HANSER

Leseprobe

zu

JavaServer™ Faces und Jakarta Server Faces 2.3

von Bernd Müller

Print-ISBN: 978-3-446-45670-9
E-Book-ISBN: 978-3-446-45977-9
E-Pub-ISBN: 978-3-446-47008-8

Weitere Informationen und Bestellungen unter
https://www.hanser-kundencenter.de/fachbuch/artikel/9783446456709
sowie im Buchhandel

Inhalt

Vorwort zur 3. Auflage

In der 2. Auflage haben wir die vielen Neuerungen von JSF 2.0 integriert. Die Version 2.1 war eher kosmetischer Natur. Die Version 2.2 brachte wiederum sehr viel Neues, die Version 2.3 ebenso. Es ist also dringend an der Zeit für eine 3. Auflage.

Wie bereits in der 2. Auflage praktiziert, soll auch diese 3. Auflage keine Darstellung aller JSF-Versionen sein. Sie werden daher in der Regel keine Bezüge auf ältere JSF-Versionen, sondern ausschließlich den aktuellen Stand 2.3 beschrieben finden. Damit geht einher, dass einige Abschnitte der früheren Auflagen verschwunden sind, andere erheblich überarbeitet wurden und das Buch damit eine neue Struktur erhalten hat. Mehr noch, Sie halten praktisch ein komplett neues Buch in der Hand.

Während dieses Buch entstand, wurde aus *JavaServer Faces* 2.3 *Jakarta Server Faces* 2.3. Diese beiden Spezifikationen sind praktisch identisch, so dass das Buch beide Systeme beschreibt. Wir sprechen im Folgenden immer von JavaServer Faces, meinen aber beide Spezifikationen.

In der 2. Auflage nutzten wir Eclipse als Build-System. Wir haben nun alle Projektbeispiele auf Maven umgestellt und verwenden WildFly als Application-Server. Da die Projektbeispiele nur offizielle APIs verwenden, sind sie auf allen zertifizierten Servern verwendbar.

Bernd Müller, April 2021

bernd.mueller@ostfalia.de

Sprechende Event-Listener-Namen

Die von uns vergebenen Bezeichner für die Event-Listener-Methoden sind schlecht gewählt. Sie widersprechen der Clean-Code-Forderung nach aussagekräftigen und sprechenden Namen (meaningful names). Wir nutzen diese Gelegenheit, um Sie auf die unserer Meinung nach besonders hohe Gefahr der Verwendung eines „falschen" Listeners hinzuweisen. Wesentlich sinnvollere Namen wären `logPageImpression()` und `redirectIfBrowserTooOld()`. Ebenfalls sinnvoll ist die Prüfung, ob der Event-Typ der richtige ist. Beim zweiten Listener ist dies `PreRenderComponentEvent`. Implementieren Sie diese Prüfung als Erweiterung der Methode `preRenderComponent()` in Listing 2.50.

Wir vervollständigen diesen Abschnitt mit der Nennung dreier Annotationen im Rahmen der System-Event-Verarbeitung von JSF. Die Annotation `@ListenerFor` registriert für ein System-Event einen Listener. Die Container-Annotation `@ListenersFor` erlaubt es, mehrere Listener zu registrieren. Die Verwendung ist allerdings nur für Unterklassen von `UIComponent` und `Renderer` erlaubt, so dass sie nur bei der Entwicklung eigener Komponenten zum Einsatz kommen.

Falls eigene Event-Klassen als Unterklassen von `ComponentSystemEvent` definiert werden, sind diese mit `@NamedEvent` zu annotieren, um sie als Komponenten-Events im `<f:event>`-Tag verwenden zu können.

2.6 HTML5

Die erste Version von JSF nahm explizit Bezug auf die HTML-Version 4.01, die im Dezember 1999 als W3C-Recommendation veröffentlicht wurde. Was die Weiterentwicklung von HTML angeht, begann danach eine lange Zeit der Stagnation. Erst im Oktober 2014 wurde HTML5 veröffentlicht [URL-HTML5]. Dann ging es allerdings Schlag auf Schlag weiter: HTML 5.1 im November 2016 [URL-HTML51], HTML 5.1 2nd Edition im Oktober 2017 [URL-HTML512] und schließlich HTML 5.2 im Dezember 2017 [URL-HTML52], alle als W3C-Recommendations. Diese schnelle Versionsfolge ist als Reaktion auf die Gründung der *Web Hypertext Application Technology Working Group (WHATWG)* zu sehen, die sich 2004 als Antwort auf die langsame HTML-Weiterentwicklung gründete. Die WHATWG ist ein Zusammenschluss von Browser-Herstellern und arbeitet alternativ an einem *„lebenden"* Standard, dem *HTML Living Standard* [URL-HTMLLS]. Unter dem Begriff *HTML5* wird häufig die letzte W3C-Recommendation und/oder der Living Standard verstanden. Diese Versionen sind zwar im Detail nicht 100% deckungsgleich, für unsere Zwecke jedoch ausreichend deckungsgleich, so dass wir letztendlich nicht unterscheiden. Erfreulicherweise haben sich W3C und WHATWG im Mai 2019 darauf geeinigt, in Zukunft an einem gemeinsamen Standard von HTML und DOM zu arbeiten, so dass in einigen Jahren hoffentlich nur noch jeweils eine Version existiert.

