Middleware and Web Services
Lecture 4: Application Server Services

doc. Ing. Tomáš Vitvar, Ph.D.

tomas@vitvar.com • @TomasVitvar • http://vitvar.com

Czech Technical University in Prague

Faculty of Information Technologies • Software and Web Engineering • http://vitvar.com/courses/mdw
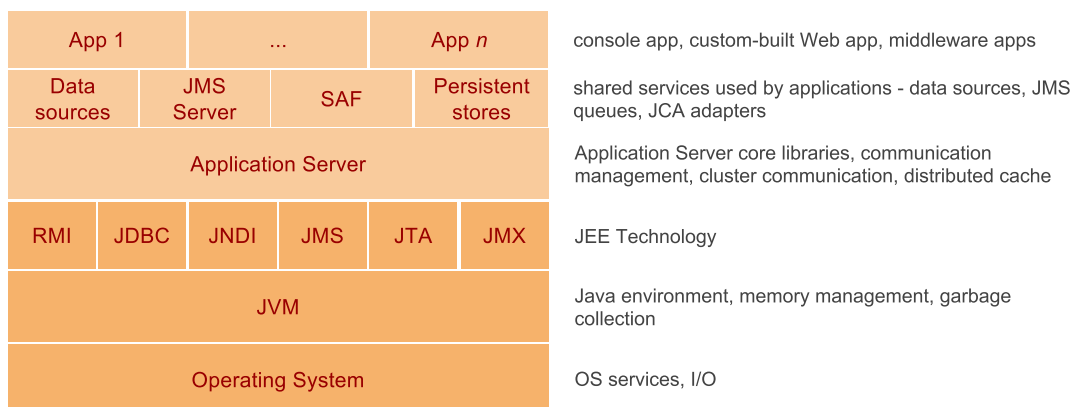

Overview

- ## Architecture
  - *Views*
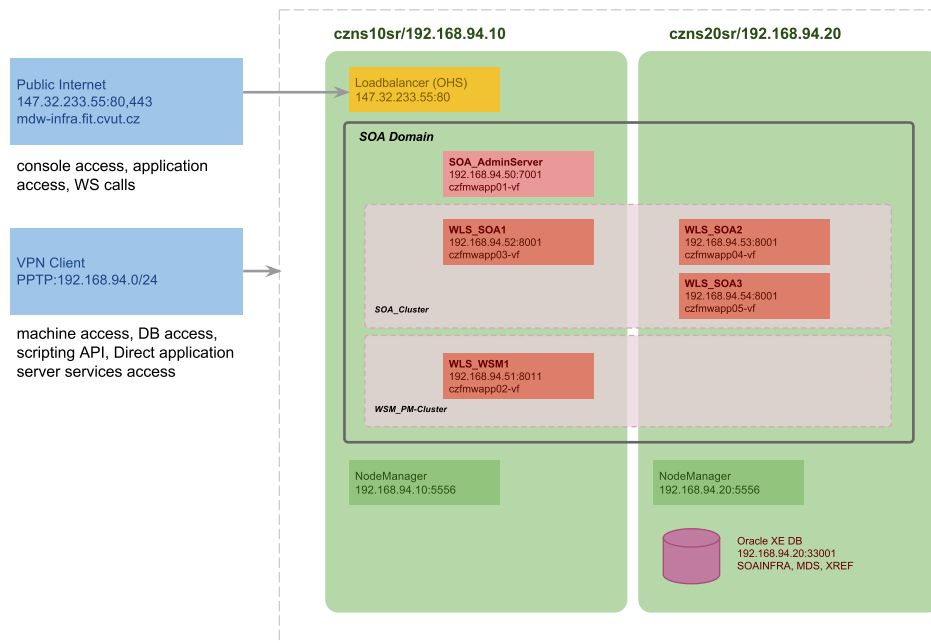- ## JEE Technologies and Services

Application Server Overview

- An environment that runs an application
  - *A client communicates with the server using an application protocol*
- Application Server
  - *Layered system*
    - → *OS, JVM, JEE components, AS services, Applications*
  - *Infrastructure*
    - → *Multiple instances of application servers run across multiple machines*
    - → *Scalability, Performance, Failover, High Availability*

Application Server Layers

App 1	...	App *n*	console app, custom-built Web app, middleware apps

Data sources	JMS Server	SAF	Persistent stores	shared services used by applications - data sources, JMS queues, JCA adapters

