Getting Started Developing Applications Guide

Table of Contents

This guide has moved to https://github.com/wildfly/quickstart/blob/10.x/guide/Introduction.asciidoc

1 Introduction

This page has moved to https://github.com/wildfly/quickstart/blob/10.x/guide/Introduction.asciidoc

2 Getting started with WildFly

This page has moved to https://github.com/wildfly/quickstart/blob/10.x/guide/GettingStarted.asciidoc

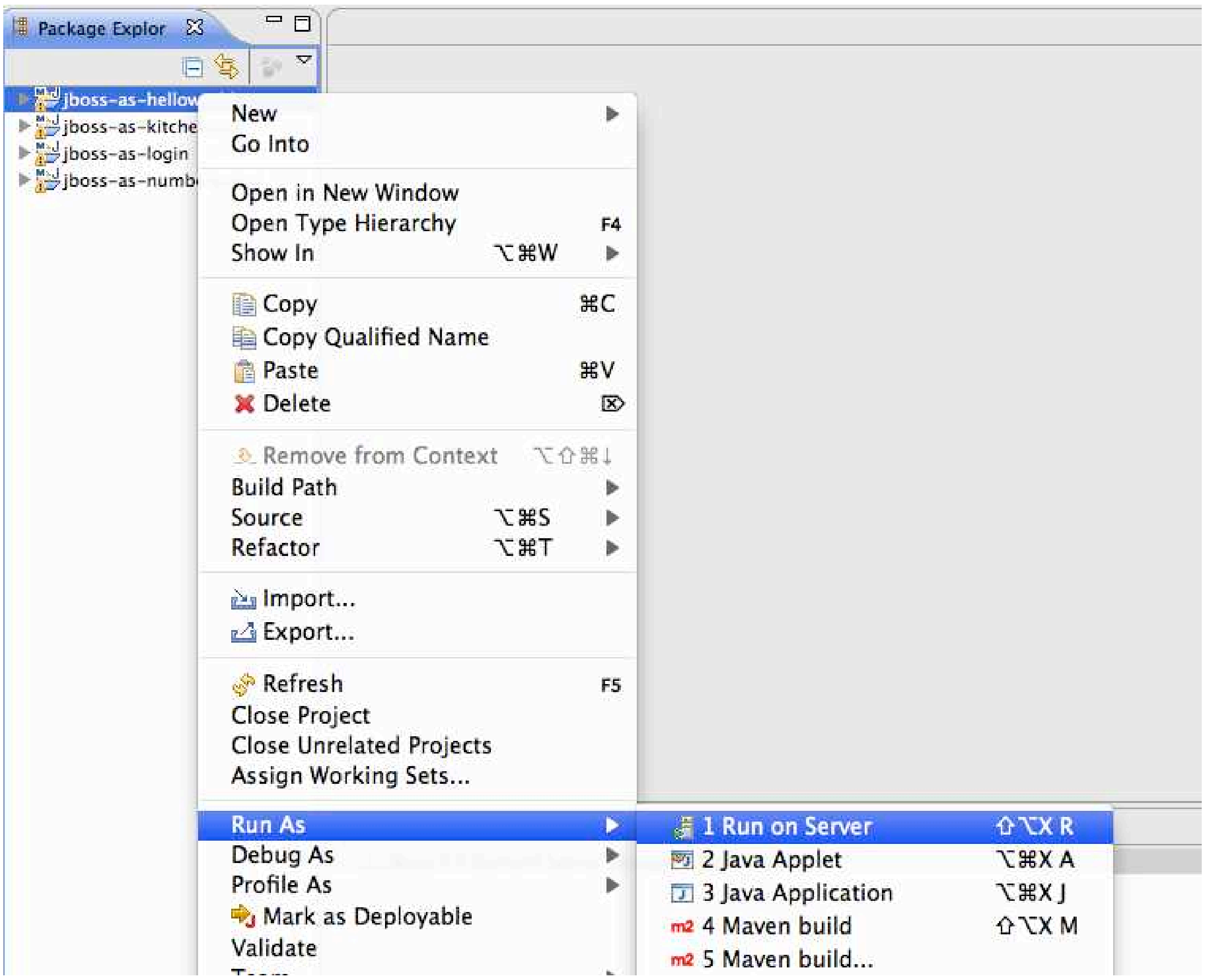
3 Helloworld quickstart

This page has moved to https://github.com/wildfly/quickstart/blob/10.x/guide/HelloworldQuickstart.asciidoc

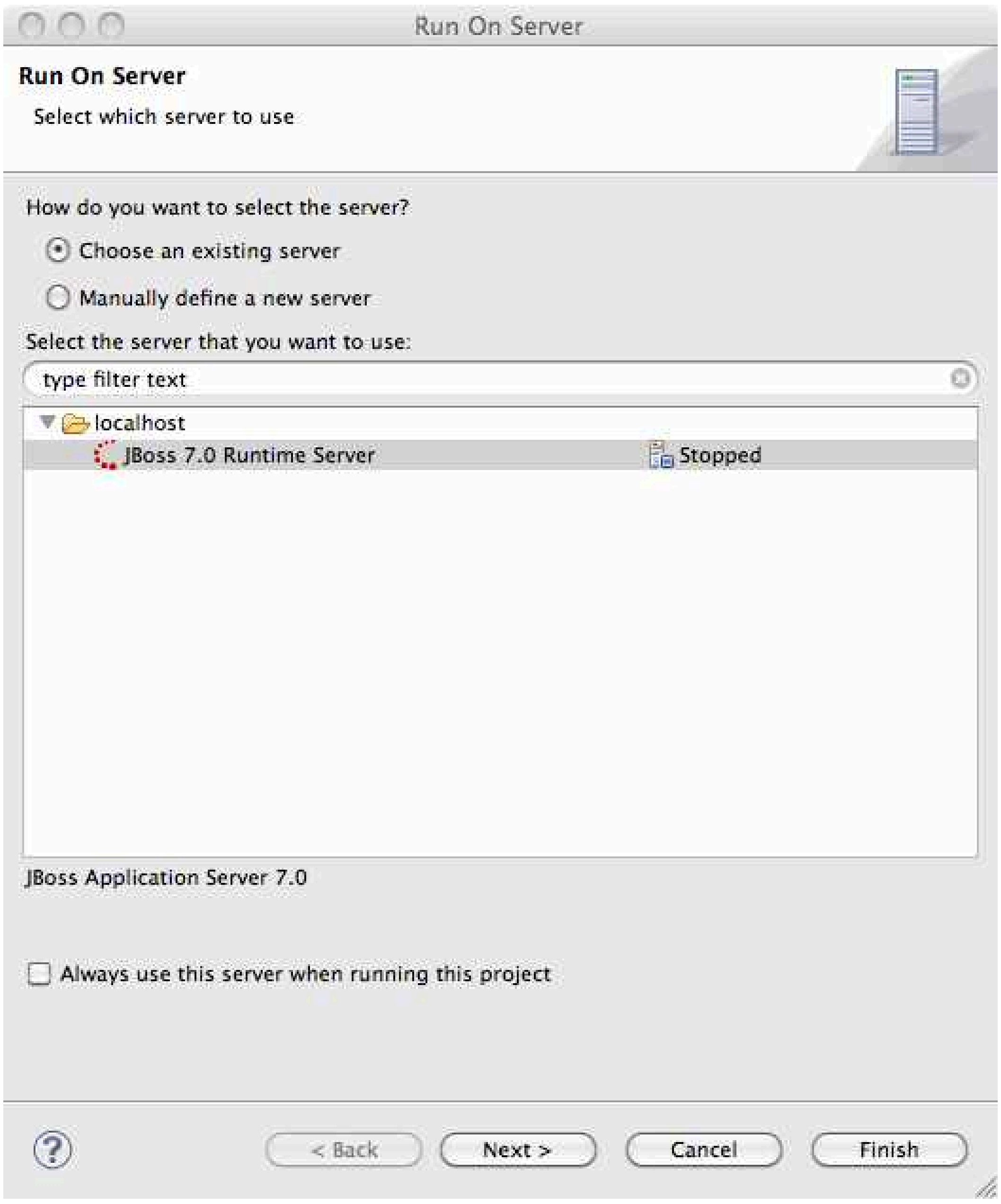
3.1 Deploying the Helloworld example using Eclipse

You may choose to deploy the example using Eclipse. You'll need to have JBoss AS started in Eclipse (as described in [Starting JBoss AS from Eclipse with JBoss Tools]) and to have imported the quickstarts into Eclipse (as described in [Importing the quickstarts into Eclipse]).

With the quickstarts imported, you can deploy the example by right clicking on the `jboss-as-helloworld` project, and choosing `Run As -> Run On Server`:



Make sure the JBoss AS server is selected, and hit `Finish`:



Run On Server

Run On Server

Select which server to use

How do you want to select the server?

[✓ Choose an existing server]

Manually define a new server

Select the server that you want to use:

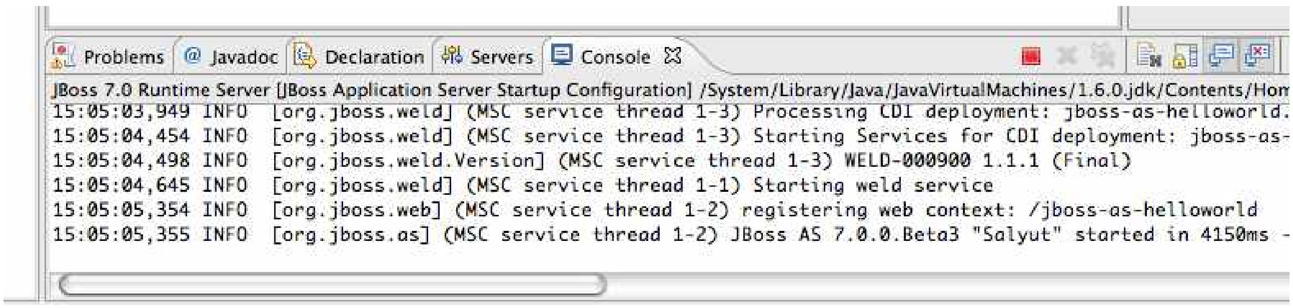
type filter text

localhost	
JBoss 7.0 Runtime Server	Stopped

JBoss Application Server 7.0

☐ Always use this server when running this project

You should see JBoss AS start up (unless you already started it in [Starting JBoss AS from Eclipse with JBoss Tools]) and the application deploy in the Console log:



```
JBoss 7.0 Runtime Server [JBoss Application Server Startup Configuration] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Hom
15:05:03,949 INFO  [org.jboss.weld] (MSC service thread 1-3) Processing CDI deployment: jboss-as-helloworld.
15:05:04,454 INFO  [org.jboss.weld] (MSC service thread 1-3) Starting Services for CDI deployment: jboss-as-
15:05:04,498 INFO  [org.jboss.weld.Version] (MSC service thread 1-3) WELD-000900 1.1.1 (Final)
15:05:04,645 INFO  [org.jboss.weld] (MSC service thread 1-1) Starting weld service
15:05:05,354 INFO  [org.jboss.web] (MSC service thread 1-2) registering web context: /jboss-as-helloworld
15:05:05,355 INFO  [org.jboss.as] (MSC service thread 1-2) JBoss AS 7.0.0.Beta3 "Salyut" started in 4150ms -
```

3.2 The helloworld example in depth

This page has moved to
http://www.jboss.org/jdf/quickstarts/jboss-as-quickstart/guide/HelloworldQuickstart/#_the_helloworld_quicksta

4 Numberguess quickstart

This page has moved to
https://github.com/wildfly/quickstart/blob/10.x/guide/NumberguessQuickstart.asciidoc

4.1 Deploying the Numberguess example using Eclipse

This page has moved to
http://www.jboss.org/jdf/quickstarts/jboss-as-quickstart/guide/NumberguessQuickstart/#_deploying_the_numb

4.2 The numberguess example in depth

This page has moved to
http://www.jboss.org/jdf/quickstarts/jboss-as-quickstart/guide/NumberguessQuickstart/#_the_numberguess_q

5 Greeter quickstart

This page has moved to https://github.com/wildfly/quickstart/blob/10.x/guide/GreeterQuickstart.asciidoc

5.1 Deploying the Login example using Eclipse

This page has moved to http://www.jboss.org/jdf/quickstarts/jboss-as-quickstart/guide/GreeterQuickstart/

5.2 The login example in depth

This page has moved to
http://www.jboss.org/jdf/quickstarts/jboss-as-quickstart/guide/GreeterQuickstart/#greeter_in_depth

6 Kitchensink quickstart

This page has moved to https://github.com/wildfly/quickstart/blob/10.x/guide/KitchensinkQuickstart.asciidoc

6.1 Deploying the Kitchensink example using Eclipse

This page has moved to
http://www.jboss.org/jdf/quickstarts/jboss-as-quickstart/guide/KitchensinkQuickstart/#_deploying_the_kitchens

6.2 The kitchensink example in depth

This page has moved to
http://www.jboss.org/jdf/quickstarts/jboss-as-quickstart/guide/KitchensinkQuickstart/#_the_kitchensink_quicks

7 Creating your own application

This page has moved to https://github.com/wildfly/quickstart/blob/10.x/guide/Archetype.asciidoc

7.1 Creating your own application using Eclipse

This page has moved to http://www.jboss.org/jdf/quickstarts/jboss-as-quickstart/guide/Archetype/

8 More Resources

Getting Started Guide	The Getting Started Guide covers topics such as server layout (what you can configure where), data source definition, and using the web management interface.
Torquebox	Torque Box allows you to use all the familiar services from JBoss AS 7, but with Ruby.
JBoss AS 7 FAQ	Frequently Asked Questions for JBoss AS 7

8.1 Developing JSF Project Using JBoss AS7, Maven and IntelliJ

JBoss AS7 is a very 'modern' application server that has very fast startup speed. So it's an excellent container to test your JSF project. In this article, I'd like to show you how to use AS7, maven and IntelliJ together to develop your JSF project.

In this article I'd like to introduce the following things:

- Create a project using Maven
- Add JSF into project
- Writing Code
- Add JBoss AS 7 deploy plugin into project
- Deploy project to JBoss AS 7
- Import project into IntelliJ
- Add IntelliJ JSF support to project
- Add JBoss AS7 to IntelliJ
- Debugging project with IntelliJ and AS7

I won't explain many basic concepts about AS7, maven and IntelliJ in this article because there are already many good introductions on these topics. So before doing the real work, there some preparations should be done firstly:

Download JBoss AS7

It could be downloaded from here: http://www.jboss.org/jbossas/downloads/

Using the latest release would be fine. When I'm writing this article the latest version is 7.1.1.Final.

Install Maven

Please make sure you have maven installed on your machine. Here is my environment:

```
weli@power:~$ mvn -version
Apache Maven 3.0.3 (r1075438; 2011-03-01 01:31:09+0800)
Maven home: /usr/share/maven
Java version: 1.6.0_33, vendor: Apple Inc.
Java home: /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home
Default locale: en_US, platform encoding: MacRoman
OS name: "mac os x", version: "10.8", arch: "x86_64", family: "mac"
```

Get IntelliJ

In this article I'd like to use IntelliJ Ultimate Edition as the IDE for development, it's a commercial software and can be downloaded from: http://www.jetbrains.com/idea/

The version I'm using is IntelliJ IDEA Ultimate 11.1

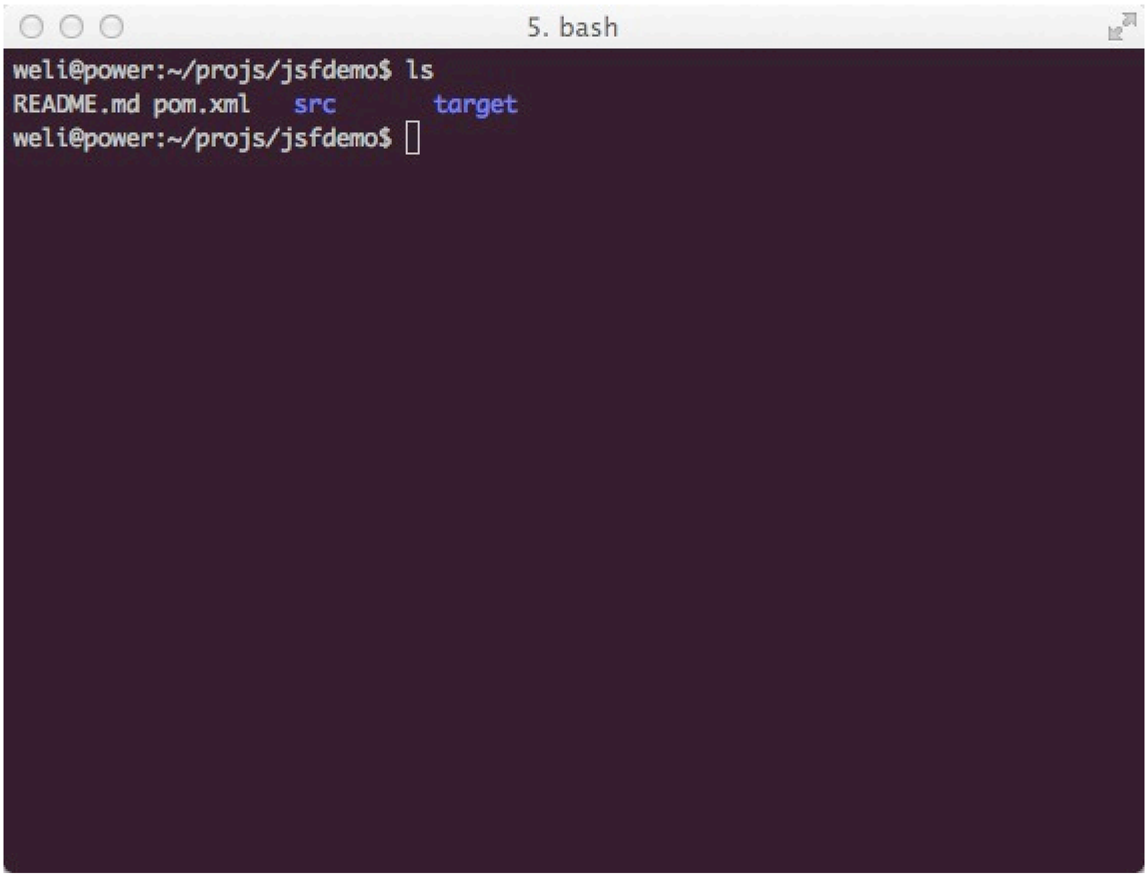
After all of these prepared, we can dive into the real work:

8.1.1 Create a project using Maven

Use the following maven command to create a web project:

```
mvn archetype:create -DarchetypeGroupId=org.apache.maven.archetypes \
-DarchetypeArtifactId=maven-archetype-webapp \
-DarchetypeVersion=1.0 \
-DgroupId=net.bluedash \
-DartifactId=jsfdemo \
-Dversion=1.0-SNAPSHOT
```

If everything goes fine maven will generate the project for us:



```
weli@power:~/projs/jsfdemo$ ls
README.md pom.xml   src       target
weli@power:~/projs/jsfdemo$ 
```

The contents of the project is shown as above.