Diese historische Betrachtung der HTML-Entwicklung zeigt das Dilemma, vor dem JSF steht. Wie kann JSF der Entwicklung von HTML möglichst zeitnah folgen? Ganz einfach: mit einem Mechanismus, der es erlaubt, neue Attribute bestehender HTML-Tags transpa-

rent durchzureichen und neue HTML-Elemente direkt in JSF-Seiten verwenden zu können und diese mit JSF-Komponenten im Komponentenbaum zu verbinden. In JSF 2.2 wurden dazu die sogenannten *Pass-Through-Attribute* und *Pass-Through-Elemente* eingeführt.

2.6.1 Pass-Through-Attribute

Das JSF-Tag `<h:inputText>` wird in HTML zu `<input type="text">` gerendert. Das `<input>`-Tag bekam mit HTML5 neue Attribute, z.B. `placeholder`, aber auch neue Attributwerte, wie z.B. `type="email"`. Mit Pass-Through-Attributen können beide Anwendungsfälle realisiert werden. Dazu wird der XML-Namensraum `http://xmlns.jcp.org/jsf/passthrough` eingebunden und die entsprechenden Attribute mit dem Namensraumkürzel als Präfix mit Werten versehen. Das folgende Beispiel zeigt dies.

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:pt="http://xmlns.jcp.org/jsf/passthrough">
  ...
  <h:inputText id="email" value="#{ptc.email}"
      pt:type="email" pt:placeholder="E-Mail" />
  ...
```

Es werden also der in HTML5 neue Input-Typ `email` und das neue Attribut `placeholder` verwendet. Das gerenderte Endergebnis lautet:

```
<input type="email" placeholder="E-Mail" ...>
```

Falls keine syntaktisch korrekte E-Mail-Adresse eingegeben wird, weigert sich der Browser, nicht JSF, das Formular abzuschicken. Listing 2.30 zeigt die entsprechende Fehlermeldung in Chrome.

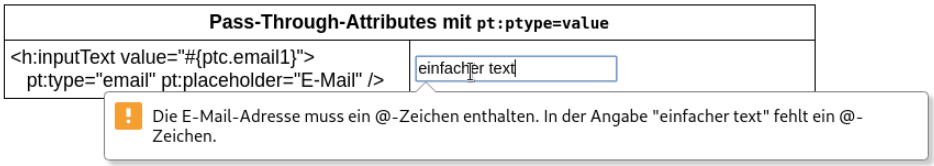

Bild 2.30 Fehlermeldung in Chrome

Pass-Through-Attribute und Namensraumpräfix

Wir raten zum Namensraumkürzel `pt` statt des im Web häufig gesehenen einfachen `p`. Dies beugt Wartungsproblemen bei der Verwendung der verbreiteten Komponentenbibliothek PrimeFaces [URL-PRIME] der Firma PrimeTex vor, die das Kürzel `p` benutzt.

JavaServer Faces stellt drei alternative Verwendungsarten von Pass-Through-Attributen bereit. Neben der bereits vorgestellten Alternative durch Attribute mit dem Pass-Through-Namensraum kann man auch die Tags `<f:passThroughAttribute>` und `<f:passThroughAttributes>` verwenden.

Zunächst das überarbeitete Beispiel unter Verwendung des Tags `<f:passThrough-Attribute>`. Im folgenden JSF-Code werden über die Tag-Attribute `name` und `value` die entsprechenden Werte gesetzt.

