

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

© Carl Hanser Verlag, München

Inhalt

Vorwort zur 3. Auflage	XIII
Vorwort	XV
1 Einleitung	1
1.1 Ziel dieses Buchs	1
1.2 Was sind JavaServer Faces?	2
1.3 Der Leser	4
1.4 Das Buch im Netz	5
1.5 Versionen, Versionen, Versionen	7
1.6 Spezifikationen, Implementierungen, Systeme	7
1.7 Totgesagte leben länger	8
1.8 Aufbau des Buchs	10
1.9 Classic Models	11
2 JavaServer Faces im Detail – die Grundlagen	15
2.1 Bearbeitungsmodell einer JSF-Anfrage	16
2.1.1 Wiederherstellung des Komponentenbaums	18
2.1.2 Übernahme der Anfragewerte	19
2.1.3 Validierung	20
2.1.4 Aktualisierung der Modellobjekte	21
2.1.5 Aufruf der Anwendungslogik	22
2.1.6 Rendern der Antwort	22
2.2 Expression-Language	25
2.2.1 Syntax	25
2.2.2 Werteausdrücke	26
2.2.3 Vergleiche, arithmetische und logische Ausdrücke	29
2.2.4 Methodenausdrücke	31

2.2.5	Vordefinierte Objektnamen	32
2.2.6	Collections	36
2.2.7	Lambdas	37
2.2.8	Konstanten	39
2.2.9	Komponentenbindungen	40
2.2.10	Verwendung der Expression-Language in Java	41
2.2.11	Weniger sinnvolle Verwendungen	41
2.3	Managed Beans	42
2.3.1	Architekturfragen und Namenskonventionen	43
2.3.2	Context and Dependency Injection	44
2.3.3	Architekturfragen zum Zweiten	48
2.3.4	Initialisierung	48
2.4	Validierung und Konvertierung	49
2.4.1	Standardkonvertierer	50
2.4.2	Konvertierung und Formatierung von Zahlen	53
2.4.3	Konvertierung und Formatierung von Kalenderdaten und Uhrzeiten ...	54
2.4.4	Konvertierung von Aufzählungstypen	56
2.4.5	Anwendungsdefinierte Konvertierer	60
2.4.6	Standardvalidierer	63
2.4.7	Validierungsmethoden	65
2.4.8	Anwendungsdefinierte Validierer	67
2.4.9	Eingabekomponenten und das <code>immediate</code> -Attribut	68
2.4.10	Bean-Validierung mit JSR 380	71
2.4.11	Anwendungsdefinierte Constraints mit Bean Validation	75
2.4.12	Gruppenvalidierung mit Bean Validation	77
2.4.13	Validierung auf Klassenebene	78
2.4.14	Fehlermeldungen	81
2.4.15	BV-Fehlermeldungen	89
2.5	Event-Verarbeitung	90
2.5.1	JSF-Events und allgemeine Event-Verarbeitung	91
2.5.2	Action-Events	92
2.5.3	Action-Events und Navigation	98
2.5.4	Befehlskomponenten mit Parametern	102
2.5.5	Befehlskomponenten und das <code>immediate</code> -Attribut	107
2.5.6	Value-Change-Events	108
2.5.7	Data-Model-Events	111
2.5.8	Phase-Events	115
2.5.9	System-Events	117

2.6	HTML5	120
2.6.1	Pass-Through-Attribute	121
2.6.2	Pass-Through-Elemente	122
2.7	Ajax	124
2.7.1	Das <code><f:ajax></code> -Tag.....	125
2.7.2	Komponentengruppen und Ajax.....	128
2.7.3	Komponentenabhängigkeiten	129
2.7.4	Validierung	131
3	Contexts and Dependency Injection	133
3.1	Beans, Scopes und Kontexte	135
3.1.1	Scopes	135
3.1.2	JSFs <code>@ViewScoped</code>	136
3.1.3	Kontexte	136
3.1.4	Ein genauerer Blick auf <code>@Inject</code>	140
3.1.5	Bean-Manager und programmatischer Zugriff auf Bean-Instanzen	144
3.2	Mehr Flexibilität mit Qualifiern und Alternativen.....	145
3.2.1	Qualifier	145
3.2.2	Vordefinierte Qualifier	148
3.2.3	Alternativen	149
3.2.4	Der Deskriptor <code>beans.xml</code>	151
3.3	Producer und Disposer	153
3.3.1	Producer-Methoden	153
3.3.2	Scopes	154
3.3.3	Producer-Field.....	155
3.3.4	Disposer-Methoden	155
3.4	Der Conversation-Scope.....	157
3.5	Events	161
3.5.1	Einfache Event-Producer und Observer	162
3.5.2	Spezialisierung durch Qualifier	164
3.5.3	Event-Metadaten.....	165
3.5.4	Events und Transaktionen	166
3.6	Interceptoren.....	167
3.7	Und was es sonst noch gibt.....	169
4	Weiterführende Themen	173
4.1	Templating.....	173
4.1.1	Der grundlegende Template-Mechanismus.....	173
4.1.2	Ein realistischeres Beispiel: unsere Projekte.....	175

4.1.3	Dynamische Templates	177
4.2	Internationalisierung und Lokalisierung	181
4.2.1	Lokalisierung	182
4.2.2	Dynamische und explizite Lokalisierung	187
4.2.3	Klassen als Resource-Bundles	190
4.2.4	Managed Beans und Lokalisierung	193
4.2.5	Resource-Bundles mit UTF-8-Codierung.....	194
4.2.6	Lokalisierte BV-Fehlermeldungen	194
4.3	Komponenten- und Client-Ids	196
4.3.1	Id-Arten und Namensräume	196
4.3.2	Client- und server-seitige Programmierung mit Ids.....	199
4.4	Verwendung allgemeiner Ressourcen	203
4.4.1	Einfache Ressourcen.....	203
4.4.2	Versionierte Ressourcen und Ressourcen-Bibliotheken	205
4.4.3	Positionierung von Ressourcen	206
4.4.4	Kombination von CSS- und Grafikressourcen	208
4.5	Ajax zum Zweiten	209
4.5.1	JSFs JavaScript-Bibliothek	210
4.5.2	Navigation.....	212
4.5.3	JavaScript mit Java	214
4.5.4	Nicht gerenderte Komponenten	215
4.5.5	Komponentenbibliotheken.....	217
4.5.6	Ajax ohne <f:ajax>.....	219
4.5.7	Das <h:commandScript>-Tag	223
4.5.8	Zu schnelle Benutzer ;-)	225
4.5.9	JavaScript und Expression-Language kombinieren	226
4.6	GET-Anfragen und der Flash-Scope.....	228
4.6.1	Einfache GET-Anfragen	228
4.6.2	View-Parameter	229
4.6.3	View-Actions.....	231
4.6.4	Der Flash-Scope	231
4.7	Zusammengesetzte Komponenten.....	235
4.7.1	Schnittstelle und Implementierung	236
4.7.2	Weitere Möglichkeiten.....	239
4.7.3	Packaging und Wiederverwendung	241
4.8	UI-Komponenten	242
4.8.1	Die Standardkomponenten.....	243
4.8.2	Render-Sätze.....	246

