

第三十四章 XDoclet生成源代码或其他文件辅助工具

XDoclet是一个很优秀的编程辅助工具，使用它可以大大减少开发Java程序所需要书写配置文件的时间。在本章中会介绍如何使用XDoclet，尤其如何在Web开发中使用XDoclet。

34.1 XDoclet入门

XDoclet是一个通用的代码生成实用程序，是一个扩展的Javadoc Doclet引擎（现已与Javadoc Doclet独立），XDoclet是EJBDoclet的后继者，而EJBDoclet是由Rickard Oberg发起的（<http://xdoclet.sourceforge.net/xdoclet/index.html>）。

它允许您使用象Javadoc标记之类的东西来向诸如类、方法和字段之类的语言特征添加元数据。随后，它利用这些额外的元数据来生成诸如部署描述符和源代码之类的相关文件。可以让你创建自己的javadoc @tags进而利用XDoclet中的Templet enging基于这些@tags生成源代码或其他文件（例如xml的deployment descriptors）。

XDoclet继承了Javadoc引擎的思想，允许根据定制Javadoc标记生成代码和其他文件。当然，XDoclet也可以访问整个解析树。这样，它就可以访问类、类的包结构和类的方法。

34.2 安装配置XDoclet

从XDoclet的主页（<http://xdoclet.sourceforge.net/xdoclet/index.html>）下载XDoclet1.2（xdoclet-bin-1.2.3.zip），解压缩到某个目录，并把这个目录称为XDOCLET_HOME，在使用XDoclet时一般要结合ANT使用，所以这里不需要额外太多的设置。

34.3 XDoclet的简单例子

使用XDoclet开发步骤包括：

- （1）编写Java文件。
- （2）添加注释。
- （3）编写build.xml（用于ant）。

34.3.1 Java源程序和添加注释

下面编写一个简单的Servlet，实现在页面显示“Hello：....”字符串的功能，Hello后的内容可以根据配置文件的内容进行改变。除了象开发其他Java程序要写的代码和注释外，还需要添加XDoclet需要的额外的注释，XDoclet就是根据这些注释（以@tags的格式出现）来生成配置文件的。下面是Java源程序。

```
package cn.ac.ict;

import javax.servlet.*;
import javax.servlet.http.*;
/**
 * @web.servlet name="HelloXDoclet"
```

```

*           display-name="Hello XDoclet"
*           load-on-startup="1"
* @web.servlet-init-param name="xdoclet"
*           value="{servlet.xdoclet}"
* @web.servlet-mapping url-pattern="/Hello/*"
*/
public class HelloXDoclet extends HttpServlet {
    public void init(ServletConfig config) throws ServletException {
        //从web.xml中获得初始化参数
        super.init(config);
    }
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException,
        java.io.IOException {
        ServletConfig config = this.getServletConfig();
        String h = config.getInitParameter("xdoclet");
        try {
            //首先设置文档类型
            response.setContentType("text/html; charset=GBK");
            //获取输出流
            java.io.PrintWriter out = response.getWriter();
            out.println("<html><head><title>A Simple
XDoclet!</title></head>");
            out.println("<body><h1>");
            out.println(" Hello:  " + h);
            out.println("</h1></body></html>");
            out.close();
        } catch (Exception e) {
            throw new ServletException(e);
        }
    }
}

```

34.3.2 书写build.xml文件

下面是编译和部署这个简单应用所需要的build.xml文件：

```

<?xml version="1.0" encoding="GBK"?>
<project name="filtering" default="deploy" basedir=".">
    <description>一个简单的XDoclet实例</description>

    <!-- 载入属性文件 -->
    <property file="build.properties"/>

    <!-- 定义类路径 -->
    <path id="web.classpath">
        <pathelement location="{tomcat.home}/common/lib/servlet-api.jar"/>
        <pathelement location="{tomcat.home}/common/lib/jsp-api.jar"/>
    </path>
    <path id="xdoclet.classpath">
        <fileset dir="{xdoclet.home}/lib">

