

Stephan Niedermeier

Cocoon 2 und Tomcat

Inkl.
Cocoon
2.2

- Umfassende Einführung in Cocoon, Tomcat, XML, XSL und XSL-FO
- Produktiver Einsatz des XML-Publishing-Systems
- Nachschlagewerk für Anwender und Entwickler

2. Auflage

Galileo Computing

Inkl. Cocoon 2.1
und Tomcat 5

Cocoon 2
und Tomcat

Niedermeier

656



Auf einen Blick

	Vorwort	23
1	Einführung in Cocoon	27
2	XML – Eine Einführung	43
3	XSL	73
4	Programmieren mit XML	99
5	Tomcat	121
6	Cocoon installieren	189
7	Die Sitemap	197
8	Module	271
9	Cocoon erweitern	277
10	Control Flow	369
11	Cocoon Forms	411
12	XSP	515
13	Datei-Upload	529
14	Datenbankzugriffe mit Cocoon	537
15	Internationalisierung	581
16	Cocoon im Einsatz: Das Portal	603
17	Cocoon offline betreiben	639
A	Tomcat	645
B	Actions	661
C	Generatoren	667
D	Matcher	695
E	Reader	697
F	Selectoren	701
G	Serializer	705
H	Transformer	719
I	FOM	747
J	XSP	773
K	Input-Module	785
L	Glossar	803
M	Wichtige und interessante Quellen	807
N	Inhalt der CD-ROM	813
	Index	815

Inhalt

Vorwort	23
---------------	----

1 Einführung in Cocoon 27

1.1	Geschichte von Cocoon	27
1.2	Anforderungen an Web-Applikationen	29
1.2.1	Trennung von Layout, Inhalt und Logik	29
1.2.2	Plattformunabhängigkeit	31
1.2.3	Personalisierung	31
1.2.4	Modularität und einfache Erweiterbarkeit	31
1.2.5	Internationalisierung	32
1.2.6	Skalierbarkeit	32
1.2.7	Multichannel Publishing	32
1.2.8	Verschiedene Datenquellen	34
1.3	Ein kleiner Einblick in Cocoon	35
1.3.1	Cocoon ist Middleware	36
1.3.2	Umgebungen	37
1.3.3	Request-Response-Zyklus	39
1.4	Die Rolle von XML in Cocoon	40
1.5	Zusammenfassung	42

2 XML – Eine Einführung 43

2.1	Am Anfang stand SGML	44
2.2	XML wird beschlossen	44
2.3	Das XML-Dokument	45
2.3.1	XML-Deklaration	45
2.3.2	XML-Name	47
2.3.3	Tags und Elemente	47
2.3.4	Attribute	49
2.3.5	Wohlgeformtheit und Gültigkeit	49
2.3.6	Knoten und ihre Beziehungen zueinander	53
2.3.7	Entity-Referenzen	54
2.3.8	CDATA	55
2.3.9	Kommentare	56
2.4	Namensräume	57
2.4.1	Default-Namensräume	59

Inhalt

2.4.2	Mehrere Namensräume	59
2.4.3	Beispiel	60
2.5	Von HTML zu XHTML	62
2.6	Die Document Type Definition (DTD)	64
2.6.1	Elemente	67
2.6.2	Attribute	68
2.6.3	Entities	69
2.6.4	DTD einbinden	70
3	XSL	73
3.1	XPath	74
3.1.1	Knotentypen	75
3.1.2	Lokalisierungspfad	75
3.1.3	Prädikate	77
3.1.4	XPath-Funktionen	78
3.1.5	XPath und Cocoon	79
3.2	XSLT	79
3.2.1	Das XSLT-Stylesheet	80
3.2.2	Der Kontextknoten	83
3.2.3	Wichtige XSLT-Instruktionen	83
3.2.4	XSL-Parameter und -Variablen	85
3.3	XSL-FO	88
3.3.1	Der Aufbau eines XSL-FO Dokuments	89
3.3.2	Der FO-Prozessor	93
3.3.3	XSL-FO und XSLT-Stylesheet	95
4	Programmieren mit XML	99
4.1	Die geeignete Entwicklungsumgebung	99
4.2	DOM	101
4.2.1	Die Vorteile von DOM	101
4.2.2	Die Nachteile von DOM	101
4.2.3	Der DOM-Parser für Java	102
4.2.4	Installieren und verwenden von Xerces	103
4.2.5	Ein erstes DOM-Programm	103
4.3	SAX	112
4.3.1	Die Vorteile von SAX	113
4.3.2	Die Nachteile von SAX	113
4.3.3	Der SAX-Parser für Java	113

4.3.4	So arbeitet SAX	113
4.3.5	Ein erstes SAX-Programm	115
4.4	SAX und DOM in Cocoon	119

5 Tomcat 121

5.1	Geschichte von Tomcat	121
5.2	Tomcat besorgen und installieren	123
5.2.1	Ein erster Testlauf	125
5.3	Die Verzeichnis-Struktur	126
5.4	Servlets	127
5.5	JavaServer Pages	133
5.6	Die Architektur	136
5.6.1	Die Konfigurationsdatei server.xml	138
5.6.2	Das Admin-Tool	142
5.7	Die Web-Applikation	144
5.7.1	Erzeugen einer War-Datei	147
5.8	Der Deployment Descriptor web.xml	149
5.8.1	Generelle Informationen	153
5.8.2	Verteilte Anwendung	153
5.8.3	Konfiguration mit Parametern	154
5.8.4	Filter	155
5.8.5	Listener	160
5.8.6	Registrieren von Servlets und JSPs	162
5.8.7	Session-Konfiguration	165
5.8.8	Registrieren von Mime-Types	165
5.8.9	Willkommensdateien	166
5.8.10	Fehlerseiten bestimmen	167
5.8.11	Tag-Library registrieren	167
5.8.12	Zugriffsrechte bestimmen	171
5.9	Der Tomcat-Manager	186
5.9.1	Benutzer einrichten	186
5.9.2	URI-Kommandos	187
5.9.3	Die HTML-Version	187

6 Cocoon installieren 189

6.1	Blöcke	189
6.2	Cocoon besorgen	191
6.3	Cocoon kompilieren	192
6.3.1	Ant	192

6.3.2	Kompilierung starten	192
6.3.3	Kompilierung anpassen	193
6.4	Installieren	194
6.4.1	Cocoon starten mit Jetty	194
6.4.2	In Tomcat installieren	195

