

iternum

EJB DEVELOPMENT

Author Karl Banke

iternum GmbH, Frankfurt

An iternum standards document

„Easing the development process“

iternum

Inhaltsverzeichnis

LIZENZ/LICENSE 2

Benutzung und Weitergabe 2

Markenrechte 2

Haftungsausschluss 2

Usage and Distribution 2

Copyrights 2

No Warranty 2

SUMMARY 1

EJB DESIGN PATTERN 2

Finding errors at compile time 2

Reduce network traffic 3

Super classes 4

Client factory 5

Naming conventions 6

Package structure 6

BUILD AND DEPLOYMENT 7

Installation of BEA Weblogic 7

Creating directory structure 7

Adjust entries 8

Filesystem structure 8

Automated build process 9

Building EJBs using ant 9

iternum

Lizenz/License

Benutzung und Weitergabe

Dieses Dokument enthält interne Standards der Iternum GmbH, Frankfurt. Es wird ohne jedliche Gewährleistung und kostenfrei zur Verfügung gestellt. Dieses Dokument darf kopiert und weitergegeben werden. Dabei muss die Quelle des Dokuments angegeben werden. Soweit Inhalte verändert worden sind, ist dies deutlich auszuweisen.

Markenrechte

Diese Dokument enthält Hinweise auf Produkt- und Firmennamen anderer Firmen. Die verwendeten Begriffe sind Warenzeichen und/oder eingetragene Warenzeichen der entsprechenden Firmen.

Haftungsausschluss

Bei der Zusammenstellung der in diesem Dokument enthaltenen Informationen wurde größte Sorgfalt angewandt. Dennoch kann die iternum GmbH nicht für die Korrektheit garantieren. Enthaltene Angaben können ohne Ankündigung geändert werden, wir gehen damit keinerlei Verpflichtungen ein. In keinem Fall kann iternum für etwaige Schäden irgendwelcher Art verantwortlich gemacht werden, die durch die Benutzung oder im Zusammenhang mit der Benutzung der hier bereitgestellten Informationen entstehen, seien es direkte oder indirekte Schäden, Folge-Schäden oder Sonderschäden einschließlich entgangenen Gewinns, oder Schäden, die aus dem Verlust von Daten entstehen.

Usage and Distribution

This document contains internal standards of iternum GmbH, Frankfurt. It is provided free of charge and with no warranty whatsoever. This document may be distributed and copied. The origin of the document must be clearly stated. If content has been changed this must be clearly indicated.

Copyrights

This document contains references to products and company names of third parties. The names used are registered trademarks and/or copyrights of the respective companies.

No Warranty

Compiling the information contained in this document has been performed with extreme care. However, the iternum GmbH can not guarantee for the correctness of this information. Any content can be changed without notice. In no case can any claims be made to iternum GmbH or the author of the document for any losses, that result from usage or due to usage of the information supplied in the document. This includes direct or indirect losses, losses including loss of data or loss of profit.

Summary

This document summarizes best practices and approaches for EJB development that have been used by the author in several projects. It provides guidelines for EJB usage and introduces a design pattern and utility framework that have been proven to ease both EJB development and understanding of EJB application written by others.

The document also contains some special information regarding the configuration and usage of the BEA Weblogic EJB Server, as of version 5.1.

EJB Design Pattern

This section introduces a design pattern that can be employed in ejb development. The motivation to use this pattern is

- Finding implementation errors at compile time
- Reduce network load
- Move common tasks to superclasses.
- Provide factories for easy client access.
- Easy deployment.

Finding errors at compile time

Any EJB implementation needs to implement all methods defined in the remote interface. Deployment of a bean is a two step process. First, the classes are compiled using the relevant jdk, then compilation is performed using an EJB compiler that is usually provided by the EJB server vendor.

There are frequent errors, because the methods defined in the remote interface are not implemented fully in the bean implementation. However, the EJB specification states that the remote interface should not implement the remote interface directly. This is because the remote interface contains methods inherited from `javax.ejb.EJBObject` that are called only by the EJB container.

The design pattern is discussed using a stateless session bean. It uses a common business interface to provide consistency between the bean implementation and the remote interface.

