WildFly Troubleshooting

In this chapter, let's see how to debug the WildFly runtime. The methods introduced here also can be applied to JBoss EAP6.

Downloading WildFly

We'll use the newest version of WildFly for troubleshooting. You can navigate to `http://www.wildfly.org/downloads/` to download it.

At the time of writing this book, the newest version of WildFly is 8.0.0.Beta1, hence I will use this version. The download page is as shown in the following screenshot:

Please download both the server distribution and its source code. After they are downloaded, please extract them into their proper place.

It's important to use the same version for the binary and source code.

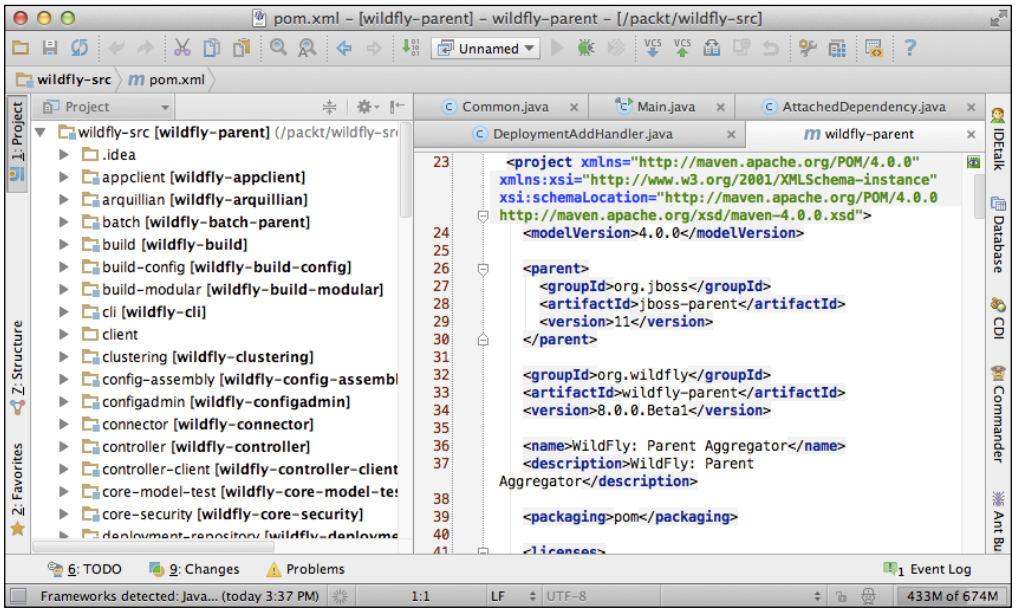
Debugging WildFly in the standalone mode

Now let's run the WildFly that we've downloaded. WildFly has provided a `debug` option for us to use:

```
$ ./standalone.sh --debug --server-config=standalone.xml
Listening for transport dt_socket at address: 8787
```

Please note that we need to specify the configuration file that we are going to use when WildFly is running in the debug mode. As shown in the preceding process, WildFly is listening to the debug port 8787. Now let's set up our IDE to debug the server. Both Eclipse and IntelliJ supports a remote debug. I'll mainly use IntelliJ to show the process, and I will introduce the usage of Eclipse briefly.

First, please import the source code of WildFly into IntelliJ. Because the WildFly source code is managed by Maven, it could be easily imported into any popular IDE. For IntelliJ, it can directly import the Maven project. The imported project will look like the following screenshot:



For Eclipse, we can use the Maven command to generate the Eclipse project files. Please run the following command in the source directory of WildFly to build it:

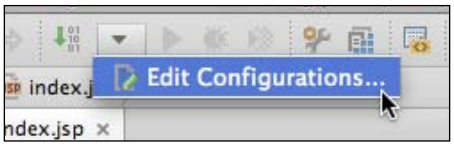
```
wildfly-src$ ./build.sh -DskipTests
```

It will download many dependencies to your machine and it will take 10 minutes to several hours to finish the building process, depending on your network speed and CPU power. After the building process is finished, please run the following Maven command to generate the Eclipse project files:

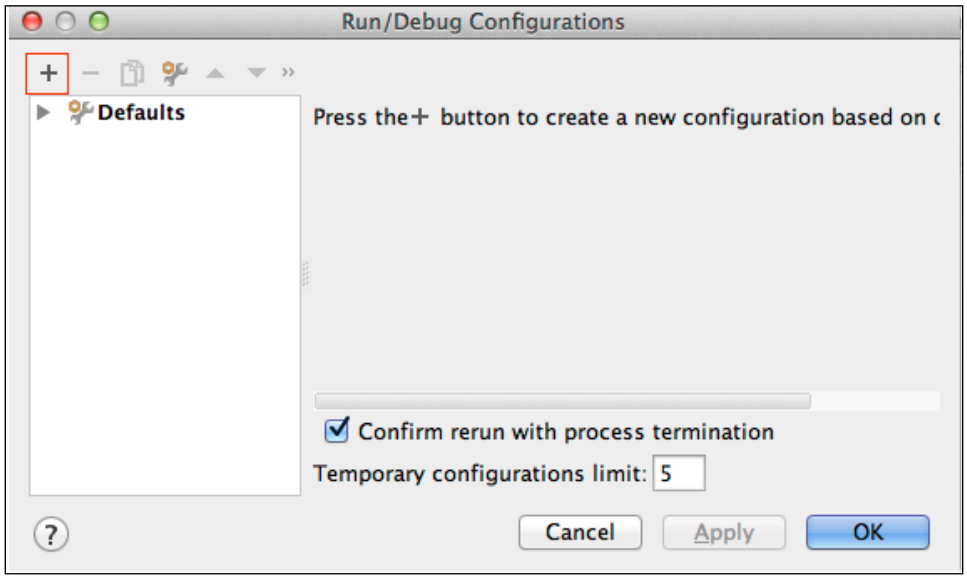
```
wildfly-src$ mvn eclipse:eclipse
```

After the process is finished, you can use the `Import Existing Project` function of Eclipse to import the WildFly project.

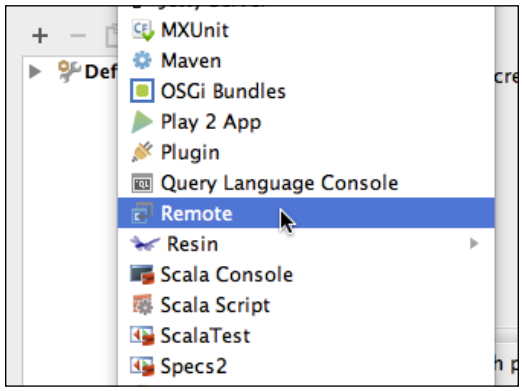
Now let's enable the remote debug in IntelliJ. Please click on the **Run/Debug Configurations** button, and choose **Edit Configurations…**. The process is shown in the following screenshot:



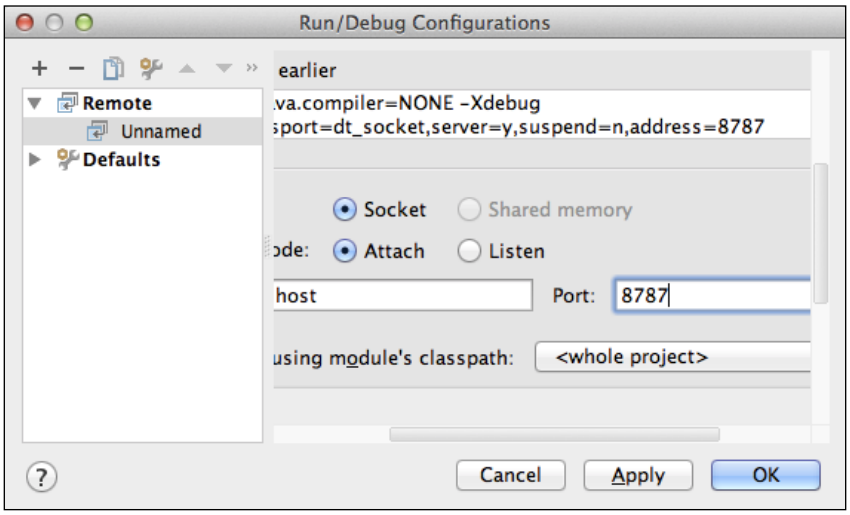
Then the configuration window will pop up. We need to add a configuration, so please click on the **+**.