4.8.3	Die JSF-Standard-Bibliotheken	246
4.8.4	Die HTML-Bibliothek.....	247
4.8.5	Die Kernbibliothek.....	250
4.8.6	Die Templating-Bibliothek (Facelets)	252
4.8.7	Die Composite-Component-Bibliothek	252
4.8.8	Die JSTL-Kern- und Funktionsbibliothek.....	252
4.8.9	Komponentenbindungen	253
4.9	Die Servlet-Konfiguration	254
4.9.1	Der Deployment-Deskriptor.....	255
4.9.2	Übersicht Kontextparameter	257
4.9.3	Zustandsspeicherung.....	259
4.9.4	Konfigurationsdateien.....	260
4.9.5	Projektphasen	261
4.9.6	Zugriff auf Konfigurationsdaten	261
5	JavaServer Faces im Einsatz: Classic Models	265
5.1	Datenzugriff und Datenmanipulation	265
5.1.1	Java Persistence API.....	266
5.1.2	Enterprise JavaBeans	270
5.1.3	Transaktionen mit JTA	274
5.1.4	Data-Sources und Persistence-Units	275
5.2	JSF im Einsatz	276
5.2.1	Übersichten	276
5.2.2	Master-Detail-Pattern	283
5.2.3	Dynamische Drop-down-Listen	290
5.2.4	Dynamische UIs.....	292
5.3	Authentifizierung und Autorisierung	299
5.3.1	Zugriffschutz für Ressourcen	299
5.3.2	Identity Store.....	300
5.3.3	Authentifizierungs- und rollenbasierte JSF-Seiten	306
5.4	Datenexport im PDF- und Excel-Format	309
5.4.1	PDF-Erzeugung.....	309
5.4.2	Excel-Erzeugung	312
5.5	Testen.....	313
5.5.1	Arquillian	313
5.5.2	Drone und Graphene	315
5.5.3	Selenium	319
5.6	H2-Web-Konsole.....	321

6	Spezialthemen	323
6.1	Die JSF-Konfiguration	323
6.1.1	XML-Konfigurationsdatei versus Annotationen	330
6.1.2	JSF erweitern: ein eigener Exception-Handler	330
6.1.3	Programmative Konfiguration	332
6.2	Web-Sockets	333
6.2.1	Das Java-API	333
6.2.2	Globaler Server-Push	335
6.2.3	Dedizierte Nachrichten an Clients	338
6.2.4	Zeitaufwendige Berechnungen	341
6.2.5	Konfiguration	343
6.3	Resource-Library-Contracts	344
6.3.1	Globales Mapping von Contracts	345
6.3.2	Contracts auf View-Ebene	349
6.3.3	Programmative Konfiguration	350
6.4	Faces-Flows	353
6.5	Native Komponenten	358
6.5.1	Der einfache Einstieg	359
6.5.2	Komponententyp, Komponentenfamilie und Renderer-Typ	360
6.5.3	Renderer	363
6.5.4	Die Zukunft: Web-Komponenten	366
6.6	JSF als zustandsbehaftetes Komponenten-Framework	373
6.7	Mobile Endgeräte	377
6.7.1	Seiteninhalte für unterschiedliche Bildschirmgrößen und Auflösungen	377
6.7.2	Progressive Web Apps und Web-App-Manifeste	379
6.8	HTTP/2 Server-Push	384
6.9	Single-Page-Applications	386
7	Verwendete Systeme	389
7.1	WildFly und JBoss EAP	390
7.2	Payara	391
7.3	TomEE	392
7.4	WildFly Bootable JAR	393
7.5	Payara Micro	394
8	Ausblick	397
8.1	Wie geht es weiter mit JSF?	397
8.2	Andere JSF-Bücher	398

A	Die Tags der Standardbibliotheken	401
A.1	HTML-Tag-Bibliothek.....	401
A.2	Kernbibliothek	408
A.3	Templating-Bibliothek (Facelets)	412
A.4	Composite-Component-Bibliothek.....	414
A.5	JSTL-Kernbibliothek.....	416
A.6	JSTL-Funktionsbibliothek	417
A.7	Pass-Through-Attribute und -Elemente	419
B	URL-Verzeichnis	421
	Literatur	427
	Stichwortverzeichnis	429

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* [\[URL-HTML5\]](#). 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.

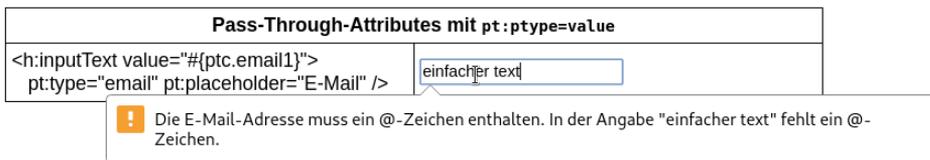


Bild 2.30 Fehlermeldung in Chrome



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 Komponentensbibliothek 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:passThroughAttribute>`. 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:passThroughAttributes>` 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 Komponenteklasse 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-Komponenteklasse der Transformation

Überzeugen Sie sich davon, dass in obigem Beispiel aus dem HTML-Element `<input>` die JSF-Komponente `<h:inputText>` und damit die Komponenteklasse `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
<code>@all</code>	Alle Komponenten der View
<code>@form</code>	Das umschließende Formular der Basiskomponente
<code>@none</code>	Keine Komponente
<code>@this</code>	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"
```

