

Einführung
in
Spring
Web
Flow

von Carsten Mjartan

codecentric

Einführung in Spring Web Flow

Das Spring Framework ist als Integrationslösung in Webanwendungen sehr beliebt und kommt in vielen Projekten zum Einsatz, in denen ein leichtgewichtiges Framework und hohe Flexibilität gewünscht ist.

Zur Unterstützung von Webprojekten bietet Spring bereits im Standard ein robustes, auf dem Model-View-Controller (MVC Model 2 [1]) Pattern basierendes Framework an. Unterstützt wird im Wesentlichen das „freie Browsen“, bei dem die Reihenfolge der Seitenaufrufe durch den Anwender weitgehend dem Zufall überlassen ist. Die bestehenden Command- und FormController-Implementierungen decken zudem häufig vorkommende Anwendungsfälle ab, z. B. die Implementierung der Behandlung einzelner Formularseiten oder mehrseitiger Assistenten.

In der Praxis erweisen sich die existierenden Controller jedoch entweder als zu spezifisch oder zu granular, wenn es um die Realisierung komplexerer Page-Flows geht. Bei mehrschrittigen Aktionen geht die Übersicht über den Gesamtprozess bei der Implementierung schnell verloren. Spring WebFlow schließt die Lücke, indem es die allgemeine Beschreibung gesteuerter Abläufe (Web Flows) ermöglicht.

Was ist ein Web Flow

Ein Web Flow beschreibt eine funktional zusammengehörige Abfolge von Benutzerinteraktionen und hat damit oft einen direkten Bezug zu Use Cases aus der objektorientierten Analyse. Am Einfachsten lässt sich das anhand eines konkreten Beispiels erklären:

In einem Web-Shop sollen Bestellungen durchgeführt werden können. Nach der Auswahl der zu bestellenden Artikel steht der Gang zur Kasse an. Spätestens ab hier ist ein Login des Anwenders notwendig und falls nötig nachzuholen. Es kann aber auch sein, dass der Anwender als Neukunde noch keinen Login besitzt und sich zuerst registrieren muss. In beiden Fällen ist es wünschenswert, dass nach Abschluss des Logins bzw. der Registrierung die Bestellung nahtlos fortgeführt wird. Login und Registrierung sind jedoch nicht nur bei einer Bestellung durchführbar, sondern auch als separate Funktionen auf der Startseite (vgl. Abb. 1).