This common interface will be called `UserSessionInterface`,

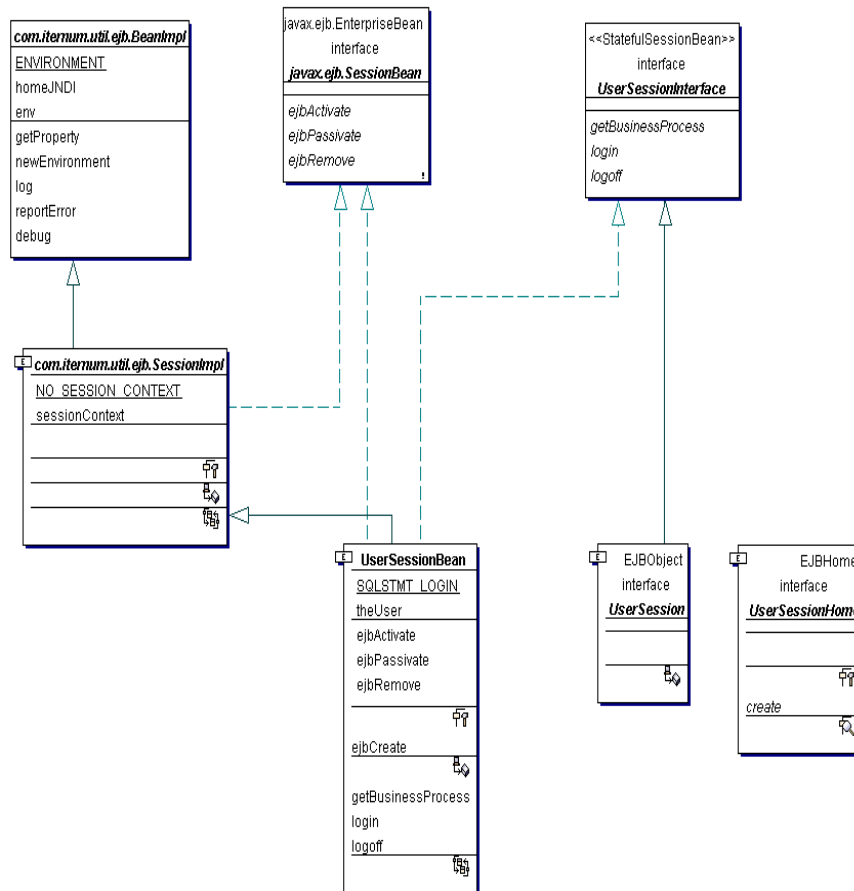
```
package com.iternum.util.ejb.session;

import java.rmi.RemoteException;

public interface UserSessionInterface
{
    ...
}
```

Since this interface is to be implemented by a remote interface, all its methods must declare a `RemoteException`. Implementation classes can choose to declare these exceptions, since remote exceptions should only be created by the container itself. The common interface guarantees that inconsistencies between remote interface and implementation method signatures can be found at compile time.

iternum



Reduce network traffic

Any access to `ejb` methods implies a client-server communication. To minimize network traffic it is advisable to keep the number of method calls as well as the amount of data transferred at a minimum.

This is accomplished by using serialisable, coarse grained „Container Objects“ that are passed „by value“, for example a class `UserData` might be passed rather than calling multiple methods on a `User` EJB interface. The entity bean model can be leveraged by putting the business logic to read and write the data object into the entity bean itself.

This has the disadvantage that no automatic synchronization between object and data-source is available, i.e. the data is not synchronised. This is frequently irrelevant but it might be necessary to synchronise by re-retrieving the data from the server anyway.

The transferred data objects itself should be as compact and simple as possible to avoid any unnecessary traffic by passing redundant or unused data.