Application Server	Application Server core libraries, communication management, cluster communication, distributed cache

RMI	JDBC	JNDI	JMS	JTA	JMX	JEE Technology

JVM	Java environment, memory management, garbage collection

Operating System	OS services, I/O

- Features
  - *AS instance appears as a single process in the OS*
    - → *you can use standard OS commands to investigate its operation*
    - → *AS listens on a single or multipe IPs (VIPs) and a tcp port*
  - *AS is a Java process*
    - → *you can use Java tools to investigate its operation*
   - → *Garbage collector stats, thread dumps, memory allocations, etc.*

Weblogic MDW Infrastructure

czns10sr/192.168.94.10

czns20sr/192.168.94.20

Public Internet
147.32.233.55:80,443
mdw-infra.fit.cvut.cz

console access, application
access, WS calls

VPN Client
PPTP:192.168.94.0/24

machine access, DB access,
scripting API, Direct application
server services access

Loadbalancer (OHS)
147.32.233.55:80

SOA Domain

SOA_AdminServer
192.168.94.50:7001
czfmwapp01-vf

WLS_SOA1
192.168.94.52:8001
czfmwapp03-vf

WLS_SOA2
192.168.94.53:8001
czfmwapp04-vf

WLS_SOA3
192.168.94.54:8001
czfmwapp05-vf

SOA_Cluster

WLS_WSM1
192.168.94.51:8011
czfmwapp02-vf

WSM_PM-Cluster

NodeManager
192.168.94.10:5556

NodeManager
192.168.94.20:5556

Oracle XE DB
192.168.94.20:33001
SOAINFRA, MDS, XREF

Terminology

- Domain
  - *A group of servers with specific configuration of applications and objects*
- Administration Server
  - *An instance of application server that manages the domain*
- Managed Server
  - *An instance of application server running instances of applications and objects*
- Cluster
  - *A group of managed servers; they contain the same copy of applications and objects*
- Machine
  - *A physical machine and OS running one or more servers (Admin or Managed)*
- Node Manager
  - *A process that provides an access to admin and managed servers on the machine*
- Load Balancer
  - *A network element that distributes client requests to managed servers based on a specific algorithm*

Console Example – Weblogic Server
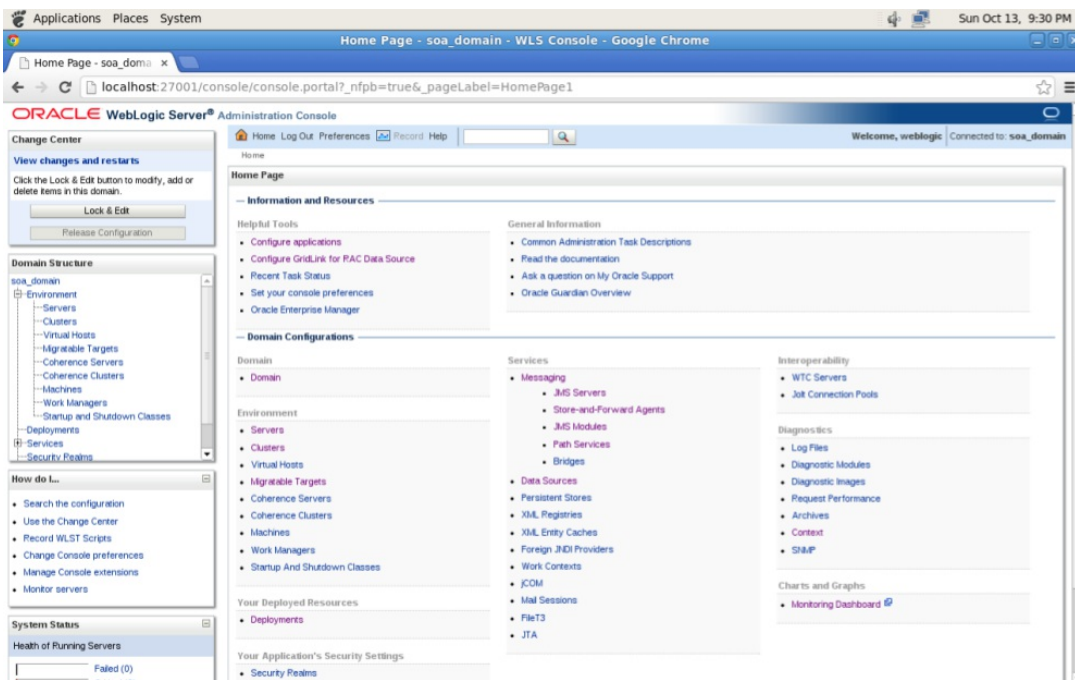
Overview

