Oracle® Insurance Policy Administration

Coherence

version 9
January 2010

Table of Contents

OVERVIEW

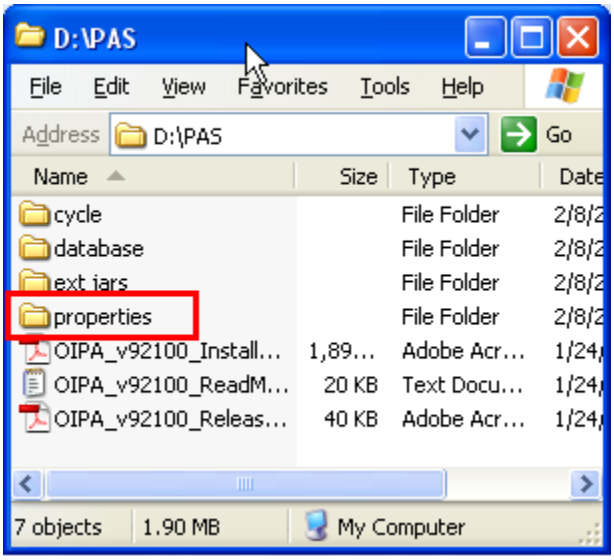
Oracle Coherence provides Oracle Insurance Policy Administration (OIPA) with replicated and distributed (partitioned) data management and caching services on top of a reliable, highly scalable peer-to-peer clustering protocol. Coherence has no single points of failure; it automatically and transparently fails over and redistributes its clustered data management services when a server becomes inoperative or is disconnected from the network. When a new server is added, or when a failed server is restarted, it automatically joins the cluster and Coherence fails back services to it, transparently redistributing the cluster load. Coherence includes network-level fault tolerance features and transparent soft re-start capability to enable servers to self-heal.

There are three different scenarios where OIPA employs Oracle Coherence. They are as follows:

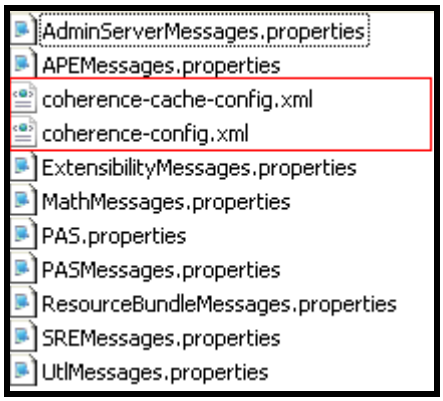
1. Caching
2. Cycle Messaging
3. Application Profiling

LOCATING CONFIGURABLE COHERENCE FILES

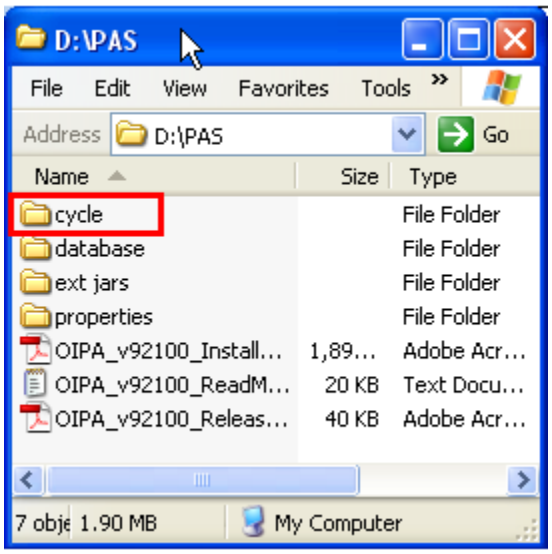
To locate the coherence configuration files for caching and application profiling, open the properties file in the OIPA deployable for eDelivery. For caching in cycle, you will find the files in each one of the cycle sub folders. The two coherence configuration files are called coherence-cach-config.xml and coherenece-config.xml. These files will be explained later in the document.



