6

Labeling Connections in UCP

This chapter discusses the following topics:

6.1 Overview of Labeling Connections in UCP

Applications often initialize connections retrieved from a connection pool before using the connection. The initialization varies and could include simple state re-initialization that requires method calls within the application code or database operations that require round trips over the network. The cost of such initialization may be significant.

Labeling connections enables an application to attach arbitrary name/value pairs to a connection. The application can request a connection with the desired label from the connection pool. By associating particular labels with particular connection states, an application can retrieve an already initialized connection from the pool and avoid the time and cost of re-initialization. The connection labeling feature does not impose any meaning on user-defined keys or values; the meaning of user-defined keys and values is defined solely by the application.

> **Note:**
>
> If you are using connection labeling, then you cannot set the `RESET_STATE` service attribute to `LEVEL1` or `LEVEL2`.

Some of the examples for connection labeling include, role, NLS language settings, transaction isolation levels, stored procedure calls, or any other state initialization that is expensive and necessary on the connection before work can be executed by the resource.

Connection labeling is application-driven and requires the use of two interfaces. The `oracle.ucp.jdbc.LabelableConnection` interface is used to apply and remove connection labels, as well as retrieve labels that have been set on a connection. The `oracle.ucp.ConnectionLabelingCallback` interface is used to create a labeling callback that determines whether or not a connection with a requested label already exists. If no connections exist, the interface allows current connections to be configured as required. The methods of these interfaces are described in detail throughout this chapter.

6.2 Implementation of a Labeling Callback in UCP

UCP uses Database Resident Connection Pooling (DRCP) tagging infrastructure to support labeling in UCP, whether you work with single labels or multiple labels. However, the behavior with multiple labels can be a little different when you use the UCP and DRCP combination instead of only UCP.

This section discusses the following topics:

- When to Use a Labeling Callback in UCP
- Creating a Labeling Callback in UCP
- Registering a Labeling Callback in UCP
- Removing a Labeling Callback in UCP

> **See Also:**
>
> "Integration of UCP with DRCP"

6.2.1 When to Use a Labeling Callback in UCP

A labeling callback is used to define how the connection pool selects labeled connections and allows the selected connection to be configured before returning it to an application. Applications that use the connection labeling feature must provide a callback implementation.

A labeling callback is used when a labeled connection is requested but there are no connections in the pool that match the requested labels. The callback determines which connection requires the least amount of work in order to be re-configured to match the requested label and then enables the connection labels to be updated before returning the connection to the application. This section describes the following topics:

6.2.2 Creating a Labeling Callback in UCP

To create a labeling callback, an application implements the `oracle.ucp.ConnectionLabelingCallback` interface. One callback is created per connection pool. The interface provides the following two methods:

- The cost Method
- The configure Method

The cost Method

This method projects the cost of configuring connections considering label-matching differences. Upon a connection request, the connection pool uses this method to select a connection with the least configuration cost.

```
public int cost(Properties requestedLabels, Properties currentLabels);
```

The configure Method

This method is called by the connection pool on the selected connection before returning it to the application. The method is used to set the state of the connection and apply or remove any labels to/from the connection.

```
public boolean configure(Properties requestedLabels, Connection conn);
```

The connection pool iterates over each connection available in the pool. For each connection, it calls the `cost` method. The result of the `cost` method is an `integer` which represents an estimate of the cost required to reconfigure the connection to the required state. The larger the value, the costlier it is to reconfigure the connection. The connection pool always returns connections with the lowest cost value. The algorithm is as follows:

- If the `cost` method returns `0` for a connection, then the connection is a match. The connection pool does not call the `configure` method on the connection found and returns the connection as it is.
- If the `cost` method returns a value greater than `0`, then the connection pool iterates until it finds a connection with a cost value of `0` or runs out of available connections.
- If the pool has iterated through all available connections and the lowest cost of a connection is `Integer.MAX_VALUE` (2147483647 by default), then no connection in the pool is able to satisfy the connection request. The pool creates and returns a new connection. If the pool has reached the maximum pool size (it cannot create a new connection), then the pool either throws an SQL exception or waits if the connection wait timeout attribute is specified.
- If the pool has iterated through all available connections and the lowest cost of a connection is less than `Integer.MAX_VALUE`, then the `configure` method is called on the connection and the connection is returned. If multiple connections are less than `Integer.MAX_VALUE`, the connection with the lowest cost is returned.

> **Note:**
>
> A cost of `0` does not imply that `requestedLabels` equals `currentLabels`.

