Connection pool tuning and management problems

WebSphere software

© 2011 IBM Corporation

This unit describes how to troubleshoot connection pool tuning and management problems in WebSphere Application Server V7.

Unit objectives

- After completing this unit, you should be able to:
- Identify connection pool problems
- Describe what to look for in the application server logs
- Enable tracing for connection manager components
- Interpret and analyze the trace data
- Explain how to use PMI to monitor connections

Connection pool tuning and management problems   © 2011 IBM Corporation

After you complete this unit, you are able to identify problems in connection pools, know how to use Tivoli Performance Viewer, to monitor a connection pool, understand connection pool tracing data, and perform the problem determination tasks to troubleshoot a connection pool problem.

What is connection pooling?

- The application server maintains a pool of ready-to-use connections to a data store.
    - Application client requests a connection using a data source or connection factory object
    - Connection is retrieved from pool
- Benefits:
    - Minimizes database session setup and tear-down
    - Improves application database access performance
    - Spreads out connection cost over repeated uses
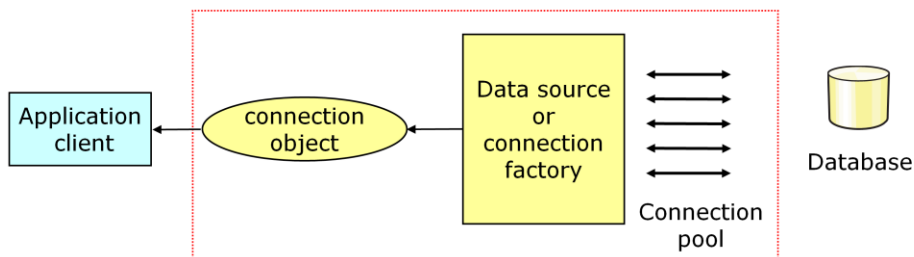
Connection pool tuning and management problems
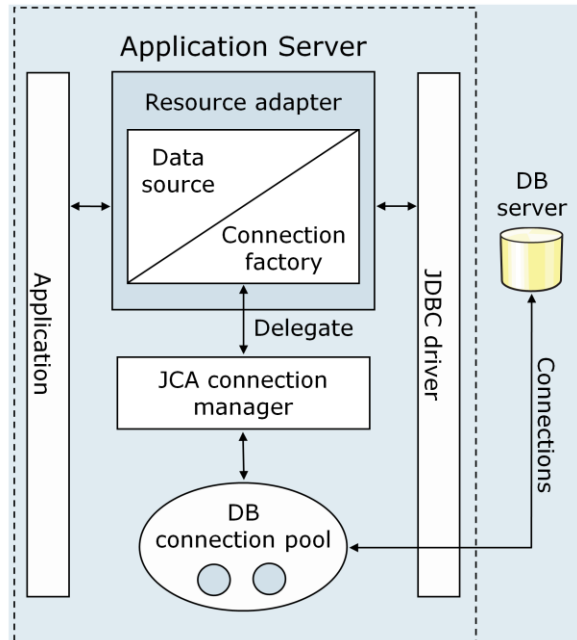
© 2011 IBM Corporation

Applications need to acquire a connection to a data store each time it wants to retrieve information from the store. The average connection object is one to two megabytes in size and contains a great deal of information about the connection context.

Creating and terminating those connections is actually a very time consuming operation and so it can easily slow down the application. To fix this problem, WebSphere Application Server uses a pool of connections that can be reused by applications.

This allows the cost of establishing each connection to be spread out across several requests and can significantly improve performance. An application that needs to access the data store will request a connection from the pool and return the connection when it is finished. An example of this work flow is illustrated on the slide.

JCA connection pooling architecture

- Connection pooling is supported by two components:
  - JCA connection manager
  - Relational resource adapter

- How it works:
  - An application acquires a data source or connection factory object from the relational resource adapter.
  - The data source or connection factory delegates the connection allocation request to the JCA connection manager.
  - The JCA connection manager retrieves a free connection from the pool or creates a new one if none is available.
  - The retrieved or created connection is returned to the application

Connection pool tuning and management problems

© 2011 IBM Corporation

WebSphere Application Sever implements connection pooling by following the Java EE Connection Architecture version 1.5 specifications.

There are several objects involved in pooling connections but they can be grouped into two basic components, the JCA connection manager and the Relational Resource Adapter.

An application that needs a database connection will go to the Resource Adapter to retrieve a Connection Factory. The Connection Factory will delegate the request to the correct JCA Connection Manager. The connection manager is responsible for either returning an existing connection from the pool of available connections or creating a new one if none are available. The application releases the connection when it is finished interacting with the database and the DB connection pool will return it to the pool.

Types of connection pools in WebSphere

- JDBC provider connection pool
    - Enables access to a relational database
    - Provides a connection using a JDBC data source
    - Managed in the administrative console under JDBC providers
- JMS provider connection pool
    - Enables access to the data store used by the default messaging engine or a WebSphere MQ provider
    - Provides a connection using a JMS connection factory
    - Managed in the administrative console under JMS providers
- EIS connection pool
    - Enables access to enterprise information systems such as CICS, legacy databases such as IMS, and other back-end systems
    - Provides a connection using a data source or connection factory
    - Managed in the administrative console under Resource Adapters
- All are managed by the WebSphere **J2C connection pool manager** (JCA connection manager)

Connection pool tuning and management problems

© 2011 IBM Corporation

WebSphere Application Server uses a J2C connection pool manager to maintain three different connection pools.

The JDBC connection pool is used to manage connections to relational databases such as DB2. This pool can be adjusted by navigating to the JDBC Providers panel of the WebSphere administrative console.

There is also a JMS pool for managing requests for connections to Service Integration Bus messaging engines or WebSphere MQ Queue Managers. This pool is maintained in the JMS Providers panel of the administrative console.

Finally, WebSphere provides an EIS connection pool that manages connections to CICS and legacy back-end systems such as IMS. This connection pool is controlled through the administrative console under the Resource Adapters panel.

Connection life cycle

- A ManagedConnection object is always in one of three states: **DoesNotExist**, **InFreePool**, or **InUse**.
- After a connection is created, it can be in either
  - InUse (if allocated to an application)
  - InFreePool state
- Between these three states are **transitions**.
  - For example, you can make the transition from the InFreePool state to InUse state if:
    - Application has called the data source or connection factory `getConnection()` method
    - A free connection is available in the pool with matching properties
- After a connection is closed with the `close()` method, it is either:
  - Returned to the free pool
  - Destroyed

Connection pool tuning and management problems

© 2011 IBM Corporation

A ManagedConnection object is always in one of three states: DoesNotExist, InFreePool, or InUse.