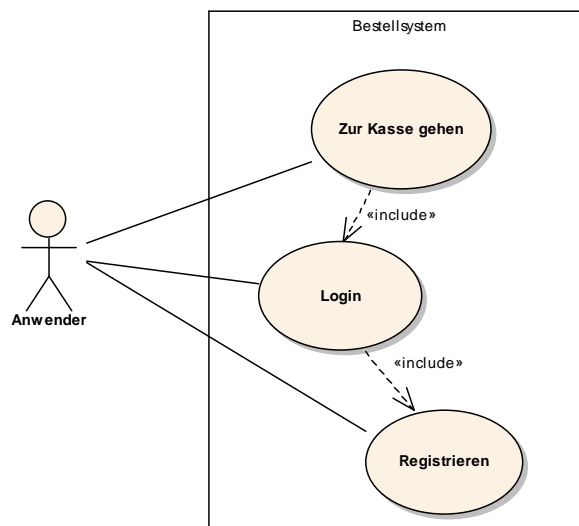
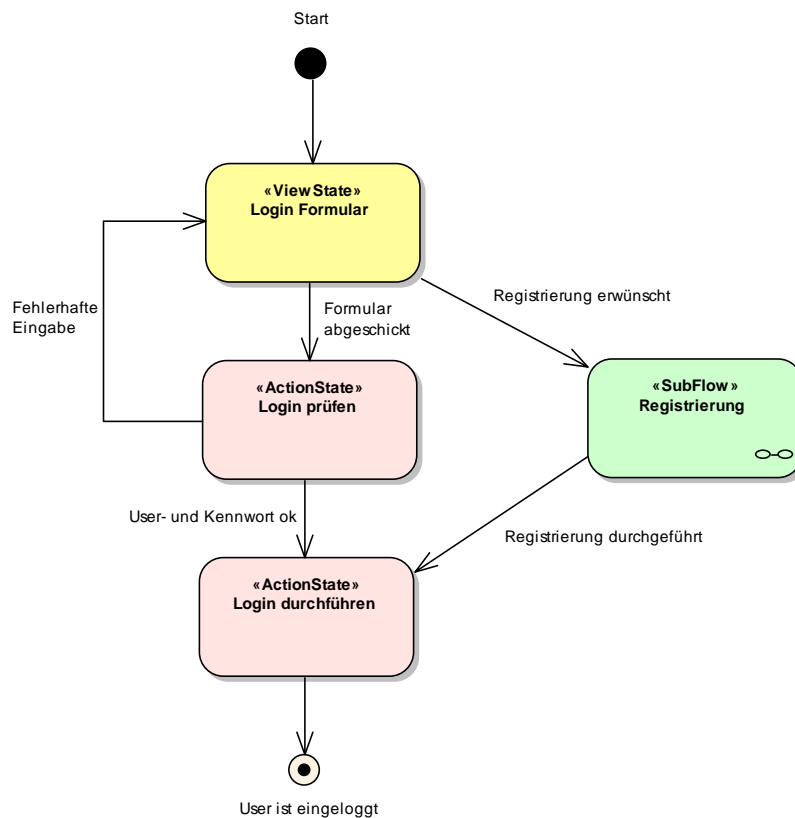


Abbildung 1: Web-Shop Beispiel - Use Cases

Die beschriebenen Use Cases lassen sich in diesem Fall 1:1 durch Spring Web Flows beschreiben, wobei insbesondere die oben beschriebene Wiederverwendung von Flows wie z. B. dem Login eine der Großen Stärken des Frameworks darstellt.

Der Web Flow für den Login selbst lässt sich wie folgt beschreiben:



Ein Web Flow besteht im Wesentlichen aus Status und Statusübergängen (Transitionen), die mögliche Abfolgen im Flow beschreiben. Die Status beschreiben dabei die Stellen im Flow, in denen Aktionen durchzuführen sind, sei es durch den Anwender oder durch Java-Code. Folgende Stustypen werden standardmäßig unterstützt:

- **View State**
Ein View State beschreibt eine vom Anwender durchzuführende Aktion. Üblicherweise wird ein Spring-konformer benannter View gerendert, es sind aber auch Redirects auf externe Seiten möglich. Fortgesetzt wird der Flow durch das Versenden eines Formulars oder durch dem Folgen eines Links auf der Seite, wobei über Request Parameter die weitere Verarbeitung gesteuert werden kann.
- **Action State**
im Action State werden üblicherweise Methoden eines Spring Beans aufgerufen. Die Beans leiten in der Regel vom Interface `org.springframework.webflow.execution.Action` ab und erhalten über einen Parameter den aktuellen Kontext des Flows. Innerhalb von Actions kann Business Logik ausgeführt werden und die Variablen im Flow manipuliert werden. Abschließend wird anhand des Rückgabewertes die weitere Verarbeitung gesteuert. Alternativ sind auch POJO-Calls möglich, bei denen in der Flow-Definition ein Mapping der Parameter und Rückgabewerte auf die Variablen im Kontext des Flows stattfindet.
- **Subflow State**
Web Flows können freistehende Flows einzeln aufgerufen und ausgeführt werden. Durch die Einbettung in andere Flows in Form von Subflows lassen sie sich allerdings auf einfache Art und Weise wiederverwenden. Innerhalb einer Subflow-Definition können Eingangsparameter an den Subflow übergeben werden bzw. Ergebnisse aus den Subflows in den eigenen Kontext übernommen werden.

