Connectors and Transaction Management (JDBC, JMS, J2CA, DataSource, Connection Pool, EIS)

Overview

Inbound and outbound connections in Geronimo are managed through the J2CA connector framework. This covers JDBC, JMS, DataSources, EIS connectivity, and connection pools. JDBC connectivity is implemented by wrapping JDBC XADataSource, ConnectionPoolDataSource, DataSource, or Driver implementations in J2CA connector wrappers. The wrappers we use are from the codehaus Tranql project.

Transaction Manager

Applications can delegate transaction management to the server infrastructure through application managed UserTransactions or for ejbs container manager transactions. The server uses a transaction manager to coordinate this application view of transaction boundaries with the resource manager view of work done on specific data stores within particular transactions. Geronimo uses a JTA TransactionManager implementation that is extended in three ways:

1. Transaction inflow as required by the J2CA inbound connector requirements
2. Pluggable transaction logging, by default using HOWL (High speed ObjectWeb Logger) for recoverable transaction managers in a server and a non-logging "logger" in application clients
3. Incremental recovery of in doubt transactions as connectors start up after a failure. This allows transactions that use a subset of the deployed connectors to be recovered completely as soon as the connectors they use are restarted, even if some other resource managers are not yet available.

Requirements for XAResource implementations to enable recovery

The XA spec does not provide any way to identify which XAResource an xid in an in-doubt transaction is associated with. In order for recovery to work without this information there needs to be a global registry of all resource managers, and recovery can only start after all resource managers are started and connected to the transaction manager. In order to avoid the need for this kind of permanent registry and reliance on the complete environment being present, geronimo associates each xid with the XAResource it is used with. This is done by requiring every XAResource to have a name, by implementing NamedXAResource (a geronimo interface). Also, each connector (or other "source" of an XAResource) is required to register the NamedXAResource with the transaction manager when it starts. The name is stored with the xid, so recovery of a particular transaction can complete as soon as all the XAResources used in that particular transaction have registered.
The name supplied by a NamedXAResource must be stable over restarts and unique to the resource manager. In geronimo we use the gbean name of the ManagedConnectionFactoryWrapper for outbound connections and the ResourceAdapter gbean name for inbound connectors.

Configuring the identity of the transaction manager.

If you are running more than one geronimo instance against a resource manager such as an XADatabase, you need to make sure that the resource manager can distinguish which geronimo instance is making each request. You do this by setting the transaction manager identity. See Configuring the Transaction Manager Identity

Connector Framework

The J2CA connector spec provides a reasonable approach to configuring and pooling outbound connections and configuring and managing inbound message delivery. Geronimo uses this for jdbc connection support by wrapping jdbc artifacts in the required J2CA artifacts, using the tranql project wrappers.

Connection Pooling (outbound connectors)

Often connections to legacy EIS systems and databases have connections that are extremely expensive to set up. Rather than creating a new connection to the external system every time one is needed, normally you use a connection pool: when a connection is needed an unused connection is supplied from the pool, and when the application is done with it the connection is put back in the pool. The J2CA spec supports this idea with these components:

Connection Factory

A connection factory (e.g. DataSource) is a component used by an application to get "connections" it can use. These connections are short lived, lightweight handles or wrappers to underlying "physical" managed connections. When an application calls datasource.getConnection(), the connection factory routes the request to the connection manager, which supplies a suitable connection handle from a cached, pooled, or new connection.

ConnectionManager

The connection manager received connection requests from the connection factory and figures out how to accomodate them. It also receives notification of transaction events and transfer of control between application components such as ejbs.
Generally, if there is a cached ManagedConnection in the environment, the connection manager returns a new handle to the cached ManagedConnection. Otherwise it looks for a suitable ManagedConnection in the pool, and returns a handle to such a connection if it exists. Otherwise it creates a new connection.

Transaction support

Connection managers in geronimo can be configured with xa, local, or no transaction support.

No transaction

No Transaction support means that connections are never enrolled in jta transactions and committing such a jta transaction has no effect on any connections managed by the connection manager. In this case the connection manager does no caching and each connection request is handled independently.

Local transaction

Local transaction support relies on the connector support of j2ca local transaction control. This maps to such things as non-xa jdbc Driver, DataSource, and ConnectionPoolDataSource implementations. In this case geronimo constructs a "fake" XAResource that enables the JTA transaction manager to control the local transaction, however xa features such as separation of prepare and commit phases and recovery are not implemented. Since the resource manager has no transaction identifier that lets it distinguish which transaction work is being done on, the connection manager has to keep track of this by caching all the connections enrolled in a jta transaction. Whenever your application requests a connection while in a jta transaction, the connection manager checks for an existing connection enrolled in this transaction and if found supplies an additional handle to it. If no connection is associated, one is fetched from the pool (or created anew) and cached with the transaction. Furthermore, closing one or all of these connection handles will not result in returning the connection to the pool. The connection is returned to the pool only when all the handles have been closed and the jta transaction completes.

XA transaction

XA transaction support should theoretically not require caching connections with transactions but in practice usually does. The XA spec indicates that any connection should be able to handle work in any transaction at any time, but few if any XA capable resource managers actually implement this. Normally you must do all work in a JTA transaction against a given resource manager over the same physical connection. Setting transaction-caching on will assure that this caching takes place.
The intent of the enlist/delist methods on XAResource appears to be to support reusing a connection in a thread as it enters and leaves "Requires New" transaction boundaries. By default geronimo does not attempt to reuse connections in this way: the connections associated with the previous transaction are left associated with it and new connection requests get new ManagedConnections from the pool. There is an experimental configuration to reuse the connections already associated with the thread.

