Cyclos - Eine Architekturanalyse

1 Grundlegende Struktur

Cyclos ist eine auf dem `Struts-Framework` basierende Webapplikation. Struts selbst folgt extensiv dem MVC-Paradigma (genauer: der Model2-Architektur) und somit ist auch die Cyclos-Architektur an dieses Paradigma gebunden.

2 Konfiguration

Die Datei *web.xml* im Verzeichnis */WEB-INF* des web-Projekts ist die zentrale Konfigurationsdatei des `Struts-Frameworks`. Sie enthält alle Links zu den entsprechenden struts-config-Dateien, grundlegende Konfigurationen, Links zu den verwendeten Servlets und Spring-Dateien, sowie den Pfad der Willkommen-Seite (Startseite) von Cyclos.

3 Requestabarbeitung

Requests werden von der Struts Controller-Komponente empfangen, im Fall Cyclos ist diese das vorgeschaltet Action-Servlet. Nun identifiziert Struts das Action-Mapping, welches für die Abarbeitung des Requests zuständig ist.
Cyclos verteilt das Action-Mapping auf mehrere kleinere Dateien, die alle den Namen struts-config_*name*.xml tragen und sich im Ordner *WEB-INF/struts-config/* des web-Projekts befinden. Der Platzhalter *name* beschreibt hierbei eine aus Entwicklersicht logische Zusammenfassung von Actions, die für eine bestimmte Funktionalität innerhalb der Cyclos-Plattform zuständig sind. Diese Trennung ist jedoch von Struts nicht vorgeschrieben, sie dient nur der besseren Übersicht.

Im Folgenden betrachten wir beispielhaft den Workflow, der zum Editieren einer Währung abgearbeitet wird:

- der Controller (das Action-Servlet) erhält vom Client den GET-Request do/admin/editCurrency
- die web.xml Konfigurationsdatei wird nach den Pfaden der einzelnen struts-configs durchsucht
- die struts-configs werden nach der entsprechenden Action durchsucht
- die zuständige Action wird ausgeführt

Jede Action wird innerhalb des Mappings in einem separaten action-Tag angelegt, dessen - teilweise optionalen - Parameter durch die Struts-API vorgeschrieben sind.

```
<action
    path="/admin/editCurrency"
    type="nl.strohalm.cyclos.controls.accounts.currencies.EditCurrencyAction"
    name="editCurrencyForm"
    scope="request"
    input="admin/_editCurrency">
    <set-property property="secure" value="true" />
    <forward name="success" path="/do/admin/editCurrency" redirect="true"></forward>
</action>
```

Das obige Beispiel veranschaulicht die Definition der Action editCurrency, welche in der Datei struts-config_currencies.xml eingetragen ist. Sie enthält als Parameter alle wichtigen Informationen für die Abarbeitung des Requests:

- path - aufrufender Request (hierüber wird die Action dem Request zugeordnet)
- type - Name und Pfad der zur Bearbeitung benötigten Java-Klasse
- name (optional) - Name des Form-Beans, falls mit dieser Aktion einer verknüpft ist
- scope (optional) - identifiziert, in welchem Bereich (Scope) auf den Form-Bean zugegriffen werden soll
- input - URI des Forms, der aufgerufen wird, falls es während der Abarbeitung zu einem Validierungsfehler kommt

Alle verarbeitenden Klassen im Cyclos liegen im controler-Verzeichnis. Jede dieser Controler-Klassen muss primär von *BaseAction* abgeleitet sein. *BaseAction* implementiert die Methode *execute()*, die von Struts automatisch aufgerufen wird, um die *FormBeans* auszulesen und die entsprechenden Aktionen auszuführen. Der Funktion werden 4 Argumente übergeben.