```

 9         value="#{ajaxController.customer.lastName}">
10     <f:ajax execute="@this" render="whole" />
11 </h:inputText>
12 Kompletter Name:
13 <h:outputText id="whole"
14     value="#{ajaxController.customer.firstName}
15         #{ajaxController.customer.lastName}" />
16 </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 Default-Ereignis `Value-Changed!` 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`

```

 1 <h:panelGrid columns="2">
 2     <f:ajax event="keyup" execute="firstname lastname" render="whole">
 3         Vorname:
 4         <h:inputText id="firstname"
 5             value="#{ajaxController.customer.firstName}" />
 6         Nachname:
 7         <h:inputText id="lastname"
 8             value="#{ajaxController.customer.lastName}" />
 9     </f:ajax>
10     Kompletter Name:
11     <h:outputText id="whole"
12         value="#{ajaxController.customer.firstName}
13             #{ajaxController.customer.lastName}" />
14 </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 Anwendungsfall 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-Listnern 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.

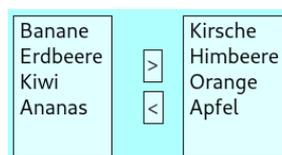


Bild 2.32 Einfache Pick-List

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 weiteren 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

A

<a> 100, 308
<absolute-ordering> 324, 325, 328
action 93
<action-listener> 325
Action-Listener-Methode 93
Action-Methode 22, 93, 231, 287
– in zusammengesetzter Komponente 238
ActionEvent 91
ActionListener 92, 96
actionListener 93, 101
ActionSource2 101
addClientBehavior() 127
addMessage() 87
@After 320
<after> 329
AFTER_COMPLETION 166
AFTER_FAILURE 166
AFTER_SUCCESS 166
afterPhase() 115
AjaxBehavior 127, 326
AjaxBehaviorEvent 91, 131
@all 126, 213
all 152
allMatch() 37
@Alternative 149
<alternatives> 150, 151
ALWAYS_PERFORM_VALIDATION_ WHEN_REQUIRED_IS_TRUE 257
Anfragewerte 19
annotated 152
Annotation 146
– Interceptor-Binding 167
– Qualifier 145
AnnotationLiteral 149
Anwendungslogik 22
@Any 148
anyMatch() 37
appendChild() 221

Applet 1
Application 41, 363
<application> 183, 324, 325, 346, 382
application
– vordefiniertes EL-Objekt 33
<application-extension> 325
<application-factory> 327
ApplicationConfigurationPopulator 332, 351
ApplicationContextFacade 100
@ApplicationMap 33
applicationScope
– vordefiniertes EL-Objekt 33
@ApplicationScoped 45, 135
@AroundConstruct 170
@AroundInvoke 168
Arquillian 313
Arquillian 314, 317, 320
arquillian.xml 315
@ArquillianResource 317, 318, 320
ArrayDataModel 114
asadmin 392
@AssertFalse 72
@AssertTrue 72, 89
Asynchron JavaScript and XML → AJAX
@Asynchronous 341
<auth-constraint> 300
authenticate() 305
AuthenticationParameters 305
AuthenticationStatus 302, 305
autocomplete 220
Automatic Dirty Checking 273
average() 37

B

<base-name> 185, 192
BASIC 301
@BasicAuthenticationMechanism-Definition 301

- Bean Validation 50
- Bean Validation 71, 74, 257
 - anwendungsdefinierte Constraints 75
- Bean-Archiv 151
- Bean-Defining- Annotations 152
- Bean-Discovery 151
- bean-discovery-mode 151, 152
- BeanManager 144
- <beans> 151, 152
- beans.xml 150, 152
 - generieren 315
- BeanValidator 63
- Bearbeitungsmodell
 - einer JSF-Anfrage 16, 230
- Befehlskomponente 19, 95, 228
- @Before 320
- <before> 329
- BEFORE_COMPLETION 166
- beforePhase() 115
- begin() 114, 157, 159
- Behavior 127, 326
- <behavior> 324, 326
- binding 40, 96, 110, 253
- BIRT 269, 302
- birt 265
- <body> 221
- BootsFaces 218, 377
- Bootstrap 377
- BrowserChecker 118
- Builder-Pattern 353
- ButterFaces 218, 377
- <button> 223

- C**
- <c:catch> 416
- <c:choose> 416
- <c:forEach> 416
- <c:if> 253, 416
- <c:otherwise> 416
- <c:set> 416
- <c:when> 416
- calculateLocale() 184
- Calendar 181
- callerQuery 302
- caption
 - Facette 113
- CascadeType 298
- Cascading Style Sheets → CSS
- cc
 - vordefiniertes EL-Objekt 33
- cc.attrs 238
- cc.clientId 238
- cc.facets 241
- <cc:actionSource> 237, 414
- <cc:attribute> 237, 238, 414
- <cc:clientBehavior> 237, 414
- <cc:editableValueHolder> 237, 414
- <cc:extension> 237, 414
- <cc:facet> 237, 240, 414
- <cc:implementation> 414
- <cc:insertChildren> 237, 239, 415
- <cc:insertFacet> 237, 240, 415
- <cc:interface> 415
- <cc:renderFacet> 237, 240, 415
- <cc:valueHolder> 237, 415
- CDI 32, 43
- CDI 142, 144
 - Klasse 142
- CDI und ServiceLoader 352
- CERT 301
- channel 336, 338
- cid 159, 161
- Classic Models ERP 10
- click() 317, 320
- client 259
- Client-Proxy 139
- Client-Window 257
- Client-Window-Id 358
- CLIENT_WINDOW_MODE 257, 358
- ClientBehavior 127
- ClientBehaviorHolder 127, 414
- @ClientEndpoint 333, 334
- clone() 80
- @Closed 344
- colgroups
 - Facette 113
- CollectionDataModel 114
- @Column 267, 268
- CompletableFuture 222, 341
- CompletionStage 341
- <component> 324, 326
- component
 - vordefiniertes EL-Objekt 33
- Component Behavior Model → Komponente, Verhaltensmodell
- Component-Binding 40
- COMPONENT_FAMILY 364
- COMPONENT_TYPE 364
- Components
 - composite 235
 - Custom 235
 - non-composite 235
- ComponentSystemEvent 117, 119, 161
- ComponentSystemEventListener 409

- CONFIG_FILES 257, 260, 328
- connectedCallback() 366
- Connection 156
- connectToServer() 335
- @Constraint 76
- ContainerProvider 335
- Context 136
- <context-param> 256, 257, 344
- contextPath 35
- Contexts and Dependency Injection → CDI
- Contextual 136
- <contract-mapping> 346
- <contracts> 346
- contracts 259, 345
- Controller
 - MVC 3
- Convention over Configuration 267
- Conversation 113, 157, 159
- Conversation-Scope 157
 - langlegig 157
 - long-running 157
 - transiente 157
- @ConversationScoped 45, 113, 136, 157, 159
- Converter<T> 60, 291
- <converter> 324
- converter 62, 63, 291
- ConverterException 61
- converterId 62
- converterMessage 62, 63, 88
- Cookie 33
- cookie
 - vordefiniertes EL-Objekt 33
