Dynamic Configuration Management For Clustered Java EE Application

By Ashish Banerjee, version 0.1, 25May2007

Execute Summary

This technical brief, first describes the dynamic application specific configuration, then describes an use-case scenario. Thereafter, design constrains and solution architecture is described. Two implementations are proposed. Also a programmer's guideline is provided for using the dynamically configurable application parameters.

It is concluded that a JNDI wrapped LDAP store is the best solution, followed by Entity Beans and finally by Java Spaces.

However, all the three implementations are transparent to the application programmer, who uses JNDI abstraction to access the parameters.

Target Audience

Java EE developers.
It is assumed that the reader has working knowledge of Java EE APIs.

Dynamic Configuration Problem Definition

Multiple instances of a Java EE application may be running in a mission critical always-on mode. An application specific configuration parameters may be needed to be modified seamlessly without re-starting any instance of the Java EE application. Also, the change is made once by the authorized administrator and is replicated across all the instances, instantly.

This technical brief does not address the Java EE deployment configuration or hot deployment. It is targeted towards application domain specific configuration parameters, that remain relatively constant during the application lifetime; Yet may be needed to be changed, once in a while, without re-starting the Java EE application. For example, in an issue tracking system, the response time for a SLA (Service Level Agreement) needs to be changed to 20 minutes from the current value to 45 minutes.

Use Case Scenario

Say an enterprise wide medical transcription, for a hospital chain, application is running over a VPN. Multiple instances of the application are running, few instances are clustered for load-balancing and other instances serve different geographical locations across the Globe. The hospital management wishes to change the turn-around time for a transcription from voice to text from current value of 45 minutes to 20 minutes. To implement this policy change, the global application administrator, opens her application console (a graphical user interface, may be browser based), and changes the required parameter.

The change is instantly replicated across all the live instances of the application across the enterprise, without needed to restart any of the application instances.

Design Constrains

The design should be standards based to enable portability across Java EE platform implementations.

The architecture should be scalable. That is, the solution should scale from single Java EE container based deployment in a SME (Small and Medium Enterprise) or in a development environment to large scale clustered deployment for enterprise and global scale operation.

The underlying clustering and replication mechanism must be transparent to the application programmer. And, the system administration can change the replication mechanism by changing deployment descriptor, without needing to change the application source code or re-compiling the code.

Solution Architecture

The solution can be divided into three parts. The first part is to set the application parameters and the second part is implementation mechanism for the underlying dynamically configurable properties. The third being the application using these parameters in its execution life cycle.

The configurable parameters are wrapped as an Interface, and its instance accessed through JNDI context.

First an Interface should be defined specific to the application parameters having a beans pattern. The implementation Class of this Interface is instantiated using JNDI context, once during application bean life cycle. Thereafter, the getter method of the configuration bean is used whenever the parameter is needed. The parameters should not be stored in a local variable.

The underlying implementation is discussed in Implementation Possibilities section, below.

Application Programmer's Guidelines

The configuration parameters being wrapped around an Interface.

The application programmers writing the Java EE components, that use the configuration parameters should always access the parameters through a JNDI context. This will make the application transparent to the underlying implementation.

The parameters must **never** be saved in a variable. Every time the application bean needs to access the configurable parameters, during the execution life cycle, it should always get the parameters through the getter method of the Interface definition.

Thus any changes in the parameters are used by the application, transparently.

Implementation Possibilities

This section considers three implementation possibilities, for the underlying bean that implements the configurable parameters Interface, in the order of preference.

All of them are transparent to the application programmer. It is possible that two distinct deployment of the same application to have one of these choices.

1. LDAP
2. Entity Beans

3. Java Spaces

LDAP

The LDAP is the preferred solution, since LDAP is widely used by enterprise for authentication and it scales very well. Enterprise grade LDAP is widely available as open source products. Java EE containers have good support for LDAP. Thus the bean implementation of the dynamically configurable parameters use LDAP to store their parameters value.

This bean implements the `javax.naming.NamespaceChangeListner` and registers interest in the change event of the LDAP DN (Distinguished Name) containing the name/value properties specific to the application. Thus any change is notified to all the instances and they fetch the new value. Thus the next time the application needs the parameters, it always gets the modified value. See [4] for more information.

Entity Beans

The Entity Beans is the next best solution for the dynamically configurable properties implementation. An Entity Bean implements the configurable Interface. It also registers itself with the JNDI service with the predefined context. The configurable properties are stored in a database.

No extra programming effort is required, as the database replication and Java EE caching mechanism is employed for transparently propagating the changes

Java Spaces

Java Spaces [5] provides a distributed persistence and object exchange mechanism. It has an open source reference implementation. Again the implementation bean uses Java Spaces to listen for changes in the configurable properties.

An innovative paper [1] describes a possible implementation of this technology.

Java Spaces has access issues across trans-routing domains.

UI (User Interfaces)

The UI can be either JMX based or a normal browser based application. The browser based application for configuration management is the preferred solution. Since, this will provide the user with a uniform user experience. Also no special programmer skills are required.

For JMX based UI. The bean implementing the configurable parameters Interface, must conform to MBean design pattern, and the Java EE container must have a MBean service running, see [3] for further details.

Conclusion

The browser based UI is more intuitive and integrated. This should be preferred over JMX console.

The Entity Bean solution is simple and intuitive. However, it depends on an underlying database and its replication capabilities. Also, the Entity cache mechanism of the Java EE container is needed, see [2] for example. However, the solution works out-of-the-box, without any need for programming the underlying implementation.

The Java Spaces based solution does not depend on any database or Java EE container replication and caching capabilities. Thus is implementation independent. Java Spaces is open source. But it may have issues with firewalls and trans-network routing.

The Best option is LDAP through JNDI encapsulation. The second best is Entity Beans. Other implementations like Java Spaces solution should only if the LDAP and Entity Beans solution is ruled out for some reason.

Annotated References

[1] This academic paper describes dynamic configuration management for clustered Java EE using Java Spaces: http://www.dhpc.adelaide.edu.au/reports/103/dhpc-103.pdf

[2] JBOSS clustering : http://www.jboss.com/developers/projects/jboss/clustering

[3] JMX home page : http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/

[4] JNDI home page: http://java.sun.com/products/jndi/

[5] Java Spaces Specification: http://www.sun.com/software/jini/specs/jini1.2html/js-title.html