort nonpersistent messages.

Increasing the size of the discardable data buffer allows more best effort nonpersistent data to be handled before the messaging engine begins to discard messages.

sib.msgstore.cachedDataBufferSize
The size in bytes of the data buffer that the messaging engine uses to contain data for which the quality of service attribute is *better than* `best effort nonpersistent` and that is held in the data store. The default value is `320000`, which is approximately 320 kilobytes.

The purpose of the cached data buffer is to optimize the performance of the messaging engine by caching in memory the data that the messaging engine might otherwise have to read from the data store. As it writes data to the data store and reads from the data store, the messaging engine attempts to add that data to the cached data buffer. The messaging engine might discard data already in the buffer to make space.

sib.msgstore.transactionSendLimit
The maximum number of operations that the messaging engine includes in each transaction. For example, each JMS send or receive is an operation that counts towards the transaction send limit. The default value is `100`.

Attention: The messaging engine uses approximate calculations to manage the data it holds in the memory buffers. Neither of the **DataBufferSize** properties gives an accurate indication of the amount of memory that the messaging engine consumes in the JVM heap. The messaging engine can consume considerably more heap storage than the **DataBufferSize** properties indicate.

To set the properties of a messaging engine to improve its interaction with its data store, use the administrative console to complete the following steps:

Procedure

1. In the navigation pane, click **Service integration -> Buses -> *bus_name* -> [Topology] Messaging engines -> *engine_name* -> [Additional Properties] Custom properties**.
2. Type the name of the property that you want to set.

3. Type the value that you want to set for that property.
4. Click **OK**.
5. Save your changes to the master configuration.

What to do next

Remember: When you change any of these properties, the new values do not take effect until you restart the messaging engine.

Tuning the JDBC data source of a messaging engine

The messaging engine needs to have the correct configuration for JDBC data source to achieve messaging performance on a service integration bus.

Before you begin

Consider whether you must configure the connection pool for the JDBC data source to achieve your requirements for messaging performance.

About this task

The messaging engine uses the connection pool to obtain its connections to the database. With a heavy workload, a messaging engine might require a large number of concurrent connections to avoid delays waiting for connections to become available in the pool. For example, a very heavily loaded messaging engine might need 50 or more connections. Complete the following steps to configure the connection pool to meet your performance requirements:

Procedure

1. Ensure that the configuration of your relational database management system (RDBMS) permits the number of connections that you require. Refer to the documentation for your RDBMS for more information.
2. Use the administrative console to set the connection pool parameters for your data source. Navigate to **Resources -> JDBC -> Data sources -> *data_source_name* -> [Additional Properties] Connection pool properties**.
   a. Set **Maximum connections** to the number of connections you require, for example, at least 50. The default number of connections is 10.

      Tip: If your messaging engine times out when requesting a database connection, check the error log. If the error log contains error message CWSIS1522E, increase the number of connections and ensure that the configuration of your RDBMS permits that number of connections.
   b. Set **Purge policy** to `EntirePool`. This policy enables the connection pool to release all connections when the messaging engine stops.

Setting tuning properties by editing the sib.properties file

Use this task to set tuning properties for the service integration environment by editing the `sib.properties` file

About this task

You can set the following tuning properties to improve the performance of components in the service integration environment.

Properties for a messaging engine

sib.trm.retry
The messaging engine to messaging engine connection retry interval, in seconds. The retry interval is the time delay left between attempts to contact neighboring messaging engines with which communications exist. The default retry interval is `30` seconds.

Properties for the component of a messaging engine that manages the data store

sib.msgstore.discardableDataBufferSize
The size in bytes of the data buffer that the messaging engine uses to contain data for which the quality of service attribute is best effort nonpersistent. The default value is `320000`, which is approximately 320 kilobytes.

The discardable data buffer contains all data for which the quality of service attribute is best effort nonpersistent. That data comprises both data that is involved in active transactions, and any other best effort nonpersistent data that the messaging engine has neither discarded nor consumed. The messaging engine holds this data entirely within this memory buffer and never writes the data to the data store. When the messaging engine adds data to the discardable data buffer, for example when the messaging engine receives a best effort nonpersistent message from a client, the messaging engine might discard data already in the buffer to make space. The messaging engine can discard only data that is not involved in active transactions. This behavior enables the messaging engine to discard best effort nonpersistent messages.

Increasing the size of the discardable data buffer allows more best effort nonpersistent data to be handled before the messaging engine begins to discard messages.

sib.msgstore.cachedDataBufferSize
The size in bytes of the data buffer that the messaging engine uses to contain data for which the quality of service attribute is *better than* `best effort nonpersistent` and that is held in the data store. The default value is `320000`, which is approximately 320 kilobytes.

The purpose of the cached data buffer is to optimize the performance of the messaging engine by caching in memory the data that the messaging engine might otherwise have to read from the data store. As it writes data to the data store and reads from the data store, the messaging engine attempts to add that data to the cached data buffer. The messaging engine might discard data already in the buffer to make space.

sib.msgstore.transactionSendLimit
The maximum number of operations that the messaging engine includes in each transaction. For example, each JMS send or receive is an operation that counts towards the transaction send limit. The default value is `100`.

To set these properties by editing the `sib.properties` file, complete the following steps:

Procedure

1. Navigate to the *profile_root*`/properties` directory, where *profile_root* is the directory in which profile-specific information is stored.
2. If the directory does not contain a sib.properties file, then copy the template sib.properties files from the *app_server_root*/properties directory, where *app_server_root* is the root directory for the installation of WebSphere Application Server.
3. Using a text editor, open the `sib.properties` file and add the name and value of the property that you want to set. The format is *name=value*`.` For example `sib.trm.retry=60`

Tuning messaging performance with service integration

To help optimize performance, you can set tuning properties that control the performance of messaging applications deployed to use service integration technologies.

About this task

To optimize the performance of messaging with service integration technologies, you can use the administrative console to set various parameters as described in the steps below. You can also set these parameters by using the wsadmin tool.

Procedure

- View the Available Message Count on a destination.

  Viewing the Available Message Count on a destination enables you to determine whether your message consumers are able to cope with your current workload. If the available message count on a given destination is too high, or is increasing over time, consider some of the tuning recommendations in this topic.

  1. Enable AvailableMessageCount statistics for a queue. If you restart the administrative server, enable **`AvailableMessageCount`** statistics again because such runtime settings are not preserved when the server is restarted.
     a. In the navigation pane, click **Monitoring and Tuning -> Performance Monitoring Infrastructure (PMI)**.
     b. In the content pane, click ***server_name***.
     c. Click the Runtime tab.
     d. In the Currently monitored statistic set, click **Custom**.
     e. On the Custom monitoring level panel, click **SIB Service** > **SIB Messaging Engines** > ***engine_name*** > **Destinations** > **Queues** > ***queue_name***.
     f. Select the AvailableMessageCount option.
     g. Click **Enable** at the top of the panel.
  2. View the available message count for a queue.
     a. In the navigation pane, click **Monitoring and Tuning -> Performance Viewer -> Current activity**.
     b. In the content pane, click ***server_name***.
     c. Click **Performance Modules** > **SIB Service** > **SIB Messaging Engines** > ***engine_name*** > **Destinations** > **Queues** > ***queue_name***.
     d. Click **View Module(s)** at the top of the Resource Selection panel, located on the left side. This displays the AvailableMessageCount data in the Data Monitoring panel, located on the right side.

        You can use the Data Monitoring panel to manage the collection of monitoring data; for example, you can use the buttons to start or stop logging, or to change the data displayed as either a table or graph.

- Ensure that application servers hosting one or more messaging engines are provided with an appropriate amount of memory for the message throughput you require.

  You can tune the initial and maximum Java Virtual Machine (JVM) heap sizes when adding a server to a messaging bus, that is when you create a messaging engine. You have the option to do this in any of the following cases:

  – When adding a single server to a bus
  – When adding a cluster to a bus
  – When adding a new server to an existing cluster that is itself a bus member

  For an application server that is a bus member of at least one bus, or a member of a cluster that is a bus member of at least one bus, the recommended initial and maximum heap sizes are 768MB.

  When you are adding a cluster to a bus, you are recommended to increase the initial and maximum JVM heap sizes for every server in the cluster to 768MB.

  – Increasing the initial heap size improves the performance for small average message sizes
  – Increasing the maximum heap size improves the performance for higher average message sizes

- Reduce the occurrence of OutOfMemoryError exceptions

  If the cumulative size of the set of messages being processed within a transaction by the service integration bus is large enough to exhaust the JVM heap, OutOfMemoryError exceptions occur. Consider one of the following options for reducing the occurrence of OutOfMemoryError exceptions when processing a large set of messages within a transaction.

  – Increase the JVM heap sizes for the application server.

    During the peak activity period, it is essential to ensure that the messaging engine has adequate heap size to handle the messaging engine failover processes. This also applies when the messaging engine is failing over to another instance in a cluster member environment when the JVM heap is nearly exhausted. In such situations, you must increase the maximum heap size value by approximately 512 MB for each of the cluster members that are eligible to host the messaging engine in a failover situation. For example, for WebSphere Application Server on z/OS, the adjunct heap value must be increased by 512 MB for cluster members associated with the messaging engine if you are running under conditions that nearly exhaust the JVM heap.

  – Reduce the cumulative size of the set of messages being processed within the transaction.

- Tune reliability levels for messages.

  The reliability level chosen for the messages has a significant impact on performance. In order of decreasing performance (fastest first), the reliability levels are:

  Best effort nonpersistent

  Express nonpersistent

  Reliable nonpersistent

  Reliable persistent

  Assured persistent

  For MDB point-to-point messaging, best effort nonpersistent throughput is more than six times greater than assured persistent. For more information about reliability levels, see Message reliability levels - JMS delivery mode and service integration quality of service.

Tuning messaging engine data stores

Obtain an overview of improving the performance of messaging engine data stores.

About this task

- “Tuning the JDBC data source of a messaging engine” on page 53
- “Controlling the memory buffers used by a messaging engine” on page 52
- Sharing connections to benefit from one-phase commit optimization

Tuning the JDBC data source of a messaging engine

The messaging engine needs to have the correct configuration for JDBC data source to achieve messaging performance on a service integration bus.

Before you begin

Consider whether you must configure the connection pool for the JDBC data source to achieve your requirements for messaging performance.

About this task

The messaging engine uses the connection pool to obtain its connections to the database. With a heavy workload, a messaging engine might require a large number of concurrent connections to avoid delays waiting for connections to become available in the pool. For example, a very heavily loaded messaging

engine might need 50 or more connections. Complete the following steps to configure the connection pool to meet your performance requirements:

Procedure

1. Ensure that the configuration of your relational database management system (RDBMS) permits the number of connections that you require. Refer to the documentation for your RDBMS for more information.
2. Use the administrative console to set the connection pool parameters for your data source. Navigate to **Resources -> JDBC -> Data sources -> *data_source_name* -> [Additional Properties] Connection pool properties**.
   a. Set **Maximum connections** to the number of connections you require, for example, at least 50. The default number of connections is 10.

      Tip: If your messaging engine times out when requesting a database connection, check the error log. If the error log contains error message CWSIS1522E, increase the number of connections and ensure that the configuration of your RDBMS permits that number of connections.
   b. Set **Purge policy** to `EntirePool`. This policy enables the connection pool to release all connections when the messaging engine stops.

Controlling the memory buffers used by a messaging engine

Every messaging engine manages two memory buffers that contain messages and message-related data. You can improve the interaction of a messaging engine with its data store by tuning the properties that set the sizes of the two buffers.

About this task

You can set the following properties to improve the interaction of a messaging engine with its data store:

`sib.msgstore.discardableDataBufferSize`
The size in bytes of the data buffer that the messaging engine uses to contain data for which the quality of service attribute is best effort nonpersistent. The default value is `320000`, which is approximately 320 kilobytes.

The discardable data buffer contains all data for which the quality of service attribute is best effort nonpersistent. That data comprises both data that is involved in active transactions, and any other best effort nonpersistent data that the messaging engine has neither discarded nor consumed. The messaging engine holds this data entirely within this memory buffer and never writes the data to the data store. When the messaging engine adds data to the discardable data buffer, for example when the messaging engine receives a best effort nonpersistent message from a client, the messaging engine might discard data already in the buffer to make space. The messaging engine can discard only data that is not involved in active transactions. This behavior enables the messaging engine to discard best effort nonpersistent messages.

Increasing the size of the discardable data buffer allows more best effort nonpersistent data to be handled before the messaging engine begins to discard messages.

`sib.msgstore.cachedDataBufferSize`
The size in bytes of the data buffer that the messaging engine uses to contain data for which the quality of service attribute is *better than* `best effort nonpersistent` and that is held in the data store. The default value is `320000`, which is approximately 320 kilobytes.

