IBM Software Group

Object Request Broker (ORB) Problem Determination and Best Practices

Paul Bullis
WebSphere L2 Support Team Lead

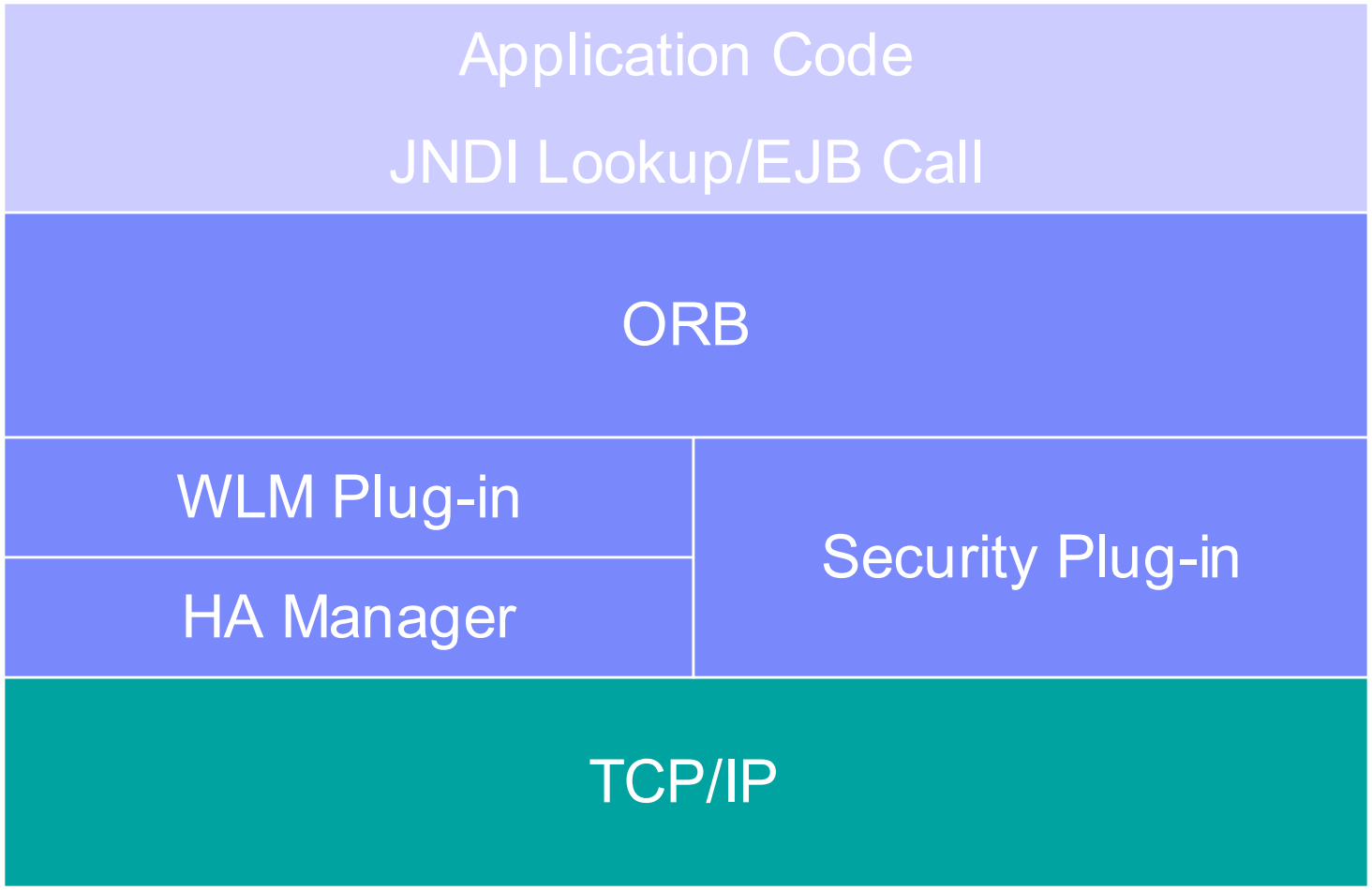
WebSphere® Support Technical Exchange

Agenda

- Introduction
- Base ORB vs. Extension ORB
- Enabling ORB Tracing
- Reading an ORB Trace
- Typical ORB Request Flows
- Common ORB Exceptions
- Best practices to avoid common ORB problems