7 Die Sitemap **197**

7.1	Pipeline	198
7.1.1	SAX-Events und SAX-Streams	201
7.1.2	Interne Pipeline	201
7.1.3	Hello-World-Beispiel	202
7.2	Protokolle	203
7.2.1	Standard-Protokolle	204
7.2.2	Pseudo-Protokolle	205
7.3	Parameter und Variablen	206
7.3.1	Sitemap-Parameter	206
7.3.2	Parameter an Gruppierungen übergeben	207
7.3.3	Variablen	207
7.4	Die Sitemap-Komponenten	208
7.4.1	Komponenten-Typ	208
7.4.2	Registrieren einer Sitemap-Komponente	209
7.4.3	Default-Komponente	210
7.4.4	Verwenden einer Sitemap-Komponente	211
7.4.5	Matcher	212
7.4.6	Generator	219
7.4.7	Transformer	222
7.4.8	Serializer	226
7.4.9	Reader	228
7.4.10	Selector	229
7.4.11	Action	232
7.4.12	Konfigurieren einer Sitemap-Komponente	235
7.5	Sitemap-Resource	236
7.5.1	Erstellen einer Sitemap-Resource	236
7.5.2	Aufruf einer Sitemap-Resource	237
7.5.3	Verwendung von Parametern	238
7.6	Action-Set	238
7.6.1	Verwendung von Parametern	240
7.6.2	Ausführung selektieren	241
7.7	Redirects	242

7.8	View	243
7.8.1	Erstellen einer View	244
7.8.2	Platzierung von Labels	245
7.8.3	Aufruf einer View	248
7.9	Aggregation	248
7.9.1	Ein kleines Beispiel	251
7.10	Sub-Sitemap	254
7.10.1	Mounten einer Sub-Sitemap	255
7.10.2	Eine Sub-Sitemap erzeugen	257
7.11	Konfiguration	258
7.11.1	Konfiguration der Root-Sitemap	258
7.11.2	Logging	259
7.11.3	Caching	262
7.12	Fehlerbehandlung	263
7.12.1	Reihenfolge der Fehlerbehandlung	265
7.12.2	Exception Selector	265
7.12.3	XPath Exception Selector	266
7.12.4	Status Codes	268

8 Module 271

8.1	Registrieren von Modulen	271
8.2	Input-Module	272
8.2.1	Verwendung innerhalb einer Sitemap	274
8.3	Output-Module	275
8.4	Database-Module	275

9 Cocoon erweitern 277

9.1	Das Komponenten-Modell	278
9.1.1	Inversion of Control	278
9.1.2	Separation of Concerns	279
9.1.3	Komponente erstellen	279
9.1.4	Komponente laden: Der Service-Manager	285
9.1.5	Flowscript und der Service-Manager	287
9.2	Komponenten erweitern	288
9.2.1	Das Interface Contextualizable	289
9.2.2	Zugriff auf andere Komponenten	292
9.2.3	Konfiguration mit verschachtelten Elementen	293
9.2.4	Konfiguration mit Parametern	296
9.2.5	Initialisieren von Komponenten	298

9.2.6	Vor dem Zerstören aufräumen	298
9.2.7	Festlegen der Instanzbildung	299
9.2.8	Logging	302
9.3	Der Service Selector	306
9.4	Source-Resolving	309
9.4.1	Source-Resolver	310
9.4.2	Source	311
9.4.3	Eigene Source erstellen	313
9.5	Eigene Sitemap-Komponenten erstellen	322
9.5.1	Setup von Sitemap-Komponenten	322
9.5.2	Zugriff auf den Output-Stream	323
9.5.3	Producer, Consumer oder Pipe?	324
9.5.4	Eigene Action erstellen	325
9.5.5	Eigenen Generator erstellen	331
9.5.6	Eigenen Transformer erstellen	336
9.5.7	Eigenen Selector erstellen	340
9.5.8	Eigenen Matcher erstellen	344
9.5.9	Eigenen Reader erstellen	348
9.5.10	Eigenen Serializer erstellen	353
9.5.11	Caching von Sitemap-Komponenten	356
9.6	Das Test-Framework von Cocoon	360
9.6.1	Vorbereitung	361
9.6.2	Die Test-Klasse SitemapComponentTestCase	362
9.6.3	Einen Testfall konfigurieren	363
9.6.4	Eine Komponente testen	365

10 Control Flow 369

10.1	Zustandsautomat	369
10.2	Was ist Control Flow?	370
10.3	Continuations	374
10.3.1	Continuation laden	375
10.3.2	Konfiguration	376
10.3.3	Das »Zurück-Problem«	377
10.4	Flowscript	378
10.4.1	Der Aufbau	378
10.4.2	Warum JavaScript?	379
10.4.3	Integration von Java	380
10.4.4	Registrieren, starten und fortsetzen	382
10.4.5	Flow Object Model	384

10.4.6	Pipeline aus einem Flowscript aufrufen	385
10.4.7	Der Cocoon Flow Debugger	387
10.5	View-Komponenten	388
10.5.1	Velocity Generator	389
10.5.2	JXTemplate Generator/Transformer	392
10.5.3	JPath Logicsheet	395
10.6	Ein kleines Beispiel	396
10.7	Javaflow	402
10.7.1	Ein kleines Beispiel	403
10.8	Apples	405
10.8.1	Ein kleines Beispiel	407

11 Cocoon Forms 411

11.1	Wichtige Eigenschaften von Formular-Frameworks	412
11.2	Grundsätzliche Funktionsweise	413
11.3	Ein kleines Beispiel	417
11.3.1	Erstellen der Sub-Applikation	419
11.3.2	Erstellen der Form-Definition	419
11.3.3	Erstellen des Form-Templates	421
11.3.4	Erstellen des Controllers (Flowscript)	423
11.3.5	Erstellen der Bestätigungsseite	424
11.3.6	Erstellen der Sitemap-Einträge	425
11.3.7	Ausführen des Beispiels	427
11.4	Form-Definition	427
11.5	Widgets	428
11.5.1	Widget-Definition	429
11.5.2	Widget-Objekt	430
11.5.3	Widget-Zustände	431
11.5.4	Datentypen	432
11.5.5	Einfache Widgets	434
11.5.6	Listen-Widgets	436
11.5.7	Tree-Widget	444
11.5.8	Gruppierende Widgets	457
11.5.9	Messages-Widget	463
11.5.10	Repeater-Widget	464
11.5.11	Action-Widgets	465
11.5.12	Upload-Widget	468
11.6	Form-Template	469
11.6.1	Instanz-Elemente	470
11.6.2	Anpassen des Layouts	471