The purpose of the cached data buffer is to optimize the performance of the messaging engine by caching in memory the data that the messaging engine might otherwise have to read from the data store. As it writes data to the data store and reads from the data store, the messaging engine attempts to add that data to the cached data buffer. The messaging engine might discard data already in the buffer to make space.

sib.msgstore.transactionSendLimit
The maximum number of operations that the messaging engine includes in each transaction. For example, each JMS send or receive is an operation that counts towards the transaction send limit. The default value is `100`.

Attention: The messaging engine uses approximate calculations to manage the data it holds in the memory buffers. Neither of the **DataBufferSize** properties gives an accurate indication of the amount of memory that the messaging engine consumes in the JVM heap. The messaging engine can consume considerably more heap storage than the **DataBufferSize** properties indicate.

To set the properties of a messaging engine to improve its interaction with its data store, use the administrative console to complete the following steps:

Procedure

1. In the navigation pane, click **Service integration -> Buses -> *bus_name* -> [Topology] Messaging engines -> *engine_name* -> [Additional Properties] Custom properties**.
2. Type the name of the property that you want to set.
3. Type the value that you want to set for that property.
4. Click **OK**.
5. Save your changes to the master configuration.

What to do next

Remember: When you change any of these properties, the new values do not take effect until you restart the messaging engine.

Increasing the number of data store tables to relieve concurrency bottleneck

Service integration technologies enables users to spread the data store for a messaging engine across several tables. In typical use this is unlikely to have a significant influence. However, if statistics suggest a concurrency bottleneck on the *SIBnnn* tables for a data store, you might try to solve the problem by increasing the number of tables.

About this task

For more information about the set of tables in a data store see Data store tables.

Data	Description
SIB000	contains information about the structure of the data in the other two tables - the “stream table”
SIB001	contains persistent objects - the “permanent item table”
SIB002	contains nonpersistent objects that have been saved to the data store to reduce the messaging engine memory requirement - the “temporary item table”

Having multiple tables means you can relieve any performance bottleneck you might have in your system. You can modify *SIBnnn* tables of the data store of a messaging engine. You can increase the number of permanent and temporary tables (*SIB001* and *SIB002*), although there is no way to increase the number of stream tables (*SIB000*).

Example

This example illustrates what the *SIBnnn* tables for a data store might look like after modification:

Data	Description
SIB000	contains information about the structure of the data in the other two tables - the “stream table”
SIB001	contains persistent objects - the “permanent item table”
SIB002	contains persistent objects - the “permanent item table”
SIB003	contains persistent objects - the “permanent item table”
SIB004	contains nonpersistent objects that have been saved to the data store to reduce the messaging engine memory requirement - the “temporary item table”
SIB005	contains nonpersistent objects that have been saved to the data store to reduce the messaging engine memory requirement - the “temporary item table”

Increasing the number of item tables for a messaging engine when tables are not automatically created

If a concurrency bottleneck occurs on the item tables, increasing the number of item tables will increase the throughput of the messaging engine.

Before you begin

Before performing this task you must ensure that the messaging engine is using a data store, and that its **Create tables** option is set to `False`.

Procedure

1. Relevant performance monitoring tools show that the throughput of a messaging engine is inefficient.
2. Use your database performance monitoring tools to examine lock statistics for the item tables for evidence of a bottleneck. Consult database documentation on how to interpret the locking statistics.
3. Create tables and increase data store attributes.
   - Create tables for the data store schema. For more information see Creating data store tables
   - Increase number of permanent tables or temporary tables, or both, for the data store. For more information see Configuring a messaging engine data store to use a data source

   You can only increase the number of permanent tables or temporary tables, not decrease them.
4. Stop and restart the WebSphere Application Server so that configuration changes take effect. The extra tables are used when the messaging engine restarts.
5. Observe the effect on throughput and lock statistics by checking performance monitoring tools. Consider whether any improvement is sufficient and modifying the data store attributes further would be beneficial

Increasing the number of item tables for a messaging engine when tables are automatically created

If a concurrency bottleneck occurs on the item tables, increasing the number of items tables will increase the throughput of the messaging engine.

Before you begin

Before performing this task you must ensure that the messaging engine is using a data store, and that its **Create tables** option is set to `True`.

Procedure

1. Relevant performance monitoring tools show that the throughput of a messaging engine is insufficient.
2. Use your database performance monitoring tools to examine lock statistics for the item tables for evidence of a bottleneck. Consult your database documentation on how to interpret the locking statistics.
3. Increase the attributes for the messaging engine data store: the number of permanent tables or temporary tables, or both. For more information see Configuring a messaging engine data store to use a data source. You can only increase the number of permanent tables or temporary tables, not decrease them.
4. Stop and restart the messaging engine so that configuration changes take effect. The extra tables are created when the messaging engine starts again.
5. Observe the effect on throughput and lock statistics by checking performance monitoring tools. Consider whether any improvement is sufficient and modifying the data store attributes further would be beneficial

Tuning one-phase commit optimization

If you have configured your messaging engine to use a data store, you can achieve better performance by configuring both the messaging engine and container-managed persistent (CMP) beans to share the same data source.

About this task

You must configure both the CMP bean and the messaging engine resource authorization so that they share the same data source.

Procedure

1. Open the administrative console.
2. Click **Applications -> Application Types -> WebSphere enterprise applications -> *application_name* -> [Enterprise Java Bean Properties] Map data sources for all 2.x CMP beans**.
3. On the content pane, select the check boxes next to all the CMP beans.
4. Select *Per application* in the **Resource authorization** drop-down list.
5. Modify the messaging engine resource authorization to *Per application* by modifying the property file `sib.properties` and adding the custom property `sib.msgstore.jdbcResAuthForConnections=Application`.

Tuning the detection of database connection loss

If a messaging engine is configured to use a data store and cannot connect to its data store, for example because the database that contains the data store is not running, the messaging engine does not start. You can tune your system to increase the chance of a successful start of the messaging engine.

About this task

In a single-server environment, when you start the application server the messaging engine attempts to start. If the database is unavailable for more than 15 minutes, the messaging engine might enter the stopped state and need to be started manually.

You can increase the chance of the messaging engine starting successfully by configuring various parameters, such as the 15 minute default timeout, on the database server or application server.

Procedure

1. On the database server, configure the operating system to minimize the amount of time taken to detect the loss of a network connection to an application server. Refer to the documentation for the operating system for details. For example, the following table lists the relevant parameters for Windows and AIX operating systems:

   Table 9. TCP/IP parameters. The first column of the table provides the list of TCP/IP parameters for the Windows operating systems. The second column of the table provides the list of TCP/IP parameters for the AIX operating systems. The third column provides the description of the parameters.

Parameter name on Windows operating systems	Parameter name on AIX operating systems	Description
KeepAliveTime	tcp_keepidle	The amount of time (in milliseconds on Windows operating systems and in 0.5 seconds on AIX operating systems) to wait before sending a keepalive request for an inactive connection.
KeepAliveInterval	tcp_keepintvl	The amount of time (in milliseconds on Windows operating systems and in 0.5 seconds on AIX operating systems) to wait for a response.
TCPMaxDataRetransmissions	tcp_keepcnt	The number of requests to send before ending the connection.

   You can calculate the total amount of time taken for the database server to detect the failure of the connection to the application server, by using the following formula:

   time to detect connection failure = keep alive time + (keep alive interval x number of requests)

   For example, for a Windows system with the parameters set according to the following table, the total amount of time taken for the database server to detect the failure of the connection to the application server is 350 seconds.

   Table 10. Example parameter values. The first column provides the parameter names. The second column provides a sample value for the parameters.

Parameter	Value
KeepAlive	300000 milliseconds
KeepAliveInterval	10000 milliseconds
TCPMaxDataRetransmissions	5

   Your database product might also have relevant parameters that you can configure, for example, the IDLE THREAD TIMEOUT parameter in DB2 for z/OS.

   When the database server detects the loss of the connection to the application server, the database releases the locks on the data store. The messaging engine can now access the data store and can therefore start successfully.

2. On the application server, tune the messaging engine to wait for an appropriate amount of time for the data store to become available. By default, the messaging engine will attempt to connect to the data store every 2 seconds for 15 minutes. Complete the rest of this step if you want to adjust these timings.

   a. Click **Service integration -> Buses -> *bus_name* -> [Topology] Messaging engines -> *engine_name* -> [Additional Properties] Custom properties** to navigate to the custom properties panel for the messaging engine.

   b. Click **New**.

   c. Type `sib.msgstore.jdbcInitialDatasourceWaitTimeout` in the Name field and an appropriate value in the Value field. This property is the time, in milliseconds, to wait for the data store to become available. The default value is 900000 (15 minutes). This time includes the time required to establish a connection to the database and to obtain the required table locks.

Ensure that the value of this property is greater than the total time taken for the database server to detect the loss of a network connection, as configured in step 1.

d. Click **OK**.

e. Click **New**.

f. Type `sib.msgstore.jdbcStaleConnectionRetryDelay` in the Name field and an appropriate value in the Value field. This property is the time, in milliseconds, to wait between attempts to connect to the data store. The default value is 2000 (2 seconds). For example, if you set the sib.msgstore.jdbcInitialDatasourceWaitTimeout property to 600000, and the sib.msgstore.jdbcStaleConnectionRetryDelay property to 3000, the messaging engine will attempt to connect every 3 seconds until 10 minutes has passed.

g. Click **OK**.

h. Save your changes to the master configuration.

i. Restart the application server.

Results

By configuring these parameters and custom properties, you minimize the amount of time taken for the database server to detect the loss of a network connection, and ensure that the messaging engine waits for a reasonable amount of time for the database connection to recover before attempting to start.

What to do next

You might want to configure the messaging engine and server to restart in the event of a database connection failure. This behavior reduces the risk of the messaging engine being in an inconsistent state when the database connection is restored.

Setting tuning properties for a mediation

Use this task to tune a mediation for performance by using the administrative console.

Before you begin

Review the guidance on when it is appropriate to tune a mediation for performance in the topic Performance tuning for mediations.

About this task

You can set the following tuning property in the administrative console to improve the performance of a mediation:

sib:SkipWellFormedCheck
Whether you want to omit the well formed check that is performed on messages after they have been processed by the mediation. Either `true` or `false`.

Note: This property is overridden for messages that have the delivery option assured persistent, and a well formed check is always performed.

To set, or unset, one or more tuning properties for a mediation, use the administrative console to complete the following steps:

Procedure

1. Display the mediation context information:
   a. Click **Service integration -> Buses -> *bus_name* -> [Destination resources] Mediations**.

b. In the content pane, select the name of the mediation for which you want to configure tuning information.
c. Click **[Additional Properties] Context information**.
2. In the content pane, click **New**.
3. Type the name of the property in the **Name** field.
4. Select the type `Boolean` from the drop-down list.
5. Type **true** in the **Context Value** field to set the property, or type **false** to unset the property.
6. Click **OK**.
7. Save your changes to the master configuration.

Enabling CMP entity beans and messaging engine data stores to share database connections

Use this task to enable container-managed persistence (CMP) entity beans to share the database connections used by the data store of a messaging engine. Performing this task has been estimated to provide a potential performance improvement of 15% for overall message throughput, but can only be used for entity beans connected to the application server that contains the messaging engine.

About this task

To enable CMP entity beans to share the database connections used by the data store of a messaging engine, complete the following steps.

Procedure

1. Configure the data store to use a data source that is not XA-capable. For more information about configuring a data store, see Configuring a JDBC data source for a messaging engine.
2. Select the **Share data source with CMP** option. This option is provided on the JMS connection factory or JMS activation specification used to connect to the service integration bus that hosts the bus destination that is used to store and process messages for the CMP bean.

   For example, to select the option on a unified JMS connection factory, complete the following steps:
   a. Display the default messaging provider. In the navigation pane, click **Resources -> JMS -> JMS providers**.
   b. Select the default provider for which you want to configure a unified connection factory.
   c. Optional: Change the **Scope** check box to set the level at which the connection factory is to be visible, according to your needs.
   d. In the content pane, under Additional Properties, click **Connection factories**.
   e. Optional: To create a new unified JMS connection factory, click **New**.

      Specify the following properties for the connection factory:

      Name Type the name by which the connection factory is known for administrative purposes.

      JNDI name
      Type the JNDI name that is used to bind the connection factory into the namespace.

      Bus name
      Type the name of the service integration bus that the connection factory is to create connections to. This service integration bus hosts the destinations that the JMS queues and topics are assigned to.
   f. Optional: To change the properties of an existing connection factory, select its name from one of the connection factories displayed. The properties for the connection factory are displayed in the content pane.
   g. Select the check box for the **Share data source with CMP** field.

h. Click **OK**.

i. Save your changes to the master configuration.

The JMS connection factory can only be used to connect to a “local” messaging engine that is in the application server on which the CMP beans are deployed.

