Java Development Techniques for Improving Performance of EP iViews

Will Carlton
US NetWeaver Regional Implementation Group, SAP Labs LLC

Learning Objectives

As a result of this workshop, you will be able to:

- **Utilize advanced caching techniques to improve performance of portal applications**
- **Utilize connection pooling techniques to improve resource utilization for R/3**
- **Utilize the new JCA based Connection Framework in EP6 to access EIS applications such as R/3 and Databases**

Agenda

Overview

Data Object Caching

JCO Connection Pooling

JCA Connector Framework

Overview

The intent of this workshop is to provide some tips for improving the performance of Java based content for the Enterprise Portal in high volume situations.

Due to the limited time and desire to provide some hands on experience to participants, we are not covering the basic topics for JCO programming, JDBC programming, etc. Please refer to the PDK documentation or the Java website for tutorials or other such information.

The differences between EP5 and EP6 will be mentioned as we work through examples so that you are aware of any coding changes that need to be considered.

We'll also talk about new functionality in EP6 that is NOT available in EP5. You will need an EP6 Portal to try these examples.

Agenda

Overview

Data Object Caching

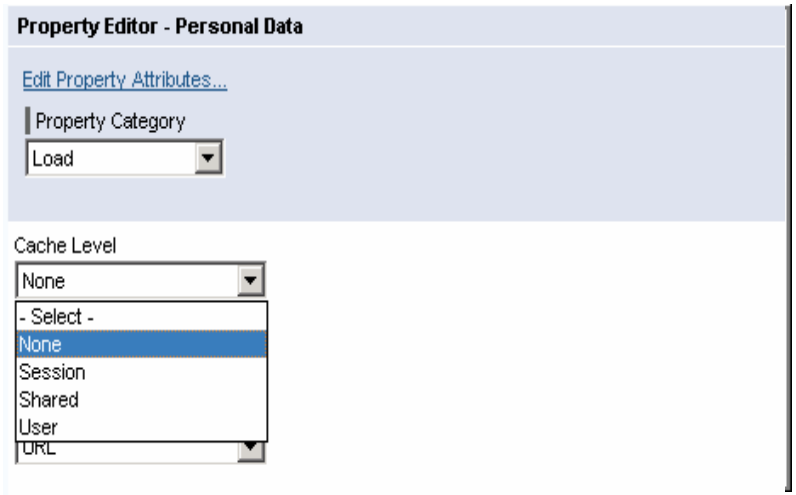
JCO Connection Pooling

JCA Connector Framework

Data Object Caching

- **The Enterprise Portal provides a mechanism to cache content from iViews in order to improve performance**
- **Rendered content can be cached by the content administrator setting various caching options for individual iViews for EP5:**

  And EP6:



- **The portal runtime engine uses the caching service to store copies of rendered content**
- **Data objects can also be cached by the caching service using the Cache API**
- **The ability to cache data objects is particularly useful when you need to perform multiple operations on a data source by many users, but the backend data source does not change frequently**

Using the Cache Service API

Import statement

```
import com.sapportals.portal.prt.service.cache.ICacheService;
```

```
import com.sapportals.portal.prt.service.cache.ICacheService;
…
//EP5
ICacheService cachesvc = (ICacheService)
                request.getService(ICacheService.KEY);

//EP6
ICacheService cachesvc = (ICacheService)
               PortalRuntime.getRunTimeResources().
               getService(ICacheService.KEY);
```

For EP5, the JAR file for the service is cacheapi.jar

For EP6, there is a compatibility JAR called com.sap.portal.compatibility50.cacheapi.jar

Cache API - Keys

- **Cached objects are identified by a unique key**
- **Keys must be formulated with a user's context as part of name, or an application context if shared among users**

```
String key ;
key = "myKey" + request.getUser().getUserId();  // EP5

key = "myKey" + request.getUser().getLogonUid(); // EP6
```

Cache API – Retrieving an Object

- **Cached objects are retrieved using the `get()` method**

```
public java.lang.Object get(java.lang.String key)
```

Returns:
the entry corresponding to the key or null if no entry was stored in the cache under this key (or if the object stored was null)

- **The object must be cast to the proper data type (use instanceof() or other check to avoid ClassCastExceptions**

```
MySpecialObject obj = null;
try {
      obj = (MySpecialObject) cachesvc.get(key);
} catch (Exception ex) {
      response.write ("Exception retrieving object: " +
                  ex.getLocalizedMessage());
}
```

