EL4J

GettingStartedDeveloper

Generated from Wiki page:
http://wiki.elca.ch/twiki/el4j/bin/view/EL4J/GettingStartedDeveloper

Version	Date	Author(s)	Visa
1.4	18 Dec 2007 - 15:26	ClaudeHumard, Main.ClaudeHumard	

Table of Contents

1 Getting Started for EL4J Developers

This section is intended for EL4J developers who work on the EL4J framework itself. If you are an EL4J user and want to build your own application on top of the EL4J framework, you don't necessarily need the EL4J source code (except if you want to view/change EL4J modules).

Prior to this section, you should have worked through the GettingStarted guide and you should have all necessary tools. Now, you will download the sources for EL4J framework, set up Eclipse and the Maven 2 repository for development.

1.1 Setting up EL4J

1.1.1 Download sourcecode of EL4J framework (external part)

To download the source code, subversion is required. See the corresponding section in the GettingStarted#Subversion guide.

- Open a cygwin console and change to your `D:\Projects\EL4J` directory.
- Check out the external repository with the following command:
    - `svn co https://el4j.svn.sourceforge.net/svnroot/el4j/trunk/el4j external`

1.1.2 Change settings

- Go to `~/.m2` directory. There should be a `settings.xml` file, which has been copied from the EL4J maven convenience-zip by the shell-script `finishInstallation.sh` executed in the GettingStarted. If there is no `settings.xml`, copy it from `D:\Projects\EL4J\external\etc\m2\` to your `~/.m2` directory.
- Change the `settings.xml` file in your `~/.m2` directory as follows:
    - Change the value of the `localRepository` tag to `D:/m2repository`
    - Remove the comments around the `proxy` tag
    - Remove the comments around the `el4j.root`, `el4j.external` and `el4j.internal` tags (under profile `el4j.general`)
    - Remove the comments around the `mirror` tags, as described in the xml comments
    - Change the value of `el4j.root` to `D:/Projects/EL4J`
    - The value of `el4j.external` should be `${el4j.root}/external`
    - Change the value of `el4j.project.home` to `${el4j.root}`

1.1.3 JDBC Drivers

Note for internal developers: skip this section.

If you are not inside the ELCA network, you will have to download the JDBC Database Drivers for Oracle and Derby/DB2 yourself, because we are not allowed to distribute them. To download and deploy these drivers into your local repository, do the following:

- Download the Oracle driver (ojdbc14.jar) from "http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/". Note: You can also use the 10g driver for version 9 database.
- Download the DB2/Derby JDBC driver (db2jcc.jar and db2jcc_license_c.jar) from "http://www.ibm.com/developerworks/db2/downloads/jcc/".
- Open a console and change to the directory where you have saved the downloaded jar files.
- Execute the following commands:
    - `mvn install:install-file -DgroupId=com.oracle -DartifactId=ojdbc14_g -Dversion=10.2.0.1.0 -Dpackaging=jar -Dfile=ojdbc14_g.jar`

- `mvn install:install-file -DgroupId=com.ibm -DartifactId=db2jcc -Dversion=20040819 -Dpackaging=jar -Dfile=db2jcc.jar`
- `mvn install:install-file -DgroupId=com.ibm -DartifactId=db2jcc_license_c -Dversion=20040819 -Dpackaging=jar -Dfile=db2jcc_license_c.jar`

Note that the used file names of the jar files in the commands above may depend on the downloaded version. The used version number (`10.2.0.1.0`) is the one used in `module-database`. Leave this version number as it is to avoid dependency conflicts.

1.1.4 Build project

- Go to `D:\Projects\EL4J\external` in your cygwin console and type `mvn clean install`. This will probably take a while (~30min) and will build the external part of the EL4J framework.
- Type `mvn eclipse:clean eclipse:eclipse -DdownloadSources=true` to let Maven 2 generate Eclipse project files.

1.1.5 Import EL4J modules into Eclipse

You could add all EL4J modules into your workspace, but this is very confusing. Therefore, we recommend the following:

- Open Eclipse with your `D:\Projects\EL4J\workspace` workspace.
- Go to `Window -> Preferences`
- Go to `Java -> Build Path -> Classpath Variable`
- Add a new variable with the name `M2_REPO` and the path `D:\m2repository`
- Add another variable, set the name to `EL4J_HOME` and the path to `D:\Projects\EL4J`

- Close the Preferences window and click on the little triangle in the uppermost right corner of your `Package Explorer`.
- Go to `Select Working Sets...`.
- Create the working sets `modules`, `applications`, `demos`, `tests` and `plugins` (`New -> Java`).
- Click on the triangle again and choose `Top Level Elements -> Working Sets`.