Inhalt

11.7	Ajax aktivieren	481
11.7.1	Anpassen des Form-Templates	481
11.7.2	Anpassen der Form-Pipeline	482
11.8	Validierung	483
11.8.1	<fd:assert/>	484
11.8.2	<fd:email/>	484
11.8.3	<fd:length/>	485
11.8.4	<fd:mod10/>	485
11.8.5	<fd:range/>	486
11.8.6	<fd:regexp/>	487
11.8.7	<fd:value-count/>	487
11.8.8	<fd:java/>	488
11.8.9	<fd:javascript/>	491
11.8.10	Fehlermeldungen internationalisieren	492
11.9	Event-Handling	494
11.9.1	Definition in der Form-Definition	495
11.9.2	Erstellen eines Java-Event-Listeners	495
11.9.3	Erstellen eines Form-Handlers	496
11.9.4	Erstellen eines JavaScript-Event-Listeners	497
11.10	Binding-Framework	498
11.10.1	Binding konfigurieren	501
11.10.2	Binding in einem Flowscrip verwenden	505
11.11	Cocoon Forms ohne Flowscrip	506
11.11.1	Verwenden des Bindings	512
12	XSP	515
12.1	Das XSP-Dokument	516
12.1.1	Ein kleines Beispiel	517
12.1.2	Wichtige Elemente	519
12.1.3	Vergleichsoperatoren	520
12.1.4	Andere Sprachen verwenden	520
12.2	Logicsheet	521
12.2.1	Built-In-Logicsheets	522
12.2.2	Eigenes Logicsheet erstellen	525
13	Datei-Upload	529
13.1	Ein kleines Beispiel	532

14 Datenbankzugriffe mit Cocoon	537
14.1 JDBC-Treiber installieren	537
14.2 Registrieren einer Datasource	538
14.2.1 Überwachen des Pools	541
14.3 Datenbankzugriffe mit Komponenten	542
14.3.1 Optimierung	545
14.4 Datenbankzugriffe mit Flowscripts	546
14.5 SQL Transformer	547
14.5.1 Abfragen an die Datenbank stellen	549
14.5.2 Updates an die Datenbank stellen	552
14.5.3 Substituieren	553
14.6 ESQL-Logicsheet	554
14.6.1 Abfragen an die Datenbank stellen	556
14.6.2 Updates an die Datenbank stellen	560
14.7 Datenbankzugriffe mit Actions	561
14.8 JNDI-Datasource	561
14.8.1 J2EE Connection registrieren	561
14.8.2 Resource Reference erzeugen	562
14.8.3 JNDI-Resource einrichten	563
14.9 Hibernate integrieren und verwenden	564
14.9.1 Hibernate besorgen und installieren	564
14.9.2 Der Hibernate-Wrapper	564
14.9.3 Der CloseHibernateSessionFilter	569
14.10 XML-Datenbanksysteme	571
14.10.1 XML-enabled und XML-native	571
14.10.2 XML-Dokumenttypen	573
14.10.3 Zugriff auf eine XML-Datenbank	574

15 Internationalisierung	581
15.1 Was steckt hinter der Internationalisierung?	581
15.2 Der I18n Transformer	581
15.2.1 Registrieren des I18n Transformers	582
15.2.2 Verwenden des I18n Transformers	583
15.2.3 Message-Katalog	584
15.2.4 XML-Dokumente für eine Übersetzung vorbereiten	588
15.2.5 Text übersetzen	588
15.2.6 Attribute übersetzen	594
15.2.7 Zahlen-, Datums- und Währungsformate lokalisieren	594

15.3	Die Locale Action	598
15.3.1	Locale Action registrieren	599
15.3.2	Verwenden der Locale Action	600
15.3.3	Das Locale Objekt	601
16 Cocoon im Einsatz: Das Portal		603
16.1	Das Projekt	603
16.2	Struktur	606
16.3	Benutzerverwaltung	607
16.3.1	Flowscript	611
16.3.2	JXTemplates erzeugen	615
16.3.3	Erstellen der Sitemap	619
16.4	News	620
16.4.1	news2html.xml – Transformation nach HTML	623
16.4.2	article2html.xml – Transformation nach HTML	626
16.4.3	article2fo.xml – Transformation nach PDF	628
16.5	Login	631
16.5.1	Erzeugen des Logins	631
16.5.2	Pipeline-Bereiche schützen	634
16.6	Zusammenfassung	637
17 Cocoon offline betreiben		639
17.1	Vorbereitungen	640
17.2	CLI-Kommandozeilen-Argumente	640
17.3	CLI-Konfigurationsdatei	642
Anhang		645
A	Tomcat	645
A.1	server.xml	645
A.1.1	<Server/>	646
A.1.2	<Service/>	646
A.1.3	<Connector/>	646
A.1.4	<Engine/>	647
A.1.5	<Host/>	648
A.1.6	<Context/>	648
A.2	web.xml	649
A.2.1	<web-app/>	650
A.2.2	<description/>	650

A.2.3	<display-name/>	650
A.2.4	<icon/>	651
A.2.5	<distributable/>	651
A.2.6	<context-param/>	651
A.2.7	<filter/>	651
A.2.8	<filter-mapping/>	652
A.2.9	<listener/>	652
A.2.10	<servlet/>	652
A.2.11	<servlet-mapping/>	653
A.2.12	<session-config/>	653
A.2.13	<mime-mapping/>	654
A.2.14	<welcome-file-list/>	654
A.2.15	<error-page/>	654
A.2.16	<jsp-config/>	655
A.2.17	<security-constraint/>	655
A.2.18	<login-config/>	655
A.2.19	<security-role/>	656
A.2.20	<env-entry/>	656
A.2.21	<ejb-ref/>	656
A.2.22	<ejb-local-ref/>	657
A.2.23	<service-ref/>	657
A.2.24	<resource-ref/>	657
A.2.25	<resource-env-ref/>	658
A.2.26	<message-destination-ref/>	658
A.2.27	<message-destination/>	658
A.2.28	<locale-encoding-mapping-list/>	659
B	Actions	661
B.1	Sendmail Action	661
B.1.1	Registrieren	661
B.1.2	Verwenden	662
B.2	Session Action	664
B.2.1	Registrieren	664
B.2.2	Verwenden	665
B.3	Die Locale Action	665
C	Generatoren	667
C.1	CSV Generator	667
C.1.1	Registrieren	667
C.1.2	Verwenden	668
C.2	Directory Generator	669
C.2.1	Registrieren	669
C.2.2	Verwenden	670