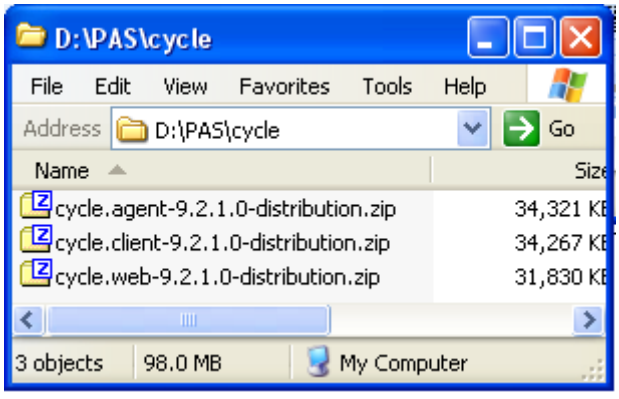
OIPA deployable folder structure



properties folder contents



OIPA deployable folder structure



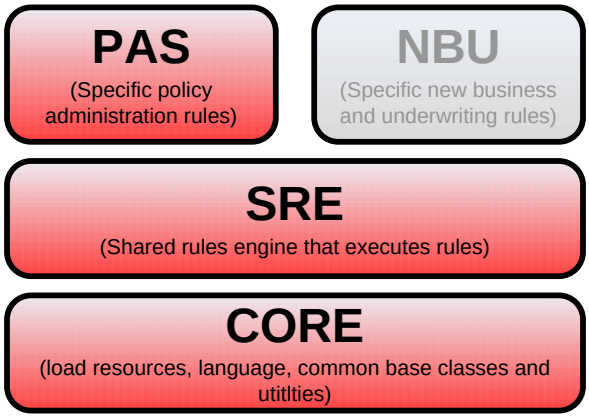
Each cycle zip folder has its own set of coherence files

CACHING

OIPA caches static configuration data and other data that rarely change. There are three cache regions in the system, each for a different module. Every cache region is configured separately, so they can each be configured differently. By implementing this model, the integration of new applications is simplified and only one new region must be added. The core and shared functionality regions will remain the same.

The three cache regions are:

1. region CORE for INSURANCE.SHARED module
2. region SRE for INSURANCE.SRE module
3. region PAS for INSURANCE.PAS module



The following figure demonstrates how caching works in the system:

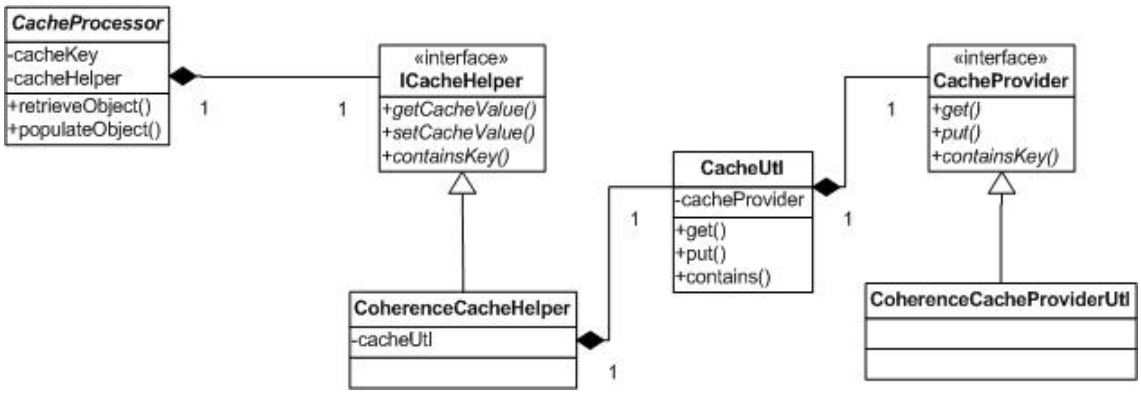


Figure 1 – Cache Diagram

CACHEPROVIDER AND ICACHEHELPER INTERFACES

CacheProvider interface defines all methods any caching provider must implement in order to manage cached objects in OIPA. Coherence is one of the caching providers employed by OIPA.

ICacheHelper interface defines all methods required by caching access in the system, including getting and setting value for a caching key and method checking if the cache contains a given caching key or not.

Using CacheProvider and ICacheHelper interfaces allows OIPA to change actual caching technology that is used for caching without impacting any client code.

CACHEPROCESSOR ABSTRACT CLASS

The abstract class, CacheProcessor, retrieves the object for the caching key from the ICacheHelper in every CacheProcessor instance. An abstract method populateObject() is declared for every CacheProcessor subclass to implement what value object to be put into the cache associated with the caching key. An updateCache() method is also available for use when cached value object needs to be updated for the caching key, such as when the system date is advanced from cycle processing, a new value is updated in the cache.

