D DOCUMENTO

Programación en Cividas

Referencias Remotas

DIRIGIDO A

Desarrolladores

FECHA

10 de Enero de 2012

Referencias Remotas

Inscripción Registro Mercantil de Pontevedra, Tomo 3116, Libro 3116, Folio 30, Hoja PO-38276 – C.I.F.: B-36.499.960

imatia
your innovation partner

Índice

Qué es una Referencia Remota

Ontimize dispone desde la versión 5.2053 de una nueva utilidad que permite al programador obtener referencias a objetos remotos (referencias remotas) desde el cliente configurando los parámetros de estos objetos en un fichero XML. En dicho fichero se podrán establecer los parámetros que se pasarán al objeto en el constructor.

Implementación de una Referencia Remota

Clase Base

La clase Base de la que deben extender las referencias remotas es:

com.cividas.server.utilsAbstractRemoteReferenceReadyToListen

Las restricciones que deben cumplir las clases que se deseen configurar mediante esta utilidad son:

1. Deben poseer un constructor público con los siguientes parámetros
   public MiReferenciaRemota (int port, EntityReferenceLocator locator, Hashtable params) throws Exception;
   donde:
   a. port: Entero que indica el puerto donde estará el objeto remoto escuchando (normalmente el mismo que el del servidor)
   b. locator: Buscador de referencias de entidades
   c. params: Hashtable que contiene los diferentes parámetros definidos el en fichero remotereferences.xml para la referencia remota .
2. Deben implementar el siguiente método:
   public Vector getCividasParametersToCheck();
   Este método devuelve un vector con el nombre de los parámetros sobre los que se creará un escuchador de posibles cambios de valor, es decir, si desde la pantalla de administración de parámetros se realiza un cambio de valor de alguno de ellos ,este cambio se verá reflejado en caliente en la referencia remota.

Captura de parámetros

Para configurar la referencia remota con parámetros de Base de Datos (Cividas parameters) se debe sobreescribir y dar contenido al método

public void initConfiguration(Hashtable params) throws Exception;

Un ejemplo de uso sería:

```
public void initConfigurationSimple(Hashtable params){
      Vector cividasConfigurationSet = getCividasParametersToCheck();
      try {
            CividasConfigurationValueSet cvSet =
                  ((ICividasConfiguration)locator).getCividasConfigurationValueSet(
                                          cividasConfigurationSet,
                                          locator.getSessionId());
            if (cvSet != null) {
                  Object currentParameter =
                              cvSet.getItemValue(CividasParameter.MI_PARAMETRO_1);
                  if (currentParameter == null)
                        currentParameter = params.get(CividasParameter. MI_PARAMETRO_1);
                  if (currentParameter != null) {
                        miparametro1 = (String) currentParameter;
                        CividasMessage.writeDebug(this,"MI_PARAMETRO_1:" + miparametro1);
                  }
                  else {
                        CividasMessage.writeDebug(this, "NO_EXISTE_ MI_PARAMETRO_1");
                  }
            }

      } catch (Exception e) {
            e.printStackTrace();
      }
}
```

En este ejemplo se puede ver como se consulta un conjunto de Cividas parameters (el vector CividasConfigurationSet), en este caso se decide dar prioridad al valor del Cividas parameter frente al que pudiese estar definido en el xml de definición remotereferences.xml, que son los recibidos en el parámetro de entrada (Hashtable params).

Referenciación de una Referencia Remota

Una vez configurada la utilidad se podrán obtener las referencias a todos los objetos indicados utilizando el método:

getRemoteReference(String name, int sessionId)

disponible en la interfaz *UtilReferenceLocator* implementada por el buscador de referencias local.

Por ejemplo:

```
try {
   ((UtilReferenceLocator) reflocator).getRemoteReference("ReferenceName", sessionId);
} catch (Exception e1) {
    e1.printStackTrace();
}
```

Definición de la Referencia Remota

A la hora de definir una referencia remota existen dos posibilidades:

Definición estándar

En primer lugar se debe configurar el fichero xml en el que se va a especificar la clase del objeto remoto:

```
<RemoteReferences>
      <RemoteReference>
            <Name>ReferenceName</Name>
            <Class>com.package.DefaultClassName</Class>
            <Param>
                  <Name>param1</Name>
                  <Value>value1</Value>
            </Param>
            <Param>
                  <Name>param2</Name>
                  <Value>value2</Value>
            </Param>
      </RemoteReference>
</RemoteReferences>
```

En este fichero se indica:

- **Name**: Nombre que se utilizará en la aplicación para pedir la referencia.
- **Class:** Clase cuya referencia desea obtenerse (paquete + clase).
- **Param:** Cada uno de los parámetros a introducir en el Hashtable que se pasa a la clase en su constructor. Para cada parámetro debe especificarse la clave que lo identifica en el Hashtable (<Name>) y su valor (<Value>).

