

Alles im Fluss

# Spring Web Flow

Reza Nazarian

*Ob Bezahlung, Registrierung oder Serviceanfrage: Die Durchführung solcher Vorgänge findet nicht mehr auf einer einzigen Webseite statt. Vielmehr lotst die Applikation den Anwender von Seite zu Seite, um einen Vorgang Schritt für Schritt abzuschließen. Welcher Entwickler hat sich in der Vergangenheit aber nicht schon einmal in dem Wirrwarr von Webformularen, Eingabevalidierungen und der Berücksichtigung von Zuständen in einer Webapplikation verloren? Dieser Artikel zeigt, wie man mit dem Framework Spring Web Flow solche auf mehrere Schritte verteilte und zustandsbehaftete Webapplikationen zügig und sicher umsetzt.*

► Üblicherweise sind Vorgänge im Web in mehrere Schritte unterteilt. Beispielsweise erfragt die erste Seite eines Bezahlvorgangs die Adresse des Anwenders, auf der zweiten Seite werden Angaben zur Bezahlung verlangt, während die letzte Seite alle bisherigen Eingaben noch einmal zur Bestätigung auflistet. Natürlich hat der Anwender auch die Möglichkeit, zu einem bereits durchgeführten Schritt zurückzukehren und seine Angaben erneut zu tätigen.

Die üblichen Herausforderungen, denen sich ein Entwickler dann stellen muss, sind vielschichtig. In der Regel nimmt ein Servlet die Eingaben des Anwenders von der angezeigten JSP entgegen und startet eine Bearbeitung inklusive Validierung. Nach erfolgreicher Validierung merkt sich das Servlet den aktuellen Zustand und leitet auf die nächste JSP weiter. Benutzt der Anwender dann den „Zurück“-Button des Browsers, stellt das Servlet den vorherigen Zustand wieder her und füllt die JSP mit den zuvor gemachten Angaben. Praktischerweise hat der Anwender auf jeder Seite noch die Möglichkeit, eine Abzweigung einzuschlagen, um sich beispielsweise für die neue Website-Community zu registrieren. Nach dem Beschreiten einer solchen Abzweigung wird er wieder zu dem ursprünglichen Vorgang zurück geführt, um dort fortfahren.

Im Zuge einer Umsetzung solcher Webapplikationen mit einfachen Servlets und JSPs verliert man sich oft in den verschiedenen Zuständen und den Zustandsübergängen. Schnell werden Servlets und JSPs überladen und unübersichtlich.

## Spring Web Flow

Spring Web Flow (s. Abb. 1, [SWF]) kommt hier ins Spiel. Es überträgt den modularen Ansatz von Spring auf zustandsbasierte Webapplikationen. Wie bei einem Aktivitätsdiagramm

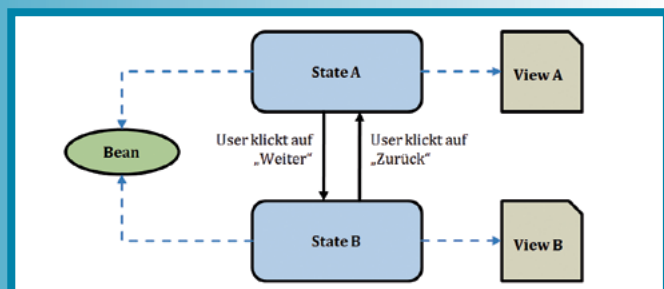


Abb. 1: Übersicht zu Spring Web Flow

werden die Aktivitäten und Übergänge einer Applikation auf einer etwas höheren Ebene in XML beschrieben. Die dabei verwendete Anwendungslogik ist isoliert in Java-Beans abgelegt, die erst einmal nichts von dem Zustand der Applikationen wissen müssen, geschweige denn Übergänge steuern. Die Anzeige übernehmen JSPs, deren Formularen man eine Bean zur Datenspeicherung zuweist. Somit erhält man eine saubere Trennung der Zustände von Anwendungslogik und Anzeige.

## Grundsätzliches

Ein Flow ist als abgeschlossener Automat anzusehen, der eindeutig benannt ist und Einstiegspunkte, Zustände und Aktionen definiert. Ereignisse des Anwenders (z. B. „Klick auf den Ok-Button“) oder programmatische Ereignisse („Benutzer ist noch nicht registriert“) lösen die Übergänge zwischen den Zuständen aus. So ein Zustand heißt erwartungsgemäß State und definiert, welche Aktionen ausgeführt werden, und beschreibt, welches Ereignis welchen Übergang auslöst. Ein State kann dabei beliebig viele Aktionen ausführen und beliebig viele Übergänge und deren Auslöser beschreiben. Grundsätzlich gibt es drei unterschiedliche Typen von States:

- ▼ **View State:** Verwendet eine JSP zur Anzeige (z. B. für ein Eingabeformular) und erfordert damit eine Anwenderinteraktion. Eingaben auf der JSP werden an eine Bean gebunden und beim Übergang zu einem weiteren State validiert.
- ▼ **Action State:** Besitzt keine Anzeige und führt beliebige Aktionen mittels Beans aus (z. B. die technische Registrierung eines Anwenders). Die Auslösung von Übergängen findet programmatisch statt.
- ▼ **Decision State:** Dient nur zur Verzweigung (if-else) zu weiteren States mithilfe von Attributen aus dem Flow. Es findet keine Auslösung von Aktionen statt.