Cache API – Storing an Object

- **Cached objects are stored using the `put()` method**

```
public java.lang.String put(java.lang.String key,
                            java.lang.Object object,
                            boolean forced)
                     throws CacheException
```

The put() method is overloaded and the only required parameter is the object to store. It returns the key used to store the object

```
boolean stored = false;
try {
      cachesvc.put(key, obj, true);
      stored = true;
}
catch (CacheException ex) {
      response.write("Exception storing object: " +
            ex.getLocalizedMessage());
}
```

Cache API – Session Cache Example

It is possible to store the cache key in a user's HTTP / Component Session.

Put the object into the cache without a key. The put() method will return the GUID type key that was used to store the object.

Store this in the component session so you can retrieve it on subsequent requests.

```
boolean stored = false;
String key;
try {
   key = cachesvc.put(obj);
   request.getComponentSession().putValue("cachekey",key);
   stored = true;
}
catch (CacheException ex) {
   response.write("Exception storing object: " +
      ex.getLocalizedMessage());
}
```

Cache API – Setting Timeout for an Object

The validity time for an object is controlled by setting a timeout period in ms on the cached object after storing it using the setTimeout() method

```
public boolean setTimeout(java.lang.String key,
                          long timeout,
                          int actionAfterTimeout,
                          boolean useFixedTimeout)
```

timeout: the object is kept in the cache at least during the timeout. Specified in ms.

actionAfterTimeout:
ICacheEntryManagement.INVALIDATE_AFTER_TIMEOUT: invalidate the entry after timeout
ICacheEntryManagement.UPDATE_AFTER_TIMEOUT: update the entry after timeout if applicable (the corresponding object stored in the cache must implement the **IUpdateable** interface).
ICacheEntryManagement.FREE_AFTER_TIMEOUT: keep the entry in the cache but without timeout anymore (thus the object can disappear from cache).

useFixedTimeout: If true, the object is kept in the cache for the given time regardless of the access to this object in the cache. If false, the object is invalidated if it has not been accessed during the given timeout.

Cache API – Removing an Object

- **Cached objects are deleted using the `invalidate()` method**

```
public void invalidate(java.lang.String key)
```

```
try {
      cachesvc.invalidate(key);
} catch (Exception ex) {
      response.write ("Exception removing object: " +
                  ex.getLocalizedMessage());
}
```

Agenda

Overview

Data Object Caching

JCO Connection Pooling

Database Connection Pooling

JCA Connector Framework

Why Connection Pooling?

- **Connection setup time for R/3 connections can be substantial in high volume portal environments**
- **Performance of JCO applications can be improved by reducing the connection setup time via connection pools**
- **The jcoclient service (EP 5.0 only) provides an efficient connection pooling mechanism to manage simultaneous connections to R/3 applications**
- **Connection pooling is handled automatically in EP6 by using the JCA based SAP Connector and the Connection Framework – please do NOT use jcoclient in EP6 for new components**

JCO Client Service (EP 5.0)

- The jcoclient service provides JCo clients to iViews
- JCo clients are obtained by accessing a pool of clients using one of the `getJCOClientPoolEntry()` methods
- Each time one of the getJCOClientPoolEntry methods is called, a new `JCOClientPoolEntry` is returned
- After obtaining the pool entry, the application extracts the contained client by calling `getClient()`.
- After calling all the necessary function modules, the client must be released to the pool by calling `release()` on the pool entry which makes it available for subsequent calls of getJCOClientPoolEntry
- The jcoclient service might throw a TimeoutException if no client is available

JCO Connection Pooling (EP 5.0)

- **The jcoclient service:**
  - **maintains one pool for each combination of user, JCo destination and HTTP session.**
  - **creates different pools for the same user if this user accesses different JCo destinations and/or is logged in from different user agents, i.e. with different HTTP sessions.**
  - **Creates clients with the user information obtained from the portal component request.**
- **JCo clients can be reused**
- **The pool keeps track of JCo clients that are in use and disconnects them if there is an HTTP session timeout**
- **The number of connections to a destination can be limited per user**
- **Clients contained in the pool are subject to a timeout, keeping the number of unused connections small**

