First JBoss Seam tutorial by laliluna.de

Table of Contents

Introduction

This tutorials shows how to setup a application using JBoss Seam, JSF and Facelets. I will show a configuration for Tomcat and the JBoss Application Server. Furthermore, you can learn how to use Maven to define the dependencies.

Version

Tutorial version 1.0 by Sebastian Hennebrueder

Library versions

- Seam 2.0.1.GA
- Tomcat 6
- JBoss Application Server 4.2 (not 4.0)

Download

PDF of this tutorial

http://www.laliluna.de/download/first-jboss-seam-en.pdf

Example project

http://www.laliluna.de/download/first-jboss-seam-source.zip

Marketing

Advanced Training and Education

Range of advanced trainings on Hibernate, EJB3, JBoss Seam, Design Patterns, and Enterprise Best Practices

The training can be customized to your needs.

Support, consulting and development

Get the amount of help you need by experienced trainers and developers. An hour of support can save you a lot of time, Code and Design Reviews insures that the best practices are being followed! Reduce solving and testing time. We can implement a part of your application, create prototypes or proof of concepts.

Setup the project

Set up a web project with your preferred IDE. We will use a WAR archive for deployment.

Seam configuration files

We need the following configuration files.

components.xml

This file configures Seam and our own components. I prefer to configure my components with annotations and not in this file but still we can do it here as well.

Place: WEB-INF folder in a WAR archive, META-INF folder in a JAR. We have a WAR archive.

Just tell Seam that we don't want it to manage transactions (at the moment).

```
<?xml version="1.0" encoding="UTF-8"?>
<components xmlns="http://jboss.com/products/seam/components"
  xmlns:core="http://jboss.com/products/seam/core"
  xmlns:persistence="http://jboss.com/products/seam/persistence"
  xmlns:security="http://jboss.com/products/seam/security"
  xmlns:drools="http://jboss.com/products/seam/drools"
  xmlns:web="http://jboss.com/products/seam/web"
  xmlns:mail="http://jboss.com/products/seam/mail"
  xmlns:transaction="http://jboss.com/products/seam/transaction"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
      "http://jboss.com/products/seam/core
 http://jboss.com/products/seam/core-2.0.xsd
       http://jboss.com/products/seam/persistence
 http://jboss.com/products/seam/persistence-2.0.xsd
       http://jboss.com/products/seam/security
 http://jboss.com/products/seam/security-2.0.xsd
       http://jboss.com/products/seam/web
 http://jboss.com/products/seam/web-2.0.xsd
       http://jboss.com/products/seam/drools
 http://jboss.com/products/seam/drools-2.0.xsd
       http://jboss.com/products/seam/mail
 http://jboss.com/products/seam/mail-2.0.xsd
       http://jboss.com/products/seam/transaction
 http://jboss.com/products/seam/transaction-2.0.xsd
       http://jboss.com/products/seam/components
 http://jboss.com/products/seam/components-2.0.xsd">

  <core:init transaction-management-enabled="false"/>
  <transaction:no-transaction/>

</components>
```

seam.properties

Includes settings which can be used in components.xml. Apart from this, it is a marker for Seam that a archive (WAR or JAR) includes Seam components. So just create an empty file.

Place: root source folder (same folder as your Java classes.

pages.xml

Includes navigation rules and is comparable to the JSF *faces-config.xml*. In addition we could add security rules as well.

Place: WEB-INF of a war project

```
<?xml version="1.0" encoding="UTF-8"?>
<pages xmlns="http://jboss.com/products/seam/pages"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://jboss.com/products/seam/pages
 http://jboss.com/products/seam/pages-2.0.xsd"
   no-conversation-view-id="/home.xhtml"
   login-view-id="/login.xhtml">

 <page view-id="*">
  <navigation>
   <rule if-outcome="home">
    <redirect view-id="/home.xhtml"/>
   </rule>
  </navigation>
 </page>

 <exception>
  <redirect view-id="/error.xhtml">
   <message>Unexpected error, please try again</message>
  </redirect>
 </exception>

</pages>
```

faces-config.xml

Normal JavaServer Faces configuration file. The default rendering of Seam is JavaServer Faces, this is why we need this config file.

We define default and supported languages and the prefix of the resource bundle.

This example will use facelets for rendering. This is why we have to define the facelets view handler in this configuration file.

Do not place JSF navigation rules here as with Seam we will use the pages.xml for navigation.