Wie gesagt, werden die States in einer XML-Datei definiert. Dabei ist der erste dort aufgeführte State auch der, in dem der Flow startet. Das Erreichen eines sogenannten End States beendet den Flow, dies geht mit dem Löschen aller Flow-Variablen und häufig mit einem Redirect einher.

## Beans

Im Flow können beliebige Beans verwendet werden, die die Daten eines Formulars aufnehmen, diese validieren oder ausführbare Methoden bereitstellen. Daneben können Beans natürlich beliebige Methoden anbieten, die in einem State aufgerufen werden. Der Flow übergibt bei Bedarf Parameter an die Methode und reagiert auf die Rückgabewerte. Methodenaufrufe finden im Flow mit der Direktive

```
<evaluate expression="$beanName.$methodenName(parameter...)" />
```

statt. Beans werden entweder über die Flow-Definition instanziiert oder stammen aus dem Spring-Kontext.

## Events

Die Auslöser von Transitionen zwischen States sind immer Events. Diese sind zum einen Aktionen des Anwenders (z. B. „Klick auf OK“, „Klick auf Abbrechen“), zum anderen Rückgabewerte von ausgeführten Methoden einer Bean (z. B. true/false oder ein String).

In den JSPs der View States lassen sich solche Events einfach an Buttons binden, indem das HTML-Attribut **Name** mit der Kon-



vention `_eventId_$eventName` gefüllt wird. In dem Flow wertet der View State das Event mit Namen `submit` dann beispielsweise mit `<transition on="submit" to="enterOptions" />` aus. Sprich, das Klicken auf den Button mit dem Wert „`_eventId_submit`“ im Attribut `Name` löst eine Transition zum State `enterOptions` aus.

Action States verwenden die Rückgabewerte von Methodenaufrufen als Events für Transitionen. Dabei lassen sich Strings, Booleans, Enums auswerten oder beliebige Objekte auf null abfragen.

## Subflows

Ein Flow kann auch weitere eigenständige Flows als Subflow beinhalten. Die Einbindung eines Subflows ist dann angebracht, wenn ein laufender Flow einen in sich geschlossenen Prozess startet und dieser dann wieder in den aufrufenden Flow zurückkehrt. Das klassische Beispiel hierfür ist die Registrierung, die aus dem Bezahlvorgang aufgerufen wird. Somit ist eine weitere Möglichkeit zur Modularisierung gegeben, um komplexe Flows in kleinere Flow-Häppchen zu unterteilen.

## Validierung

Beans, die an einen View State und an ein Formular der zugehörigen JSP gebunden sind, nehmen die vom Anwender eingegebenen Daten auf. Dieses Data-Binding findet mithilfe der Tag-Bibliothek Spring Forms [SF] statt. Hiermit wird die Bean als Model an das Formular gebunden und für die Formularfelder das entsprechende Property der Bean angegeben. Alle gemachten Eingaben fließen so automatisch von dem Formular in die Bean.

Für jeden View State ist es nun möglich, diese Eingaben zu validieren. Die Validierung kann die Bean sogar selbst vornehmen. Dafür muss sie lediglich eine Methode anbieten, die gemäß der Namenskonvention `validate${state}` benannt ist (z. B. `validateEnterAddress`). Nachdem der Anwender das Formular ausgefüllt und abgeschickt hat, findet ein automatischer Aufruf dieser Methode statt. Die Validierungsmethode schiebt bei fehlerhaften Eingaben des Anwenders Fehlermeldungen in den Message Context. Werden solche vom Flow entdeckt, kehrt der Flow automatisch in den letzten Zustand zurück, um dort die Fehlermeldungen über die Tag-Bibliothek von Spring Forms anzuzeigen. Alternativ lagert man komplexe oder häufig wiederkehrende Validierungen in eine Validator-Klasse aus. Diese besitzt beliebig viele `validate`-Methoden, die wie in einer Bean über die Namenskonvention `validate${state}` an einen State gebunden sind.