3. Deploy the CMP beans onto the application server that contains the messaging engine, and specify the same data source as that used by the messaging engine. You can use the administrative console to complete the following steps:
   a. Optional: To determine the data source used by the messaging engine, click **Servers -> Server Types -> WebSphere application servers -> *server_name* -> [Server messaging] Messaging engines -> *engine_name* -> [Additional Properties] Message store**.

      The **Data source name** field displays the name of the data source, which is by default:

      `jdbc/com.ibm.ws.sib/`*`engine_name`*

   b. Click **Applications -> New Application -> New Enterprise Application**.
   c. On the first Preparing for the application installation page, specify the full path name of the source application file (.ear file, otherwise known as an EAR file), then click **Next**.
   d. On the second Preparing for the application installation page, complete the following steps:
      1) Select the check box for **Generate Default Bindings**. Data source bindings (for EJB 1.1 JAR files) are generated based on the JNDI name, data source, user name, and password options. This results in default data source settings for each EJB JAR file. No bean-level data source bindings are generated.
      2) Under Connection Factory Bindings, select the check box for **Default connection factory bindings:**, then type the JNDI name for the data source and optionally select a **Resource authorization** value.
      3) Click **Next** to display the Install New Application pages. The contents of the application that you are installing determines which pages are available.
4. If your application uses EJB modules that contain CMP beans that are based on the EJB 1.x specification, for **Map default data sources for modules containing 1.x entity beans**, specify a JNDI name for the default data source for the EJB modules. The default data source for the EJB modules is optional if data sources are specified for individual CMP beans.
5. If your application has CMP beans that are based on the EJB 1.x specification, for **Map data sources for all 1.x CMP**, specify a JNDI name for data sources to be used for each of the 1.x CMP beans. The data source attribute is optional for individual CMP beans if a default data source is specified for the EJB module that contains CMP beans.
6. Click **Finish**. If neither a default data source for the EJB module nor a data source for individual CMP beans are specified, a validation error displays and the installation is cancelled.
7. Complete other pages as needed.
8. On the Summary page, verify the cell, node, and server onto which the application modules will install.
   a. Beside **Cell/Node/Server**, click **Click here**.
   b. Verify the settings on the Map modules to servers page that is displayed. Ensure that the application server that is specified contains the messaging engine and its data store.
   c. Specify the web servers as targets that will serve as routers for requests to this application. This information is used to generate the plug-in configuration file (plugin-cfg.xml) for each web server.
   d. Return to the Summary page.
   e. Click **Finish**.

Results

For more information about installing applications, see Installing enterprise application files with the console.

Chapter 8. Tuning security

This page provides a starting point for finding information about how to maintain, improve and harden your security configurations.

Tuning, hardening, and maintaining security configurations

After installing WebSphere Application Server, there are several considerations for tuning, strengthening, and maintaining your security configuration.

About this task

The following topics are covered in this section:

Procedure

- Tuning security configurations You can tune your security configuration to balance performance with function. You can achieve this balance following considerations for tuning general security, Common Secure Interoperability version 2 (CSIv2), Lightweight Directory Access Protocol (LDAP) authentication, web authentication, and authorization. For more information on tuning security, see “Tuning security configurations.”
- Hardening security configurations Several methods exist that you can use to protect your infrastructure and applications from different forms of attack. For more information on hardening your security, see “Hardening security configurations” on page 70.
- Securing passwords in files Password encryption and encoding can add protection to passwords existing in files. For more information on encoding and encrypting passwords, see “Securing passwords in files” on page 73.

What to do next

For additional information about hardening security configurations, see the WebSphere Application Server security web page.

Tuning security configurations

You can tune security to balance performance with function. You can achieve this balance following considerations for tuning general security, Common Secure Interoperability version 2 (CSIv2), Lightweight Directory Access Protocol (LDAP) authentication, web authentication, and authorization.

About this task

Performance issues typically involve trade-offs between function and speed. Usually, the more function and the more processing that are involved, the slower the performance. Consider what type of security is necessary and what you can disable in your environment. For example, if your application servers are running in a Virtual Private Network (VPN), consider whether you can disable Secure Sockets Layer (SSL). If you have a lot of users, can they be mapped to groups and then associated to your Java Platform, Enterprise Edition (Java EE) roles? These questions are things to consider when designing your security infrastructure.

Procedure

- Consider the following recommendations for tuning general security.
  - Consider disabling Java 2 security manager if you know exactly what code is put onto your server and you do not need to protect process resources. Remember that in doing so, you put your local resources at some risk.

- Consider increasing the cache and token timeout if you feel your environment is secure enough. By increasing these values, you have to re-authenticate less often. This action supports subsequent requests to reuse the credentials that already are created. The downside of increasing the token timeout is the exposure of having a token hacked and providing the hacker more time to hack into the system before the token expires. You can use security cache properties to determine the initial size of the primary and secondary hashtable caches, which affect the frequency of rehashing and the distribution of the hash algorithms.

  See the article Authentication cache settings for a list of these properties.
- Consider changing your administrative connector from Simple Object Access Protocol (SOAP) to Remote Method Invocation (RMI) because RMI uses stateful connections while SOAP is completely stateless. Run a benchmark to determine if the performance is improved in your environment.
- Use the wsadmin script to complete the access IDs for all the users and groups to speed up the application startup. Complete this action if applications contain many users or groups, or if applications are stopped and started frequently. WebSphere Application Server maps user and group names to unique access IDs in the authorization table. The exact format of the access ID depends on the repository. The access ID can only be determined during and after application deployment. Authorization tables created during assembly time do not have the proper access IDs. See the Commands for the AdminApp article for more information about how to update access IDs.
- Consider tuning the Object Request Broker (ORB) because it is a factor in enterprise bean performance with or without security enabled. Refer to the information about ORB tuning guidelines.
- If using SSL, enable the SSL session tracking mechanism option as described in the information about session management settings.
- In some cases, using the unrestricted Java Cryptography Extension (JCE) policy file can improve performance. Refer to the information about tuning Web Services Security.
- Distributing the workload to multiple Java virtual machines (JVMs) instead of a single JVM on a single machine can improve the security performance because there is less contention for authorization decisions.

• Consider the following steps to tune Common Secure Interoperability version 2 (CSIv2).
- Consider using Secure Sockets Layer (SSL) client certificates instead of a user ID and password to authenticate Java clients. Because you are already making the SSL connection, using mutual authentication adds little overhead while it removes the service context that contains the user ID and password completely.
- If you send a large amount of data that is not very security sensitive, reduce the strength of your ciphers. The more data you have to bulk encrypt and the stronger the cipher, the longer this action takes. If the data is not sensitive, do not waste your processing with 128-bit ciphers.
- Consider putting only an asterisk (*) in the trusted server ID list (meaning trust all servers) when you use identity assertion for downstream delegation. Use SSL mutual authentication between servers to provide this trust. Adding this extra step in the SSL handshake performs better than having to fully authenticate the upstream server and check the trusted list. When an asterisk (*) is used, the identity token is trusted. The SSL connection trusts the server through client certificate authentication.
- Ensure that stateful sessions are enabled for CSIv2. This is the default, but requires authentication only on the first request and on any subsequent token expirations.
- Consider changing the values for the CSIv2 session cache. Changing these values can avoid resource shortages. Refer to the Common Secure Interoperability Version 2 outbound communications topic for more information.
- If you are communicating only with WebSphere Application Server Version 5 or higher servers, make the Active Authentication Protocol CSI, instead of CSI and SAS. This action removes an interceptor invocation for every request on both the client and server sides.

  Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

• Consider the following steps to tune Lightweight Directory Access Protocol (LDAP) authentication.

1. In the administration console, click **Security > Global security**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select **Standalone LDAP registry** and click **Configure**.
3. Select the **Ignore case for authorization** option in the stand-alone LDAP registry configuration, when case-sensitivity is not important.
4. Select the **Reuse connection** option.
5. Use the cache features that your LDAP server supports.
6. Choose either the IBM Tivoli Directory Server or SecureWay directory type, if you are using an IBM Tivoli Directory Server. The IBM Tivoli Directory Server yields improved performance because it is programmed to use the new group membership attributes to improve group membership searches. However, authorization must be case insensitive to use IBM Tivoli Directory Server.
7. Choose either iPlanet Directory Server (also known as Sun ONE) or Netscape as the directory if you are an iPlanet Directory user. Using the iPlanet Directory Server directory can increase performance in group membership lookup. However, use **Role** only for group mechanisms.

- Consider the following steps to tune web authentication.
  - Increase the cache and token timeout values if you feel your environment is secure enough. The web authentication information is stored in these caches and as long as the authentication information is in the cache, the login module is not invoked to authenticate the user. This supports subsequent requests to reuse the credentials that are already created. A disadvantage of increasing the token timeout is the exposure of having a token stolen and providing the thief more time to hack into the system before the token expires.
  - Enable single sign-on (SSO). To configure SSO, click **Security > Global security**. Under Web security, click **Single sign-on (SSO)**.

    SSO is only available when you configure **LTPA** as the authentication mechanism in the Authentication mechanisms and expiration panel. Although you can select Simple WebSphere Authentication Mechanism (SWAM) as the authentication mechanism on the Authentication mechanisms and expiration panel, SWAM is deprecated in Version 8.5 and does not support SSO. When you select SSO, a single authentication to one application server is enough to make requests to multiple application servers in the same SSO domain. Some situations exist where SSO is not a desirable and you do not want to use it in those situations.
  - Disable or enable the **Web Inbound Security Attribute Propagation** option on the Single sign-on (SSO) panel if the function is not required. In some cases, having the function enabled can improve performance. This improvement is most likely for higher volume cases where a considerable number of user registry calls reduces performance. In other cases, having the feature disabled can improve performance. This improvement is most likely when the user registry calls do not take considerable resources.
  - The following two custom properties might help to improve performance when security attribute propagation is enabled:
    - **com.ibm.CSI.propagateFirstCallerOnly**

      The default value of this property is `true`. When this custom property is set to `true` the first caller in the propagation token that stays on the thread is logged when security attribute propagation is enabled. When this property is set to `false`, all of the caller switches are logged, which can affect performance.
    - **com.ibm.CSI.disablePropagationCallerList**

      When this custom property is set to `true` the ability to add a caller or host list in the propagation token is completely disabled. This function is beneficial when the caller or host list in the propagation token is not needed in the environment.
- Consider the following steps to tune authorization.
  - Map your users to groups in the user registry. Associate the groups with your Java Platform, Enterprise Edition (Java EE) roles. This association greatly improves performance when the number of users increases.

  - Judiciously assign method-permissions for enterprise beans. For example, you can use an asterisk (*) to indicate all the methods in the method-name element. When all the methods in enterprise beans require the same permission, use an asterisk (*) for the method-name to indicate all methods. This indication reduces the size of deployment descriptors and reduces the memory that is required to load the deployment descriptor. It also reduces the search time during method-permission match for the enterprise beans method.
  - Judiciously assign security-constraints for servlets. For example, you can use the `*.jsp` URL pattern to apply the same authentication data constraints to indicate all JavaServer Pages (JSP) files. For a given URL, the exact match in the deployment descriptor takes precedence over the longest path match. Use the `*.jsp`, `*.do`, `*.html` extension match if no exact matches exist and longest path matches exist for a given URL in the security constraints.
- Use new tuning parameters when using Java 2 security. The new tuning parameters can improve performance significantly, and introduce a new concept called *Read-only Subject*, which enables a new cache for J2C Auth Subjects when using container-managed auth data aliases. If the J2C auth subject does not need to be modified after it is created, the following new tuning parameters can be used to improve Java 2 Security performance:
  - com.ibm.websphere.security.auth.j2c.cacheReadOnlyAuthDataSubjects=true
  - com.ibm.websphere.security.auth.j2c.readOnlyAuthDataSubjectCacheSize=50 (This is the maximum number of subjects in the hashtable of the cache. Once the cache reaches this size, some of the entries are purged. For better performance, this size should be equal to the number of unique subjects (cache based on uniqueness of user principal + auth data alias + managed connection factory instance) when role-based security and Java 2 security are used together).
- Use new tuning parameters to improve the performance of Security Attribute Propagation. The new tuning parameters can be set through custom properties in the administrative console to reduce the extra overhead of Security Attribute Propagation:
  - com.ibm.CSI.disablePropagationCallerList=true
  - com.ibm.CSI.propagateFirstCallerOnly=true (use if you want to track the first caller only).

Results

You always have a trade off between performance, feature, and security. Security typically adds more processing time to your requests, but for a good reason. Not all security features are required in your environment. When you decide to tune security, create a benchmark before making any change to ensure that the change is improving performance.

What to do next

In a large scale deployment, performance is very important. Running benchmark measurements with different combinations of features can help you to determine the best performance versus the benefit of configuration for your environment. Continue to run benchmarks if anything changes in your environment, to help determine the impact of these changes.

Secure Sockets Layer performance tips

Use this page to learn about Secure Sockets Layer (SSL) performance tips. Be sure to consider that performance issues typically involve trade-offs between function and speed. Usually, the more function and the more processing that are involved, the slower the performance.