- ActionMapping - das `<action/>`-Element, welches die auszuführende Aktionsklasse beschreibt
- ActionForm - das FormBean-Objekt, mit dem Formulardaten ausgelesen werden können
- HttpServletRequest - die zu verarbeitende HTTP-Anfrage
- HttpServletResponse - das Response-Objekt, welches an den Client zurückgegeben wird

```
@Override
public final ActionForward execute(final ActionMapping actionMapping,
                                   final ActionForm actionForm,
                                   final HttpServletRequest request,
                                   final HttpServletResponse response) throws Exception {

  ...
  // Create an action context
  final ActionContext context = new ActionContext(actionMapping, actionForm, request,
                                                  response, user, fetchService);
  request.setAttribute("formAction", actionMapping.getPath());
  ...
  // Process the action
  final ActionForward forward = executeAction(context);

  return forward;
  ...
}
```

Bevor die eigentliche Aktion bearbeitet wird, muss vorher noch geprüft werden, ob der Benutzer korrekt angemeldet ist und die Session wird entsprechend geladen.

Der *ActionContext* kapselt alle wichtigen Daten in einem Objekt. Desweiteren wird dem Request der Pfad des ActionMapping-Objektes angefügt, damit das ActionServlet später weiß, welche Aktion weiter abgearbeitet werden soll. Schließlich wird die Methode *executeAction()* aufgerufen. Diese erzeugt ein *ActionForward*-Objekt, welches an das aufgerufene *ActionServlet* übergeben wird.

Im Fall der Aktion „alle Währungen anzeigen“ würde diese Methode in der Klasse *ListCurrenciesAction* folgenden Aufbau haben:

```
@Override
protected ActionForward executeAction(ActionContext context) throws Exception {
    HttpServletRequest request = context.getRequest();
    List<Currency> currencies = currencyService.listAll();
    request.setAttribute("currencies", currencies);
    request.setAttribute("editable",
       getPermissionService().checkPermission("systemCurrencies", "manage"));
    return context.getInputForward();
}
```

In Cyclos ist es so geregelt, dass nun auf eine oder mehrere Service-Klassen zugegriffen wird. Im obigen Beispiel wird auf die Klasse *currencyService* zugegriffen, in der eine Methode liegt, die konkret zum Auflisten aller Währungen dient: *listAll()*.

```
public List<Currency> listAll() {
    if (cachedCurrencies == null) {
        cachedCurrencies = currencyDao.listAll();
    }
    return cachedCurrencies;
}
```

Die Service-Klassen können im Cyclos nicht direkt auf die Datenbank zugreifen. Ein Zugriff auf die Datenbank ist nur über ein DAO-Objekt (Data Access Object) möglich. Daher findet in der Service-Methode ein Aufruf der entsprechenden DAO-Klasse statt. Wie im nächsten Beispiel ersichtlich kann die entsprechende DAO-Klasse dann die entsprechende Datenbank-Anfrage durchführen und das Ergebnis zurückliefern.

```
public List<Currency> listAll() {
    return list("from Currency c order by c.name", null);
}
```

Die Konfiguration und Zusammenarbeit zwischen den Entity, DAO und Service-Klassen wird in Cyclos allerdings mittels Spring und auf Datenbankebene durch Heibernate realisiert.

Im Allgemeinen ruft der Controller die Modelschicht auf, in Cyclos sind das die Service-Klassen, um Daten zu verarbeiten und in die Datenbank zu schreiben bzw. aus der Datenbank zu lesen. Das Ergebnis wird an den Controller zurück gegeben. Im Falle der *listAll()*-Funktion werden alle Währungen aus der Datenbank dem Namen nach geordnet in der Liste *currencies* gespeichert. Sind diese Daten wichtig für direkt nachfolgende Aktionen bzw. für das Erstellen der Webseite, müssen die Daten dem Request hinzugefügt werden. Dies geschieht mit Hilfe der Methode *setAttribut()*. Im obigen Beispiel wird die Währungsliste und ein boolscher Wert dem Request angefügt, um sie später in der Java-Server-Page wieder auslesen zukönnen.

Schließlich wird ein *ActionForward*-Objekt erzeugt. Dieses Objekt ist eine *JavaBean* und wird vom Controller verwendet, um festzulegen, welche Seite als Nächstes angezeigt werden soll. *context.getInputForward()* teilt Struts mit, dass die JSP-Datei geladen werden soll, welche im *input*-Tag des ActionMapping-Objekts angegeben ist.

