Database Access (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. Introduction

Publishing dynamic content or creating web-applications eventually involves database access. Apache Cocoon offers a number of different approaches to access (object) relational and XML databases. This document provides an overview of the different ways to access (object) relational databases.

This document will not explain how to set up database connectivity with Apache Cocoon. For this, see here.

Basically, there are three different approaches available: Actions, logicsheets, and transformers. Each approach has its pros and cons.

2. Actions Approach

Actions are code to be executed during pipeline setup. The outcome of an action can change how a pipeline is assembled. For example, a pipeline may produce an alternative page to display upon failure of a particular database operation.

Actions are especially great for inserting, changing, or deleting data. Employing the pipeline-switching features of actions will simplify your pages. Such actions are concerned with only one view: either the success or failure of an operation.

Actions can be useful, even when data is not provided by users. For example, you could store tracking information in a database in a central location without the need to modify every page.

Database actions can read and return data from a database. This is useful when the pipeline assembly depends upon such data. It's also useful when setting up an environment for XSP processing.

Once the database meta data is captured in an XML descriptor file, making use of these actions is simply a matter of placing them in a pipeline. This is a major advantage of the action approach. No programming is required, not even SQL query writing.

For more detailed information, read: Database Actions.

3. ESQL Logicsheet Approach

The use of logicsheets is limited to XSPs. ESQL is currently available for Java-based XSPs. Its interface is modeled largely on JDBC. Thus, it is advantageous to be familiar with JDBC.

ESQL is great when reading data from a database. However, it is less attractive to use when it has to react to operation failures. This is due to the fact that it adds a layer of complexity to an XSP file, making it more difficult to understand and maintain.

Complex layouts of the data are easy to achieve. ESQL allows the arbitrary nesting of queries and connections. It also provides support for stored procedures and complex data types. ESQL provides a means to create a structured representation of the database data with a single tag. This is useful when generating reports to use with other XML-aware software or to be formated with XSL or CSS2. XML data can be retrieved from the database and included in the output. With some supported database management systems, ESQL supports skipping part of the resultset as well as limiting the result. Given the full power of Java available within XSP, any processing of the data is possible.

For more detailed information, read: ESQL Taglib.

4. SQL Transformer Approach

An approach using the SQL transformer can be combined with any kind of page. This will result in slightly cleaner pages as you don't need some of the setup that an ESQL approach requires.

On the other hand, it is more or less impossible to react to operation failures. This is due to the fact that the pipeline is already assembled and the necessary logic to handle such failures is not available inside the SQL transformer, unless of course, you are willing to write a custom transformer. Thus, the transformer approach is best for retrieving data. Creating an XML representation of the query result is even simpler than when using the ESQL logicsheet. The transformer also supports stored procedures. No programming is required, apart from writing SQL.

For more detailed information, read: SQL Transformer.

1. Comments

add your comments