- Architecture
  - *Views*
- JEE Technologies and Services

Application Server from the OS View

- Process ID, command line arguments

```
5   $ ps ax | grep WLS_SOA
6    1820 ?         Sl   289:15 /opt/oracle/jrockit/bin/java -jrockit
7   -Xms768m -Xmx1536m -Dweblogic.Name=WLS_SOA1 -Djava.security.policy=
8   /opt/oracle/11g/fmw/wlserver_10.3/server/lib/weblogic.policy
9   -Dweblogic.ProductionModeEnabled=true
10  ...
```

- Open files by the process

```
5   $ ll -l /proc/1820/fd
6   lr-x------ 1 oracle oinstall 64 Oct 12 16:53 0 -> /dev/null
7   l-wx------ 1 oracle oinstall 64 Oct 12 16:53 1 -> /opt/oracle/11g/domains/soa_domain/se
8   lr-x------ 1 oracle oinstall 64 Oct 12 16:53 10 -> /opt/oracle/11g/fmw/oracle_common/mo
9   lr-x------ 1 oracle oinstall 64 Oct 12 16:53 100 -> /opt/oracle/11g/fmw/modules/com.bea
10  ...
```

- Open sockets by the process

```
5   $ netstat -anp | grep 1820
6   tcp  0 0 192.168.94.52:8001       0.0.0.0:*             LISTEN      1820/java
7   tcp  0 0 192.168.94.10:8088       0.0.0.0:*             LISTEN      1820/java
8   tcp  0 0 192.168.94.10:39763      192.168.94.20:33001   ESTABLISHED 1820/java
9   tcp  0 0 192.168.94.52:8001       192.168.94.20:59589   ESTABLISHED 1820/java
10  tcp  0 0 192.168.94.10:33498      192.168.94.20:33001   ESTABLISHED 1820/java
11  tcp  0 0 192.168.94.10:33504      192.168.94.20:33001   ESTABLISHED 1820/java
12  ...
```

Application Server from the JVM View

- Thread dumps
  - *All threads that the application server uses, a snapshot on all the threads*
  - *Prints stack trace of currently run threads*

```
5   $ jrockit 1820 print_threads
```

- Command line arguments
  - *Prints all command line arguments of the JVM process*
    → *Memory settings, log file locations, etc.*

```
5   $ jrockit 1820 command_line
```

- Java flight recordings
  - *Recordings of the JVM process in time (usually 5 minutes)*
 - *Shows memory usages, garbage collections phases, threads statuses, etc.*

Overview

- Architecture
- JEE Technologies and Services
  - *Remote Method Invocation*
  - *Java Database Connectivity*
  - *Java Naming and Directory Interface*
  - *Application Server and JNDI*
  - *Two-phase Commit*

Overview

- Communication among Java-based applications
  - *Methods of a Java class can be invoked by other Java class remotely*
  - *Uses Java Remote Method Protocol (JRMP)*
    - → *Java-specific application protocol over TCP/IP*
  - *Basis for JEE technologies, such as JMS*

- Terminology
  - ***Client*** – *a program that invokes a remote method*
  - ***Server*** – *a program that exports a remote object*
  - ***Stub*** – *a representation of the client-side object for communication*
  - ***Skeleton*** – *a representation of the server-side object for communication*
  - ***Registry*** – *a component that holds a stub*
  - ***Marshalling/Unmarshalling*** – *a process of transforming memory representation of the object to a form suitable for network transmittion and vice-cersa*