```
<h:inputText value="#{ptc.email}">
  <f:passThroughAttribute name="type" value="email"/>
  <f:passThroughAttribute name="placeholder" value="E-Mail"/>
</h:inputText>
```

Beim Tag `<f:passThroughAttributes>` verlagert sich die Definition von Attributen und Werten von der JSF-Seite nach Java, da das `value`-Attribut von `<f:passThrough-Attributes>` ein EL-Werteausdruck vom Typ `Map<String, Object>` ist. Die Überarbeitung stellt sich auf JSF-Seite dann folgendermaßen dar:

```
<h:inputText value="#{ptc.email}">
  <f:passThroughAttributes value="#{ptc.attributes}" />
</h:inputText>
```

In Java ist eine Map bereitzustellen, die die entsprechenden HTML-Attribute als Schlüssel und HTML-Attributwerte als Werte codiert. Der Getter `getAttributes` der Klasse `PassThroughController.java` in Listing 2.51 leistet genau dies.

Listing 2.51 Pass-Through-Attribute durch Methode (Klasse `PassThroughController.java`)

```
@Named("ptc")
@RequestScoped
public class PassThroughController {

  ...
  public Map<String, Object> getAttributes() {
    return new HashMap<String, Object>() {{
        put("type", "email");
        put("placeholder", "E-Mail");
    }};
  }
  ...
}
```

HTML5 Date-Picker

HTML5 definiert einen Date-Picker durch ein Eingabeelement vom Typ `date` (`<input type="date">`). Implementieren Sie eine Datumseingabe über dieses Element mit Bindung an ein Property vom Typ `LocalDate`. Tipp: Der übergebene Wert ist von der Form `YYYY-MM-DD`.

2.6.2 Pass-Through-Elemente

Mit Pass-Through-Elementen ist es möglich, direkt HTML zu verwenden, diese HTML-Elemente aber trotzdem mit JSF-Komponenten im Komponentenbaum

zu verbinden. Der Schlüssel hierzu ist ein weiterer XML-Namensraum, diesmal `http://xmlns.jcp.org/jsf`. Wird ein Attribut eines HTML-Elements mit diesem Namensraum verwendet, wird das HTML-Element zu einem Pass-Through-Element und damit zu einem JSF-Tag. Die Entscheidung, welches JSF-Tag verwendet wird, fällt auf Grund des HTML-Elements, bei Mehrdeutigkeiten plus Attributwert. Bei einem `<input>` mit `type="email"` oder `type="date"` wird `<h:inputText>` verwendet, bei `type="button"` ein `<h:commandButton>`. Die vollständige Zuordnungstabelle ist relativ umfangreich und im JavaDoc des Interface `TagDecorator` im Package `javax.faces.view.facelets` wiedergegeben. Wir verzichten daher hier auf eine Darstellung, da zudem die Zuordnung relativ intuitiv ist und häufig nicht benötigt wird. Eine Ausnahme hiervon ist etwa die Verwendung der Komponentenklasse in einer Komponentenbindung (Abschnitt 2.2.9).

Unser bisheriges Beispiel zur Eingabe einer E-Mail-Adresse sieht mit Pass-Through-Elementen dann wie folgt aus:

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:jsf="http://xmlns.jcp.org/jsf">
  ...
  <input jsf:id="email" jsf:value="#{ptc.email}"
      type="email" placeholder="E-Mail" />
  ...
```

Wie bereits erwähnt, erzeugt JSF hierfür eine `<h:inputText>`-Komponente. Da die Attribute `id` und `value` den Präfix des Pass-Through-Elemente-Namensraums haben, werden diese Attribute direkt an das `<h:inputText>` weitergereicht. Die Attribute `type` und `placeholder` besitzen diesen Präfix nicht und werden von JSF daher wie Pass-Through-Attribute behandelt.

Pass-Through-Elemente und Namensraumpräfix

Der von uns verwendete Präfix `jsf` für den Namensraum der Pass-Through-Elemente hat sich in der Praxis durchgesetzt. Wir raten, diesen Präfix zu verwenden.

JSF-Komponentenklasse der Transformation

Überzeugen Sie sich davon, dass in obigem Beispiel aus dem HTML-Element `<input>` die JSF-Komponente `<h:inputText>` und damit die Komponentenklasse `HtmlInputText` wird.

Wir haben gesehen, dass HTML-Elemente durch die Verwendung von Pass-Through-Elementen in JSF-Komponenten transformiert werden. Da es jedoch sehr viel mehr HTML-Elemente als JSF-Komponenten gibt, stellt sich die Frage, ob auch HTML-Elemente mit Pass-Through-Elementen verwendet werden können, für die es keine JSF-Entsprechung gibt? Die Antwort lautet: ja! Ermöglicht wird dies durch eine generische JSF-Komponente, repräsentiert durch das Tag `<jsf:element>`, das in JSF-Seiten nicht verwendet wird. Die dahinter stehende Komponente ist `UIPanel`, die sonst lediglich zur Gruppierung von Sohn-

komponenten dient. Die Details sollen hier nicht weiter ausgeführt, sondern an einem Beispiel erläutert werden. Listing 2.52 zeigt die Verwendung des HTML-Elements `<progress>`.