JCO Connection Pooling – Getting a Client

```
IJCOClientService clientSvc =
    (IJCOClientService)request.getService(IJCOClientService.KEY);
IJCOClientPoolEntry poolEntry = null;
try
{
    poolEntry = clientSvc.getJCOClientPoolEntry("SAP_HR", request);
}
catch (TimeoutException e)
{
    response.write("No connection available. Please try again.");
    return;
}
catch (Exception e)
{
    response.write("Error obtaining connection: " +
                e.getLocalizedMessage());
    return;
}
JCO.Client client;
client = poolEntry.getClient();
// Use the client as usual to make R/3 calls
```

Returning the connection to the pool

After using a JCo client, it should be returned to the pool to make it available for further use by calling release() on the pool entry.

```
poolEntry =
clientService.getJCOClientPoolEntry("SAP_HR", request);
…
client = poolEntry.getClient();
client.execute(myFunction); // example call
// process the result of the call
…
// release the client back to the pool
poolEntry.release();
```

Handling Exceptions with Pooling

- If an error occurs when using a connection, you should NOT return it to the pool
- Use the `delete()` method to remove it from the pool to prevent further use by other applications.

```
poolEntry = …;
…
client = poolEntry.getClient();
try {
      client.execute(myFunction); // example call
      …
      poolEntry.release();
} catch (JCO.Exception ex) {
      // handle error, then delete the pool entry
      poolEntry.delete();
}
```

Configuration of the Pool Service

- If an error occurs when using a connection, you should NOT return it to the pool
- Use the `delete()` method to remove it from the pool to prevent further use by other applications.

Additional Information

- **See the javadoc in the 5.0 PDK for JCO and jcoclient service**
- **Michael Hermann has written an excellent SDN article on JCO Connection Pooling. See the article at http://www.sdn.sap.com/. Look for "JCO Client Service".**
- **Article on EP6 compatibility service on iViewStudio (soon to be on SDN):**
  http://www.iviewstudio.com/main/sub_content/pdk/jcoclientservicetutorial_ep60SP3.html
- **Use the JCA based SAP Connector for new EP6 development!**

Agenda

Overview

Data Object Caching

JCO Connection Pooling

JCA Connector Framework

SAP Connector Framework Overview

SAP provides out-of-the box connectors for business applications and databases (SAP Connector, JDBC Connector)

- **All are based on the Connector Framework**
- **SAP Connector is based on the SAP Java Connector (JCo)**

The Connector Framework provides the portal with the ability to integrate enterprise applications by retrieving data from these applications

The Connector Framework is a generic framework based on the JCA standard with extended capabilities