The following are two types of Secure Sockets Layer (SSL) performance:
- Handshake
- Bulk encryption and decryption

When an SSL connection is established, an SSL handshake occurs. Once a connection is made, SSL performs bulk encryption and decryption for each read-write. The performance cost of an SSL handshake is much larger than that of bulk encryption and decryption.

To enhance SSL performance, decrease the number of individual SSL connections and handshakes.

Decreasing the number of connections increases performance for secure communication through SSL connections, as well as non-secure communication through simple Transmission Control Protocol/Internet Protocol (TCP/IP) connections. One way to decrease individual SSL connections is to use a browser that supports HTTP 1.1. Decreasing individual SSL connections can be impossible if you cannot upgrade to HTTP 1.1.

Another common approach is to decrease the number of connections (both TCP/IP and SSL) between two WebSphere Application Server components. The following guidelines help to verify the HTTP transport of the application server is configured so that the Web server plug-in does not repeatedly reopen new connections to the application server:

- Verify that the maximum number of keep alives are, at minimum, as large as the maximum number of requests per thread of the web server (or maximum number of processes for IBM HTTP Server on UNIX). Make sure that the web server plug-in is capable of obtaining a keep alive connection for every possible concurrent connection to the application server. Otherwise, the application server closes the connection once a single request is processed. Also, the maximum number of threads in the web container thread pool should be larger than the maximum number of keep alives, to prevent the keep alive connections from consuming the web container threads.

  Note: HTTP Transports have been deprecated. For instructions on how to set a maximum keep alive value for channel based configurations, see HTTP transport channel settings.
- Increase the maximum number of requests per keep alive connection. The default value is 100, which means the application server closes the connection from the plug-in following 100 requests. The plug-in then has to open a new connection. The purpose of this parameter is to prevent denial of service attacks when connecting to the application server and preventing continuous send requests to tie up threads in the application server.
- Use a hardware accelerator if the system performs several SSL handshakes.

  Hardware accelerators currently supported by WebSphere Application Server only increase the SSL handshake performance, not the bulk encryption and decryption. An accelerator typically only benefits the web server because Web server connections are short-lived. All other SSL connections in WebSphere Application Server are long-lived.
- Use an alternative cipher suite with better performance.

  The performance of a cipher suite is different with software and hardware. Just because a cipher suite performs better in software does not mean a cipher suite will perform better with hardware. Some algorithms are typically inefficient in hardware, for example, Data Encryption Standard (DES) and triple-strength DES (3DES); however, specialized hardware can provide efficient implementations of these same algorithms.

  The performance of bulk encryption and decryption is affected by the cipher suite used for an individual SSL connection. The following chart displays the performance of each cipher suite. The test software calculating the data was Java Secure Socket Extension (JSSE) for both the client and server software, which used no cryptographic hardware support. The test did not include the time to establish a connection, but only the time to transmit data through an established connection. Therefore, the data reveals the relative SSL performance of various cipher suites for long running connections.

  Prior to establishing a connection, the client enables a single cipher suite for each test case. After the connection is established, the client times how long it takes to write an integer to the server and for the server to write the specified number of bytes back to the client. Varying the amount of data had negligible effects on the relative performance of the cipher suites.

An analysis of the previous data reveals the following:

- Bulk encryption performance is only affected by what follows the WITH in the cipher suite name. This is expected since the portion preceding the WITH identifies the algorithm used only during the SSL handshake.

- MD5 and Secure Hash Algorithm (SHA) are the two hash algorithms used to provide data integrity. MD5 is generally faster than SHA, however, SHA is more secure than MD5.
- DES and RC2 are slower than RC4. Triple DES is the most secure, but the performance cost is high when using only software.
- The cipher suite providing the best performance while still providing privacy is SSL_RSA_WITH_RC4_128_MD5. Even though SSL_RSA_EXPORT_WITH_RC4_40_MD5 is cryptographically weaker than RSA_WITH_RC4_128_MD5, the performance for bulk encryption is the same. Therefore, as long as the SSL connection is a long-running connection, the difference in the performance of high and medium security levels is negligible. It is recommended that a security level of high be used, instead of medium, for all components participating in communication only among WebSphere Application Server products. Make sure that the connections are long running connections.

Tuning security performance

Use the following procedures to tune the performance, without compromising your security settings.

About this task

Enabling security decreases performance. The following tuning parameters provide ways to minimize this performance impact.

Procedure

- Disable security on any application servers that do not need security. You can disable security in the administrative console by clicking **Security > Global security** and deselecting the **Enable administrative security** option.
- Fine-tune the **Authentication cache timeout** value on the Authentication mechanisms and expiration panel in the administrative console. For more information, see the Global security settings topic.
- Configure the security cache properties. For more information, see the Authentication cache settings topic.
- Enable the **Enable SSL ID tracking** option on the Session management panel in the administrative console.
- Improve the performance of Web Services Security by downloading a Java Cryptography Extension (JCE) unlimited jurisdiction policy file that does not have restrictions on cryptography strength. See the information about tuning Web Services Security for Version 8.0 applications for details.
- Read the Secure Sockets Layer performance tips and “Tuning security configurations” on page 65 topics for more information.

Hardening security configurations

There are several methods that you can use to protect the WebSphere Application Server infrastructure and applications from different forms of attack. Several different techniques can help with multiple forms of attack. Sometimes a single attack can leverage multiple forms of intrusion to achieve the end goal.

About this task

For example, in the simplest case, network sniffing can be used to obtain passwords and those passwords can then be used to mount an application-level attack. The following issues are discussed in IBM WebSphere Developer Technical Journal: WebSphere Application Server V5 advanced security and system hardening:

Procedure

- Take preventative measures to protect the infrastructure.
- Make applications less vulnerable to attack.

- At a minimum, ensure administrative security is enabled in all WebSphere processes. This protects access to the administrative ConfigService interface and managed beans (MBeans) that enables control over the WebSphere process if it is compromised.
- Ensure Secure Sockets Layer (SSL) is used whenever possible, and mutual SSL whenever possible. However, mutual SSL requires all clients to supply a trusted personal certificate in order to connect.
- Remove any unnecessary certificate authority (CA) signer certificates from your trust stores.
- Change default keystore passwords during or after profile creation using the `AdminTask changeMultipleKeyStorePasswords` command.
- Change your Lightweight Third-Party Authentication (LTPA) keys periodically. You can configure the automatic regeneration of LTPA keys if necessary.
- Common Secure Interoperability version 2 (CSIv2) inbound Basic authentication is supported in this release of WebSphere Application Server. The authentication default is 'required'.

What to do next

Note: In this release of WebSphere Application Server, more security hardening features of the server are enabled by default. However, if the features are not enabled after migration you can enable them yourself. See the Security hardening features enablement and migration article for more information.

For additional information about hardening security configurations, see the WebSphere Application Server security web page.

Enablement and migration considerations of Security hardening features

In this release of WebSphere Application Server, more security hardening features of the server are enabled out-of-the-box by default. When migrating, the settings that were enabled prior to migration are retained. However, if the features are not enabled after migration you can enable them yourself.

To ensure that WebSphere Application Server configuration is set to be secure by default, the following defaults have been changed as part of the new security hardening features in WebSphere Application Server Version 8.0:

- Enablement of Secure Sockets Layer (SSL)-required on Common Secure Interoperability version 2 (CSIv2) transport by default

  The following settings for the CSIv2 transport layer exist: TCP/IP for a TCP/IP connection, SSL-supported for a TCP/IP or an SSL connection, and SSL-required for an SSL connection only. SSL-required is the new default in this release of WebSphere Application Server. Switching to SSL-required as the default setting ensures that all CSIv2 connections into and out of the server are using the secure SSL connection.
- Enablement of the HttpOnly attribute on LTPA cookies by default

  When the com.ibm.ws.security.addHttpOnlyAttributeToCookies custom property is set to `true`, the HttpOnly attribute is added to those security cookies (LTPA and WASReqURL cookies) that are created by the server. The HttpOnly attribute is a browser attribute created to prevent client side applications (such as Java scripts) from accessing cookies to prevent some cross-site scripting vulnerabilities. This attribute is now configurable in the administrative console. Prior to WebSphere Application Server Version 8.0, the com.ibm.ws.security.addHttpOnlyAttributeToCookies custom property default was `false`. For WebSphere Application Server Version 8.0, the default is now `true` for both the LTPA cookie and the Session Cookie.

  For more information see the custom property com.ibm.ws.security.addHttpOnlyAttributeToCookies in the Security custom properties article.
- Enablement of session security integration by default

Only authenticated users can access sessions created in secure pages. The session management facility uses the security infrastructure to determine the authenticated identity associated with a client HTTP request, and either retrieves or creates a session. For more information on session security, read the Session security support article.

Along with enabling session security integration, credential persistence is enabled as well. This allows login information to be available to unprotected web clients to enable additional access to user information. For more information on credential persistence, see the “Use available authentication data when an unprotected URI is accessed” feature in the web authentication settings article.

Enabling the new security hardening features after migration

If the new security features are not enabled after migration, you can enable them yourself using the administrative console or by scripting.

Enablement of SSL by default on CSIv2

To enable SSL by default for inbound and outbound transports on CSIv2:

If you are using the administrative console, select **Security** > **Global security** > **RMI/IIOP** > **CSIv2 inbound communications**. In the Transport box, select **SSL- required** from the pull-down list and then click **Apply**.

Repeat the same steps for CSIv2 outbound communications and click **Security** > **Global security** > **RMI/IIOP** > **CSIv2 outbound communications**. In the Transport box, select **SSL- required** from the menu list and then click **Apply**.

If you want to enable SSL by default for inbound and outbound transports on CSIv2 using scripting, use the configureCSIInbound and configureCSIOutbound commands. See the Configuring Common Secure Interoperability authentication using scripting topic for more information.

For the client side, edit the sas.client.props file. Change com.ibm.CSI.performTransportAssocSSLTLSRequired to `true` and change com.ibm.CSI.performTransportAssocSSLTLSSupported to `false`.

Enablement of the HttpOnly cookie attribute

To enable the HttpOnly attribute on cookies attribute by default:

If you are using the administrative console, click **Security** > **Global security** > **Custom properties**. Click**New** and enter `com.ibm.ws.security.addHttpOnlyAttributeToCookies` for the Name and `true` for the Value.

You can also enable the **HttpOnly** attribute using the administrative console by clicking **Security** > **Global security** > **Single sign-on (SSO)**. Click Set security cookies to **HTTPOnly** to help prevent cross-site scripting attacks, and then click **Apply**.

To enable the HttpOnly attribute on cookies attribute by default using scripting, use the setAdminActiveSecuritySettings command.

Enablement of session security integration

To enable session security integration for each server by using the administrative console, select **Servers** > **Server types** > **WebSphere application servers** > **server1** > **Session management**. Select the **security integration** check box.

To enable persisting credentials from the administrative console, click **Security** > **Global security** > **Web and SIP security** > **General settings**. Select the **Use available authentication data when an unprotected URI is accessed** check box.

Security hardening features enablement troubleshooting

When the new security hardening features are enabled you might see some differences in system behavior depending upon which environment you might have used in the past.

For example, if you are coming from an environment where CSIv2 transport was set to the previous default of SSL-supported, you do not experience any differences, as SSL-supported communicates with both TCP/IP and SSL connections. If a problem is encountered, however, certificates might not have been exchanged correctly to enable the client and server to communicate. Read about the Secure communications using Secure Sockets Layer (SSL) topic for more information.

If you worked in an environment where TCP/IP is used for the connection to CSIv2, you might experience connection problem to the SSL-enabled CSIv2 connection. The server configuration can be modified to SSL-supported or to TCP/IP if SSL is not required.

For the HttpOnly attribute, when the attribute is added to the security cookies, the browser prevents client side scripts from accessing these cookies. In most case this should be the default behavior to minimize cross-site scripting vulnerabilities. If there is an absolute need to allow client-side scripts to access WebSphere security cookies, and you are aware of the possible consequences, then the setting of the HttpOnly attribute can be disabled.

However, the HttpOnly attribute can possibly uncover client-side scripts that are used to access WebSphere cookies, and can then use them even though it was not intended to do so. If this happens, the web application that enables the scripts to access the WebSphere cookies must be evaluated.

For session security integration enablement, when session integrated security is enabled you might receive an UnauthorizedSessionRequestException exception on servlets if they access a session that belongs to authenticated identities other than to the identity that currently owns the session. If you do not want this checking to occur, you can disable session security from the server that is experiencing the problem.

Securing passwords in files

Password encoding and encryption deters the casual observation of passwords in server configuration and property files.

About this task

The following topics can be used to add protection for passwords located in files:

Procedure

- Encoding passwords in files WebSphere Application Server contains some encoded passwords that are not encrypted. The **PropFilePasswordEncoder** utility is included to encode these passwords. For more information on encoding passwords in a file, see “Encoding passwords in files.”
- Enabling custom password encryption You need to protect passwords that are contained in your WebSphere Application Server configuration. You can added protection by creating a custom class for encrypting the passwords. For more information on custom password encryption, see “Enabling custom password encryption” on page 77.