In Cyclos repräsentiert das input-Attribut allerdings nie direkt den Pfad einer JSP-Seite, sondern referenziert den Namen einer *Tiles*-Definition. Um diese Definition heraus zu finden, verfährt Struts nach dem selben Prinzip wie für die *actions*. In der *struts-config.xml* sind alle tiles-defs-Dateien aufgelistet. Alle tiles-defs_*name*.xml Dateien befinden sich im Verzeichnis *WEB-INF/tiles-defs/*. Diese werden der Reihe nach geparst, bis der Wert des *name*-Attributes der *definition* mit dem Wert des *input*-Attributes der *action* übereinstimmt. Für den input=“admin/_listCurrencies“befindet sich die zu suchende Definition in der tiles-defs_currencies.xml Datei.

```
<definition name="admin/_listCurrencies" extends=".adminLayout">
    <put name="body" value="/pages/accounts/currencies/listCurrencies.jsp"/>
</definition>
```

- name - gibt an, welcher Bereich der Seite neu geladen werden soll
- value - der Pfad der JSP-Datei. In Cyclos befinden sich alle dieser Dateien im pages-Ordner

Innerhalb dieser JSP-Dateien kann man nun auf die Attribute des Request zugreifen, die im Controller estellt wurden. Dies geschieht mittels ihrem *Attributnamen*. In der listCurrencies.jsp wird eine Übersicht der möglichen Zahlungsmittel in Form eine Tabelle erzeugt. Über eine Schleife werden iterativ alle Währungen durchlaufen und jeweils eine neue Zeile der Tabelle hinzugefügt. Der Wert *editable* gibt an, ob dem Benutzer die Möglichkeit zusteht, Währungen zu bearbeiten.

```
<c:forEach var="currency" items="${currencies}">
  ...
  <td align="left">${currency.name}</td>
  <td align="left">${currency.symbol}</td>
  ...
</c:forEach>
```

Schließlich wird diese generierte HTML-Seite als Response an den Client zurück geschickt.

3.1 Formulare

Am Beispiel für das Editieren von Währungen soll nun der Einsatz und die Wirkungsweise von Formularen erläutert werden. Die Rechte zum Bearbeiten von Währungen liegen beim Administrator. Der Aufbau der Menüliste wird durch die *menuAdmin.jsp*-Datei beschrieben, welche sich im Verzeichnis */web/pages/general/layout/* befindet.

```
<cyclos:menu key="menu.admin.accounts">
    <cyclos:menu url="/do/admin/listCurrencies" key="menu.admin.accounts.currencies" module=
        "systemCurrencies" operation="view" />
    ...
</cyclos:menu>
```

Der Ausschnitt beschreibt den Menüpunkt *Accounts*. Er besteht aus mehreren Untermenüs. Klickt man auf diesen Link, so wird ein Request mit der angegebenen URL an den Server geschickt und es wird die bereits beschriebene Aktion ausgeführt. Alle Module, die in Cyclos zur Verwendung kommen, müssen in der Datei *Permission.java* im Ordner */src/nl/strohalm/cyclos/setup* aufgelistet werden. Sie werden während des Setup statisch erzeugt. Das Attribut *key* referenziert die Überschrift des Menüs. Der resultierende Wert ist kontextabhängig von der eingestellten Sprache.

- Spracheinstellung - Die Sprache wird in der *cyclos.properties*-Datei eingestellt. Als Voreinstellung ist die englische Sprache angegeben „cyclos.embedded.locale = en_US“
- Wertzuweisung - für jede Sprache muss es eine Datei geben, in der jedem *key* einen Text zugewiesen wird. Diese Dateien befinden sich im Verzeichnis */web/WEB-INF/classes*

In der *ApplicationResources_en_US.properties* aus dem Verzeichnis */web/WEB-INF/classes* findet man folgenden Eintrag:

```
...
menu.admin.accounts.currencies=Manage Currencies
...
```