Introduction

- Object Request Broker (ORB) manages the interaction between clients and servers, using the Internet InterORB Protocol (IIOP)
- Provides a framework for clients to locate objects in the network and to call operations on those objects as if the remote objects are located in the same running process as the client
- IBM WebSphere Application Server uses the ORB to facilitate client/server RMI communication as well as to communicate between components


Application Code
JNDI Lookup/EJB Call
ORB
WLM Plug-in
HA Manager
Security Plug-in
TCP/IP

Base ORB vs. Extension ORB

- Base ORB
  - Underlying code that implements the Java implementation of the CORBA ORB
  - Provided as a part of the SDK shipped with WebSphere Application Server in all versions
  - Base ORB classes packaged in ibmorb.jar, ibmorbutil.jar, ibmorbapi.jar and iwsorbutil.jar found under the <WAS_ROOT>/java/jre/lib or <WAS_ROOT>/java/jre/lib/ext directories (varies based on WebSphere version)
  - Package Names
    - com.ibm.CORBA.*
    - com.ibm.rmi.*
    - javax.rmi.CORBA.*
    - org.omg.CORBA.*
  - Properties File
    - <WAS_ROOT>/java/jre/lib/orb.properties

Base ORB vs. Extension ORB (Cont'd)

- Extension ORB
  - WebSphere code responsible for managing the ORB instance in the AppServer
  - Provides Plug-in framework for Interceptors (i.e WLM, Security)
  - Extension ORB classes packaged in iwsorb.jar and bootstrap.jar found under the <WAS_ROOT>/lib directory (file name may differ between WebSphere versions)
  - Package Names
    - com.ibm.ws.orb.*
    - com.ibm.ws.orbimpl.*
    - com.ibm.CORBA.services.*

Enabling ORB Tracing

- Two types of ORB tracing
  - Comm Tracing (Base ORB)
    - Shows in coming/out going requests in packet style format
    - Enabled by adding the following two system properties:
      -Dcom.ibm.CORBA.CommTrace=true
      -Dcom.ibm.CORBA.Debug=true
    - Can also be enabled through the Admin Console ORB Service panel ORB tracing checkbox:

      Servers > Application Servers > SERVER_NAME > Container Services > ORB Service

Enabling ORB Tracing (Cont'd)

- ORBRas Tracing (Extension ORB)
  - WebSphere style trace output
  - Can be enabled as a runtime trace (does not require restart)
  - Trace string:

    ORBRas=all (V6.0 and V6.1)

    ORBRas=all=enabled (V5.1)
- Both Comm and ORBRas tracing will be written to the WebSphere trace log if enabled on an AppServer
  - Comm trace written to orbtrc.<timestamp>.txt file in the current directory if enabled on a stand-alone java client

Reading an ORB Trace

- ORBRas trace points of interest
  - logVersions – Prints the ORB and SDK version during AppServer startup.

```
[5/12/07 22:42:24:062 CDT] 0000000a ORBRas         3 com.ibm.rmi.util.Version
   logVersions:120 Thread-1 IBM Java ORB build orb142-20061109 (SR7)
[5/12/07 22:42:24:062 CDT] 0000000a ORBRas         3 com.ibm.rmi.util.Version
   logVersions:124 Thread-1 J2RE 1.4.2 IBM Windows 32 build cn142-20061124
   (SR7) (JIT enabled: jitc)
```

  - readProperties – Dumps the current set of ORB properties to the trace log. Can be used to determine if properties are being picked up properly and what those properties are set to.

```
[5/12/07 22:42:24:078 CDT] 0000000a ORBRas         < com.ibm.CORBA.iiop.ORB
   readProperties:1641 Thread-1 Exit {
com.ibm.CORBA.BootstrapHost=bullis.austin.ibm.com - Source: ORB.init,
com.ibm.CORBA.CommTrace=true - Source: ORB.init,
com.ibm.CORBA.LocateRequestTimeout=180 - Source: ORB.init,
com.ibm.CORBA.BootstrapPort=9810 - Source: ORB.init,
com.ibm.CORBA.ListenerPort=0 - Source: ORB.init,
com.ibm.CORBA.LocalHost=bullis.austin.ibm.com - Source: ORB.init,
com.ibm.CORBA.RequestTimeout=180 - Source: ORB.init}
com.ibm.CORBA.ServerName=server1 - Source: ORB.init
```

Reading an ORB Trace (Cont'd)