Encoding passwords in files

The purpose of password encoding is to deter casual observation of passwords in server configuration and property files. Use the **PropFilePasswordEncoder** utility to encode passwords stored in properties files. WebSphere Application Server does not provide a utility for decoding the passwords. Encoding is not sufficient to fully protect passwords. Native security is the primary mechanism for protecting passwords used in WebSphere Application Server configuration and property files.

About this task

WebSphere Application Server contains several encoded passwords in files that are not encrypted. WebSphere Application Server provides the **PropFilePasswordEncoder** utility, which you can use to

encode passwords. The purpose of password encoding is to deter casual observation of passwords in server configuration and property files. The **PropFilePasswordEncoder** utility does not encode passwords that are contained within XML or XMI files.

Table 11. XML and XMI files that contain encoded passwords. Instead, WebSphere Application Server automatically encodes the passwords in these files. XML and XMI files that contain encoded passwords include the following:

File name	Additional information
profile_root/config/cells/*cell_name*/security.xml	The following fields contain encoded passwords: • LTPA password • JAAS authentication data • User registry server password • LDAP user registry bind password • Keystore password • Truststore password • Cryptographic token device password
war/WEB-INF/ibm_web_bnd.xml	Specifies the passwords for the default basic authentication for the resource-ref bindings within all the descriptors, except in the Java cryptography architecture
ejb jar/META-INF/ibm_ejbjar_bnd.xml	Specifies the passwords for the default basic authentication for the resource-ref bindings within all the descriptors, except in the Java cryptography architecture
client jar/META-INF/ibm-appclient_bnd.xml	Specifies the passwords for the default basic authentication for the resource-ref bindings within all the descriptors, except in the Java cryptography architecture
ear/META-INF/ibm_application_bnd.xml	Specifies the passwords for the default basic authentication for the run as bindings within all the descriptors
profile_root/config/cells/*cell_name* /nodes/*node_name*/servers/ *server_name*/security.xml	The following fields contain encoded passwords: • Keystore password • Truststore password • Cryptographic token device password • Session persistence password
profile_root/config/cells/*cell_name* /nodes/*node_name*/servers/ *server_name*/resources.xml	The following fields contain encoded passwords: • WAS40Datasource password • mailTransport password • mailStore password • MQQueue queue mgr password
• *profile_root*/config/cells/*cell_name* /ws-security.xml • *profile_root*/config/cells/*cell_name* /nodes/*node_name*/servers/*server_name*/ws-security	
ibm-webservices-bnd.xmi	
ibm-webservicesclient-bnd.xmi	

Table 12. The PropFilePasswordEncoder utility - Partial File List. You use the **PropFilePasswordEncoder** *utility to encode the passwords in properties files. These files include:*

File name	Additional information
profile_root /properties/sas.client.props	Specifies the passwords for the following files: • com.ibm.ssl.keyStorePassword • com.ibm.ssl.trustStorePassword • com.ibm.CORBA.loginPassword
profile_root /properties/sas.tools.properties	Specifies passwords for: • com.ibm.ssl.keyStorePassword • com.ibm.ssl.trustStorePassword • com.ibm.CORBA.loginPassword
profile_root /properties/sas.stdclient.properties	Specifies passwords for: • com.ibm.ssl.keyStorePassword • com.ibm.ssl.trustStorePassword • com.ibm.CORBA.loginPassword
profile_root /properties/wsserver.key	
profile_root/profiles/AppSrvXX/properties/sib.client.ssl.properties	Specifies passwords for: • com.ibm.ssl.keyStorePassword • com.ibm.ssl.trustStorePassword

Table 12. The PropFilePasswordEncoder utility - Partial File List (continued). You use the **PropFilePasswordEncoder** *utility to encode the passwords in properties files. These files include:*

File name	Additional information
profile_root/UDDIReg/scripts/UDDIUtilityTools.properties	Specifies passwords for: • trustStore.password

To encode a password again in one of the previous files, complete the following steps:

Procedure

1. Access the file using a text editor and type over the encoded password. The new password is shown is no longer encoded and must be re-encoded.
2. Use the `PropFilePasswordEncoder.bat` or the `PropFilePasswordEncode.sh` file in the *profile_root*`/bin` directory to encode the password again.

   If you are encoding files that are not SAS properties files, type `PropFilePasswordEncoder` *"file_name" password_properties_list*

   Important: When you use the **PropFilePasswordEncoder** utility, a prompt asks whether a backup version of the original file is required. If a backup version is required, a backup file (.bak), is created with the clear text password. Examine the results and then delete this backup file. It contains the unencrypted password. If you do not want to see this prompt, edit the PropFilePasswordEncoder utility and add the following Java system property as a parameter: `-Dcom.ibm.websphere.security.util.createBackup=true` or `-Dcom.ibm.websphere.security.util.createBackup=false`

   A `true` value for the Java system property creates a backup file and a `false` value disables the backup file.

   where:

   "file_name" is the name of the z/SAS properties file, and *password_properties_list* is the name of the properties to encode within the file.

   Note: Only the password should be encoded in this file using the **PropFilePasswordEncoder** tool.

   Use the **PropFilePasswordEncoder** utility to encode WebSphere Application Server password files only. The utility cannot encode passwords that are contained in XML files or other files that contain open and close tags. To change passwords in these files, use the administrative console or an assembly tool such as the Rational Application Developer.

Results

If you reopen the affected files, the passwords are encoded. WebSphere Application Server does not provide a utility for decoding the passwords.

Example

The following example shows how to use the **PropFilePasswordEncoder** tool:

```
PropFilePasswordEncoder C:\WASV8\WebSphere\AppServer\profiles\AppSrv\properties
\sas.client.props com.ibm.ssl.keyStorePassword,com.ibm.ssl.trustStorePassword
```

where:

PropFilePasswordEncoder is the name of the utility that you are running from the profile_root/profiles/ profile_name/bin directory.

`C:\WASV6\WebSphere\AppServer\profiles\AppSrv\properties\sas.client.props` is the name of the file that contains the passwords to encode.

com.ibm.ssl.keyStorePassword is a password to encode in the file.

com.ibm.ssl.trustStorePassword is a second password to encode in the file.

PropFilePasswordEncoder command reference:

The **PropFilePasswordEncoder** command encodes passwords that are located in plain text property files. This command encodes both Secure Authentication Server (SAS) property files and non-SAS property files. After you encode the passwords, a decoding command does not exist.

Note: If you need to custom encode passwords in property files, manually edit the PropFilePasswordEncoder.sh or PropFilePasswordEncoder.bat file before issuing this command. See the topic Implementing custom password encryption for a description of the lines that need to be added to this file.

Note: To enable PropFilePasswordEncoder to print out more a debug message than in previous releases, update the command by entering the following:

```
-Dcom.ibm.websphere.security.passwordEncoderDebug=true
```

Syntax

The command syntax is as follows:

```
PropFilePasswordEncoder "file_name" { passwordPropertiesList | -SAS } { -noBackup | -Backup }
    [ -profileName profile ] [ -help | -? ]
```

Parameters

The following option is available for the **PropFilePasswordEncoder** command:

file_name
This required parameter specifies the name of the file in which passwords are encoded.

passwordPropertiesList
This parameter is required if you are encoding passwords in property files other than the `sas.client.props` file. Specify one or more password properties that you want to encode. The password properties list should be delimited by commas.

-SAS
This parameter is required if you are encoding passwords in the `sas.client.props` file.

-noBackup
This parameter is optional and the default. The script does not create a backup file. The default value can be altered by adding following Java System Property: `"-Dcom.ibm.websphere.security.util.createBackup=true"`.

-Backup
This parameter is optional. The script creates a backup file, `<file_name>.bak`, which contains passwords in clear text.

-profileName
This parameter is optional. The profile value specifies an application server profile name. The script uses the password encoding algorithm that it retrieves from the specified profile. If you do not specify this parameter, the script uses the default profile.

-help or -?
If you specify this parameter, the script ignores all other parameters and displays usage text.

Enabling custom password encryption

You need to protect passwords that are contained in your WebSphere Application Server configuration. After creating your server profile, you can added protection by creating a custom class for encrypting the passwords.

Before you begin

Create your custom class for encrypting passwords. For more information, see Plug point for custom password encryption.

About this task

Complete the following steps to enable custom password encryption.

Procedure

1. Add the following system properties for every server and client process. For server processes, update the `server.xml` file for each process. Add these properties as a genericJvmArgument argument preceded by a **-D** prefix.

   ```
   com.ibm.wsspi.security.crypto.customPasswordEncryptionClass=
          com.acme.myPasswordEncryptionClass
   com.ibm.wsspi.security.crypto.customPasswordEncryptionEnabled=true
   ```

   Tip: If the custom encryption class name is com.ibm.wsspi.security.crypto.CustomPasswordEncryptionImpl, it is automatically enabled when this class is present in the classpath. Do not define the system properties that are listed previously when the custom implementation has this package and class name. To disable encryption for this class, you must specify `com.ibm.wsspi.security.crypto.customPasswordEncryptionEnabled=false` as a system property.
2. Choose one of the following methods to configure the WebSphere Application Server runtime to load the custom encryption implementation class:
   - Place the custom encryption class in a Java archive (JAR) file that resides in the `${WAS_INSTALL_ROOT}/classes` directory, which you have created.

     gotcha: WebSphere Application Server does not create the `${WAS_INSTALL_ROOT}/classes` directory. For more information on the classes directory, see the topic, "Creating a classes subdirectory in your profile for custom classes".
   - Place the custom encryption class in a Java archive (JAR) file that resides in the `${WAS_HOME}/lib/ext` directory.
3. Restart all server processes.
4. Edit each configuration document that contains a password and save the configuration. All password fields are then run through the **WSEncoderDecoder** utility, which calls the plug point when it is enabled. The `{custom:alias}` tags are displayed in the configuration documents. The passwords, even though they are encrypted, are still Base64-encoded. They seem similar to encoded passwords, except for the tags difference.
5. Encrypt any passwords that are in client-side property files using the **PropsFilePasswordEncoder** (.bat or .sh) utility. This utility requires that the properties listed previously are defined as system properties in the script to encrypt new passwords instead of encoding them.
6. To decrypt passwords from client Java virtual machines (JVMs), add the properties listed previously as system properties for each client utility.
7. Ensure that all nodes have the custom encryption classes in their class paths prior to enabling this function.

Results

Custom password encryption is enabled.

What to do next

If custom password encryption fails or is no longer required, see “Disabling custom password encryption.”

Disabling custom password encryption:

If custom password encryption fails or is no longer required, perform this task to disable custom password encryption.

Before you begin

Enable custom password encryption.

About this task

Complete the following steps to disable custom password encryption.

Procedure

1. Change the com.ibm.wsspi.security.crypto.customPasswordEncryptionEnabled property to be `false` in the `security.xml` file, but leave the com.ibm.wsspi.security.crypto.customPasswordEncryptionClass property configured. Any passwords in the model that still have the {`custom:alias`} tag are decrypted by using the customer password encryption class.
2. If an encryption key is lost, any passwords that are encrypted with that key cannot be retrieved. To recover a password, retype the password in the password field in plaintext and save the document. The new password must be written out using encoding with the {xor} tag with scripting or from the administrative console.

   ```
   com.ibm.wsspi.security.crypto.customPasswordEncryptionClass=
          com.acme.myPasswordEncryptionClass
   com.ibm.wsspi.security.crypto.customPasswordEncryptionEnabled=false
   ```
3. Restart all processes to make the changes effective.
4. Edit each configuration document that contains an encrypted password and save the configuration. All password fields are then run through the **WSEncoderDecoder** utility, which calls the plug point in the presence of the {`custom:alias`} tag. The {`xor`} tags display in the configuration documents again after the documents are saved.
5. Decrypt and encode any passwords that are in client-side property files using the **PropsFilePasswordEncoder** (.bat or .sh) utility. If the encryption class is specified, but custom encryption is disabled, running this utility converts the encryption to encoding and causes the {`xor`} tags to display again.
6. Disable custom password encryption from the client Java virtual machines (JVMs) by adding the system properties listed previously to all client scripts. This action enables the code to decrypt passwords, but this action is not used to encrypt them again. The {`xor`} algorithm becomes the default for encoding. Leave the custom password encryption class defined for a time in case any encrypted passwords still exist in the configuration.

Results

Custom password encryption is disabled.

Chapter 9. Tuning Session Initiation Protocol (SIP) applications

This page provides a starting point for finding information about SIP applications, which are Java programs that use at least one Session Initiation Protocol (SIP) servlet written to the JSR 116 specification.

SIP is used to establish, modify, and terminate multimedia IP sessions including IP telephony, presence, and instant messaging.