Architecture
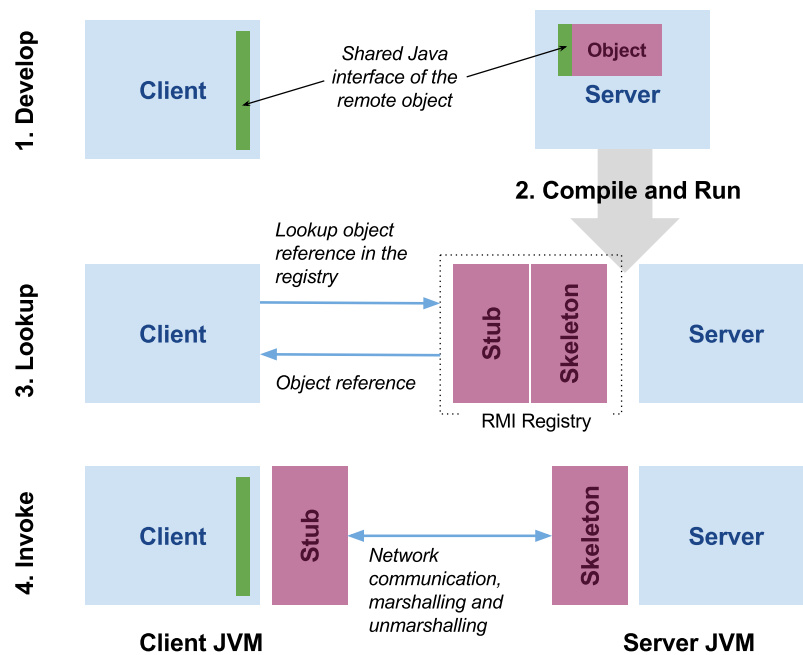
RMI Implementation in Java – Interface

- ## Shared interface

```
1  import java.rmi.Remote;
2  import java.rmi.RemoteException;
3
4  // shared interface between a client and a server to
5  // invoke methods on the remote object
6  public interface HelloRMIInterface extends Remote {
7      public String calculate(int a, int b) throws RemoteException;
8  }
```

- ## RMI Server

```
1  import java.rmi.Naming;
2  import java.rmi.RemoteException;
3  import java.rmi.RMISecurityManager;
4  import java.rmi.server.UnicastRemoteObject;
5  import java.rmi.registry.LocateRegistry;
6
7  public class Server extends UnicastRemoteObject implements HelloRMIInterface {
8
9      // implementation of the interface method
10     public int calculate(int a, int b) throws RemoteException {
11         return a+b;
12     }
```

RMI Implementation in Java – Server

- RMI Server (cont.)

```java
// start the server and register the object in the rmi registry
public static void main(String args[]) {
    try {
        // install a security manager (uses a security policy)
        if (System.getSecurityManager() == null) {
            RMISecurityManager sm = new RMISecurityManager();
            System.setSecurityManager(sm);
        }

        // create rmi registry
        LocateRegistry.createRegistry(1099);

        // create remote object
        Server obj = new Server();

        // Bind the object instance to the name "HelloRMI"
        // 0.0.0.0 denotes the service will listen on all network interfaces
        Naming.rebind("//0.0.0.0/HelloRMI", obj);

        System.out.println("RMI server started, " +
            "listening to incoming requests...");
    } catch (Exception e) {
        System.out.println("Error occurred: " + e.getLocalizedMessage());
    }
}
```

RMI Implementation in Java – Client

- RMI Client

```java
import java.rmi.Naming;

public class Client {

    public static void main(String args[]) throws Exception {
        // get a reference to the remote object
        // assuming the server is running on the localhost
        HelloRMIInterface o = (HelloRMIInterface)
            Naming.lookup("//localhost/HelloRMI");

        // call the object method
        System.out.println(o.calculate(3, 4));
    }
}
```
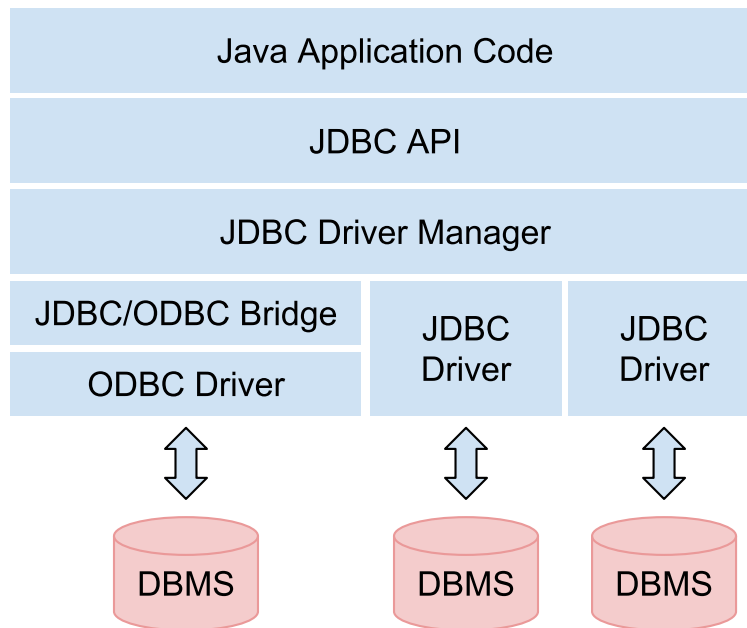
Overview