The following is the pseudo-code of the retrieveObject() provided by CacheProcessor.

```
IF cacheHelper contains cacheKey THEN
get cache value of cacheKey from cacheHelper

ELSE

invoke populateObject() to get cache value
put cache value into cacheHelper associated with cacheKey
```

ACCESSING CACHE

Wherever a piece of data as cache candidate is requested, one anonymous inner class that extends CacheProcessor is defined and instantiated to determine how the value object is created when it is not yet in the cache. The value returned from CacheProcessor retrieveObject() method will be used by client code. It could be either found from the cache region for that module with the caching key, or newly created and put into the cache region associated with the caching key. This is transparent so the difference is not visible to the client code.

The following is example code of caching companyDcl for every companyGuid.

```
String key = "CompanyDcl[CompanyGUID=\"" + companyGuid + "\"]";
CacheProcessor cacheProcessor = new CacheProcessor( cacheHelper, key ) {

     @Override
     protected Object populateObject() {

         CompanyDcl companyDcl = findByPrimaryKey( CompanyDcl.class, companyGuid );
         return companyDcl;
     }
};
return cacheProcessor.retrieveObject();
```

COHERENCECACHEPROVIDERUTL AND CACHEUTL

The utility class CoherenceCacheProviderUtl implements the CacheProvider interface. It enforces convention for interacting with Coherence. Every CoherenceCacheProviderUtl instance gets a cache region for a given name from the Coherence CacheFactory, which creates a clustered NamedCache instance when necessary. The CoherenceCacheProviderUtl object delegates all caching accesses to its Coherence NamedCache object.

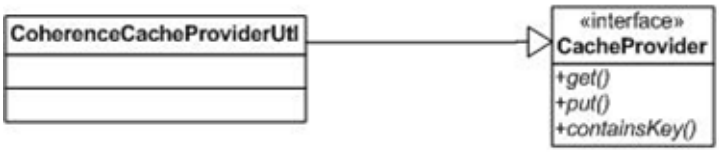


Figure 2 – Coherence CacheProviderUtl

Every instance of utility class CacheUtl wraps a CacheProvider object. CacheUtl also maintains a class level map of mappings from cache region name and cache provider type to a CacheUtl instance. This guarantees only one CacheProvider per cache region per cache provider type.

THREE COHERENCECACHEHELPER CLASSES

CoherenceCacheHelper classes are ICacheHelper implementations that use Coherence for caching. There are three CoherenceCacheHelper classes in packages. They are as follows:

1. com.adminserver.dal.helper
2. com.adminserver.sre.helper
3. com.adminserver.pas.dal.helper

Each class corresponds to cache region CORE, SRE, and PAS respectively.

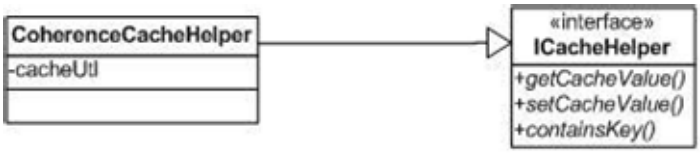


Figure 3 – CoherenceCacheHelper

These three CoherenceCacheHelper classes have mostly identical code, except that each declares its own unique cache region name (i.e., “CORE”, “SRE”, “PAS”, and each register at Spring with a different bean name).

Every CoherenceCacheHelper has its own unique CacheUtl instance, which is mapped from the cache region name for the Coherence cache provider type.

COHERENCE CONFIGURATION FOR CACHING

There are three cache mappings defined in the Coherence caching scheme mapping configuration for the cache name that are “CORE”, “SRE”, “PAS”. The mapped schemes can be different or the same. In the example below you can see each region name identified and an associated scheme map. You may reference the Oracle Coherence documentation on Oracles Technology network for a full listing of available configuration parameters.