Tuning SIP servlets for Linux

This page describes preliminary SIP servlet tuning for Linux 2.6 kernel.

About this task

A Session Initiation Protocol (SIP) servlet under load might retransmit messages or drop calls. The UDP socket queues might fill. A review of the verbose garbage collection output might show that there are fairly long garbage collection times, for example, 0.5 to 1.5 seconds. The cause of this problem is that the Ethernet driver, Linux® operating system, WebSphere® Application Server, or any combination of the items are not tuned for SIP applications. You can apply the following levels of tuning.

Note: The following recommendations have been tested on Red Hat Enterprise Linux 4 only and are provided as is without any implied warranty.

Procedure

1. Tuning the Linux Ethernet driver.

   Linux Ethernet driver tuning begins by selecting the best Ethernet driver. For example, the HS20 blades recommended driver is the tg3-3.43b driver (or later), which can be found at the website for Broadcom Ethernet NIC Driver Downloads. The following shell commands have been used to tune the Linux kernel Ethernet driver:

   ```
   /sbin/ifconfig eth0 txqueuelen 2000
   /sbin/ifconfig eth1 txqueuelen 2000
   ethtool -s eth0 autoneg off speed 1000 duplex full
   ethtool -A eth0 autoneg off rx on tx on
   ethtool -C eth0 adaptive-rx off adaptive-tx off rx-
   usecs 20 rx-frames 5 tx-usecs 60 tx-frames 11
   ethtool -G eth0 rx 511 rx-jumbo 255 tx 511
   ```

   Depending upon the Ethernet driver that is installed, some of these options might need to change.

2. Tuning the Linux kernel.

   Linux kernel tuning uses the following commands:

   ```
   echo 16777216 > /proc/sys/net/core/rmem_max
   echo 2097152 > /proc/sys/net/core/rmem_default
   echo 16777216 > /proc/sys/net/core/wmem_max
   echo 2097152 > /proc/sys/net/core/wmem_default
   echo 10000000 > /proc/sys/net/core/optmem_max
   echo 4096 87380 16777216 > /proc/sys/net/ipv4/tcp_rmem
   echo 4096 87380 16777216 > /proc/sys/net/ipv4/tcp_wmem
   echo 8388608 8388608 8388608 > /proc/sys/net/ipv4/tcp_mem
   echo 400 > /proc/sys/net/unix/max_dgram_qlen
   echo 400 > /proc/sys/net/core/message_burst
   echo 2800 > /proc/sys/net/core/mod_cong
   echo 1000 > /proc/sys/net/core/lo_cong
   echo 200 > /proc/sys/net/core/no_cong
   echo 2900 > /proc/sys/net/core/no_cong_thresh
   echo 3000 > /proc/sys/net/core/netdev_max_backlog
   ```

This configuration might not be optimum for a given application and you might need to adjust the configuration to achieve the best performance. However, you might use these values as a starting point.

3. SIP tuning for WebSphere Application Server. SIP tuning for WebSphere Application Server is completed using the following steps:
   a. Create a separate thread pool for the SIP servlet container. Follow this path in the administrative console:
      1) Click **Servers** > **Server Types** > **WebSphere application servers** > ***server_name***.
      2) Under Additional properties, click **Thread Pools > New**.
      3) In the Name field, enter `SipContainer`.
      4) In the **Minimum Size** and **Maximum Size** fields, enter `15`. These values should be adequate for most applications.
      5) Click **OK**.
   b. Create custom properties for the SIP Servlet container. Follow this path in the administrative console:
      1) Click **Servers** > **Server Types** > **WebSphere application servers** > ***server_name***.
      2) Click **SIP container**.
      3) Under **Additional properties**, click **Custom Properties > New**.
      4) In the **Name** field, enter `javax.sip.max.object.pool.size`.
      5) In the **Value** field, enter `1000`.
      6) Click **OK**.
      7) In the **Name** field, enter `max.tu.pool.size`.
      8) In the **Value** field, enter `1000`.
      9) Click **OK**.
      10) In the Name field, enter `com.ibm.sip.sm.lnm.size`.
      11) In the Value field, enter `8`.
      12) Click **OK**.
   c. Create custom properties for the SIPUDP channel if User Datagram Protocol (UDP) is the primary transport for SIP traffic. Follow this path in the administrative console:
      1) Click **Servers** > **Server Types** > **WebSphere application servers** > ***server_name***.
      2) Click **SIP container > Transport Chain > SIPCInboundDefaultUDP > UDP Inbound channel (UDP1)**.
      3) Under **Additonal Properties**, click **Custom Properties > New**.
      4) In the **Name** field, enter `receiveBufferSizeSocket`.
      5) In the **Value** field, enter `3000000`.
      6) Click OK.
      7) In the **Name** field, enter `sendBufferSizeSocket`.
      8) In the **Value** field, enter `3000000`.
   d. Specify the SIP servlet container general properties. Follow this path in the administrative console:
      1) Click **Servers** > **Server Types** > **WebSphere application servers** > ***server_name***.
      2) Enter the Maximum application sessions value. The Maximum application sessions value can be calculated as: *Maximum call hold time or session timeout x Call rate x Safety factor*.
      3) Enter the Maximum messages per averaging period value. The Maximum messages per averaging period value can be calculated as: *Maximum call hold time or session timeout x Maximum rate of SIP messages x Safety factor*.

4) Enter the Maximum dispatch queue size value. The Maximum dispatch queue size value can be calculated as: *Maximum rate of SIP messages x Maximum latency in SIP processing x Safety factor*.
5) Set the thread pool to the newly created SIP container thread pool (to the drop down name "SipContainer").

e. Tune the Java virtual machine (JVM) garbage collection policy. Follow this path in the administrative console:
1) Click **Servers** > **Server Types** > **WebSphere application servers** > ***server_name***.
2) Under Server Infrastructure, click **Java and Process Management** > **Process Definition**.
3) Under **Additional Properties**, click **Java Virtual Machine**.
4) In the Generic JVM arguments field, enter the following value as one continuous line: "`-Xmn150m -Xgcpolicy:gencon -Xgc:scvNoAdaptiveTenure,scvTenureAge=1,stdGlobalCompactToSatisfyAllocate -Xcompactexplicitgc -XX:MaxDirectMemorySize=256000000`".

Note:

- You might add a value of 1500 MB to the initial heap size and Maximum heap size fields.
- It is recommended that you enable the **Verbose garbage collection** option during performance testing or tuning operations.
- If your application allocates objects greater than 64 KB in size, it might be beneficial to reserve a large object area (LOA) in the heap. This is done by adding the JVM argument -Xloaminimum0.*xy*, where *xy* indicates the percentage of the heap to reserve for large objects. For more information about the JVM arguments, refer to the Java Diagnostics Guide 6.

Chapter 10. Tuning web applications

This page provides a starting point for finding information about web applications, which are comprised of one or more related files that you can manage as a unit, including:

- HTML files
- Servlets can support dynamic web page content, provide database access, serve multiple clients at one time, and filter data.
- Java ServerPages (JSP) files enable the separation of the HTML code from the business logic in web pages.

IBM extensions to the JSP specification make it easy for HTML authors to add the power of Java technology to web pages, without being experts in Java programming. More introduction...

Tuning URL cache

Tuning URL invocation cache

The URL invocation cache holds information for mapping request URLs to servlet resources. This cache is Web container-based, and shared for all Web container threads. A cache of the requested size is created for each Web container thread that is available to process a request. The default size of the invocation cache is 50. If more than 50 unique URLs are actively being used (each JavaServer Page is a unique URL), you should increase the size of the invocation cache.

Before you begin

A larger cache uses more of the Java heap, so you might also need to increase the maximum Java heap size. For example, if each cache entry requires 2KB, maximum thread size is set to 25, and the URL invocation cache size is 100; then 5MB of Java heap are required.

About this task

To change the size of the invocation cache:

Procedure

1. In the administrative console, click **Servers > Server Types > WebSphere application servers** and select the application server that you are tuning.
2. Click **Java and Process Management**.
3. Click **Process Definition** under Additional Properties.
4. Click **Java Virtual Machine** under Additional Properties.
5. Click **Custom Properties** under Additional Properties.
6. Specify **invocationCacheSize** in the Name field and the size of the cache in the Value field. The default size for the invocation cache is 500 entries. Since the invocation cache is no longer thread-based, the invocation cache size specified by the user is multiplied by ten to provide similar function from previous releases. For example, if you specify an invocation cache size of 50, the web container will create a cache size of 500.
7. Click **Apply** and then **Save** to save your changes.
8. Stop and restart the application server.

Results

The new cache size is used for the URL invocation cache.

Tuning sessions

Session management tuning

WebSphere Application Server session support has features for tuning session performance and operating characteristics, particularly when sessions are configured in a distributed environment. These options support the administrator flexibility in determining the performance and failover characteristics for their environment.

Table 13. Summary of tuning features. The following table summarizes the tuning features, including whether they apply to sessions tracked in memory, in a database, with memory-to-memory replication, or all. Some features are easily manipulated using administrative settings; others require code or database changes.

Feature or option	Goal	Applies to sessions in memory, database, or memory-to-memory
Write frequency	Minimize database write operations.	Database
Session affinity	Access the session in the same application server instance.	All
Multirow schema	Fully utilize database capacities.	Database
Base in-memory session pool size	Fully utilize system capacity without overburdening system.	All
Write contents	Allow flexibility in determining what session data to write	Database
Scheduled invalidation	Minimize contention between session requests and invalidation of sessions by the Session Management facility. Minimize write operations to database for updates to last access time only.	Database
Tablespace and row size	Increase efficiency of write operations to database.	Database (DB2 only)

Tuning parameter settings

Use this page to set tuning parameters for distributed sessions.

To view this administrative console page, click **Servers** > **Server Types** > **WebSphere application servers** > ***server_name*** > **Session management** > **Distributed environment settings** > **Custom tuning parameters**.

Tuning level

Specifies that the session management facility provides certain predefined settings that affect performance.

Select one of these predefined settings or customize a setting. To customize a setting, select one of the predefined settings that comes closest to the setting desired, click **Custom settings**, make your changes, and then click **OK**.

Very high (optimize for performance)

Information	Value
Write frequency	Time based
Write interval	300 seconds
Write contents	Only updated attributes
Schedule sessions cleanup	true
First time of day default	0

Information	Value
Second time of day default	2

High

Information	Value
Write frequency	Time based
Write interval	300 seconds
Write contents	All session attributes
Schedule sessions cleanup	false

Medium

Information	Value
Write frequency	End of servlet service
Write contents	Only updated attributes
Schedule sessions cleanup	false

Low (optimize for failover)

Information	Value
Write frequency	End of servlet service
Write contents	All session attributes
Schedule sessions cleanup	false

Custom settings

Information	Value
Write frequency default	Time based
Write interval default	10 seconds
Write contents default	All session attributes
Schedule sessions cleanup default	false

Tuning parameter custom settings

Use this page to customize tuning parameters for distributed sessions.

To view this administrative console page, click **Servers** > **Server Types** > **WebSphere application servers** > ***server_name*** > **Session management** > **Distributed environment settings** > **Custom tuning parameters** > **Custom settings**.

Write frequency

Specifies when the session is written to the persistent store.

Information	Value
End of servlet service	A session writes to a database or another WebSphere Application Server instance after the servlet completes execution.
Manual update	A programmatic sync on the IBMSession object is required to write the session data to the database or another WebSphere Application Server instance.
Time based	Session data writes to the database or another WebSphere Application Server instance based on the specified Write interval value. Default: 10 seconds

Write contents

Specifies whether updated attributes are only written to the external location or all of the session attributes are written to the external location, regardless of whether or not they changed. The external location can be either a database or another application server instance.

Information	Value
Only updated attributes	Only updated attributes are written to the persistent store.
All session attribute	All attributes are written to the persistent store.

Schedule sessions cleanup

Specifies when to clean the invalid sessions from a database or another application server instance.

Information	Value
Specify distributed sessions cleanup schedule	Enables the scheduled invalidation process for cleaning up the invalidated HTTP sessions from the external location. Enable this option to reduce the number of updates to a database or another application server instance required to keep the HTTP sessions alive. When this option is not enabled, the invalidator process runs every few minutes to remove invalidated HTTP sessions. When this option is enabled, specify the two hours of a day for the process to clean up the invalidated sessions in the external location. Specify the times when there is the least activity in the application servers. An external location can be either a database or another application server instance.
First Time of Day (0 - 23)	Indicates the first hour during which the invalidated sessions are cleared from the external location. Specify this value as a positive integer between 0 and 23. This value is valid only when schedule invalidation is enabled.
Second Time of Day (0 - 23)	Indicates the second hour during which the invalidated sessions are cleared from the external location. Specify this value as a positive integer between 0 and 23. This value is valid only when schedule invalidation is enabled.

Best practices for using HTTP sessions

This topic presents best practices for the implementation of HTTP sessions.

best-practices: Browse the following recommendations for implementing HTTP sessions.