- **Decision State**

Im Decision State ist eine einfache Ablaufsteuerung anhand von Kontextattributen des Flows möglich.

- **End State**

Zum Abschluss des Flows lässt sich angeben, wohin zu welcher Webseite bzw. zu welchem Folge-Flow ein Redirect stattfinden soll. Falls der Flow als Sub-Flow gestartet wurde, entscheidet hingegen der aufrufende Flow über den weiteren Ablauf.

Jede Statusbeschreibung (abgesehen vom End State) enthält eine oder mehrere Transitionen, also „Pfade“ zu Folgestatus. Neben den üblichen Events können auch Exceptions berücksichtigt werden und zum Statusübergang führen. An den Transitionen können ebenfalls noch spezifische Aktionen definiert werden, z. B. ist es beim Login unnötig, die Formulardaten zu erfassen, wenn der Anwender „Registrieren“ wählt, daher wird die entsprechende Aktion („bindAndValidate“, siehe unten) nur beim Login durchgeführt.

Die Web Flows werden intern auf Basis von Java-Objektstrukturen abgebildet und können ohne Weiteres direkt in Java implementiert werden. In der Regel werden die Objektstrukturen allerdings zur Laufzeit nach dem Builder-Pattern generiert und durch den Entwickler in Form von XML-Dateien beschrieben. Die Flow-Definition des oben beschriebenen Login-Flows sieht beispielsweise wie folgt aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<flow xmlns="http://www.springframework.org/schema/webflow"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/webflow
      http://www.springframework.org/schema/webflow/spring-webflow-1.0.xsd">

  <start-state idref="viewLogin" />

  <view-state id="viewLogin" view="flows/login">
    <render-actions>
      <action bean="loginAction" method="setupForm"/>
    </render-actions>
    <transition on="login" to="checkLogin">
      <action bean="loginAction" method="bindAndValidate"/>
    </transition>
    <transition on="register" to="register"/>
  </view-state>

  <action-state id="checkLogin">
    <action bean="loginAction" method="checkLogin"/>
    <transition on="error" to="viewLogin"/>
    <transition on="success" to="doLogin" />
  </action-state>

  <subflow-state id="register" flow="register-flow">
    <attribute-mapper>
      <output-mapper>
        <mapping source="username"
              target="flowScope.login.username"/>
      </output-mapper>
    </attribute-mapper>
    <transition on="finish" to="doLogin"/>
  </subflow-state>

  <action-state id="doLogin">
    <action bean="loginAction" method="doLogin"/>
    <transition to="finish"/>
  </action-state>