8.1.2 Add JSF into project

The JSF library is now included in maven repo, so we can let maven to manage the download for us. First is to add repository into our pom.xml:

```
<repository>
  <id>jvnet-nexus-releases</id>
  <name>jvnet-nexus-releases</name>
  <url>https://maven.java.net/content/repositories/releases/</url>
</repository>
```

Then we add JSF dependency into pom.xml:

```
<dependency>
    <groupId>javax.faces</groupId>
    <artifactId>jsf-api</artifactId>
    <version>2.1</version>
    <scope>provided</scope>
</dependency>
```

Please note the 'scope' is 'provided', because we don't want to bundle the jsf.jar into the war produced by our project later, as JBoss AS7 already have jsf bundled in.

Then we run 'mvn install' to update the project, and maven will download jsf-api for us automatically.

8.1.3 Writing Code

Writing JSF code in this article is trivial, so I've put written a project called 'jsfdemo' onto github:

https://github.com/liweinan/jsfdemo

Please clone this project into your local machine, and import it into IntelliJ following the steps described as above.

8.1.4 Add JBoss AS 7 deploy plugin into project

JBoss AS7 has provide a set of convenient maven plugins to perform daily tasks such as deploying project into AS7. In this step let's see how to use it in our project.

We should put AS7's repository into pom.xml:

```
<repository>
    <id>jboss-public-repository-group</id>
    <name>JBoss Public Repository Group</name>
    <url>http://repository.jboss.org/nexus/content/groups/public/</url>
    <layout>default</layout>
    <releases>
        <enabled>true</enabled>
        <updatePolicy>never</updatePolicy>
    </releases>
    <snapshots>
        <enabled>true</enabled>
        <updatePolicy>never</updatePolicy>
    </snapshots>
</repository>
```

And also the plugin repository:

```
<pluginRepository>
    <id>jboss-public-repository-group</id>
    <name>JBoss Public Repository Group</name>
    <url>http://repository.jboss.org/nexus/content/groups/public/</url>
    <releases>
        <enabled>true</enabled>
    </releases>
    <snapshots>
        <enabled>true</enabled>
    </snapshots>
</pluginRepository>
```

And put jboss deploy plugin into 'build' section:

```
<plugin>
    <groupId>org.jboss.as.plugins</groupId>
    <artifactId>jboss-as-maven-plugin</artifactId>
    <executions>
        <execution>
            <phase>package</phase>
            <goals>
                <goal>deploy</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

I've put the final version pom.xml here to check whether your modification is correct:

https://github.com/liweinan/jsfdemo/blob/master/pom.xml

Now we have finished the setup work for maven.

8.1.5 Deploy project to JBoss AS 7

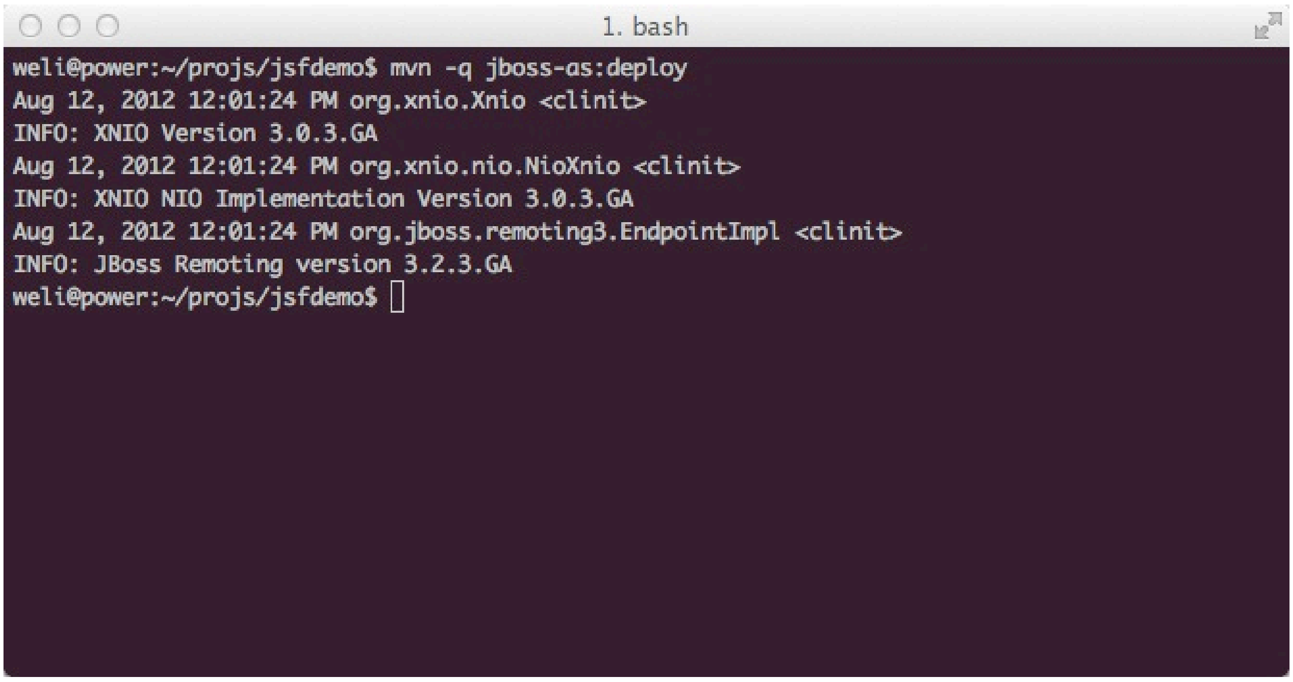
To deploy the project to JBoss AS7, we should start AS7 firstly. In JBoss AS7 directory, run following command:

```
bin/standalone.sh
```

AS7 should start in a short time. Then let's go back to our project directory and run maven command:

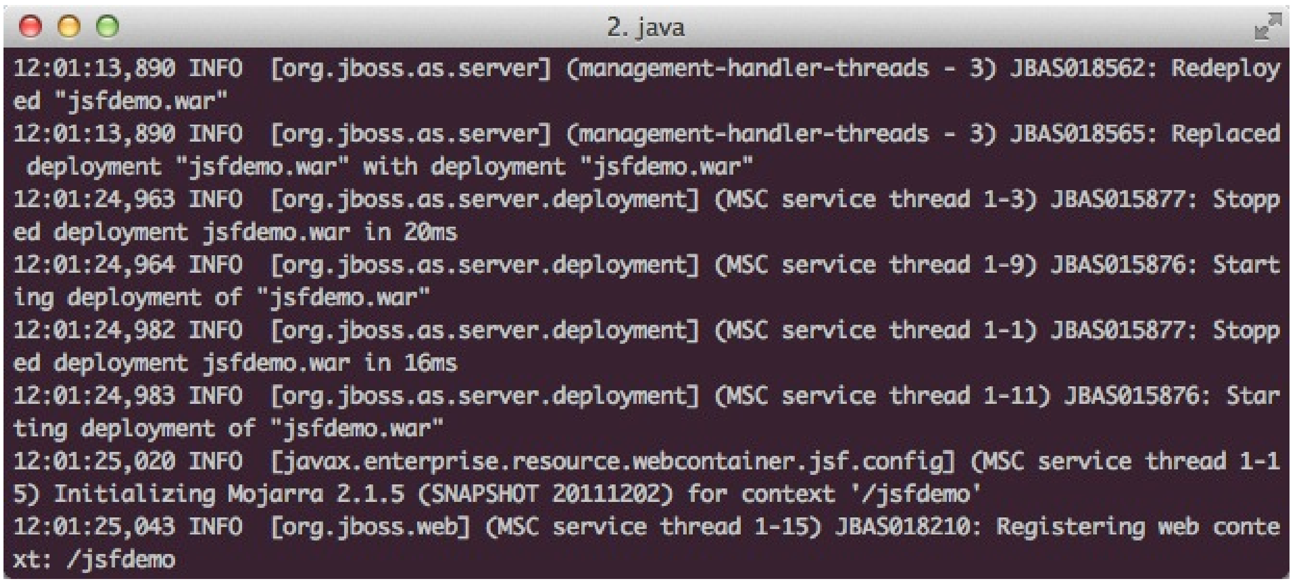
```
mvn -q jboss-as:deploy
```

Maven will use some time to download necessary components for a while, so please wait patiently. After a while, we can see the result:



```
weli@power:~/projs/jsfdemo$ mvn -q jboss-as:deploy
Aug 12, 2012 12:01:24 PM org.xnio.Xnio <clinit>
INFO: XNIO Version 3.0.3.GA
Aug 12, 2012 12:01:24 PM org.xnio.nio.NioXnio <clinit>
INFO: XNIO NIO Implementation Version 3.0.3.GA
Aug 12, 2012 12:01:24 PM org.jboss.remoting3.EndpointImpl <clinit>
INFO: JBoss Remoting version 3.2.3.GA
weli@power:~/projs/jsfdemo$
```

And if you check the console output of AS7, you can see the project is deployed:

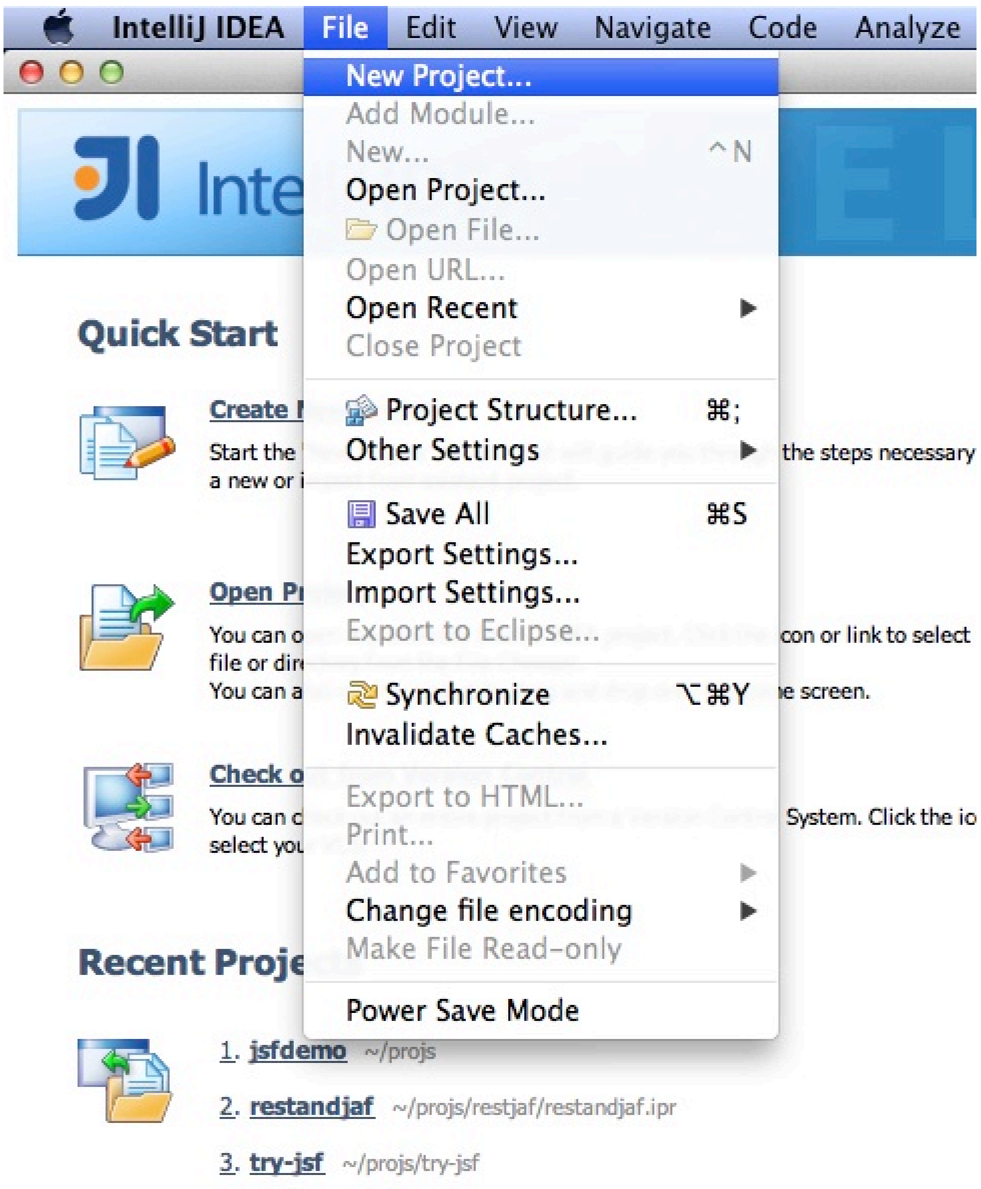


