Apache Struts und XDoclet

- Attributorientierte Programmierung unter Struts -

Manfred Wolff, wolff@manfred-wolff.de

In der Bewertung von Struts spielt immer wieder die Komplexität des Frameworks eine große Rolle. Große, nicht zur wartende Struts Konfigurationen, sind dabei sicher ein Teil des Problemes. Lösung kann hier die Aufspaltung der XML-Dateien in mehreren Dateien bieten oder die Benutzung von Struts Modulen. Auch das Wildcard-Mapping, welches mit Struts 1.2 eingefügt wurde, kann zur Linderung des Problems beitragen, löst es aber nicht wirklich.

In diesem Artikel wird die attributorientierte Programmierung mit XDoclet vorgestellt, welche helfen kann XML-Dateien zu generieren. Zugrunde liegt die Version 1.2.3 des XDoclet-Tools. Diese Version ist mit Struts 1.2.x kompatibel. Features, die erst mit Struts 1.3 eingeführt wurden, z.B. das Vererben von Struts Konfigurationen, werden von der vorliegenden XDoclet-Version nicht unterstützt.

Inhaltsverzeichnis

Einführung

Die `struts-config.xml` ist die zentrale Konfigurationsdatei des Struts-Frameworks. Hier wird der Workflow der Anwendung konfiguriert. Aus einem Pfadmapping wird eine bestimmte Konfiguration ermittelt und dann der nächste Schritt gefolgert:

```
1     <action
2       path="/login"
3       type="org.mwolff.application.action.LogonAction"
4       name="personenForm"
5       scope="request"
6       validate="false"
7       input="/pages/definitions/Login.jsp">
8         <forward
9           name="succeeded"
10          path="/pages/definitions/SuceededLogin.jsp"/>
11        <forward
12          name="failure"
13          path="/pages/definitions/FailureLogin.jsp"/>
14    </action>
```

Listing 1: Actionmapping unter Struts

Im Listing 1 wird ein solches typisches Mapping unter Struts dargestellt. Das Mapping beschreibt einerseits welche Action-Klasse den Request verarbeiten soll (Zeile 3) und zum Anderen wie es nach erfolgreichem oder fehlerhaften Programmfluss weiter gehen soll (Zeile 8 und 11).

Es erfordert nicht viel Fantasie um zu sehen, dass eine XML-Datei mit mehreren hundert solchen Einträgen schnell “ausufert” und vor allem nicht mehr wartbar ist. Es werden mehr und mehr “Leichen” in der Datei vorkommen, doppelte Definitionen etc. Die Lösungen die Struts für dieses Problem anbietet, schaffen nicht wirklich Abhilfe. So kann die Konfigurationsdatei in meherer einzelne Dateien zerlegt werden oder die Struts Module genutzt werden. Aber auch in diesen kleineren Einheiten wird man früher oder später auf die gleichen Probleme stoßen; das Problem ist sozusagen rekursiv.

XDoclet

XDoclet ist ein Framework, welches Doclets (vergleichbar mit den Annotations aus Java 5) aus dem Quelltext herausliest und in Handlungsanweisungen ummünzt. Der Quelltext wird mit den Doclets (Metatdaten) angereichert. Im Gegensatz zu Annotations werden Doclets nur in Kommentaren eingesetzt. Doclets kennen wir alle aus der JavaDoc. Hier werden in Kommentaren bestimmte Teile ausgezeichnet (`@param`), die dann später z.B. zu einer

HTML-Dokumentation führen. Doclets können auf verschiedenen Ebenen hinzugefügt werden: Auf Klassenebene und auf Methodenebene.

Ziel dieses Artikels ist es zu erreichen, dass möglichst viele XML-Dateien generiert und nicht mehr von Hand editiert und gewartet werden. Insbesondere geht es um die `struts-config.xml` und um die Validierungsregeln, die oft in der `validation.xml` zu finden sind. In beiden Fällen wird es Teile geben die dynamisch und contextabhängig generiert werden müssen und andere Teile, die statisch sind. XDoclet bietet die Möglichkeit statische- und dynamsiche Teile zusammenzuführen. Was genau generiert werden soll, wird in Templates festgelegt, die XDoclet für die verschiedenen Frameworks bereits mitliefert.