- Architecture
- JEE Technologies and Services
  - *Remote Method Invocation*
  - *Java Database Connectivity*
  - *Java Naming and Directory Interface*
  - *Application Server and JNDI*
  - *Two-phase Commit*

Overview

- Uniform API to access any kind of tabular data
  - *No need to deal with specific APIs of DBMS vendors*
- JDBC components
  - *JDBC API*
    - → *defines methods to execute SQL statements and retrieve results*
  - *Driver Manager*
    - → *provides drivers that provide access to a specific DBMS*
    - → *they implement a specific protocol to access the DBMS*
  - *JDBC-ODBC Bridge*
    - → *a software bridge which provides access via ODBC drivers*
    - → *ODBC driver is a driver in C for accessing DBMS*

JDBC Architecture


```
                    Java Application Code

                         JDBC API

                    JDBC Driver Manager

    JDBC/ODBC Bridge      JDBC        JDBC
                          Driver      Driver
      ODBC Driver


         DBMS             DBMS        DBMS
```

JDBC Example Implementation in Java

- ## JDBC Client

```
1   import java.sql.*;
2
3   public class JDBCClient {
4
5       public static void main(String args[]){
6       // database url
7       String db_url = "jdbc:oracle:thin:@czns20sr:33001:XE";
8
9       // username and password
10      String username = "myUsername";
11      String password = "myPassword";
12
13      try {
14          // Register JDBC driver (MySql driver)
15          Class.forName("oracle.jdbc.driver.OracleDriver");
16
17          // Open a connection
18          Connection con = DriverManager.getConnection(
19          db_url, username, password);
20
21          // Create and execute query statement
22          Statement stmt = con.createStatement();
23          String sql = "SELECT id, first, last, age FROM Employees";
24          ResultSet rs = stmt.executeQuery(sql);
25
```

JDBC Example Implementation in Java

- JDBC Client (cont.)

```
25          // Loop and extract received data
26          while (rs.next()) {
27              int id  = rs.getInt("id");
28              int age = rs.getInt("age");
29              String first = rs.getString("first");
30              String last = rs.getString("last");
31          }
32
33          // Release the connections
34          rs.close();
35          stmt.close();
36          conn.close();
37      }catch(SQLException se){
38
39          //Handle errors for JDBC
40          se.printStackTrace();
41      }catch(Exception e){
42
43          //Handle errors for Class.forName
44          e.printStackTrace();
45      }
46   }
47 }
48
```

Overview

- Architecture

- JEE Technologies and Services
  – *Remote Method Invocation*
  – *Java Database Connectivity*
  – *Java Naming and Directory Interface*
  – *Application Server and JNDI*
  – *Two-phase Commit*

Overview

- Objectives
  - *Allows to access objects by names in various directory systems and their attributes*
  - *Independent of any specific directory service implementation*
  - *Enables to distribute Java objects across various systems in the environment*

- Terminology
  - *Binding – association between a name and a object*
  - *Context – a set of bindings*

- JNDI Provides:
  - *a mechanism to bind an object to a name.*
  - *a directory lookup interface*
  - *a pluggable service provider interface (SPI) – any directory service implmentation can be plugged in*

JNDI Packages

- Naming Package
  - *interfaces to access naming services*
  - *Context: looking up, binding/unbinding, renaming, objects*

- Directory Package
  - *allows to retrieve attributes of objects, and to search for objects*

- Event Package
  - *allows for event notification in naming and directory services*
  - *For example, object was added, object changed, etc.*