Before a connection is created, it must be in the DoesNotExist state. After a connection is created, it can be in either the InUse or the InFreePool state, depending on whether it is allocated to an application.

Between these three states are transitions. These transitions are controlled by guarding conditions. A guarding condition is one in which true indicates when you can take the transition into another legal state.

For example, you can make the transition from the InFreePool state to InUse state only if:

  - the application has called the data source or connection factory getConnection() method (the getConnection condition)

  - a free connection is available in the pool with matching properties (the freeConnectionAvailable condition)

Applications should explicitly close connections by calling close() on the connection object. After a connection is closed with the close() method, it is either returned to the free pool or destroyed.

Shareable versus unshareable connections (1 of 2)

- Shared connections (default)
  - Single connection used by multiple `getConnection()` calls within the same transactional context or container boundary
    - The connection is not released back to the pool even when closed
    - Connection is released when transaction or parent method completes
  - Default behavior — often more scalable

- Common issues
  - Shared connections can be held too long, exhausting the pool, showing symptoms of a connection leak

Java Connector Architecture (JCA) manages the connection pooling and it enables the use of shareable or unshareable connections.

In WebSphere Application Server, connections are shareable by default. The use of shareable connections allows for greater scalability. Instead of creating new physical connections on every getConnection() invocation, the physical connection (that is, managed connection) is shared through multiple connection handles, as long as each getConnection request has the same connection properties

When the application closes a shareable connection it is not returned to the free pool. Rather, it remains ready for another request within the same transaction for a connection to the same resource.

Shareable connections obtained by an application within a local transaction containment are kept reserved in the shared pool for use within that local transaction containment until the it ends, even if the application explicitly closes the connection.

The benefits of this are improved performance and a reduction in the number of physical connections that need to be managed.

This behavior is beneficial to many applications, but can have unintended consequences for others.

- Unshared connections:
  - Each `getConnection()` call results in a connection being allocated from the connection pool.
  - The connection is released back to the pool upon a `close()` call.
  - Many different connections can be used within the same transaction.

- Common issues:
  - Unshared connections might exhaust the pool as many may be used within a transaction
  - Wasted connections

An unshareable connection is not shared with other components in the application. Each getConnection() call results in a connection being allocated from the connection pool, so your application always maintains a direct link with a physical connection (managed connection).

The connection always has a one-to-one relationship between the connection handle and the managed connection.

In most cases, the connection does not close until the application closes it, and you can use a cached unshared connection handle across multiple transactions.

Some possible drawbacks of using unshared connections :

Inefficient use of your connection resources. For example, if within a single transaction you get more than one connection (with the same properties) using the same data source or connection factory (same resource-ref) then you use multiple physical connections when you use unshareable connections, which can cause you to exhaust the connection pool.

Wasted connections. It is important not to keep the connection handle open (that is, your application does not call the *close()* method) any longer then it is needed. As long as an unshareable connection is open, the physical connection is unavailable to any other component, even if your application is not currently using that connection. Unlike a shareable connection, an ushareable connection is not closed at the end of a transaction or servlet call.

Detecting connection management-related problems

• Look in the WebSphere SystemOut.log and SystemErr.log files for these types of messages:

Message prefix or type	Message source
DSRA or CWWRA	§JCA resource adapter
CONM or CWWCM	§WebSphere Version 4 connection manager
	§Legacy connection manager used to support J2EE 1.2 applications
J2CA or CWWJC	§Java EE connector (J2C Connection pool manager)
	§Most recent JCA 1.5 compliant connection manager
WSCL or CWWSC	§WebSphere client (Java EE application client manager)
WTRN or CWWTR	§WebSphere transaction manager
SQLException or database error code	§Database manager

Connection pool tuning and management problems

© 2011 IBM Corporation

If you are experiencing connection related problems, have a look at the SystemOut and SystemErr logs in the appropriates profile's log directory.

WebSphere Application Server maintains connection pools for multiple connection types, so in the logs you will see various message prefixes are listed on this slide along with the connection types.

The best way to determine if you are experiencing connection problems is to search the logs for any of the message prefixes then correlate them to the appropriate message source.

Typical connection pool runtime problem symptoms

- Sporadic failure to connect to an existing data source or connection factory:
  - The application has been working and connecting to an existing data source or connection factory.
  - You start to experience poor user response time and see intermittent failures in getting a connection.
- Next, look at the WebSphere logs to see if it additionally shows:
  - No specific exception
    - Probable cause: improperly tuned connection pool settings
  - ConnectionWaitTimeoutException
    - Probable cause 1: improperly tuned connection pool settings
    - Probable cause 2: connection leak due to poor coding
  - StaleConnectionException
    - Probable cause: stale connection

Connection pool tuning and management problems

© 2011 IBM Corporation

Most people find out they are experiencing a connection pool problem by noticing symptoms in their application's behavior instead of noticing events in the SystemOut and SystemErr logs.

There is typically a problem with a connection pool when an application experiences sporadic failures when trying to connect to a data source. This means the application was able to connect to the data source and work normally but then started to see intermittent failures or a increase in user response time. In either case, the next step is to check the log files to help narrow down the possible cause for the sporadic behavior.

There are several possible outcomes from checking the log files.

One outcome is that you do not find connection exceptions. In this case, the problem is likely due to something other than the connection pool.

Another outcome is you find ConnectionWaitTimeoutException message in the log files then there are two probable causes. You either need to change the connection pool settings or there is a connection leak somewhere in the system.

Finally, if you find StaleConnectionException in the log then, as the exception indicates, there is probably a problem with connections going stale.