```

```

    <include name="*.jar" />
  </fileset>
  <path refid="web.classpath" />
</path>

<!-- 初始化, 建立目录 -->
<target name="init">
  <mkdir dir="${dist.dir}" />
  <mkdir dir="${dist.dir}/WEB-INF" />
  <mkdir dir="${dist.dir}/WEB-INF/classes" />
</target>

<!-- XDoclet 的 WebDoclet 任务 -->
<target name="webdoclet" depends="init">
  <taskdef
    name="webdoclet"
    classpathref="xdoclet.classpath"
    classname="xdoclet.modules.web.WebDocletTask" />

  <webdoclet destDir="${dist.dir}/WEB-INF" force="${xdoclet.force}">
    <deploymentdescriptor Servletspec="2.4" xmlencoding="GBK" />
    <fileset dir="${src.dir}" includes="**/*.java" />
  </webdoclet>
</target>

<!-- 编译与部署 -->
<target name="deploy" depends="webdoclet">
  <javac srcdir="${src.dir}" destdir="${dist.dir}/WEB-INF/classes">
    <classpath refid="web.classpath" />
  </javac>
  <jar destfile="${tomcat.home}/webapps/${app.name}.war"
basedir="${dist.dir}" />
</target>
</project>

```

其中最主要的就是使用了XDoclet的WebDoclet任务，它是XDoclet预定义好的，在使用时需要使用<taskdef>元素引入这个任务，它的实现类是xdoclet.modules.web.WebDocletTask。在上面的build文件中包含了一个属性配置文件build.properties，它的内容要根据Tomcat和XDoclet安装的情况进行修改，大致内容如下。

```

##### 环境设置 #####
# Tomcat安装主目录
tomcat.home=C:/Tomcat 5.0
# xdoclet安装目录
xdoclet.home=C:/xdoclet-1.2.3
# web的临时目录
dist.dir=./dist
# 源文件目录
src.dir=./src
# 发布的程序名
app.name=FirstXDoclet
# Servlet参数, 可以改变

```

```
servlet.xdoclet=XDoclet
```

34.3.3 用XDoclet实现

把所有的文件都准备好后，这个文件夹的布局如图34.1。在servlet目录下运行ant命令，运行效果如图34.2。



图34.1 Xdoclet应用程序结构

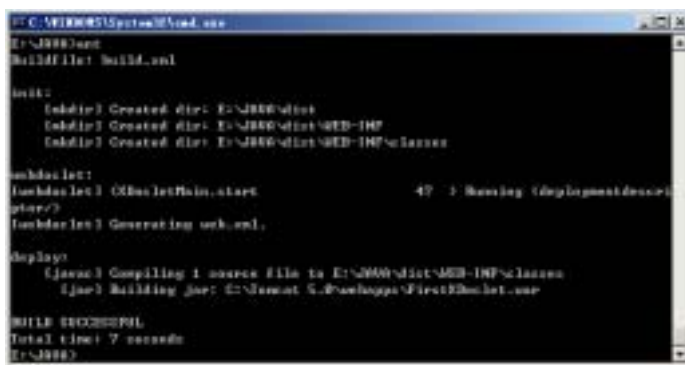


图34.2 运行ant

然后启动Tomcat，在浏览器地址栏中输入：<http://localhost:8080/FirstXDoclet/Hello>。可以看到如下画面，如图34.3。