Inhalt

C.3	File Generator	673
	C.3.1 Registrieren	673
	C.3.2 Verwenden	673
C.4	HSSF Generator	674
	C.4.1 Registrieren	674
	C.4.2 Verwenden	674
C.5	HTML Generator	675
	C.5.1 Registrieren	675
	C.5.2 Verwenden	676
C.6	Image Directory Generator	676
	C.6.1 Registrieren	676
	C.6.2 Verwenden	676
C.7	JSP Generator	677
	C.7.1 Registrieren	677
	C.7.2 Verwenden	677
C.8	JXTemplate Generator	678
	C.8.1 Registrieren	678
	C.8.2 Verwenden	678
C.9	LinkStatus Generator	679
	C.9.1 Registrieren	679
	C.9.2 Verwenden	679
C.10	Request Generator	680
	C.10.1 Registrieren	680
	C.10.2 Verwenden	681
C.11	Script Generator	682
	C.11.1 Registrieren	683
	C.11.2 Verwenden	683
C.12	Search Generator	684
	C.12.1 Registrieren	685
	C.12.2 Verwenden	686
C.13	ServerPages Generator	688
	C.13.1 Registrieren	688
	C.13.2 Verwenden	688
C.14	Stream Generator	689
	C.14.1 Registrieren	689
	C.14.2 Verwenden	689
C.15	Velocity Generator	690
	C.15.1 Registrieren	690
	C.15.2 Verwenden	690

C.16	XPath Directory Generator	690
C.16.1	Registrieren	691
C.16.2	Verwenden	691
D	Matcher	695
D.1	Wildcard URI Matcher	695
D.1.1	Registrieren	695
D.1.2	Verwenden	696
E	Reader	697
E.1	JSP Reader	697
E.1.1	Registrieren	697
E.1.2	Verwenden	698
E.2	Resource Reader	698
E.2.1	Registrieren	698
E.2.2	Verwenden	699
F	Selectoren	701
F.1	Browser Selector	701
F.1.1	Registrieren	701
F.1.2	Verwenden	702
F.2	Host Selector	703
F.2.1	Registrieren	703
F.2.2	Verwenden	704
G	Serializer	705
G.1	HTML Serializer	705
G.1.1	Registrieren	705
G.1.2	Verwenden	707
G.2	PDF Serializer	707
G.2.1	Registrieren	707
G.2.2	Verwenden	707
G.3	PS Serializer	708
G.3.1	Registrieren	708
G.3.2	Verwenden	708
G.4	SVG/JPEG Serializer	709
G.4.1	Registrieren	709
G.4.2	Verwenden	710
G.5	SVG/PNG Serializer	710
G.5.1	Registrieren	711
G.5.2	Verwenden	712
G.6	SVG/TIFF Serializer	712
G.6.1	Registrieren	712
G.6.2	Verwenden	714

Inhalt


G.7	SVG/XML Serializer	714
	G.7.1 Registrieren	714
	G.7.2 Verwenden	715
G.8	XHTML Serializer	715
	G.8.1 Registrieren	716
	G.8.2 Verwenden	716
G.9	XML Serializer	717
	G.9.1 Registrieren	717
	G.9.2 Verwenden	718
H	Transformer	719
H.1	Augment Transformer	719
	H.1.1 Registrieren	719
	H.1.2 Verwenden	720
H.2	CInclude Transformer	720
	H.2.1 Registrieren	721
	H.2.2 Verwenden	721
H.3	EncodeURL Transformer	724
	H.3.1 Registrieren	724
	H.3.2 Verwenden	725
H.4	Filter Transformer	726
	H.4.1 Registrieren	727
	H.4.2 Verwenden	727
H.5	l18n Transformer	728
	H.5.1 Registrieren	728
	H.5.2 Verwenden	730
H.6	Log Transformer	731
	H.6.1 Registrieren	731
	H.6.2 Verwenden	731
H.7	Read DOM Session Transformer	732
	H.7.1 Registrieren	732
	H.7.2 Verwenden	732
H.8	SourceWriting Transformer	733
	H.8.1 Registrieren	733
	H.8.2 Verwenden	734
H.9	SQL Transformer	735
	H.9.1 Registrieren	735
	H.9.2 Verwenden	736
H.10	Tee Transformer	738
	H.10.1 Registrieren	739
	H.10.2 Verwenden	739

H.11	Write DOM Session Transformer	739
H.11.1	Registrieren	740
H.11.2	Verwenden	740
H.12	XInclude Transformer	741
H.12.1	Registrieren	741
H.12.2	Verwenden	741
H.13	XSLT Transformer	742
H.13.1	Registrieren	743
H.13.2	Verwenden	744
I	FOM	747
I.1	Cocoon	747
I.1.1	Properties von Cocoon	748
I.1.2	Funktionen von Cocoon	749
I.2	Request	753
I.2.1	Funktionen von Request	753
I.3	Response	759
I.3.1	Funktionen von Response	759
I.4	Session	759
I.4.1	Funktionen von Session	761
I.5	Context	763
I.5.1	Funktionen von Context	764
I.6	Cookie	765
I.6.1	Funktionen von Cookie	765
I.7	Log	767
I.7.1	Funktionen von Log	768
I.8	WebContinuation	769
I.8.1	Properties von WebContinuation	769
I.8.2	Funktionen von WebContinuation	770
J	XSP	773
J.1	Automatisch importierte Klassen	773
J.2	Implizite Objekte	774
J.3	XSP-Elemente	775
J.3.1	<xsp:page/>	776
J.3.2	<xsp:structure/>	776
J.3.3	<xsp:include/>	777
J.3.4	<xsp:init-page/>	777
J.3.5	<xsp:exit-page/>	778
J.3.6	<xsp:logic/>	778
J.3.7	<xsp:expr/>	779
J.3.8	<xsp:element/>	780