Connection pooling problem determination path

Connection pool tuning and management problems

© 2011 IBM Corporation
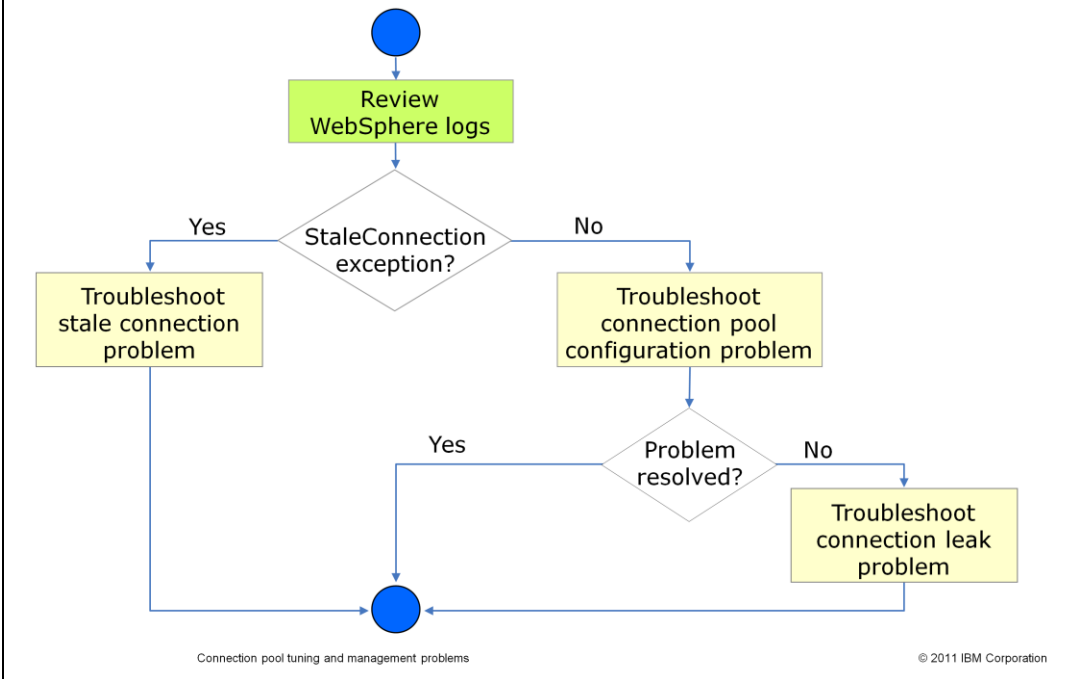
If it helps, you can think of troubleshooting a connection pool problem in terms of a decision tree.

The tree starts with the assumption that you are seeing sporadic behavior from your application or you have another reason to believe there is a problem in the connection pool.

From there, you review the logs and look for Stale Connection Exceptions. If you find any the next step is to begin troubleshooting a stale connection problem. Otherwise, it is best to review the connection pool configuration and make sure it is not causing the problem. If the configuration checks out then you should begin troubleshooting a possible connection leak in the connection pool.

From here, you will take a detailed look at each of the three troubleshooting steps.

Troubleshooting connection pool configuration in the problem determination path

Connection pool tuning and management problems

© 2011 IBM Corporation
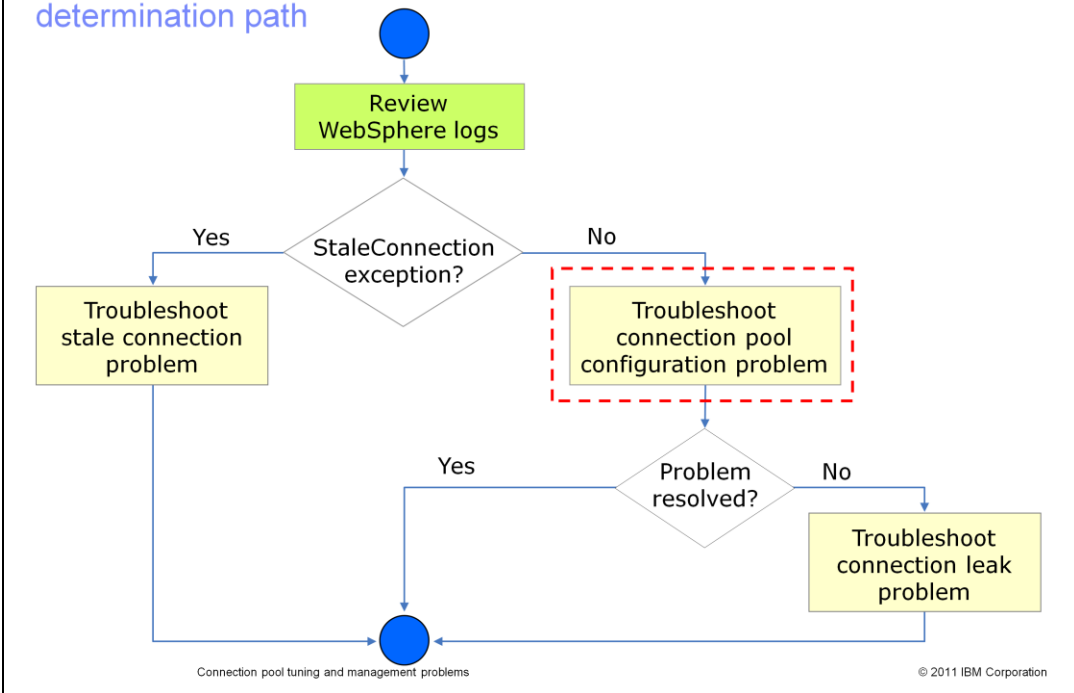
In the case where the symptom for a connection pool problem is not accompanied by a StaleConnectionException in the WebSphere logs, start your problem determination effort by looking at the connection pool configuration to rule out any performance tuning issues.

The need for connection pool tuning

- Keeping connections in a pool consumes resources
  - Connections are large objects (1 – 2 MB each)

- An improperly tuned connection pool can result in:
  - Poor user response if the client is consistently waiting for a free connection
  - Application exceptions if the client cannot get a connection within the specified wait timeout interval
  - Reduced server throughput if unused connections are wasting system resources

- Connection pools need to be properly tuned to ensure optimal performance:
  - Maximize the chances that connections are available when needed
  - Minimize the number of idle connections
  - Minimize the number of orphaned connections

Connection pool tuning and management problems   © 2011 IBM Corporation

Connection pools allow you to set a range for the number of connections that will maintained by WebSphere Application Server. It is important to get the tuning parameters right otherwise you might cause problems for the application.

Setting the pool size too small can slow down the application because it will have constantly wait for free connections. Setting it too large will waste resources and possibly impact the server's throughput.

In general, you want to try and tune a connection pool to achieve three goals.

First, you want to maximize the chance that connections are available when needed. This means setting the connection pool size so that it is big enough to have free connections when they are needed.

Second, you want to minimize the number of idle connections because connections that are not being used are overhead that reduces the server's throughput.

Finally, you want to set the connection timeout so that it minimizes the number of orphaned connections but does not interfere with connections that are operating normally.