Security (Credentials)

In the J2CA 1.5 spec security considerations are confined to providing the credentials for outbound connections. There is no provision for credential inflow for inbound messaging. This hole is being addressed in the next spec revision.
There are three possible sources for credentials for outbound connections:

1. ManagedConnectionFactory configuration. Most ManagedConnectionFactory implementations include user and password config-properties. In the absence of other credentials these are used to authenticate to the resource manager. The values of these properties are generally difficult to conceal securely as they are in some geronimo configuration files on disk and need to be accessible to geronimo in order to create connections. However, this is by far the most frequently used option. This will result in all work in your e.g. database being done by the same user, which basically represents the identity of the geronimo server.
2. Application managed security. Many connection factories such as DataSource provide a way to request connections with a particular security context, such as dataSource.getConnection(user, password). This requires your app to have detailed knowledge of the credentials needed which is rarely appropriate.
3. Container managed security. With this scheme you deploy a LoginModule specific to the connector you are using that installs suitable credentials into the Subject on login. When a connection is requested these credentials are extracted from the Subject and used in obtaining the connection. A wide variety of schemes can be used here including

- Fixed credentials. Every subject gets the same credentials. This has the same effect as the default credentials scheme except that the credentials may be stored in an external more secure location more easily.
- User credentials. The credentials are the user and password of the user themselves. In this case the work will be done in the resource manager (database) as the user themselves. Unless the connector supports reauthentication (changing the user on an existing connection) this will prevent effective connection pooling.
- Mapped credentials. The credentials for the resource manager are determined from the user identity. For instance the resource manager user could depend on the user being a member of a particular enterprise role or group. In this case without reauthentication support connection pooling should be set up so each resource manager user gets a separate pool.

  ⚠ TODO container managed security example

Pooling configuration

Effective connection pooling relies on all the connections in a pool being effectively identical. We won't discuss the impact of reauthentication support here: the only place I know it is available is in some Oracle connectors, and only through using some special packages. Geronimo allows you to set up pooling for a single connection manager/ManagedConnectionFactory with either one pool or multiple pools. Multiple pools are distinguished by credentials, either from application managed security or container managed security. Given a request geronimo looks for a pool with matching credentials and creates one if there is none. In particular this means that pool size is per credential set, so with 10 distinct users and a pool size of 10, you may have 100 connections in the multiple pools.
Within a single pool, you can specify how hard geronimo works to assure that a connection matches the request.

- If you are certain that all connections are really identical, specify select-one-assume-match and geronimo will do no matching. This is the fastest option but will result in errors if connections are not in fact identical.
- If connections are generally the same but you aren't quite so sure, match-one will pick a connection from the pool and match it with the request. If the match fails, the selected connection will be destroyed.
- If the connections are not the same, use match-all. Geronimo will match each request will all the pooled connections to select the best match. This is likely to work badly with concurrent loads.

ManagedConnectionFactory and ManagedConnection

An outbound connector includes ManagedConnectionFactory and ManagedConnection implementations. The ManagedConnection represents an actual "physical" connection to the resource manager (database). The ManagedConnectionFactory contains configuration for the desired ManagedConnections such as the location of the resource manager. It also contains matching code to select a connection matching a request from a set of connections. The ManagedConnection also can provide connection handles that the connection factory can return.
In geronimo we wrap the ManagedConnectionFactory with a gbean so it can be configured like other geronimo components.

JDBC

JDBC support is generally provided through wrapping JDBC artifacts in J2CA artifacts from the codehaus tranql project. Some databases such as Firebird also provide J2CA implementations that can be deployed directly. Tranql provides a framework for constructing outbound resource adapters and a variety of published adapters.

Generic Wrapper

All JDBC implementations provide at least a Driver implementation. Tranql provides a generic connector org.tranql:tranql-connector-ra:1.4:rar that can be used with any Driver implementation. This provides only local transaction support and requires that you encode any specific connection properties in the jdbc url.

Database specific wrappers

Most JDBC implementations also provide DataSource, ConnectionPoolDataSource, or XADataSource implementations. The tranql framework is designed to make it easy to write database specific wrappers for these classes. With such wrappers properties that can be set on these connection factories or "managed connection factories" are exposed as ManagedConnectionFactory config-properties and can be documented in the ra.xml. Currently tranql provides wrappers for DB2 (xa), Derby (local and xa, embedded and client), mysql (local and xa), oracle (local and xa), and postgres (local and xa).

JMS

As of J2EE 1.4 JMS implementations are expected to provide a J2CA resource adapter for use in EE environments. Geronimo relies on this for JMS support. Geronimo provides ActiveMQ as the default JMS provider.

EIS

The J2CA spec envisages connection between EE servers and EIS through J2CA connectors. J2CA provides for outbound connections as already described and inbound connections to MDBs. Along with JMS style asynchronous inbound messaging, in which the message may be delivered in a container controlled transaction, message delivery may be performed in an adapter provided transaction. Currently there are no facilities for security context import but this is expected in the next J2CA spec revision.