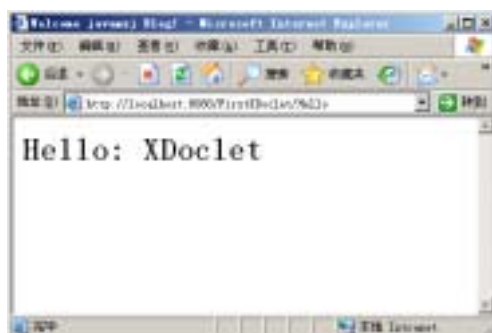


图34.3 XDoclet简单例子效果

34.4 XDoclet生成配置文件过程介绍

XDoclet是一个扩展的Javadoc Doclet引擎，而且通过上面的例子，可以很明确的看出来，由于开发者在源程序中书写了某些注释文字，XDoclet就像Javadoc工具那样把这些注释信息提取出来，形成了配置文件，例如上面的例子所生成的配置文件web.xml的内容如下：

```
<?xml version="1.0" encoding="GBK"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
  <distributable/>
  <servlet>
    <display-name>Hello Servlet</display-name>
```

```

        <servlet-name>HelloServlet</servlet-name>
        <servlet-class>cn.ac.ict.HelloServlet</servlet-class>

        <init-param>
            <param-name>hello</param-name>
            <param-value>XDoclet</param-value>
        </init-param>

        <load-on-startup>1</load-on-startup>

    </servlet>

    <servlet-mapping>
        <servlet-name>HelloServlet</servlet-name>
        <url-pattern>/Hello/*</url-pattern>
    </servlet-mapping>

</web-app>

```

下面根据程序中的注释对上面生成的配置信息解释一下，通过这些来理解XDoclet的工作过程。

在build.xml文件中有如下语句：

```
<deploymentdescriptor Servletspec="2.4" xmlencoding="GBK"/>
```

它指定使用了Servlet 2.4版本，xml文件的编码方式是GBK，因此XDoclet根据这些信息在web.xml文件中生成了如下信息：

```
<?xml version="1.0" encoding="GBK"?>
```

```

    <web-app xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">

```

其中的第一行是由xml的编码方式决定的。而Servletspec="2.4"这句则决定了<web-app>属性的最后一行。

下面来看看Servlet文件中的注释都起了什么作用，在Servlet文件中有如下注释：

```

/**
 * @web.servlet name="HelloServlet"
 *             display-name="Hello Servlet"
 *             load-on-startup="1"
 * @web.servlet-init-param name="hello"
 *             value="{servlet.xdoclet}"
 * @web.servlet-mapping url-pattern="/Hello/*"
 */

```

首先@web.servlet name标签指定了这个Servlet在web.xml中配置时使用的名字（<servlet-name>元素的值），display-name用于生成<display-name>元素，load-on-startup用于生成<load-on-startup>元素，也就是说通过第一个标签生成了如下的配置信息：

```

    <display-name>Hello Servlet</display-name>
    <servlet-name>HelloServlet</servlet-name>
    <load-on-startup>1</load-on-startup>

```

然后是@web.servlet-init-param标签，它指定了Servlet的初始化参数，用于生成<init-param>元素，其中name属性生成子元素<param-name>、value 属性生成子元素

<param-value>。也就是说，这个标签最终生成的配置信息是：

```
<init-param>
  <param-name>hello</param-name>
  <param-value>XDoclet</param-value>
</init-param>
```

最后是@web.servlet-mapping标签，它指定了这个Servlet的映射信息，url-pattern指定了映射到这个Servlet的URL，这个标签生成了如下的配置信息：

```
<servlet-mapping>
  <servlet-name>HelloServlet</servlet-name>
  <url-pattern>/Hello/*</url-pattern>
</servlet-mapping>
```

其中<servlet-name>元素由@web.servlet name标签指定。

这些就是配置文件的形成来源，XDoclet能够生成配置文件就是根据这些注释来实现的，如果注释写的不正确或者不规范就无法生成想要的信息。

34.5 使用XDoclet进行Web开发

使用XDoclet可以为开发EJB、Servlet、Filter等提供很大的方便，使用XDoclet进行Web开发更是可以省去很多工作，而且根据上面的介绍，读者也可以了解到使用XDoclet关键要用合适的标签写好注释，下面就结合XDoclet为Web开发提供的标签介绍如何使用XDoclet加速Web开发的进程。