Example of Coherence Configuration for Caching in OIPA Version 9

(This file can be found in the properties folder and is named coherence-cache-config.xml)

```
<cache-config>

<!-- ============================ -->
<!-- Map Caches to the NearScheme -->
<!-- ============================ -->
  <caching-scheme-mapping>
     <cache-mapping>
        <cache-name>CORE</cache-name>
        <scheme-name>SampleNearScheme</scheme-name>
     </cache-mapping>
     <cache-mapping>
        <cache-name>PAS</cache-name>
        <scheme-name>SampleNearScheme</scheme-name>
     </cache-mapping>
     <cache-mapping>
        <cache-name>SRE</cache-name>
        <scheme-name>SampleNearScheme</scheme-name>
     </cache-mapping>
     <cache-mapping>
        <cache-name>Cycle</cache-name>
        <scheme-name>SampleNearScheme</scheme-name>
     </cache-mapping>
     <cache-mapping>
        <cache-name>cacheForAsProfiler</cache-name>
        <scheme-name>SampleLimitedPartitionedScheme</scheme-name>
     </cache-mapping>
  </caching-scheme-mapping>


  <caching-schemes>

     <!-- ====================== -->
     <!-- Local In-memory Cache -->
     <!-- ====================== -->
        <local-scheme>
           <scheme-name>SampleMemoryScheme</scheme-name>
        </local-scheme>

     <!-- ================================== -->
     <!-- Size limited Local In-memory Cache -->
     <!-- ================================== -->
        <local-scheme>
           <scheme-name>SampleMemoryLimitedScheme</scheme-name>
           <low-units>1000</low-units>
           <high-units>5000</high-units>
        </local-scheme>


     <!-- ============================ -->
     <!-- Distributed In-memory Cache -->
     <!-- ============================ -->
```

```
        <distributed-scheme>
           <scheme-name>SamplePartitionedScheme</scheme-name>
           <backing-map-scheme>
              <local-scheme>
                 <scheme-ref>SampleMemoryScheme</scheme-ref>
              </local-scheme>
           </backing-map-scheme>
        </distributed-scheme>


     <!-- ================================== -->
     <!-- Distributed Limited In-memory Cache -->
     <!-- ================================== -->
        <distributed-scheme>
           <scheme-name>SampleLimitedPartitionedScheme</scheme-name>
           <backing-map-scheme>
              <local-scheme>
                 <scheme-ref>SampleMemoryLimitedScheme</scheme-ref>
              </local-scheme>
           </backing-map-scheme>
        </distributed-scheme>

     <!-- ======================== -->
     <!-- Cache Cluster Definition -->
     <!-- ======================== -->
        <near-scheme>
          <scheme-name>SampleNearScheme</scheme-name>
             <front-scheme>
                <local-scheme>
                   <scheme-ref>SampleMemoryLimitedScheme</scheme-ref>
                </local-scheme>
             </front-scheme>
             <back-scheme>
                <distributed-scheme>
                   <scheme-ref>SamplePartitionedScheme</scheme-ref>
                </distributed-scheme>
             </back-scheme>
        </near-scheme>


     <!-- ================ -->
     <!-- Cycle Invocation -->
     <!-- ================ -->
        <invocation-scheme>
           <scheme-name>CycleMessagingService</scheme-name>
           <service-name>InvocationService</service-name>
           <autostart>true</autostart>
        </invocation-scheme>

  </caching-schemes>

</cache-config>
```

CYCLE MESSAGING

OIPA provides a subsystem for batch processing for insurance transactions called Cycle. Cycle is a high performance distributed subsystem designed to process as many pending transactions as possible in the shortest amount of time. The cycle subsystem drives processing of transactions through a cycle group, which is comprised of a set of cycle agents.

Each cycle agent exists in its own JVM process. Cycle agents may be spread across multiple machines. The cycle agents in the same cycle group communicate with each other in order to coordinate cycle processing by sending cycle messages to each other.

When configuring a cycle client or agent, the coherence configuration file must be defined at runtime. The script that is used to start cycle as a service, startup.sh or startup.bat, must be modified to include the correct absolute path - Example: `"JAVA_OPTS –DrootDir=. -Dtangosol.coherence.override=/opt/oracle/oipa/properties/coherence-config.xml"`

Please see the OIPA Cycle documentation found on the Insurance Documentation section of Oracle's Technology Network for more information.

OIPA cycle uses both coherence invocation service and coherence cache for cycle group member communications. Cycle messages are distributed through a coherence invocation service. Cycle group state information is shared among cycle agents in the same cycle group by using a coherence cache region which is called "Cycle" in the system. Therefore, it is required that all cycle agents in the same cycle group be configured in the same coherence cluster.