- createServerSocket – Prints the hostname/port for each socket opened by the ORB to listen for new requests. Useful for verifying which ports an AppServer is listening on in the trace.

```
[5/12/07 22:42:36:500 CDT] 0000000a ORBRas         3
   com.ibm.ws.orbimpl.transport.WSTransport
   createServerSocket(ServerConnectionData) Thread-1 The ServerSocket being
   returned is ServerSocket[addr=0.0.0.0/0.0.0.0,port=0,localport=9810]
```

- getConnection (Client) – Printed when the client is making an outbound connection, providing the hostname/port, IOR, and operation for the request.

```
[5/13/07 21:19:12:594 CDT] 00000035 ORBRas         3
   com.ibm.ws.orbimpl.transport.WSTransport getConnection(Profile,
   ClientDelegate, operationName) WebContainer : 0  method entry:
   host=bullis.austin.ibm.com port=1151 clientDelegate=<IOR_REMOVED>
   operationName=getProperties
```

- createStreamObject (Server) – Printed when an incoming request is being processed by the ORB. Provides the hostname/port information about the incoming ORB request.

```
[5/13/07 21:19:12:609 CDT] 0000009d ORBRas         >
   com.ibm.rmi.iiop.Connection createStreamObject:1175
   RT=8:P=743844:O=0:WSTCPTransportConnection[addr=9.3.161.180,port=1151,local
   =3105] Entry
```

Reading an ORB Trace (Cont'd)

- createRequest_WLM (Client) – Indicates that the WLM plug-in will be used to create the outbound request as opposed to createRequest

```
[5/22/07 23:42:00:766 CDT] 0000000a ORBRas         >
   com.ibm.CORBA.iiop.ClientDelegate@226f6690 _createRequest_WLM:2025 Thread-1
   Entry
```

- ClientRequestImpl <init> (Client) – Printed during outbound client call. Operation and Request ID included in output which can be used to track the request in both the client and server.

```
[5/13/07 21:19:12:609 CDT] 00000035 ORBRas         >
   com.ibm.rmi.iiop.ClientRequestImpl <init>:121 WebContainer : 0 Entry
   opName[getProperties] RequestId[21]
```

- sendFragment (Server) – Printed when the server is sending an ORB fragment back to the client. The Request ID is included which can be used to correlate the client and server traces.

```
[5/12/07 22:46:25:500 CDT] 00000041 ORBRas         3
   com.ibm.rmi.iiop.ServerResponseImpl sendFragment:188 ORB.thread.pool : 0
   reqId 21 next fragment offset_hex 400
```

Reading an ORB Trace (Cont'd)

- Comm trace points of interest
  - GIOP message printed for Incoming and Outgoing requests
    - GIOP message type (line 3)
    - Date and time that the message was recorded (line 4)
    - Information that is useful to identify the thread that is running when the message records, and other thread-specific information (line 5)
    - Local and remote TCP/IP ports used for the interaction (lines 6 through 9)
    - GIOP version, byte order, an indication of whether the message is a fragment, and message size (lines 10 through 13)
  - GIOP message type can be Locate Request Message, Request Message, Locate Reply Message or Reply Message
  - Each outgoing request from the client will have a matching incoming request on the server if the ORB message is successfully sent between the two ORBs
    - When using SSL/Global Security, the outgoing request for the remote server port will not match the request header on the server side
    - ConnectionTableImpl getConnection Exit trace point will have correct port in client trace
  - InfoCenter article detailing Comm trace:

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.nd.doc/info/ae/ae/rorb_traceo.html

Reading an ORB Trace (Cont'd)

```
IN COMING:
Request Message
Date:          May 12, 2007 11:28:55 PM CDT
Thread Info:
   RT=7:P=987688:O=0:WSTCPTransportConnection[addr=9.67.7.21
   ,port=2428,local=9900]
Local Port:    9900 (0x26AC)
Local IP:      9.3.161.180
Remote Port:   2428 (0x97C)
Remote IP:     9.67.7.21
GIOP Version:  1.2
Byte order:    big endian
Fragment to follow: No
Message size:  348 (0x15C)
--
Request ID:          236
Response Flag:       WITH_TARGET
Target Address:       0
Object Key:          length = 145 (0x91)
Operation:           create
Service Context:     length = 1 (0x1)
      Context ID:    1229081865 (0x49424D09)
Data Offset:         161
```