34.5.1 开发Struts

@struts标签用于为开发Struts提供支持，其中它类级别的标签如表34.1。

表34.1 @struts的类级别的标签

标签	描述
@struts.action	用于定义action类和它的属性
@struts.action-exception	定义action类的异常处理器
@struts.action-forward	为action类定义局部转发
@struts.action-set-property	为action类创建set-property
@struts.dynaform	定义一个动态的Form Bean和它的属性
@struts.form	定义一个Form Bean和它的属性

(1) @struts.action用于定义action类和它的属性，它的参数及其描述如表34.2。

表34.2 @struts.action标签的参数

参数	类型	描述	是否必须
name	text	Action的名字，在这个Struts应用中是唯一的	是
type	text	为这个action实例化的类，默认就是当前类	否
className	text	用于为这个action提供服务的ActionMapping的子类的全名	否
path	text	这个action符号的路径	是
scope	text	定义action的范围，为request、session、application的一种	是
input	text	为这个action提供输入的路径	是
roles	text	允许访问这个ActionMapping对象的安全角色名，用逗号分隔	否
validate	text	这个action的验证标志，默认为true	是
parameter	text	这个action的可选参数	是

例如下面的一段注释

```
@struts.action
```

```
path="/struts/foo"
```

将会生成如下的配置信息：

```
<action
  path="/struts/foo"
  type="test.web.StrutsAction"
  unknown="false"
  validate="true"
>
```

(2) @struts.action-exception标签定义action类的异常处理器，它的参数及其描述如表34.3。

表34.3 @struts.action-exception的参数及其描述

参数	类型	描述	是否必须
key	text	这个异常处理器用来寻找错误信息的关键词	是
type	text	向处理器注册的异常的完整类名	是
className	text	这个异常类使用的Bean类名	否
handler	text	异常类的完整类名	否
path	text	异常发生时，用于完成本次请求的资源的路径	否
scope	text	用于访问错误对象的上下文	否

(3) @struts.action-forward标签为action类定义局部转发，它的参数及其描述如表34.4。

表34.4 @struts.action-forward标签的参数

参数	类型	描述	是否必须
name	text	转发的名字	是
path	text	转发的路径	是
redirect	bool	是否重定向到其他资源	否

例如如下的注释：

```
/* @struts.action-forward
 * name="success"
 * path="/struts/getAll.do"
 * redirect="false"
 */
```

将会生成如下的配置信息：

```
<forward
  name="success"
  path="/struts/getAll.do"
  redirect="false"
/>
```

(4) @struts.form标签定义一个Form Bean和它的属性，它的参数及其描述如表34.5。

表34.5 @struts.form标签的参数

参数	类型	描述	是否必须
name	text	为这个FORM定义一个唯一的名字	是
extends	text	定义产生的FORM必须扩展的类	否
implements	text	定义产生的FORM必须实现的接口	否
include-pk	bool	是否在form中包含pk值，默认为true	否
include-all	bool	是否包含所有的持久值	否

例如下面是一个Struts的Action组件StrutsDispatchAction.java生成配置信息的例子，它的源代码：

```
package test.web;
```

```

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.actions.DispatchAction;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

/**
 * Simple class to test Jakarta Struts generation (Jakarta Struts 1.2 beta
2 only).
 *
 * @struts.action
 *   path="/struts/bar"
 *
 * @struts.action-forward
 *   name="success"
 *   path="/struts/getAll.do"
 *   redirect="false"
 */
public final class StrutsDispatchAction extends DispatchAction
{
    public ActionForward execute(ActionMapping mapping, ActionForm form,
HttpServletResponse response)
    {
        return mapping.findForward("success");
    }
}

```

这个文件被XDoclet解析后会为这个Action生成如下的配置信息：

```

<action
  path="/struts/bar"
  type="test.web.StrutsDispatchAction"
  unknown="false"
  validate="true"
>
  <forward
    name="success"
    path="/struts/getAll.do"
    redirect="false"
  />
</action>