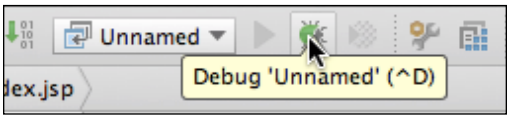
We need to connect to the remote debug port of WildFly, so please choose **Remote**.



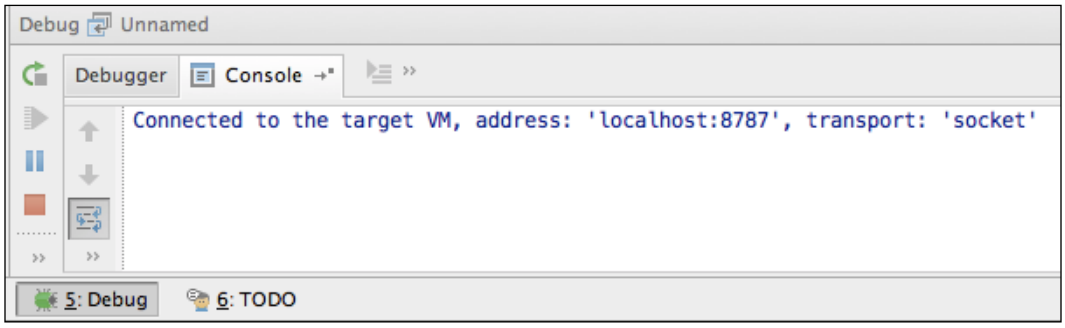
In the remote configuration, please edit the **Port:** parameter and change it to `8787`. The configuration is as shown in the following screenshot:



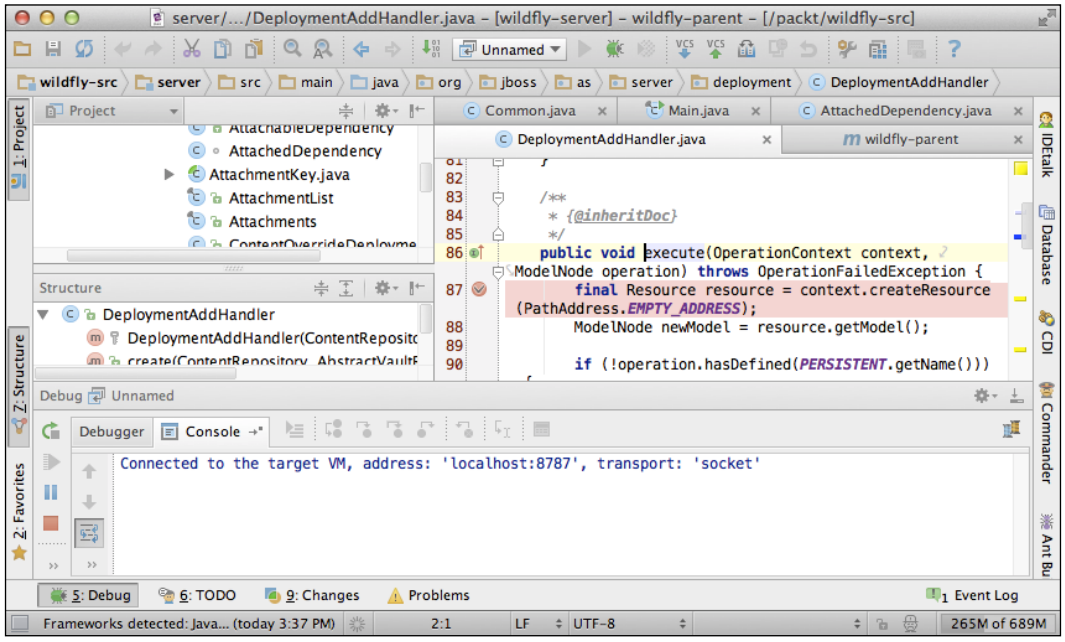
That's all for the configuration. Please save it and now we can use it to debug. Please click on the debug button.



