Session Failover for Java EE Applications – Developer's Guide

Applies to:

SAP NetWeaver Composition Environment 7.1 and higher

SAP NetWeaver Process Integration 7.1 and higher

Application Server Java (AS Java).

Summary

This document outlines the main features of SAP NetWeaver AS Java session failover and provides guidelines for developers on how to enable this functionality for Java EE applications and how to configure their systems for failover.

Author: Vanya Maneva

Company: SAP

Created on: 15 February 2009

Author Bio

Vanya Maneva has been with SAP for 2 years. She is an information developer responsible for the SAP NetWeaver Application Server Java architecture and configuration documentation and the SAP Technology Troubleshooting Guide on SDN.

Table of Contents

Introduction

To answer the needs of business scenarios where hundreds of concurrent users work with multiple applications, the architecture of AS Java provides central session management, which has flexible mechanisms for enhanced control of all kinds of sessions (EJB, HTTP, etc.) and smooth and transparent failover in case of a server crash.

This document provides a general overview of session failover and techniques of Application Server Java (AS Java) and guidelines for developers on how to enable it for Java EE applications.

Glossary

Session – the place where the state of a user accessing an application between several requests is kept. For example, a shopping cart containing certain items the user has picked to buy when visiting an online shop.

Active session – the server process is currently processing a request to this session

Inactive session – the server process is currently not processing a request to this session

Session domain – the basic configurable object in the Session management system used for an abstraction of storage for sessions with similar life cycle.

Session context – a group of similar session domains (for example, HTTP session context contains all HTTP session domains)

Session Failover: Overview

Session failover is a strategy used to improve the robustness of AS Java by protecting user sessions from server failures. This is achieved by separating active from inactive sessions and storing them in a safe session storage.

Usually, at a certain moment, each server process has a number of active sessions (it processes the request for these sessions) and a number of inactive sessions. All server processes in the instance have access to a session storage for inactive sessions. If a problem, such as out-of-memory errors, JVM bugs, and so on, occurs and the server process crashes, this area is not damaged. The active sessions are lost, but all the inactive sessions stored in the session store remain unaffected and can be mapped into a different server process when a request for that session needs to be processed.

AS Java offers three kinds of storage for inactive sessions: **database**, **file system** and **shared memory**.

Of course, session failover comes at the expense of performance. In a standard scenario with shared memory persistence (for example, the pet store use-case used in this document), performance degradation is approximately 3%. The exact overhead depends on your concrete application and scenario so it should be measured.

There are two directions, in which performance can be optimized:

- Optimize the session state as described later in **Developing a Java EE Application with Failover in Mind**
- Choose the type of the persistent storage (database, file system or shared memory) that best fits the needs of your scenario

Session Storage Types

Shared memory based failover provides the best performance of the three ones because of the fast access. It is enabled by the use of SAP's own implementation of shared memory specially adapted for the failover case. The shared memory is common for all server processes in one Java instance. It is, however, only possible to move sessions between server processes of a Java instance and not between Java instances, which does not protect against hardware failure.

If cluster (not just instance) scope is vital for you, you can use the **database storage** type for session failover. Bear in mind that this type of failover has the slowest speed.

File system allows fast storage similar to shared memory. It can be a good solution if you have shared file systems and RAM disks. A shared file system provides a cluster-wide solution, while a normal one is limited within the Java instance.

In conclusion, when choosing the storage type, you have to consider the following factors (also summarized in the table below):

- Shared memory storage is suitable in most cases, provides the best speed and requires very low configuration efforts, but is limited within the Java instance.
- File system storage has no size limitations – storage depends on the size of the file system, but it requires additional configuration and external software.
- Database is the most reliable as it can protect against hardware failures and requires no additional configuration because database is integrated in the AS Java. You can use database storage if you are ready to have a slower failover speed.

	Shared Memory	**File System**	**Database**
Session Failover Speed	fastest	depends on hard disk	slow
Session Failover Scope	Java instance	Java instance (AS Java cluster - if the file system is shared)	AS Java cluster
Configuration Effort	very low	complex	None

Guidelines for Developers of Failover Enabled Java EE Applications

When developing Java EE applications, you have to consider that it is easier to develop a failover enabled application from scratch than to transform an already created one, which was not developed with failover in mind.

Developing a Java EE Application with Failover in Mind

You can make every pure standard based Java EE Application failover safe if you meet the following requirements:

- **The underlying infrastructure must be failover enabled**.

  Usually, business applications use different frameworks and technologies (EJBs, servlets, etc.) You can write a failover enabled application only if the used technologies and frameworks support failover too. In AS Java, the main technologies and containers are failover enabled, such as Web Container; EJB Container; Web Dynpro, etc.

- **All objects stored in the session must be serializable**.

A session, which contains even one non-serializable object cannot be serialized and cannot be stored in the session store, therefore, this session is not failover enabled. Non-serializable objects are most system resources, such as threads, sockets, DB connections, etc. In general, such system resources should not be stored in the session if you can obtain them from other storages at runtime.