Listing 2.52 HTML-Element ohne passende JSF-Komponente (`progress.xhtml`)

```
<progress jsf:id="progress" max="59" value="#{progressController.value}">
  <f:ajax event="click" render="progress" />
</progress>
```

Das `<progress>`-Tag besitzt keine JSF-Entsprechung, wird aber durch das Pass-Through-Element `jsf:id="progress"` zu einer JSF-Komponente. Zunächst jedoch zum `<progress>`-Tag selbst. Es repräsentiert den Fortschrittsprozess einer Task (Fortschrittsbalken), der von 0 bis zu dem im Attribut `max` angegebenen, einheitslosen Wert reicht. Das Attribut `value` gibt den augenblicklichen Wert an. Im Beispiel ist der Maximalwert 59, da wir den Wert von 0 bis 59, nämlich den Sekundenanteil der aktuellen Zeit, laufen lassen. Das enthaltene `<f:ajax>`-Tag macht das `<progress>`-Element klicksensitiv: Ein Mausklick führt dazu, dass das Element neu gerendert wird, wozu über den EL-Ausdruck im `value`-Attribut der Sekundenanteil der aktuellen Zeit vom Server geholt wird. Dass das Zusammenspiel von HTML5, Pass-Through-Elementen und JSF funktioniert, sieht man daran, dass im `<progress>`-Tag die Komponenten-Id `progress` definiert und im `<f:ajax>`-Tag verwendet wird.

Wir sind nun am Ende unserer Darstellung von Pass-Through-Attributen und Pass-Through-Elementen angekommen und wollen den Abschnitt mit einer kleinen Übung abschließen. HTML5 erlaubt die Verwendung sogenannter *Custom Data Attributes*. Dies sind Attribute, die mit dem String `data-` beginnen und einen beliebigen Wert haben dürfen. Die intendierte Verwendungsmöglichkeit ist die lokale Speicherung bzw. Bereitstellung von Daten innerhalb der Seite.

Klickbarer Text

Das `<h:outputText>`-Tag besitzt kein `onclick`-Attribut. Realisieren Sie mit Hilfe des `<h:outputText>`-Tags und Pass-Through-Attributen einen klickbaren Text, der mit JavaScripts `alert()` einen Text ausgibt, der über ein Custom-Data-Attribut übergeben wird.

2.7 Ajax

Wir haben das `<f:ajax>`-Tag bereits mehrfach in Beispielen verwendet, sind aber nicht weiter darauf eingegangen. Dieser Abschnitt 2.7 befasst sich nun explizit mit dem `<f:ajax>`-Tag, während wir in Abschnitt 4.5 detailliert auf Ajax im Allgemeinen sowie die Implementierung und Integration innerhalb der JavaServer Faces eingehen.

2.7.1 Das <f:ajax>-Tag

Mit dem <f:ajax>-Tag werden eine oder mehrere UI-Komponenten mit Ajax-Funktionalität versehen. Die Verwendung innerhalb einer oder als Wrapper um mehrere andere Komponenten bestimmt diese Funktionalität, also die Art und Weise, wie der Ajax-Request initiiert wird. Dies richtet sich nach der Komponentenart, kann aber in einem gewissen Bereich überschrieben werden. Nachdem der Request abgeschickt wurde, müssen auf dem Server bestimmte Komponenten *ausgeführt* werden, die Spezifikation spricht von *execute*. Danach müssen (eventuell andere) Komponenten gerendert werden. Die Verwendung von Ajax verändert das in Abschnitt 2.1 beschriebene und in Bild 2.1 auf Seite 17 grafisch dargestellte Bearbeitungsmodell einer JSF-Anfrage nicht prinzipiell, sondern grenzt lediglich die in den einzelnen Phasen zu verwendenden Komponenten ein. Die Phasen 1 bis 5 bilden die Execute-Phase, die Phase 6 die Render-Phase. Zu den wichtigsten Attributen des <f:ajax>-Tags gehören daher das execute- und das render-Attribut.

Um das prinzipielle Vorgehen darstellen zu können, entwickeln wir ein kleines Beispiel, eine weitere Kundeneingabe. Die Klasse Customer besitzt lediglich die beiden Properties firstname und lastname und muss daher nicht explizit dargestellt werden. Die JSF-Seite customer-ajax.xhtml in Listing 2.53 besitzt zwei Eingabekomponenten (Zeilen 3/4 und 6/7).