## Setup für Spring Web Flow

In der Regel wird Spring Web Flow in eine Applikation integriert, die auf Spring Web MVC [SWMVC] basiert. Für das Setup von Web Flow benötigt man noch eine Handvoll Bean-Definitionen in einer Spring-Konfiguration. Einstiegspunkt für Flows ist der Flow-Controller, der einer URL zugewiesen ist (z. B. `/flows`). Der Flow-Registry wird jeder Flow über eine eindeutige ID (z. B. `registration`) bekannt gemacht. Der Flow ist dann beispielsweise über `/flows/registration` zu erreichen. Wichtig ist auch noch die Definition des View Resolvers. Dieser löst die Namen der Views, die von einem View State verwendet werden, auf. Beispielsweise findet sich bei Verwendung der ID `enterAddress` für die View die Datei „`enterAddress.jsp`“ in `WEB-INF/jsp/` (s. Listing 1).

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>
  <flow:flow-registry id="flowRegistry"
    flow-builder-services="flowBuilderServices">
    <flow:flow-location path="/WEB-INF/flows/registration.xml"
      id="registration" />
  </flow:flow-registry>

  <flow:flow-executor id="flowExecutor" flow-registry="flowRegistry"/>

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

  <bean id="viewResolverInternal" class=
    "org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/jsp/" />
    <property name="suffix" value=".jsp" />
  </bean>
</beans>
```

Listing 1: Das notwendige Setup für Spring Web Flow

## Durchlauf eines Beispiels

Eine Erläuterung der Funktionsweise findet im Folgenden anhand eines Beispiels (s. Abb. 2) statt, das den Flow zur Registrierung eines Nutzers auf einer Website darstellt.

Der Flow ist über die ID `registration` beim Flow-Controller angemeldet (s. Listing 2). Ein Einsprung findet dementsprechend über `/flows/registration` statt. Über die Direktive `var` findet eine Instanziierung der beiden Beans `registrationBean` und `registrationHelper` statt. Diese Beans übernehmen die Datenhaltung und die Logik des folgenden Flows.

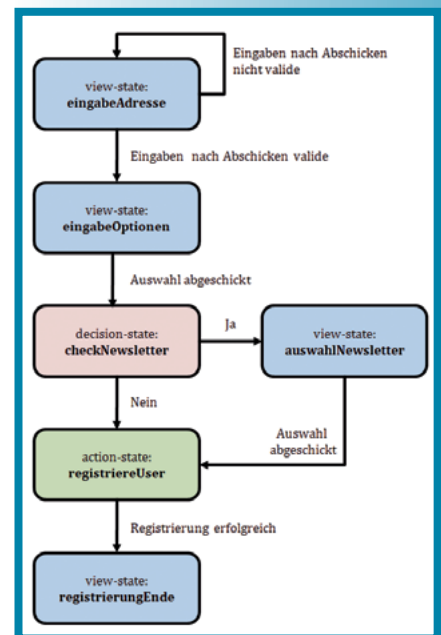


Abb. 2: Web Flow für einen Registrierungsprozess

```
<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-2.0.xsd">

  <var name="registrationBean"
    class="com.js.webflow.registrationBean" />
  <var name="registrationHelper"
    class="com.js.webflow.registrationHelper" />

  <view-state id="enterAddress" model="registrationBean"
    view="enterAddress">
    <transition on="submit" to="enterOptions" />
  </view-state>
```

```

<view-state id="enterOptions" model="registrationBean"
  view="enterOptions">
  <transition on="submit" to="checkNewsletter" />
</view-state>

<decision-state id="checkNewsletter">
  <if test="registrationBean.hasSelectedNewsletter()"
    then="enterNewsletterType" else="registerUser"/>
</decision-state>

<action-state id="registerUser">
  <evaluate expression=
    "registrationHelper.register(registrationBean)" />
  <transition on="success" to="registrationEnd"/>
</action-state>

<view-state id="enterNewsletterType" model="registrationBean"
  view="enterNewsletterType">
  <transition on="submit" to="registerUser" />
</view-state>