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

```
<faces-config version="1.2"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
 http://java.sun.com/xml/ns/javaee/web-facesconfig_1_2.xsd">
  <application>
    <message-bundle>messages</message-bundle>
    <locale-config>
 <default-locale>en</default-locale>
 <supported-locale>en</supported-locale>
    </locale-config>
    <view-handler>com.sun.facelets.FaceletViewHandler</view-handler>
  </application>
</faces-config>
```

web.xml

Place: well it is where it always should be in a webapp

Defines Seam and JSF specific settings, filters and servlets.

```
<?xml version="1.0" ?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://java.sun.com/xml/ns/javaee"
 xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
 http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
 id="WebApp_ID" version="2.5">

   <!-- Seam -->
   <listener>
      <listener-class>org.jboss.seam.servlet.SeamListener</listener-class>
   </listener>
   <filter>
        <filter-name>Seam Filter</filter-name>
        <filter-class>org.jboss.seam.web.SeamFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>Seam Filter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

  <!-- Facelets development mode (disable in production) -->
   <context-param>
      <param-name>facelets.DEVELOPMENT</param-name>
      <param-value>true</param-value>
   </context-param>

   <!-- JSF -->
   <context-param>
      <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
      <param-value>client</param-value>
   </context-param>
   <context-param>
```

```
      <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
      <param-value>.xhtml</param-value>
   </context-param>
   <servlet>
      <servlet-name>Faces Servlet</servlet-name>
      <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
      <load-on-startup>1</load-on-startup>
   </servlet>
   <servlet-mapping>
      <servlet-name>Faces Servlet</servlet-name>
      <url-pattern>*.seam</url-pattern>
   </servlet-mapping>
</web-app>
```

Add libraries

We start with a minimal configured JBoss Seam. As a consequence with this library set you cannot use Seam Email, Seam PDF, ...

JBoss 4.2 provides already the jsf-api and jsf-impl libraries. Don't deploy them. Tomcat or Jetty will need them of course.

Manually

Copy the following libraries to the WEB-INF lib folder. You can find them in the lib folder of your JBoss Seam download.

- dom4j-1.6.1-jboss.jar
- commons-beanutils-1.7.0.jar
- hibernate-validator-3.0.0.ga.jar
- javassist-3.3.ga.jar
- jboss-el-2.0.1.GA.jar
- jboss-seam-2.0.1.GA.jar
- jboss-seam-ui-2.0.1.GA.jar
- jsf-api-1.2_04-p02.jar
- jsf-facelets-1.1.11.jar
- jsf-impl-1.2_04-p02.jar
- jta-1.1.jar
- persistence-api-1.0.jar

Maven

Alternatively, you may use Maven. I like it just for managing my libraries. We need a JBoss repository. My IntelliJ project deployed unintentionally the el-api.jar which was loaded by Maven. This provokes a problem on Tomcat 6 and JBoss 4.2. Just make sure that it is not deployed or exclude it.

Here is the pom.xml file we need:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>de.laliluna</groupId>
  <artifactId>seamWeb</artifactId>

  <packaging>war</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>seamWeb</name>
  <url>http://maven.apache.org</url>

  <build>
    <finalName>seamWeb</finalName>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>1.5</source>
          <target>1.5</target>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-idea-plugin</artifactId>
        <configuration>
          <downloadSources>true</downloadSources>
          <downloadJavadocs>true</downloadJavadocs>
        </configuration>
      </plugin>
    </plugins>

  </build>
  <repositories>
    <repository>
      <id>repository.jboss.org</id>
      <url>http://repository.jboss.org/maven2</url>
      <snapshots>
        <enabled>true</enabled>
      </snapshots>
    </repository>
  </repositories>

  <dependencies>

    <!-- JBoss Seam minimal -->
    <dependency>
      <groupId>org.jboss.seam</groupId>
      <artifactId>jboss-seam</artifactId>
      <version>2.0.1.GA</version>
      <scope>compile</scope>
    </dependency>
    <dependency>
      <groupId>org.jboss.seam</groupId>
      <artifactId>jboss-seam-ui</artifactId>
      <version>2.0.1.GA</version>
      <scope>compile</scope>
    </dependency>
    <!-- Seam makes use of Hibernate validator and sadly depends on JTA and JPA
even if we don't use it -->
    <dependency>
      <groupId>javax.transaction</groupId>
      <artifactId>jta</artifactId>
      <version>1.1</version>
    </dependency>
    <dependency>
      <groupId>javax.persistence</groupId>
      <artifactId>persistence-api</artifactId>
      <version>1.0</version>
    </dependency>
    <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-validator</artifactId>
      <version>3.0.0.ga</version>
    </dependency>

    <!-- JSF and facelets -->
    <dependency>
      <groupId>javax.faces</groupId>
      <artifactId>jsf-api</artifactId>
      <version>1.2_04-p02</version>
    </dependency>
    <dependency>
      <groupId>javax.faces</groupId>
      <artifactId>jsf-impl</artifactId>
      <version>1.2_04-p02</version>
    </dependency>
    <dependency>
      <groupId>com.sun.facelets</groupId>
      <artifactId>jsf-facelets</artifactId>
      <version>1.1.11</version>
    </dependency>


    <!-- I like to have log4j without Hibernate as well -->
    <dependency>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
      <version>1.2.14</version>
      <scope>compile</scope>
    </dependency>

  </dependencies>
```

```
</project>
```

Hello world

Create a index.jsp in the root folder. It will redirect us to the hello page.