Key connection pool parameters

- Maximum connections
  - Specifies the maximum number of connections that can be created in the pool
  - Default value is 10
  - A value of 0 allows the number of physical connections to grow infinitely and causes the connection timeout value to be ignored

- Connection timeout
  - Specifies the interval, in seconds, after which a connection request times out and a ConnectionWaitTimeoutException is thrown.
  - Default value is 180 seconds (three minutes).
  - A value of 0 instructs the pool manager to wait as long as necessary until a connection becomes available.

There are a few of the connection pool parameters that play a significant role in achieving the goal of an optimized connection pool.

The first of these parameters is the maximum connections count. This value governs the maximum size of the connection pool. If the pool has already reached the maximum size it will not allow a new connection to be created and will instead force a request to wait for an existing connection to free up. However, you can set the maximum value to 0 and allow the pool to grow without constraint. This will also cause the Connection Timeout value to be ignored.

The connection timeout is how long a connection request will wait for a free connection before it quits and creates a ConnectionWaitTimeoutException. You can also disable the connection timeout by setting it to 0 and allowing a request to wait as long as it takes to receive a connection. Setting the connection pool to 0 in and allowing unconstrained growth can be a bad idea.

Resources should be load tested in order to find the best value and should always be bounded. Failure to do so can lead to memory exhaustion and worse, overloading the resources that are being connected to.

Connection pool parameters in the administrative console

Data sources

Data sources > **PLANTSDB** > Connection pools

Use this page to set properties that impact the timing of connection management tasks, which can affect the performance of your application. Consider the default values carefully; your application requirements might warrant changing these values.

Configuration

General Properties

Scope
cells:was7host01Cell01:nodes:was7host01Node01:servers:server1

* Connection timeout
180   seconds

* Maximum connections
10   connections

* Minimum connections
1   connections

* Reap time
180   seconds

* Unused timeout
1800   seconds

* Aged timeout
0   seconds

Purge policy
EntirePool
EntirePool
FailingConnectionOnly

Apply   OK   Reset   Cancel

Additional Properties

▪ Advanced connection pool properties

▪ Connection pool custom properties

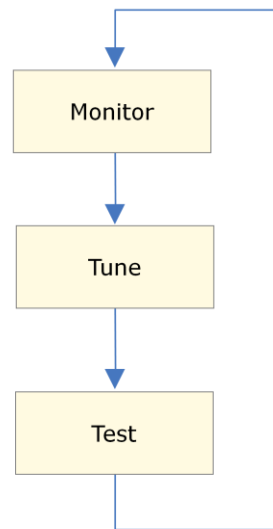Connection pool tuning and management problems   © 2011 IBM Corporation

There are several other connection pool properties that can be configured in the WebSphere AdminConsole.

Minimum Connections, for example, specifies the number of physical connections that should be maintained. Note, this does not mean the connection pool will start with the minimum number of connections but that it will not go beneath that value once it reaches it.

Many of the connection properties interact with each other. For example, the Reap time specifies, in seconds, the interval between runs of the connection pool's maintenance thread. This value will affect the accuracy of both the Unused timeout and the Aged timeout.

Connection pool tuning tasks

- Monitor connection pool runtime behavior
  - Enable PMI and select required statistic set
  - View connection pool performance metrics using Tivoli Performance Viewer
  - Generate tuning advice using TPV Performance Advisor
- Tune connection pool parameters
  - Make one change at a time
  - Apply recommendations and best practices
- Test application
  - Use a load generation tool to simulate production-like loads
  - Compare results with original baseline
  - Document results
- Repeat the monitor-tune-test cycle until the problem is resolved and performance goals are met

Monitor

Tune

Test

Connection pool tuning and management problems   © 2011 IBM Corporation

Performance tuning, in general, is an iterative and incremental process consisting of multiple Monitor-Tune-Test cycles. Having the right tools, including a load generation tool to simulate real-world users for load testing, is critical to ensure successful results.

The art of performance tuning is a mixture of documentation, test data, and experience. There are some tools that can assist with this practice such as the Tivoli Performance Advisor provided in WebSphere Application Server V7 by way of the AdminConsole, but the suggestions/help that it offers, still need to be verified through load testing.

The general method for getting the correct value is to increase the timeout and connection parameters until the timeout issue disappears and then backing them off until any wasted resources are recovered.

Note that the Performance Monitoring Infrastructure (PMI) is enabled by default in WebSphere V7 and can used to measure performance metrics during the testing.

Monitoring the connection pool using Tivoli Performance Viewer

- These key connection pooling performance metrics should be monitored:

Metric name	Description	What to look for
PoolSize	Size of the connection pool	§Increases as new connections are created (up to the value of *maximum connections*) and decreases when connections are destroyed §A significant number of creates and destroys is an indication that the pool size (*maximum connections*) should be adjusted §Counter is already enabled as part of the *basic* (default) PMI statistic set
PercentUsed	Average percent of the pool that is in use	§If consistently low, you can want to decrease the pool size §Counter is already enabled as part of the *basic* (default) PMI statistic set
WaitingThreadCount	Average number of threads that are concurrently waiting for a connection	§The optimal value for the pool size is that which reduces this value §Counter is already enabled as part of the *basic* (default) PMI statistic set
PercentMaxed	Average percent of the time that all connections are in use	§Ensure that you are not consistently maxed a 100% §Counter requires the selection of either *all* or *custom* PMI statistic set

Connection pool tuning and management problems © 2011 IBM Corporation

Here are some of the key metrics that WebSphere Application Server monitors in the PMI. These metrics are displayed in TPV under the JDBC Connection Pools and JCA Connection Pools modules.

Monitoring the PoolSize metric will allow you to examine how the application uses connections from the pool and note trends and behaviors.

The WaitingThreadCount metric is a good way to determine if your pool is sized too small. This metric tracks the number of threads that are having to wait for a connection to become available before completing their requested operations.

Finally, the PercentMaxed value is a key point for determining if your pool size is too small. It is ok to hover in the high percentages for short periods of time however, a pool that is 100% used for long periods of time is bound to have waiting threads and long response times that will adversely affect the application performance.