CYCLEMESSAGETYPE AND CYCLEMESSAGE

Enumeration CycleMessageType defines the type of cycle messages in OIPA. There are five different types of cycle messages. The five types are as follows:

1. InitializeGroupProcessing
2. MemberCommand
3. MemberStatus
4. GroupCommand
5. GroupStatus

Every CycleMessage object has information identifying the type of cycle message as well as details about the sending cycle agent, including identification, status and other necessary information such as timestamp, etc.

MESSAGEDISPATCHER INVOCABLE

Coherence invocable objects are cluster-portable objects that can be invoked on any set of remote agents. When an invocable object is received by the invocation service on an agent, the invocation service deserializes the object, executes its init() method, then invokes it by executing its run() method.

The cycle subsystem defines an invocable implementation in class MessageDispatcher. Every MessageDispatcher instance carries one cycle message, and its execution dispatches the cycle message to all message listeners registered to the local cycle group agent for the corresponding message type.

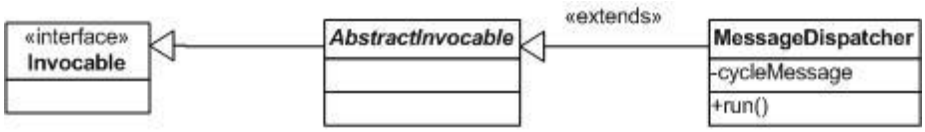


Figure 4 – Coherence Invocable

The following is the implementation of MessageDispatcher run() method.

```
BaseCycleGroupConnection cycleGroupConnection =
                                                    SpringBeanUtl.getBeanByName(
"cycleGroupConnection" );
cycleGroupConnection.dispatchLocalCycleMessage( cycleMessage );
```

CycleGroupDcl and Cycle Cache Region

CycleGroupDcl class defines information about the cycle group, such as current cycle group status, current cycle type, information of the current controller cycle agent, and all cycle agents identified by member ID's, etc. Group information in CycleGroupDcl is shared across all agents in the group through Coherence cache using a fixed string caching key.

Cycle defines its own cache region, named "Cycle", where cycle agents share cycle group information. Any group information update made by a member, including the group controller, is ported to the other remote members by Coherence.

The cycle cache region does not have to be configured differently in the Coherence caching scheme mapping from the other caching regions in the system, though it can be configured differently if necessary.

Initialization

Both the Coherence cache and invocation service are initialized in the connect() method class CoherenceCycleGroupConnection, which is executed when a cycle agent connects to a cycle group.

Coherence Configuration for Cycle Invocation

The following is an example of Coherence configuration for cycle invocation.

```
<cache-config>
    ……
<caching-schemes>
    ……
    <invocation-scheme>
        <scheme-name>CycleMessagingService</scheme-name>
        <service-name>InvocationService</service-name>
        <autostart>true</autostart>
    </invocation-scheme>
</caching-schemes>
</cache-config>
```

APPLICATION PROFILING

OIPA provides a profiling subsystem that gathers information for the purposes of providing insights into system execution. The OIPA profiling subsystem sends lifecycle events of request processing and transaction processing to registered listeners. Profiling event information can be used for performance analysis and improvement, system optimization, auditing and reports.

OIPA provides one profiling listener implementation that uses Coherence cache to pass profiling event information to a separated profiling application that runs in the same Coherence cluster.