Reading an ORB Trace (Cont'd)

```
OUT GOING:
Locate Request Message
Date:          May 13, 2007 9:29:06 PM CDT
Thread Info:   Thread-1
Local Port:    3305 (0xCE9)
Local IP:      127.0.0.1
Remote Port:   9900 (0x26AC)
Remote IP:     9.3.161.180
GIOP Version:  1.2
Byte order:    big endian
Fragment to follow: No
Message size:  56 (0x38)
--
Request ID:        6 (0x6)
Target Address:     0
Object Key:        length = 44 (0x2C)
                   4A4D4249 00000010 11C328FE 00000000
                   00000000 00000000 00000000 00000024
                   00000008 00000001 00000000

0000: 47494F50 01020003 00000038 00000006
0010: 00000000 0000002C 4A4D4249 00000010
0020: 11C328FE 00000000 00000000 00000000
0030: 00000000 00000024 00000008 00000001
0040: 00000000
```

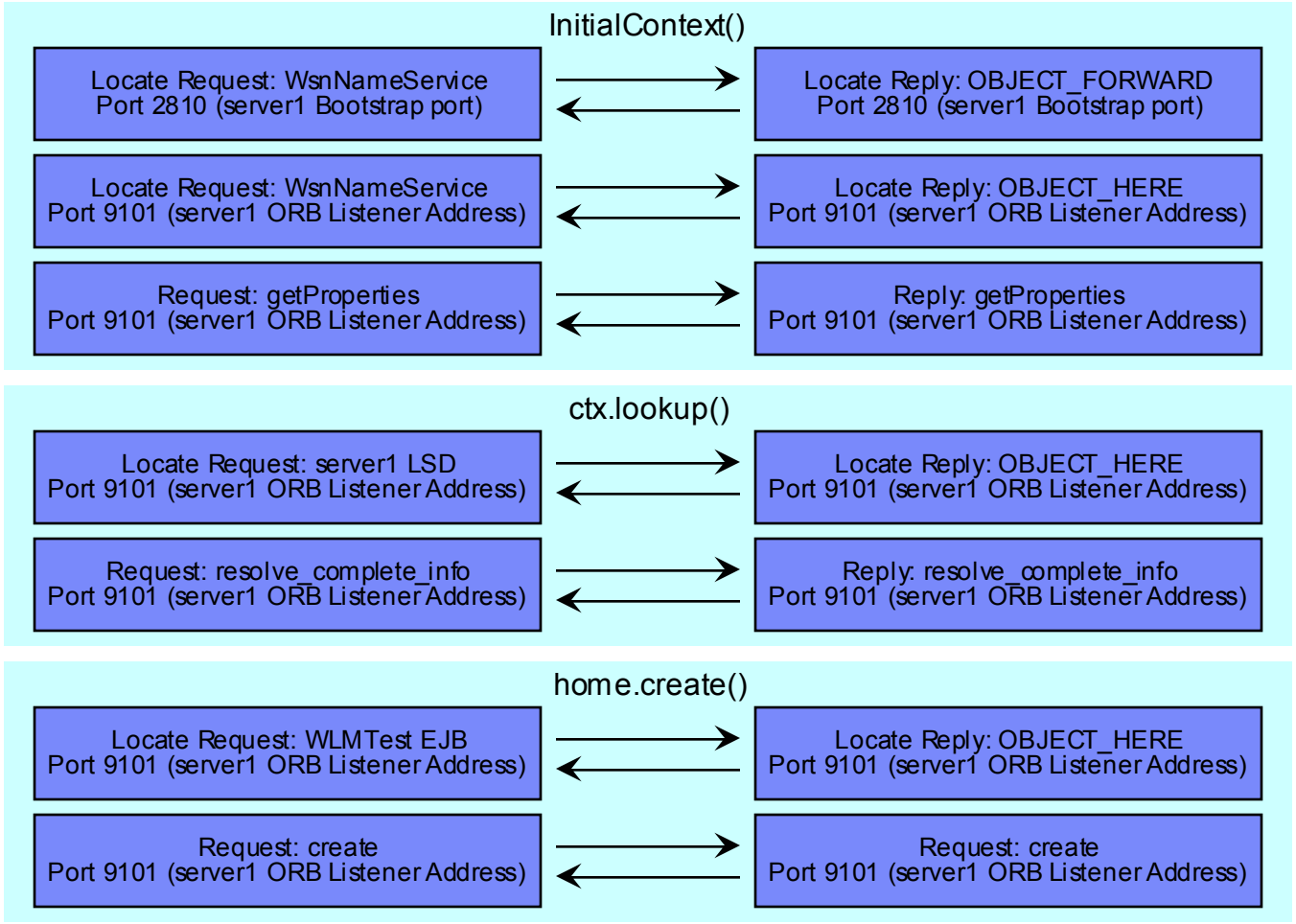
```
IN COMING:
Locate Request Message
Date:          May 13, 2007 9:29:06 PM CDT
Thread Info:
   RT=15:P=987688:O=0:WSTCPTransportConnection[addr=9.
   3.161.180,port=3305,local=9900]
Local Port:    9900 (0x26AC)
Local IP:      9.3.161.180
Remote Port:   3305 (0xCE9)
Remote IP:     9.3.161.180
GIOP Version:  1.2
Byte order:    big endian
Fragment to follow: No
Message size:  56 (0x38)
--
Request ID:        6 (0x6)
Target Address:     0
Object Key:        length = 44 (0x2C)
                   4A4D4249 00000010 11C328FE 00000000
                   00000000 00000000 00000000 00000024
                   00000008 00000001 00000000

0000: 47494F50 01020003 00000038 00000006
0010: 00000000 0000002C 4A4D4249 00000010
0020: 11C328FE 00000000 00000000 00000000
0030: 00000000 00000024 00000008 00000001
0040: 00000000
```