Die grundsätzlich Vorgehensweise ist in Abbildung 1 zu sehen:

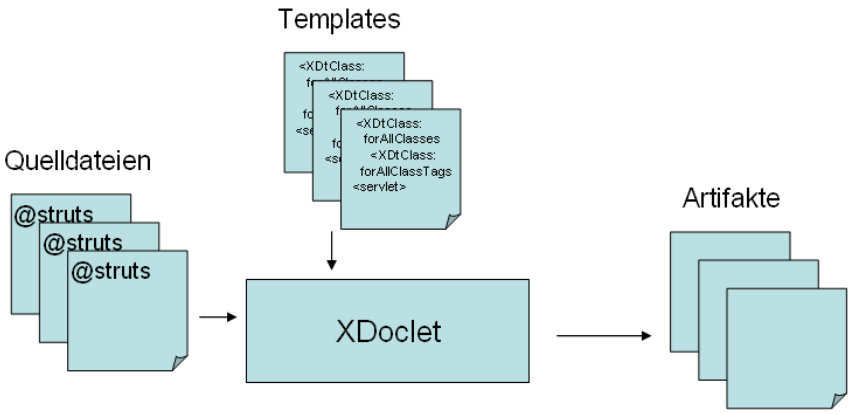

Abbildung 1: XDoclet - Schematische Darstellung

Struts und XDoclet

Will man die `struts-config.xml` in die Kontrolle von XDoclet übergeben so sind zunächst die statischen und dynamischen Teile der Konfiguration zu ermitteln:

- StatischeTeile werden in Templates hinterlegt. Jede Sektion der Konfiguration hat einen eigenen statischen Teil.
- Dynamische Teile werden mit Hilfe von Doclets in den entsprechenden Action-Klassen vereinbart. Zur Laufzeit werden dann die statischen und die dynamischen Teile der Struts Konfiguration zusammengefügt.

In der Tabelle 1 ist beschrieben, welche Dateien für die statischen Teile erwartet werden.

Dateiname	Bedeutung
`global-forwards.xml`	Datei, welche die globalen Forwards der Struts Konfiguration enthält
`struts-actions.xml`	Datei, welche die statischen Action-Definitionen der Struts Konfiguration enthält.
`struts-controller.xml`	Datei, in der der Controller configuriert wird.
`global-exceptions.xml`	Datei, in der die globalen Exception Handler definiert werden
`struts-forms.xml`	Datei, in der die statischen Forms definiert werden
`struts-message-resources.xml`	Datei, in der die Message-Ressources definiert werden
`struts-plugins.xml`	Datei, in der die Plugins definiert werden

Tabelle 1: Statische Definition der Struts Konfiguration

Beispielsweise könnte eine statische Sektion für globale Forwards wie in Listing 2 aussehen.

```
1  <global-forwards>
2    <forward
3        name="welcome"
4        path="/pages/Welcome.jsp"/>
5    <forward name="language"
6        path="/pages/Language.jsp"/>
7  </global-forwards>
```

Listing 2: Statische Vereinbarung von globalen Forwards

Dabei sollten die Form-Beans und die Action-Mappings im Quelltext durch XDoclets vereinbart werden, alle anderen Teile in globalen Dateien. Auch Wildcard-Mappings gehören in globalen, also statischen, Dateien.

Einsatz von Doclets

Im Folgenden wird anhand eines kleinen Beispiels (das klassische Login) der Einsatz von XDoclet beschrieben. Im Struts-Umfeld werden folgende Klassen benötigt:

- Die Bean, welche die Daten transportiert. In unserem Falle werden VO-Objekte (*value objects*) erzeugt, die dann auch in der Businessschicht genutzt werden können (`Login.java`).