To achieve session serializability for objects that are part of a session, consider the following requirements for a serializable class:

- The class itself and all referenced objects must implement the *java.io.Serializable* interface.
- Mark all non-serializable fields as transient (and implement the state to be recreated after failover). Note that field initializers and constructors are not executed for serializable classes during deserialization. This means that the transient fields are initialized to the default value appropriate for its type (null for Objects and 0 for primitive data). Also, if you do any other critical operations in constructors, you should support deserialization as well.
- Have access to the no-arg constructor of its first non-serializable superclass. (*Chapter 1.10 of Java Object Serialization in Java Language Specification*).
- Do not use serialization of inner classes (nested classes that are not static member classes), including local and anonymous classes. Inner classes declared in non-static contexts contain implicit non-transient references to enclosing class instances, serializing such an inner class instance will result in serialization of its associated outer class instance as well.
- We strongly recommend that you do not use methods to control the serialization and deserialization. If possible, do not use the following methods:

  writeObject / readObject

  writeReplace / readResolve

You can find a more detailed description of Java Object Serialization in *Java Language Specification*.

- **It is essential that you keep session size as small as possible.**

The session should contain the complete user state, but no redundant information.

- To minimize the size of persisted session state, avoid caching common data in the session. Such data should be marked as transient because it can be recreated.
- Avoid storing large object graphs in the session as the large object graph results in a lot of metadata in the persistent state.

- Keep the names of objects and attributes short. Since these names are included in the serialized object data, each character you remove from them results in a one-byte reduction in data size.

- **Try to store values instead of references in the session.**
- **Consider delta failover -** the ability to persist only the changed parts/attributes of the session. To minimize the data that is serialized and persisted on each request, only the changed attributes (not the whole session) are serialized. As a result, performance is improved.

  You have to write the application in such a way, that the attributes of the session are separated and in different requests, only some of the session attributes are touched.

  If you need to persist a lot of data in a session state, consider grouping the data into different attributes designed so that when there are different requests to the application, only a minimal number of them will be changed on each request. This could significantly reduce the time for persistence of changed data.

  

  Attribute objects must not have reference to each other, otherwise you cannot fully benefit from delta failover.

  Example:
  In your application code, you have to use a data structure for a shopping cart with some characteristics for pets, such as a Cat and a Dog. Half of your requests use the Cat data, the other half uses the Dog data.
  If you implement your data structure as one shopping cart object **SC**, which contains the Cat data and the Dog data, in you code you have to store this data as one attribute in the session, you cannot benefit from delta failover:

  ```
  session.setAttribute("shopping cart", SC);
  ```

  If you implement the animal data management as separate objects: **A1** – containing the Cat data and **A2** – containing the Dog data, you can gain about a 50% better performance of your requests (depending on the complexity of the A1 and A2 objects):

  ```
  session.setAttribute("Cat data", A1);

  session.setAttribute("Dog data ", A2);
  ```

- **Use the failover simulation mode to test the Java EE applications.**

  

  We recommend that you use the failover simulation mode during development as it shows possible issues right away. Test your whole scenario to make sure that there are no non-serializable objects in the session at any time.

  The failover simulation mode is used for testing, debugging and validation purposes only and is not suitable for productive usage.

  The minimal AS Java cluster installation on which you can test the failover enabled application is: a Java instance with at least two server processes. For every request, the session is moved by the Internet Communication Manager (ICM) to another server process in the Java instance in order to test whether the session is correctly serialized and read from the session store.

  To activate failover simulation mode:

  1. Open the file with system profile configuration located in: *\usr\sap\<SID>\SYS\profile*.
  2. Switch activation property: *icm/HTTP/J2EE/force_server_switch = 1*.

3. Restart the system. After the restart, failover simulation mode is activated for all Java EE applications.
4. Test, trying to cover all possible scenarios of the application. Thus, you will achieve maximum similarity to real productive usage so results will be most reliable.
5. If your applications behave as expected, they have passed the test successfully. To check this, you can search the trace files for any traces left, or, check the state of your Web sessions and storage in the SAP Management Console.

Enabling Session Failover for an Application

You can configure session failover for a single application during development.

In the `application-j2ee-engine.xml` file of the application, add the following entries:

```
<fail-over-enable                                    - to enable failover
         mode="on_request"                           - shows serialization time
         delta="on"                                  - to enable delta failover,
otherwise full failover is enabled
         xsi:type="fail-over-enableType_enable"/>
```

Example:

```
<?xml version="1.0" encoding="UTF-8"?>

<application-j2ee-engine

   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

   xsi:noNamespaceSchemaLocation="application-j2ee-engine.xsd">

   <fail-over-enable

          mode="on_request"

          delta="on"

          xsi:type="fail-over-enableType_enable"/>

</application-j2ee-engine>
```

More information: application-j2ee-engine.xds

Another way to configure session failover for a single application is via the Cofig Tool. You can set a configuration for the particular application following the procedure:

1. Start the Config Tool by double-clicking the *configtool* script file from *<SAP_install_dir>/<system_name>/<instance_name>/j2ee/configtool* directory.
2. Choose *cluster-data*→ *<template>* → *<instance>* → *managers* → *SessionManager*.
3. From the list of properties, select *session.failover.app.config*.