Typical ORB Request Flows

- EJB lookup/create call from client to EJB deployed in a stand alone AppServer
- Client code
  - Lines that result in remote ORB calls highlighted in red

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,"com.ibm.websphere.naming
.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL,"corbaloc::localhost:2810");
Context ctx = new InitialContext(env);
TestEJBHome home = (TestEJBHome)
PortableRemoteObject.narrow(ctx.lookup("ejb/ejbs/TestEJBHome"),Te
stEJBHome.class);
TestEJB bean = home.create();
```

Typical ORB Request Flows (Cont'd)

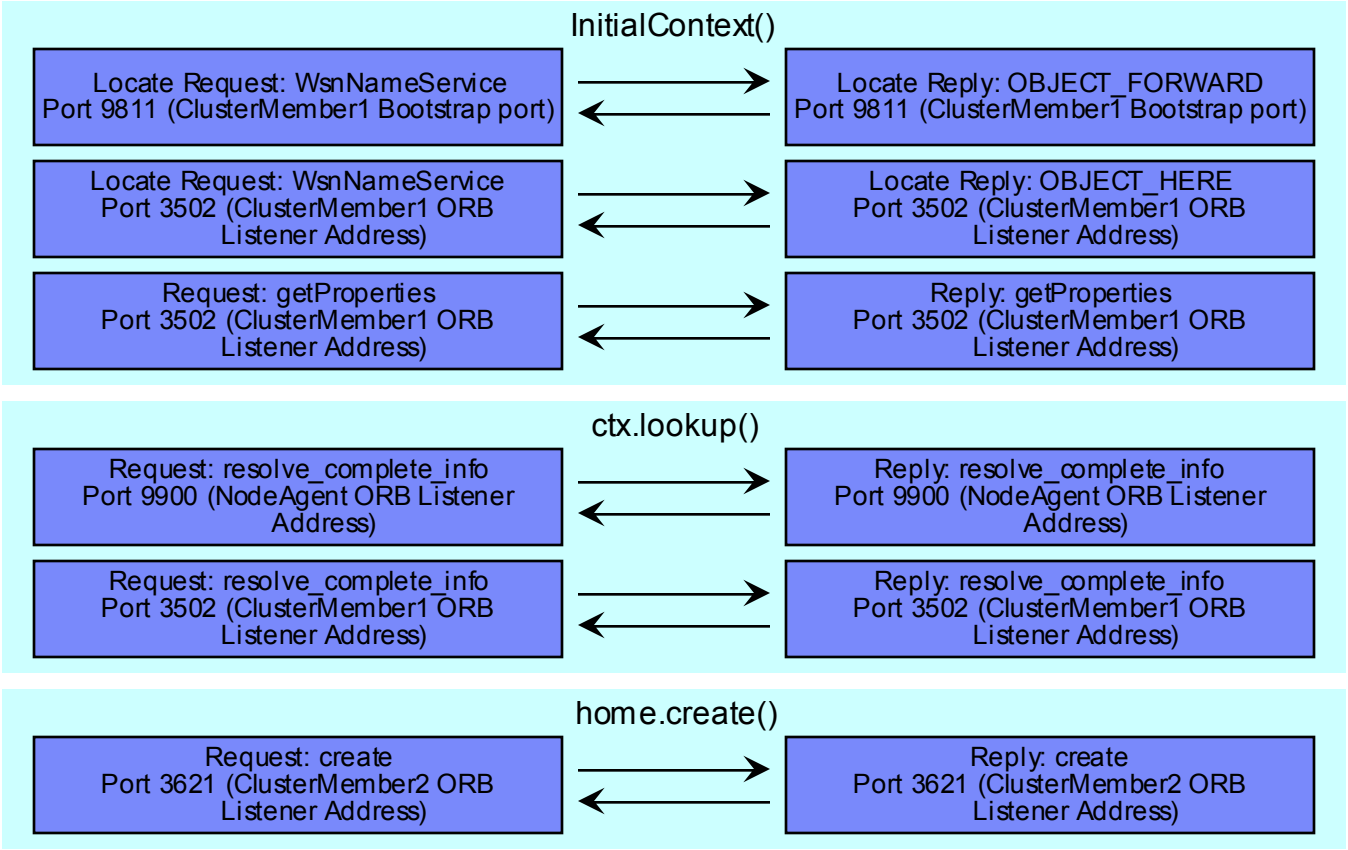


Typical ORB Request Flows (Cont'd)

- EJB lookup/create call from client to EJB deployed in a ND cluster or non-clustered ND AppServer
- Client code
  - Lines that result in remote ORB calls highlighted in red
  - Additional server added to the PROVIDER_URL for initialContext() failover

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,"com.ibm.websphere.naming
.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL,"corbaloc::localhost:9811,:localhost
:9812");
Context ctx = new InitialContext(env);
TestEJBHome home = (TestEJBHome)
PortableRemoteObject.narrow(ctx.lookup("ejb/ejbs/TestEJBHome"),Te
stEJBHome.class);
TestEJB bean = home.create();
```