Inhalt

J.3.9	<xsp:attribute/>	781
J.3.10	<xsp:content/>	782
J.3.11	<xsp:pi/>	782
J.3.12	<xsp:comment/>	783
J.3.13	<xsp:param/>	783
K	Input-Module	785
K.1	BaseLink Module	785
K.1.1	Konfiguration in cocoon.xconf	785
K.1.2	Verwendung in der Sitemap	785
K.2	DateInput Module	786
K.2.1	Konfiguration in cocoon.xconf	786
K.2.2	Verwendung in der Sitemap	786
K.3	Defaults Module	787
K.3.1	Konfiguration in cocoon.xconf	787
K.3.2	Verwendung in der Sitemap	787
K.4	FlowAttribute Module	787
K.4.1	Konfiguration in cocoon.xconf	788
K.4.2	Verwendung in der Sitemap	788
K.5	GlobalInput Module	788
K.5.1	Konfiguration in cocoon.xconf	788
K.5.2	Verwendung in der Sitemap	789
K.6	HeaderAttribute Module	790
K.6.1	Konfiguration in cocoon.xconf	790
K.6.2	Verwendung in der Sitemap	790
K.7	PropertiesFile Module	791
K.7.1	Konfiguration in cocoon.xconf	791
K.7.2	Verwendung in der Sitemap	791
K.8	RandomNumber Module	792
K.8.1	Konfiguration in cocoon.xconf	792
K.8.2	Verwendung in der Sitemap	792
K.9	RawRequestParameter Module	793
K.9.1	Konfiguration in cocoon.xconf	793
K.9.2	Verwendung in der Sitemap	793
K.10	RealPath Module	794
K.10.1	Konfiguration in cocoon.xconf	794
K.10.2	Verwendung in der Sitemap	794
K.11	RequestAttribute Module	794
K.11.1	Konfiguration in cocoon.xconf	795
K.11.2	Verwendung in der Sitemap	795

K.12	Request Module	795
K.12.1	Konfiguration in cocoon.xconf	796
K.12.2	Verwendung in der Sitemap	796
K.13	RequestParameter Module	796
K.13.1	Konfiguration in cocoon.xconf	797
K.13.2	Verwendung in der Sitemap	797
K.14	SessionAttribute Module	797
K.14.1	Konfiguration in cocoon.xconf	798
K.14.2	Verwendung in der Sitemap	798
K.15	Session Module	798
K.15.1	Konfiguration in cocoon.xconf	799
K.15.2	Verwendung in der Sitemap	799
K.16	SystemProperty Module	799
K.16.1	Konfiguration in cocoon.xconf	800
K.16.2	Verwendung in der Sitemap	800
K.17	XMLFile Module	801
K.17.1	Konfiguration in cocoon.xconf	801
K.17.2	Verwendung in der Sitemap	802
L	Glossar	803
M	Wichtige und interessante Quellen	807
N	Inhalt der CD-ROM	813
N.1	Beispiele	813
N.1.1	Cocoon	813
N.1.2	DOM und SAX	814
N.1.3	Tomcat	814
N.1.4	XSLT und XSL-FO	814
N.2	Listings	814
N.3	Software	814
	Index	815



Hier erhalten Sie einen ersten Einblick in Cocoon und seine primären Aufgabengebiete.

1 Einführung in Cocoon

In diesem Kapitel möchte ich Sie an Cocoon heranzuführen und Ihnen einen Überblick über die Einsatzgebiete geben. Dabei betrachte ich unter anderem die wesentlichen Aspekte moderner Web-Applikationen und beleuchte deren Entsprechungen in Cocoon.

1.1 Geschichte von Cocoon

Im Jahre 1998 arbeitete der italienische Student Stefano Mazzocchi an einer Möglichkeit, effektiv das Layout der damals für Java-Projekte aktuellen Website <http://java.apache.org> von dessen Inhalt trennen und erst bei der Darstellung wieder verbinden zu können. Zu diesem Zeitpunkt war eine solche Vorgehensweise noch nicht sehr weit verbreitet. Mazzocchi entschied sich für die Trennung der beiden Bereiche XSLT einzusetzen, das zu diesem Zeitpunkt erst als Working-Draft verfügbar und von der heutigen Leistungsfähigkeit noch weit entfernt war.

Ende 1998 hatte Mazzocchi ein Java-Servlet geschrieben, das die Zusammenführung von Layout und Inhalt durch XSL-Transformationen übernahm. Cocoon 1.0 war somit geboren. Nach und nach fand das System immer mehr Anhänger und wurde von da an aktiv weiterentwickelt.

Cocoon 1.0

Nach einem Jahr Entwicklung hatte Cocoon 1.x bereits einen stattlichen Umfang erreicht. Da XML zu diesem Zeitpunkt noch in den Kinderschuhen steckte, war es nicht verwunderlich, dass manche Teile von Cocoon bestimmten Beschränkungen unterlagen. So war beispielsweise die Transformation von größeren XML-Dokumenten nur umständlich oder gar nicht möglich, weil Cocoon 1.x als Basis für die XML-Verarbeitung DOM verwendete. Daneben war die Architektur nicht optimal für eigene Erweiterungen ausgelegt. Dies sollte sich jedoch grundlegend ändern.

Cocoon 2 Im November 1999 startete Stefano Mazzocchi einen Aufruf, Cocoon grundlegend umzuschreiben und die Architektur zukunftsweisend zu gestalten. Nach langen zwei Jahren war es endlich soweit: Cocoon 2 (C2) wurde im Jahre 2000 veröffentlicht. Neben der offenen Architektur waren Performance, Stabilität, Skalierbarkeit und Modularität die wichtigsten Punkte, die es bei der Entwicklung von Cocoon 2 zu berücksichtigen galt und die in vorbildlicher Weise umgesetzt wurden.

Cocoon 2 wurde als eine »abstrakte Maschine« konzipiert, die unabhängig von einer Servlet-Umgebung lauffähig ist. Es ist somit auch als Standalone-Applikation verwendbar und kann über ein Kommandozeilen-Tool verwaltet oder direkt in andere Java-Applikationen eingebettet werden. Am häufigsten aber wird Cocoon als Web-Applikation innerhalb eines Servlet-Containers betrieben. Die Schnittstelle hierbei bildet nach wie vor ein Servlet.

An die Stelle der zeit- und speicherintensiven Verarbeitung von XML-Dokumenten durch DOM traten SAX und ein ausgefeilter Caching-Mechanismus. Das Erzeugen und Ausliefern von Dokumenten wurde damit um ein Vielfaches beschleunigt.

Durch die Verwendung verschiedener gängiger Design-Patterns wie beispielsweise »Inversion of Control« oder »Separation of Concerns« und eine zentralisierte Konfigurationsweise ist es nun wesentlich einfacher, Cocoon zu betreiben und zu erweitern. Eigene Web-Applikationen können nun in kürzester Zeit entwickelt und mit zahlreichen Features ausgestattet werden.