- count() 37
- Create, Read, Update, Delete → CRUD
- createComponent() 363
- createResource() 382
- createTag 359
- createValueExpression() 376
- Credential 305
- Cross-Site Request Forgery → CRRF
- Cross-Site-Scripting 215
- CRUD 12, 271
- CSRF 260, 329
- CSS 199
- current() 142, 144
- Custom Data Attributes 124
- customElements 366
- @CustomFormAuthenticationMechanism-
Definition 301, 302
- D**
- Data-Source 272, 275
 - Default- 275
 - JDBC- 275
- @DatabaseIdentityStoreDefinition 301,
302
- <datalist> 220, 238, 295
- DataModel 111
 - erweitern 115
- DataModelEvent
 - DataModelEvent 91
- DataModelEvent 111
- DataModelListener 92
- DataSource 156
- @DataSourceDefinition 275
- dateStyle 56
- DateTimeConverter 50, 257, 408
- DATETIMECONVERTER_DEFAULT_TIMEZONE_
IS_SYSTEM_TIMEZONE 257
- @DecimalMax 72, 89
- @DecimalMin 72
- decode() 245
- @Decorator 152
- Default 77
- @Default 148, 164
- <default-locale> 183
- Default-Methode
 - als Getter in EL-Werteausdruck 126
- <default-render-kit-id> 325
- <default-validators> 325
- DEFAULT_SUFFIX 257
- Dependency Injection 42, 188
- @Dependent 45, 136, 139, 152
- @Deployment 314, 317
- Deployment-Deskriptor 151, 176, 255, 323
- <details> 347
- DIGEST 301
- @Digits 72
- disable 78
- DISABLE_DEFAULT_BEAN_VALIDATOR 257
- DISABLE_FACELET_JSF_VIEWHANDLER 257
- DISABLE_FACESSERVLET_TO XHTML 258
- disabled 308
- dispatchEvent() 202
- Disposed-Parameter 155
- Disposer-Methode 155
- @Disposes 155, 156
- distinct() 37
- Document 352
- Document Object Model → DOM
- DOM 126, 196
- DoubleRangeValidator 63, 410
- Drone 315
- @Drone 317, 318

during 166

E

EAP 390

Eclipse Enterprise for Java → EE4J

EditableValueHolder 414

EE4J 389

EJB 3, 12, 270

EJB-JAR 151

EL 25

<el-resolver> 325

@ElementCollection 269

ElementType 146

@Email 72, 74

EmailValidator 67

@EmailValidator 67

ENABLE_VALIDATE_WHOLE_BEAN 81, 258

ENABLE_WEBSOCKET_ENDPOINT 258, 344

encodeActionURL() 214

encodeBegin() 359, 361, 364

encodeChildren() 361–363

encodeEnd() 359, 361, 364

end() 114, 157, 159

endElement() 361, 364

@Endpoint 333

Enterprise Archive → EAR

Enterprise JavaBeans → EJB

Enterprise Resource Planning → ERP

Entity 3

@Entity 12, 267, 269

Entity-Control-Boundary-Pattern 44

EntityManager 13, 272, 276

equals() 267, 270

errorClass 86

Evaluation

– Deferred- 25

– Immediate- 25

Event

– Action- 92

– Data-Model- 111, 290

– Phase- 115, 327

– System 160

– Value-Change- 108

Event 162

event 129

Event-Listener 91, 92

– Ajax 213

– in zusammengesetzter Komponente 238

EventMetadata 166

EventObject 91

<exception-handler-factory> 327, 331

ExceptionHandlerFactory 331

ExceptionHandlerWrapper 332

ExceptionQueuedEvent 117

ExpectedConditions 320

Expression Language 21

Expression Language → EL

Expression-Language 25, 134

ExpressionFactory 41

EXTENDED 272

Extension-Mapping 256

<external-context-factory> 327

ExternalContext 33, 100, 188, 189, 214, 224,

232, 262, 339

externalContext

– vordefiniertes EL-Objekt 33, 204

ExternalContext 310

F

<f:actionListener> 95, 101, 250, 408

<f:ajax> 24, 71, 100, 108, 124, 125, 130, 250, 387, 408

– delay 225

– execute 71, 108, 125, 131

– listener 131

– render 71, 125, 130, 215

<f:attribute> 102, 106, 251, 408

<f:attributes> 107, 251, 408

<f:convertDateTime> 50, 54, 56, 58, 250, 408

<f:converter> 62, 63, 250, 408

<f:convertNumber> 50, 53, 55, 186, 250, 408

<f:event> 118, 119, 160, 250, 409

<f:facet> 47, 113, 240, 251, 282, 409

<f:importConstants> 39, 59, 60, 251, 409

<f:loadBundle> 187, 251, 409

<f:metadata> 229, 251, 281, 284, 409

<f:viewParam> 229, 281

<f:param> 102, 104–106, 229, 251, 281, 409

<f:passThroughAttribute> 121, 251, 409

<f:passThroughAttributes> 121, 122, 251, 409

<f:phaseListener> 116, 250, 409

<f:selectItem> 57, 58, 130, 251, 292, 409

<f:selectItems> 57, 59, 60, 130, 188, 193, 251, 292, 410

<f:setPropertyActionListener> 102–104, 250, 287, 410

<f:subview> 250, 410

<f:validateBean> 63, 78–80, 251, 410

<f:validateDoubleRange> 63, 64, 251, 410

<f:validateLength> 63, 64, 251, 410

<f:validateLongRange> 20, 63, 64, 251, 410

<f:validateRegex> 63, 64, 251, 410

- <f:validateRequired> 63, 64, 251, 410
 - <f:validateWholeBean> 63, 78, 80, 251, 410
 - <f:validator> 67, 251, 411
 - <f:valueChangeListener> 110, 250, 411
 - <f:verbatim> 251, 411
 - <f:view> 182–184, 187, 189, 250, 281, 349, 350, 374, 411
 - <f:viewAction> 284, 411
 - <f:viewParam> 251, 281, 411
 - <f:websocket> 251, 336, 338, 411
 - Facelets 3, 325
 - <facelets-processing> 327
 - FACELETS_BUFFER_SIZE 258
 - FACELETS_DECORATORS 258
 - FACELETS_LIBRARIES 258
 - FACELETS_REFRESH_PERIOD 258
 - FACELETS_RESOURCE_RESOLVER 258
 - FACELETS_SKIP_COMMENTS 180, 258
 - FACELETS_SUFFIX 258
 - FACELETS_VIEW_MAPPINGS 258
 - <faces-config-extension> 324, 326
 - faces-config.xml 24, 43, 84, 183, 255, 323, 346, 350, 353, 382
 - <faces-config> 330
 - <faces-context-factory> 327
 - Faces-Flows 327, 353
 - faces-redirect 100, 119, 159
 - Faces-Servlet 4, 43, 99, 160, 255
 - @FacesComponent 326, 359–361
 - FacesContext 18, 33, 41, 61, 89, 91, 188, 189, 214, 263
 - facesContext
 - vordefiniertes EL-Objekt 33, 119
 - @FacesConverter 61–63, 326
 - FacesEvent 91
 - FacesMessage 61, 85, 87, 109, 318
 - @FacesRenderer 330, 363, 364
 - @FacesValidator 67, 330
 - Facette 113, 240, 409
 - anwendungsdefiniert 240
 - <factory> 324, 327, 331
 - <factory-extension> 327
 - Familienstand 56
 - fetch() 221, 222
 - <file-extension> 327
 - filter() 37
 - finalizer() 356
 - findComponent() 97, 201, 376
 - findFirst() 37
 - fire() 162, 170
 - fireAsync() 170
 - Flash 231
 - Scope 231
 - Flash 33, 233, 234
 - flash 233
 - vordefiniertes EL-Objekt 33, 233
 - flatMap() 37
 - Flow-Call-Knoten 353
 - <flow-definition> 324, 353
 - Flow-Finalizer 356, 357
 - Flow-Initializer 356, 357
 - Flow-Scope 353, 357