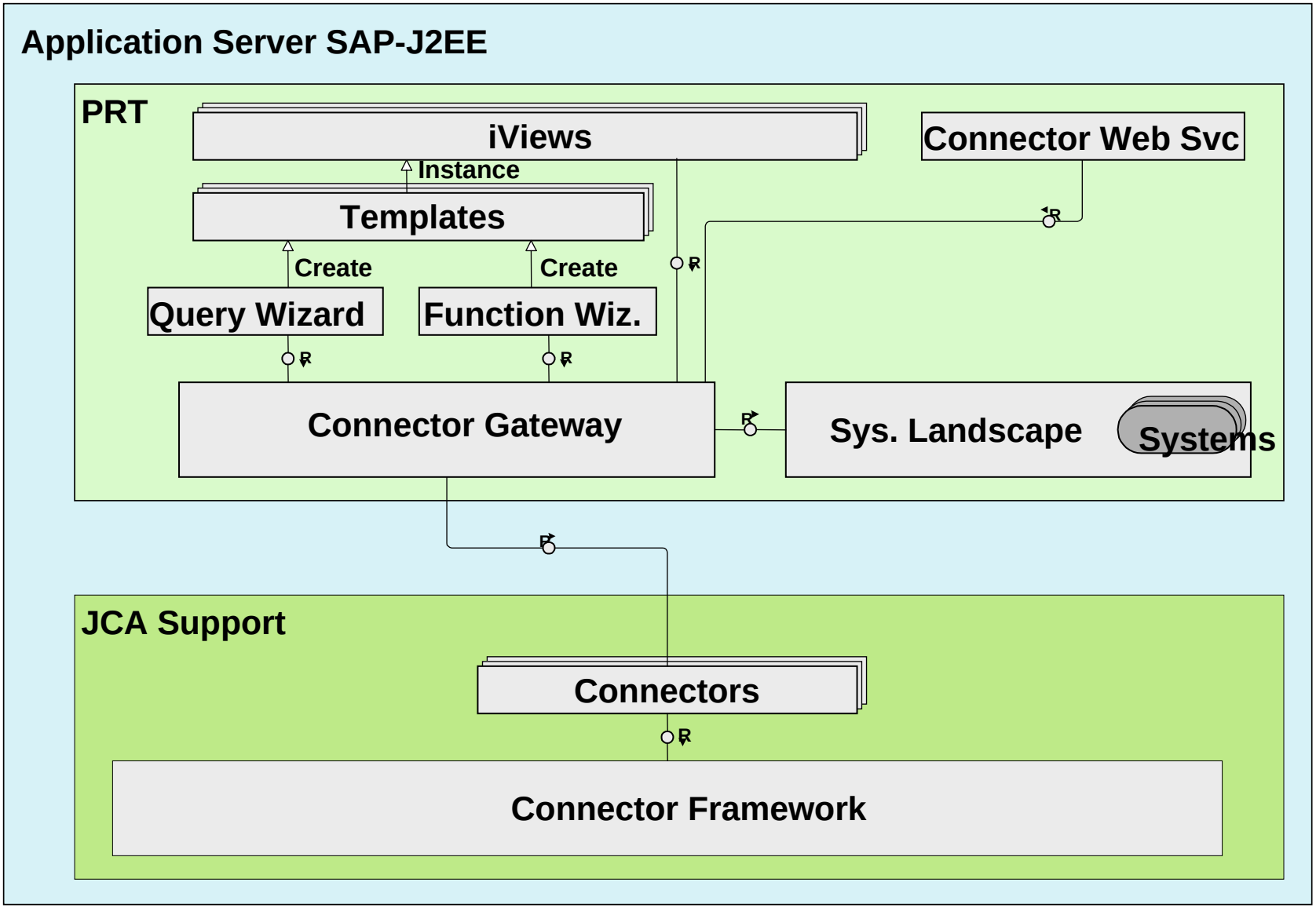
Connectors run on the SAP J2EE Engine and do not require the PRT or portal platform

SAP Connector Framework Overview (cont.)

There are different scenarios of how to use the Connector Framework:

- **Create a new iView based on the Connector Wizard**
- **Create a Portal Application that accesses Connector via the Portal Service connector service**
- **Implement a new Connector using the Eclipse Wizard (PDK)**
- **Connectors can be called as Web Services**

SAP J2EE Architecture



Using the Connector Framework (CF) API

CF Basics for a simple Query

Use the Connector Gateway Service to obtain connections to backend systems

Use the ConnectionProperties object to facilitate SSO

Obtain a connection to the back-end system

Create a Query object to describe the query you wish to execute on the back-end system

Execute the Query and check return status

Obtain the output – usually returns an IRecordset object

Iterate through recordset and deal with data

CF API – Required Imports and Libraries

Using the Connector Framework requires you to import several packages and add certain libraries to your Eclipse project

Import statements:

```
import com.sapportals.portal.ivs.cg.ConnectionProperties;
import com.sapportals.portal.ivs.cg.IConnectorGatewayService;
import com.sapportals.portal.ivs.cg.IConnectorService;

import com.sapportals.connector.connection.IConnection;
```

Libraries required in Eclipse Project:

```
<J2EE>/additional-lib/activation.jar
<J2EE>/additional-lib/connector.jar
<J2EE>/additional-lib/com/sapportals/connectorframework/framework/GenericConnector.jar
<J2EE>/additional-lib/com/sapportals/connectorframework/framework/ConnectorHelper.jar
<IRJ>/WEB-INF/portalapps/com.sap.portal.ivs.connectorserviceapi.jar
<J2EE>/additional-lib/jta.jar
```

CF API – Getting a Connection

Use the Connector Gateway Service to obtain connections to back-end systems

```
import com.sapportals.portal.ivs.cg.ConnectionProperties;
import com.sapportals.portal.ivs.cg.IConnectorGatewayService;
import com.sapportals.portal.ivs.cg.IConnectorService;
import com.sapportals.connector.connection.IConnection;
....
IConnectorGatewayService cgService =
       (IConnectorGatewayService)
       PortalRuntime.getRuntimeResources().
       getService(IConnectorGatewayService.KEY);
```

CF API – Getting a Connection (cont.)