- Die Actionklasse, die das Login ausführen soll. Die Klasse erbt von `DynaValidatorForm`, damit sie auch mit dynamischen Validatoren versehen werden kann.
- Die Struts Form-Bean wird aus dem `Login` VO generiert und bekommen den Namen `LoginForm`.

Außerdem wird der entsprechende Eintrag in der `struts-config.xml` generiert und die Validatoreinträge in der `validation.xml`.

Fallstricke

Zunächst etwas zur Ernüchterung: Der Test von XDoclet in meiner Umgebung hat ein paar Probleme gemacht, so dass ich das XDoclet Apachemodul patchen musste. Hier kurz die Änderungen, die ich machen musste, weil der versprochene Funktionsumfang nicht eingehalten wurde [1].

- Entgegen der Dokumentation kann in der Form-Bean nicht angegeben werden, von welcher Klasse die generierte Klasse erben soll. Es wird immer von `ActionForm` abgeleitet. Die Änderung in den neuen Templates sind so vorgenommen worden, dass grundsätzlich von `DynaActionForm` abgeleitet wird.
- Entgegen der Dokumentation wurden nur Action-Klassen in die Generierung aufgenommen, die von der Standard Struts-Action bzw. der Standard Struts-Formbean erben. Die Änderungen in den Templates sind so vorgenommen worden, dass alle Klassen durchsucht werden.

Generierung von Form-Beans

Um ein Form-Bean in die Struts Konfiguration einzubinden, ist ein `@struts-form` Tag im Klassenkommentar der betroffenen Klasse einzubinden. Die zu generierende Form bekommt dann automatisch das Suffix `Form`. Im Listing 3 ist die Vereinbarung zu sehen.

Das `@struts-form` - Tag hat einige weitere Parameter, die gesetzt werden können. Eine vollständige Auflistung aller Möglichkeiten gibt es hier: `http://xdoclet.sourceforge.net/xdoclet/tags/apache-tags.html`

- `implements` Hier kann ein Interface angegeben werden, welches die Action-Form realisieren soll.

[1] Die Änderungen können hier downgeloaded werden `http://manfred-wolff.de/data/xdoclet-apache-module-mwolff-1.2.3.jar`

```
1 package org.mwolff.action;
2 /**
3  * @struts.form
4  */
5 public class Login implements java.io.Serializable
6 {
7    //...
8 }
```

Listing 3: Vereinbarung einer Form-Bean in einer VO-Klasse

- `include-pk` Gibt an, ob ein pk-Feld in die Form inkludiert werden soll. Der Defaultwert ist `true`.
- `include-all` Gibt an, ob alle Felder des VO in die Form übernommen werden sollen. Der Defaultwert ist `false`.

Das XDoclet-Framework generiert aus dieser Anwendung den entsprechenden Form-Bean Eintrag in der Struts Konfiguration und die entsprechende Form-Bean Klasse. Genaugenommen wird dies in zwei Schritten vollzogen: Der XDoclet ejbdoclet-task generiert die Form, der webdoclet-task den Eintrag in der Konfiguration, dazu aber mehr im Kapitel *Konfiguration von XDoclet*.

```
1     <form-bean
2       name="loginForm"
3       type="org.mwolff.action.LoginForm"
4     />
```

Listing 4: Generierter Eintrag in der struts-config.xml

Grundsätzlich können mehrere Form-Beans aus einem VO generiert werden. Welche Elemente in welche Form-Bean übertragen werden sollen, kann dann auf Methodenebene konfiguriert werden.

```
1   /** @struts.form-field name=''loginForm''
2    */
3   public String getName() {
4   ..
5   }
```

Listing 5: Auszeichnung von Feldern in der Formbean

Generierung von Action-Mappings

Das Action-Mapping ist ähnlich einfach aufgebaut: Es muss sowohl das Mapping aufgebaut wird sowie alle Forwards und Exceptionhandler, die benötigt werden.