 - FlowBuilder 355
 - @FlowBuilderParameter 355
 - @FlowDefinition 353, 354
 - @FlowMap 33, 357
 - flowScope 357
 - vordefiniertes EL-Objekt 33
 - @FlowScoped 355
 - flushBuffer() 310
 - fn:contains() 417
 - fn:containsIgnoreCase() 417
 - fn:endsWith() 417
 - fn:escapeXml() 417
 - fn:indexOf() 417
 - fn:join() 417
 - fn:length() 417
 - fn:replace() 417
 - fn:split() 418
 - fn:startsWith() 418
 - fn:substring() 217, 418
 - fn:substringAfter() 418
 - fn:substringBefore() 418
 - fn:toLowerCase() 217, 418
 - fn:toUpperCase() 418
 - fn:trim() 418
 - footer 240
 - Facette 113, 282
 - forClass 61
 - forEach() 37
 - FORM 301
 - @form 126, 217
 - formAssociated 369
 - format() 88
 - @FormAuthenticationMechanismDefinition 301
 - fromOutcome() 356
 - FULL_STATE_SAVING_VIEW_IDS 258
 - Future 341
 - @Future 72
 - @FutureOrPresent 72
- ## G
- Galleon 393

- @GeneratedValue 267, 268
 - getActionURL() 214
 - getApplication() 41, 87, 119, 161, 214, 262
 - getAsObject() 50, 60, 291
 - getAsString() 50, 60, 291
 - getAttributes() 263, 361, 364
 - getAvailableLocales() 194
 - getBasicRemote() 334, 335
 - getBeanManager() 144
 - getCallerPrincipal() 305–307
 - getChildren() 96, 245, 254, 376
 - getClientId() 87, 201, 361
 - getComponent() 96, 110, 119
 - getContentType() 381
 - getDataModel() 111
 - getDefaultLocale() 190
 - getDefaultRenderKitId() 246
 - getELContext() 41
 - getElementById() 196, 202, 221
 - getExceptionHandler() 331
 - getExpressionFactory() 41
 - getFamily() 360, 361
 - getFlash() 234
 - getId() 197
 - getId() 201
 - getInitParameter() 262
 - getInjectionPoint() 166
 - getInputStream() 381
 - getLocale() 194
 - getMessageBundle() 87
 - getMethod() 168
 - getNavigationHandler() 119, 161
 - getNewValue() 109
 - getOldValue() 109
 - getParameters() 168
 - getParent() 96, 376
 - getPartialResponseWriter() 214
 - getPartialViewContext() 214
 - getPhaseId() 115
 - getProjectStage() 211, 261, 262
 - getRendererType() 360
 - getRenderKitId() 246
 - getRendersChildren() 363
 - getRequestLocale() 189
 - getRequestParameterMap() 224
 - getRequestPath() 381, 382
 - getResourceBundle() 193
 - getRowData() 111, 112
 - getRowindex() 111
 - getSession() 305
 - getStateManager() 262
 - getSupportedLocale() 189
 - getTarget() 168
 - getTimeout() 159
 - getUnhandledExceptionQueuedEvents() 332
 - getUserPrincipal() 307, 308
 - getValidators() 87
 - getViewHandler() 214
 - getViewId() 357
 - getWebSocketContainer() 335
 - getWrapped() 332, 381, 382
 - GlassFish
 - Referenzimplementierung 8
 - globalOnly 83, 85
 - Graphene 315, 320
 - GregorianCalendar 181
 - groups 77
 - groupsQuery 303
 - guardHttp() 317, 318
 - GUI 3, 63
 - Guice 133
- ## H
- <h:body> 206, 221, 249, 402
 - <h:button> 249, 281, 282, 363, 402
 - <h:column> 14, 36, 249, 402
 - <h:commandButton> 22, 101, 249, 363, 402
 - <h:commandLink> 100, 101, 103, 249, 363, 402
 - <h:commandScript> 101, 209, 223, 249, 402
 - <h:dataTable> 14, 36, 249, 278, 279, 403
 - <h:doctype> 249, 403
 - <h:form> 206, 248, 403
 - <h:graphicImage> 103, 187, 203, 210, 213, 249, 403
 - library 204
 - name 204
 - value 204
 - <h:head> 206, 221, 249, 403
 - <h:inputFile> 403
 - <h:inputHidden> 248, 404
 - <h:inputSecret> 80, 248, 404
 - <h:inputText> 62, 64, 210, 248, 404
 - <h:inputTextarea> 62, 95, 248, 404
 - <h:link> 249, 283, 363, 404
 - <h:message> 62, 64, 83, 249, 404
 - <h:messages> 83, 85, 249, 405
 - automatisch einfügen 261
 - <h:outputFormat> 106, 249, 405
 - <h:outputLabel> 249, 405
 - <h:outputLink> 100, 249, 405
 - <h:outputScript> 202, 203, 206, 207, 210, 211, 219, 220, 223, 247, 249, 372, 405

- <h:outputStylesheet> 203, 206, 247, 249, 372, 405
 - <h:outputText> 210, 249, 405
 - <h:panelGrid> 28, 47, 129, 249, 406
 - <h:panelGroup> 47, 74, 83, 119, 217, 249, 406
 - <h:selectBooleanCheckbox> 57, 202, 248, 375, 406
 - <h:selectManyCheckbox> 58, 248, 406
 - <h:selectManyListbox> 58, 248, 406
 - <h:selectManyMenu> 58, 248, 407
 - <h:selectOneListbox> 57, 131, 248, 407
 - <h:selectOneMenu> 56–60, 62, 71, 130, 187, 248, 290, 407
 - <h:selectOneRadio> 57, 193, 248, 407
 - handle() 332
 - handleNavigation() 119, 161, 332
 - hashCode() 270
 - <head> 221
 - header 240
 - Facette 113
 - vordefiniertes EL-Objekt 33, 119
 - @HeaderMap 33, 119
 - headerValues
 - vordefiniertes EL-Objekt 33
 - @HeaderValuesMap 33
 - hidden 348
 - href 100
 - HTML-Bibliothek 247, 308
 - HTML5 120
 - html5
 - JSF-Unterstützung 120
 - HtmlBody 402
 - HtmlColumn 402
 - HtmlCommandButton 101, 247, 402
 - HtmlCommandLink 101, 247, 363, 402
 - HtmlCommandScript 101, 402
 - HtmlDataTable 403
 - HtmlDoctype 403
 - HTMLElement 366
 - HtmlForm 197, 403
 - HtmlGraphicImage 403
 - HtmlHead 403
 - HtmlInputFile 403
 - HtmlInputHidden 404
 - HtmlInputSecret 404
 - HtmlInputText 123, 360, 376, 404
 - HtmlInputTextarea 96, 404
 - HtmlMessage 404
 - HtmlMessages 405
 - HtmlCommandButton 363
 - HtmlOutcomeTargetButton 230, 363, 402
 - HtmlOutcomeTargetLink 230, 363, 404
 - HtmlOutputFormat 405
 - HtmlOutputLabel 405
 - HtmlOutputLink 405
 - HtmlOutputText 40, 405
 - HtmlPanelGrid 406
 - HtmlPanelGroup 406
 - HtmlSelectBooleanCheckbox 406
 - HtmlSelectManyCheckbox 406
 - HtmlSelectManyListbox 406
 - HtmlSelectManyMenu 247, 407
 - HtmlSelectOneListbox 407
 - HtmlSelectOneMenu 247, 254, 407
 - HtmlSelectOneRadio 407
 - HTMLULElement 370, 372
 - HTTP 45, 157
 - -Protokoll 16
 - -Request-Header 32
 - Accept-Language-Header 183
 - Zeichensatz für 182
 - HTTP-Status-Code 320 100
 - HttpServletRequest 33, 305
 - HttpServletResponse 305
 - HttpSession 170, 260, 305
- ## I
- ICEfaces 218
 - Id
 - explizite 198, 199
 - implizite 198, 199
 - @Id 12, 267–269
 - IEEE 754-1895 53
 - immediate
 - Befehlskomponente 107
 - immediate 19, 108
 - Eingabekomponente 68, 71
 - import.sql 270
 - IN_PROGRESS 166
 - includeViewParams 231
 - infoClass 86
 - Initialisierungsparameter → Kontextparameter