- Enable Security integration for securing HTTP sessions

  HTTP sessions are identified by session IDs. A session ID is a pseudo-random number generated at the runtime. Session hijacking is a known attack HTTP sessions and can be prevented if all the requests going over the network are enforced to be over a secure connection (meaning, HTTPS). But not every configuration in a customer environment enforces this constraint because of the performance impact of SSL connections. Due to this relaxed mode, HTTP session is vulnerable to hijacking and because of this vulnerability, WebSphere Application Server has the option to tightly integrate HTTP sessions and WebSphere Application Server security. Enable security in WebSphere Application Server so that the sessions are protected in a manner that only users who created the sessions are allowed to access them.
- Release HttpSession objects using `javax.servlet.http.HttpSession.invalidate()` when finished.

  HttpSession objects live inside the web container until:

  - The application explicitly and programmatically releases it using the javax.servlet.http.HttpSession.invalidate method; quite often, programmatic invalidation is part of an application logout function.
  - WebSphere Application Server destroys the allocated HttpSession when it expires (default = 1800 seconds or 30 minutes). The WebSphere Application Server can only maintain a certain number of HTTP sessions in memory based on session management settings. In case of distributed sessions, when maximum cache limit is reached in memory, the session management facility removes the least recently used (LRU) one from cache to make room for a session.

.

- Avoid trying to save and reuse the HttpSession object outside of each servlet or JSP file.

  The HttpSession object is a function of the HttpRequest (you can get it only through the req.getSession method), and a copy of it is valid only for the life of the service method of the servlet or JSP file. You *cannot* cache the HttpSession object and refer to it outside the scope of a servlet or JSP file.
- Implement the java.io.Serializable interface when developing new objects to be stored in the HTTP session.

  Serializability of a class is enabled by the class implementing the java.io.Serializable interface. Implementing the java.io.Serializable interface allows the object to properly serialize when using distributed sessions. Classes that do not implement this interface will not have their states serialized or deserialized. Therefore, if a class does not implement the Serializable interface, the JVM cannot persist its state into a database or into another JVM. All subtypes of a serializable class are serializable. An example of this follows:

  ```
  public class MyObject implements java.io.Serializable {...}
  ```

  Make sure all instance variable objects that are not marked transient are serializable. You cannot cache a non-serializable object.

  In compliance with the Java Servlet specification, the distributed servlet container must create an IllegalArgumentException for objects when the container cannot support the mechanism necessary for migration of the session storing them. An exception is created only when you have selected distributable.
- The HTTPSession API does not dictate transactional behavior for sessions.

  Distributed HTTPSession support does not guarantee transactional integrity of an attribute in a failover scenario or when session affinity is broken. Use transactional aware resources like enterprise Java beans to guarantee the transaction integrity required by your application.
- Ensure the Java objects you add to a session are in the correct class path.

  If you add Java objects to a session, place the class files for those objects in the correct class path (the application class path if utilizing sharing across web modules in an enterprise application, or the web module class path if using the Servlet 2.2-complaint session sharing) or in the directory containing other servlets used in WebSphere Application Server.

  Because the HttpSession object is shared among servlets that the user might access, consider adopting a site-wide naming convention to avoid conflicts.
- Avoid storing large object graphs in the HttpSession object.

  In most applications each servlet only requires a fraction of the total session data. However, by storing the data in the HttpSession object as one large object, an application forces WebSphere Application Server to process all of it each time.
- Utilize Session Affinity to help achieve higher cache hits in the WebSphere Application Server.

  WebSphere Application Server has functionality in the HTTP Server plug-in to help with session affinity. The plug-in reads the cookie data (or encoded URL) from the browser and helps direct the request to the appropriate application or clone based on the assigned session key. This functionality increases use of the in-memory cache and reduces hits to the database or another WebSphere Application Server instance
- Maximize use of session affinity and avoid breaking affinity.

  Using session affinity properly can enhance the performance of the WebSphere Application Server. Session affinity in the WebSphere Application Server environment is a way to maximize the in-memory cache of session objects and reduce the amount of reads to the database or another WebSphere

Application Server instance. Session affinity works by caching the session objects in the server instance of the application with which a user is interacting. If the application is deployed in multiple servers of a server group, the application can direct the user to any one of the servers. If the users starts on server1 and then comes in on server2 a little later, the server must write all of the session information to the external location so that the server instance in which server2 is running can read the database. You can avoid this database read using session affinity. With session affinity, the user starts on server1 for the first request; then for every successive request, the user is directed back to server1. Server1 has to look only at the cache to get the session information; server1 never has to make a call to the session database to get the information.

You can improve performance by not breaking session affinity. Some suggestions to help avoid breaking session affinity are:

- Combine all web applications into a single application server instance, if possible, and use modeling or cloning to provide failover support.
- Create the session for the frame page, but do not create sessions for the pages within the frame when using multi-frame JSP files. (See discussion later in this topic.)

- When using multi-framed pages, follow these guidelines:
  - Create a session in only one frame or before accessing any frame sets. For example, assuming there is no session already associated with the browser and a user accesses a multi-framed JSP file, the browser issues concurrent requests for the JSP files. Because the requests are not part of any session, the JSP files end up creating multiple sessions and all of the cookies are sent back to the browser. The browser honors only the last cookie that arrives. Therefore, only the client can retrieve the session associated with the last cookie. Creating a session before accessing multi-framed pages that utilize JSP files is recommended.
  - By default, JSP files get a HTTPSession using `request.getSession(true)` method. So by default JSP files create a new session if none exists for the client. Each JSP page in the browser is requesting a new session, but only one session is used per browser instance. A developer can use `<% @ page session="false" %>` to turn off the automatic session creation from the JSP files that do not access the session. Then if the page needs access to the session information, the developer can use `<%HttpSession session = javax.servlet.http.HttpServletRequest.getSession(false); %>` to get the already existing session that was created by the original session creating JSP file. This action helps prevent breaking session affinity on the initial loading of the frame pages.
  - Update session data using only one frame. When using framesets, requests come into the HTTP server concurrently. Modifying session data within only one frame so that session changes are not overwritten by session changes in concurrent frameset is recommended.
  - Avoid using multi-framed JSP files where the frames point to different web applications. This action results in losing the session created by another web application because the JSESSIONID cookie from the first web application gets overwritten by the JSESSIONID created by the second web application.
- Secure all of the pages (not just some) when applying security to servlets or JSP files that use sessions with security integration enabled, .

  When it comes to security and sessions, it is all or nothing. It does not make sense to protect access to session state only part of the time. When security integration is enabled in the session management facility, all resources from which a session is created or accessed must be either secured or unsecured. You cannot mix secured and unsecured resources.

  The problem with securing only a couple of pages is that sessions created in secured pages are created under the identity of the authenticated user. Only the same user can access sessions in other secured pages. To protect these sessions from use by unauthorized users, you cannot access these sessions from an unsecured page. When a request from an unsecured page occurs, access is denied and an `UnauthorizedSessionRequestException` error is created. (`UnauthorizedSessionRequestException` is a runtime exception; it is logged for you.)
- Use manual update and either the sync() method or time-based write in applications that read session data, and update infrequently.

  With END_OF_SERVICE as write frequency, when an application uses sessions and anytime data is read from or written to that session, the LastAccess time field updates. If database sessions are used, a

new write to the database is produced. This activity is a performance hit that you can avoid using the Manual Update option and having the record written back to the database only when data values update, not on every read or write of the record.

To use manual update, turn it on in the session management service. (See the previous tables for location information.) Additionally, the application code must use the `com.ibm.websphere.servlet.session.IBMSession` class instead of the generic HttpSession. Within the IBMSession object there is a sync method. This method tells the WebSphere Application Server to write the data in the session object to the database. This activity helps the developer to improve overall performance by having the session information persist only when necessary.

Note: An alternative to using the manual updates is to utilize the timed updates to persist data at different time intervals. This action provides similar results as the manual update scheme.

- Implement the following suggestions to achieve high performance:
  - If your applications do not change the session data frequently, use Manual Update and the sync function (or timed interval update) to efficiently persist session information.
  - Keep the amount of data stored in the session as small as possible. With the ease of using sessions to hold data, sometimes too much data is stored in the session objects. Determine a proper balance of data storage and performance to effectively use sessions.
  - If using database sessions, use a dedicated database for the session database. Avoid using the application database. This helps to avoid contention for JDBC connections and allows for better database performance.
  - Verify that you have the latest fix packs for the WebSphere Application Server.
- Utilize the following tools to help monitor session performance.
  - Run the com.ibm.servlet.personalization.sessiontracking.IBMTrackerDebug servlet. - To run this servlet, you must have the servlet invoker running in the web application you want to run this from. Or, you can explicitly configure this servlet in the application you want to run.
  - Use the WebSphere Application Server Resource Analyzer which comes with WebSphere Application Server to monitor active sessions and statistics for the WebSphere Application Server environment.
  - Use database tracking tools such as "Monitoring" in DB2. (See the respective documentation for the database system used.)

Chapter 11. Tuning web services

This page provides a starting point for finding information about web services.

Web services are self-contained, modular applications that can be described, published, located, and invoked over a network. They implement a services oriented architecture (SOA), which supports the connecting or sharing of resources and data in a very flexible and standardized manner. Services are described and organized to support their dynamic, automated discovery and reuse.

Tuning Web Services Security

When using Web Services Security for message-level protection of SOAP message in WebSphere Application Server, the choice of configuration options can affect the performance of the application.

Tuning Web Services Security for Version 8.5 applications

The Java Cryptography Extension (JCE) is integrated into the software development kit (SDK) Version 1.4.*x* and later. This is no longer an optional package. However, the default JCE jurisdiction policy file shipped with the SDK enables you to use cryptography to enforce this default policy. In addition, you can modify the web services security configuration options to achieve the best performance for web services security protected applications.

About this task

Using the unrestricted JCE policy files

Due to export and import regulations, the default JCE jurisdiction policy file shipped with the SDK enables you to use strong, but limited, cryptography only. To enforce this default policy, WebSphere Application Server uses a JCE jurisdiction policy file that might introduce a performance impact. The default JCE jurisdiction policy might have a performance impact on the cryptographic functions that are supported by Web Services Security. If you have web services applications that use transport level security for XML encryption or digital signatures, you might encounter performance degradation over previous releases of WebSphere Application Server. However, IBM and Oracle Corporation provide versions of these jurisdiction policy files that do not have restrictions on cryptographic strengths. If you are permitted by your governmental import and export regulations, download one of these jurisdiction policy files. After downloading one of these files, the performance of JCE and Web Services Security might improve.

Attention: Fix packs that include updates to the Software Development Kit (SDK) might overwrite unrestricted policy files. Back up unrestricted policy files before you apply a fix pack and reapply these files after the fix pack is applied.

Important: Your country of origin might have restrictions on the import, possession, use, or re-export to another country, of encryption software. Before downloading or using the unrestricted policy files, you must check the laws of your country, its regulations, and its policies concerning the import, possession, use, and re-export of encryption software, to determine if it is permitted.

For WebSphere Application Server platforms using IBM Developer Kit, Java Technology Edition Version 6, you can obtain unlimited jurisdiction policy files by completing the following steps:

1. Go to the following website: http://www.ibm.com/developerworks/java/jdk/security/index.html
2. Click **Java SE 6**
3. Scroll down and click **IBM SDK Policy files**.

   The Unrestricted JCE Policy files for the SDK website is displayed.
4. Click **Sign in** and provide your IBM intranet ID and password or register with IBM to download the files.

5. Select the appropriate Unrestricted JCE Policy files and then click **Continue**.
6. View the license agreement and then click **I Agree**.
7. Click **Download Now**.

Results

After following these steps, two Java Archive (JAR) files are placed in the JVM `jre/lib/security/` directory.

Using configuration options to tune WebSphere Application Server

When using WS-Security for message-level protection of SOAP message in WebSphere Application Server, the choice of configuration options can affect the performance of the application. The following guidelines will help you achieve the best performance for your WS-Security protected applications.

1. Use WS-SecureConversation when appropriate for JAX-WS applications. The use of symmetric keys with a Secure Conversation typically performs better than asymmetric keys used with X.509.