Im Listing 6 ist zunächst die Vereinbarung im Quelltext der entsprechenden Action zu sehen.

```
1  /**
2   * @struts.action name="loginForm"
3   *                path="/login"
4   *                unknown="true"
5   *
6   * @struts.action-forward name="hasSucceeded"
7   *                        path="/pages/login/Login.jsp"
8   *
9   */
10 public class LoginAction extends Action {
11 ...
12 }
```

Listing 6: Vereinbarung eines Action-Mappings

Auf die gleiche Art und Weise können Exceptionhandler (`@struts.action-exception`) und zusätzliche Konfigurationsparameter `@struts.action-set-property` vereinbart werden. Als Attribute für das `@struts.action` Tag sind alle Attribute erlaubt, die auch Struts verwendet also:

- `name` Gibt den Namen der Form-Bean an, die zu diesem Mapping korrespondiert.
- `type` Gibt den Typ der Action an. Dieser Parameter kann bei XDoclet weggelassen werden, weil die Klasse angenommen wird, in der das Mapping vereinbart wird.
- `path` Der Pfad für das Action-Mapping.
- `scope` Der Scope für das Action-Mapping. Default wird request angenommen.
- `input` Der Pfad in dem bei einem Validierungsfehler verzweigt werden soll.
- `roles` Eine kommaseparierte Liste von Rollennamen, die dieses Mapping ausführen dürfen.
- `validate` Gibt an, ob die validate()-Methode der Action-Form aufgerufen werden soll.

- `parameter` Die Optionalen Parameter für diese Action.

Generierung von Validatoren

Ein zweiter wichtiger Punkt bei der Generierung von Struts-Spezifischen Dingen ist die Erzeugung der `validation.xml`, die Validatorregeln für die einzelnen Forms beinhaltet. Die Validatorregeln werden auf Methodenebene vereinbart, wie das Beispiel in Listing 7 zeigt:

```
1     /**
2      * The Password of the account.
3      */
4     private String password;
5
6     /**
7      * @struts.form-field
8      * @struts.validator type="required"
9      *                   arg0resource="login.account"
10     * @return Returns the account.
11     */
12    public String getAccount() {
13        return this.account;
14    }
```

Listing 7: Vereinbarung eines Validatoreintrags

Einige Validatoren benötigen weitere Argumente, wie z.B. die Validierung eines Datum-Objekts. Auch hier ein Beispiel zur Verdeutlichung:

```
1     /**
2      * @struts.form-field
3      * @struts.validator type="date"
4      *                   arg0resource="kurs.beginn"
5      * @struts.validator-var name="datePatternStrict"
6      *                       value="dd.MM.yyyy"
7      * @return Returns the beginDate.
8      */
9     public String getBeginDate() {
10        return this.beginDate;
11    }
```

Listing 8: Komplexere Validatoren

Die weiteren gültigen Parameter sind der Originaldokumentation von XDoclet zu entnehmen, die was die Attribute angeht sehr gut und vollständig ist.

Konfiguration von XDoclet

Zum Schluss muss die Generierung der entsprechenden XML-Dateien noch angestoßen werden. Dies wird über zwei ant-tasks realisiert.

Im Folgenden wird das Listing 9 ausführlich erläutert.