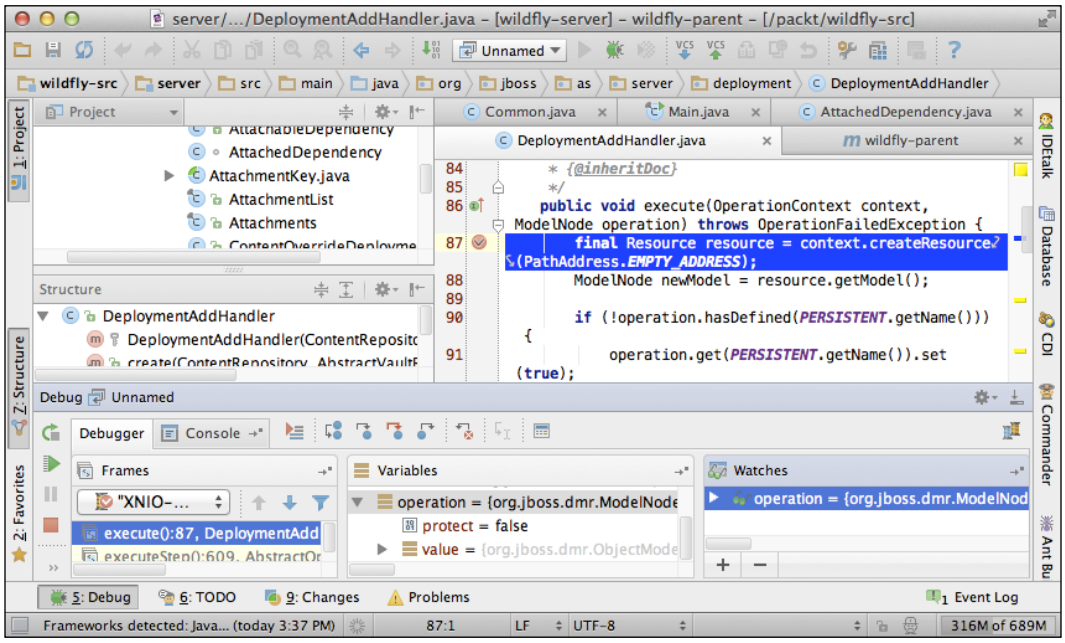
Then we can see that IntelliJ is connected to the WildFly debug port as shown in the following screenshot:



Now we can set a breakpoint in the source code of WildFly. For example, if I want to check the project deployment process, I'll set a breakpoint on the `execute()` method of `org.jboss.as.server.deployment.DeploymentAddHandler` as shown here in the following screenshot:

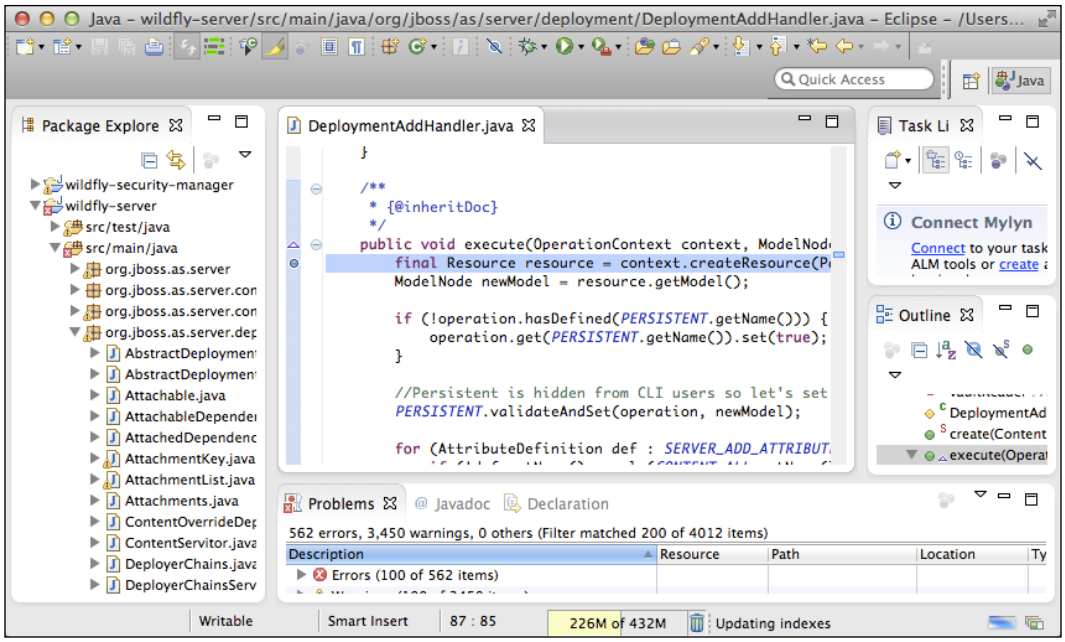


Then when I deploy a project into WildFly, the process will pause on the breakpoint.

