Ask the Experts

JNDI Naming configuration and problem determination

24 October 2013

WebSphere® Support Technical Exchange

ON DEMAND BUSINESS™

Agenda

- Introduce the panel of experts
- Brief overview of JNDI Naming in WebSphere Application Server
- Answer questions submitted by email
- Open telephone lines for questions
- Summarize highlights

Panel of Experts

Panelist	Role at IBM
David Tiler	WebSphere Application Server Level 2 Support Analyst
Krishna Jaladhi	WebSphere Application Server Level 2 Support Analyst
Toan Nguyen	WebSphere Application Server Level 2 Support Analyst
Randal Anders	WebSphere Application Server Level 2 Support Analyst
Al Gilchrist	WebSphere Application Server Level 2 Support Analyst

Introduction

- We will be covering a number of questions that cover various Java Naming and Directory Interface (JNDI) Topics:

  - NameNotFoundExceptions
  - Common Troubleshooting Steps
  - Direct and Indirect JNDI Lookups
  - EJB 3.x JNDI Lookup Issues
  - Persistent namespace

- Covering WebSphere Application Server V7.0, V8.0, V8.5
- Applicable to Distributed Platforms

Introduction [continued]

- WebSphere Application Server provides a name server and JNDI implementation to allow access to resources
  - Enterprise JavaBeans (EJBs)
  - Data Sources, Connection Factories
  - JavaMail Providers, etc.
- Objects are bound to JNDI namespace when JVM starts or when application binds object, and are stored in hierarchical structure within a context
- Application obtains InitialContext and performs JNDI lookup to access resource
  - Can be done within JVM or remotely

Question 1

- What are some common reasons why I might get a NameNotFoundException when looking up a data source?

Answer to Question 1

- JNDI name of data source might not match JNDI name being looked up

  Check SystemOut.log and find WSVR0049I for data source:
  WSVR0049I: Binding myDB2DS as jdbc/myDB2DS

  Check application source code, find class and line number on stack trace
  DataSource ds = (DataSource)ctx.lookup("jdbc/myDB2DS");

- Ensure that JNDI lookup is done on the correct JVM

  Check PROVIDER_URL in application source code

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
  "com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL, "corbaloc:iiop:myhost.mycompany.com:2809");
Context ctx = new InitialContext(env);
DataSource ds = (DataSource)ctx.lookup("jdbc/myDB2DS");
```

  Default is to perform the lookup on same JVM

- If looking up on a remote JVM, use fully qualified JNDI name

  cell/nodes/nodeName/servers/serverName/jdbc/myDB2DS

  Fully qualified name can be found in dumpNameSpace output

Answer to Question 1 (continued)

- Data source might not be bound to the namespace due to an error
  - Look for WSVR0049I or error binding the data source
- Check the scope of the data source
  - Can be created at cell, node, cluster, server, or application scope
  - Data source needs to be defined at a scope that is visible to the application (under the same server, cluster, node, or cell)
- Resource reference might not be defined for the data source
  javax.naming.NameNotFoundException: Name comp/env/jdbc not found in context "java:"
  - This indicates that indirect JNDI lookup is being done, but resource reference is not defined
  - More details in Question 3
- Data source might not have been defined at all, or there is a typo in the JNDI name that is being looked up

Question 2

- What are the common troubleshooting steps that I can take to resolve a JNDI lookup problem?

Answer to Question 2

- If using corbaloc (***direct***) method, use ***dumpNameSpace*** to display contents of namespace from the machine where the application client is running.
  - was_root/bin/dumpNameSpace.sh -host myhost.mycompany.com -port 901 OR
  - was_root/bin/dumpNameSpace.sh –url corbaloc:iiop:myhost.mycompany.com:901

Code Sample

```
env.put(Context.PROVIDER_URL,"corbaloc::abc.ibm.com:2809");
ctx = new InitialContext(env);
Object obj = ctx.lookup("jdbc/PlantsByWebSphereDataSource");
```

dumpNamespace Sample (server1)