 - @Initialized 169
 - initializer() 356
 - initParam 261, 262
 - vordefiniertes EL-Objekt 33
 - @InitParameterMap 33, 262, 263
 - @Inject 35, 49, 135, 277, 352
 - Injection-Point 140
 - InjectionPoint 144
 - innerHTML 221, 366
 - <input> 121, 123
 - Instance 149
 - Interceptor 167

- @Interceptor [152, 168](#)
 - @InterceptorBinding [167](#)
 - <interceptors> [169](#)
 - Internationalisierung [181](#)
 - INTERPRET_EMPTY_STRING_SUBMITTED_VALUES_AS_NULL [65, 96, 201, 258](#)
 - Inversion of Control → IoC
 - InvocationContext [168](#)
 - IoC [42](#)
 - is [372](#)
 - isAjaxRequest() [214](#)
 - ISO
 - 3166 [182](#)
 - 4217 [54](#)
 - 8859 [182](#)
 - 639 [182](#)
 - Country Codes [182](#)
 - Language Codes [182](#)
 - isSavingStateInClient() [262](#)
 - isTransient() [159](#)
 - isUserInRole() [307](#)
 - isValid() [76](#)
 - itemLabel [58](#)
 - itemValue [58](#)
 - IterableDataModel [114](#)
 - iterator() [37](#)
- J**
- j_password [299](#)
 - j_username [299](#)
 - JAR [151](#)
 - JASPI [305, 309](#)
 - Java Archive → JAR
 - Java Authentication and Authorization Service → JAAS
 - Java Authentication SPI for Containers → JASPI
 - Java Community Process → JCP
 - Java Naming and Directory Interface → JNDI
 - Java Persistence API → JPA
 - Java Persistence Query Language → JPQL
 - Java Specification Request → JSR
 - Java Transaction API → JTA
 - java.lang.annotation
 - Package [146](#)
 - java.security
 - Package [306](#)
 - java.security.acl
 - Package [309](#)
 - java.text
 - SimpleDateFormat [54](#)
 - java.text.DecimalFormat [53](#)
 - java.util
 - Collection [37](#)
 - Locale [56](#)
 - Package [181](#)
 - java:comp/DefaultDataSource [275](#)
 - JavaScript [199](#)
 - JavaServer Faces → JSF
 - JavaServer Pages [3](#), → JSP
 - JavaServer Pages Standard Tag Library → JSTL
 - javax.annotation
 - Package [150, 156, 163, 188](#)
 - javax.annotation.sql
 - Package [275](#)
 - javax.ejb
 - Package [271](#)
 - javax.enterprise
 - Package [133](#)
 - javax.enterprise.context
 - Package [46, 157, 159, 170](#)
 - javax.enterprise.context.spi
 - Package [136](#)
 - javax.enterprise.event
 - Package [162](#)
 - javax.enterprise.inject
 - Package [170](#)
 - javax.enterprise.inject
 - Package [148, 152](#)
 - javax.enterprise.inject.spi
 - Package [142, 144, 166](#)
 - javax.faces
 - Ressourcen-Bibliothek [208, 210](#)
 - javax.faces.application
 - Package [332, 351, 380](#)
 - javax.faces.bean
 - Package [46](#)
 - javax.faces.behavior.event [224](#)
 - javax.faces.ClientWindow [358](#)
 - javax.faces.component
 - Package [101, 243](#)
 - javax.faces.component.html
 - Package [248](#)
 - javax.faces.context
 - Package [234](#)
 - javax.faces.convert
 - Package [50, 60](#)
 - javax.faces.event
 - Package [91, 110, 115](#)
 - javax.faces.Messages
 - Resource-Bundle [82](#)
 - javax.faces.model
 - Package [92, 111](#)
 - javax.faces.partial.ajax [224](#)
 - javax.faces.render

- Package 246
- javax.faces.validator
 - Package 63, 66
- javax.faces.view
 - Package 46
- javax.faces.view.facelets
 - Package 123
- javax.faces.ViewState 18
- javax.faces.ViewState 224, 329, 376
- javax.inject
 - Package 133, 148
- javax.persist
 - Package 267
- javax.security.enterprise.identity-store
 - Package 303
- javax.servlet.http
 - Package 170
- javax.sql
 - Package 156
- javax.transaction
 - Package 274
- javax.validation.constraints
 - Package 71
- javax.validation.groups
 - Package 77
- javax.websocket
 - Package 333
- javax.websocket.server
 - Package 333
- JAX-RS 226
- JBoss Seam 326
- jboss-cli.sh 390
- jboss-deployment-structure.xml 322
- jboss-web.xml 305
- JCP 7, 254
- JDK Enhancement Proposals → JEP
- JEP
 - 226 194
- jfwid 358
- JNDI 144, 156, 275
- @JoinColumn 269
- JPA 3, 12, 166, 266
 - automatic Dirty-Checking 108
- JPQL 12, 267
- JSF-Konfiguration 323
- jsf.ajax.request() 210, 223
- jsf.js
 - JavaScript-Ressource-Name 210
- jsf.push.close() 343, 344
- jsf.push.open() 343, 344
- <jsf:element> 123

- JSONArray 227
- JSP 3, 25
- JSP Standard Tag Library → JSTL
- JSTL 25, 217
 - Funktionsbibliothek 247
 - Kernbibliothek 247
- <jta-data-source> 276
- JVM 1, 20

K

- keep 234
- keep() 234
- Kernbibliothek 247
- keyup 129
- Komponente 242
 - HTML-Standard- 247
 - native 326, 358
 - Verhaltensmodell einer 127
 - zusammengesetzte 235, 326
- Komponentenbaum 18, 176, 281
 - Wiederherstellung 18
- Komponentenbindung 40, 123
- Komponentenfamilie 247, 359, 360, 364
- Komponententyp 359, 360
- Kontextparameter 32, 33, 74, 198, 257
 - Mojarra-spezifisch 259
 - MyFaces-spezifisch 259
- Konversation → Conversation-Scope
- Konvertierer
 - für JPA-Entitäts 291
 - für Standarddatentypen 50
- Konvertierung 49

L

- layout 85
- Lazy-Loading-Pattern 280
- Lebenszyklus 16
 - beenden 309
 - mit immediate 68, 107
- LengthValidator 63, 87, 410
- Lesezeichen (Bookmark) 230
- <lifecycle> 116, 324
- <lifecycle-factory> 327
- LIFECYCLE_ID 258
- limit() 37
- <link> 35, 207, 247, 381
- ListDataModel 114
- Listener
 - Action- 92
- @ListenerFor 120
- @ListenersFor 120
- ListResourceBundle 190

LocalDateTime 55
 Locale 181, 182
 locale 56, 183, 187
 <locale-config> 183, 325
 LocalTime 55
 login() 305
 @LoginToContinue 302
 logout() 305
 Lokalisierung 182
 LongRangeValidator 63, 410

M

managed 63
 Managed Bean 18, 26, 42, 255
 – und Lokalisierung 193
 <managed-bean> 324, 327
 @ManagedBean 43, 46, 327
 – Common Annotation 188
 – Migration 48
 @ManagedProperty 49, 233
 – Migration 48
 @ManyToMany 269
 @ManyToOne 269
 map() 37, 38
 markAsStartNode() 355
 Master-Detail-Pattern 283
 @Max 72, 74
 max() 37
 <message-bundle> 325
 MessageFormat 83, 88
 /META-INF 242
 /META-INF/contracts 345
 /META-INF/resources 203, 345
 /META-INF/services 351
 metadata-complete 330
 Method Expression → Methodenausdruck
 Method-Call-Knoten 353
 Methodenausdruck 26, 356
 @Min 72, 74