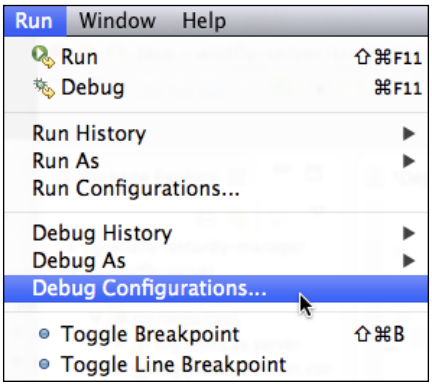


From the preceding screenshot, we can see that the remote debug is a powerful tool to debug WildFly in runtime. The technique used by WildFly is called **Java Platform Debugger Architecture**. This is the standard debug framework for Java. You can search JPDA on Google to learn more about it.

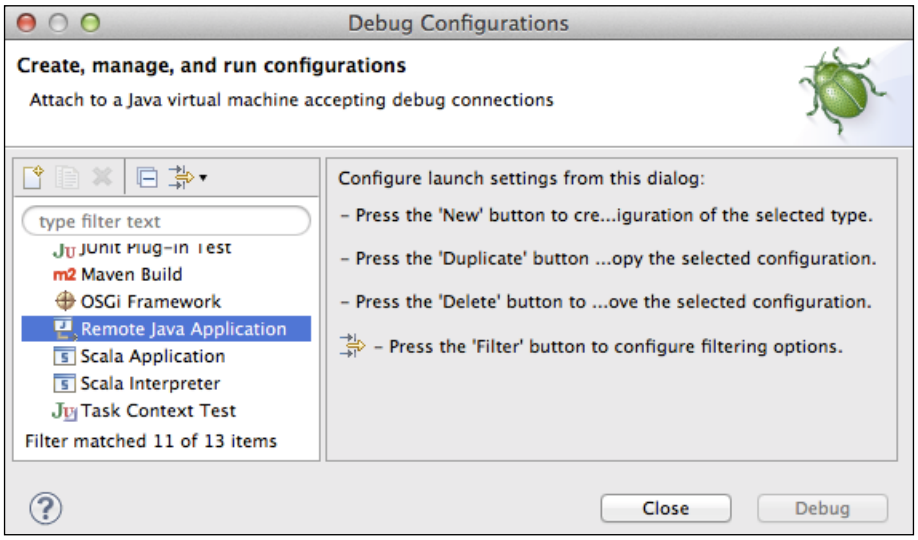
Now let's check how to process a remote debug in Eclipse. The first step is the same—start WildFly with the debug option enabled. Then we set the breakpoint on the line of code we want, as shown in the following screenshot:



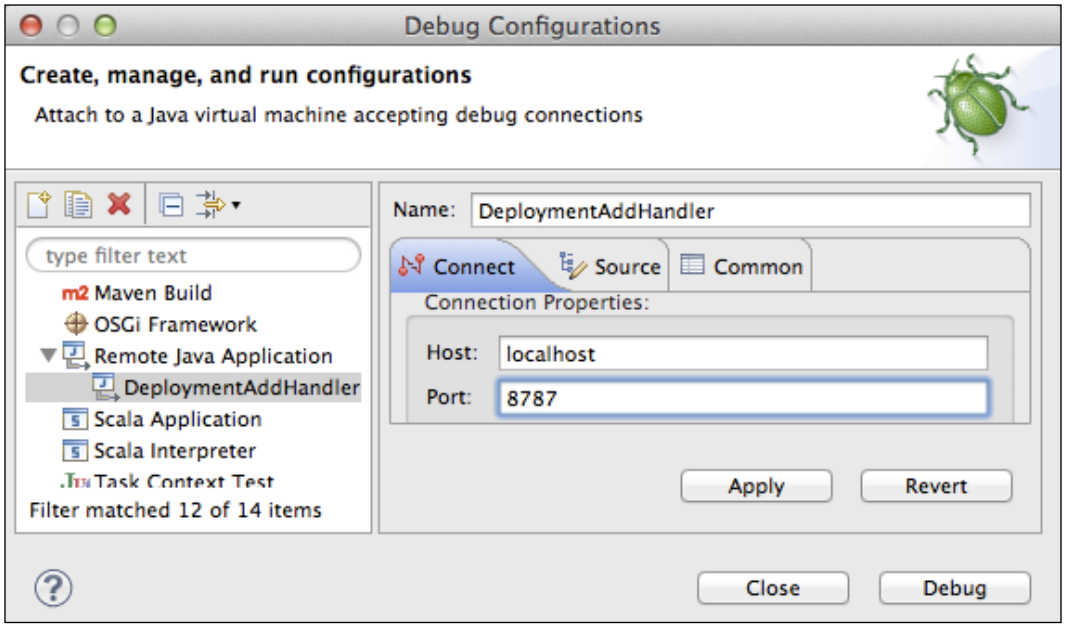
As shown in the preceding screenshot, we also put the breakpoint on the project deployment code. Then select **Run** from the menu and click on **Debug Configurations...**.