Listing 2.53 JSF-Seite customer-ajax.xhtml zur Eingabe und Anzeige des Kundennamens

```
1  <h:panelGrid columns="2">
2    Vorname:
3    <h:inputText id="firstname"
4                 value="#{ajaxController.customer.firstName}" />
5    Nachname:
6    <h:inputText id="lastname"
7                 value="#{ajaxController.customer.lastName}" />
8    <h:commandButton action="#{ajaxController.save}" value="Speichern">
9      <f:ajax execute="@form" render="@form" />
10   </h:commandButton>
11   <h:panelGroup />
12   Kompletter Name:
13   <h:outputText id="whole"
14                 value="#{ajaxController.wholeCustomerName()}" />
15 </h:panelGrid>
```

Wird die Schaltfläche in Zeile 8/10 betätigt, erfolgt ohne Vorhandensein des <f:ajax>-Tags in Zeile 9 ein normaler HTTP-Request durch das <h:commandButton>-Tag. Das Formular wird submitted, JSF durchläuft den kompletten Bearbeitungszyklus und die Antwort wird an den Client geschickt, der sie rendert. Wenn, wie dargestellt, das <f:ajax>-Tag in Zeile 9 vorhanden ist, erfolgt bei Betätigung der Schaltfläche ein XML-HTTP-Request [URL-XMLHTTP]. Das Formular wird submitted, JSF durchläuft ebenfalls den kompletten Bearbeitungszyklus. Es werden jedoch nur die Komponenten exekutiert, die im execute-Attribut angegeben sind. Genauso werden nur die Komponenten gerendert, die im render-Attribut aufgeführt sind. Nur für diese im render-Attribut aufgeführten Komponenten

wird eine XML-Repräsentation an den Client geschickt und die entsprechenden HTML-Elemente im Client werden durch JavaScript im DOM aktualisiert.

Um diese Ajax-Funktionalität tatsächlich demonstrieren zu können, wird im Listing in Zeile 13/14 die Methode `wholeCustomerName()`, die lediglich Vor- und Nachname des Kunden zurückgibt, zur Ausgabe verwendet:

```
public String wholeCustomerName() {
  return customer.getFirstName() + " " + customer.getLastName();
}
```

Properties und Java-8-Default-Methoden

Wir haben bewusst keinen Getter `getWholeCustomerName()` verwendet, weil dies dem Property-Pattern der JavaBeans-Spezifikation [URL-JB] widerspricht. Sollten Sie dies allerdings bevorzugen und dann den EL-Ausdruck `#{ajaxController.wholeCustomerName}` verwenden, ist dies problemlos möglich. Problematisch wird es, wenn die Methode als Java-8-Default-Methode eines Interface definiert ist. Da die Expression-Language die Werteausdrücke per Reflection auswertet, wird die Default-Methode nicht gefunden. Sie müssen stattdessen den EL-Ausdruck `#{ajaxController.getWholeCustomerName()}` verwenden.

Zurück zu Ajax: Im Beispiel ist der Wert von `execute` und `render` jeweils `@form`. `@form` ist ein Schlüsselwort, das neben anderen im JavaDoc der Klasse `SearchKeywordResolver` definiert wird und das Formular der verwendeten Komponente benennt. Die für das `<f:ajax>`-Tag relevanten Schlüsselwörter sind in Tabelle 2.10 wiedergegeben.

Tabelle 2.10 Ajax-Schlüsselwörter (`execute`- und `render`-Attribute)

Schlüsselwort	Bedeutung
@all	Alle Komponenten der View
@form	Das umschließende Formular der Basiskomponente
@none	Keine Komponente
@this	Die Basiskomponente

Die Verwendung von `@form` in den beiden Attributen führt also dazu, dass alle Komponenten des Formulars sowohl exekutiert als auch gerendert werden. Damit ist das Verhalten praktisch identisch zur alleinigen Verwendung des `<h:commandButton>`-Tags. Der Unterschied ist, dass ohne Ajax die Antwort per HTTP erfolgt und der Browser die komplette Seite rendert, während mit Ajax die Antwort per XML-HTTP-Request erfolgt und JavaScript den DOM aktualisiert. Als Anwender erkennen wir dies durch ein leichtes Zittern der Browser-Darstellung im Nicht-Ajax-Falle.

Sollen die Möglichkeiten von Ajax sinnvoller verwendet werden, müssen wir also im `execute`- und `render`-Attribut möglichst konkret, am besten minimalistisch, werden und für `execute` nur die Eingaben, für `render` nur die Ausgaben benennen, im Beispiel also etwa

```
<f:ajax execute="firstname lastname" render="whole" />
```

statt

```
<f:ajax execute="@form" render="@form" />
```

schreiben. Der Default-Wert für `execute` ist `@this`, für `render` ist es `@none`. Sind mehrere Komponenten-Ids anzugeben, so werden diese durch Leerzeichen getrennt hintereinander angegeben.

Sinnloses `@all`

Die ursprüngliche Idee des Schlüsselworts `@all` war es, alle Komponenten der View zu umfassen. Da HTML/HTTP verschachtelte Formulare verbietet und bei mehreren Formularen innerhalb einer View immer nur ein Formular submitted werden kann, ist die Verwendung von `@all` identisch zur Verwendung von `@form`. Wir raten, nur `@form` zu verwenden.

