Chapter 1. Selector

Table of Contents

Introduction

The selector package makes it very easy to add conditional expression evaluation to your applications. The syntax of the selector expression is based on a a subset of the SQL92 conditional expression syntax. In order to evaluate the selector, the values for the various identifiers in the selector expression need to be provided. The selector package allows for great flexability in specifying the source of the identifier values. The source can either be a Map containing the identifer name-value pairs or by registering a class that implements the IValueProvider interface with the selector. During the evaulation of the selector, the selector invokes callbacks on either the Map or registered IValueProvider to get the value of the desired identifier. Implementations of the IValueProvider for JMS MapMessages and TIBCO Rendezvous is provided and an implemenatation for JavaBean properites is planned.

This package was originally used in the context of building Message oriented middleware (MOM) applications. JMS defines the concept of selectors which are used by JMS clients to filter messages. A JMS selector matches a message if the selector evaluates to true when the message's headers and properties are substituted for their corresponding identifiers in the selector. A JMS Client application can optionally specify a selector when they create a message consumer for a specific destination. If a selector is specified, then all messages are filtered through the selector before being delivered to the consumer. The selector package was created so that the selector concept could be extended when using JMS to apply to the JMS message body and also for use with non-JMS based messaging middleware, such as TIBCO Rendezvous. The nature of processing messages on the client side in a single callback method usually results in an ugly 'switch' statement to route messages with particular body contents to the appropriate business logic. Externalizing this selection process with a SQL string effectively organizes the message processing and makes the selection criteria transparent to developers. The reactor package complements the selector to provide....

Selector Syntax

A message selector is a String whose syntax is based on a subset of the SQL92 conditional expression syntax. The order of evaluation of a selector is from left to right within precedence level. Parenthesis can be used to change the evaluation order. Predefined selector literals and operator names are shown here in uppercase, however, they are case insensitive.

A selector contains Literals which are:

- A string literal is enclosed in single quotes
- An exact numeric literal is a numeric value without a decimal point, such as 57, -91, and +62; numbers in the range of Java long as supported.
- An approximate numeric literal is a numeric value is scientific notation, such as 7E3, -57.9E2, or a

numeric value with a decimal such as 7.1, and -95.6 numbers is the range of Java double are supported.
- The boolean literals TRUE and FALSE.

More to come...

Creating Selectors

Evaluating Selectors

JMS

The JMS specification restricts selector identifiers to header fields and properties values. However, applications frequently require the ability to filter/select messages based on their content. One option for applications is to place relevant message fields as properties in order to allow selection. However, one quickly discovers developers leveraging this feature as a means to direct processing logic in the client. This results in an overpopulation and repetition of data in the body and header. A better alternative is to use the built-in JMS selector to reduce message flow to the client to an acceptable level, and then use the same approach in the client to organize message processing.

To allow applications to filter messages based on content, the JMSValueProvider provides the following extension to the JMS specification: identifier names can contain '.' (dot). This extension can be used by value provider classes to extract values of message content fields. Some JMS vendors support the nesting of MapMessage within a MapMessage. The JMSValueProvider supports this extension.

Thread Safety

The implementation makes heavy use of the immutable pattern. All classes in the package are immutable. As a result, no internal or external synchronization is required. Instances of Selector may be shared freely between threads.

Performance

The selector implementation is highly performant and uses little memory. Once a selector has been parsed and the in-memory representation has been built, evaluating it is extremely efficient.

For example, consider the following selector: String selector = "JMSPriority >= 0 AND Quantity > 100 AND MessageName in ('Login', 'Logout', 'Query', 'CreateOrder', 'CreateWatchlist', 'Trade', 'OrderAction', 'Orderbook')";

This selector can be evaluated using the JMS value provider at a rate of about 200,000 evaluations per second on a low-end PC running Windows 2000 with 512 MB of RAM. As a comparison, the JBOSS open source selector implementation is roughly four times slower.