Tivoli Performance Viewer connection pool monitoring example
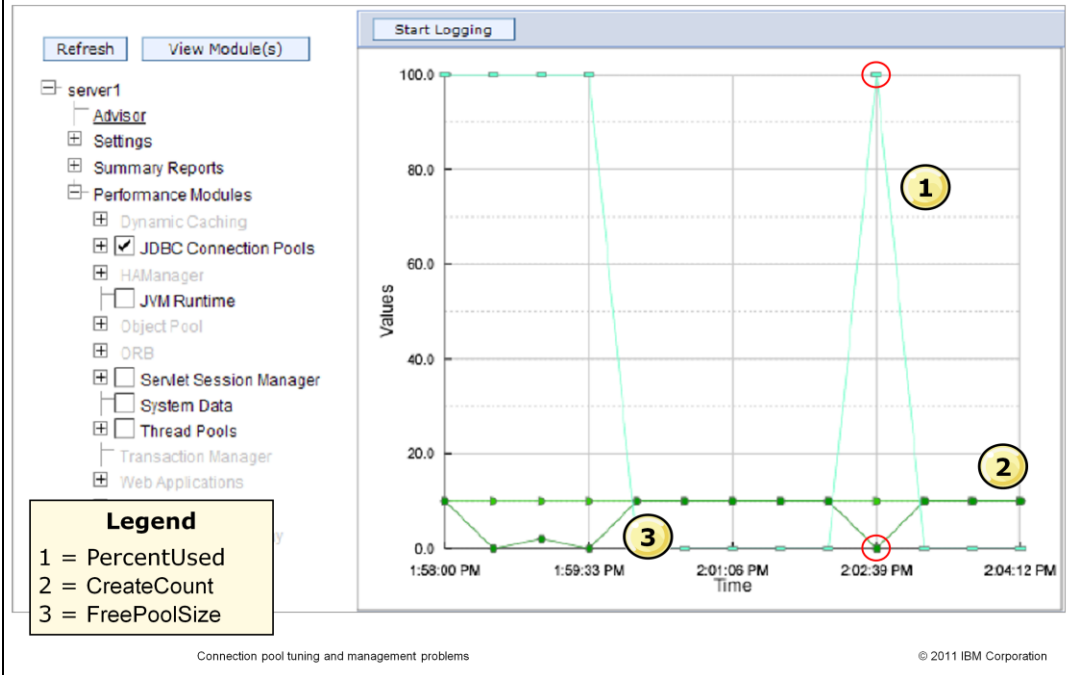
This is a screen capture of the Tivoli Performance Viewer displaying information on the JDBC Connection Pools.

The cyan colored graph plots the PercentUsed metric. The dark green graph plots the FreePoolSize metric. And the green graph plots the CreateCount metric.

In this example, ten connections have been created, reaching the default maximum connections value for the pool. Notice that, as expected, when the FreePoolSize is zero indicating no connection available in the pool, the PercentUsed value is at 100%.

Generating tuning advice using TPV Performance Advisor

- TPV Performance Advisor can provide configuration advice for connection pool size:
  - Advice opens in the Performance Advisor section of Tivoli Performance Viewer
  - Based on collected PMI data over the last one-minute interval
  - Uses IBM-defined recommendations for advice basis

- Limitations:
  - Pool sizing advice cannot be generated if your timeout values are too high (pools are not returning back to minimum values)
  - Tivoli Performance Viewer Advisor only gives recommendations when processor usage is greater than or equal to 50%

TPV Performance Advisor is one of the ways that WebSphere Application Server can provide tuning advice.TPV Performance Advisor runs on demand and outputs recommendations to a graphical interface in the administrative console.

It's recommendations are based on collected PMI data over the last one-minute interval. For example, if it observes that the number of connections is continuously low then it will recommend that you lower the size of the connection pool.

The TPV Performance Advisor can be accessed and configured in the AdminConsole.

Performance Advisor tuning advice example

Runtime Events > **Message Details**

Runtime events propagating from the server

General Properties

Message
> TUNE0206W: Decreasing the connection pool for data source jdbc/TradeDataSource by setting the minimum size to 0 and the maximum size to 3 may improve performance. Additional explanatory data follows. Pool utilization: 1.2%. This alert has been issued 1 time(s) in a row. The threshold will be updated to reduce the overhead of the analysis.

Message type
> Warning

Explanation
> Decreasing the size supports better pooling and frees memory resources.

User action
> From the administrative console, click: Resources > JDBC Providers > JDBC_provider > Data Sources > data_source > Connection pool properties.

Message Originator
> com.ibm.ws.performance.tuning.serverAlert.TraceResponse

Source object type
> RasLoggingService

Timestamp
> Oct 19, 2004 9:28:35 AM EDT

Thread Id
> 15

Node name
> laptop-rzhouNode01

Server name
> server1

Connection pool tuning and management problems   © 2011 IBM Corporation

This slide shows an example of connection pool tuning advice generated by the Runtime Performance Advisor.

From the screen capture you can see a tuning recommendation stating that the minimum size of the TradeData source pool should be reduced to zero and the maximum size of the pool should be three based on the data collected by the advisor component.

The User action explains where to go in the AdminConsole, and the Node and Server name to make the property change on, if you choose to do so.

Note that the threshold for the message adjusts automatically to prevent the system from issuing multiple, redundant messages for the same values.

Tuning the connection pool

- Goal is to create a large enough pool that can handle a peak load but does not unnecessarily take up system resources.
  - Unused connections during non-peak periods can be controlled with the minimum connections parameter.
- In order to successfully tune the connection pool, you need to know two pieces of information:
  - The requests per second that occur during a peak
  - How long the database takes to respond to each type of operation
    - SELECT, INSERT, UPDATE, and so on
- Key parameters to tune:
  - Maximum connections:
    - Optimal value for pool size is that which reduces the value of concurrent waiters for a connection (WaitingThreadCount)
  - Connection timeout:
    - Value should be based on a combination of how long the database operations take to complete and the number of concurrent waiters for a connection

Connection pool tuning and management problems   © 2011 IBM Corporation

When tuning, the correct parameter values can only be discovered through trial and error. In particular, the two parameters that will have the greatest effect on correcting connection pool configuration errors are the connection timeout and maximum connections.

If the time taken to complete a database operation is greater than the amount of time a thread is willing to wait for a resource, then increasing the number of available connections will not solve the problem. Conversely, if the connections are short-lived, then increasing their number can lead to the application server being overloaded in other areas during a peak in requests because the extra connections are unnecessarily consuming resources.

Also, the number of idle connections during off peak periods should be weighed against the pool ramp up time when a peak occurs. By understanding the nature of these parameters and the nature of the database operations that will occur during a peak load, an optimal configuration can be achieved leading to optimal performance with the lowest possible overhead.