Por cada objeto remoto a configurar se deberá añadir una entrada en el fichero con la estructura indicada:

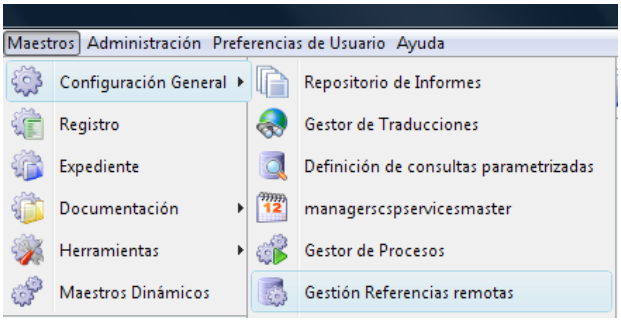
<RemoteReference> . . . </RemoteReference>

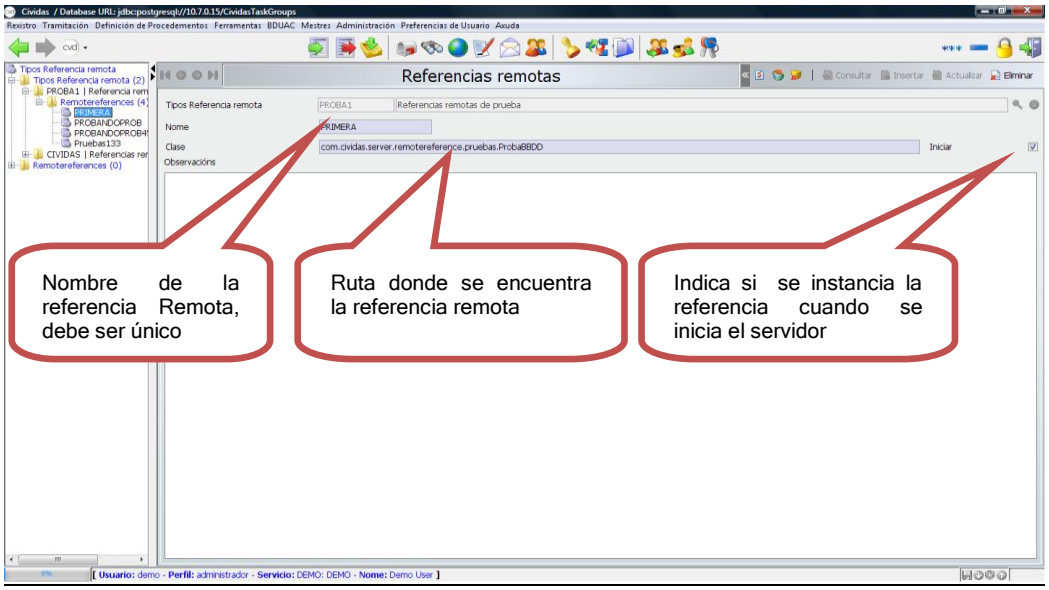
Para establecer cuál es el fichero xml de configuración se especifica en el properties del buscador de referencias (**locator.properties)** la propiedad *RemoteReferences*, por ejemplo:

```
RemoteReferences=com/ontimize/miaplicacion/server/conf/remotereferences.xml
```

Definición en base de datos

Existe en la aplicación un maestro donde dar de alta referencias remotas, dentro del menú Maestros, configuración general, Gestión Referencias remotas:





Cabe destacar que para las referencias remotas definidas de esta forma (en base de datos), los parámetros que necesite utilizar los debe conseguir de los cividas parameters. Para ello se utiliza el método explicado en el apartado 2.1:

public void initConfiguration(Hashtable params) throws Exception

Anexo I .- Buenas prácticas:

A la hora de definir las referencias remotas en el fichero de configuración debe tenerse en cuenta que el nombre indicado en el parámetro *Name* debe ser único y se utilizará como clave para solicitar la referencia al objeto, en este punto cabe destacar que tampoco este nombre podrá coincidir con las referencias remotas que estén definidas en Base de Datos:

```
<RemoteReference>
      <Name>ReferenceName</Name>
      …
</RemoteReference
```

Por lo que se recomienda definir dichos valores en nuestra aplicación de la forma:

```
public static final String REMOTE_OBJECT = "ReferenceName";
```

Por ejemplo:

```
public static final String GIS_OBJECT = "GIS_Reference";
public static final String WORKFLOW_OBJECT = "WF_Reference";
```

Sería bueno también que la referencia remota implementase una interfaz remota, y que se hiciese el cast a esta interfaz a la hora de coger la referencia mediante el método antes indicado.

Ejemplo:

```
MiInterfazRemota referenciaRemota=(MiInterfazRemota)((UtilReferenceLocator)
                                    reflocator).getRemoteReference("ReferenceName", sessionId);
```

Un ejemplo de cómo se tendrían que manejar las posibles conexiones que precisasen los métodos de la referencia remota sería el siguiente:

```
public EntityResult metodoSinConexion(Object requestID, int sessionID) throws Exception{
      EntityResult result = new EntityResult();
      Connection con = null;
      OntimizeConnection conexion = null;
      try { // We are going to do all of this through an unique conection
            conexion = ((RemoteReferenceLocator) locator).getConnectionManager().
                                                      getOntimizeConnection();
            con = conexion.getConnection();
            con.setAutoCommit(false);
            result = metodoConConexion(requestID, sessionID, con);
            con.commit(); // Finally we execute all the database operations
      }
```

```
        } catch (Exception e) {
                e.printStackTrace();
                try {
                        conexion.getConnection().rollback();
                } catch (SQLException sqle) {
                        sqle.printStackTrace();
                        result.setCode(EntityResult.CONTENT_WRONG);
                        result.setMessage(sqle.getMessage());
                        return result;
                }
                result.setCode(EntityResult.CONTENT_WRONG);
                result.setMessage(e.getMessage());
                return result;
          } finally {
                try {
                        conexion.getConnection().setAutoCommit(true);
                } catch (SQLException e) {
                        e.printStackTrace();
                        result.setCode(EntityResult.CONTENT_WRONG);
                        result.setMessage(e.getMessage());
                }
                disconnect(conexion);
          }
          return result;
    }
```

Cabe destacar que la llamada al método de desconexión '**disconnect(conexion)**;' se realiza en la cláusula **finally{ }** para que sea llamado siempre, evitando así dejar conexiones abiertas contra la base de datos.