 min() 37
 Model
 – MVC 3
 @Model 170
 Model View Controller → MVC
 Mojarra 8, 55, 344, 399
 mojarra.ab() 223
 @Monitored 167
 mouseover 129
 MVC 43, 44
 MyFaces 8, 344

N

<name> 324, 328
 @Named 13, 21, 27, 45, 47, 138, 170, 277
 – vordefinierter Qualifier 148
 @NamedEvent 120
 @NamedQuery 12, 267, 269
 namespace 361
 NamingContainer 197
 Navigation
 – -Handler 98
 – explizite 101, 329
 – implizite 98, 101, 329
 – lokalisierte 194
 <navigation-handler> 325
 <navigation-rule> 324, 329
 @Negative 72
 @NegativeOrZero 72
 @none 126
 none 152
 noneMatch() 37
 @NormalScope 139
 noSelectionOption 292
 @NotBlank 72
 @NotEmpty 72
 notifyObserver 170
 @NotNull 72, 74, 257, 267
 @Null 72
 NumberConverter 50, 408

O

Object Relational Mapper → OR-Mapper
 Objektname
 – vordefinierter 32, 35, 138, 261
 Observer 161
 – Methode 161, 162
 – Methode (transaktional) 166
 – Pattern 161
 – Resolution 165
 @Observes 161, 163, 337, 344
 @ObservesAsync 170
 OmniFaces 218, 384, 399
 onChange 69, 128
 onclick 202, 287, 343
 @OnClose 334
 @OnError 334
 @OneToMany 267, 268
 @OneToOne 269
 onfocus 201
 onkeyup 128
 @OnMessage 334
 onmessage> 336
 onmouseout 200

- onmouseover 200
- @OnOpen 334
- OpenAjax-Alliance 210
- @Opened 155, 344
- OpenWebBeans 134
- <option> 295
- OR-Mapper 267
- <ordering> 324, 328, 329
- org.openqa.selenium
 - Package 320
- ORM 267
- <other> 329
- outcome 228, 229
- P**
- param
 - vordefiniertes EL-Objekt 33
 - <param-name> 256, 257, 344
 - <param-value> 256, 257, 344
- paramValues
 - vordefiniertes EL-Objekt 33
- Partial-State-Saving 365
- <partial-view-context-factory> 327
- Partial-View-Processing 18
- Partial-View-Rendering 18
- PARTIAL_STATE_SAVING 258, 376
- Pass-Through
 - XML-Namensraum 121
- Pass-Through-Attribut 121, 220, 236
- Pass-Through-Element 121, 122, 236
- @Past 72, 74
- @PastOrPresent 72
- @Path 222
- @PathParam 334
- @Pattern 72
- pattern 56, 186
- Pbkdf2PasswordHash 303
- PDF 23
- peek() 37
- <persistence> 276
- Persistence-Unit 276
- <persistence-unit> 276
- persistence.xml 276, 315
- @PersistenceContext 13, 272
- PersistenceContextType 272
- Phase
 - Execute 17, 125
 - Render 17, 125
- Phase-Event
 - Verwendungsbeispiel 116
- Phase-Listener 24
- <phase-listener> 116, 327
- PhaseEvent 115
- PhaseId 115
 - ANY_PHASE 115
 - APPLY_REQUEST_VALUES 115
 - INVOKE_APPLICATION 115
 - PROCESS_VALIDATIONS 115
 - RENDER_RESPONSE 115
 - RESTORE_VIEW 115
 - UPDATE_MODEL_VALUES 115
- PhaseListener 92, 115, 409
- Plain Old Java Object → POJO
- POJO 3, 42, 135, 162, 270
- POM 317, 393
- populateApplicationConfiguration()
 - 351
- Portable-Extension 136, 144
- @Positive 72, 74
- @PositiveOrZero 72
- Post-Back 16, 138, 329, 373
- Post-Redirect-Get-Pattern 100, 228, 231
- PostAddToViewEvent 117
- @PostConstruct 138, 188, 189, 277
- PostConstructApplicationEvent 117
- PostConstructCustomScopeEvent 117
- PostConstructViewMapEvent 117
- PostKeepFlashValueEvent 117
- PostPutFlashValueEvent 117
- PostRenderViewEvent 117
- PostRestoreStateEvent 117
- PostValidateEvent 117
- PreClearFlashEvent 117
- @PreDestroy 188
- PreDestroyApplicationEvent 117
- PreDestroyCustomScopeEvent 117
- PreDestroyViewMapEvent 117
- prependId 202
- PreRemoveFlashValueEvent 117
- PreRemoveFromViewEvent 117
- PreRenderComponentEvent 117, 118
- preRenderView 160
- PreRenderViewEvent 91, 117, 118, 160
- PreValidateEvent 117
- PrimeFaces 218
 - Mobile 377
- Principal 306
- @Priority 150, 163, 169, 170
- proceed() 168
- <process-as> 327
- processAction() 96
- processValueChange() 108, 110
- Producer-Methode 143, 153
- @Produces 153, 155, 156, 353, 354

<progress> 124
 Progressive Web App → PWA
 PROJECT_STAGE 258, 261
 ProjectStage 261
 Promise 222
 Properties 184
 <properties> 276
 <property> 276
 <property-resolver> 325
 PropertyResourceBundle 184
 <protected-views> 324, 329, 377
 Präfix-Mapping 256
 @Push 246, 336, 337, 340, 342
 PushContext 246, 336, 337
 PushContext 337, 340, 342
 PWA 379, 384, 386

Q

Qualifier 49
 – bei Producer-Methode 154
 – Default 148
 – vordefinierte 148
 @Qualifier 147
 @QueryParam 222

R

Redirect 100, 214, 231, 285
 redirect() 214
 reduce() 37
 <referenced-bean> 324, 329
 RegexValidator 63, 410
 Relocation 203
 Remote Method Invocation → RMI
 Render-Kit 246, 330, 363
 <render-kit> 324
 <render-kit-factory> 327
 Render-Satz 246
 rendered 89, 216, 281, 308
 Renderer 120, 246, 363
 Renderer-Typ 247, 360, 364
 RENDERER_TYPE 364
 RenderKit 246
 RenderKitFactory 246
 renderResponse() 70, 91
 request 35
 – vordefiniertes EL-Objekt 33, 204
 @RequestCookieMap 33
 @RequestMap 33
 @RequestParamMap 33
 @RequestParamValuesMap 33
 requestScope
 – vordefiniertes EL-Objekt 33

@RequestScoped 45, 136, 170, 274, 277
 required 64, 284
 requiredMessage 88
 RequiredValidator 63, 64, 410
 Resource 380
 @Resource 156
 resource 35
 – vordefiniertes EL-Objekt 33, 204
 Resource-Bundle 184
 <resource-bundle> 185, 192, 193, 325
 <resource-handler> 325, 382
 Resource-Library-Contracts 203
 <resource-library-contracts> 325, 346
 RESOURCE_CONTRACT_XML 345
 ResourceBundle 87, 181, 184, 193
 @ResourceDependencies 208
 @ResourceDependency 208, 372
 ResourceHandler 203, 204, 380
 ResourceHandlerWrapper 380, 382
 /resources 203, 259, 345
 ResourceWrapper 380, 381
 @Resource 337
 responseComplete() 91, 214, 310
 ResponseStateManager 246
 ResponseWriter 361
 Ressource 203
 Ressourcen-Identifikator 205
 Ressourcenverzeichnis 203, 238
 restoreState() 365
 ResultDataModel 114
 ResultSetDataModel 114
 @Retention 146
 RetentionPolicy 146
 Return-Knoten 353, 356, 357
 returnNode() 356
 <role-name> 300
 @RolesAllowed 274
 @RunAsClient 317, 318
 @RunWith 314, 317, 320

S

saveState() 365
 ScalarDataModel 114
 Scope 135
 – Conversation 157
 – Custom 46, 136, 139
 – Default 136
 – Dependent 136, 154, 156, 312
 – normal 139, 152
 – passivierbarer 140, 159, 355
 – Pseudo 139
 <script> 207, 220