   Set the property as shown in the format:

   <domain path> [storage:STORAGE_TYPE], [mode:MODE_TYPE];

   <domain path> consists of *<session context><%><session domain>*

   For example, if you set the property in the following way:

   HTTP_Session_Context%TestApp storage:file, mode: onRequest;

   you will enable session failover for the Web application named TestApp with storage in the file system and persistency on request.

Enabling Session Failover for a User Scenario

If there are numerous Java EE applications deployed on your system, you may want to configure session failover for a given scenario, without being sure which applications are involved in it. To do this, you can use the session recording feature and you will identify all applications that need to be configured for failover so that your scenario has failover configured as follows:

1. Activate the feature through a telnet command:
    - To add the session group of shell commands, in the command line, enter "**ADD SESSION**".
    - To start the recording, enter "***RECORD [file_name] [session store type]***". S*ession store type can be DB; SHM or file.* For example: `RECORD app_config.txt db`.
2. Play the scenario. All applications participating in the scenario will be automatically marked.
3. Stop recording by entering the telnet command: **"RECORD stop".** An automatically generated configuration string will be displayed in the console and saved in the file.
4. Append the generated configuration string as value of the *session.failover.app.config* property.

Configuring Session Failover for Already Developed Applications

Using the Config Tool, you can enable/disable session failover for a particular application or activate session failover centrally, define where and when to persist sessions, activate delta failover and so on.

4. Start the Config Tool by double-clicking the *configtool* script file from *<SAP_install_dir>/<system_name>/<instance_name>/j2ee/configtool* directory.
5. Choose View →*Expert Mode*.
6. Choose *cluster-data*→ *<template>* → *<instance>* → *managers* → *SessionManager*.
7. From the list of properties below, select the property you want to modify.
8. In the *Custom value* field, set the required value.
9. To apply the new value, choose *Set Custom Value*.
10. To save the changes, choose  *Apply changes*.
11. For the changes to take effect, restart the cluster.

List of Session Manager Properties:

Property	Meaning	Possible values
session.failover.enabled	Enables/disables session failover globally (for all applications). This property overrides the specific configurations for the different applications. Usually used by administrators for mass failover configuration for all applications.	True – failover enabled False – failover disabled If true/false is not set, highest priority has the value Not set – all other configurations apply
session.user_context.persistency	Defines the persistent storage of user context.	DB – stored in the database SHARED_MEMORY – in the shared memory
session.persistent.storage	Defines where you want to store sessions. This is a default property, which can be overridden by session.failover.app.config.	DB – stored in the database SHARED_MEMORY – in the shared memory FILE – on the file system
session.persistent.mode	Specifies when you want to persist sessions.	ON_REQUEST – session is stored when it passes from active to inactive state ON_THREAD_END - session is stored after the request processing thread finishes ON_SHUTDOWN - session is persisted only when the server shuts down
session.failover.app.config	Defines session failover ability and storage on context, domain (application), etc. level.	format: <domain path> [storage:STORAGE_TYPE], [mode:MODE_TYPE]; <domain path> *<domain path>* consists of *<session context><%><session domain>*

atttribute.delta.failover	Triggers delta failover. Only the changed attributes are persisted, thus increasing the performance. Attribute objects must not have reference to each other. This is a default property.	True – delta failover enabled False – delta failover disabled
mark.get.chunk	Applicable only when delta failover is enabled. Set to true if an attribute is marked as changed on *getattribute*, not only on *putattribute.*	True – the attribute is persisted on *getattribute* and *putattribute* False – the attribute is persisted only on *putattribute*

The configuration priority of the above mentioned properties is the following (starting with the highest):

1. session.failover.enabled
2. session.failover.app.config
3. failover-enable in the application-j2ee-engine.xml file of the application

Additional Configuration Settings

If you have enabled session failover and chosen shared memory as the type of session storage, you can also configure the shared memory size.

1. Open the Config Tool.
2. Choose the instance you want to configure.
3. Select the *VM parameters* tab.
4. In the *Memory* tab, choose *Enable* for parameter *globalArea*. Create a new custom parameter and eventually configure the size (in megabytes) of the shared memory store (depending on number of users, session size). The size to be set should be approximately equal to the number of expected clients multiplied by the average size of your application sessions.
5. Restart AS Java.

Conclusion

In conclusion, software developers, quality assurance specialists and system administrators can easily utilize the new advanced session failover capabilities of AS Java. The infrastructure offers capabilities for configuration and choice of failover strategies and storage types depending on your use case, as well as additional tools (such as Telnet commands), which enable appropriate configuration and debugging. Of course, session failover is paid in performance, but you can control this price by carefully choosing the type of the persistent storage, considering the size and complexity of the user data that is put in the session and the number of the applications in the failover safe scenario.

Related Content

AS Java Cluster Architecture

Adding and Removing Server Processes

Java Object Serialization Specification by Sun Microsystems

Copyright