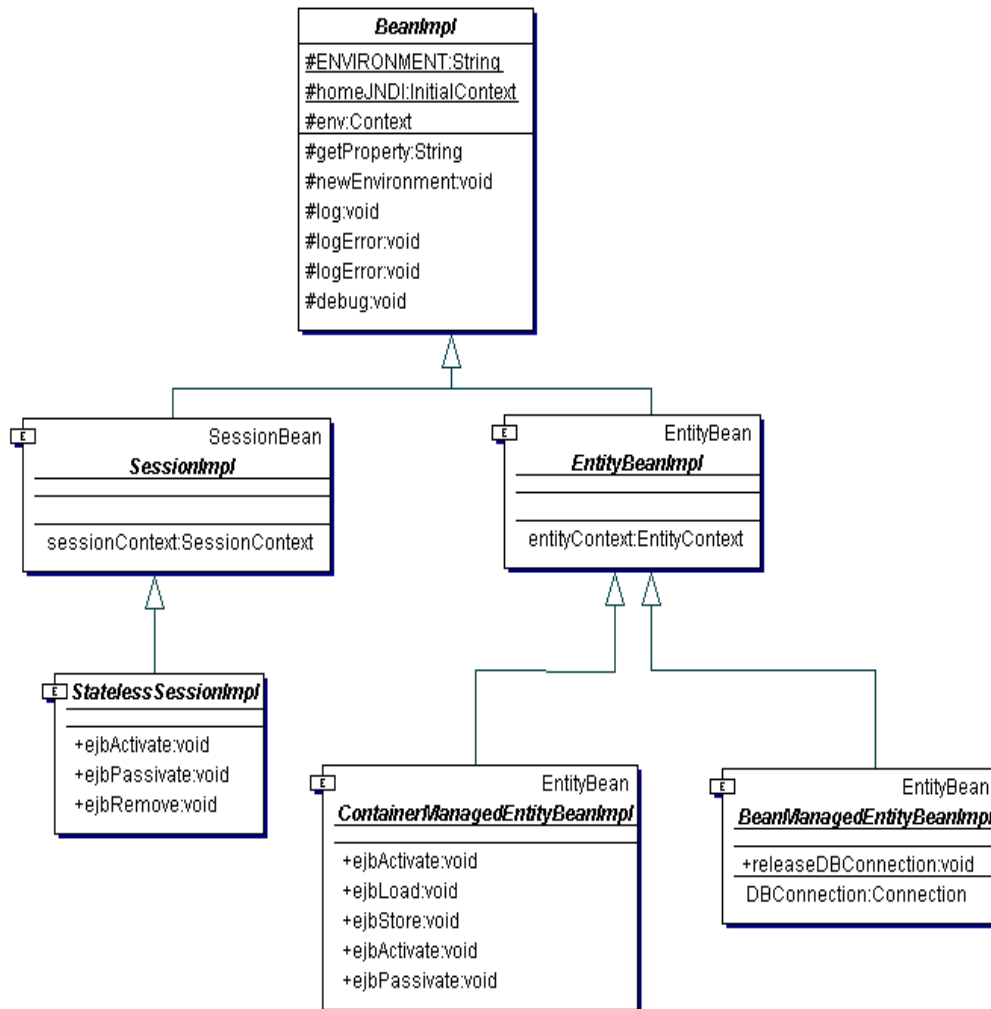
iternum

Super classes

To minimise recurring implementation work it is good practice to put common functionality in EJB superclasses.

Some methods defined in the EJB framework are either never called by the container for certain implementations or an empty implementation will be sufficient. For example, the `ejbActivate()` and `ejbPassivate()` methods of `javax.ejb.SessionBean` are never called for stateless session beans.

Superclasses used for EJBs can be found in package `com.iternum.util.ejb`. They may also offer hooks into common functionality, like logging or parameter retrieval from the deployment descriptor or some other type of configuration file. The superclass for beans that use bean managed persistence contains methods for managing the database connection.



Client factory

Each EJB provides a factory to easily retrieve a reference to the bean or to its home interface. The exact structure of the factory might differ due to the security policy in place.

For the client it can be an advantage to preserve the initial context if no authentication is required. If the access to a bean or parts thereof requires authentication the user can pass an existing context or create a new one with username and password. Since authentication is a certain overhead it should not be used in each and every EJB access.