- Go to `File -> Import` and choose Existing Projects into Workspace
- Click on Next and on the next page on Browse next to "Select root directory". Go to `D:\Projects\EL4J\external\framework\modules`, click on Ok and then Finish.
- Move the projects in your `modules` set.
- Repeat the same with
  - `D:\Projects\EL4J\external\framework\tests` (for `tests`),
  - `D:\Projects\EL4J\external\applications\demos` (for `demos`)
  - `D:\Projects\EL4J\external\applications\templates\common` (for `applications`).
  - `D:\Projects\EL4J\external\maven` (for `plugins`).
- Additionally you can exclude external libraries. Click on the triangle and on `Filters...`. Choose `Libraries from external` there.

1.1.6 Setup Checkclipse

- Open Eclipse and go to `Window -> Preferences` and there to `Checkclipse`
- Add `D:\Projects\EL4J\external\etc\checkstyle\checks.xml` as the `Checkstyle Configuration File`
- Add `D:\Projects\EL4J\external\etc\checkstyle\checks.properties` as the `Checkstyle Properties File`

1.1.7 Internal developers

Note for internal developers: in addition to this section, please follow the corresponding section in the InternalGettingStarted#Setting_up_EL4J guide.

1.2 Initial development

By now, you should

- Have a local copy of the EL4J repository (don't forget to update now and then with `svn up`)
- This copy of EL4J should compile with `mvn clean install` without errors
- Have set up Eclipse to work with EL4J
- Understand the basic concepts of Maven and be able to include new dependencies and use them
- A basic understanding of Spring, especially about the `Application Context`, about the use of configuration files and IoC?.

If so, you're ready for the next step - go directly into EL4J! You will learn the structure of the EL4J framework, get to know some of the EL4J Demos and learn how to debug a project.

1.2.1 The EL4J framework

First, the EL4J framework has following structure:

- `applications`
  - `templates` Contains the two examples `keyword` and `refdb` out of which we create our templates
  - `demos` Demos that explain a specific functionality of the framework.
- `etc` Contains additional content like the checkclipse files, log4j configuration, etc.
- `framework`
  - `modules` The framework modules of EL4J external
  - `tests` (Integration) Tests, which test two or more (framework) modules.
- `maven`
  - `archetypes` The archetype you used earlier
  - `helpers` Some helpers you don't have to worry about now
  - `plugins` Maven plugins that were developed by the EL4J team
- `sandbox` The place where we try out new things
- `site` Configuration and additional documents for the website generation
- `skin` The "skin" of the website
- `src` Source folder for the website generation. This will hopefully be removed in the future.

1.2.2 EL4J Demos

EL4J comes with a few demos that show how to use a specific feature of EL4J (like the statistics functionality). You will find them all in your demo working set in Eclipse. They are all executable. Please read the corresponding `README.txt` files for further instructions.
You could try to take a closer look at the `Benchmark Demo`. How is remoting done in EL4J? What kind of protocols does EL4J support? What is `Implicit Context Passing`?

Note for internal developers: for additional material, see the web application template section in the InternalGettingStarted#Web_Application_Template guide.

1.2.3 Developing with Eclipse

Eclipse should only be used to write code and test small parts of the project. Most other development tasks should be executed with Maven, especially the unit tests due to the following reasons:

- Eclipse projects *do not separate compile and test scope* as Maven does. This can be dangerous, for example if the directory test resources contain Spring bean xml files in the mandatory directory.
- Maven does always have dependent jar artifacts as jar files in the classpath. In Eclipse, depending to execution level/directory of the goal mvn eclipse:eclipse , some dependencies are in classpath as jar and some directly as directory with its classes. The test classes itself are always in classpath via directory.
- Eclipse has its own compiler. There are some cases, for example with Java 5 syntax, that tests work only if the classes are compiled with the Eclipse compiler. If they are compiled with a Sun s compiler, the tests fail. At the end tests should work with both compilers (so using the stricter compiler (as with maven) improves compatibility).

1.2.4 Debugging

Maven allows you to debug any executed command in Eclipse. To do so you have to:

- Call `debugmaven` in your cygwin console
- Set your breakpoints in Eclipse
- Invoke the Maven command that you want, e.g. `mvn clean install`
- Go to `Run -> Debug...` in Eclipse.
- There, create a new `Remote Java Application`
- Set the `Connection Properties` Host: `localhost` and Port: `8000`
- Click on "Apply" and "Debug"

More info on this can be found under DebuggingHowTo.

Note for internal developers: You can debug a Maven command at the Leaffy Server by changing the Host to `leaffy` as well

-- ClaudeHumard - 18 Dec 2007