Während die Befehlskomponenten `<h:commandButton>` und `<h:commandLink>` im Ajax-Standardfall auf Action-Events reagieren, reagieren Eingabekomponenten auf Value-Change-Events. Zu den Eingabekomponenten zählen `<h:inputText>`, `<h:inputTextarea>`, `<h:inputSecret>` und alle Komponenten, die mit `<h:select...>` beginnen. Wo oder besser wie wird aber das Standardverhalten einer Komponente definiert? Über das Verhaltensmodell einer Komponente (Component Behavior Model) können einer Komponente zusätzliche Verhaltensweisen zugewiesen werden. Realisiert wird dies über das Interface `Behavior`, das als einziges Sub-Interface `ClientBehavior` besitzt. Als einzige konkrete Implementierung existiert die Klasse `AjaxBehavior`, über die das angesprochene Ajax-Verhalten realisiert wird. Die Klasse `UIComponentBase` enthält entsprechende Methoden, die in diesem Zusammenhang relevant sind, z.B. `addClientBehavior()`, `getClientBehaviors()`, `getEventNames()` und `getDefaultEventName()`. Dies sind gerade die vier Methoden, die das Interface `ClientBehaviorHolder` enthält. Alle UI-Komponenten, die dieses Interface realisieren, können mit Hilfe des `<f:ajax>`-Tags mit Ajax-Funktionalitäten versehen werden. Da die Implementierung dieses Interface aber nur für eigenentwickelte Ajax-Komponenten interessant ist, wollen wir den Ausflug in die Ajax-Implementierung von JSF an dieser Stelle abbrechen und uns wieder dem Beispiel widmen.

Das neue Ziel ist es, auf die Befehlskomponente `<h:commandButton>` des Beispiels verzichten zu können und bei einer Änderung des Vor- oder Nachnamens ohne weitere Benutzeraktivität den kompletten Namen zu aktualisieren. Listing 2.54 zeigt die Überarbeitung.

Listing 2.54 Überarbeitete JSF-Seite `customer-ajax.xhtml`

```
1 <h:panelGrid columns="2">
2   Vorname:
3   <h:inputText id="firstname"
4         value="#{ajaxController.customer.firstName}">
5     <f:ajax execute="@this" render="whole"  />
6   </h:inputText>
7   Nachname:
8   <h:inputText id="lastname"
```

```
        value="#{ajaxController.customer.lastName}">
    <f:ajax execute="@this" render="whole"  />
  </h:inputText>
  Kompletter Name:
  <h:outputText id="whole"
      value="#{ajaxController.customer.firstName}
             #{ajaxController.customer.lastName}" />
</h:panelGrid>
```

Die beiden Eingabekomponenten in Listing 2.54 enthalten nun jeweils ein `<f:ajax>`-Tag, wobei die `execute`-Attribute die jeweilige Eingabe (`@this`), die `render`-Attribute die Id der Ausgabekomponente des kompletten Namens enthalten. Wir haben den Wert der Ausgabe ebenfalls überarbeitet und konkatenieren nun Vor- und Nachname mit Hilfe eines einfachen EL-Ausdrucks statt mit einer Java-Methode, um den Leser noch einmal an die Mächtigkeit der Expression-Language zu erinnern.

Wenn Sie das Beispiel ausprobieren, werden Sie feststellen, dass der komplette Name erst aktualisiert wird, wenn der Eingabefokus eine der beiden Eingaben verlässt. Was ist der Grund hierfür? Das bereits für Eingabekomponenten genannte Defaul-Ereignis ValueChanged! Ein solches Event tritt ein, wenn sich der Wert einer Eingabe geändert hat *und* die Eingabekomponente den Fokus verliert. Dies entspricht dem JavaScript `onchange`-Event. Soll auf die Eingabe noch feingranularer reagiert werden, kann ein entsprechend anderes JavaScript-Event verwendet werden, z.B. `onkeyup`, was wir gleich tun werden.

2.7.2 Komponentengruppen und Ajax

Das `<f:ajax>`-Tag kann nicht nur innerhalb einer Komponente, sondern auch als Wrapper um mehrere Komponenten verwendet werden. Die entsprechende Ajax-Funktionalität wird dann für alle Komponenten der Gruppe aktiviert. In Listing 2.54 wurden zwei identische `<f:ajax>`-Tags verwendet, was optimiert werden kann. Listing 2.55 zeigt die überarbeitete Version, in der das `<f:ajax>`-Tag die beiden Eingabekomponenten umfasst.