```
<% response.sendRedirect("hello.seam"); %>
```

Create the facelets page *hello.xhtml*.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:s="http://jboss.com/products/seam/taglib"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:ui="http://java.sun.com/jsf/facelets">
  <head>
    <title>Hello world</title>
  </head>
  <body>
  <p>Welcome to your first Seam application</p>

  </body>
</html>
```

Deploy your application to Tomcat or JBoss and test it. With my project name, I used the following URL

http://localhost:8080/seamWeb/ [???]

Writing a Seam component

The next step is to improve the hello page. Our application will deal with hedgehogs and the first thing to be done is to print a list of hedgehogs.

Create a class *Hedgehog*.

```
package de.laliluna.seam;

public class Hedgehog {

  private String name;

  public Hedgehog() {
  }

  public String getName() {
    return name;
  }
  @Override
  public String toString() {
    final StringBuilder sb = new StringBuilder();
    sb.append("Hedgehog");
    sb.append("{name='").append(name).append('\'');
    sb.append('}');
    return sb.toString();
  }

  public void setName(String name) {
    this.name = name;
  }

  public Hedgehog(String name) {
    this.name = name;
  }
}
```

Create a class *HedgehogService*. It is a Seam component. The *@Name* annotation marks it as component and defines the name. In addition we can define the context where this component is placed in with the *@Scope*.

We will see later that we can reference the component using the name. Our service class provides a data model for a JSF data table - *hedgehogs*.

```
package de.laliluna.seam;

import org.jboss.seam.annotations.Name;
import org.jboss.seam.annotations.Scope;
import org.jboss.seam.annotations.datamodel.DataModel;
import org.jboss.seam.ScopeType;
import java.util.List;
import java.util.ArrayList;

@Name("hedgehogService")
@Scope(ScopeType.SESSION)
public class HedgehogService {

  @DataModel
  private List<Hedgehog> hedgehogs;

  public HedgehogService() {
    hedgehogs = new ArrayList<Hedgehog>();
    hedgehogs.add(new Hedgehog("Holger"));
    hedgehogs.add(new Hedgehog("Pete"));
    hedgehogs.add(new Hedgehog("Sebastian"));
  }

  public String getHello() {
    return "Your hedgehogService says hello";
  }
}
```

Finally we will add some messages to the English resource bundle.

Create a file *messages_en.properties* in the src folder.

```
hello=Welcome to your Seam application
hedgehog.name=Name
```

Improve the *hello.xhtml* to print the hedgehogs, messages from our resource bundle and the *getHello* method.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
        PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:s="http://jboss.com/products/seam/taglib"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:ui="http://java.sun.com/jsf/facelets">
<head>
  <title>Hello</title>
</head>
<body>
<p>#{messages.hello}</p>

<p>#{hedgehogService.hello}</p>

<h:dataTable value="#{hedgehogs}" var="h">
  <h:column>
    <f:facet name="header">#{messages['hedgehog.name']}</f:facet>
    #{h.name}
  </h:column>
</h:dataTable>
</body>
</html>
```

That's it. Test your application again.

Tips for debugging

When your application starts up it will print all components being created (Seam and your own) and the context they are placed in. Validate that your new component is displayed as well.

Seam has a debug page showing useful information.

Add the library *jboss-seam-debug.jar* in the WEB-INF folder. Alternatively add the following dependency to your *pom.xml*:

```
 <dependency>
      <groupId>org.jboss.seam</groupId>
      <artifactId>jboss-seam-debug</artifactId>
      <version>2.0.1.GA</version>
    </dependency>
```

In the *components.xml* activate the debug mode.

```
  <core:init transaction-management-enabled="false" debug="true"/>
```

Open your browser and point to the following URL to see the debug page.

http://localhost:8080/seamWeb/debug.seam

Add Hibernate

In this step we will leverage our application to load hedgehogs from the database.

In this example, I will use a PostgreSQL database. To change the database, just change the hibernate.cfg.xml settings to your database and replace the library.

Add libraries

In addition we need the following libraries.

- antlr-2.7.6.jar
- asm-1.5.3.jar
- commons-beanutils-1.7.0.jar
- asm-attrs-1.5.3.jar
- hibernate-3.2.5.ga.jar
- commons-collections-2.1.1.jar
- c3p0-0.9.1.2.jar
- postgresql-8.2-507.jdbc3.jar
- backport-util-concurrent-3.0.jar
- commons-logging-1.0.4.jar
- hibernate-annotations-3.3.0.ga.jar
- hibernate-commons-annotations-3.3.0.ga.jar
- ehcache-1.3.0.jar
- cglib-2.1_3.jar
- jsr107cache-1.0.jar

If you prefer Maven, here are the additional dependencies.

