Modules

Table of contents

1 Introduction....2
2 Implementing a module....2
3 The Module Descriptor File....3
4 Implementing Usecases in Modules....3
4.1 Declaring the Usecase....3
4.2 Calling the Module Sitemap....4
4.3 Adding Menu Items....4

1. Introduction

Modules are packages providing a certain set of resources or functionality.

Examples:

- a resource type (e.g., docbook module)
- a repository implementation (e.g., jdbc module)
- a collection of XSLTs (e.g., content2svg module)

Some modules are included in the Lenya distribution:

- **lucene** - search functionality
- **sitetree** - manage documents in a tree-like structure
- **jcr** - store content in a JCR (http://www.jcp.org/en/jsr/detail?id=170) repository
- **xhtml** - XHTML-based resource type
- **links** - resource type to manage link lists
- **lenyadoc** - adds the `lenyadoc://` protocol

2. Implementing a module

All resources of a module are located in a single directory. The following is an example directory structure, all files are optional and depend on the nature of the module.

```
mymodule/
  config/                   configuration files
    module.xml              module descriptor
    cocoon-xconf/           patches for cocoon.xconf
      components.xconf
  usecases/
    mymodule.jx             usecase view
  resources/
    images/                 image files
    css/                    CSS files
    schemas/                XML schemas (RNG, XSD, ...)
    samples/                Samples (in case of resource type modules)
  java/
    src/                    Java source files
    test/                   Java test classes
    lib/                    Java libraries
  xslt/                     XSLT stylesheets
  sitemap.xmap              main module sitemap
```

To add a module to your Lenya installation, declare it in `local.build.properties`:

```
modules.root.dirs=src/modules:src/webapp/lenya/pubs/default/modules:/home/john/modules/mymodu
```

When the module is deployed, the following steps are executed:

- the module files are copied to `context://lenya/modules`
- Java sources are compiled, libraries are installed
- `cocoon.xconf` is patched

3. The Module Descriptor File

Each module must be described using a *module descriptor* XML file `module.xml`, located in the `config` directory of the module. The descriptor is validated when the module is deployed, so be careful! :)

A typical module descriptor looks like this:

```
<module xmlns="http://apache.org/lenya/module/1.0">
  <id>org.myproject.lenya.modules.myeditor</id>
  <depends module="org.apache.lenya.modules.usecase"/>
  <depends module="org.apache.lenya.modules.webdav"/>
  <export package="org.myproject.lenya.modules.myeditor.api"/>
  <package>org.myproject.lenya.modules</package>
  <version>0.1-dev</version>
  <name>My Own Editor</name>
  <lenya-version>1.4-dev</lenya-version>
  <description>
    This is my own editor.
    For more information, visit http://myproject.org/editor.
  </description>
</module>
```

The `id` must start with the `package` of the module. It is not allowed to deploy two modules with the same ID.

If your module uses code from other modules, you have to add a `<depends>` for each of these modules.

The `<export package="..."/>` statement(s) declare public packages, which means that these packages are accessible from other modules. You should aim for long-term stability of all interfaces and classes in exported packages, since they represent the Java API of your module.

4. Implementing Usecases in Modules

4.1. Declaring the Usecase

For example, imagine you implement a `newsletter` module, containing a `send` usecase. To declare usecases, add a patch file for `cocoon.xconf`, for instance `newsletter/config/cocoon-xconf/usecases.xconf`:

```
<xconf xpath="/cocoon/usecases"
       unless="/cocoon/usecases/component-instance[@name = 'newsletter.send']">
  <component-instance name="newsletter.send"
                      logger="lenya.usecases.newsletter.send"
                      class="org.myproject.lenya.newsletter.usecases.Send">
    <view template="modules/newsletter/usecases/send.jx" menu="false"/>
  </component-instance>
</xconf>
```

As you can see in the view declaration, the JX templates are typically stored in the `<module>/usecases` directory. The Java source files go into the `<module>/java/src` directory, they are compiled automatically by the Lenya build process.

4.2. Calling the Module Sitemap

The following URL syntax is used to make a call to a module sitemap:

```
cocoon://modules/<module>/**
```

The module sitemap is located at `newsletter/sitemap.xmap`. To request the latest newsletter and display it on the confirmation screen using the CInclude approach (for more information, see documentation about the usecase framework), you could for instance use the URI `cocoon://modules/newsletter/latestNewsletter.xml`:

```
<page:page
  xmlns:jx="http://apache.org/cocoon/templates/jx/1.0"
  xmlns:page="http://apache.org/cocoon/lenya/cms-page/1.0"
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:i18n="http://apache.org/cocoon/i18n/2.1"
  xmlns:cinclude="http://apache.org/cocoon/include/1.0"
  >

  <page:body>
    <h1><i18n:text>Send Newsletter</i18n:text></h1>
    <form>
      ...
      <jx:import uri="templates/messages.jx"/>

      <cinclude:includexml>
<cinclude:src>cocoon://modules/newsletter/latestNewsletter.xml</cinclude:src>
      </cinclude:includexml>

      <input name="submit" type="submit" value="Send Newsletter"/>
    </form>
  </page:body>
</page:page>
```

Another useful option is to use a module URL as the usecase view, e.g. to export some XML. Note that the attribute `uri` is used instead of `template`. The usecase class `DummyUsecase` can be used because no Java code shall be executed.

```
<xconf xpath="/cocoon/usecases"
       unless="/cocoon/usecases/component-instance[@name = 'newsletter.download']">
  <component-instance name="newsletter.send"
                      logger="lenya.usecases.newsletter.send"
                      class="org.apache.lenya.cms.usecase.DummyUsecase">
    <view uri="cocoon://modules/newsletter/downloadNewsletter.xml"/>
  </component-instance>
</xconf>
```

4.3. Adding Menu Items

A module can provide menu items which are added to the publication menu. To insert the menu items of a module, you have to add the module declaration to `publication.xconf`:

```
<publication>
  ...
  <module name="newsletter"/>
  ...
</publication>
```

If there is a `menus.xmap` sitemap in the module's root directory, a request of the form `<area>.xml` is sent into this sitemap. The matching pipeline could look like this:

```
<map:match pattern="**">
  <map:generate type="serverpages" src="config/menu.xsp"/>
  <map:serialize type="xml"/>
</map:match>
```

The server page `<module>/config/menu.xsp` delivers a menu XML which includes the items to be inserted:

```
<xsp:page ...>
  <menu>
    <menus>
      <menu i18n:attr="name" name="File">
        <block admin="false">
          <item uc:usecase="newsletter.send" href="?">
            <i18n:text>Send Newsletter</i18n:text>
          </item>
        </block>
      </menu>
    </menus>
  </menu>
</xsp:page>
```