```
12:01:13,890 INFO  [org.jboss.as.server] (management-handler-threads - 3) JBAS018562: Redeploy
ed "jsfdemo.war"
12:01:13,890 INFO  [org.jboss.as.server] (management-handler-threads - 3) JBAS018565: Replaced
 deployment "jsfdemo.war" with deployment "jsfdemo.war"
12:01:24,963 INFO  [org.jboss.as.server.deployment] (MSC service thread 1-3) JBAS015877: Stopp
ed deployment jsfdemo.war in 20ms
12:01:24,964 INFO  [org.jboss.as.server.deployment] (MSC service thread 1-9) JBAS015876: Start
ing deployment of "jsfdemo.war"
12:01:24,982 INFO  [org.jboss.as.server.deployment] (MSC service thread 1-1) JBAS015877: Stopp
ed deployment jsfdemo.war in 16ms
12:01:24,983 INFO  [org.jboss.as.server.deployment] (MSC service thread 1-11) JBAS015876: Star
ting deployment of "jsfdemo.war"
12:01:25,020 INFO  [javax.enterprise.resource.webcontainer.jsf.config] (MSC service thread 1-1
5) Initializing Mojarra 2.1.5 (SNAPSHOT 20111202) for context '/jsfdemo'
12:01:25,043 INFO  [org.jboss.web] (MSC service thread 1-15) JBAS018210: Registering web conte
xt: /jsfdemo
```

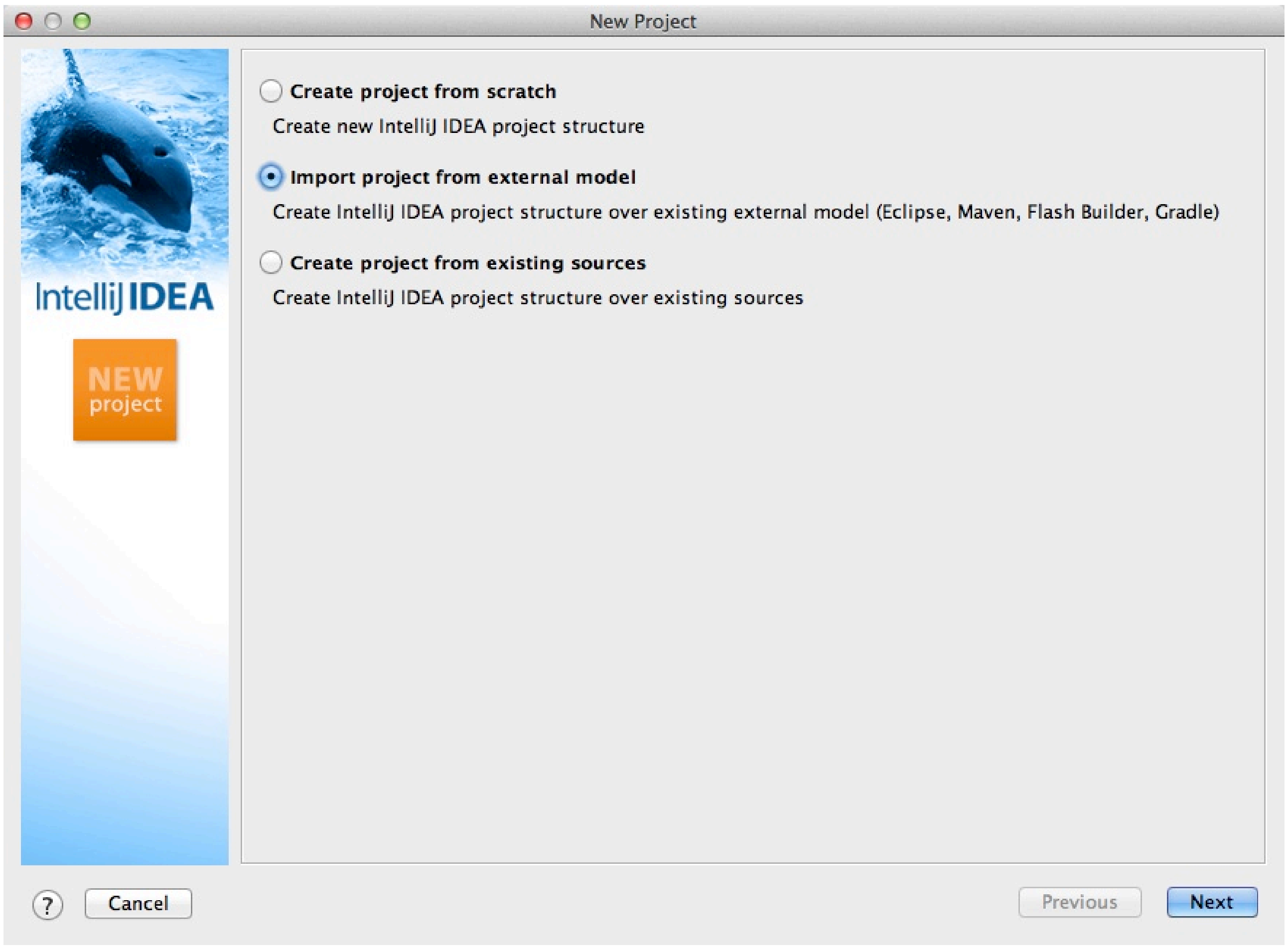
Now we have learnt how to create a JSF project and deploy it to AS7 without any help from graphical tools. Next let's see how to use IntelliJ IDEA to go on developing/debugging our project.

8.1.6 Import project into IntelliJ

Now it's time to import the project into IntelliJ. Now let's open IntelliJ, and choose 'New Project...':



The we choose 'Import project from external model':



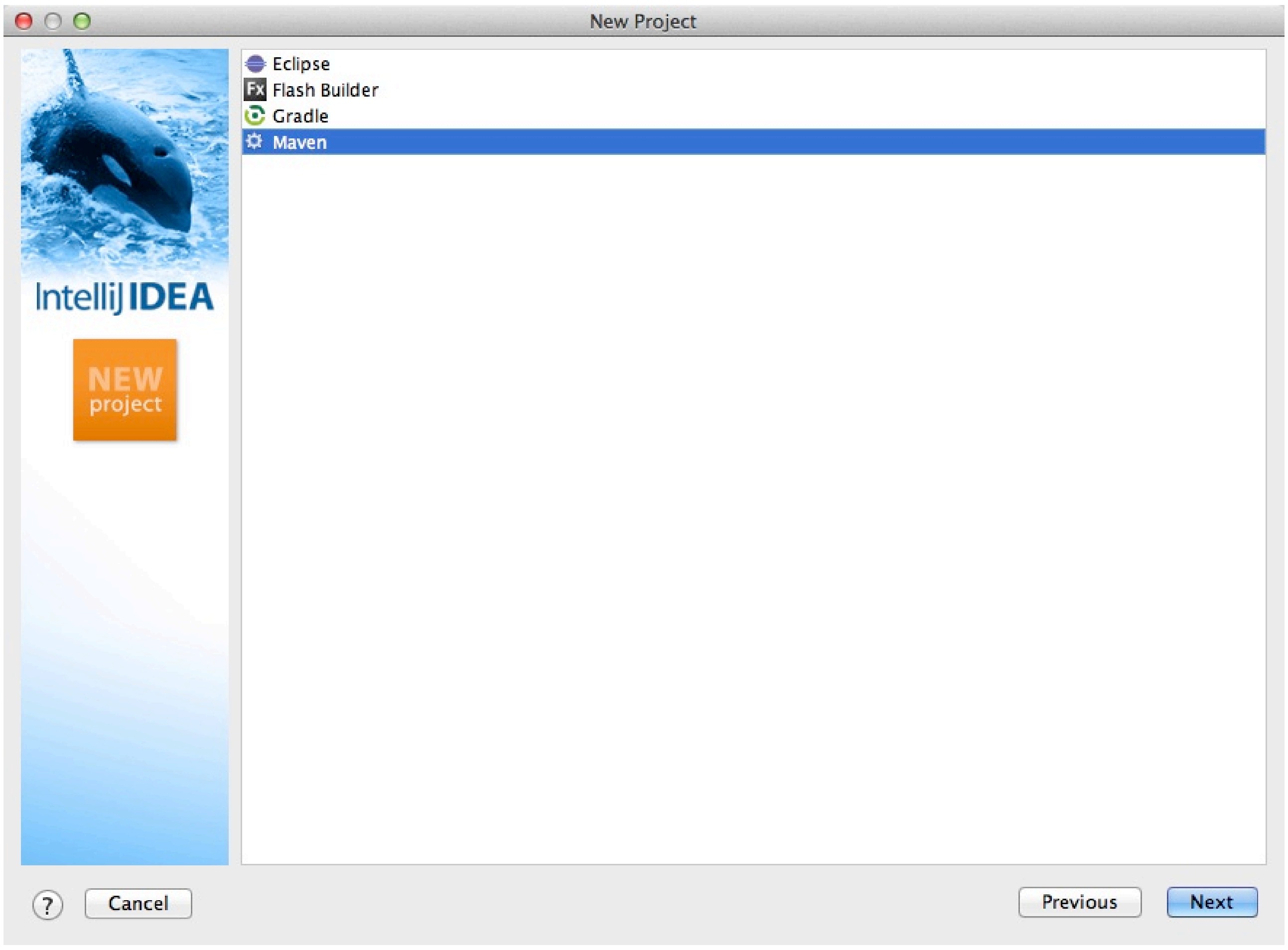
New Project

( ) **Create project from scratch**
Create new IntelliJ IDEA project structure

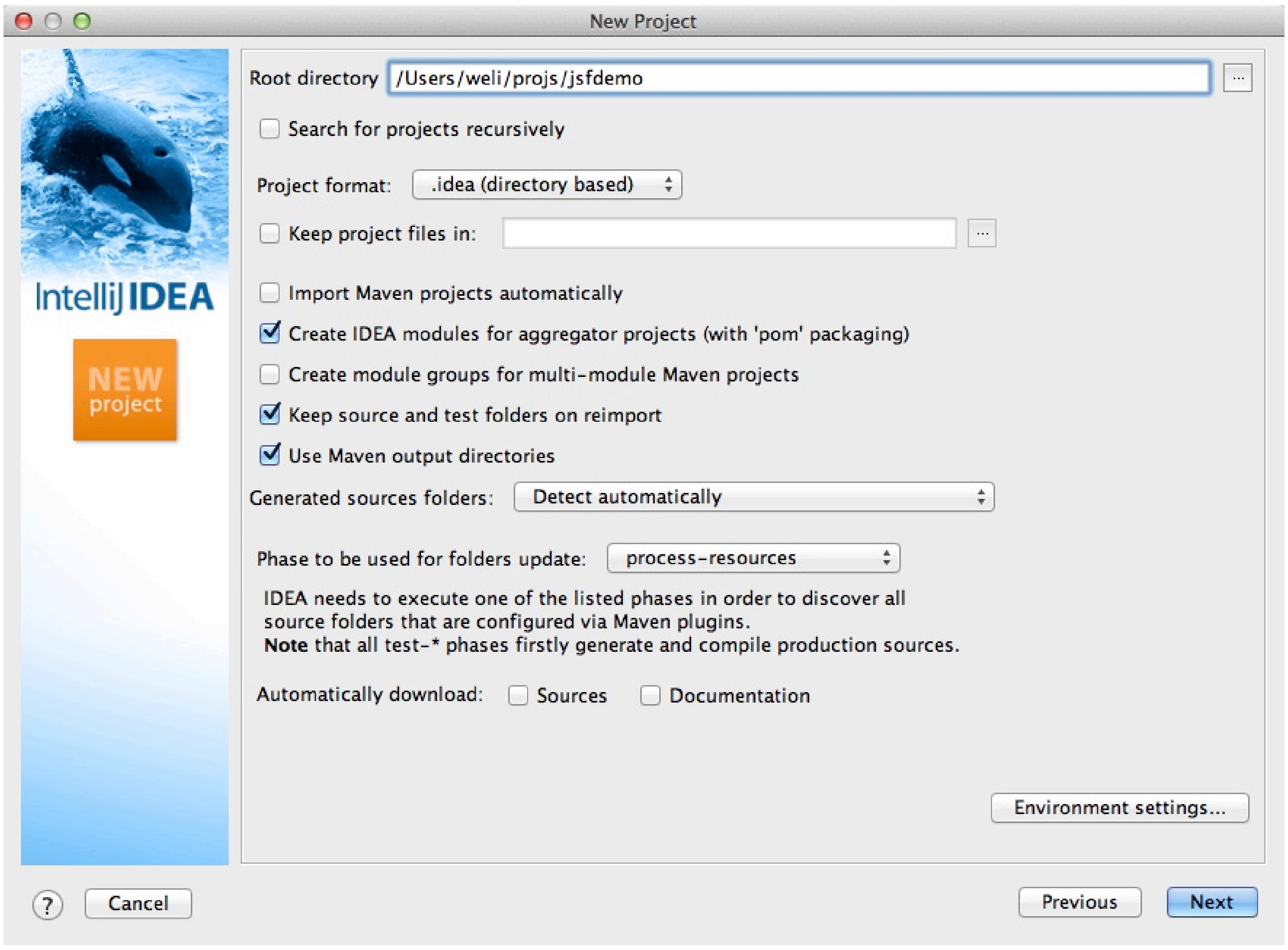
(●) **Import project from external model**
Create IntelliJ IDEA project structure over existing external model (Eclipse, Maven, Flash Builder, Gradle)

( ) **Create project from existing sources**
Create IntelliJ IDEA project structure over existing sources

Next step is choosing 'Maven':



Then IntelliJ will ask you the position of the project you want to import. In 'Root directory' input your project's directory and leave other options as default:


New Project

Root directory /Users/weli/projs/jsfdemo

☐ Search for projects recursively

Project format: .idea (directory based)

☐ Keep project files in:

☐ Import Maven projects automatically

☑ Create IDEA modules for aggregator projects (with 'pom' packaging)

☐ Create module groups for multi-module Maven projects

☑ Keep source and test folders on reimport

☑ Use Maven output directories

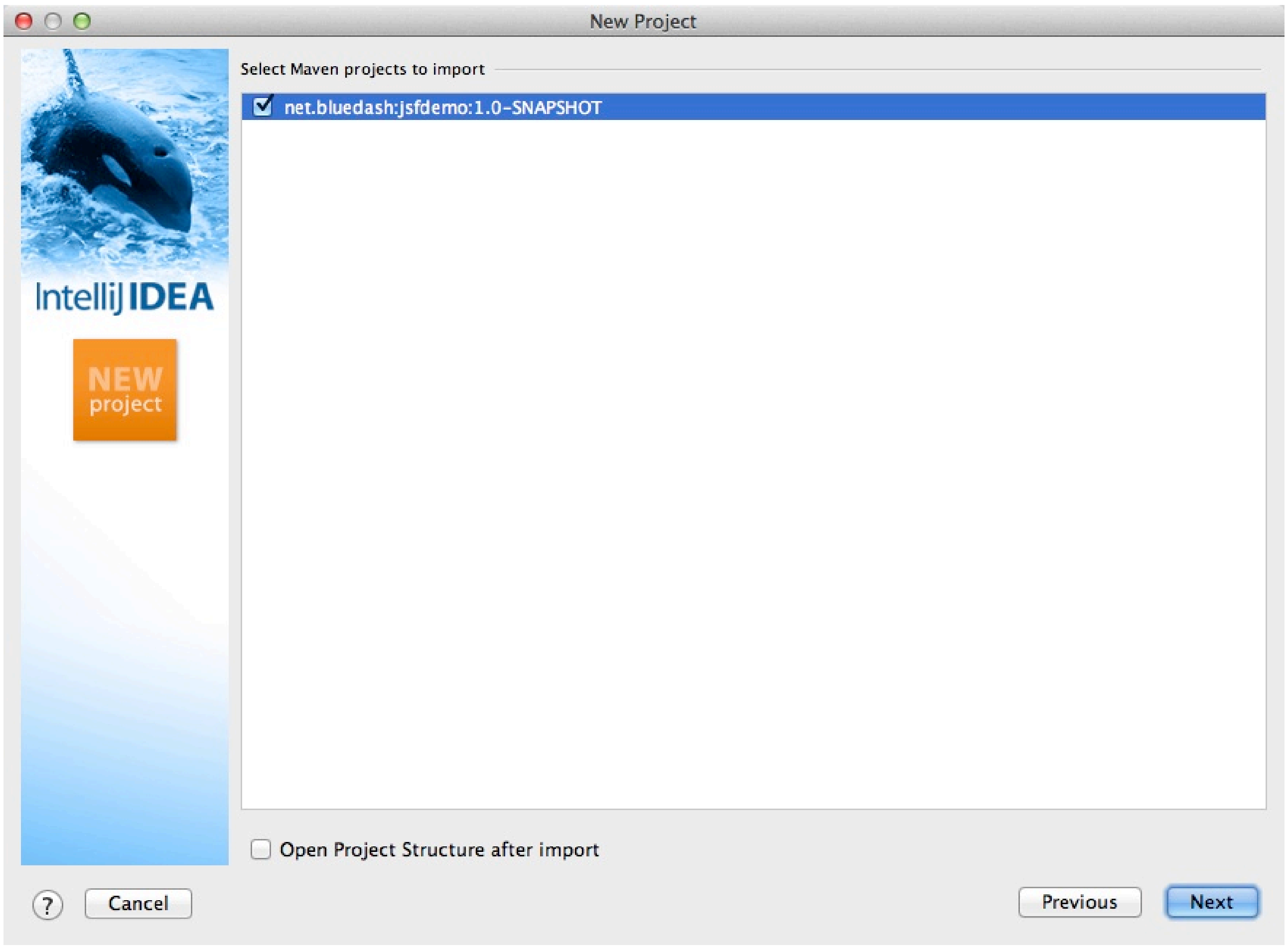
Generated sources folders: Detect automatically

Phase to be used for folders update: process-resources

IDEA needs to execute one of the listed phases in order to discover all source folders that are configured via Maven plugins.
Note that all test-* phases firstly generate and compile production sources.

Automatically download: ☐ Sources ☐ Documentation

For next step, just click 'Next':

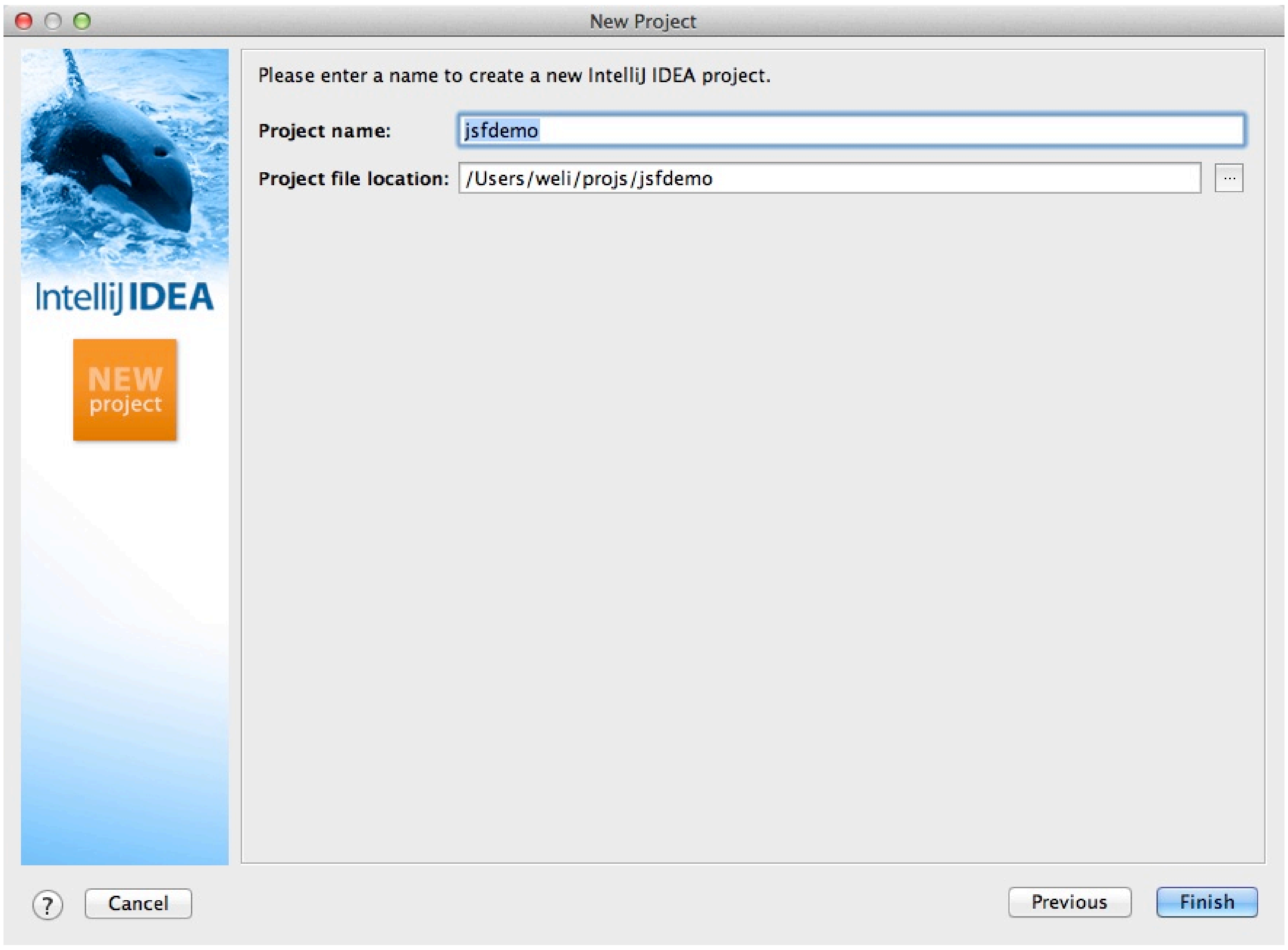


Select Maven projects to import

- [✓] net.bluedash:jsfdemo:1.0-SNAPSHOT

- [ ] Open Project Structure after import

Finally click 'Finish':



Please enter a name to create a new IntelliJ IDEA project.

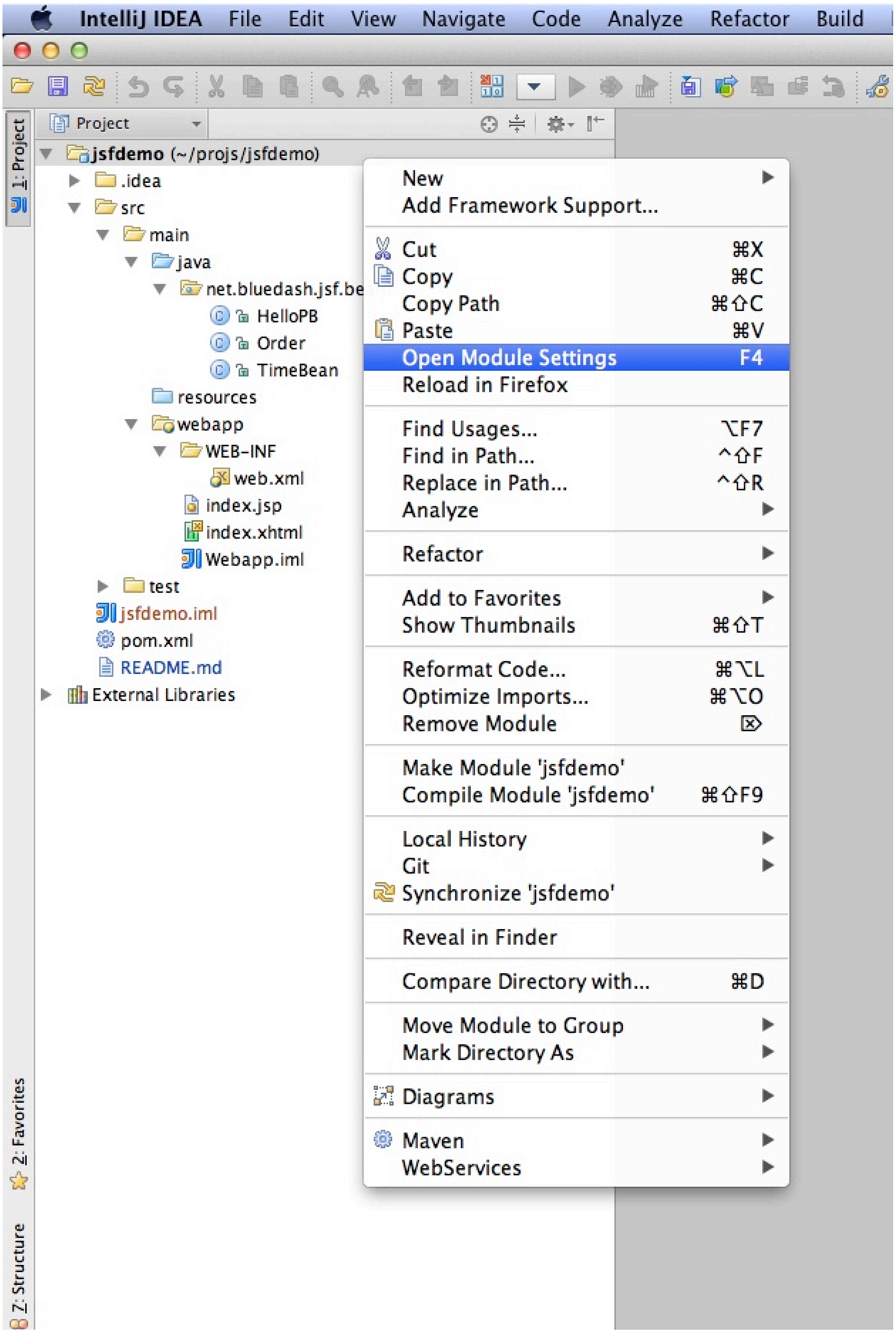
Project name: jsfdemo

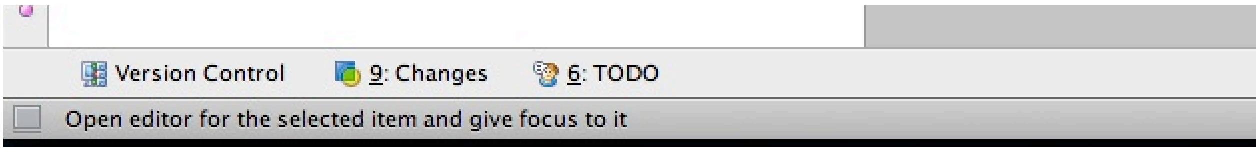
Project file location: /Users/weli/projs/jsfdemo

Hooray! We've imported the project into IntelliJ now 🙂

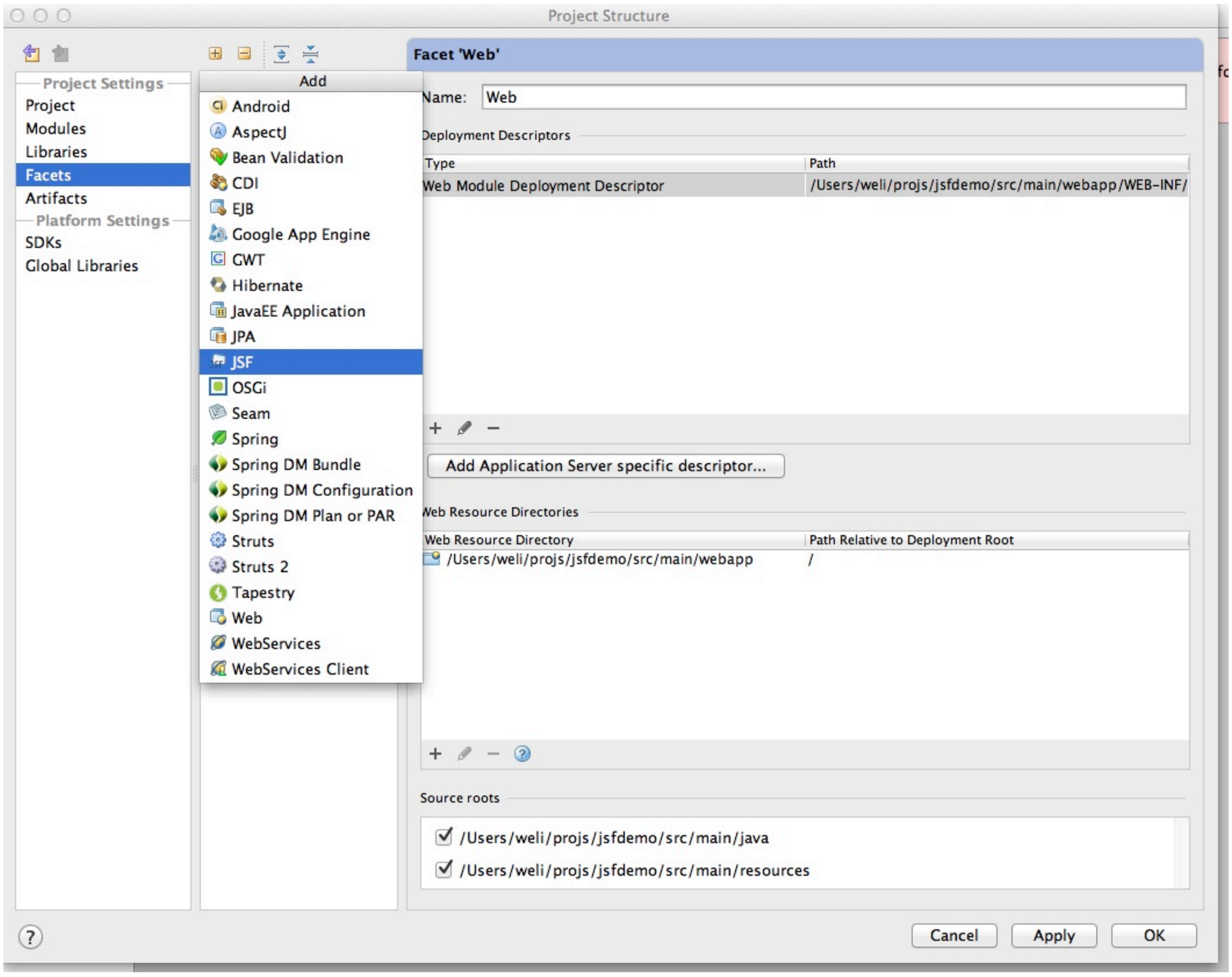
8.1.7 Adding IntelliJ JSF support to project

Let's see how to use IntelliJ and AS7 to debug the project. First we need to add 'JSF' facet into project. Open project setting:


IntelliJ IDEA File Edit View Navigate Code Analyze Refactor Build
Project
jsfdemo (~/projs/jsfdemo)
.idea
src
main
java
net.bluedash.jsf.be
HelloPB
Order
TimeBean
resources
webapp
WEB-INF
web.xml
index.jsp
index.xhtml
Webapp.iml
test
jsfdemo.iml
pom.xml
README.md
External Libraries
New
Add Framework Support...
Cut ⌘X
Copy ⌘C
Copy Path ⌘⇧C
Paste ⌘V
Open Module Settings F4
Reload in Firefox
Find Usages... ⌥F7
Find in Path... ^⇧F
Replace in Path... ^⇧R
Analyze
Refactor
Add to Favorites
Show Thumbnails ⌘⇧T
Reformat Code... ⌘⌥L
Optimize Imports... ⌘⌥O
Remove Module ⌦
Make Module 'jsfdemo'
Compile Module 'jsfdemo' ⌘⇧F9
Local History
Git
Synchronize 'jsfdemo'
Reveal in Finder
Compare Directory with... ⌘D
Move Module to Group
Mark Directory As
Diagrams
Maven
WebServices
1: Project
2: Favorites
7: Structure



Click on 'Facets' section on left; Select 'Web' facet that we already have, and click the '+' on top, choose 'JSF':



Name: Web

Deployment Descriptors

Type	Path
Web Module Deployment Descriptor	/Users/weli/projs/jsfdemo/src/main/webapp/WEB-INF/

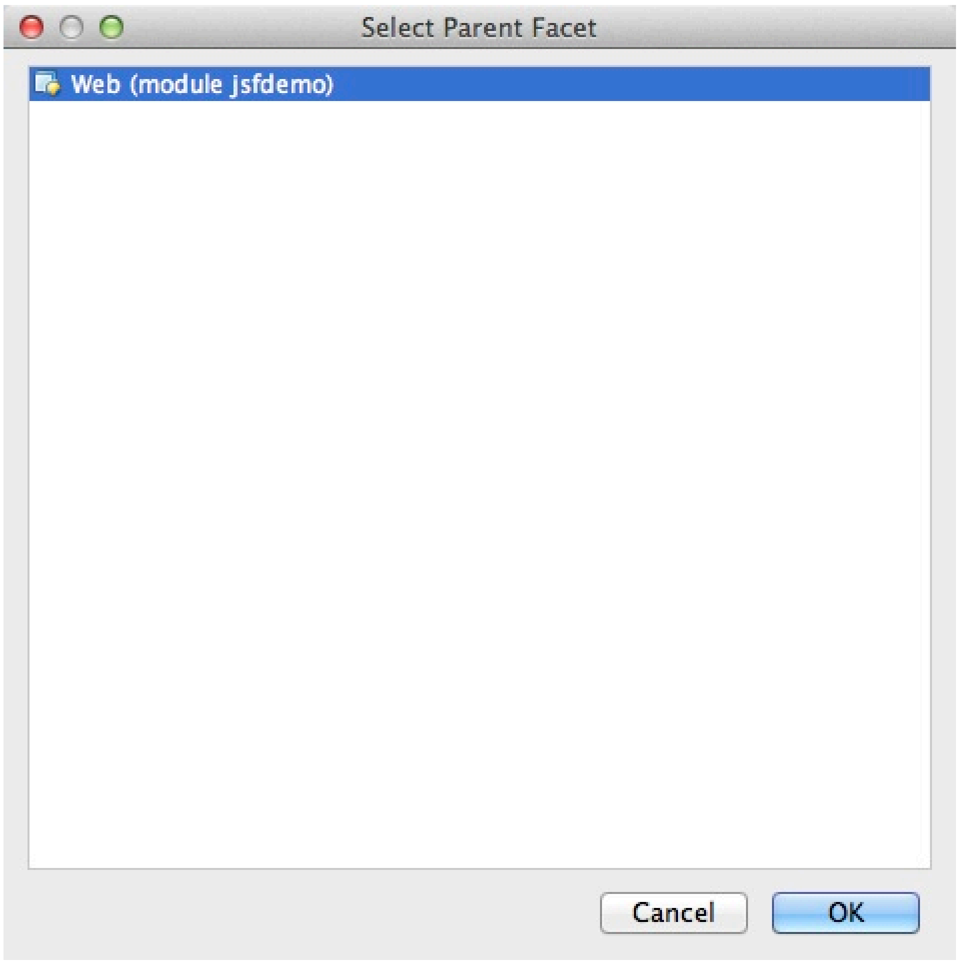
Web Resource Directories

Web Resource Directory	Path Relative to Deployment Root
/Users/weli/projs/jsfdemo/src/main/webapp	/

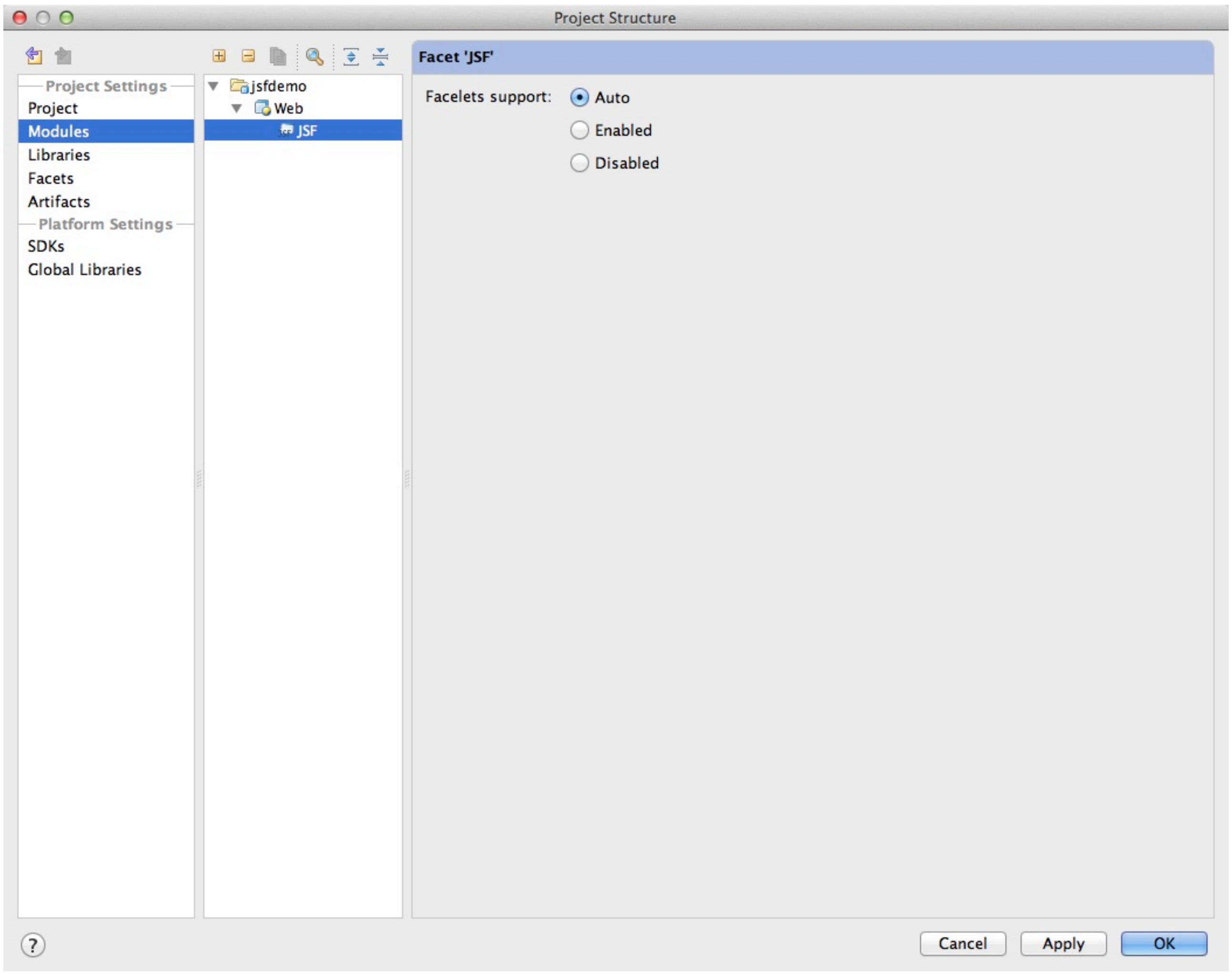
Source roots

- ✓ /Users/weli/projs/jsfdemo/src/main/java
- ✓ /Users/weli/projs/jsfdemo/src/main/resources

Select 'Web' as parent facet:



Click 'Ok':



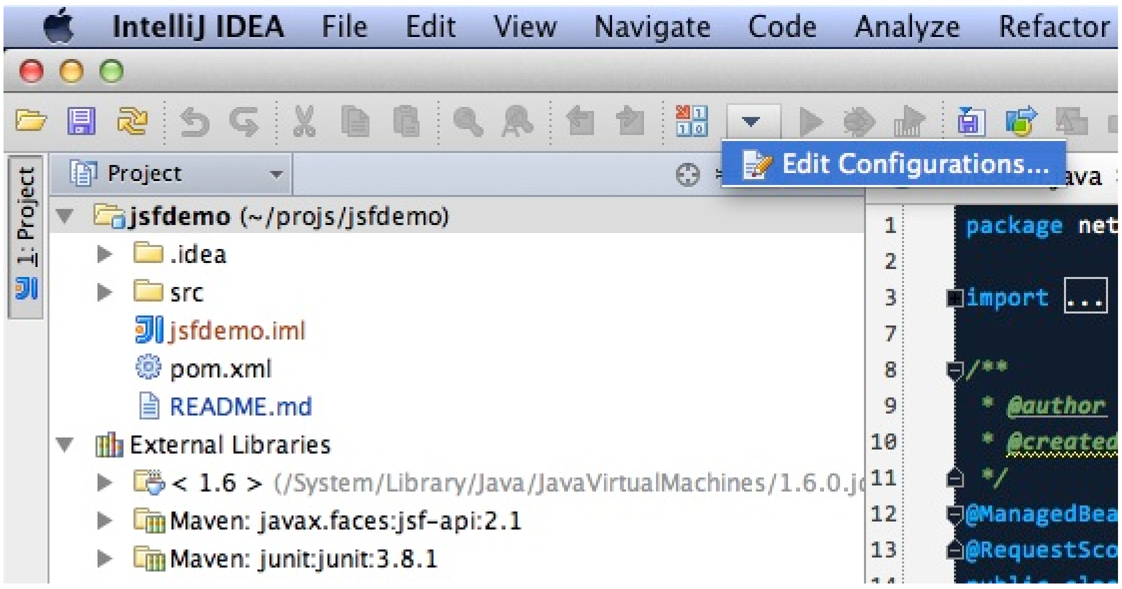
Facet 'JSF'

Facelets support: [✓ Auto] Enabled Disabled

Now we have enabled IntelliJ's JSF support for project.

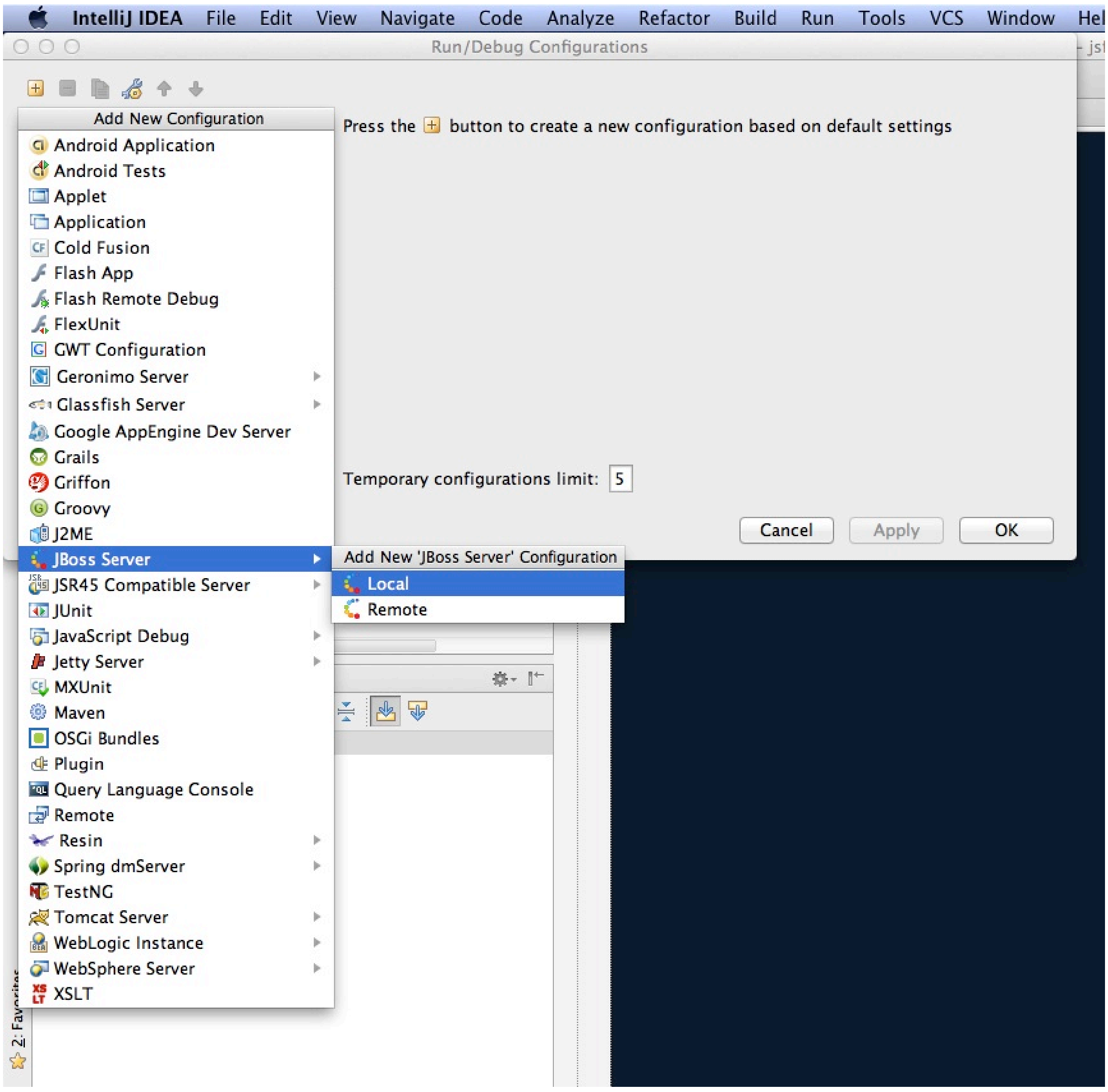
8.1.8 Add JBoss AS7 to IntelliJ

Let's add JBoss AS7 into IntelliJ and use it to debug our project. First please choose 'Edit Configuration' in menu tab:



```
package net
import ...
/**
 * @author
 * @created
 */
@ManagedBea
@RequestSco
```

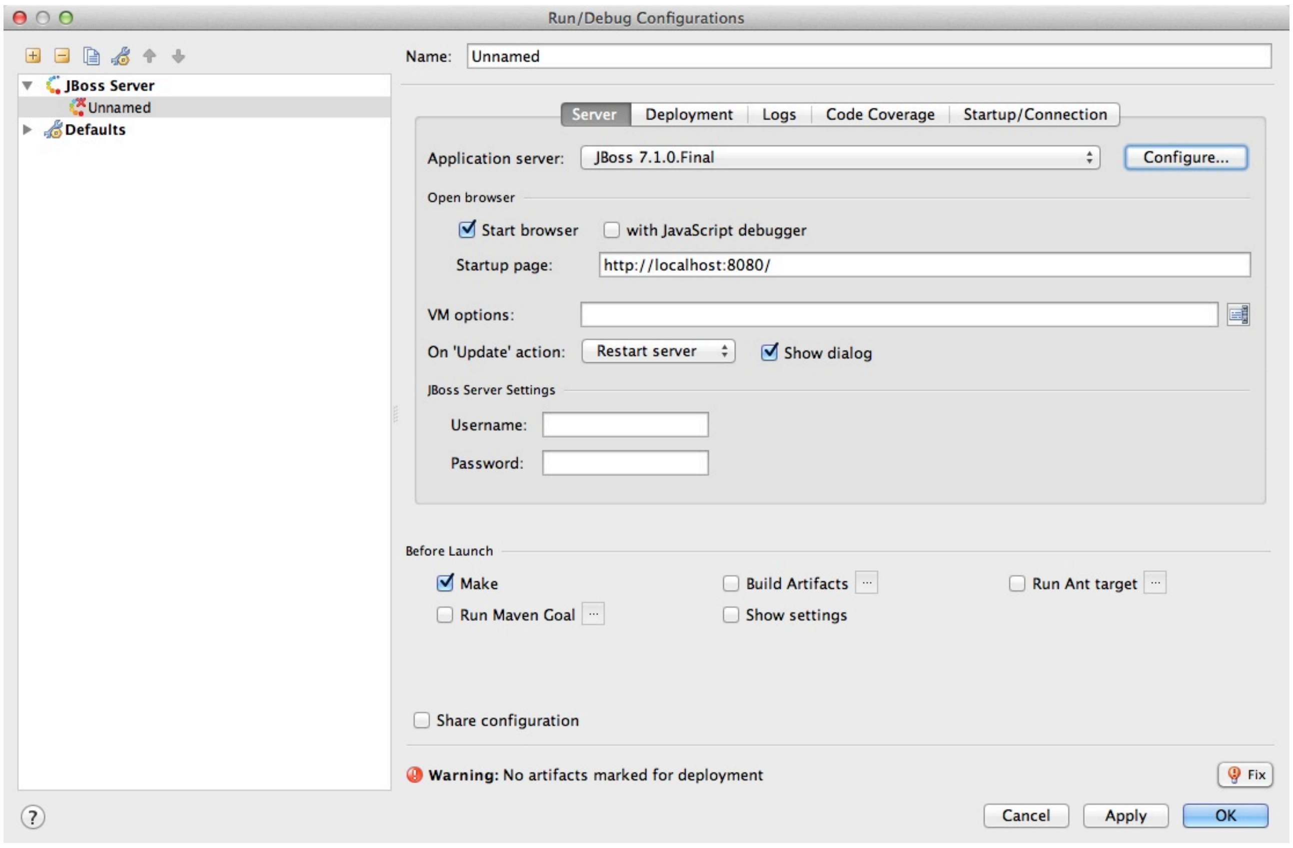
Click '+' and choose 'JBoss Server' -> 'Local':



Press the button to create a new configuration based on default settings

Temporary configurations limit: 5

Click 'configure':


Run/Debug Configurations

JBoss Server
Unnamed
Defaults

Name: Unnamed

Server | Deployment | Logs | Code Coverage | Startup/Connection

Application server: JBoss 7.1.0.Final [Configure...]

Open browser

- [✓] Start browser
- [ ] with JavaScript debugger

Startup page: http://localhost:8080/

VM options:

On 'Update' action: Restart server [✓] Show dialog

JBoss Server Settings

Username:

Password:

Before Launch

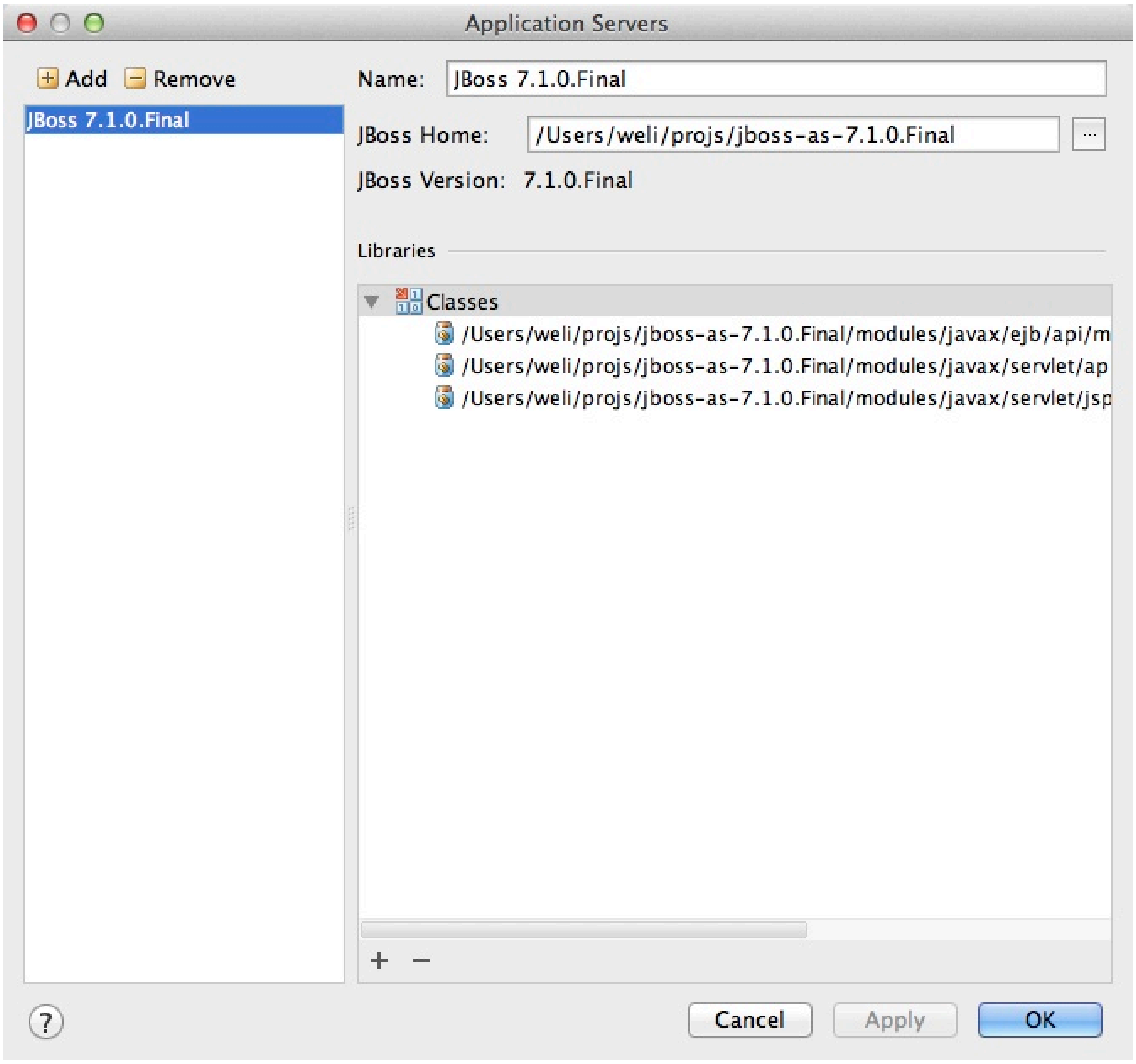
- [✓] Make
- [ ] Build Artifacts
- [ ] Run Ant target
- [ ] Run Maven Goal
- [ ] Show settings

- [ ] Share configuration

Warning: No artifacts marked for deployment [Fix]

[Cancel] [Apply] [OK]

and choose your JBoss AS7:



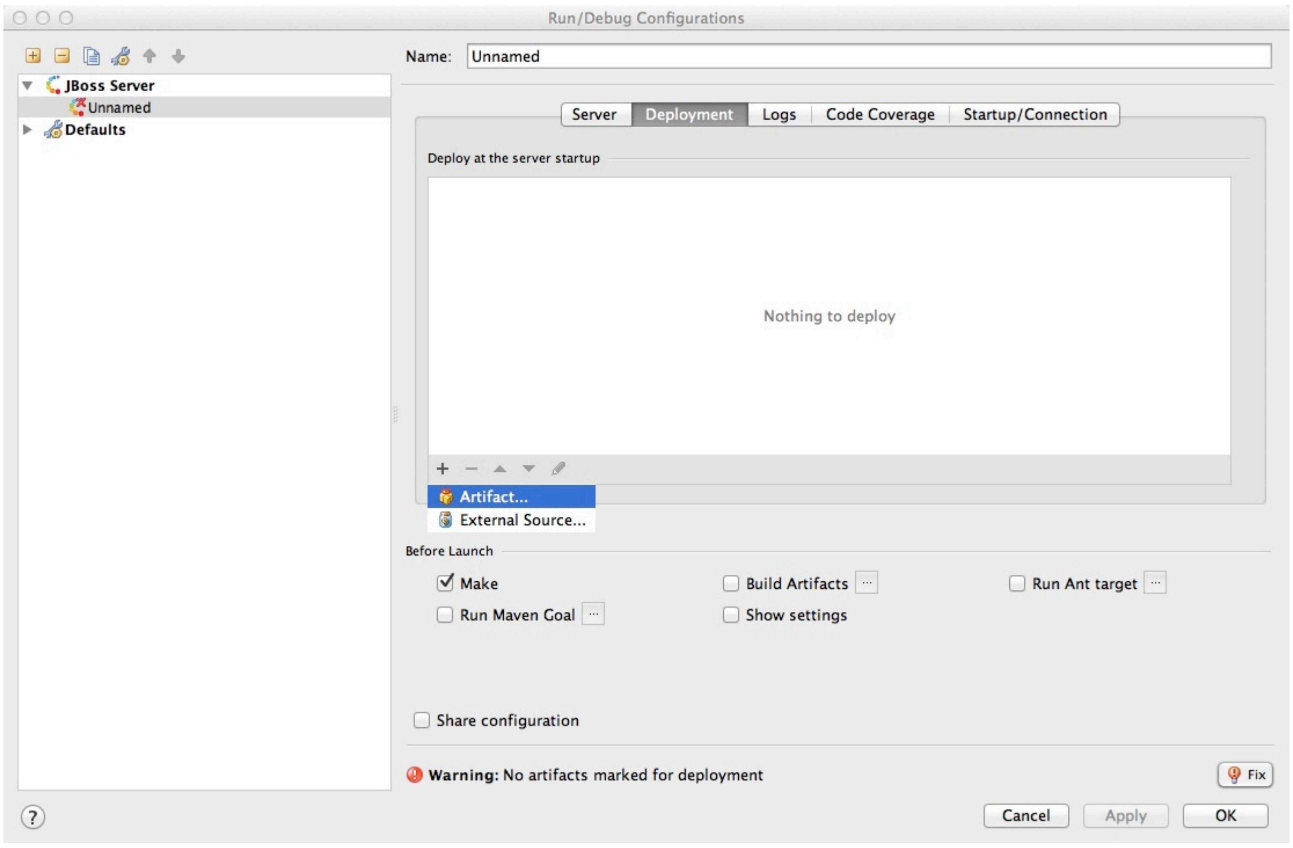
Name: JBoss 7.1.0.Final

JBoss Home: /Users/weli/projs/jboss-as-7.1.0.Final

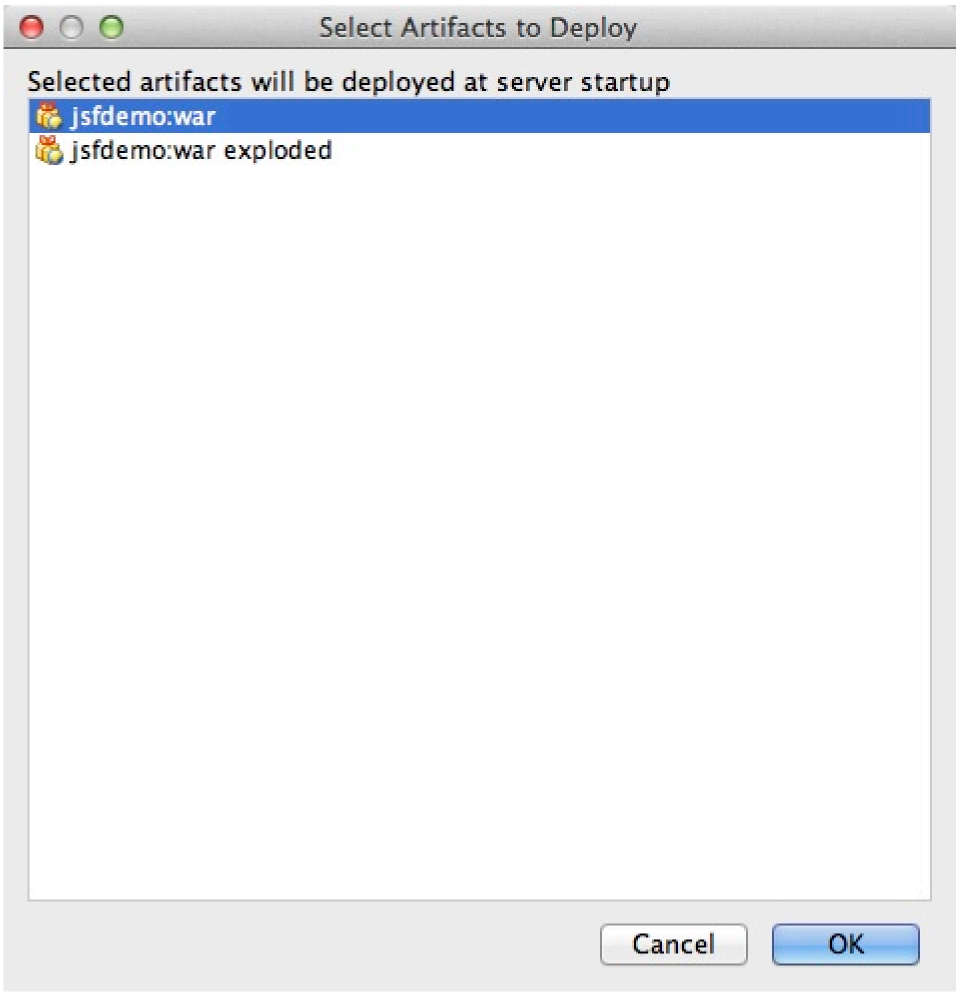
JBoss Version: 7.1.0.Final

Classes
/Users/weli/projs/jboss-as-7.1.0.Final/modules/javax/ejb/api/m
/Users/weli/projs/jboss-as-7.1.0.Final/modules/javax/servlet/ap
/Users/weli/projs/jboss-as-7.1.0.Final/modules/javax/servlet/jsp

Now we need to add our project into deployment. Click the 'Deployment' tab:



Choose 'Artifact', and add our project:



Selected artifacts will be deployed at server startup

jsfdemo:war

jsfdemo:war exploded

Leave everything as default and click 'Ok', now we've added JBoss AS7 into IntelliJ

8.1.9 Debugging project with IntelliJ and AS7

Now comes the fun part. To debug our project, we cannot directly use the 'debug' feature provided by IntelliJ right now(maybe in the future version this problem could be fixed). So now we should use the debugging config provided by AS7 itself to enable JPDA feature, and then use the remote debug function provided by IntelliJ to get things done. Let's dive into the details now:

First we need to enable JPDA config inside AS7, open 'bin/standalone.conf' and find following lines:

```
# Sample JPDA settings for remote socket debugging
#JAVA_OPTS="$JAVA_OPTS -Xrunjdwp:transport=dt_socket,address=8787,server=y,suspend=n"
```

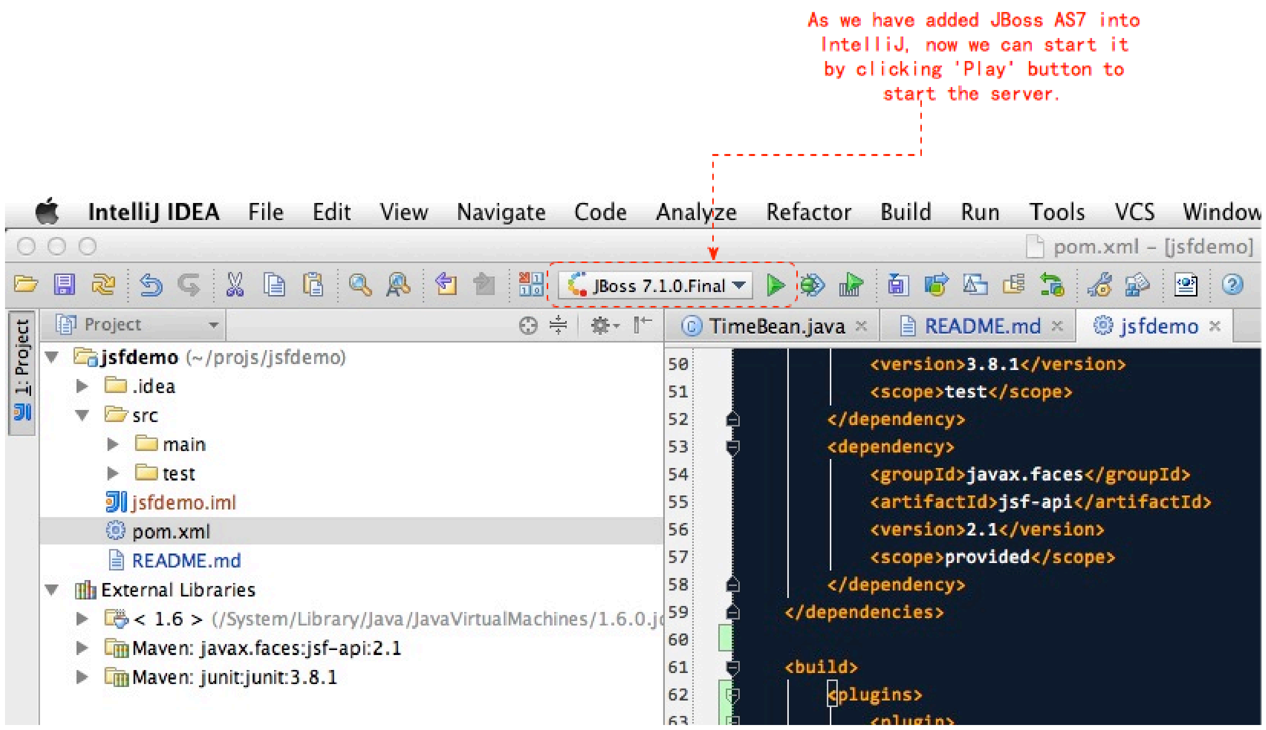
Enable the above config by removing the leading hash sign:

```
# Sample JPDA settings for remote socket debugging
JAVA_OPTS="$JAVA_OPTS -Xrunjdwp:transport=dt_socket,address=8787,server=y,suspend=n"
```

With WildFly you can directly start the server in debug mode:

```
bin/standalone.sh --debug --server-config=standalone.xml
```

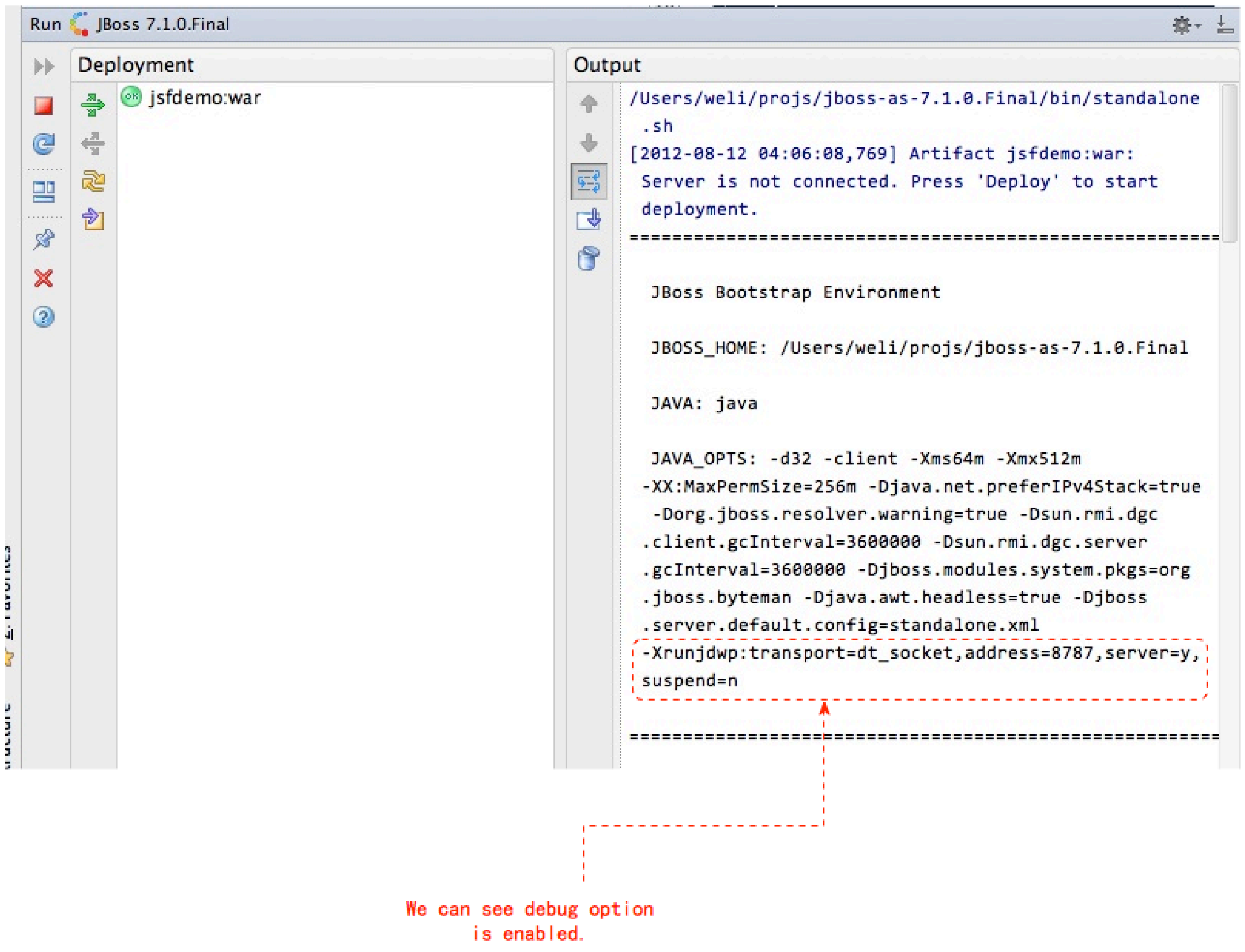
Now we start AS7 in IntelliJ:


As we have added JBoss AS7 into IntelliJ, now we can start it by clicking 'Play' button to start the server.
IntelliJ IDEA File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window
pom.xml - [jsfdemo]
JBoss 7.1.0.Final
Project
TimeBean.java
README.md
jsfdemo
jsfdemo (~/projs/jsfdemo)
.idea
src
main
test
jsfdemo.iml
pom.xml
README.md
External Libraries
< 1.6 > (/System/Library/Java/JavaVirtualMachines/1.6.0.j
Maven: javax.faces:jsf-api:2.1
Maven: junit:junit:3.8.1

```
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>javax.faces</groupId>
            <artifactId>jsf-api</artifactId>
            <version>2.1</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
```

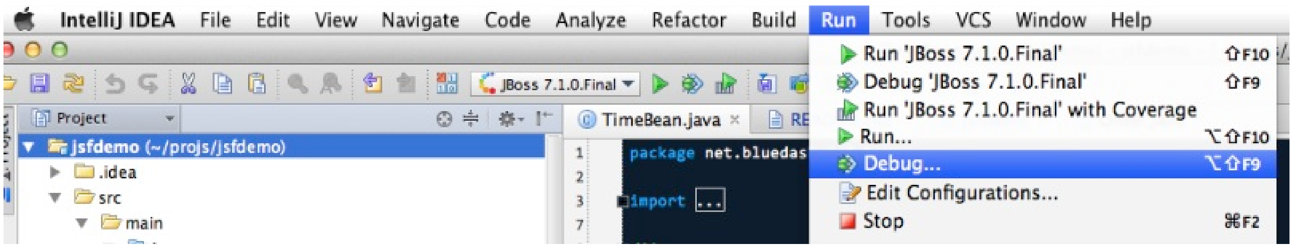
Please note we should undeploy the existing 'jsfdemo' project in AS7 as we've added by maven jboss deploy plugin before. Or AS7 will tell us there is already existing project with same name so IntelliJ could not deploy the project anymore.

If the project start correctly we can see from the IntelliJ console window, and please check the debug option is enabled:

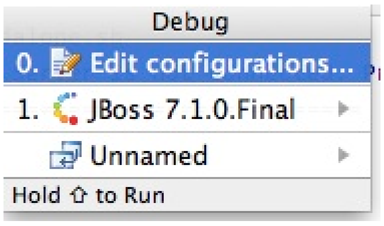


```
/Users/weli/projs/jboss-as-7.1.0.Final/bin/standalone
 .sh
[2012-08-12 04:06:08,769] Artifact jsfdemo:war:
 Server is not connected. Press 'Deploy' to start
 deployment.
=======================================================

  JBoss Bootstrap Environment

  JBOSS_HOME: /Users/weli/projs/jboss-as-7.1.0.Final

  JAVA: java

  JAVA_OPTS: -d32 -client -Xms64m -Xmx512m
 -XX:MaxPermSize=256m -Djava.net.preferIPv4Stack=true
  -Dorg.jboss.resolver.warning=true -Dsun.rmi.dgc
 .client.gcInterval=3600000 -Dsun.rmi.dgc.server
 .gcInterval=3600000 -Djboss.modules.system.pkgs=org
 .jboss.byteman -Djava.awt.headless=true -Djboss
 .server.default.config=standalone.xml
 -Xrunjdwp:transport=dt_socket,address=8787,server=y,
 suspend=n