Connection pool tuning best practices

- Maximum connections setting:
  - As a first guess, try doubling the number of the maximum connections parameter.
  - If this solves the problem, then start reducing the number of connections to determine where the failure or bottleneck threshold is.
  - Better performance is generally achieved if this value is set lower than the value for the maximum size of the web container thread pool.

- Connection timeout setting:
  - If a ConnectionWaitTimeoutException is found in the WebSphere logs:
    - Obtain the average database operations duration for the application.
    - Start with a value that is five seconds longer than this average.
    - Gradually increase it until the problem is resolved or the setting is at the highest value that the client will tolerate.

- Before you increase the pool size, consult the database administrator.
  - Ensure that the database server is configured to handle the maximum pool size setting

The database connection pool Minimum and Maximum connections values are often misunderstood. If you set a maximum of 40 connections and a minimum of 10 connections, the pool will not start with 10 connections. The value of 10 connections minimum, is actually a low water mark. Until there are 10 connections required concurrently, the pool will only contain the maximum amount of concurrent connections required up to that point. Therefore, if the number of concurrent connections has only ever reached six, then the pool will contain six connections. Once the number of connections needed exceeds 10, the number of connections in the pool will not drop below 10 until the pool is cleaned out.

For example, after the reap time expires, all unused connections are destroyed until the Minimum connections threshold is reached. Configuring a data sources should be done in consultation with the database administrator.

Connection timeout setting is in seconds, after which a connection request times out and a ConnectionWaitTimeoutException is thrown.

This value indicates the number of seconds that a connection request waits when there are no connections available in the free pool and no new connections can be created. This typically occurs because the maximum value of connections in the particular connection pool has been reached. For example, if Connection Timeout is set to 300, and the maximum number of connections are all in use, the pool manager waits for 300 seconds for a physical connection to become available. If a physical connection is not available within this time, the pool manager initiates a ConnectionWaitTimeout exception.

The connection pool size should not be larger than the number of agents or connections allowed on the database server. This can become an issue, especially in a cluster, because each application server will allocate its own pool. To compute the maximum connections the database must support, multiply the connection pool size by the size of the cluster.

Troubleshooting connection leaks in the problem determination path

Connection pool tuning and management problems

© 2011 IBM Corporation
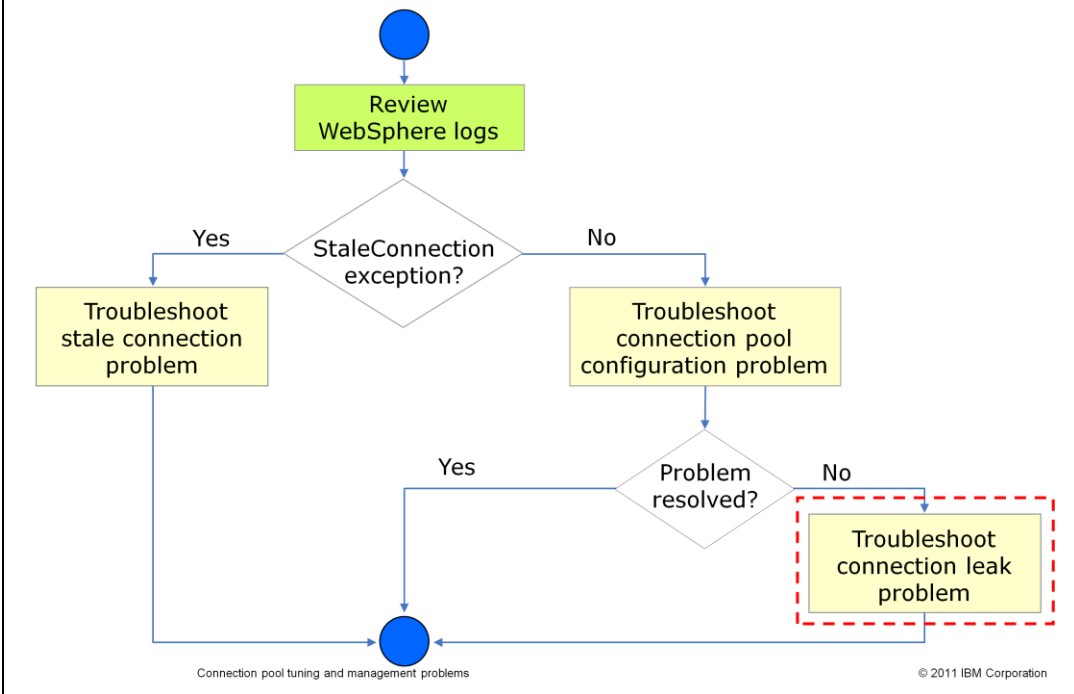
After you have ruled out the possibility of a stale connection and connection pool tuning problem, consider the possibility of a connection leak.

Connection leaks

- A connection leak is a situation that arises when allocated connections are not properly released back to the pool after use.

- It causes:
    - User response time to increase
    - Eventual system lock-up if all worker threads are waiting for a connection
    - ConnectionWaitTimeoutExceptions to be thrown when the connection timeout threshold is reached

Connection pool tuning and management problems   © 2011 IBM Corporation

A connection leak is typically identified by a ConnectionWaitTimeoutException in the WebSphere logs.

WebSphere Application Server is smart enough to eventually time-out orphaned connections and return them to the pool, but for an application that makes frequent use of database connections, this might not be enough corrective action to maintain throughput levels. New connections can get queued up waiting for the database while old connections are waiting to be timed out. This can cause user response time to increase, and eventually system lock-up of all worker threads that are waiting for a connection which in turn will bring the application server to a grinding halt.

Common causes of connection leaks

- Poorly written applications often do not properly release database connections.
  - Forget to call `connection.close()`
  - Appears most often in an exception case
    - Connections should be closed in a `finally{}` block (try-catch-finally)
  - Also caused by one method getting a connection, invoking multiple methods, and then forgetting to close the connection when done

- Orphaned connections will only return to the pool after timeout.
  - Can cause a backup of new connections waiting for old connections to time out
  - New connections that have waited too long create a ConnectionWaitTimeoutException

Connection pool tuning and management problems

© 2011 IBM Corporation

The most common reason for a connection leak is when an application does not defensively manage the connections it requests from WebSphere Application Server.

This often happens because the application does not properly close connections. The connections should be called in the finally code block of a try/catch/finally construct to ensure that connections are closed properly.

Unfortunately, connection leaks have traditionally been hard to diagnose because the error messages do not typically provide specific enough information about the source of the problem. A source code review is typically needed to find where the connections are not being properly closed.