6.2.2.1 Example of Labeling Callback in UCP

The following example demonstrates a simple labeling callback implementation that implements both the `cost` and `configure` methods. The callback is used to find a labeled connection that is initialized with a specific transaction isolation level.

```
class MyConnectionLabelingCallback
  implements ConnectionLabelingCallback {

  public MyConnectionLabelingCallback()
  {
  }

  public int cost(Properties reqLabels, Properties currentLabels)
  {
    // Case 1: exact match
    if (reqLabels.equals(currentLabels))
    {
      System.out.println("## Exact match found!! ##");
```

```
      return 0;
    }

    // Case 2: some labels match with no unmatched labels
    String iso1 = (String) reqLabels.get("TRANSACTION_ISOLATION");
    String iso2 = (String) currentLabels.get("TRANSACTION_ISOLATION");
    boolean match =
      (iso1 != null && iso2 != null && iso1.equalsIgnoreCase(iso2));
    Set rKeys = reqLabels.keySet();
    Set cKeys = currentLabels.keySet();
    if (match && rKeys.containsAll(cKeys))
    {
      System.out.println("## Partial match found!! ##");
      return 10;
    }

    // No label matches to application's preference.
    // Do not choose this connection.
    System.out.println("## No match found!! ##");
    return Integer.MAX_VALUE;
  }

  public boolean configure(Properties reqLabels, Object conn)
  {
    try
    {
      String isoStr = (String) reqLabels.get("TRANSACTION_ISOLATION");
      ((Connection)conn).setTransactionIsolation(Integer.valueOf(isoStr));
      LabelableConnection lconn = (LabelableConnection) conn;

      // Find the unmatched labels on this connection
      Properties unmatchedLabels =
       lconn.getUnmatchedConnectionLabels(reqLabels);

      // Apply each label <key,value> in unmatchedLabels to conn
      for (Map.Entry<Object, Object> label : unmatchedLabels.entrySet())
      {
        String key = (String) label.getKey();
        String value = (String) label.getValue();
        lconn.applyConnectionLabel(key, value);
      }
    }
    catch (Exception exc)
    {
      return false;
    }
    return true;
  }
}
```

6.2.3 Registering a Labeling Callback in UCP

A pool-enabled data source provides the `registerConnectionLabelingCallback(ConnectionLabelingCallback callback)` method for registering labeling callbacks. Only one callback may be registered on a connection pool. The following example demonstrates registering a labeling callback that is implemented in the `MyConnectionLabelingCallback` class:

```
MyConnectionLabelingCallback callback = new MyConnectionLabelingCallback();
pds.registerConnectionLabelingCallback( callback );
```

6.2.4 Removing a Labeling Callback in UCP

A pool-enabled data source provides the `removeConnectionLabelingCallback()` method for removing a labeling callback. The following example demonstrates removing a labeling callback.

```
pds.removeConnectionLabelingCallback( callback );
```

6.3 Integration of UCP with DRCP

Natively, DRCP supports connection tagging, which is a single label without weights. So, labeling with a single label works transparently if you use UCP with DRCP.

> **Note:**
>
> Oracle recommends that the maximum pool size of UCP should not be bigger than the size of DRCP. If the UCP pool size is bigger than the DRCP size, then you must set the `setValidateConnectionOnBorrow` property to `off`. Otherwise, UCP keeps invalidating and closing the connections that are not associated with DRCP at that moment, and keeps creating fresh connections.

> **See Also:**
>
> Overview of Validating Connections in UCP

Multiple label UCP connections work, but they have the following behavior changes:

- The `cost` method in the `ConnectionLabelingCallback` API is not invoked if you use UCP with DRCP using connection labeling
- UCP can invoke the `configure` method in the `ConnectionLabelingCallback` API more with DRCP configuration than without DRCP configuration.

> **See Also:**
>
> *Oracle Database JDBC Developer's Guide* for more information about DRCP

6.4 Applying Connection Labels in UCP

Labels are applied on a borrowed connection using the `applyConnectionLabel` method from the `LabelableConnection` interface. This method is typically called from the `configure` method of the labeling callback. Any number of connection labels may be cumulatively applied on a borrowed connection. Each time a label is applied to a connection, the supplied key/value pair is added to the collection of labels already applied to the connection. Only the last applied value is retained for any given key.

> **Note:**
>
> A labeling callback must be registered on the connection pool before a label can be applied on a borrowed connection; otherwise, an exception is thrown.

The following example demonstrates initializing a connection with a transaction isolation level and then applying a label to the connection:

```
String pname = "property1";
String pvalue = "value";
Connection conn = pds.getConnection();

// initialize the connection as required.

conn.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);

((LabelableConnection) conn).applyConnectionLabel(pname, pvalue);
```

In order to remove a given key from the set of connection labels applied, apply a label with the key to be removed and a `null` value. This may be used to clear a particular key/value pair from the set of connection labels.

Related Topics

- Implementation of a Labeling Callback in UCP

6.5 Borrowing Labeled Connections from UCP

A pool-enabled data source provides two `getConnection` methods that are used to borrow a labeled connection from the pool. The methods are shown below:

```
public Connection getConnection(java.util.Properties labels )
   throws SQLException;

public Connection getConnection( String user, String password,
                                 java.util.Properties labels )
   throws SQLException;
```

The methods require that the label be passed to the `getConnection` method as a `Properties` object. The following example demonstrates getting a connection with the label `property1, value`.

```
String pname = "property1";
String pvalue = "value";
Properties label = new Properties();
label.setProperty(pname, pvalue);

Connection conn = pds.getConnection(label);
```

6.6 Checking Unmatched Labels in UCP

A connection may have multiple labels that each uniquely identifies the connection based on some desired criteria. The `getUnmatchedConnectionLabels` method is used to verify which connection labels matched from the requested labels and which did not. The method is used after a connection with multiple labels is borrowed from the connection pool and is typically

used by a labeling callback. The following example demonstrates checking for unmatched labels.

```
String pname = "property1";
String pvalue = "value";
Properties label = new Properties();
label.setProperty(pname, pvalue);

Connecion conn = pds.getConnection(label);
Properties unmatched = ((LabelableConnection)
   connection).getUnmatchedConnectionLabels (label);
```

6.7 Removing a Connection Label in UCP

The `removeConnectionLabel` method is used to remove a label from a connection. This method is used after a labeled connection is borrowed from the connection pool. The following example demonstrates removing a connection label.

```
String pname = "property1";
String pvalue = "value";
Properties label = new Properties();
label.setProperty(pname, pvalue);
Connection conn = pds.getConnection(label);
((LabelableConnection) conn).removeConnectionLabel(pname);
```