<end-state id="registrationEnd" view="registrationEnd" />
</flow>

```

Listing 2: Ein exemplarischer Web Flow

Der erste und somit automatisch ausgeführte State des Flows ist der View State `enterOptions`. Hier soll der Benutzer Name, Adresse usw. angeben. Die zur Anzeige verwendete JSP wird über das Attribut `view` angegeben. Für alle Flows ist an zentraler Stelle definiert, wo die JSPs zu finden sind (beispielsweise `WEB-INF/jsp`). Demnach wird an dieser Stelle nach einer JSP mit dem Namen `enterOptions.jsp` gesucht. Für die Datenspeicherung wird die Bean `registrationBean` über das Attribut `model` verwendet. In der JSP sind somit alle zu speichernden Felder auf die Bean abgebildet.

Schickt der Benutzer das Formular in der JSP nun ab, so wird das Event `submit` gesendet. In diesem Fall findet die Transition zum State `enterOptions` statt. Aber noch nicht ganz, denn erst einmal müssen die Formulardaten durch die Validierung. Diese ist in der `registrationBean` definiert und wird automatisch aufgerufen. Schlägt die Validierung fehl, so werden die entsprechenden Fehlermeldungen im Formular angezeigt und die Transition findet nicht statt.

Im Erfolgsfall sieht der Benutzer den nächsten View State `enterOptions`, um weitere Optionen wie den Empfang von Newsletters zu wählen. Auch hier speichert die entsprechende JSP (`enterOptions.jsp`) die Daten in die `registrationBean`. Das Absenden des Formulars löst hier schließlich die Transition zum Decision State `checkNewsletter` aus.

Die Ausführung des Decision States ist für den Benutzer nicht sichtbar. Hier gabelt sich der Flow. Unsere Bean bietet die Methode `hasSelectedNewsletter()` an, die `true` zurückliefert, wenn der Benutzer im vorherigen View State den Newsletter bestellt hat. In diesem Fall leitet der Decision State den Benutzer zum View State `enterNewsletterType` weiter, um eine Auswahl anzubieten, ob ein Verschicken als HTML- oder als Text-Newsletter gewünscht ist. Auch hier speichert die zugewiesene JSP die Angaben in die `registrationBean`. Nach dem Abschicken des Formulars findet eine Weiterleitung auf den Action State `registerUser` statt. Liefert `hasSelectedNewLetter()` im Decision State ein `false` zurück, so leitet der Decision State gleich an den Action State `registerUser` weiter.

Ein Action State ist wie ein Decision State für den Benutzer erst einmal nicht sichtbar. Über die Direktive `evaluate expression` führt der State beliebig viele Methoden einer Bean aus und kann auf die Rückgabewerte reagieren. In unserem Fall verwendet der Action State die Methode `register` der zweiten Bean `registrationHelper`, um die eigentliche Registrierung vorzunehmen. Für die erforderlichen

Parameter übergibt der Flow unsere prall gefüllte `registrationBean`. `register` liefert Null zurück, wenn die Registrierung fehlschlägt, oder eben nicht Null, wenn die Registrierung erfolgreich ist. Der Erfolgsfall löst eine Transition zu unserem letzten State, dem End State `registrationEnd`, aus.

Das Erreichen des End States schließt den Flow ab. In dem Beispiel ist zudem eine optionale JSP zugewiesen, die nichts anderes macht, als dem Benutzer die erfolgreiche Registrierung mitzuteilen.

Anzumerken ist noch, dass die Verwendung des „Back“- und des „Refresh“-Buttons im Browser das Framework nicht aus dem Tritt bringt. Der vorherige Zustand wird wiederhergestellt oder der bestehende wird einfach nicht verändert.

## Testen von Flows

Das Testen von Flows muss nicht im Kontext der Webapplikation stattfinden. Vielmehr bietet Spring Web Flow die Testklasse `AbstractFlowExecutionTests` an, die vom JUnit-Testfall abgeleitet ist und isoliertes Testen von Flows ermöglicht. Dabei lassen sich zusammen mit einem Mock-Kontext Zustände des Flows herstellen und Events simulieren, um das Verhalten des Flows gezielt zu testen.

## Fazit

Mit Spring Web Flow steht ein ausgereiftes Framework zur Verfügung, das die Lesbarkeit und den Überblick über mehrstufige und zustandsbehaftete Webanwendung ermöglicht. Die Trennung von Ablauflogik in XML und dem eigentlichen Programmcode in abgeschlossenen Java-Beans beschleunigt die Realisierung und erhält die Flexibilität aller Komponenten. Sowohl gestandene Spring-Entwickler als auch Neulinge finden sich schnell zurecht.

Auch für die Unterstützung durch eine Entwicklungsumgebung ist gesorgt: Die auf Eclipse basierende IDE „SpringSource Tool Suite“ [SSTS] ist auf die Entwicklung mit dem Spring-Framework ausgelegt. Speziell für Spring Web Flow bietet sie hilfreiche Wizards und Editoren an.

## Links

[SF] Spring Forms, <http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/view.html#view-jsp-formtaglib>

[SSTS] SpringSource Tool Suite,

<http://www.springsource.com/products/sts>

[SWF] Spring Web Flow, <http://www.springsource.org/webflow>

[SWMVC] Spring Web MVC,

<http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/mvc.html>



**Reza Nazarian** ist Senior Consultant bei der Unternehmensberatung Acando in Hamburg. Seine fachlichen Schwerpunkte liegen in den Bereichen Portal-Technologien und Content-Management-Systeme. Reza Nazarian ist seit 2002 in der IT-Branche tätig und hat Erfahrungen als Berater, Technischer Projektmanager und Software-Ingenieur gesammelt.  
E-Mail: [reza.nazarian@acando.de](mailto:reza.nazarian@acando.de)