Factory classes for EJBs using authentication

A factory for EJBs with authentication creates or receives a context, performs a lookup to the home interface and returns either the remote interface or, in case of entity beans, the home interface itself.

```
public static Manager getManager(javax.naming.Context
context) throws NoSuchObjectException {
    // do something
}
public static Manager getManager(String username, String
password) throws NoSuchObjectException {
    // do something
}
```

Factory classes for EJBs without authentication

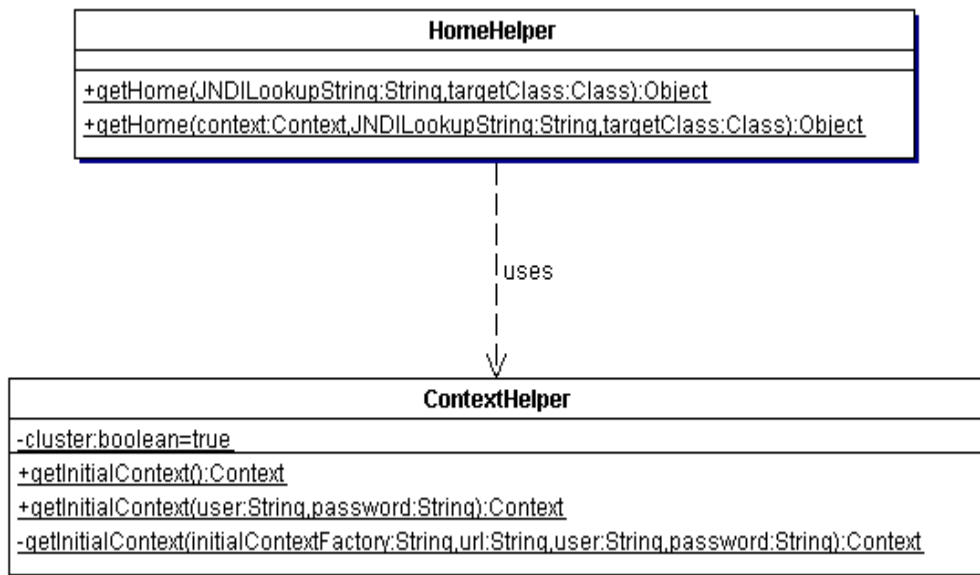
These use a default context to retrieve the EJB.

```
public static SomeManager getSomeManager() throws
NoSuchObjectException {
    // do something
}
```

If no clustering is used, the factory can store a reference to the home interface that can be reused without the need for subsequent lookup. References of the remote interface itself should never be stored in the factory.

Lookup of the home interface and creation of a context are performed using utility classes. They can also act as facades that hide clustering of the server.

iternum



Naming conventions

Naming conventions in the design pattern.

- The remote interface describes the functionality or purpose of the bean `<EJBType>`
- The common business interface appends „Interface“ `<EJBType>Interface`
- The bean implementation appends „Bean“ `<EJBType>Bean`
- The home interface appends „Home“ `<EJBType>Home`
- Transport objects usually append „Data“ `<EJBType>Data`
- The factory classes for accessing the ejb append „Factory“ `<EJBType>Factory`

All other naming and coding conventions are according to the Java Coding Conventions [3].

Package structure

All Classes that are used by both, client and server (remote interface, home interface, support classes) are located in a package `<ejbtype>`.

The implementation class is located in a package `<ejbtype>.server`.

The factory class for accessing the remote instance is located in a package `<ejbtype>.client`.

Build and deployment

This paragraph describes the build process used for assembling an EJB/JSP/Servlet application during development. Parts of the process described are specific to the BEA Weblogic server as of version 5.1.

The areas covered are

- Installation of BEA weblogic server
- Filesystem structure for source and jsp
- Automated built using ant

Installation of BEA Weblogic

Installation of BEA Weblogic works basically „out of the box“ using the installation tools provided. However it is advisable to work in a directory outside the provided installation in order to avoid interference with the original installation and to ease the process of upgrading to a newer server version.

Creating directory structure

This requires copying some files and directory structures to a new directory. The directories will be referenced as follows:

- `$WL_HOME` the original installation
- `$WL_DEV` the new „development instance“

The following files need to be copied to `$WL_DEV` for a Unix installation.