- Other packages
  - *LDAP – allows to access LDAP services*
  - *Service Provider Interface – allows to develop various naming/directory services*
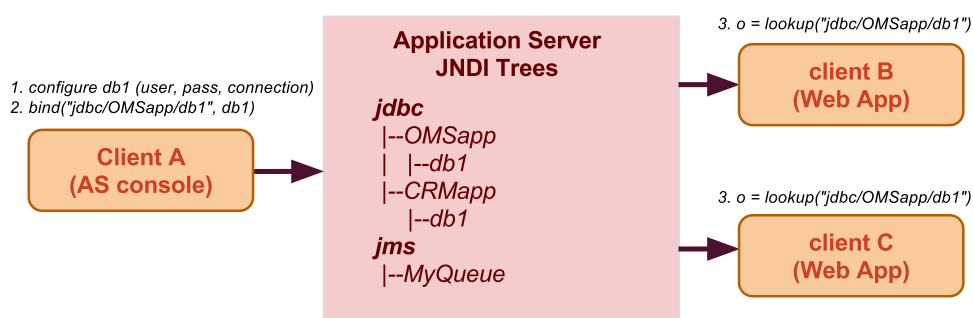
JNDI Architecture

Binding

Attributes

Context

DirContext

Initial Context

Application

Service Provider Interface

Naming Manager

FileSystem Service Provider

LDAP Service Provider

RMI Service Provider

File System

LDAP server

RMI Registry
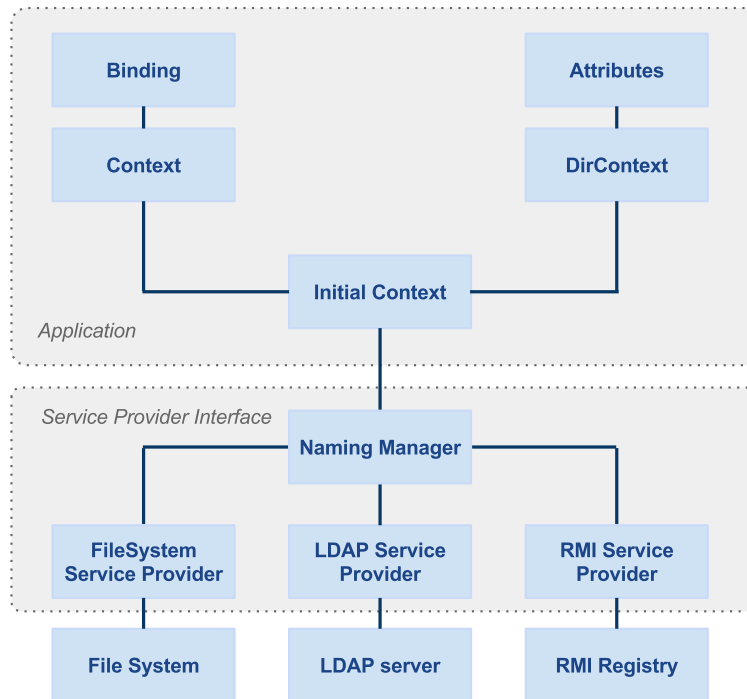
Application Server and JNDI

- ● Distribution of objects
  - – *Application Server provides central directory for various kinds of objects*
    - → *Datasources, JMS queues and topics, etc.*
  - – *Clients store objects in the central directory*
    - → *Administrator configures objects using Application Server Console or via AS API*
  - – *Clients retrieve objects from the central directory*

- ● Benefits
  - – *replication of objects across clients*
  - – *central configuration of objects' parameters*
  - – *scalability – allowing/disabling connections as required*

Overview

- Architecture
- JEE Technologies and Services
  - *Remote Method Invocation*
  - *Java Database Connectivity*
  - *Java Naming and Directory Interface*
  - *Application Server and JNDI*
  - *Two-phase Commit*

Application Server and JNDI

1. configure db1 (user, pass, connection)
2. bind("jdbc/OMSapp/db1", db1)

Client A (AS console)

Application Server JNDI Trees

jdbc
```
|--OMSapp
|   |--db1
|--CRMapp
    |--db1
```
jms
```
|--MyQueue
```

3. o = lookup("jdbc/OMSapp/db1")

client B (Web App)

3. o = lookup("jdbc/OMSapp/db1")

client C (Web App)

- Example Scenario
  - *Client A creates a datasource, configures it and registeres it in the JNDI tree*
    - → *Client A is a Admin server console app; this task is performed by the administrator*
  - *Client B and C lookup the object under specific JNDI name and retrieves the object from the tree*
    - → *They get the object from the tree and use it to connect to the DB*
    - → *They do not need to know any DB specific details, the object is pre-configured from the server*

Example Datasource in Weblogic

Settings for order-db

Configuration | Targets | Monitoring | Control | Security | Notes

General | Connection Pool | Oracle | ONS | Transaction | Diagnostics | Identity Options

Click the *Lock & Edit* button in the Change Center to modify the settings on this page.

