Virtual Developer Day

Oracle WebLogic Server 12*c*

Modern, Lightweight Development
with Java EE 6 and Oracle Coherence

ORACLE
TECHNOLOGY NETWORK

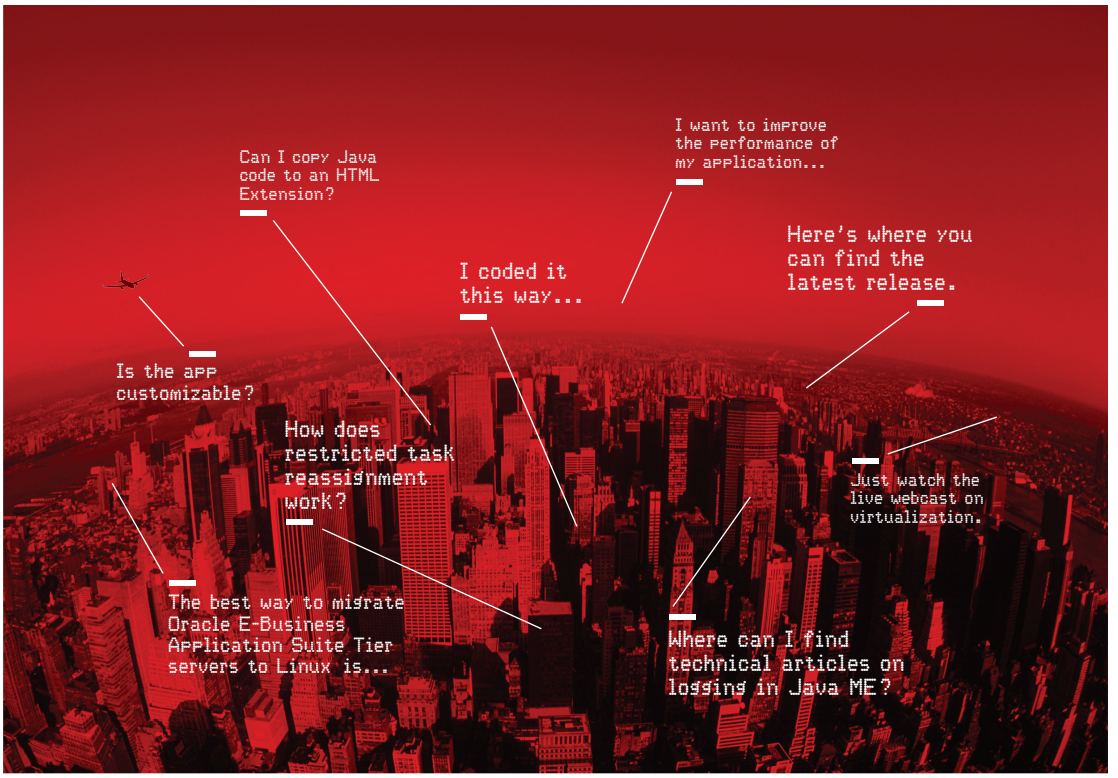
Hands on Lab Manual

Coherence Introduction

ORACLE
TECHNOLOGY NETWORK

http://www.oracle.com/technetwork

ORACLE



Exercise: Your first Coherence-based Java Application

Objective:
Develop a simple Java console-based application to access, update and removing simple types of information from a Coherence clustered cache.

Duration:
30 minutes.

Knowledge:
Unlike Client-Server applications, where client applications typically "connect" and "disconnect" from a server application, Coherence-based Clustered applications simply "ensure" they are in a Cluster, after which they may use the services of the Cluster. More specially, Coherence-based applications typically do not "connect" to a Cluster of applications; they become part of the Cluster.

Useful Tip:
To change the level of logging produced by Coherence, use the following JVM parameter: -Dtangosol.coherence.log.level=*level* where *level* is a number between 0 and 9. The default is 5. A value of 0 means no logging. A value of 9 means extremely verbose. A value of 3 is often useful enough for most application development.

Solution Files
home/oracle/labs/Coh_labs/YourFirstCoherenceApplication.java

Steps
1. Start the default coherence cache server.

cd /labs/wls1211/coherence_3.7
bin/cache-server.sh

2. Start the Coherence Console in a separate terminal window
/labs/wls1211/coherence_3.7/bin
coherence.sh

cache mycache
get message
put message "This is a message in mycache"

3. In the Coherence Java Documentation (javadoc), shipped in the $COHERENCE_HOME$/lib folder, investigate the methods available on the CacheFactory class.

4. Develop a simple Java console application that uses the CacheFactory class to join a cluster (using the ensureCluster method) and then leave the cluster (using the shutdown method).

5. Extend your application to display (using the System.out.println) the Cluster object returned from the “ensureCluster” method.

6. Using javadoc, investigate the methods available on the “NamedCache” interface.

7. Extend your application to use the CacheFactory method “getCache” to acquire a NamedCache for the cache called “mycache”

8. With the NamedCache instance, use the “get” method to retrieve the value for the key “message” (the same key used in the Exercise: Testing a Coherence Installation)

9. Output the value to the standard out using the System.out.println(…); method.

10. Using the running Coherence Shell, change the value of the key “message”. Re run your application to see the changed values.

11. Rerun your application with the JVM parameter “-Dtangosol.coherence.distributed.localstorage=false”. Is the output of the application different from previous executions?

12. Shutdown all of your Cache Server and Coherence Shell instances then rerun your application (with the above JVM parameter set). Is the output different?

Questions:

1. If you change the value of the “message” key in your application (using the “put”) method, is the new value available via the Coherence Shell?

2. What does the JVM parameter -Dtangosol.coherence.distributed.localstorage=false do to a Coherence Java process?

Exercise 2: Caching an Object (using Java Serialization)

Objective:
Create a simple domain object that can be placed into a Coherence Cache.

Duration:
30 minutes.

Prerequisites:
A solid understanding of the rules for Java Serialization.
http://java.sun.com/developer/technicalArticles/Programming/serialization/ has an overview.

Knowledge:
Placing any non-primitive Object into a Coherence Cache requires the said class to be Java Serializable, the reason being, such instances may need to be transported across process boundaries – ie: between Java Virtual Machines, across networks or processes on the same physical machine. The standard way to achieve such transport is to Serialize, transmit and Deserialize the object. Coherence requires Objects placed in a Cache to be Serializable.

Solution Files
home/oracle/labs/Coh_labs/EndOfDayStockSummary.java
home/oracle/labs/Coh_labs/CacheAnObject.java

Steps

1. Create a Java class called EndOfDayStockSummary that implements the java.io.Serializable interface to capture the open, high, low, close and adjusted close prices (in dollars) of a Stock (called a symbol) for a specific date together with the trading volume. Example attribute definitions for the class could be as follows;
private String symbol;
private long date;
private double openPrice;
private double highPrice;
private double lowPrice;
private double closePrice;
private double adjustedClosePrice;
private long volume;

2. Define a method called “getKey()” to return a String that is a combination of the symbol and date attributes.

3. Declare a serialVersionUID.

4. Define a public default constructor – EndOfDayStockSummary()
5. Define a public constructor that accepts values for all of the attributes specified above.

6. Create a console application called CacheAnObject (which contains a main method) that creates an instance of an EndOfDayStockSummary class and places (using the NamedCache put method) the said instance into a Coherence cache called “dist-eodStockSummaries” and then retrieves it to display on the console.

Questions

1. What happens if you forget to implement a default no arguments constructor for the EndOfDayStockSummary class? If you don’t know, comment it out and attempt to run your application again.