=======================================================
```

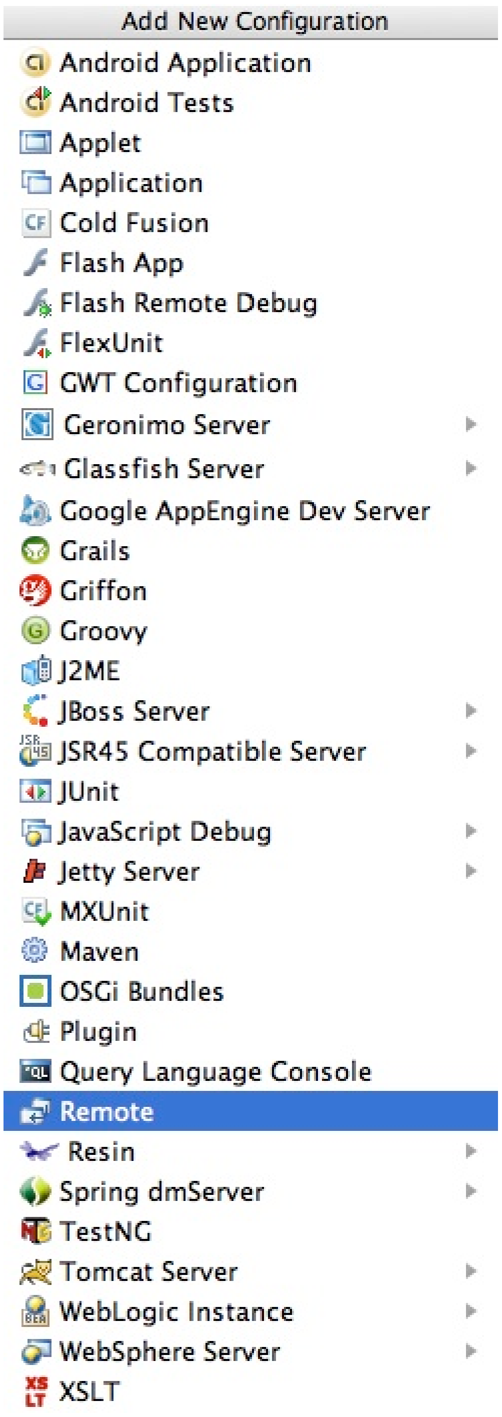
Now we will setup the debug configuration, click 'debug' option on menu:


IntelliJ IDEA File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Run 'JBoss 7.1.0.Final' ⇧F10
Debug 'JBoss 7.1.0.Final' ⇧F9
Run 'JBoss 7.1.0.Final' with Coverage
Run... ⌥⇧F10
Debug... ⌥⇧F9
Edit Configurations...
Stop ⌘F2
Project
jsfdemo (~/projs/jsfdemo)
.idea
src
main
TimeBean.java
package net.bluedas
import ...

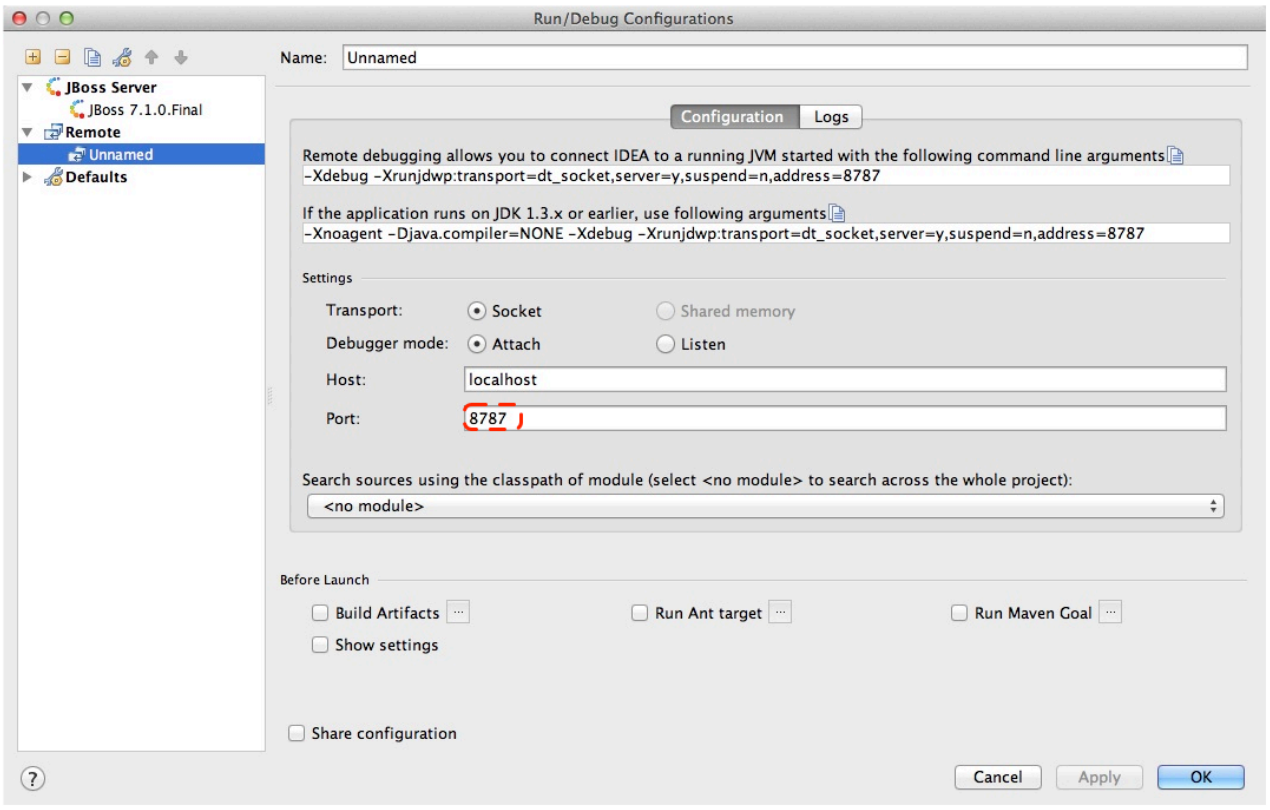
Choose 'Edit Configurations':



Then we click 'Add' and choose Remote:



Set the 'port' to the one you used in AS7 config file 'standalone.conf':



Name: Unnamed

Remote debugging allows you to connect IDEA to a running JVM started with the following command line arguments

```
-Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=8787
```

If the application runs on JDK 1.3.x or earlier, use following arguments

```
-Xnoagent -Djava.compiler=NONE -Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=8787
```

Settings

Transport: [✓ Socket] Shared memory

Debugger mode: [✓ Attach] Listen

Host: localhost

Port: 8787

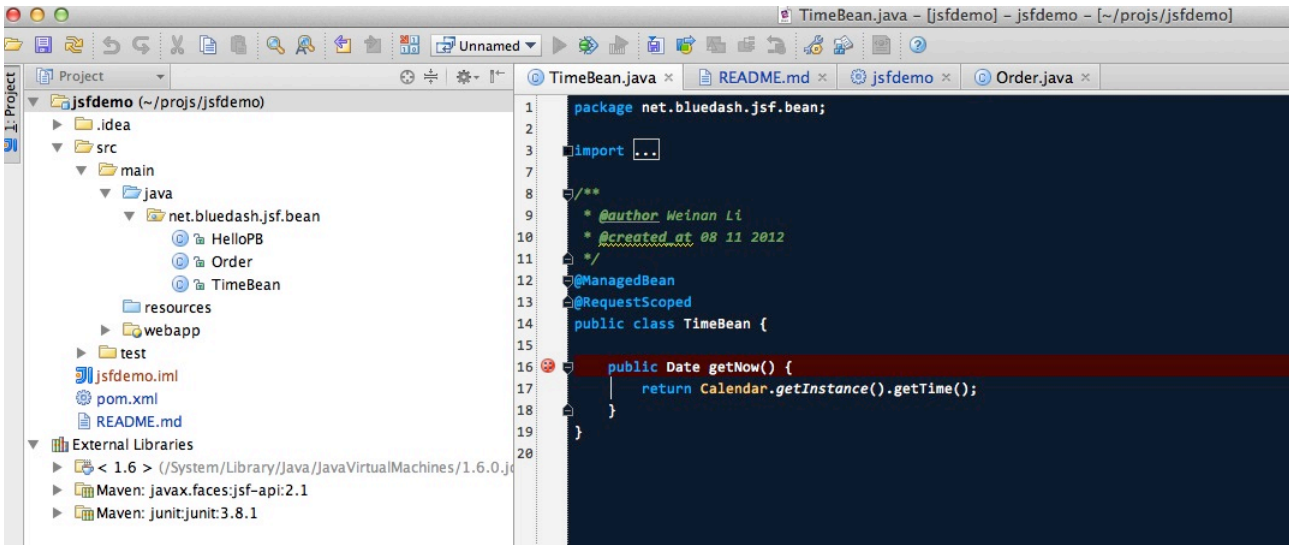
Search sources using the classpath of module (select <no module> to search across the whole project): <no module>

Before Launch

Build Artifacts · Run Ant target · Run Maven Goal · Show settings

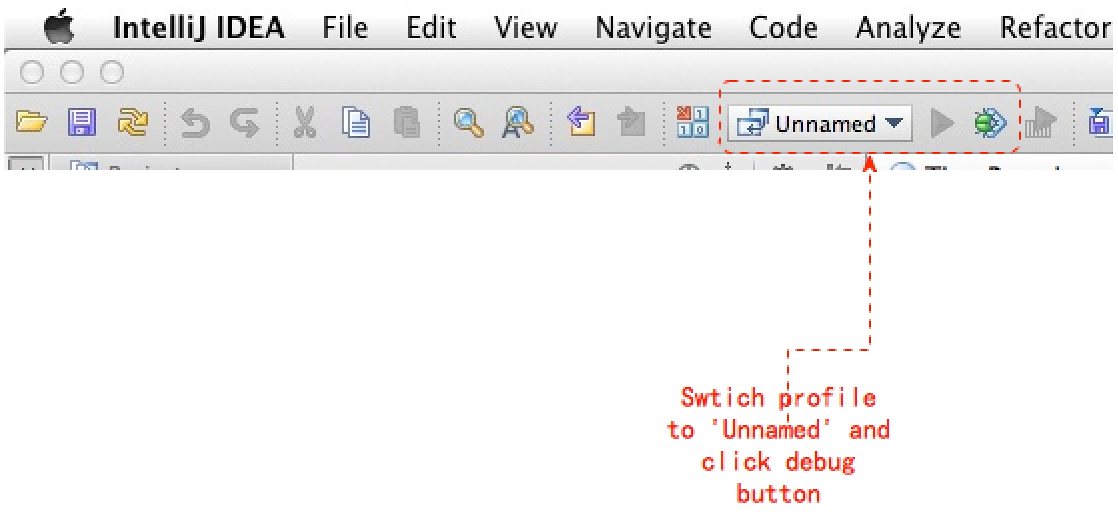
Share configuration

Leave other configurations as default and click 'Ok'. Now we need to set breakpoints in project, let's choose TimeBean.java and set a breakpoint on 'getNow()' method by clicking the left side of that line of code:

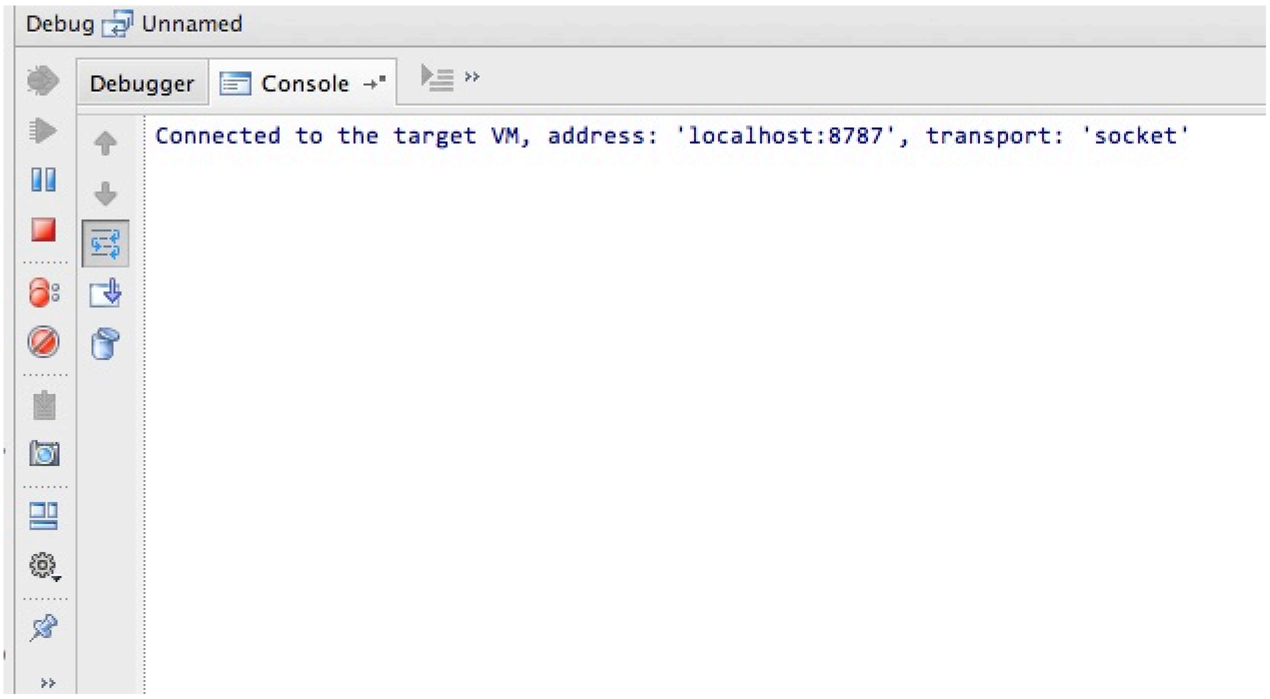


```java
package net.bluedash.jsf.bean;

import ...

/**
 * @author Weinan Li
 * @created_at 08 11 2012
 */
@ManagedBean
@RequestScoped
public class TimeBean {

    public Date getNow() {
        return Calendar.getInstance().getTime();
    }
}
```

Now we can use the profile to do debug:

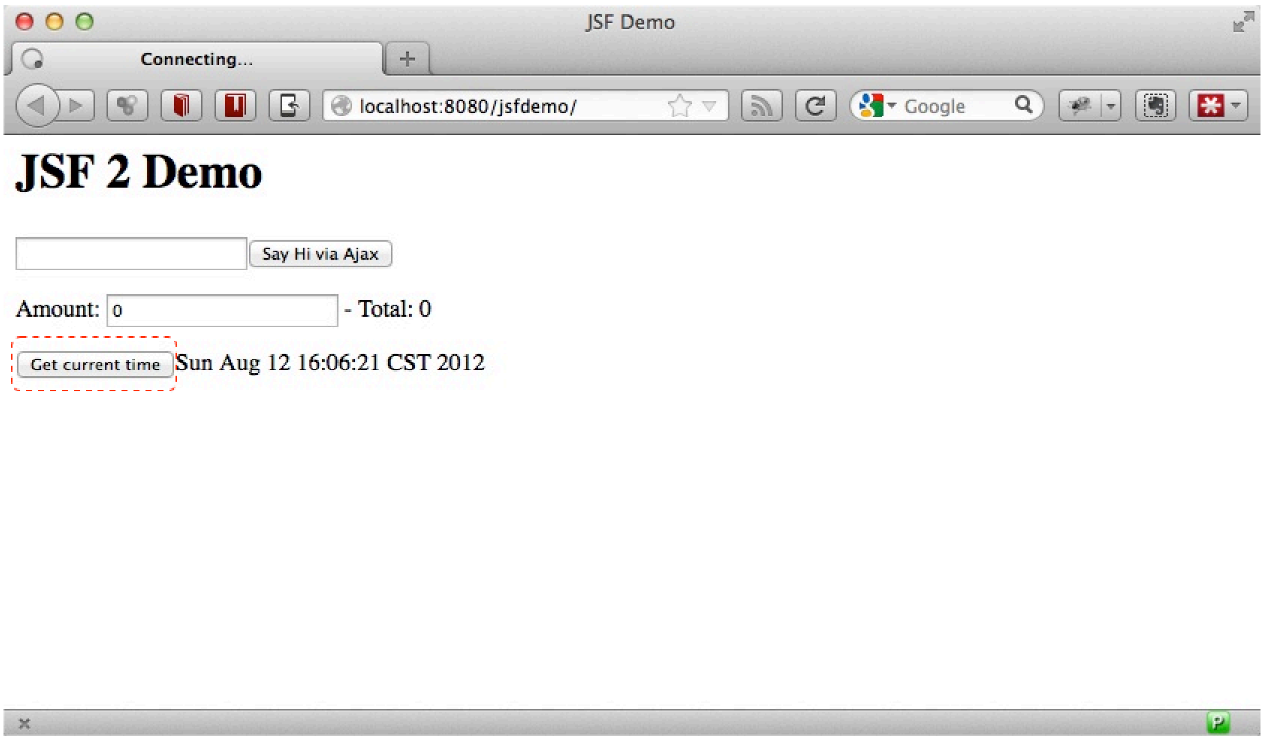

IntelliJ IDEA File Edit View Navigate Code Analyze Refactor
Unnamed
Swtich profile to 'Unnamed' and click debug button

If everything goes fine we can see the console output:



```
Connected to the target VM, address: 'localhost:8787', transport: 'socket'
```

Now we go to web browser and see our project's main page, try to click on 'Get current time':



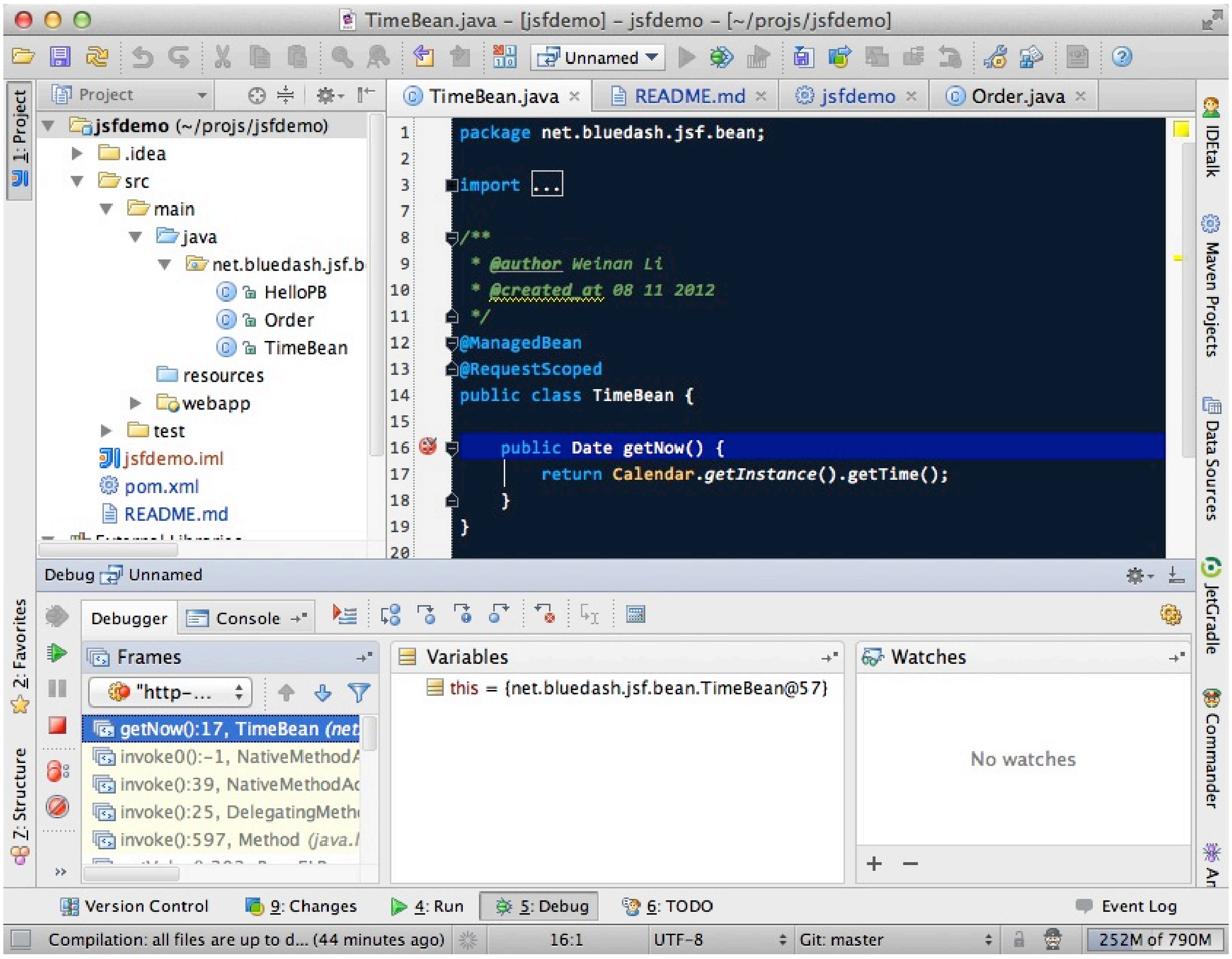
JSF 2 Demo

Say Hi via Ajax

Amount: 0 - Total: 0

Get current time Sun Aug 12 16:06:21 CST 2012

Then IntelliJ will popup and the code is pausing on break point:



```
package net.bluedash.jsf.bean;

import ...