- `Zeile 3-5`: Definition des `EJBDocletTask`, der mit XDoclet ausgeliefert wird.
- `Zeile 7-10`: Aus den VOs werden Struts Form-Beans generiert. Evtl. bereits bestehende Beans werden
- `Zeile 12-19`: Mit diesem Task werden aus den VO-Objekten die Struts Form-Bean Objekte erzeugt.
- `Zeile 24-25`: Definition des `WebDocletTask`, der mit XDoclet ausgeliefert wird.
- `Zeile 27-30`: Definition des Pfades, indem die `struts-confix.xml` abgelegt werden soll. Mit dem Paramter `mergeDir` wird festgelegt, wo die statischen Teile der Konfiguration zu finden sind.
- `Zeile 32-35`: Definition der Klassen, die in diesen Prozess eingebunden werden sollen. Dies sind zum Einen die Action-Klassen als auch die vom vorherigen Task generierten Struts Form-Beans.
- `Zeile 37-40`: Hier wird jetzt die Struts Konfiguration gebaut. Festzulegen ist der Name der zu generierenden Datei sowie einmal mehr das Verzeichnis, in dem die statischen Teile zu finden sind.
- `Zeile 42-45`: Das gleiche für die XML-Datei der Validatorenvereinbarungen, auch hier wäre es noch möglich statische Teile einzubinden.

Folgende Parameter werden in diesem Beispiel benötigt und müssen gesetzt werden:

Bezeichner	**Bedeutung**
`xdoclet.class.path`	Pfad zu den Bibliotheken für Ant.
`src`	Pfad zu den Sourcen (`/src/java`).
`webapp`	Pfad zur Webanwendung (`/src/webapp`).

Tabelle 2: Benötigte Variablen im Ant-Beispiel

Hier ein weiterer Ausschnitt aus der build.xml, in dem diese Pfade gesetzt werden.

```
1  <target name="ejbdoclet">
2
3     <taskdef name="ejbdoclet"
4              classpathref="xdoclet.class.path"
5              classname="xdoclet.modules.ejb.EjbDocletTask"/>
6
7     <delete>
8        <fileset dir="${src}/org/mwolff/example/vo/"
9                 includes="**/*Form.java"/>
10    </delete>
11
12    <ejbdoclet destdir="${src}"
13               excludedtags="@version,@author"
14               ejbspec="2.0">
15      <fileset dir="${src}">
16         <include name="org/mwolff/example/vo/*.java" />
17      </fileset>
18      <strutsform />
19    </ejbdoclet>
20 </target>
21
22 <target name="webdoclet">
23
24    <taskdef name="webdoclet" classpathref="xdoclet.class.path"
25             classname="xdoclet.modules.web.WebDocletTask"/>
26
27    <webdoclet destdir="${webapp}/WEB-INF"
28               force="true"
29               verbose="true"
30               mergedir="./src/struts-meta-example">
31
32    <fileset dir="${src}">
33       <include name="org/mwolff/example/action/*.java" />
34       <include name="org/mwolff/example/vo/*Form.java" />
35    </fileset>
36
37    <strutsconfigxml validatexml="true"
38                     version="1.1"
39                     destinationFile="struts-config.xml"
40                     mergedir="./src/struts-meta-example"/>
41
42       <strutsvalidationxml
43             destDir="${webapp}/WEB-INF"
44             destinationFile="struts-examples.xml"
45             validatexml="false"/>
46
47    </webdoclet>
48 </target>
```

Listing 9: Ant Tasks zur Erzeugung der XML-Dateien

```
1 <property name="src" value="./src/java"/>
2 <property name="webapp" value="./src/webapp"/>
3
4 <path id="xdoclet.class.path">
5    <fileset dir="./lib">
6       <include name="**/*.jar"/>
7    </fileset>
8 </path>
```

Listing 10: Vereinbarung von Variablen

Fazit

XDoclet ist eine sehr einfache Möglichkeit XML-Deploymentdescriptoren zu erzeugen. Dieses ist nicht nur für Struts-Anwendungen interessant. Es können auch EJB-, Hibernate-, Spring- und über 20 weitere Descriptoren erzeugt werden. Sicherlich ist dieses nicht der Königsweg - dieses ist MDA.

Ein lauffähiges Beispiel (als struts-blank) für die Struts 1.2 Version ist hier zu bekommen: `http://manfred-wolff.de/data/struts-template-1.2.zip` außerdem eine Webanwendung, die direkt im Servlet-Conteiner deployt werden kann `http://manfred-wolff.de/data/struts-template-1.2.war`.

Listings