```

注意，如何使用上面的例子在最后一小节介绍。

34.5.2 Servlet过滤器

@web.filter标签用于定义Servlet过滤器，它只能对Servlet加注释。它的参数及其描述如

表34.6。

表34.6 @web.filter标签的参数及其描述

参数	类型	描述	是否必须
name	text	过滤器的名字,在当前Web应用中唯一的	是
display-name	text	过滤器的显示名称	否
icon	text	过滤器的图标	否
description	text	描述信息	否

定义一个完成的过滤器还需要其他几个标签的配合,分别介绍如下:

@web.filter-init-param用于为过滤器指定初始化参数,它的参数及其描述如表34.7。

表34.7 @web.filter-init-param的参数及其描述

参数	类型	描述	是否必须
name	text	参数的名字	是
value	text	参数的值	否
description	text	描述信息	否

@web.filter-mapping标签用于为过滤器定义映射信息,它的参数及其描述如表34.8。

表34.8 @web.filter-mapping标签的参数及其描述

参数	类型	描述	是否必须
url-pattern	text	过滤器符合的URL	否
servlet-name	text	过滤器Servlet的名字	否
dispatcher	text	与这个过滤器有关的请求转发器	否

例如下面的一个过滤器的例子,它用到了关于过滤器定义的标签,下面是TimerFilter.java的源代码:

```
package test.web;

import javax.servlet.*;
import javax.servlet.http.HttpServletRequest;
import java.io.IOException;
/* @web.filter
 *   display-name="Timer Filter"
 *   name="TimerFilter"
 *
 * @web.filter-init-param
 *   name="param1"
 *   value="value1"
 *
 * @web.filter-init-param
 *   name="param2"
 *   value="value2"
 *
 * @web.filter-mapping
 *   url-pattern="*.xml"
 */
public class TimerFilter implements Filter {

    private FilterConfig config = null;

    public void init(FilterConfig config) throws ServletException {
        this.config = config;
    }
}
```

```

    }

    public void destroy() {
        config = null;
    }

    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain) throws IOException,
ServletException {
        long before = System.currentTimeMillis();

        chain.doFilter(request, response);

        long after = System.currentTimeMillis();

        String name = "";
        if (request instanceof HttpServletRequest)
            name = ((HttpServletRequest) request).getRequestURI();

        config.getServletContext().log(name + ": " + (after - before) + "ms");
    }
}

```

这个文件被XDoclet解析后会为这个Filter生成如下的配置信息：

```

<filter>
  <filter-name>TimerFilter</filter-name>
  <display-name>Timer Filter</display-name>
  <filter-class>test.web.TimerFilter</filter-class>
  <init-param>
    <param-name>param1</param-name>
    <param-value>value1</param-value>
  </init-param>
  <init-param>
    <param-name>param2</param-name>
    <param-value>value2</param-value>
  </init-param>
</filter>

<filter-mapping>
  <filter-name>TimerFilter</filter-name>
  <url-pattern>*.xml</url-pattern>
</filter-mapping>

```

注意，如何使用上面的例子在最后一小节介绍。

34.5.3 自定义标签

@jsp.tag标签用于给自定义标签加注释，它的参数及其描述如表34.9。

表34.9 @jsp.tag标签的参数及其描述

参数	类型	描述	是否必须
name	text	JSP标签的名字	是

tei-class	text	JSP的tei类名	否
body-content	text	JSP标签体内容：tagdependent、JSP或empty，其默认值是JSP	否
display-name	text	标签的显示名称	否
small-icon	text	标签的小图标	否
large-icon	text	标签的大图标	否
description	text	标签的描述信息	否