Cocoon 2.1 Eine weitere deutliche Verbesserung brachte die Cocoon-Generation ab Version 2.1, die Mitte 2003 veröffentlicht wurde. Darin wurden einige zukunftsweisende Mechanismen implementiert, die es möglich machten, Web-Applikationen noch einfacher und übersichtlicher zu erstellen. Die bedeutendste Veränderung ist dabei sicherlich die Einführung der Continuation-Technik, welche mit Flowscripts eine komfortable Alternative zu XSP und Actions bietet.

Cocoon 2.1.5 Der nächste Meilenstein in der Entwicklung von Cocoon war die Einführung eines umfangreichen Formular-Frameworks, welches in der Alpha- und Beta-Phase unter dem Namen *Woody* zusammen mit Cocoon ausgeliefert wurde. Ab Version 2.1.5 trägt dieses Framework den offiziellen Namen *Cocoon Forms*. Diese Neuerung ist für die Entwicklung von Web-Applikationen so bedeutend, dass ihm ein ganzes Kapitel in diesem Buch gewidmet ist: Kapitel 11, »Cocoon Forms«.



1.2 Anforderungen an Web-Applikationen

Bevor wir uns in diesem Kapitel ansehen, wie Cocoon grundsätzlich funktioniert und welche grundlegenden Features es bietet, wollen wir zunächst betrachten, welche Eigenschaften jede Web-Applikation bieten sollte.

Für die Entwicklung moderner Web-Applikationen müssen heutzutage wesentlich mehr Punkte berücksichtigt werden als das einfache Erzeugen einer HTML-Seite und deren Auslieferung. Die Punkte sind natürlich von Anforderung zu Anforderung verschieden, können aber allgemein in folgende Kategorien eingeteilt werden:

- ▶ Trennung von Layout, Inhalt und Logik
- ▶ Plattformunabhängigkeit
- ▶ Personalisierung
- ▶ Modularität und einfache Erweiterbarkeit
- ▶ Internationalisierung
- ▶ Skalierbarkeit
- ▶ Multichannel Publishing (verschiedene Zielplattformen)
- ▶ Zugriff auf verschiedene Datenquellen (XML, SQL, LDAP usw.)
- ▶ Einfache und standardisierte Wartung

Die Realisierung vieler der hier genannten Punkte ist in verschiedenen Web-Applikationen häufig auf dieselbe Weise erforderlich. Daher existieren für die meisten Aufgaben sogenannte *Frameworks*. Dabei handelt es sich um eine Sammlung verschiedener Lösungen und Vorgehensweisen für bestimmte Probleme.

Framework

Cocoon ist ein solches Framework. Es bietet Ihnen Lösungen zu allen oben genannten Punkten, indem es Ihnen verschiedene Werkzeuge zur Verfügung stellt und für die Realisierung Ihrer Web-Applikation einen gewissen Rahmen vorgibt.

1.2.1 Trennung von Layout, Inhalt und Logik

Der wichtigste Punkt bei der Realisierung einer Web-Applikation ist die Trennung von Layout, Inhalt und Logik. Diese Notwendigkeit wurde in den vergangenen Jahren auch von den meisten Web-Entwicklern erkannt.

MVC

Dieses Konzept wird aktuell von nahezu allen Web-Frameworks unterstützt und durch ein Design-Pattern, das unter dem Namen Model-View-Controller (MVC) bekannt ist, umgesetzt.

Model 2 MVC ist nicht neu und wurde erstmals in der Programmiersprache Smalltalk im Jahre 1980 eingesetzt. Für Web-Applikationen, deren Kommunikation zustandslos ist, wurde MVC entsprechend angepasst. Diese spezielle Variante von MVC wird Model 2 genannt. Die einzelnen Schichten bilden jeweils einen eigenen Bereich einer Applikation mit festen Zuständigkeiten:

▶ **Model**

Diese Schicht stellt die Datenquelle sowie die Businesslogik für die Applikation zur Verfügung.

▶ **View**

Übernimmt die Darstellung der Daten aus dem Model für den Endanwender.

▶ **Controller**

Der Controller – auch Ablauflogik genannt – nimmt Eingaben vom Benutzer entgegen und bildet diese auf Funktionen des Models oder des Views ab.

Die einzelnen Schichten können Sie sich beispielsweise folgendermaßen vorstellen: Eine Klasse, die den Zugriff auf eine Datenbank realisiert, stellt die Model-Schicht dar. Die Daten selbst werden in der Regel als sogenannte Beans¹ zur Verfügung gestellt. Die View-Schicht könnte in der Web-Programmierung eine HTML-Seite sein, in die Beans platziert werden. Hinzu kommt der entsprechende Mechanismus um die Daten aus den Beans in das HTML-Dokument zu schreiben. Der Controller ist letztendlich die Instanz, welche den gesamten Applikationsfluss verwaltet und View- sowie Model-Schicht miteinander verbindet.

In Cocoon wird der Controller meist von Actions, Flowscripts und zum Teil auch von der Sitemap realisiert. Die Model-Schicht übernehmen häufig bestimmte Sitemap- oder eigene Komponenten und die View-Schicht kann durch verschiedene Transformer (beispielsweise der XSL Transformer oder der JXTemplate Transformer) abgedeckt werden. Diese werden Sie noch ausführlich kennen lernen.

1 Klasse, die die Daten repräsentiert und über entsprechende Zugriffsmethoden verfügt.

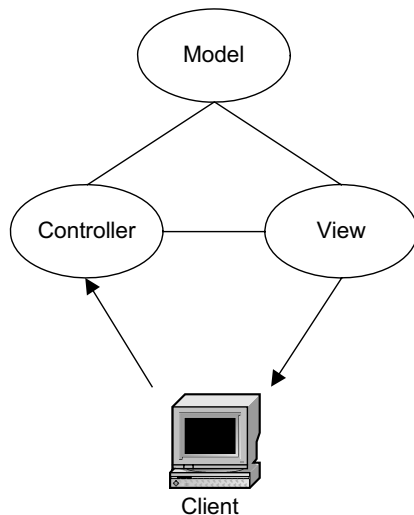


Abbildung 1.1 Das MVC-Konzept im Request-Response-Zyklus

1.2.2 Plattformunabhängigkeit

Unter dieser Voraussetzung versteht man, dass eine Anwendung nicht an ein bestimmtes Betriebssystem gebunden ist. Da Cocoon vollständig in Java geschrieben ist, ist diese Voraussetzung von vornherein gegeben. Sie können Cocoon auf jedem Betriebssystem betreiben, für das es eine aktuelle Java-Laufzeitumgebung gibt. Vor allem sind dies die Betriebssysteme Unix, Linux und Windows.