Als Ergebnis der Action */admin/listCurrencies* wird eine Übersicht aller Währungen erstellt. Ist man als Administrator angemeldet, so hat man zusätzlich die Möglichkeit, die einzelnen Zahlungsmittel zu bearbeiten bzw. zu löschen.

```
...
<cyclos:script src="/pages/accounts/currencies/listCurrencies.js" />
...
<c:choose><c:when test="${editable}">
   <img currencyId="${currency.id}" src="<c:url value="/pages/images/edit.gif" />" class="
        edit details" />
   <img currencyId="${currency.id}" src="<c:url value="/pages/images/delete.gif" />" class="
        remove" />
</c:when><c:otherwise>
   <img currencyId="${currency.id}" src="<c:url value="/pages/images/view.gif" />" class="
        view details" />
</c:otherwise></c:choose>
...
```

Zusätzlich zum generierten HTML-Code wird dem Response eine JavaScript-Datei übergeben. Dieses Script definiert die Events, welche beim Klicken auf die Bilder ausgelöst werden sollen.

```
...
'img.details': function(img) {
   setPointer(img);
   img.onclick = function() {
      self.location = pathPrefix + "/editCurrency?currencyId=" + img.getAttribute("
           currencyId");
   }
},
...
```

Die JS-Funktion *self.location* zwingt dem Browser zum Neuladen der angegeben Seite, in diesem Falle *do/editCurrency?currencyId=ID*. Es wird ein Request an den Server geschickt und das Servlet bearbeitet folgende Action:

```
<form-beans>
   <form-bean name="editCurrencyForm" type="nl.strohalm.cyclos.controls.accounts.currencies.
        EditCurrencyForm" />
</form-beans>

<action
   path="/admin/editCurrency"
   type="nl.strohalm.cyclos.controls.accounts.currencies.EditCurrencyAction"
   name="editCurrencyForm"
   scope="request"
   input="admin/_editCurrency">
   <set-property property="secure" value="true" />
   <forward name="success" path="/do/admin/editCurrency" redirect="true"></forward>
</action>
```

Das Attribut *name* bestimmt die Form-Bean, die als *FormAction*-Objekt der execute-Methode der *EditCurrencyAction.jsp* übergeben wird.

Alle Controller-Klassen, die Form-Beans verwenden, werden nicht direkt von *BaseAction* abgeleiten, sondern von der Klasse *BaseFormAction*. Diese Klasse überschreibt die Methode *executeAction()*.

```
...
@Override
protected final ActionForward executeAction(final ActionContext context) throws Exception {
   if (isFormPreparation(context)) {
      return handleDisplay(context);
   } else if (isFormValidation(context)) {
      return handleValidation(context);
   } else if (isFormSubmission(context)) {
      return handleSubmit(context);
   } else {
      return context.sendError("errors.invalid_request");
   }
}

protected ActionForward handleDisplay(final ActionContext context) throws Exception {
   prepareForm(context);
   return context.getActionMapping().getInputForward();
}
...
```

Diese Methode definiert eine Fallunterscheidung der Request-Anfrage.

- bei einer **GET**-Anfrage wird die Funktion *handleDisplay()* ausgeführt
- eine **POST**-Anfrage bewirkt den Aufruf der Funktion *handleSubmit()*

Die Funktion *self.location()* löst eine GET-Anfrage aus. Die Funktion *prepareForm()* wird durch die Klasse *EditCurrencyAction* implementiert. *handleSubmit()* wird durch derer überschrieben.

prepareForm() erstellt die Formular-Bean und lädt die Währung aus der Datenbank mittels der vom Browser übergebenen Variable *currencyId*. Falls die ID keinen Wert hat, wird eine neue Währung erstellt. Anschließend werden die Daten wieder dem Request-Objekt angefügt.

```
@Override
protected void prepareForm(final ActionContext context) throws Exception {
   final HttpServletRequest request = context.getRequest();
   final EditCurrencyForm form = context.getForm();
   final long id = form.getCurrencyId();
   final boolean isInsert = id <= 0L;
   ...
   Currency currency;
   if (isInsert) {
       currency = new Currency();
       editable = true;
   } else {
       currency = currencyService.load(id);
   }
   ...
   request.setAttribute("currency", currency);
   ...
}
```

Der Aufruf „return context.getActionMapping().getInputForward()“ bewirkt, dass die Kontrolle an das *ActionServlet* zurückgegeben wird und die unter *input* angegebene JSP-Datei wird geladen. Zusätzlich wird der Datei noch das *ActionMapping*-Objekt übergeben.