34.5.4 运行演示例子

在34.5.1节和34.5.2节中介绍了两个例子，下面介绍如何演示这两个例子，首先要编写一个build.xml文件，其内容如下。

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<project name="XDoclet Examples" default="compile" basedir=".">
  <property file="build-dist.properties"/>

  <path id="samples.class.path">
    <fileset dir="${lib.dir}">
      <include name="*.jar"/>
    </fileset>
    <fileset dir="${samples.lib.dir}">
      <include name="*.jar"/>
    </fileset>
    <fileset dir="${dist.lib.dir}">
      <include name="*.jar"/>
    </fileset>
  </path>

  <target name="init">
    <tstamp>
      <format property="TODAY" pattern="d-MM-yy"/>
    </tstamp>

    <taskdef
      name="webdoclet"
      classname="xdoclet.modules.web.WebDocletTask"
      classpathref="samples.class.path"
    />
  </target>

  <target name="prepare" depends="init">
    <mkdir dir="${samples.classes.dir}"/>
    <mkdir dir="${samples.gen-src.dir}"/>
    <mkdir dir="${samples.meta-inf.dir}"/>
  </target>

  <target name="webdoclet" depends="prepare" description="Generate
deployment descriptors (run actionform to generate forms first)">
```

```

<echo>+-----+</echo>
      <echo>|                                     |</echo>
      <echo>| R U N N I N G   W E B D O C L E T           |</echo>
      <echo>|                                     |</echo>

<echo>+-----+</echo>

  <webdoclet
    destdir="${samples.gen-src.dir}"
    mergedir="parent-fake-to-debug"
    excludedtags="@version,@author,@todo"
    addedtags="@xdoclet-generated at ${TODAY},@copyright The XDoclet
Team,@author XDoclet,@version ${version}"
    force="${samples.xdoclet.force}"
    verbose="false"
  >

    <fileset dir="${samples.java.dir}">
      <include name="**/*Servlet.java"/>
      <include name="**/*Filter.java"/>
      <include name="**/*Tag.java"/>
      <include name="**/*Action.java"/>
      <include name="**/jsf/*.java"/>
      <!-- For Spring validation.xml -->
      <include name="**/Name.java"/>
      <include name="**/Address.java"/>
    </fileset>

    <!-- For ActionForms and generating validation.xml -->
    <fileset dir="${samples.gen-src.dir}"
includes="**/*Form.java"/>

    <deploymentdescriptor
      servletspec="2.3"
      destdir="${samples.web-inf.dir}"
    >
      <taglib
        uri="http://java.sun.com/jstl/ea/core"
        location="/WEB-INF/c.tld"
      />
      <contextparam
        name="foobar1"
        value="a test value"
        description="context parameter with a description"
      />
      <contextparam
        name="foobar2"
        value="another test value"
      />
    </deploymentdescriptor>

```

```

        <strutsconfigxml
            destdir="${samples.web-inf.dir}"
        />

        <strutsvalidationxml
            destdir="${samples.web-inf.dir}"
        />

    </webdoclet>
</target>

<target name="compile" depends="webdoclet">

<echo>+-----+</echo>
    <echo>|                                     |</echo>
    <echo>| C O M P I L I N G   S O U R C E S   |</echo>
    <echo>|                                     |</echo>

<echo>+-----+</echo>

    <javac
        destdir="${samples.classes.dir}"
        classpathref="samples.class.path"
        debug="on"
        deprecation="on"
        optimize="off"
    >

        <src path="${samples.java.dir}"/>
        <src path="${samples.gen-src.dir}"/>

    </javac>

</target>

<target name="clean">
    <delete dir="${samples.dist.dir}"/>
</target>

</project>

```

这个文件中的关键部分就是webdoclet目标。

在初始化目标中定义了名为webdoclet的任务，在webdoclet目标调用了这个任务，并使用了两个标签<strutsconfigxml>和<deploymentdescriptor>分别用于生成web.xml文件和struts-config.xml文件，还调用了<strutsvalidationxml>来生成Struts的验证信息。

在本代码开头包含了一个文件，这个文件中声明了很多需要用到的属性，其代码如下：