Connection leak diagnosis tasks

- Enable connection leak trace facility
  – Use administrative console

- Run and monitor application
  – Wait for ConnectionWaitTimeoutExceptions to occur

- Review and analyze trace file
  – Locate source of leak and resolve problem

To assist in locating the cause of a connection leak, there are tools built into WebSphere Application Server that can help narrow down the search. The most useful of which is the connection leak trace facility.

Connection leak tracing will allow you to gather more detailed information about the leak when a ConnectionWaitTimeoutExceptions occur. Also the trace utility can help you determine if connections are not being closed or if the application should be redesigned to use fewer connections.

Connection leak trace facility

- A connection leak trace facility is available in WebSphere to provide detailed diagnostic information.
  - Prints stack traces of all open connections to trace.log when a ConnectionWaitTimeoutException occurs
  - Enables you to narrow the search for the responsible source code
  - Lightweight with lower performance overhead than standard connection manager tracing (1 – 5% impact)

- Limitation:
  - Connection leak trace facility only prints a stack trace of those connections that have been in use for more than 10 seconds.

When a thread times out waiting on connection from a full connection pool, it will create a ConnectionWaitTimeoutException. When this exception is thrown, the connection leak tracer will print out the stack traces for every open connection. It does so only when a problem has occurred, providing instant recognition of when it occurred and incurs reduced overhead when compared to the full WebSphere tracing mechanism.

This feature is useful because it shows you the call stacks for all open connections at the time of the exception. This enables you to significantly narrow your search area when you look at the application's source code to try and find the responsible code. It is also helpful to IBM support, because it will help distinguish between application problems and WebSphere defects.

When you enable the connection leak trace facility, for every time interval, the WebSphere Application Server connection pool manager checks how long a connection has been in use and prints the stack trace to the trace log file. Currently the default time interval is unchangeable and defaults to ten seconds. If you have a need to change the default value, contact IBM technical support to obtain an iFix that allows you to add a custom property to the data source configuration.

Enabling the connection leak trace facility

- Enabled using a standard trace string WAS.j2c=finest
  - Includes ConnLeakLogic trace

Logging and Tracing

Logging and Tracing > server1 > Change Log Detail Levels

Use log levels to control which events are processed by Java logging. Click Components to specify a log detail level for individual components, or click Groups to specify a log detail level for a predefined group of components. Click a component or group name to select a log detail level. Log detail levels are cumulative; a level near the top of the list includes all the subsequent levels.

Configuration

General Properties

Change Log Detail Levels

Components

Groups

*=info: WAS.j2c=finest

* [All Components]
ConfigError

Connection pool tuning and management problems

© 2011 IBM Corporation

The connection leak trace facility is enabled through the AdminConsole.

To enable the facility, start by navigating to Troubleshooting, then Logs and Trace. Select the application server you want to trace on and then select Diagnostic Trace, where you can enable standard connection pool trace string WAS.j2c=finest

Make sure logging is enabled and then click the Change log detail levels link. You can then specify the required trace level under the ConnLeakLogic category.

- Search trace.log for the string: Connection Leak Logic Information.
- If present, there are connections that have been in use for more than 10 seconds.
- Analyze their stack trace to identify suspect application methods

```
Connection Leak Logic Information:
  MCwrapper id df00df0   Managed connection WSRdbManagedConnectionImpl@a8c0a8c   State:STATE_T
    Start time inuse Mon Jul 09 02:06:26 EDT 2007 Time inuse 11 (seconds)
    Last allocation time Mon Jul 09 02:06:26 EDT 2007
    getConnection stack trace information:
      com.ibm.ejs.j2c.ConnectionManager.allocateConnection(ConnectionManager.java:712)
      com.ibm.ws.rsadapter.jdbc.WSJdbcDataSource.getConnection(WSJdbcDataSource.java:431
      com.ibm.ws.rsadapter.jdbc.WSJdbcDataSource.getConnection(WSJdbcDataSource.java:400
      com.ibm.connleak.LeakServlet.doGet(LeakServlet.java:30)
      javax.servlet.http.HttpServlet.service(HttpServlet.java:743)
      javax.servlet.http.HttpServlet.service(HttpServlet.java:856)
      com.ibm.ws.webcontainer.servlet.ServletWrapper.service(ServletWrapper.java:989)
      com.ibm.ws.webcontainer.servlet.ServletWrapper.handleRequest(ServletWrapper.java:5
      com.ibm.ws.wswebcontainer.servlet.ServletWrapper.handleRequest(ServletWrapper.java
      com.ibm.ws.webcontainer.webapp.WebApp.handleRequest(WebApp.java:3163)
```

The connection pool trace (WAS.j2c=finest), that you enabled before, will create a log file called trace.log by default. There are a few key lines to look for when you start evaluating the trace files.

You will first want to look for a line that contains the string Connection Leak Logic Information, followed by a colon. This indicates the start of the connection leak logic output. From there, you should check the time in use and the stack trace for each of the connections.

In this example trace, there are connections that have been in use for more than 10 seconds and is therefore a good suspect for a source of a leaking connection.

Troubleshooting stale connections in the problem determination path
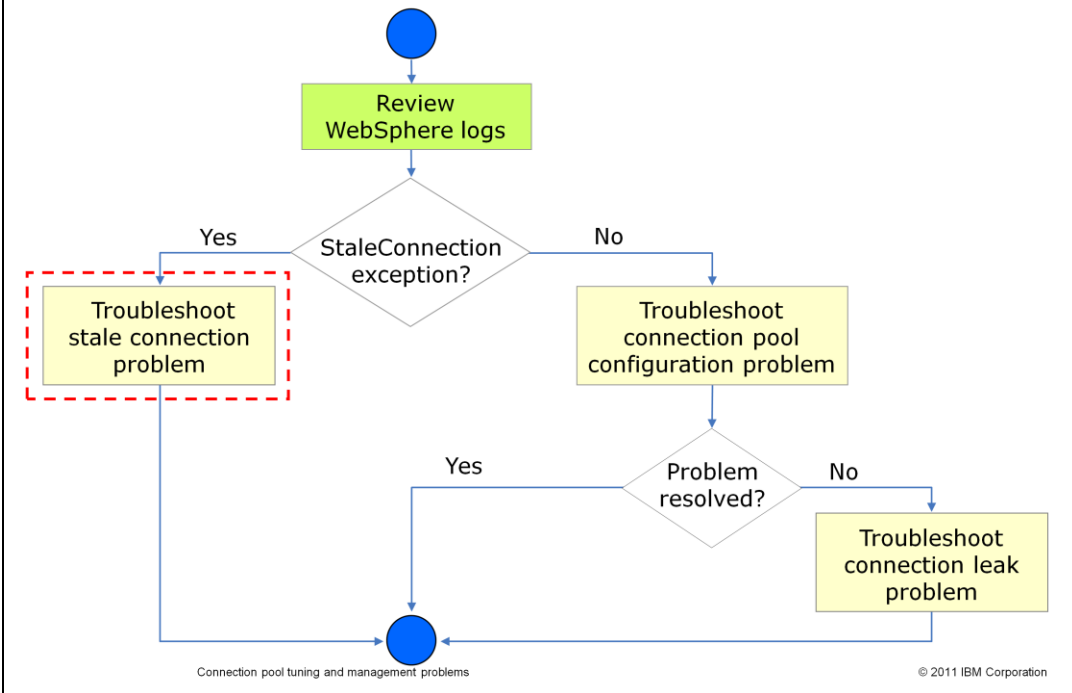
If you find StaleConnectionExceptions in the the WebSphere Application Server logs then your choices are clear; start looking for stale connection problems