Listing 2.55 Nochmals überarbeitete JSF-Seite `customer-ajax.xhtml`

```
<h:panelGrid columns="2">
  <f:ajax event="keyup" execute="firstname lastname" render="whole">
    Vorname:
    <h:inputText id="firstname"
          value="#{ajaxController.customer.firstName}" />
    Nachname:
    <h:inputText id="lastname"
          value="#{ajaxController.customer.lastName}" />
  </f:ajax>
  Kompletter Name:
  <h:outputText id="whole"
      value="#{ajaxController.customer.firstName}
             #{ajaxController.customer.lastName}" />
</h:panelGrid>
```

Eine weitere Änderung in der überarbeiteten Version ist die Verwendung des Werts `keyup` für das `<f:ajax>`-Attribut `event`. Das Attribut `event` erlaubt es, den Event-Typ, für den das Ajax-Ereignis ausgelöst wird, anzugeben und somit vom Default abzuweichen. Als Werte sind die bereits angesprochenen JavaScript-Events erlaubt, wobei der Präfix `on` jedoch entfällt. Aus dem JavaScript-Event `onkeyup` wird also der `<f:ajax>`-Typ `keyup`. Im Beispiel wird nun bei jedem KeyUp-Event, also bei jedem Tastaturanschlag, genauer beim Loslassen einer Taste ein Event gefeuert, was zur Aktualisierung des kompletten Namens führt.

Durchspielen der Beispiele

Spielen Sie alle drei Beispiele durch und vergleichen Sie die verschiedenen Ansätze.

Ajax-Request durch Mouse-Over

Ändern Sie das Beispiel in Listing 2.53 so ab, dass zum Abschicken des Formulars die Schaltfläche nicht gedrückt werden muss, sondern das Anfahren der Schaltfläche mit der Maus (Mouse-Over-Effekt) ausreicht. Das entsprechende Event ist `mouseover`.

Verschachteln von `<f:ajax>`

Da man das `<f:ajax>`-Tag innerhalb einer oder als Wrapper um mehrere andere Komponenten verwenden kann, kann man es prinzipiell auch verschachteln. Die entsprechende Ajax-Funktionalität ergibt sich dann pro JSF-Komponente additiv aus den definierten Events der umgebenden und des inneren `<f:ajax>`-Tags.

Klicksensitive Tabelle

Das `<h:panelGrid>`-Tag ist keine Eingabekomponente und hat damit auch kein Standard-Ajax-Verhalten. Es wird in HTML zu einer Tabelle (`<table>`) gerendert. Überarbeiten Sie das Listing 2.53 nochmals, so dass das Panel-Grid klicksensitiv wird, also bei einem einfachen Klick auf die entsprechende Region der komplette Name des Kunden aktualisiert wird. Das entsprechende Event ist `click`.

2.7.3 Komponentenabhängigkeiten

Unter einer Komponentenabhängigkeit wollen wir an dieser Stelle verstehen, dass die Darstellung oder Inhalte einer JSF-Komponente von einer anderen JSF-Komponente abhängen. So können etwa in einem Online-Shop abhängig von der gewählten Zahlungsmöglichkeit verschiedene Daten benötigt werden, z.B. bei Zahlung per Kreditkarte die Kreditkartennummer, bei Bankeinzug die IBAN. Es ist hier also eine gewisse Flexibilität des UI gefordert. Dies ist bereits mit den bisher bekannten Möglichkeiten des `<f:ajax>`-Tags rea-

lisierbar und wird hier nur der Übersichtlichkeit wegen in einem eigenen Abschnitt behandelt.

Als Anwendungfall wählen wir nicht verschiedene Zahlungsmöglichkeiten eines Online-Shops, sondern die Auswahl einer Sprache und – darauf basierend – die Auswahl eines Landes, in dem diese Sprache gesprochen wird. Das zugrunde liegende Konzept einer *Lokalisierung* wird ausführlich in Abschnitt 4.2 behandelt. Für die Demonstration der Abhängigkeit zweier Komponenten genügt es zu wissen, dass jedes Java-SDK eine Reihe von Lokalisierungen unterstützt, die über verschiedene ISO-Normen zwischen Sprachen und Ländern unterscheiden. Für die Sprache Deutsch gibt es u.a. die Lokalisierungen `de_DE`, `de_AT` und `de_CH`, also Deutsch für Deutschland, Österreich und die Schweiz. Das Listing 2.56 realisiert mittels `<h:selectOneMenu>` zwei Drop-down-Menüs, wobei das erste zur Auswahl der Sprache und das zweite – in Abhängigkeit der Wahl des ersten – zur Auswahl des Lands dient.