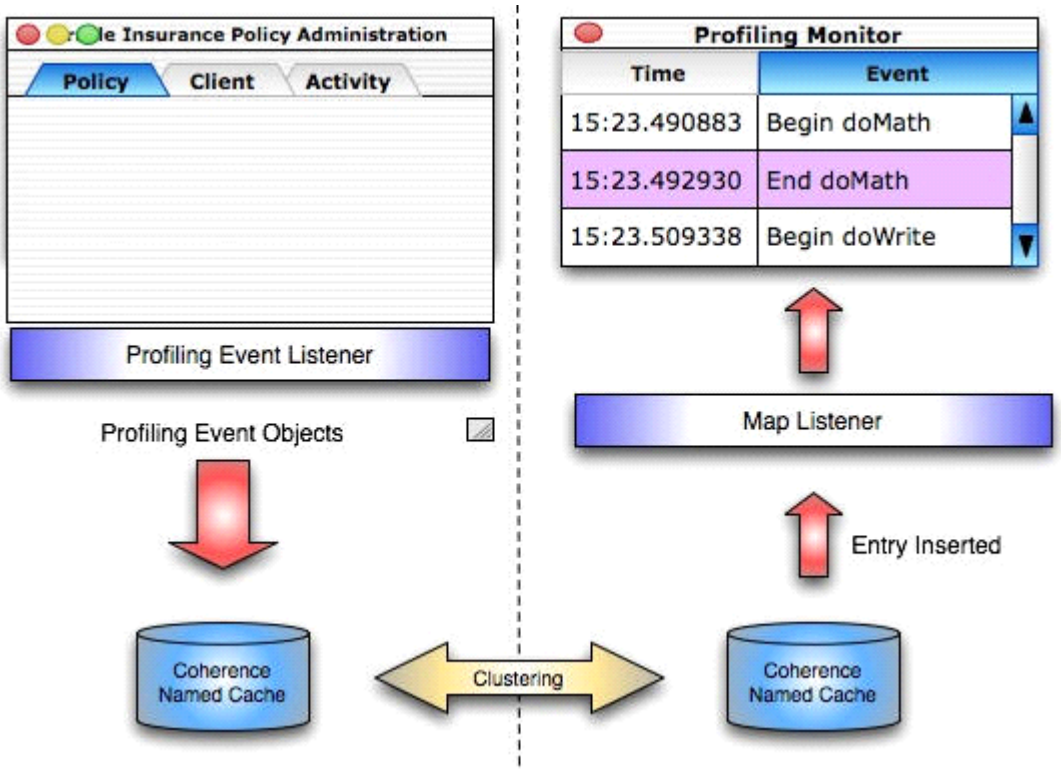


Figure 5 – Profiling Diagram

PROFILINGEVENTCACHEBLL AND PROFILINGEVENTCACHEENTRYKEY

Class ProfilingEventCacheBll provides methods to save a profiling event object or a list of profiling event objects into a Coherence NamedCache. Every cache entry of an event object or an event object list is associated with a caching key of ProfilingEventCacheEntryKey instance, which has a time stamp and a string UUID.

Right now the provided Coherence based profiling listener implementation saves screen profiling events individually, and saves lists of transaction profiling events at a fixed one second interval.

PROFILINGEVENTCACHEHELPER

ProfilingEventCacheHelper extends ProfilingEventCacheBll. It creates the Coherence NamedCache with a default cache region name “cacheForAsProfiler”. All access to this Coherence cache in the provided profiling listener are through ProfilingEventCacheHelper.

COHERENCE CONFIGURATION FOR PROFILING EVENT CACHE

The Coherence cache configuration for OIPA profiling is similar to that for static data caching. It can be configured differently if necessary.

EXTERNAL PROFILING APPLICATION

An external profiling application is not part of OIPA. In order to access information from the profiling event listener through Coherence cache, external profiling application must be in the same Coherence cluster and use the same cache name. Typically a map listener is configured to be invoked whenever a new cache entry is inserted that has a profiling event object, or a list of profiling event objects, so the profiling event data can be captured for the monitoring application to use.

COHERENCE CLUSTER CONFIGURATION

Coherence cluster configuration defines all the nodes in the cluster, along with a pointer to the cache factory configuration file. This is necessary in order for coherence to run. During the configuring of the application server, an argument must be passed to the JVM to define the location of this coherence configuration file, coherence-config.xml. The argument to be passed is “tangosol.coherence.over”, along with the absolute path to the configuration file.

Example: “`-Dtangosol.coherence.override=/opt/oracle/oipa/properties/coherence-config.xml`”

Sample Coherence Cluster Configuration

(This file can be found in the properties folder and is named `coherence-cache-config.xml`.)

```
<coherence>
   <cluster-config>
      <unicast-listener>
            <well-known-addresses>
                  <socket-address id="1">
                     <address>localhost</address>
                     <port>8080</port>
                  </socket-address>
            </well-known-addresses>
      </unicast-listener>
   </cluster-config>
   <logging-config>

          <severity-level>5</severity-level>
       </logging-config>
   <configurable-cache-factory-config>
         <init-params>
            <init-param>
               <param-type>java.lang.String</param-type>
               <param-value>/opt/oracle/oipa/properties/coherence-cache-
config.xml</param-value>
            </init-param>
         </init-params>
   </configurable-cache-factory-config>
</coherence>
```