Using Databases in Apache Cocoon (2.1 legacy document)

Table of contents

Table of contents

Warning:
This document was copied as is from the Cocoon 2.1 documentation, but has not yet been fully reviewed or moved to its new home.

1. How do I choose my database?

Apache Cocoon is flexible in the way it allows you to make connections to a database. There are basically two ways: by redefining all the connection parameters in each page you use a database, or using a pooled connection. The first method is slow and doesn't scale well. The second method is more scalable, and depending on your database will realize true improvements.

1.1. Installing the Driver

Independent of how you choose to get and maintain your JDBC connections, you have to load the driver so Cocoon can use it (unless you are using a J2EE container--more on that later). This is an init parameter in your web.xml file. The following snippet will show you how:

<init-param> <param-name>load-class</param-name> <param-value> <!-- For PostgeSQL Database: --> postgresql.Driver <!-- For Oracle Database: --> oracle.jdbc.driver.OracleDriver </param-value> </init-param>

You can place as many Driver classes in this parameter you want. They are separated by white space or commas.

1.2. Defining a Data Source

Cocoon allows you to specify a pooled data source that you can use for throughout the Cocoon system. There are two different types of data sources: JDBC and J2EE. The difference is in who controls the connection. The JDBC data source lets Cocoon handle all the pooling logic. The J2EE data source tells Cocoon how to pull the DataSource object from a J2EE container (thats Java 2 Enterprise Edition)--the major caveat is that Cocoon must be installed as part of a Enterprise Application.

The following snippet of cocoon.xconf shows the section where the DataSourceComponent is specified. You can have more than one in this location. The code will have one connection for the JDBC data source, and one connection for the J2EE data source.

<datasources> <jdbc name="MyConnectionName"> <pool-controller min="5" max="10"/> <dburl>jdbc:oracle:thin:@localhost:1521:mydatabase</dburl> <user>mylogin</user> <password>myPassword</password> </jdbc> <j2ee name="MyJ2eeConnection"> <dbname>cocoonDB</dbname> </j2ee> </datasources>

1.2.1. The JDBC Connection Properties

The JDBC connection has up to five different properties--but only one is absolutely required.

- dburl: This is absolutely required. Without it JDBC can't connect to the database.
- user: This is only required if the database admin requires you to log in to the database.
- password: This is only required if the database admin requires a password to connect to the database.
- pool-controller: This has two parameters with defaults. If it is not specified, the defaults are used.
  - min: The minimum number of connections the pool will keep available at one time. Defaults to zero (0).
  - max: The maximum number of connections the pool will have created at the same time. Defaults to three (3).

    - oradb: If you have an Oracle database, you should add the attribute "oradb" and set it to true.
- auto-commit: If you need to ensure an autocommit is set to true or false, then create the "auto-commit" element.

1.2.2. The J2EE Connection Property

The J2EE connection has only one property and it is absolutely required. Cocoon uses JNDI to look up the DataSource with the name you specified in "dbname".

1.3. Using the Data Source Component

No matter how you defined your DataSourceComponent, you access it the same way. Because The DataSourceComponent is a Component, your class needs to implement the Avalon Serviceable interface. The Avalon Framework will give your class a ServiceManager. At that point, it is up to you how and when you pull the DataSourceComponent out of the ServiceManager.

import org.apache.avalon.framework.service.ServiceManager; import org.apache.avalon.framework.service.ServiceSelector; import org.apache.cocoon.Roles; import org.apache.avalon.excalibur.datasource.DataSourceComponent; import java.sql.Connection; // .... Skip a lot of lines until we are in the method you use // to initialize the DataSourceComponent .... private DataSourceComponent datasource; public void service(ServiceManager manager) { ServiceSelector selector = (ServiceSelector) manager.lookup(Roles.DB_CONNECTION); this.datasource = (DataSourceComponent) selector.select("MyConnectionName"); } // .... Skip more lines until we actually need to use the datasource private void meMethod() { Connection myConnection = this.datasource.getConnection(); // .... perform SQL code here myConnection.close(); }

Notice that once you obtained your connection, you did nothing out of the ordinary to return the connection to the pool? This is by design, and a result of the JDBC specification. Basically the JDBC specification states that if a driver implements pooled connections, then it should not alter the way those connections are used. This maintains the portability of your code.

1. Comments

add your comments