  <end-state id="finish" view="externalRedirect:/index.jsp"/>
</flow>
```

Die Referenzdokumentation zu Spring Web Flow [2] enthält eine genauere Beschreibung aller Möglichkeiten. Interessant sind hierbei auch die Beispielprojekte sowie die dazu gehörige Dokumentation.

Variablen und Scopes

In den Web Flows kommt der Zwischenspeicherung von Flow-Variablen eine zentrale Bedeutung zu. Zu den bekannten Request- und Session- und Application-Scopes kommen hilfreiche neue Scopes dazu:

- **Flash-Scope**
Im Flash-Scope gespeicherte Werte bleiben nur bis zum Verlassen (submit) des nächsten View-State im Kontext. Dieser Scope eignet sich damit besonders für die Speicherung von Fehlermeldungen und Nachrichten, die nur auf der Folgeseite eines Requests relevant sind, einen Browser-Refresh aber überleben sollen.
- **Flow-Scope**
Der Flow-Scope bezieht sich auf den aktuell ausgeführten Flow und bezieht Parent- und Subflows nicht mit ein. Hierin gespeicherte Werte verfallen bei Abschluss des Flows.
- **Conversation Scope**
Der Conversation Scope umfasst alle Flows, inkl. der Subflows. Eine explizite Übergabe von Werten an Subflows ist damit nicht mehr notwendig. In der Praxis sollte der Conversation Scope mit Bedacht genutzt werden, da sich dies ggf. negativ auf die Wiederverwendbarkeit und die Lesbarkeit der Web Flows auswirkt.

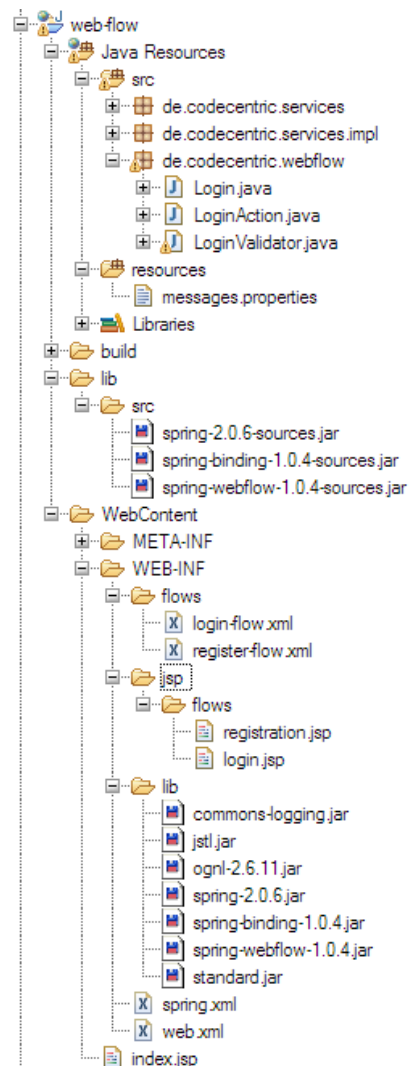
Während der Ausführung von Flows wird die Zwischenspeicherung von Variablen an ein „Flow Execution Repository“ delegiert. Dieses wird immer, wenn ein Flow durch eine User-Interaktion unterbrochen wird, über den aktuellen Status informiert und liefert diesen bei der Fortsetzung des Flows zurück. Der dafür notwendige „Flow-Execution-Key“, der Schlüssel, der zur Identifikation des aktuellen Flow-Status dient, ist daher über Formular- oder Linkparameter an den Flow-Controller zu übergeben.

Neben der einfachen Implementierung, die nur den letzten Stand des Flows speichert, existiert auch zwei Varianten eines Continuation-fähigen Execution Repositories. Diese speichern auch den Status vorhergehender Schritte im Flow ab und unterstützen so in Kombination mit einem „Redirect After Post“ Mechanismus die Nutzung des Back-Buttons im Browser. Da jeder Request im Flow einen eindeutigen Execution Key an den Server sendet, kann, wenn man im Flow zurückgeht, auch der alte Status zum Zeitpunkt dieses Request angezeigt werden, selbst wenn der Server von dem Click auf den Back-Button selbst nichts mitbekommt.

Die standardmäßig aktivierte Variante dieses Continuation-fähigen Repositories hält die Flow-Daten im Server vor. Daneben existiert auch eine Variante, die den Flow-Status clientseitig im Browser ablegt, allerdings sollte hierbei besonderes auf die Anwendungssicherheit und Datenmengen geachtet werden [2].

Aufsetzen eines Spring Web Flow Projekts

Für die Nutzung von Spring Web Flow in eigenen Web-Projekten sind zunächst die Bibliotheken des Projekts in das WEB-INF/lib Verzeichnis des Projekts zu kopieren. Daneben werden die Bibliotheken des Spring-Frameworks sowie für das OGNL-Projekt



benötigt, welches als Implementierung für die Auswertung von Expressions in Web-Flows dient.

Obwohl im Folgenden der Einsatz mit Spring MVC beschrieben wird und konzeptuell hier die meisten Gemeinsamkeiten bestehen, lässt sich Spring Web Flow aufgrund seiner modularen Struktur auch als Controller-Komponente für andere Web-Frameworks einsetzen. So existieren bspw. Plug-Ins für Struts 2 und ICEFaces sowie eine Integration in das Grails Projekt. Die Unterstützung von Struts 1.x, Java-Server Faces (JSF) und des JSR-168 Portlet Standards (über die Spring Portlet Integration) ist ebenfalls gegeben.

Konfiguration von Web und Spring

Wie bei Spring Webprojekten üblich wird in der `web.xml` das Spring-DispatcherServlet konfiguriert:

```
<servlet>
  <servlet-name>spring</servlet-name>
  <servlet-class>
    org.springframework.web.servlet.DispatcherServlet
  </servlet-class>
  <init-param>
    <param-name>
      contextConfigLocation
    </param-name>
    <param-value>
      /WEB-INF/spring.xml
    </param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>spring</servlet-name>
  <url-pattern>*.html</url-pattern>
</servlet-mapping>
```

Die Konfiguration von Spring Web Flow geschieht in unserem Beispiel innerhalb der Spring-Konfigurationsdatei `/WEB-INF/spring.xml`.

Innerhalb der Spring Konfiguration für den Web-Layer wird zunächst der Controller für Spring Web Flow konfiguriert. Der Controller ist demnach über die URI `/flows.html` erreichbar. Der View-Resolver sorgt für das Mapping von View-Namen auf konkrete Pfade auf die JSPs innerhalb des `/WEB-INF/jsp` Ordners und gilt sowohl für Spring MVC als auch für die Flows. Zuletzt wird noch die Position von Übersetzungen für die l18n-Unterstützung konfiguriert.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:flow="http://www.springframework.org/schema/webflow-config"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
    http://www.springframework.org/schema/webflow-config
    http://www.springframework.org/schema/webflow-config/spring-webflow-config-1.0.xsd">

  <!-- Web Layer Konfiguration -->

  <bean name="/flows.html"
    class="org.springframework.webflow.executor.mvc.FlowController">
    <property name="flowExecutor" ref="flowExecutor" />
  </bean>

  <!-- Maps flow view-state view names to JSP templates -->
  <bean id="viewResolver"
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
```

```

    <property name="viewClass"
        value="org.springframework.web.servlet.view.JstlView" />
    <property name="prefix" value="/WEB-INF/jsp/" />
    <property name="suffix" value=".jsp" />
</bean>

<bean id="messageSource"
    class="org.springframework.context.support.ResourceBundleMessageSource">
    <property name="basename" value="messages" />
</bean>
...

```

Neben dem Controller sind für Spring Web Flow noch andere Beans zu instanzieren. Hierfür existiert eine eigene Schemaerweiterung für Spring 2.0. In der unten stehenden Konfiguration wird definiert, dass die Flows unter `/WEB-INF/flows` zu suchen sind und auf `-flow.xml` enden. Die Repository-Konfiguration sieht die Unterstützung von Continuations vor.

```

...

<!-- Spring Web Flow Konfiguration -->

<flow:executor id="flowExecutor" registry-ref="flowRegistry">
    <flow:repository type="continuation" />
</flow:executor>

<flow:registry id="flowRegistry">
    <flow:location path="/WEB-INF/flows/**/*-flow.xml" />
</flow:registry>

...

```

Damit ist Spring Web Flow bereits lauffähig. Für die Unterstützung der konkreten Flows sind allerdings noch einige Spring-Beans zu definieren, hier am Beispiel der Action-Implementierung für den oben beschriebenen Login-Flow. Backing-Beans für Flows können auch in separaten Konfigurationsdateien definiert werden, die dann im Flow über das `<import/>` Tag referenziert werden. In diesem Fall wird die Konfiguration allerdings zur Laufzeit gelesen und initialisiert, was zu zusätzlichem Runtime-Overhead führt.