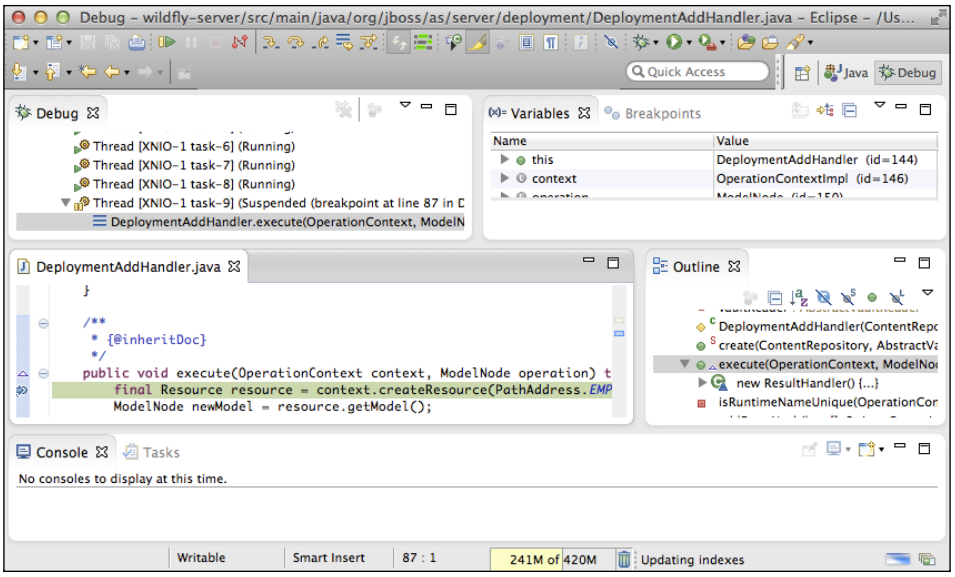
Then we need to select **Remote Java Application** as shown in the following screenshot:



Then we change the debug port in the **Port:** field to `8787`.