/**
 * @author Weinan Li
 * @created at 08 11 2012
 */
@ManagedBean
@RequestScoped
public class TimeBean {

    public Date getNow() {
        return Calendar.getInstance().getTime();
    }
}
```

```
getNow():17, TimeBean (net
invoke0():-1, NativeMethodA
invoke():39, NativeMethodAc
invoke():25, DelegatingMeth
invoke():597, Method (java.l
```

```
this = {net.bluedash.jsf.bean.TimeBean@57}
```

```
Compilation: all files are up to d... (44 minutes ago)    16:1    UTF-8    Git: master    252M of 790M
```

And we could inspect our project now.

8.1.10 Conclusion

In this article I've shown to you how to use maven to create a project using JSF and deploy it in JBoss AS7, and I've also talked about the usage of IntelliJ during project development phase. Hope the contents are practical and helpful to you 🙂

8.1.11 References

- *JBoss AS7: Using JPDA to debug the AS source code*
- *Importing JBoss 7 Bundled Dependency Versions Through Maven*
- *Maven Getting Started - Developers*
- *JSF 2.1 project using Eclipse and Maven 2:http*
- *Practical RichFaces*
- *Oracle Mojarra JavaServer Faces*
- *JBoss AS7 Maven Plugin*

8.2 Getting Started Developing Applications Presentation & Demo

8.2.1 Introduction

This document is a “script” for use with the quickstarts associated with the Getting Started Developing Applications Guide. It can be used as the basis for demoing/explaining the Java EE 6 programming model with JBoss AS 7.

There is an associated presentation – JBoss AS - Getting Started Developing Applications – which can be used to introduce the Java EE 6 ecosystem.

The emphasis here is on the programming model, not on OAM/dev-ops, performance etc.

8.2.2 Prerequisites for using the script

- JBoss AS 7 downloaded and installed
- Eclipse Indigo with m2eclipse and JBoss Tools installed
- The quickstarts downloaded and imported into Eclipse
- Make sure `$JBOSS_HOME` is set.
- Make sure `src/test/resources/arquillian.xml` has the correct path to your JBoss AS install for kitchensink
- Make sure your font size is set in Eclipse so everyone can read the text!

8.2.3 Import examples into Eclipse and set up JBoss AS

TODO

8.2.4 The Helloworld Quickstart

Introduction

This quickstart is extremely basic, and is really useful for nothing more than showing than the app server is working properly, and our deployment mechanism is working. We recommend you use this quickstart to demonstrate the various ways you can deploy apps to JBoss AS 7.

Using Maven

1. Start JBoss AS 7 from the console

```
$JBOSS_HOME/bin/standalone.sh
```

2. Deploy the app using Maven

```
mvn clean package jboss-as:deploy
```

> The quickstarts use the jboss-as maven plugin to deploy and undeploy applications. This plugin uses the JBoss AS Native Java Detyped Management API to communicate with the server. The Detyped API is used by management tools to control an entire domain of servers, and exposes only a small number of types, allowing for backwards and forwards compatibility.

3. Show the app has deployed in the terminal
4. Visit http://localhost:8080/jboss-as-helloworld
5. Undeploy the app using Maven

```
mvn jboss-as:undeploy
```

Using the Command Line Interface (CLI)

1. Start JBoss AS 7 from the console (if not already running)

```
$JBOSS_HOME/bin/standalone.sh
```

2. Build the war

```
mvn clean package
```

3. Start the CLI

```
$JBOSS_HOME/bin/jboss-admin.sh --connect
```

> The command line also uses the Deptyped Management API to communicate with the server. It's designed to be as "unixy" as possible, allowing you to "cd" into nodes, with full tab completion etc. The CLI allows you to deploy and undeploy applications, create JMS queues, topics etc., create datasources (normal and XA). It also fully supports the domain node.

4. Deploy the app

```
deploy target/jboss-as-helloworld.war
```

5. Show the app has deployed

```
undeploy jboss-as-helloworld.war
```

Using the web management interface

1. Start JBoss AS 7 from the console (if not already running)

   ```
   $JBOSS_HOME/bin/standalone.sh
   ```

2. Build the war

   ```
   mvn clean package
   ```

3. Open up the web management interface http://localhost:9990/console

   > The web maangement interface offers the same functionality as the CLI (and again uses the Detyped Management API), but does so using a pretty GWT interface! You can set up virtual servers, interrogate sub systems and more.

4. Navigate `Manage Deployments -> Add content`. Click on choose file and locate `helloworld/target/jboss-as-helloworld.war`.
5. Click `Next` and `Finish` to upload the war to the server.
6. Now click `Enable` and `Ok` to start the application
7. Switch to the console to show it deployed
8. Now click `Remove`

Using the filesystem

1. Start JBoss AS 7 from the console (if not already running)

```
$JBOSS_HOME/bin/standalone.sh
```

2. Build the war

```
mvn clean package
```

> Of course, you can still use the good ol' file system to deploy. Just copy the file to `$JBOSS_HOME/standalone/deployments`.

3. Copy the war

```
cp target/jboss-as-helloworld.war $JBOSS_HOME/standalone/deployments
```

4. Show the war deployed

> The filesystem deployment uses marker files to indicate the status of a deployment. As this deployment succeeded we get a `$JBOSS_HOME/standalone/deployments/jboss-as-helloworld.war.deployed` file. If the deployment failed, you would get a `.failed` file etc.

5. Undeploy the war

```
rm $JBOSS_HOME/standalone/deployments/jboss-as-helloworld.war.deployed
```

6. Show the deployment stopping!
7. Start and stop the appserver, show that the deployment really is gone!

> This gives you much more precise control over deployments than before

Using Eclipse

1. Add a JBoss AS server
    1. Bring up the Server view
    2. Right click in it, and choose `New -> Server`
    3. Choose JBoss AS 7.0 and hit Next
    4. Locate the server on your disc
    5. Hit Finish
2. Start JBoss AS in Eclipse
    1. Select the server
    2. Click the Run button
3. Deploy the app
    1. right click on the app, choose `Run As -> Run On Server`
    2. Select the AS 7 instance you want to use
    3. Hit finish
4. Load the app at http://localhost:8080/jboss-as-helloworld

Digging into the app

1. Open up the helloworld quickstart in Eclipse, and open up `src/main/webapp`.
2. Point out that we don't require a `web.xml` anymore!
3. Show `beans.xml` and explain it's a marker file used to JBoss AS to enable CDI (open it, show that it is empty)
4. Show `index.html`, and explain it is just used to kick the user into the app (open it, show the meta-refresh)
5. Open up the `pom.xm` - and emphasise that it's pretty simple.
    1. There is no parent pom, everything for the build is **here**
    2. Show that we are enabling the JBoss Maven repo - explain you can do this in your POM or in system wide (`settings.xml`)
    3. Show the `dependencyManagement` section. Here we import the JBoss AS 7 Web Profile API. Explain that this gives you all the versions for all of the JBoss AS 7 APIs that are in the web profile. Explain we could also depend on this directly, which would give us the whole set of APIs, but that here we've decided to go for slightly tighter control and specify each dependency ourselves
    4. Show the import for CDI, JSR-250 and Servlet API. Show that these are all provided - we are depending on build in server implementations, not packaging this stuff!
    5. Show the plugin sections - nothing that exciting here, the war plugin is out of date and requires you to provide `web.xml` 😉 , configure the JBoss AS Maven Plugin, set the Java version to 6.
6. Open up `src/main/java` and open up the `HelloWorldServlet`.
    1. Point out the `@WebServlet` - explain this one annotation removes about 8 lines of XML - no need to separately map a path either. This is much more refactor safe
    2. Show that we can inject services into a Servlet
    3. Show that we use the service (line 41)
       #Cmd-click on `HelloService`
    4. This is a CDI bean - very simple, no annotations required!
    5. Explain injection
        1. Probably used to string based bean resolution
        2. This is typesafe (refactor safe, take advantage of the compiler and the IDE - we just saw that!)
        3. When CDI needs to inject something, the first thing it looks at is the type - and if the type of the injection point is assignable from a bean, CDI will inject that bean

8.2.5 The numberguess quickstart

Introduction

This quickstart adds in a "complete" view layer into the mix. Java EE ships with a JSF. JSF is a server side rendering, component orientated framework, where you write markup using an HTML like language, adding in dynamic behavior by binding components to beans in the back end. The quickstart also makes more use of CDI to wire the application together.

Run the app

1. Start JBoss AS in Eclipse
2. Deploy it using Eclipse - just right click on the app, choose `Run As -> Run On Server`
3. Select the AS 7 instance you want to use
4. Hit finish
5. Load the app at http://localhost:8080/jboss-as-numberguess
6. Make a few guesses

Deployment descriptors src/main/webapp/WEB-INF

Emphasize the lack of them!

No need to open any of them, just point them out

1. `web.xml` - don't need it!
2. `beans.xml` - as before, marker file
3. `faces-config.xml` - nice feature from AS7 - we can just put `faces-config.xml` into the WEB-INF and it enables JSF (inspiration from CDI)
4. `pom.xml` we saw this before, this time it's the same but adds in JSF API

Views

1. `index.html` - same as before, just kicks us into the app
2. `home.xhtml`
    1. Lines 19 - 25 – these are messages output depending on state of beans (minimise coupling between controller and view layer by interrogating state, not pushing)
3. Line 20 – output any messages pushed out by the controller
4. Line 39 - 42 – the input field is bound to the guess field on the game bean. We validate the input by calling a method on the game bean.
5. Line 43 - 45 – the command button is used to submit the form, and calls a method on the game bean
6. Line 48, 49, The reset button again calls a method on the game bean

Beans

1. `Game.java` – this is the main controller for the game. App has no persistence etc.
    1. `@Named` – As we discussed CDI is typesafe, (beans are injected by type) but sometimes need to access in a non-typesafe fashion. @Named exposes the Bean in EL - and allows us to access it from JSF
    2. `@SessionScoped` – really simple app, we keep the game data in the session - to play two concurrent games, need two sessions. This is not a limitation of CDI, but simply keeps this demo very simple. CDI will create a bean instance the first time the game bean is accessed, and then always load that for you
    3. `@Inject maxNumber` – here we inject the maximum number we can guess. This allows us to externalize the config of the game
    4. `@Inject rnadomNumber` – here we inject the random number we need to guess. Two things to discuss here
        1. Instance - normally we can inject the object itself, but sometimes it's useful to inject a "provider" of the object (in this case so that we can get a new random number when the game is reset!). Instance allows us to `get()` a new instance when needed
        2. Qualifiers - now we have two types of Integer (CDI auto-boxes types when doing injection) so we need to disambiguate. Explain qualifiers and development time approach to disambiguation. You will want to open up `@MaxNumber` and `@Random` here.
    5. `@PostConstruct` – here is our reset method - we also call it on startup to set up initial values. Show use of `Instance.get()`.
2. `Generator.java` This bean acts as our random number generator.
3. `@ApplicationScoped` explain about other scopes available in CDI + extensibility.
    1. `next()` Explain about producers being useful for determining bean instance at runtime
    2. `getMaxNumber()` Explain about producers allowing for loose coupling

8.2.6 The login quickstart

Introduction

The login quickstart builds on the knowledge of CDI and JSF we have got from numberguess. New stuff we will learn about is how to use JPA to store data in a database, how to use JTA to control transactions, and how to use EJB for declarative TX control.

Run the app

1. Start JBoss AS in Eclipse
2. Deploy it using Eclipse - just right click on the app, choose `Run As -> Run On Server`
3. Select the AS 7 instance you want to use
4. Hit finish
5. Load the app at http://localhost:8080/jboss-as-login
6. Login as admin/admin
7. Create a new user

Deployment Descriptors

1. Show that we have the same ones we are used in `src/main/webapp` – `beans.xml`, `faces-config.xml`
2. We have a couple of new ones in `src/main/resources`
    1. `persistence.xml`. Not too exciting. We are using a datasource that AS7 ships with. It's backed by the H2 database and is purely a sample datasource to use in sample applications. We also tell Hibernate to auto-create tables - as you always have.
    2. `import.sql` Again, the same old thing you are used to in Hibernate - auto-import data when the app starts.
3. `pom.xml` is the same again, but just adds in dependencies for JPA, JTA and EJB

Views

1. `template.xhtml` One of the updates added to JSF 2.0 was templating ability. We take advantage of that in this app, as we have multiple views
    1. Actually nothing too major here, we define the app "title" and we could easily define a common footer etc. (we can see this done in the kitchensink app)
    2. The `ui:insert` command inserts the actual content from the templated page. `#home.xhtml`
    3. Uses the template
    4. Has some input fields for the login form, button to login and logout, link to add users.
    5. Binds fields to credentials bean}}
    6. Buttons link to login bean which is the controller
2. `users.xhtml`
    1. Uses the template
    2. Displays all users using a table
    3. Has a form with input fields to add users.
    4. Binds fields to the newUser bean
    5. Methods call on userManager bean

Beans

1. `Credentials.java` Backing bean for the login form field, pretty trivial. It's request scoped (natural for a login field) and named so we can get it from JSF.
2. `Login.java`
    1. Is session scoped (a user is logged in for the length of their session or until they log out}}
    2. Is accessible from EL
    3. Injects the current credentials
    4. Uses the userManager service to load the user, and sends any messages to JSF as needed
    5. Uses a producer method to expose the @LoggedIn user (producer methods used as we don't know which user at development time)
3. `User.java` Is a pretty straightforward JPA entity. Mapped with `@Entity`, has an natural id.
4. `UserManager.java` This is an interface, and by default we use the ManagedBean version, which requires manual TX control
5. `ManagedBeanUserManager.java` - accessible from EL, request scoped.
    1. Injects a logger (we'll see how that is produced in a minute)
    2. Injects the entity manager (again, just a min)
    3. Inject the UserTransaction (this is provided by CDI)
    4. `getUsers()` standard JPA-QL that we know and love - but lots of ugly TX handling code.
    5. Same for `addUser()` and `findUser()` methods - very simple JPA but...
    6. Got a couple of producer methods.
        1. `getUsers()` is obvious - loads all the users in the database. No ambiguity - CDI takes into account generic types when injecting. Also note that CDI names respect JavaBean naming conventions
        2. `getNewUser()` is used to bind the new user form to from the view layer - very nice as it decreases coupling - we could completely change the wiring on the server side (different approach to creating the newUser bean) and no need to change the view layer.
6. `EJBUserManager.java`
    1. It's an alternative – explain alternatives, and that they allow selection of beans at deployment time
    2. Much simple now we have declarative TX control.
    3. Start to see how we can introduce EJB to get useful enterprise services such as declarative TX control
7. `Resources.java`
    1. {EntityManager}} - explain resource producer pattern

8.2.7 The kitchensink quickstart

Introduction

The kitchensink quickstart is generated from an archetype available for JBoss AS (tell people to check the [Getting Started Developing Applications] Guide for details). It demonstrates CDI, JSF, EJB, JPA (which we've seen before) and JAX-RS and Bean Validation as well. We add in Arquillian for testing.

Run the app

1. Start JBoss AS in Eclipse
2. Deploy it using Eclipse - just right click on the app, choose `Run As -> Run On Server`
3. Select the AS 7 instance you want to use
4. Hit finish
5. Load the app at http://localhost:8080/jboss-as-kitchensink
6. Register a member - make sure to enter an invalid email and phone - show bean validation at work
7. Click on the member URL and show the output from JAX-RS

Bean Validation

1. Explain the benefits of bean validation - need your data always valid (protect your data) AND good errors for your user. BV allows you to express once, apply often.
2. `index.xhtml`
    1. Show the input fields – no validators attached
    2. Show the message output
3. `Member.java`
    1. Hightlight the various validation annotations
4. Java EE automatically applies the validators in both the persistence layer and in your views

RS

1. `index.xhtml` - Show that URL generation is just manual
2. `JaxRsActivator.java` - simply activates JAX-RS
3. `Member.java` - add JAXB annotation to make JAXB process the class properly
4. `MemberResourceRESTService.java`
    1. `@Path` sets the JAX-RS resource
    2. JAX-RS services can use injection
    3. `@GET` methods are auto transformed to XML using JAXB
5. And that is it!

Arquillian

1. Make sure JBoss AS is running
2.
```
mvn clean test -Parq-jbossas-remote
```
    1. Explain the difference between managed and remote
3. Make sure JBoss AS is stopped
4.
```
mvn clean test -Parq-jbossas-managed
```
5. Start JBoss AS in Eclipse
6. Update the project to use the `arq-jbossas-remote` profile
7. Run the test from Eclipse
    1. Right click on test, `Run As -> JUnit Test`
8. `MemberRegistrationTest.java`
    1. Discuss micro deployments
    2. Explain Arquilian allows you to use injection
    3. Explain that Arquillian allows you to concentrate just on your test logic