Listing 2.56 Abhängige Komponenten (`select-localization.xhtml`)

```
1  <h:panelGrid columns="2">
2    <f:facet name="header">
3      Abhängigkeiten zwischen Komponenten
4    </f:facet>
5
6    Sprache wählen:
7    <h:selectOneMenu id="language" value="#{lsc.language}">
8      <f:selectItem itemLabel="bitte auswählen" itemValue="#{null}"  />
9      <f:selectItems value="#{lsc.languages()}"  />
10     <f:ajax render="locale" />
11   </h:selectOneMenu>
12
13   Lokalisierung wählen:
14   <h:selectOneMenu id="locale" value="#{lsc.locale}">
15     <f:selectItems value="#{lsc.locales(lsc.language)}" />
16     <f:ajax render="selected" />
17   </h:selectOneMenu>
18
19   Ausgewählt wurde:
20   <h:outputText id="selected" value="#{lsc.locale}" />
21
22 </h:panelGrid>
```

Die initiale Darstellung im Browser ist in Bild 2.31 dargestellt. Wird das erste Drop-down angeklickt, werden die Sprachen zur Auswahl angezeigt. Der Methode `languages()`, die im `value`-Attribut des `<f:selectItems>`-Tags in Zeile 9 aufgerufen wird, gibt die Liste aller Sprachen zurück und ist damit für die Auswahl des Drop-down-Menüs verantwortlich. Das `<f:ajax>`-Tag in Zeile 10 reagiert ohne das `event`-Attribut auf eine Änderung der Komponente, hier also auf die Selektion durch den Benutzer. Da auch das Attribut `execute` nicht gesetzt ist, gilt hierfür der Default-Wert `@this`, es wird also die `<h:selectOneMenu>`-Komponente exekutiert. Das `render`-Attribut benennt das zweite Drop-down als die zu aktualisierende Komponente. Da die Auswahlmöglichkeiten dieses Menüs im `<f:selectItems>`-Tag in Zeile 15 über die Methode `locales()` erzeugt werden,

diese aber als Parameter die zuvor selektierte Sprache verwendet, werden die zur Sprache passenden Lokalisierungen angezeigt. Das zweite `<f:ajax>`-Tag dient lediglich zur Anzeige der Auswahl.

Abhängigkeiten zwischen Komponenten	
Sprache wählen:	bitte auswählen
Lokalisierung wählen:	zuerst Sprache wählen
Ausgewählt wurde:	

Bild 2.31 Abhängige Drop-down-Menüs (`select-localization.xhtml`)

`<f:ajax>` mit `listener`-Attribut

Das `<f:ajax>`-Tag kennt noch eine Reihe weiterer Attribute, die z.T. noch in Abschnitt 4.5 eingeführt werden. Hier soll nur das `listener`-Attribut erwähnt werden. Es entspricht dem `actionListener`-Attribut der Befehlskomponenten, die wir in Abschnitt 2.5.2 dargestellt haben. Sie können derartig registrierte Listener analog zu Action-Listenern verwenden. Der Parameter der Listener-Signatur ändert sich von `ActionEvent` auf `AjaxBehaviorEvent`.

Pick-List mit Listener

Erstellen Sie eine einfache Pick-List, etwa wie in Bild 2.32 dargestellt. Für die linke und rechte Liste können Sie `<f:selectOneListbox>`, für die Schaltflächen `<h:commandButton>` verwenden. Die Schaltflächen versehen Sie mit `<f:ajax>` und registrieren einen Listener, der die selektierte Auswahl in die andere Liste verschiebt.

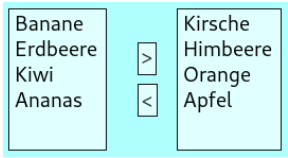

Bild 2.32 Einfache Pick-List

2.7.4 Validierung

Die Funktionalität des `<f:ajax>`-Tags fügt sich nahtlos in das Bearbeitungsmodell einer JSF-Anfrage ein. Die im `execute`-Attribut genannten Komponenten werden exekutiert, also in der Phase 3 auch konvertiert und validiert. Man sollte sich als JSF-Entwickler dieser Tatsache explizit bewusst sein, benötigt aber keine weitere Mechanismen für ihre Verwendung. Werden, wie in Listing 2.55 geschehen, mehrere Eingaben mit `<f:ajax>` versehen,

erfolgt eine weitaus häufigere Validierung als mit einem einfachen HTTP-Request. Je nach verwendetem Event kann dies zu einer deutlichen Erhöhung der Server-Last führen. In Listing 2.55 ist das Event `keyup`. Werden Vor- und Nachname des Kunden mit dem BV-Constraint `@Size(min=3)` versehen, führt die Eingabe der ersten beiden Zeichen des Vor- und Nachnamens jeweils zu Validierungsfehlern. Das Event `blur` wäre hier wahrscheinlich sinnvoller.

Stichwortverzeichnis

C

G

H

K

L

T

W

X

Z