1.2.3 Personalisierung

Auf vielen kommerziellen Seiten im Internet gehört das Thema Personalisierung zu einem unverzichtbaren Instrument. Zum einen, um dem Kunden eine an seine Bedürfnisse angepasste Website anbieten zu können, zum anderen, um seine »Surf-Gewohnheiten« festzuhalten und später auszuwerten.

Die Implementierung einer solchen Funktionalität ist von Ihrer Applikation abhängig. Cocoon stellt Ihnen dabei die nötigen Werkzeuge zur Verfügung.

1.2.4 Modularität und einfache Erweiterbarkeit

Unabhängige Bausteine, die sich unter bestimmten Bedingungen beliebig und einfach kombinieren lassen, stellen eine wichtige Eigenschaft einer modernen Web-Applikation dar.

Cocoon stellt Ihnen eine Vielzahl verschiedener Komponenten zur Verfügung, die durch eine einfache Deklaration sofort eingebunden werden können. Sollte trotzdem für Ihr spezielles Problem einmal eine Komponente fehlen, so können Sie relativ einfach eine neue erstellen oder eine vorhandene erweitern. Zu diesem Zweck existiert eine hervorragende Struktur von Interfaces und Basisklassen, die eine solche Erweiterung äußerst einfach machen.

1.2.5 Internationalisierung

Im Zeitalter der Globalisierung ist es auf vielen Websites erforderlich, den Inhalt in verschiedenen Sprachen und angepasst auf bestimmte Lokalitäten darzustellen.

Cocoon stellt hierfür eine spezielle Komponente, den I18n Transformer, zur Verfügung. Er unterstützt Sie dabei, sowohl Texte als auch Datums-, Zahl- und Währungsformate in die jeweilige Zielsprache zu übersetzen.

1.2.6 Skalierbarkeit

Das Wort Skalierbarkeit hat in diesem Zusammenhang zwei Bedeutungen. Zum einen ist damit gemeint, dass Ihre Applikation auch einer großen »Anfrageflut« standhalten muss, und deshalb die Leistungsfähigkeit entsprechend erweitert werden kann. Zum anderen muss die Web-Applikation selbst mit den eigenen Anforderungen mitwachsen können, um damit möglichst viele Probleme zentral zu lösen.

Vor allem durch einen umfangreichen Caching-Mechanismus und die Fähigkeit, Komponenten konfigurativ zu poolen, ist Cocoon auch einer großen Anzahl von Anfragen ohne Weiteres gewachsen. Einige Benchmarks der letzten Zeit unterstreichen diese Tatsache.

Durch die erweiterbare Komponentenarchitektur und die Vielzahl der unterschiedlichsten Schnittstellen ist eine Cocoon-Applikation stets erweiterbar und kann optimal mit den eigenen Anforderungen wachsen.

1.2.7 Multichannel Publishing²

Zu den größten Stärken von Cocoon gehört die Fähigkeit, Inhalte in verschiedensten Formaten ohne aufwändige Programmierung oder Erweiterung der Web-Applikation darzustellen.

2 Auch Cross-Media-Publishing genannt.



Angenommen, Sie möchten Ihre aktuellen Geschäftszahlen publizieren: Dies soll zum einen in Form einer HTML-Seite geschehen. Zum anderen werden diese Zahlen zusätzlich von der Buchhaltung in Form einer Excel-Datei, von den Kunden als Offline-Version im PDF-Format und der Druckerei im PostScript-Format benötigt. Viele Frameworks würden an dieser Stelle an ihre Grenzen stoßen, da der Aufwand zum Erzeugen und Ausliefern der verschiedenen Formate häufig enorm ist. Dabei stellt sich auch hier das Problem, Inhalt, Layout und Logik voneinander zu trennen. Dies nicht zuletzt, um die programmierten Werkzeuge auch für andere Dokumente verwenden zu können. Im Fall eines HTML-Dokuments ließe sich dieses Problem noch relativ einfach lösen, da es hier für fast jede Sprache bereits gut funktionierende Ansätze gibt. Doch was ist beispielsweise im Fall eines PDF- oder Excel-Dokuments? Nicht selten wird hierfür eine spezielle Lösung programmiert, die in anderen Fällen nicht mehr verwendet werden kann.

Cocoon setzt hierbei auf die Vielzahl bereits vorhandener Lösungen und integriert diese über eigene Schnittstellen. So wird beispielsweise für das Erzeugen von PDF- und PostScript-Dokumenten das Apache-XML-Projekt FOP³ oder für Excel-Dokumente das Apache-Jakarta-Projekt POI⁴ verwendet. Somit ist sichergestellt, dass sich alle Teilbereiche unabhängig von Cocoon kontinuierlich weiter entwickeln.

Die am häufigsten verwendeten Ausgabeformate, die von Cocoon standardmäßig erzeugt werden, sind:

- ▶ HTML/XHTML
- ▶ PDF
- ▶ PostScript
- ▶ XML
- ▶ XLS (Excel)
- ▶ SVG
- ▶ Zip

Daneben gibt es noch eine Vielzahl weiterer Zielformate, für die bereits erzeugende Komponenten existieren.

³ <http://xmlgraphics.apache.org/fop>

⁴ <http://jakarta.apache.org/poi>

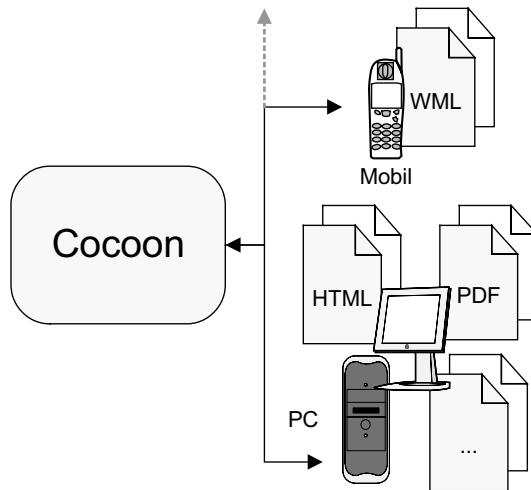


Abbildung 1.2 Multichannel Publishing mit Cocoon

1.2.8 Verschiedene Datenquellen

Dass in Web-Applikationen Datenbanken für das Speichern und Abrufen von Informationen eingesetzt werden, ist eigentlich selbstverständlich. Nicht selbstverständlich ist aber die Art und Weise, wie Verbindungen zu Datenbanken hergestellt und verwaltet werden. Hierbei gilt es häufig folgende Probleme zu lösen:

- ▶ Abstraktion der Datenbankverbindung
- ▶ Zentrale Konfiguration der Verbindungsdaten
- ▶ Mehrere gleichzeitige Datenquellen
- ▶ Verbindungs-Pooling
- ▶ Überwachung der Verbindungen

JDBC Da Cocoon in Java programmiert ist, und somit eventuelle Erweiterungen ebenfalls diese Sprache verwenden können, bietet sich für den Zugriff auf Datenbanken *JDBC* an. Dadurch ist bereits der erste Punkt »Abstraktion der Datenbank« erfüllt. Mittels JDBC lassen sich nahezu alle existierenden Datenbanken einbinden. Ein Austausch kann dabei ohne Änderungen des Codes der Web-Applikation erfolgen.