Innerhalb des Flows kann wie bei Spring-MVC mit Command-Objekten und Validatoren gearbeitet werden. In diesem Fall wird beim Aufruf von „`bindAndValidate`“ in der Action die Klasse `Login` als Command-Objekt instanziiert und mit den Request-Parameterwerten gefüllt sowie über die Klasse `LoginValidator` validiert. Die Passwort-Prüfung wird an den `UserService` delegiert, daher wird dieser ebenfalls injiziert.

```

...

<!-- Backing Beans -->

<bean id="loginAction" class="de.codecentric.webflow.LoginAction">
    <property name="formObjectClass" value="de.codecentric.webflow.Login" />
    <property name="formObjectName" value="login" />
    <property name="validator">
        <bean class="de.codecentric.webflow.LoginValidator" />
    </property>
    <property name="userService" ref="userService" />
</bean>

<!-- Application Services -->

<bean id="userService" class="de.codecentric.services.impl.UserServiceImpl" />

</beans>

```

Implementierung der LoginAction-Klasse

Die Klasse `LoginAction` kapselt die Business-Logik des Login-Flows, insbesondere die Methoden `checkLogin()` und `doLogin()`. Die Klasse leitet von `FormAction` ab, das bereits Methoden für das Binding und Validieren der Request-Parameter bietet als Unterklasse von dem `MultiAction`-Interfaces mehrere Action-Methoden in einer Klasse erlaubt.

`checkLogin()` wird im ActionState „checkLogin“ aufgerufen und kontrolliert den Usernamen und das Kennwort auf Gültigkeit. Die eigentliche Prüfung wird an den `UserService` delegiert. Interessant ist hier im Fehlerfall die Speicherung der Fehlermeldung im Flash-Scope.

`doLogin()` führt den eigentlichen „Login“ aus, indem nach erfolgreicher Prüfung der Username in der Session gespeichert wird (Alternativ ist z. B. auch ein programmatischer Login via Acegi/Spring Security denkbar).

Der Zugriff auf die bekannten Request- und Response-Objekte wird in Spring Web Flow durch eigene Klassen gekapselt. Dies mag wie eine Einschränkung wirken, dadurch wird jedoch auch Unabhängigkeit von der bestehenden View-Implementierung erreicht. `doLogin()` wird auch bei einer alternativen Registrierung aufgerufen. In diesem Fall liegt nur der aus dem Registrierungs-Flow übergebene Username im Login-Command-Objekt gespeichert vor.

```
package de.codecentric.webflow;

import org.springframework.webflow.action.FormAction;
import org.springframework.webflow.execution.Event;
import org.springframework.webflow.execution.RequestContext;

import de.codecentric.services.UserService;

public class LoginAction extends FormAction {
    private UserService userService;

    public LoginAction() {
    }

    public Event checkLogin(RequestContext rc) {
        Login login = (Login) rc.getFlowScope().get("login", Login.class);

        if (userService.isValidLogin(login.getUsername(), login.getPassword())) {
            return success();
        } else {
            rc.getFlashScope().put("errorMessageI18nKey", "login.failed");
            return error();
        }
    }

    public Event doLogin(RequestContext rc) {
        Login login = (Login) rc.getFlowScope().get("login", Login.class);
        rc.getExternalContext().getSessionMap()
            .put("currentUser", login.getUsername());
        return success();
    }

    public void setUserService(UserService userService) {
        this.userService = userService;
    }
}
```

Erstellung von JSP-Views

Der einzige View im Login-Flow zeigt das Login-Formular und wird hier unter Verwendung einer JSP und der Spring Form-TagLib realisiert.

Im Kontext des Views findet sich in `${flowExecutionKey}` der aktuelle Flow-Zustand, der über den Parameter „`_flowExecutionKey`“ an den Flow-Controller wieder zu übergeben ist. Darüber hinaus kann im Submit-Button der „login“-Event übergeben werden, indem hier ein Parameter „`_eventId_login`“ gesetzt wird.