```
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate</artifactId>
  <version>3.2.5.ga</version>
  <scope>compile</scope>
</dependency>
<dependency>
```

```
    <groupId>c3p0</groupId>
    <artifactId>c3p0</artifactId>
    <version>0.9.1.2</version>
    <scope>compile</scope>
  </dependency>
  <dependency>
    <groupId>net.sf.ehcache</groupId>
    <artifactId>ehcache</artifactId>
    <version>1.3.0</version>
  </dependency>
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-annotations</artifactId>
    <version>3.3.0.ga</version>
    <scope>compile</scope>
  </dependency>
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-commons-annotations</artifactId>
    <version>3.3.0.ga</version>
  </dependency>
  <!-- other -->
  <dependency>
    <groupId>postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>8.2-507.jdbc3</version>
    <scope>compile</scope>
  </dependency>
```

Configuration

We need a Hibernate configuration file *hibernate.cfg.xml* in the root source folder.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
          "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
          "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
<session-factory>
 <!--  postgre SQL configuration-->
 <property name="connection.url">
  jdbc:postgresql://localhost:5432/learninghibernate
 </property>
 <property name="connection.username">postgres</property>
 <property name="connection.password">p</property>
 <property name="connection.driver_class">
 org.postgresql.Driver
 </property>
 <property name="dialect">
  org.hibernate.dialect.PostgreSQLDialect
 </property>
 <property name="c3p0.max_size">4</property>
 <property name="c3p0.min_size">1</property>

 <property name="cache.provider_class">
  org.hibernate.cache.EhCacheProvider
 </property>
 <property name="transaction.factory_class">
  org.hibernate.transaction.JDBCTransactionFactory
 </property>
  <property name="transaction.flush_before_completion">true</property>
  <property name="connection.release_mode">after_statement</property>
  <property name="hbm2ddl.auto">update</property>
 <property name="default_batch_fetch_size">4</property>
 <mapping class="de.laliluna.seam.Hedgehog" />
</session-factory>
</hibernate-configuration>
```

In the *components.xml* we have to change the transaction handling. The factory is just a work around for a naming problem. We cannot inject a Hibernate session with the name *session* into the transaction, but

I wanted to use this component name in my application.

```
<!--
  <core:init transaction-management-enabled="false" debug="true"/>
  <transaction:no-transaction/>
  -->

  <core:init debug="true"/>
  <persistence:hibernate-session-factory name="hibernateSessionFactory"/>
  <persistence:managed-hibernate-session name="xsession"
                                         auto-create="true"

 session-factory="#{hibernateSessionFactory}"/>
  <transaction:hibernate-transaction session="#{xsession}"  />
  <factory name="session"
           scope="STATELESS"
           auto-create="true"
           value="#{xsession}"/>
```

Code

We must add the loading and saving of hedgehogs to the *HedgehogService* class.

First we inject the component session into our class.

Then we define that the Data model has a factory method which is loading our hedgehogs.

I changed the Scope to conversation so that the hedgehogs are reloaded when we create a new hedgehog.

```
@Name("hedgehogService")
@Scope(ScopeType.CONVERSATION)
public class HedgehogService implements Serializable {
  @In
  private Session session;
  @DataModel
  private List<Hedgehog> hedgehogs;

  @Factory("hedgehogs")
  public void loadHedgehogs() {
    hedgehogs = session.createCriteria(Hedgehog.class).list();
  }

  public void createHedgehog() {
    session.save(new Hedgehog("Name " + new Date()));
  }
```

The *Hedgehog* class needs the Hibernate specific annotations.

```
@Entity
public class Hedgehog implements Serializable{

  @Id
  @GeneratedValue(strategy = GenerationType.SEQUENCE)
  private Long id;

// getter and setter are omitted
```

Frontend

Finally we need to improve our hello.xhtml. We add an action link to create new hedgehogs.

```
<s:link action="#{hedgehogService.createHedgehog}">Create a hedgehog</s:link>

<s:fragment rendered="#{empty hedgehogs}">
  No hedgehogs
</s:fragment>
<h:dataTable value="#{hedgehogs}" var="h" rendered="#{! empty hedgehogs}">
  <h:column>
    <f:facet name="header">#{messages['hedgehog.name']}</f:facet>
    #{h.name}
  </h:column>
</h:dataTable>
```

Conclusion

It is quite simple to create a web application with the JBoss Seam framework. Though the framework is intended to be an integration framework for JSF and EJB3, we can easily use it with plain Pojo classes on a Tomcat application server. We even get transaction handling for free.

The advantage of creating the project manually is that we only have required libraries instead of the full package.