Save

Applications get a database connection from a data source by looking up the data source on the Java Naming and provides the connection to the application from its pool of database connections.

This page enables you to define general configuration options for this JDBC data source.

Name: order-db

JNDI Name:

jdbc/order-db

☐ **Row Prefetch Enabled**

Row Prefetch Size: 48

Stream Chunk Size: 256

Save

Targets

- Object
  - *A service provided by the applicaton server, e.g. datasources, JMS queue, SAF*
- Types of services
  - *Pinned services*
    - → *Objects targeted to a single server only*
  - *Cluster services*
    - → *Objects targeted to all servers in the cluster*

Example Target Configuration

Cluster-wide JNDI Tree

- Cluster
  - *Every managed server has its own JNDI tree*
  - *Servers in a cluster sync up their JNDI trees as per the target configuration*
    - → *A stub of the object appears in every managed server's JNDI tree*

- When a client retrieves an object from the tree
  1. *Client connects to the cluster using the cluster address*
  2. *Client creates an initial context (represents a naming service)*
  3. *Client uses the initial context to lookup objects*
  4. *The naming service can failover to a next available managed server when lookup fails*

Example JNDI Tree on Weblogic Server

JNDI Implementation in Java

- Lookup for bound object

```
1   import javax.naming.InitialContext;
2   import java.util.*;
3   import javax.sql.*;
4
5   ...
6
7   Properties p = new Properties();
8
9   // configure the service provider url: "t3://localhost:7001"
10  p.put(Context.PROVIDER_URL,
11      "t3://czfmwapp03-vf:8001,czfmwapp04-vf:8001,czfmwapp05-vf:8001");
12
13  // configure the initial context factory.
14  // we use WebLogic context factory
15  p.put(Context.INITIAL_CONTEXT_FACTORY,
16      "weblogic.jndi.WLInitialContextFactory");
17  InitialContext ctx = new InitialContext(p);
18
19  dataSource =
20      (DataSource) ctx.lookup("jdbc/order-db");
21
22  // invoke the object method
23  Connection c = dataSource.getConnection();
24
```

Overview

- Architecture
- JEE Technologies and Services
  - *Remote Method Invocation*
  - *Java Database Connectivity*
  - *Java Naming and Directory Interface*
  - *Application Server and JNDI*
 - *Two-phase Commit*

Overview

- Coordination of a distributed transaction
  - *All transaction operations are completed across multiple resources; or none is completed*
  - *Able to deal with many types of failures (process, network, communication)*
- Terminology
  - ***Transaction Manager*** – *manages transactions, coordinates decisions for commit or rollback, and coordinates failure recovery*
  - ***Resource Manager*** – *manages an access to a resource that participates in the transacction, e.g. DBMS, JMS*
  - ***Agreement*** – *an agreement message send by the Resource Manager, whether the operation was processed successfuly*
  - ***Acknowledgment*** – *a message about a status of the operation execution*
  - ***Rollback*** – *operation that returns the Resource Manager state to its pre-transaction state.*

X/Open – eXtended Architecture (XA)

- Standard for executing distrubuted transactions
  - *Specifies how the coordinator will roll up the transaction against involved different systems.*
  - *Based on the Two-phase Commit protocol.*
  - *Defines interface between the coordinator and each system node.*
  - *Single transaction access to multiple resources (e.g. message queues, databases, etc.)*
  - *Defined in the eXtended Architecture Specification* ⧉
- Wide technological support
  - *Java Transaction API (JTA)* ⧉ *– distributed transactions in a Java environment.*
  - *Supported in the Oracle Service Bus through a JMS queue.*
  - *MySQL Relational Database Management System (since v5.0)*