Stale connections

- A stale connection problem arises when a connection held by a client is not longer valid.

- This situation can occur for many reasons, including:
    - A connection is no longer usable because of a database failure.
    - An attempt is made to reuse an orphaned connection (applies only to version 4.0 data sources).
    - A connection is closed by the version 4.0 data source auto connection cleanup feature and is no longer usable

A stale connection is essentially a connection that is held by a client but is no longer a valid connection.

One way this can happen is if the application tries to get a connection and fails, because the database is not started or for some reason is no longer available.

Stale connections can also occur in Version 4.0 data sources when the connection is closed by the connection cleanup feature but the client is still trying to use it. This will happen if the connection has not been in use at least twice the unused timeout value. At this point, the connection is orphaned and the client will error if it tries to use the connection again.

Recovering from a stale connection

- In general, a stale connection condition indicates that the connection to the database has gone bad.
  - The connection cannot be recovered and must be completely closed rather than be returned to the pool.

- Recovering from stale connections is a joint effort between the application server runtime and the application developer:
  - The application server will purge the connection pool based on its PurgePolicy setting and eliminate the bad connection.
  - The application developer can explicitly catch a stale connection exception and programmatically recover from bad connections (for example, get a new one).

An individual connection cannot be recovered once it creates a StaleConnectionException. Instead, the best way to recover from this type of exception is by explicitly catching it. Catching a StaleConnectionException while running within the context of a transaction will allow you the avoid having to repeat the entire transaction.

One option is to try and complete the pending transaction with a new connection. It is important to note that the application server will also take actions to recover from a StaleConnectionException depending on the PurgePolicy setting. It can either clear the entire connection pool, assuming that if one connection went bad then all other connections will likely have the same problem, or just clear the stale connection.

Other stale connection troubleshooting tasks

- Check database or firewall timeout settings.
  - Consult with database administrator or network system administrator for the presence of these timeouts.
  - If present, they can close connections and cause StaleConnectionExceptions.

- Determine if a specific query is getting the exception.
  - Examine the SQLState and SQLCode returned with the exception.

- Trace the problem using one or more of these options:
  - WAS.database
  - RRA
  - WAS.j2c

Connection pool tuning and management problems

© 2011 IBM Corporation

There are several other reasons why a connection might become stale, many of which exist beyond the control of WebSphere Application Server.

One common reason is a discrepancy between the firewall timeout settings and the connection timeout settings. It is generally a good practice to make sure the connection pool aged timeout is less than the firewall's timeout and that both are less than the database timeout.

It is also possible that you are experiencing a StaleConnectionException because the returned SQLCode maps to a StaleConnection. If you aren't able to find the source of the problem by taking a quick look at the various components involved in the connection then your best bet is to turn on tracing and gather more information. This is can be very useful when a connection is unusable because of a SQLException that did not immediately map to a StaleConnectionException but eventually resulted in one being thrown.

Connection pool troubleshooting tools (1 of 2)

- Diagnostic Provider ConnMgrDP for specific data sources
  - View state data

| Cancel | Save | State specification settings |

Node ◇	Server ◇	Name ◇	Value ◇
You can administer the following resources:			
was7host01Node01	server1	approximateNumberFreeConnections	0
was7host01Node01	server1	approximateNumberSharedConnections	10
was7host01Node01	server1	approximateNumberUnSharedConnections	0
was7host01Node01	server1	approximateNumberWaitingConnections	20

Connection pool tuning and management problems   © 2011 IBM Corporation

The infrastructure for implementing diagnostic providers was first introduced in WebSphere Application Server V6.1. Based on that infrastructure, each individual component of the system can implement its own diagnostic provider, tailored to that particular component.

Connection manager (ConnMgrDP) provides configuration and state information about data sources and connection factories. This includes information about managed connections, active transactions, connection pool information, and other useful data.

Connection pool troubleshooting tools (2 of 2)

The IBM Database Connection Pool Analyzer tool employs a heuristic analysis engine to help you resolve problems related to Java Database Connectivity (JDBC) connection pools and find JDBC connection leaks.

First, you must enable traces (database, RRA, and j2c) for WebSphere to create a log file that outputs diagnostic information about the connection pool, which in turn is uploaded to the tool for data analysis.

The tool performs operations like parsing trace files for th eJDBC connection pool manager, detecting available JDBC data sources and configurations, counting the number of waiters for connections and number of connections in use, finding the Java stack traces associated with JDBC connections, and providing charts of JDBC connections for each JDBC data source.

Unit summary

Now that you have completed this unit, you should be able to:

- Identify connection pool problems
- Describe what to look for in the application server logs
- Enable tracing for connection manager components
- Interpret and analyze the trace data
- Explain how to use PMI to monitor connections

© 2011 IBM Corporation

Now that you have completed this unit, you should be able to identify connection pool problems, describe what to look for in the WebSphere Application Server logs, enable tracing for connection manager components, interpret and analyze the trace data, and describe how to use PMI to monitor connections.

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send email feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_WA5721G10_ConnectionPool_edited.PPT

This module is also available in PDF format at: ../WA5721G10_ConnectionPool_edited.pdf

You can help improve the quality of IBM Education Assistant content by providing feedback.