   Note: The use of WS-SecureConversation is supported for JAX-WS applications only, not JAX-RPC applications.
2. Use the standard token types provided by WebSphere Application Server. Use of custom tokens is supported, but higher performance is achieved with the use of the provided token types.
3. For signatures, use only the exclusive canonicalization transform algorithm. See the W3 Recommendation web page (http://www.w3.org/2001/10/xml-exc-c14n#) for more information.
4. Whenever possible, avoid the use of the XPath expression to select which SOAP message parts to protect. The WS-Security policies shipped with WebSphere Application Server for JAX-WS applications use XPath expressions to specify the protection of some elements in the security header, such as Timestamp, SignatureConfirmation, and UsernameToken. The use of these XPath expressions is optimized, but other uses are not.
5. Although there are Websphere Application Server extensions to WS-Security that can be used to insert nonce and timestamp elements into SOAP message parts before signing or encrypting the message parts, you should avoid the use of these extensions for improved performance.
6. There is an option to send the base-64 encoded CipherValue of WS-Security encrypted elements as MTOM attachments. For small encrypted elements, the best performance is achieved by avoiding this option. For larger encrypted elements, the best performance is achieved by using this option.
7. When signing and encrypting elements in the SOAP message, specify the order as sign first, then encrypt.
8. When adding a timestamp element to a message, the timestamp should be added to the security header before the signature element. This is accomplished by using the `Strict` or `LaxTimestampFirst` security header layout option in the WS-Security policy configuration.
9. For JAX-WS applications, use the policy-based configuration rather than WSS API-based configuration.

What to do next

In IBM WebSphere Application Server Version 6.1 and later, Web Services Security supports the use of cryptographic hardware devices. There are two ways in which to use hardware cryptographic devices with Web Services Security. See Hardware cryptographic device support for Web Services Security for more information.

Tuning Web Services Security for Version 5.x applications

The Java Cryptography Extension (JCE) policy is integrated into the IBM Software Development Kit (SDK) Version 1.4.*x* and is no longer an optional package. However, due to export and import regulations, the default JCE jurisdiction policy file shipped with the SDK enables you to use strong, but limited, cryptography only.

About this task

To enforce this default policy, WebSphere Application Server uses a JCE jurisdiction policy file that might introduce a performance impact. The default JCE jurisdiction policy might have a performance impact on the cryptographic functions that are supported by Web Services Security. If you have web services applications that use transport level security for XML encryption or digital signatures, you might encounter performance degradation over previous releases of WebSphere Application Server. However, IBM and Sun Microsystems provide versions of these jurisdiction policy files that do not have restrictions on cryptographic strengths. If you are permitted by your governmental import and export regulations, download one of these jurisdiction policy files. After downloading one of these files, the performance of JCE and Web Services Security might improve.

Procedure

1. AIX Linux Windows For WebSphere Application Server platforms using IBM Developer Kit, Java Technology Edition Version 1.4.2, including the AIX, Linux, and Windows platforms, you can obtain unlimited jurisdiction policy files by completing the following steps:
   a. Go to the following website: http://www.ibm.com/developerworks/java/jdk/security/index.html.
   b. Click **Java 1.4.2**.
   c. Click **IBM SDK Policy files**. The Unrestricted JCE Policy files for SDK 1.4 website is displayed.
   d. Enter your user ID and password or register with IBM to download the policy files. The policy files are downloaded onto your machine.
2. Solaris HP-UX For WebSphere Application Server platforms using the Sun-based Java SE Development Kit 6 (JDK 6) Version 1.4.2, including the Solaris environments and the HP-UX platform, you can obtain unlimited jurisdiction policy files by completing the following steps:
   a. Go to the following website: http://java.sun.com/j2se/1.4.2/download.html.
   b. Click **Other Downloads**.
   c. Locate the JCE Unlimited Strength Jurisdiction Policy Files 1.4.2 information and click **Download**. The policy files are downloaded onto your machine.

Results

After following either of these sets of steps, two Java Archive (JAR) files are placed in the JVM directory.

```
jre/lib/security/
C:\Program Files\ibm\jre\lib\security
```

Tuning web services reliable messaging applications

Modifying certain settings, such as heap size, can help to improve the performance of your system.

Procedure

1. Tune the heap size of the JVM. Increasing the heap size, for example to 1536, can avoid out-of-memory errors, especially in the following situations:
   - When you are composing WS-ReliableMessaging with WS-Notification.
   - When you are using a Reliable Asynchronous Message Profile (WS-I RSP) policy set.
   - When you are using WS-ReliableMessaging in situations where a lot of recovery is required.

   For more information about tuning the JVM, refer to Tuning the IBM virtual machine for Java.
2. If your application uses a managed quality of service, tune the sib.wsrm.tokenLockTimeout property of the messaging engine that is specified in the policy binding for the application. See the service integration custom properties information to learn more about this property, and why you might want to change its value.

Tuning bus-enabled web services

You can use the administrative console or a Jacl script to tune performance settings for service integration bus-enabled web services.

About this task

Bus-enabled web services dynamically use a fast-path route through the bus where possible. This fast-path route is used if the following criteria are met:

- The inbound port and outbound port for the service are on the same server.
- There are no mediations on the path from the inbound port to the outbound port.

Further optimizations can be made, if your configuration also meets the following criteria:

- The inbound template WSDL URI is the same location as the Outbound Target Service WSDL location URI.
- The inbound service template WSDL service name matches the outbound WSDL service name.
- The inbound service template port name matches the outbound WSDL port name.
- The mapping of the namespaces is disabled (that is, you have set the inbound service property **com.ibm.websphere.wsgw.mapSoapBodyNamespace** to `false`).
- Operation-level security is not enabled on the outbound service.

If your web services use the fast-path route, you need not tune mediations or the service integration bus. However it is good practise to do so, because a typical environment will have at least one non-fast-path (for example, mediated) service.

To improve the performance of bus-enabled web services you can tune the following parameters:

- The Java virtual machine heap size. This helps ensure there is enough memory available to process large messages, or messages with large attachments.
- The maximum number of instances of a message-driven bean that are permitted by the activation specification for the service integration technologies resource adapter. This throttles the number of concurrent clients serviced.
- The maximum batch size for batches of messages to be delivered to a client. By default, only a single message is delivered to a message-driven bean instance at one time; you can improve performance by allowing messages to be sent in batches to a message-driven bean.
- The number of threads available to service requests for each client. That is, the number of threads available in the default thread pool, the web container thread pool and the mediation thread pool for a given application server.
- The number of threads available in the mediation thread pool. This assumes that your mediations use concurrent support where appropriate, as explained in Concurrent mediations.

If you have mediations that act on SOAP headers, you can improve performance by inserting the associated header schemas (`.xsd` files) into the SDO repository.

Procedure

- Optional: Use the administrative console to tune bus-enabled web services by completing the following steps:
  1. Use the topic Tuning the IBM virtual machine for Java to set the JVM heap size to a larger value than the default value (256 megabytes). The value should generally be as large as possible without incurring paging.
  2. Use the topic Tuning service integration messaging to tune the maximum number of instances of a message-driven bean, the maximum batch size for batches of messages for a bean, and the number of threads available to service requests for a bean.

3. Use the topic Tuning the application serving environment to tune the general application serving environment, in particular the size of the web container thread pool. In a server that is exclusively serving requests to bus-enabled web services, the default thread pool and the web container thread pool should be the same size.
4. Use the topic Configuring the mediation thread pool to configure the number of threads available to concurrent mediations.

- Optional: If you have mediations that act on SOAP headers, insert the associated schemas (`.xsd` files) into the SDO repository as described in "Including SOAP header schemas in the SDO repository."

Including SOAP header schemas in the SDO repository

Use this task to improve mediation performance by inserting the SOAP header schema into the SDO repository.

About this task

Mediations accessing SOAP headers should ensure that the SOAP header schema is made available to the SDO repository. This simplifies access to the header fields (see Web Services code example) and can provide a significant performance benefit. Usually the schema (`.xsd` file) for a SOAP header is already available to the application developer.

Here is an example of a header (used for routing) that is passed in the SOAP message:

```
<soapenv:Header>
<hns0:myClientToken xmlns:hns0="http://www.ibm.com/wbc">
           <UseRoutingId>true</ UseRoutingId >
           <RoutingID>5</ RoutingID >
       </hns0: myClientToken >
</soapenv:Header>
```

Here is an example of an associated header schema:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.ibm.com/wbc"
           elementFormDefault="unqualified">
<xs:element name=" myClientToken">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="UseRoutingId" type="xs:string"/>
      <xs:element name="RoutingID" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

To insert the schema into the SDO repository, complete the following steps:

Procedure

1. Create a script that contains the following code:
   - For Jython, create a script called `sdoXSDImport.py`:

     ```
     #
     xsdFile=sys.argv[0]
     xsdKey=sys.argv[1]
     sdoRep=AdminControl.queryNames("*,type=SdoRepository,node=%s" % AdminControl.
     getNode)
     print AdminControl.invoke(sdoRep , importResource([xsdKey , xsdFile]))
     ```

   - For Jacl, create a script called `sdoXSDImport.jacl`:

```
#
set xsdFile [lindex $argv 0]
set xsdKey  [lindex $argv 1]
set sdoRep [$AdminControl queryNames *,type=SdoRepository,node=[$AdminControl
 getNode]]
puts [$AdminControl invoke $sdoRep importResource [list $xsdKey $xsdFile]]
```

Note: To create an equivalent script for removing a resource from the SDO repository, take a copy of this script and modify the final line as follows:

- Using Jython:

  ```
  AdminControl.invoke(sdoRep , "removeResource" , [[xsdKey , "false"]])
  ```

- Using Jacl:

  ```
  $AdminControl invoke $sdoRep removeResource [list $xsdKey false]
  ```

2. Use the wsadmin scripting client to insert the schema into the SDO repository by entering the following command.

   - To use the Jython script:

     ```
     wsadmin -lang jython -f sdoXSDImport.py your_header.xsd your_header_namespace
     ```

   - To use the Jacl script:

     ```
     wsadmin -f sdoXSDImport.jacl your_header.xsd your_header_namespace
     ```

   where

   - *your_header*.xsd is the name of the file that contains your header schema.
   - *your_header_namespace* is the target namespace for the header. For example http://yourCompany.com/yourNamespace.

Chapter 12. Tuning Work area

This page provides a starting point for finding information about work areas, a WebSphere extension for improving developer productivity.

Work areas provide a capability much like that of global variables. They enable efficient sharing of information across a distributed application.

For example, you might want to add profile information as each customer enters your application. By placing this information in a work area, it is available throughout your application, eliminating the need to hand-code a solution or to read and write information to a database.

Work area service performance considerations

The work area service is designed to address complex data passing patterns that can quickly grow beyond convenient maintenance. A work area is a note pad that is accessible to any client that is capable of looking up Java Naming Directory Interface (JNDI). After a work area is established, data can be placed there for future use in any subsequent method calls to both remote and local resources.

You can utilize a work area when a large number of methods require common information or if information is only needed by a method that is significantly further down the call graph. The former avoids the need for complex parameter passing models where the number of arguments passed becomes excessive and hard to maintain. You can improve application function by placing the information in a work area and subsequently accessing it independently in each method, eliminating the need to pass these parameters from method to method. The latter case also avoids unnecessary parameter passing and helps to improve performance by reducing the cost of marshalling and de-marshalling these parameters over the Object Request Broker (ORB) when they are only needed occasionally throughout the call graph.

When attempting to maximize performance by using a work area, cache the UserWorkArea partition that is retrieved from JNDI wherever it is accessed. You can reduce the time spent looking up information in JNDI by retrieving it once and keeping a reference for the future. JNDI lookup takes time and can be costly.

Additional caching mechanisms available to a user-defined partition are defined by the configuration property, “Deferred Attribute Serialization”. This mechanism attempts to minimize the number of serialization and deserialization calls. Refer to the Work area partition service article for further explanation of this configuration attribute.

The maxSendSize and maxReceiveSize configuration parameters can affect the performance of the work area. Setting these two values to 0 (zero) effectively turns off the policing of the size of context that can be sent in a work area. This action can enhance performance, depending on the number of nested work areas an application uses. In applications that use only one work area, the performance enhancement might be negligible. In applications that have a large number of nested work areas, there might be a performance enhancement. However, a user must note that by turning off this policing it is possible that an extremely large amount of data might be sent to a server.

Performance is degraded if you use a work area as a direct replacement to passing a single parameter over a single method call. The reason is that you incur more overhead than just passing that parameter between method calls. Although the degradation is usually within acceptable tolerances and scales similarly to passing parameters with regard to object size, consider degradation a potential problem before utilizing the service. As with most functional services, intelligent use of the work areas yields the best results.

The work area service is a tool to simplify the job of passing information from resource to resource, and in some cases can improve performance by reducing the overhead that is associated with a parameter passing when the information is only sparsely accessed within the call graph. Caching the instance

retrieved from JNDI is important to effectively maximize performance during runtime.

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

APACHE INFORMATION. This information may include all or portions of information which IBM obtained under the terms and conditions of the Apache License Version 2.0, January 2004. The information may also consist of voluntary contributions made by many individuals to the Apache Software Foundation. For more information on the Apache Software Foundation, please see http://www.apache.org. You may obtain a copy of the Apache License at http://www.apache.org/licenses/LICENSE-2.0.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Intellectual Property & Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

Trademarks and service marks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. For a current list of IBM trademarks, visit the IBM Copyright and trademark information Web site (www.ibm.com/legal/copytrade.shtml).

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other company, product, or service names may be trademarks or service marks of others.

Index