Two-phase Commit

- Two-phase commit scenario
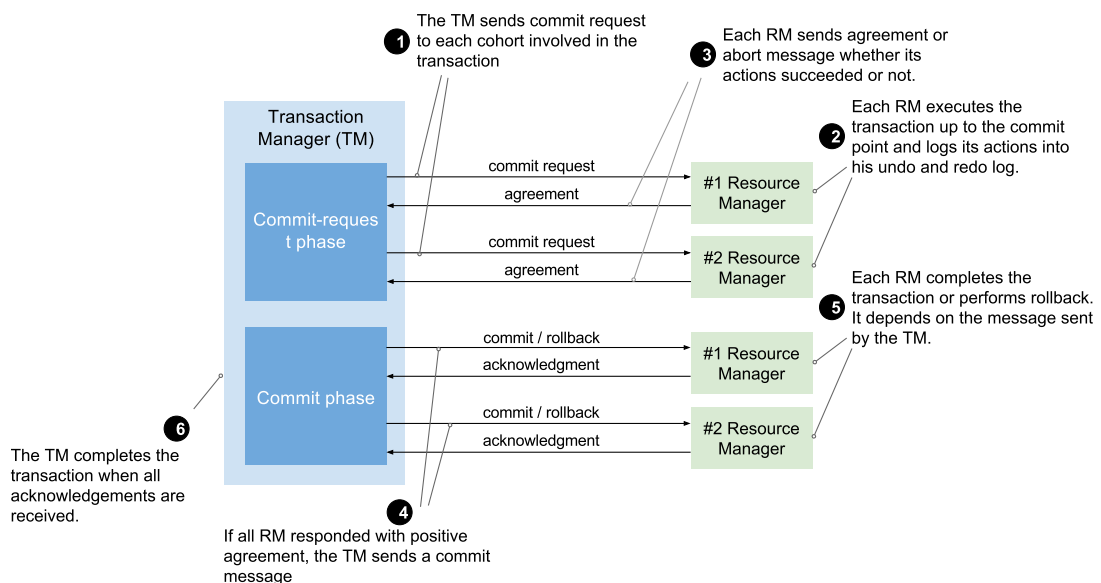
XA Example Implementation in Java

- Distributed Transaction

```
1   import java.sql.*;
2   import javax.sql.*;
3   import javax.naming.*;
4   import java.util.*;
5
6   public class Server {
7
8       public static void main(String args[]) {
9
10          // Initialize context
11          Hashtable parms = new Hashtable();
12          parms.put(Context.INITIAL_CONTEXT_FACTORY,
13              "weblogic.jndi.WLInitialContextFactory");
14          parms.put(Context.PROVIDER_URL, "t3://localhost:7001");
15          InitialContext ctx = new InitialContext(parms);
16
17          // Perform a naming service lookup to get UserTransaction object
18          javax.transaction.UserTransaction usertx;
19          usertx = (UserTransaction) ctx.lookup("java:comp/UserTransaction");
20
21          try {
22              //Start a new user transaction.
23              usertx.begin();
```

XA Example Implementation in Java

- Distributed Transaction (cont.)

```
24      // Establish a connection with the first database
25      javax.sql.DataSource data1;
26      data1=(javax.sql.DataSource)ctx.lookup("java:comp/env/jdbc/DataBase1");
27      java.sql.Connection conn1 = data1.getConnection();
28      java.sql.Statement stat1 = conn1.getStatement();
29      // Establish a connection with the second database
30      javax.sql.DataSource data2;
31      data2=(javax.sql.DataSource)ctx.lookup("java:comp/env/jdbc/DataBase2");
32      java.sql.Connection conn2 = data2.getConnection();
33      java.sql.Statement stat2 = conn2.getStatement();
34
35      // Execute update query to both databases
36      stat1.executeUpdate(...);
37      stat2.executeUpdate(...);
38
39      // Commit the transaction
40      // Apply the changes to the participating databases
41      usertx.commit();
42
43      //Release all connections and statements.
44      stat1.close();
45      stat2.close();
46      conn1.close();
47      conn2.close();
```

XA Example Implementation in Java

- Distributed Transaction (cont.)

```java
48          // Catch any type of exception
49          catch (java.lang.Exception e) {
50              try {
51                  e.printStackTrace();
52
53                  // Rollback the transaction
54                  usertx.rollback();
55                  System.out.println("The transaction is rolled back.");
56              } catch(java.lang.Exception ex) {
57                  e.printStackTrace();
58                  System.out.println("Exception is caught. Check stack trace.");
59              }
60          }
61      }
```