Typical ORB Request Flows (Cont'd)



Common ORB Exceptions

- Common ORB exceptions and solutions:

  http://www-1.ibm.com/support/docview.wss?rs=180&uid=swg21237101

- Exceptions that are not typically ORB related
  - Workload Management exceptions
    - org.omg.CORBA.NO_IMPLEMENT: No Available Target
      - WLM attempted to route a request, but based on the current cluster data, there isn't a target which would be able to service that request
    - org.omg.CORBA.NO_IMPLEMENT: No Cluster Data Available
      - Error is thrown when WLM attempts to choose a target to route a request, but has no cluster data and is unable to gather any data
    - org.omg. CORBA.TRANSIENT: SIGNAL_RETRY
      - The ORB attempts to send a request out to a target that WLM chooses and then never receives a reply whether the request was successful or not
      - Root cause can be an ORB/TCPIP failure
  - Security Exception
    - org.omg.CORBA.NO_PERMISSION
      - Security exception while trying to communicate to a server with Global security enabled

ORB Best Practices

- Use ORB clients and servers that are at the same WebSphere and SDK version
  - Application clients and thin clients should also match the target server runtime as closely as possible
  - Clients and Servers at different SDK versions can cause client callbacks, which if not configured correctly, can be blocked by firewalls
- Do not call Orb.init() in an application
  - WebSphere provides an ORB instance for applications to use
  - Calling Orb.init() in an application can cause port conflicts because both ORB instances are using the same port configuration
- Hard code AppServer ORB Listener Address ports
  - Adds complexity to managing large environments
  - Reduces the unknowns of dynamic port assignment to client applications

Summary

- Difference between Base ORB and Extension ORB
- How to enable ORB tracing and where the data is written
- How to read an ORB trace and follow a request from client to server
- Typical ORB request flows in both Base and ND environments
- Common ORB exceptions that are not due to an ORB problem
- Best practices to avoid ORB problems

Additional WebSphere Product Resources

- Discover the latest trends in WebSphere Technology and implementation, participate in technically-focused briefings, webcasts and podcasts at: www.ibm.com/developerworks/websphere/community/
- Learn about other upcoming webcasts, conferences and events: www.ibm.com/software/websphere/events_1.html
- Join the Global WebSphere User Group Community: www.websphere.org
- Access key product show-me demos and tutorials by visiting IBM Education Assistant: ibm.com/software/info/education/assistant
- Learn about the Electronic Service Request (ESR) tool for submitting problems electronically: www.ibm.com/software/support/viewlet/probsub/ESR_Overview_viewlet_swf.html
- Sign up to receive weekly technical My support emails: www.ibm.com/software/support/einfo.html

Questions and Answers