- `weblogic.properties`
- `weblogic.policy`
- `setEnv.sh`
- `startWeblogic.sh`

The following directories need to be copied along with all included files

- `servletimages`
- `license`

The following directory needs to be created in `$WL_HOME`

- `myserver`

This directory is the actual server directory. All class files, ejbs and web pages will go in this directory. The following directories will be created

- `myserver/serverclasses` for classes used by the server and the servlets that do not require dynamic reloading
- `myserver/clientclasses` for classes used by the client of the server

iternum

- `myserver/servletclasses` for classes that comprise servlets and that require dynamic reloading
- `myserver/ejb-jars` for placement of `ejb-jar` deployment units

It is advisable to recursively copy the `public_html` directory to `myserver`, in order to have access to some default pages for testing purposes. Some other files and directories in `myserver` are needed or created by the server. These include

- Log Files (`weblogic.log`, `access.log`)
- Directories for temporary storing the compiled classes (`tmp_deployments`, `classfiles`).

Adjust entries

In order to pick up files from the right location some entries in `weblogic.properties` and `weblogic.policy` and in the script files need to be adjusted. In `weblogic.policy` access must be granted to the `$WL_DEV` directory.

In the `weblogic.properties` file this affects at least the following entries

- `weblogic.system.HelpPageURL`
- `weblogic.httpd.servlet.classpath`
- `workingDir`

In the scripts this mainly affects the classpaths to include the `serverclasses`, `clientclasses` and `servletclasses` directories in the right points in the classpath. In addition the path for any deployed `ejb` must of course be set accordingly.

Filesystem structure

The filesystem for development is structured in a certain way to ease automated build and deployment using the `jakarta ant` build tool. This tool takes care of compiling source code into java classes, assemble and compile `ejb jar` files and copy various classes to another location based on patterns in the name of the path or the file.

The development root directory `$DEV_ROOT` must be outside the development installation and should be included in some source code control system, like `cvs` or `perforce`. It needs to have the following structure.

- `$DEV_ROOT/src` Contains the Java source code
- `$DEV_ROOT/classes` Contains the compiled Java classes
- `$DEV_ROOT/dd` Contains the deployment descriptors
- `$DEV_ROOT/public_html` Contains all JSP pages, HTML pages, images and stylesheets.
- `$DEV_ROOT/sql` Contains database initialisation script

The deployment descriptors for all `ejbs` are placed in the `dd` directory. One descriptor set per `ejb` must be used to allow for automated build using `ant`. The descriptors need to follow a

naming convention where they are prefixed with a common name per ejb, e.g. for bean address that uses container managed persistence:

- `person-ejb-jar.xml`
- `person-weblogic-ejb-jar.xml`

- `person-weblogic-cmp-rdbms-jar.xml`

Automated build process

The automated build process is performed using the jakarta ant build tool as of version 1.4alpha. It defines a number of interdependent tasks similar to make and allows to restrict build to the parts of a project that have changed, only.

Ant can be downloaded from <http://jakarta.apache.org/ant/index.html>. It provides extensive installation instructions and instruction for use. An ant project consists of a number of tasks that together comprise the build and deployment process. For ejb / jsp / servlet development at least the following tasks should be defined.

- Compile all Java source files
- Copy the relevant files to the `serverclasses`, `servletclasses` and `clientclasses` directories
- Build all ejbs together and deploy the ejbs to the server.
- Copy all files in `public_html` in the respective server directory.

Additional tasks may include automated checkout and test runs, creation of WAR files etc.

Building EJBs using ant

To build an ejb using ant the following needs to be provided.

- The path where to search for class files
- The path where to search for deployment descriptors
- The path where to put the deployment jar
- External to ant a proper classpath must be set using weblogics `setEnv.sh`

A typical ant task that scans the directory `dd` for deployment descriptors, the directory `${classdir}` for class files needed for the ejb and creates a jar file for each ejb in `temp` looks like this.

```
<target name="personjar">
  <ejbjar descriptor="dd"
          srcdir="${classdir}"
          >
    <weblogic destdir="temp"
              classpath="${wl_classpath}" keepgeneric="true">
    </weblogic>
  <exclude name="*weblogic*" />
</target>
```

iternum

</ejbjar>
</target>