Die Registrierung ließe sich ebenfalls über einen Submit-Button triggern, im Folgenden Beispiel wurde allerdings ein normaler Link gewählt. Hierbei sind der `FlowExecutionKey` und der Event direkt an den Link anzuhängen.

Als Alternative zum standardmäßig verwendeten Parameter-basierten URL-Schema ist auch ein mehr ReST-basiertes Schema konfigurierbar. In Version 2.0 wird es hier noch mal Änderungen geben.

```
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Login</title>
</head>
<body>

<form:form action="flows.html" method="post" commandName="login">
  <input type="hidden" name="_flowExecutionKey"
    value="${flowExecutionKey}">
  <form:errors />
  <c:if test="${!empty errorMessageI18nKey}">
    <fmt:message key="${errorMessageI18nKey}" />
  </c:if>
  <table border="0">
    <tr>
      <td>Name</td>
      <td><form:input path="username" /></td>
      <td><form:errors path="username" /></td>
    </tr>
    <tr>
      <td>Password</td>
      <td><form:password path="password" /></td>
      <td><form:errors path="password" /></td>
    </tr>
    <tr>
      <td>&nbsp;</td>
      <td>
        <input type="submit" name="_eventId_login" value="Login">
        <a href="flows.html?_flowExecutionKey=${flowExecutionKey}
          &_eventId_register=Register">Register</a>
      </td>
    </tr>
  </table>
</form:form>

</body>
</html>
```

Aufruf des Flows

Zum Start des unter `/WEB-INF/flows/login-flow.xml` abgelegten Login-Flows ist im Browser nun folgende URL aufzurufen bzw. von anderen Seiten zu verlinken:

```
http://localhost:8080/webflow/flows.html?_flowId=login-flow
```

Weitere Entwicklung von Spring Web Flow

Zur Zeit ist die Version 2.0 von Web Flow in Entwicklung, deren Fertigstellung für März nächsten Jahres anvisiert ist. Der aktuelle Milestone 2.0M2 lässt schon erahnen, dass eine Menge Verbesserungen Eingang in das Neue Release finden werden. Hier nur einige der Verbesserungen und Features:

- **Spring Scoping für Web Flow-Scopes**
Mit Spring 2.0 und Web Flow 2.0 können als Scope von Spring Beans die entsprechenden Flow-Scopes genutzt werden.
- **Verbesserte Unterstützung von Java Server Faces**, insbesondere
 - Alternative Nutzung von Unified EL anstelle von OGNL
 - Zugriff auf Faces Managed Beans über EL im Flow
 - Verbessertes Error Handling
 - Behebung von Problemen bzgl. „Redirect after Post“
- **Vererbung von Flow-Definitionen**
Flows können als Templates für speziellere Flows dienen
- **Unterstützung paralleler Subflows**
Unterstützung von Flows, in denen mehrere Subflows in beliebiger Reihenfolge durchgeführt werden können
- **Flow Managed Persistence Context**
Umsetzung des Session-per-Conversation Patterns, ähnlich dem Extended PersistenceContext in JBoss Seam. Ein Commit findet erst am Flow-Abschluss statt.
- **Neues URI-Pattern für den Flow-Controller**
Ein neues ReST-basiertes Schema wird zukünftig zum empfohlenen Schema

Links & Literatur

[1] MVC Model 2 http://de.wikipedia.org/wiki/Model_View_Controller#MVC_2

[2] Spring Web Flow 1.0.5 / 2.0M2 Referenzdokumentation <http://www.springframework.org/documentation>

[3] Spring Framework 2.0.7 Referenzdokumentation <http://www.springframework.org/documentation>

[4] Spring Jira Issue Tracker

<http://opensource.atlassian.com/projects/spring/secure/ReleaseNote.jspa?projectId=10050&styleName=Html&version=10640>