```
…
39 (top)/nodes/vanno/servers/server1/jdbc/petstore/PetStoreDB  PetStoreDB
40 (top)/nodes/vanno/servers/server1/jdbc/PlantsByWebSphereDataSource  PLANTSDB
41 (top)/nodes/vanno/servers/server1/jdbc/CatalogDB  CatalogDB
```

Answer to Question 2 (continued)

- If using java: Namespace (***indirect***) method, check mapping from the sample code to resource.xml to the ibm-resource.bnd.xmi, and finally with the object as seen in the namespace.

Code sample:

```
ctx = new InitialContext();
Object obj = ctx.lookup("java:comp/env/plantsby/PlantsDS");
```

Note: Mapping is configured between the resources.xml and ibm-resource-bnd.xmi files.

web.xml:

```
<resource-ref id="ResourceRef_1">
   <res-ref-name>plantsby/PlantsDS</res-ref-name>
   <res-type>javax.sql.DataSource</res-type>
   <res-auth>Container</res-auth>
</resource-ref>
```

ibm-web-bnd.xmi:

```
<resRefBindings xmi:id="ResourceRefBinding_1" jndiName="jdbc/PlantsByWebSphereDataSource">
   <bindingResourceRef href="WEB-INF/web.xml#ResourceRef_1"/>
</resRefBindings>
```

Sample piece of the dumpNameSpace Output:

```
. . .
39 (top)/nodes/server/servers/server1/jdbc/petstore/PetStoreDB  PetStoreDB
40 (top)/nodes/server/servers/server1/jdbc/PlantsByWebSphereDataSource  PLANTSDB
41 (top)/nodes/server/servers/server1/jdbc/CatalogDB  CatalogDB
. . .
```

Answer to Question 2 (continued)

If the configuration looks to be setup properly, when compared with the preceding sample, then get a dump of the applications, java:, namespace using the wsadmin utility.

Syntax for dumping java: namespace

Examples dumping the java: namespace

Question 3

- What are the differences and advantages of doing a direct and indirect JNDI lookup?

Answer to Question 3

- When a direct lookup is done, the actual JNDI name of the resource is looked up
  - Example: jdbc/myDataSource
- When an indirect lookup is done, a resource reference is looked up, not actual JNDI name
  - Example: java:comp/env/myResRef
- Resource reference is defined in deployment descriptor (web.xml or ejb-jar.xml) or annotation
  - Resource reference binding maps the resource reference to actual JNDI name of resource

Answer to Question 3 (continued)

- Direct JNDI Lookup
  - Advantage: Fewer steps in configuration
  - Disadvantage: Application not portable, must be changed if JNDI name changes
- Indirect JNDI Lookup
  - Advantage: Application is portable
    - If JNDI name changes, only need to change resource reference binding
  - Disadvantage: More configuration steps
- Indirect JNDI lookup is Java EE best practice

Question 4

- How can I resolve EJB 3.x (Enterprise Bean) JNDI lookup failures?

Answer to Question 4

- It is important to understand how EJBContainer binding EJB 3.0 and EJB 3.1 application.

- Use dumpNamespace tool and find how bindings look like in Namespace

(top)/nodes/sriharshaNode11/servers/server1/ejb/EJB31
(top)/nodes/sriharshaNode11/servers/server1/ejb/EJB31/EJB31EJB.jar
(top)/nodes/sriharshaNode11/servers/server1/ejb/EJB31/EJB31EJB.jar/Clock#com.ibm.ejb3.clock.view.ClockRemote

- EJB 3.0 and EJB 3.1 application bindings overview
http://www14.software.ibm.com/webapp/wsbroker/redirect?version=phil&product=was-base-iseries&topic=cejbbindingsejbfp

- Use SystemOut messages to find information on how EJBContainer is binding the EJBs to the namespace.

- Example messages are shown in next slide.

Answer to Question 4 (continued)

WSVR0200I: Starting application: EJB31
WSVR0037I: Starting EJB jar: EJB31EJB.jar
CNTR0167I: The server is binding the com.ibm.ejb3.clock.view.ClockRemote interface of the Clock enterprise bean in the EJB31EJB.jar module of the EJB31 application. The binding location is: ejb/EJB31/EJB31EJB.jar/Clock#com.ibm.ejb3.clock.view.ClockRemote
CNTR0167I: The server is binding the com.ibm.ejb3.clock.view.ClockRemote interface of the Clock enterprise bean in the EJB31EJB.jar module of the EJB31 application. The binding location is: com.ibm.ejb3.clock.view.ClockRemote
CNTR0167I: The server is binding the com.ibm.ejb3.clock.view.ClockRemote interface of the Clock enterprise bean in the EJB31EJB.jar module of the EJB31 application. The binding location is: java:global/EJB31/EJB31EJB/Clock!com.ibm.ejb3.clock.view.ClockRemote
CNTR0167I: The server is binding the com.ibm.ejb3.clock.Clock interface of the Clock enterprise bean in the EJB31EJB.jar module of the EJB31 application. The binding location is: ejblocal:EJB31/EJB31EJB.jar/Clock#com.ibm.ejb3.clock.Clock
CNTR0167I: The server is binding the com.ibm.ejb3.clock.Clock interface of the Clock enterprise bean in the EJB31EJB.jar module of the EJB31 application. The binding location is: ejblocal:com.ibm.ejb3.clock.Clock
CNTR0167I: The server is binding the com.ibm.ejb3.clock.Clock interface of the Clock enterprise bean in the EJB31EJB.jar module of the EJB31 application. The binding location is: java:global/EJB31/EJB31EJB/Clock!com.ibm.ejb3.clock.Clock
CNTR0167I: The server is binding the com.ibm.ejb3.clock.view.ClockLocal interface of the Clock enterprise bean in the EJB31EJB.jar module of the EJB31 application. The binding location is: ejblocal:EJB31/EJB31EJB.jar/Clock#com.ibm.ejb3.clock.view.ClockLocal
CNTR0167I: The server is binding the com.ibm.ejb3.clock.view.ClockLocal interface of the Clock enterprise bean in the EJB31EJB.jar module of the EJB31 application. The binding location is: ejblocal:com.ibm.ejb3.clock.view.ClockLocal
CNTR0167I: The server is binding the com.ibm.ejb3.clock.view.ClockLocal interface of the Clock enterprise bean in the EJB31EJB.jar module of the EJB31 application. The binding location is: java:global/EJB31/EJB31EJB/Clock!com.ibm.ejb3.clock.view.ClockLocal
WSVR0057I: EJB jar started: EJB31EJB.jar

Question 5

- When the size of the namestore.xml file in my environment grows rapidly, causing memory-related issues, how can I safely remove the entries in this file?

Answer to Question 5

- namestore.xml stores cell-scoped persistent partition of the name space

- namestore.xml persists JNDI bindings prefixed by **cell/persistent**

Example:

Context ic = new InitialContext();
*ic.**bind**("cell/persistent/Hello", "hello");*

- bindings should be removed if no longer needed

Example:

Context ic = new InitialContext();
*ic.**unbind**("cell/persistent/Hello");*

Answer to Question 5 (continued)

- Writing to cell persistent is expensive, not for rapidly changing bindings
- Deployment Manager must be running for cell persistent update operations

Open Lines for Questions

Connect with us!

1. **Get notified on upcoming webcasts**
   Send an e-mail to wsehelp@us.ibm.com with subject line “wste subscribe” to get a list of mailing lists and to subscribe
2. **Tell us what you want to learn**
   Send us suggestions for future topics or improvements about our webcasts to wsehelp@us.ibm.com
3. **Be connected!**
   Connect with us on Facebook
   Connect with us on Twitter

Summary

Additional WebSphere Product Resources

- Learn about upcoming WebSphere Support Technical Exchange webcasts, and access previously recorded presentations at:
  http://www.ibm.com/software/websphere/support/supp_tech.html
- Discover the latest trends in WebSphere Technology and implementation, participate in technically-focused briefings, webcasts and podcasts at:
  http://www.ibm.com/developerworks/websphere/community/
- Join the Global WebSphere Community:
  http://www.websphereusergroup.org
- Access key product show-me demos and tutorials by visiting IBM® Education Assistant:
  http://www.ibm.com/software/info/education/assistant
- View a webcast replay with step-by-step instructions for using the Service Request (SR) tool for submitting problems electronically:
  http://www.ibm.com/software/websphere/support/d2w.html
- Sign up to receive weekly technical My Notifications emails:
  http://www.ibm.com/software/support/einfo.html