```

xdoclet.root.dir=E:/xdoclet-1.2.3
lib.dir = ${xdoclet.root.dir}/lib
dist.lib.dir = ${lib.dir}

```

```

samples.dir = .
samples.dist.dir = ${samples.dir}/target
samples.lib.dir = ${samples.dir}/lib
samples.src.dir = ${samples.dir}/src
samples.java.dir = ${samples.src.dir}/java
samples.gen-src.dir = ${samples.dist.dir}/gen-src

```

```

samples.meta-inf.dir = ${samples.dist.dir}/meta-inf
samples.web-inf.dir = ${samples.dist.dir}/web-inf
samples.merge.dir = ${samples.src.dir}/merge
samples.classes.dir = ${samples.dist.dir}/classes
samples.web.dir = ${samples.src.dir}/web
samples.xdoclet.force = false

```

读者在运行这个程序的时候需要把xdoclet的根路径修改一下。各个文件都改写完成后，其整个文件夹的结构如图34.4。

然后在samples目录下使用ant命令，可以看到效果如图34.5

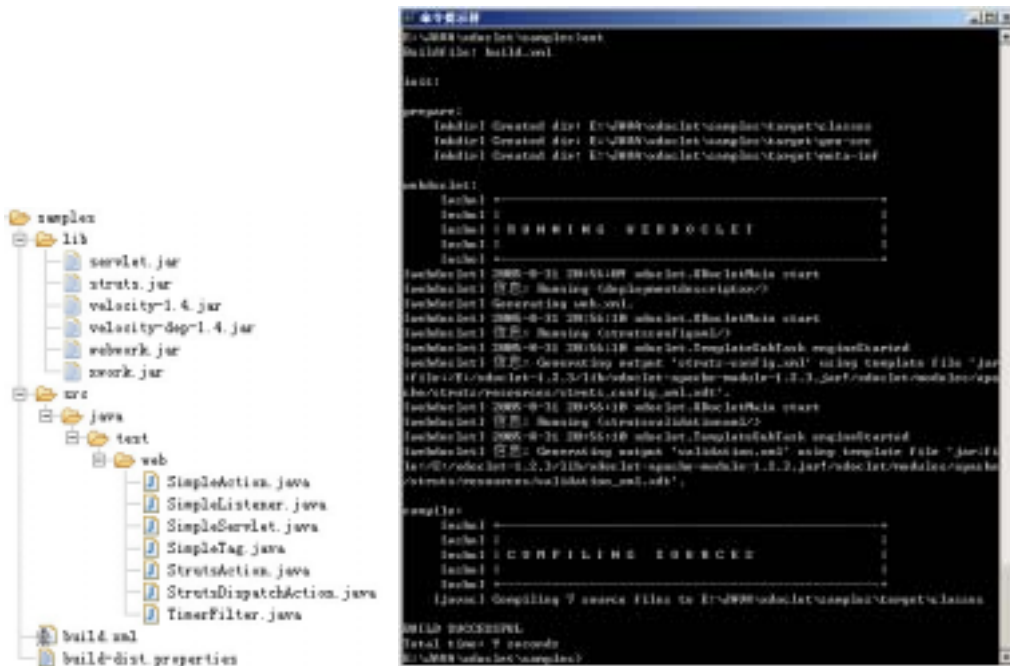


图34.4 代码结构



图34.5 ant运行效果

然后就可以在samples\target\web-inf目录下查看生成的文件。

34.6 小结

XDoclet是一个通用的代码生成实用程序，它对于简化很多Java应用程序的开发是很有帮助的。本章从一个简单的例子开始，介绍了XDoclet的使用，并介绍了XDoclet中用于Web开发使用几个标签，读者可以对照例子更好的理解XDoclet的使用。读者应该重点学习XDoclet中用于Web开发的标签。