- <search-expression-handler> 325
 - <search-keyword-resolver> 325
 - SearchKeywordResolver 126
 - <security-constraint> 300
 - Security-Context 305
 - <security-role> 300
 - SecurityContext 305
 - Select 317, 320
 - @Select 318
 - select() 142
 - SelectItem 58, 59
 - Selenium 318, 319
 - send() 337
 - SEPARATOR_CHAR 198, 258
 - Serializable
 - Interface 46
 - SERIALIZE_SERVER_STATE 259
 - server 259
 - Server-Push 384
 - @ServerEndpoint 334
 - ServiceLoader 332, 351
 - Servlet 3
 - Faces 17
 - <servlet> 256
 - <servlet-class> 256
 - Servlet-Konfiguration 99, 254
 - <servlet-mapping> 256
 - <servlet-name> 256
 - ServletContext 100
 - ServletContextImpl 100
 - session
 - vordefiniertes EL-Objekt 33
 - @SessionMap 33
 - sessionScope
 - vordefiniertes EL-Objekt 33
 - @SessionScoped 45, 135
 - setId() 197
 - setKeepMessages() 233, 234
 - setParameters() 168
 - setRendererType() 363
 - setResponseContentType() 214
 - setStyle() 201
 - setTimeout() 159
 - setValueExpression() 376
 - SFSB 270
 - showDetail 85
 - showSummary 85
 - ShrinkWrap 314, 317
 - Simple Object Access Protocol → SOAP
 - Single Page Application → SPA
 - @Singleton 270
 - @Size 72, 74, 89, 195
 - SLSB 270
 - sorted() 37, 38
 - SPA 386
 - Spring-Framework 133
 - standalone.xml 390, 391
 - Standardkonvertierer 50
 - startElement() 361, 364
 - @Startup 171
 - <state-manager> 325
 - STATE_SAVING_METHOD 32, 259
 - @Stateful 270, 273
 - Stateful Session-Beans 270
 - StateHolder 365
 - @Stateless 13, 270, 341
 - Stateless Session-Bean 270
 - Stereotyp 170
 - @Stereotype 152
 - stream() 38
 - <style> 176, 247
 - substream() 37
 - sum() 37
 - <summary> 347
 - <supported-locale> 183
 - SVG 23
 - Switch-Knoten 353
 - <system-event-class> 118
 - <system-event-listener> 118, 325
 - <system-event-listener-class> 118
 - SystemEvent 91, 117
 - SystemEventListener 118
- ## T
- @Table 267, 269
 - <tag-handler-delegate-factory> 327
 - TagDecorator 123
 - tagName 359
 - @Target 146
 - target 220
 - Template 173
 - -Client 173, 282
 - Templating-Bibliothek 247
 - @Test 314, 317, 320
 - then() 222
 - @this 126, 128, 130
 - TimerService 337
 - timeStyle 56
 - timeZone 56
 - <title> 176
 - toArray() 37
 - toList() 37, 38
 - TRANSACTION 272
 - @Transactional 274

TransactionPhase 166
 Transaktion 166
 transient 374
 Tree 371
 TreeNode 370
 type 56, 96, 110, 118

U

<ui:component> 412
 <ui:composition> 173, 412
 – template 174, 176, 179
 <ui:debug> 180, 412
 <ui:decorate> 347, 412
 <ui:define> 173, 174, 284, 412
 – name 179
 <ui:fragment> 308, 412
 <ui:include> 175, 176, 387, 412
 – src 178, 179
 <ui:insert> 173, 175, 413
 – name 179
 <ui:param> 347, 413
 <ui:remove> 55, 180, 413
 <ui:repeat> 36, 226, 413
 – varStatus 226, 227
 UIColumn 243
 UICommand 101, 243
 UIComponent 3, 61, 91, 120, 197, 201, 243, 245,
 361
 UIComponentBase 127, 243, 245, 361, 365
 UIData 197, 243
 UIForm 197, 243, 245
 UIGraphic 244
 UIImportConstants 244, 245
 UIInput 229, 244
 UIMessage 244
 UIMessages 244
 UINamingContainer 197, 244
 UIOutcomeTarget 230, 244
 UIOutput 244, 405
 UIPanel 123
 UIParameter 245
 UISelectBooleann 245
 UISelectItem 245, 409
 UISelectItems 245, 410
 UISelectMany 245
 UISelectOne 245
 UIViewAction 245
 UIViewParameter 229, 245, 411
 UIViewRoot 33, 35, 119, 194, 201, 213, 245,
 281, 409, 411
 uiViewRoot 194
 UIWebsocket 245

Unified EL 25
 URI 18
 <url-pattern> 256, 300, 329, 346
 user 338
 userAgentNeedsUpdate() 381
 UsernamePasswordCredential 305
 UTF-8 182

V

validate() 66
 VALIDATE_EMPTY_FIELDS 75, 259
 validatedValue 196
 Validation
 – class-level 79
 – cross-field 79
 – multi-field 79
 validationGroups 78
 <validator> 324
 validator 66
 Validator<T> 63, 66, 67
 ValidatorException 64, 66
 validatorMessage 66, 88
 Validierung 20, 49
 – auf Klassenebene 79
 – mehrere Eingaben 79
 Value Expression 26
 Value-Change-Event 21
 Value-Change-Listener 108, 110, 374
 ValueChangeEvent 70
 ValueChangeEvent 91
 ValueChangeListener 92, 108, 110
 valueChangeListener 109
 ValueHolder 415
 <var> 185, 192
 <variable-resolver> 325
 VDL 3, 242, 252
 @Vetoed 152
 View 18, 46
 – -Action 228
 – -Parameter 19, 228, 229, 281
 – <UIViewRoot> 201
 – MVC 3
 – zustandslose 374, 375, 377, 397
 view
 – vordefiniertes EL-Objekt 33, 99, 119
 View Declaration Language 3, → VDL
 View-Action 230
 <view-declaration-language-factory>
 327
 <view-handler> 325
 View-Id 18, 98, 355
 View-Knoten 353, 355

View-Scope 46, 292, 386
ViewHandler 184, 325
@ViewMap 33
viewNode() 355
VIEWROOT_PHASE_LISTENER_QUEUES_
EXCEPTIONS 259
viewScope
– vordefiniertes EL-Objekt 33
@ViewScoped 46, 109, 136, 139, 152
<visit-context-factory> 327
@Volljaehrig 76

W

WAR 151
Web Components 366
Web Service Description Language → WSDL
<web-app> 256
Web-App-Manifest 379
Web-Components 359
WEB-INF-Verzeichnis
– CDI 151
– JSF 323
– Servlet 255
Web-Profil 2
<web-resource-collection> 300
<web-resource-name> 300
Web-Socket 258
web.xml 32, 74, 176, 198, 255, 321
WEBAPP_CONTRACTS_DIRECTORY 259, 345
WEBAPP_RESOURCES_DIRECTORY 259
WebArchive 314, 317
WebDriver 317, 318
WebDriverWait 320
WebElement 317, 320
@WebElement 318
WebServlet 321

WEBSOCKET_ENDPOINT_PORT 259, 344
WebSocketContainer 335
WebSocketEvent 91, 92, 344
Weld 134
Werteausdruck 21, 26, 40, 336
WildFly 390
window 366
WML 3, 23
writeAttribute() 361, 364

X

XML-Namensraum
– http://xmlns.jcp.org/jsf/composite
236, 238, 252, 414
– http://xmlns.jcp.org/jsf/core 408
– http://xmlns.jcp.org/jsf/facelets
252, 412
– http://xmlns.jcp.org/jsf/f 20, 250
– http://xmlns.jcp.org/jsf/html 20,
247, 401
– http://xmlns.jcp.org/jsf/jsf 123
– http://xmlns.jcp.org/jsf/passthrough
121, 419
– http://xmlns.jcp.org/jsf 419
– http://xmlns.jcp.org/jsp/jstl/core
252, 416
– http://xmlns.jcp.org/jsp/jstl/functions
252, 417
XMLHTTPREQUEST 125
XMLHttpRequest 209, 222
XPath 25

Z

Zustandsspeicherung 365
– partielle 258, 376