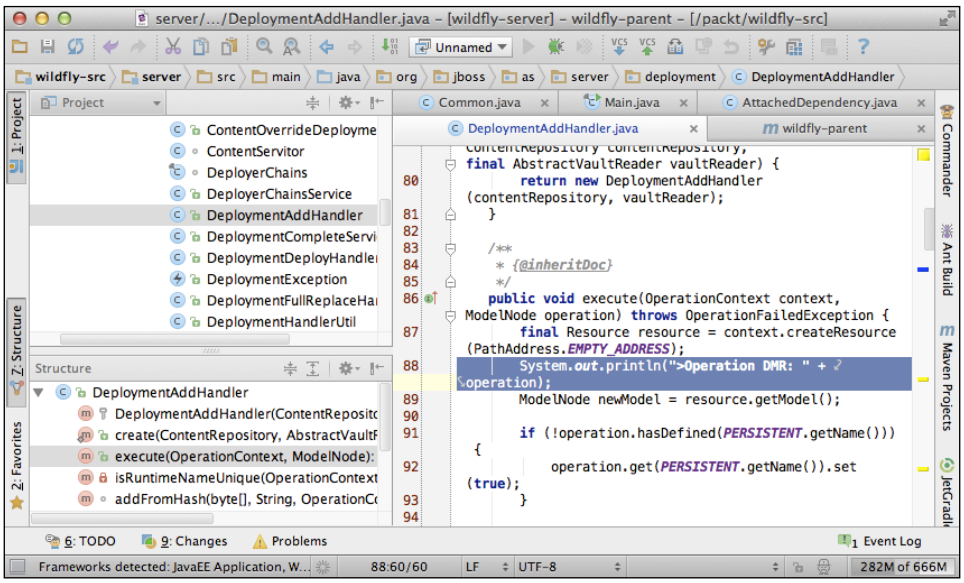


Now we can deploy a project and Eclipse will pause WildFly on the breakpoint as shown in the following screenshot:



Building and replacing WildFly modules

Sometimes we may want to modify some parts of WildFly. Thanks to the modular design of WildFly, we just need to rebuild the module we've patched and replace the existing one. For example, let's add a line of code in **DeploymentAddHandler** as shown in the following screenshot:



As shown in the preceding screenshot, we have added a line of code to print the operation object. Because this class is in the server module, we just need to use the Maven command to rebuild it:

```
wildfly-src/server$ mvn clean install -DskipTests
```

After the build process is finished, we get the JAR file of this module as follows:

```
wildfly-src/server$ ls target/wildfly-server-8.0.0.Beta1.jar

target/wildfly-server-8.0.0.Beta1.jar
```

Now we need to use this JAR file to replace the existing one in WildFly. First let's back up the old one by using the following command:

```
wildfly$ cd modules/system/layers/base/org/jboss/as/server/main/
wildfly$ mv wildfly-server-8.0.0.Beta1.jar wildfly-server-
  8.0.0.Beta1.jar.orig
```

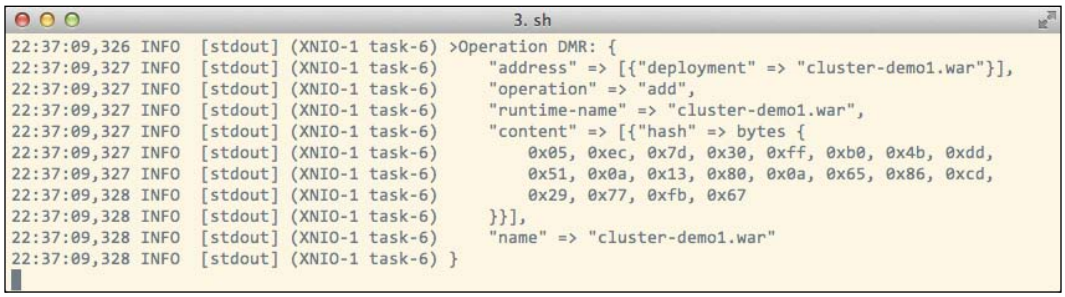
Then we need to copy the new one into the directory by using the following command:

```
wildfly/modules/system/layers/base/org/jboss/as/server/main$ cp
  /packt/wildfly-src/server/target/wildfly-server-8.0.0.Beta1.jar
  .
```

If the name of the module changes, we'll also need to update `module.xml` of this module as follows:

```
<resource-root path="wildfly-server-8.0.0.Beta1.jar"/>
```

Because our patched JAR file has the same name as the original one, we don't need to edit it. Now let's restart WildFly and deploy a project. We can see that our print command is working as shown in the following screenshot:



Debugging a project in the domain mode

WildFly doesn't provide a debug option in `domain.sh`. Because there are multiple server processes running in parallel in the domain mode, we should tell WildFly which server we want to debug. We can open `host.xml` and see a sample configuration for debug as follows:

```
<server name="server-one" group="main-server-group" auto-
  start="true">
  <!-- Remote JPDA debugging for a specific server
  <jvm name="default">
  <jvm-options>
    <option value="-
      agentlib:jdwp=transport=dt_socket,address=8787,
      server=y,suspend=n"/>
  </jvm-options>
  </jvm>
  -->
<socket-bindings port-offset="0"/>
</server>
```

After enabling the preceding section, we can remote debug server-one by using the port 8787. We can also use the sample configuration in other server sections, and just make sure they are using a different port for debugging. Then we can open multiple IDE instances and ask them to connect to different ports to debug these servers at the same time.