Die *editCurrency.jsp* generiert eine Eingabemaske, über die der Benutzer Eigenschaften der Währung bearbeiten kann. Von Bedeutung ist folgender Tag:

```
<ssl:form method="post" action="${formAction}">
```

Er teilt dem Browser mit, dass das Request an den Server nach dem *submit* per POST-Anfrage geschickt werden soll. Und die eben beschriebene Action */admin/editCurrency* wird erneut bearbeitet. Vor dem Abschicken des Formulars wird es clientseitig validiert.

Diesmal wird die Methode *handleSubmit()* ausgewertet.

```
@Override
protected ActionForward handleSubmit(final ActionContext context) throws Exception {
   final EditCurrencyForm form = context.getForm();
   Currency currency = getDataBinder().readFromString(form.getCurrency());
   ...
   currency = currencyService.save(currency);
   ...
   return ActionHelper.redirectWithParam(context.getRequest(), context.getSuccessForward(),
       "currencyId", currency.getId());
}
```

Die Funktion *getDataBinder().readFromString()* wandelt Plaintext in ein Datenobjekt um, welches von Java interpretiert werden kann. Anschließend wird das Currency-Objekt in der Datenbank abgespeichert. *ActionHelper.redirectWithParam()* bewirkt den Aufruf einer weiteren *Action*. Es wird innerhalb des *ActionMapping*-Objekts derjenige *forward*-Tag ausgewählt, dessen *name*-Attribut=“success“ ist. *path* gibt die nächste Aktion an, die aufgerufen werden soll, ohne vorher eine JSP-Datei auszuführen. In diesem Fall wird die selbe Aktion erneut aufgerufen als GET-Anfrage. Dabei werden dem Request das Attribut *currencyId* hinzugefügt. Am Ende wird entsprechend wieder die Formularmaske der eben bearbeiteten Währung angezeigt.

```
<forward name="success" path="/do/admin/editCurrency" redirect="true"></forward>
```

4 Zusammenfassung

- Cyclos folgt dem **MVC**-Prinzp

  das **Model** befindet sich im *services*-Ordner

  der **Controller** befindet sich im *controls*-Verzeichnis

  der **View** ist im Ordner *pages* im Webverzeichnis lokalisiert

- die **web.xml** ist die zentrale Konfigurationsdatei

  in ihr werden alle Konfigurationsdateien für Spring aufgelistet

  sie konfiguriert das **Action Servlet**

- Cylcos verwendet das **Tiles**-Plugin, um die Webseite in logische Bereiche zu untergliedern

  alle *tiles-defs*-Dateien befinden sich im *tiles-defs*-Odner im Webverzeichnis

  sie bestimmen, welche JSP-Dateien ausgeführt werden sollen

- die auszuführenden **Actions** werden in den *struts-config*-Dateien beschrieben

  sie übergeben den **Request** an den Controller

  sie rufen die *JSP*-Datei auf, welche als **Response** an den Client zurückgeschickt wird

- der **Controller** stellt die Verbindung zwischen dem *Action Servlet* und der *Modelschicht* dar

  nur der *Controller* kennt die aufgerufene *Action*

  alle *controls*-Klassen müssen direkt/indirekt von der Klasse *BaseAction* abgeleitet sein

  Struts ruft automatisch die Methode **execute()** auf

  sollen zusätzlich **Form-Beans** eingesetzt werden, muss die Klasse von **BaseFormAction** erben

  bei Formularen wird beim *Request* zwischen **GET** und **POST** unterschieden

  der *Controller* lässt die Daten vom *Model* verarbeiten und speichert sie im *Request*-Objekt ab

  soll die Kontrolle an das *Action Servlet* zurückgegeben werden, muss ein **ActionForward**-Objekt erzeugt werden

- in Abhängigkeit vom Rückgabewert der *Controllers* hat das *Servlet* zwei Möglichkeiten

  es wird ein **Response**-Objekt erstellt und an den Browser geschickt

  der **Request** wird zur weiteren Bearbeitung an eine andere **Action** geleiten