Use the ConnectionProperties object to fill in the required connection parameters for SSO

The Locale is required for language texts, etc., that may be dependent on the user

The user is required to obtain user mapping info or logon ticket for back-end system

```
import com.sapportals.portal.ivs.cg.ConnectionProperties;

...
ConnectionProperties cp = new
             ConnectionProperties(request.getLocale(),
                                  request.getUser());
...
```

CF API – Getting a Connection (cont.)

Use the getConnection() method of the Gateway service to obtain the actual connection to the back-end

There are several ways to obtain the connection. This version requires the System Alias and a ConnectionProperties object

IMPORTANT: Make sure you close connection when you are done! Failure to do so will cause you to run out of connections due to connection pooling.

```
...
ConnectionProperties cp = new
              ConnectionProperties(request.getLocale(),
                                   request.getUser());

connection = cgService.getConnection("Pubs",cp);
...
// do some stuff
connection.close(); // IMPORTANT!!! MAKE SURE YOU CLOSE CONN.
```

System Alias

Connection Properties

CF API – Queries

A Query can be used for simple interactions with back-end systems

- Some resource adapters may not provide a query interface
- A query is usually the easiest way to interact with a RDBMS, however

You build a query using an IQuery object

- Obtained from the Connection object

Execution of the query will result in boolean success/failure flag

- The result of the query (IRecordset) can be obtained if successful

```
import com.sapportals.connector.execution.objects.IQuery;
...
IQuery query = connnection.newQuery();
String qstr = "SELECT title_id,title,price FROM titles " +
       "WHERE title_id LIKE 'P%'");
boolean success = query.execute(qstr);
...
```

CF API – Using the Data

The IRecordSet is very similar to a JDBC type recordset, but is generic and can actually contain tabular data from virtually any type of EIS system

Some methods of the IRecordSet interface:

next()	Next record
previous()	Previous record
getInt(String field)	Get Integer value for field
getString(String field)	Get String value for field
getColumnName(int col)	Get Name of column
deleteRow()	Deletes the current row
setInt(String field, int val)	Sets Integer value of field name
setString(int field,String val)	Sets String value of column number

CF API – Using the Data (cont.)

The recordset also contains metadata (IRecordMetaData) that can be used to dynamically determine column names and other attributes at runtime

```
IRecordMetaData rsMD = rs.retrieveMetaData();
response.write("<table><tr>");
int columnCount = rsMD.getColumnCount();
for (int index = 1; index <= columnCount; index++) {
      response.write("<th>" +
            rsMD.getColumnLabel(index) + "</th>");
}
```

Some Methods of IRecordMetaData

getColumnCount()	Get number of Columns
getColumnLabel(int column)	Get text description for the Column
getColumnName(int column)	Get the name of the Column
getColumnType(int column)	Get type of the Column

CF API – Retrieving Query Results

An IRecordset object containing all the rows resulting from the query can be retrieved from the query object using the retrieveRecordset() method of the query object

Metadata can also be obtained from the recordset using retrieveMetaData(). Metadata is often useful for determining what columns were returned by the query

```
import com.sapportals.connector.execution.structures.IRecordMetaData;
import com.sapportals.connector.execution.structures.IRecordSet;
...
IRecordSet rs = query.retrieveRecordSet();
IRecordMetaData rsMD = rs.retrieveMetaData();
response.write("<table><tr>");
int columnCount = rsMD.getColumnCount();
for (int index = 1; index <= columnCount; index++) {
        response.write("<th>"  + rsMD.getColumnLabel(index) + "</th>");
}
```

CF API – Retrieving Query Results (cont.)

The IRecordSet object contains the rows resulting from the query

You can iterate through the recordset and retrieve data using the next() method of the recordset of the query can be obtained if the query was successful

You can use the getXXX() methods (getString(), getInteger(), etc.) to retrieve values for each column of the row

```
...
// write data
response.write("</tr>");
while (rs.next()) {
        response.write("<tr>");
        for (int index = 1; index <= columnCount; index++) {
                response.write("<td>" + rs.getString(index) + "</td>");
        }
        response.write("</tr>");
}
response.write("</table>");
...
```

Questions?

Q&A

Copyright 2003 SAP AG. All Rights Reserved