Für die übrigen Punkte bringt Cocoon das Konzept der *Datasources* mit. Hierdurch werden alle Daten-Ressourcen an einer zentralen Stelle unter einem eindeutigen Namen registriert und konfiguriert. Daneben kann für jede einzelne Datasource ein Tuning des standardmäßig vorhandenen Verbindungspools erfolgen. Auch das Logging kann entsprechend konfi-



guriert werden. Ebenso können JNDI-Ressourcen auf diese Weise eingebunden werden.

Neben Datenbanken können auch andere Datenquellen Informationen liefern. Beispielsweise ein LDAP-Server um Benutzerinformationen zu erhalten oder eine andere Website, mit der Inhalte ausgetauscht werden sollen. Hierfür stellt Cocoon wiederum verschiedene Komponenten bereit. Diese Komponenten kommunizieren mit dem jeweiligen Server und stellen die empfangenen Informationen in der Regel in Form von XML zur Verfügung. Dieses XML-Dokument kann anschließend von weiteren Cocoon-Komponenten verarbeitet werden. Auf diese Weise ist auch hier eine Schnittstelle vorhanden, an die zahlreiche Datenquellen angeschlossen werden können. Die einzige Voraussetzung hierfür ist, dass die Daten in Form von XML darstellbar sind. Für viele solche Anbindungen existieren in Cocoon bereits Komponenten. So ist beispielsweise der *LDAP-Transformer* verfügbar, um mit einem LDAP-Server kommunizieren zu können oder der *Web Service Proxy Generator*, um Inhalte anderer Websites in Form von XML einzubinden.

Andere
Datenquellen

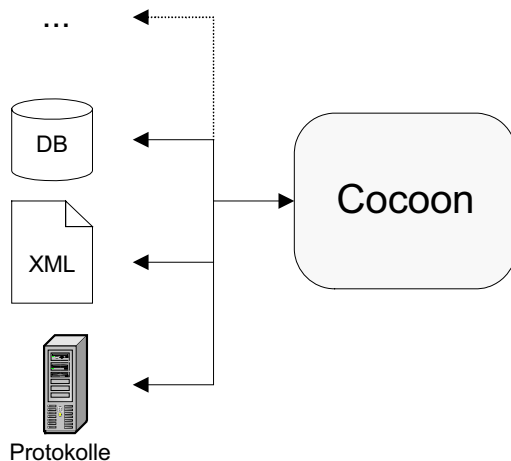


Abbildung 1.3 Cocoon besitzt fertige Schnittstellen zu verschiedenen Datenquellen.

1.3 Ein kleiner Einblick in Cocoon

Nachdem sie erfahren haben, welche grundsätzlichen Eigenschaften Web-Applikationen und -Frameworks mitbringen sollten und wie diese in Cocoon realisiert werden, wollen wir uns in diesem Abschnitt ansehen, welche Stellung Cocoon in einem Server-Park besitzen kann und wie dessen allgemeine Funktionsweise aussieht.

1.3.1 Cocoon ist Middleware

Three-Tier-Model Um eine bestmögliche Wart-, Erweiter- und Austauschbarkeit gewährleisten zu können, ist man in der Servertechnik dazu übergegangen, die einzelnen Bereiche einer Anwendung in verschiedene Gruppen zu untergliedern und diese Gruppen entsprechend zu verteilen. Eine solche Architektur nennt man *n-Tier-, n-Schicht- oder Multi-Tier-Architecture*. Das »n« steht dabei für die Anzahl der verschiedenen Schichten. In den meisten Fällen findet das *Three-Tier-Modell* Verwendung. Die Schichten unterteilen sich dabei in die drei Gruppen *User tier*, *Business tier* und *Data tier*, wie in der nachfolgenden Grafik zu sehen ist.



Abbildung 1.4 Die Three-Tier-Architektur

Das User tier ist für die Darstellung der Daten des Endanwenders verantwortlich. In der Web-Entwicklung übernimmt dies in der Regel der Web-Browser des Clients. Das Business tier ist der Logikbereich einer Anwendung. Cocoon füllt häufig genau diese Position in einer Web-Applikation aus. Es reagiert auf Eingaben, die durch das User tier entgegen genommen wurden, führt komplexe Businesslogik aus und erzeugt die entsprechenden Antworten. Darüber hinaus kann Cocoon hervorragend für die Konsolidierung von Systemen verwendet werden, da es die benötigten Daten konvertieren und anschließend stets in einem einheitlichen Format (XML) anbieten kann. Auf der anderen Seite kommuniziert Cocoon mit dem Data tier, welches für die Datenhaltung im allgemeinen Sinne zuständig ist. Dies können beispielsweise Datenbanken, CRM-Systeme oder Warenwirtschaftssysteme sein.

Bei einer Multi-Tier-Architektur mit mehr als 3 Schichten kann es je nach Design vorkommen, dass Cocoon lediglich für eine Teilaufgabe zuständig ist. Beispielsweise die Aufbereitung der Präsentationsschicht oder die Verwaltung der Schnittstellen nach »außen«. Dies muss je nach Anforderungen neu bewertet werden. Grundsätzlich ist aber wichtig, dass Cocoon generell in der Lage ist, den vollständigen Bereich des Business tiers auszufüllen.

1.3.2 Umgebungen

Wie Sie zu Beginn dieses Kapitels bereits erfahren haben, wurde Cocoon als abstrakte Maschine entworfen und ist somit außer an Java an keine konkrete Umgebung gebunden. Sie können Cocoon sowohl als Stand-alone-Applikation als auch integriert in andere Java-Anwendungen oder als Web-Applikation in einem beliebigen Servlet-Container betreiben. Die letztere Variante wird den Schwerpunkt in diesem Buch bilden.

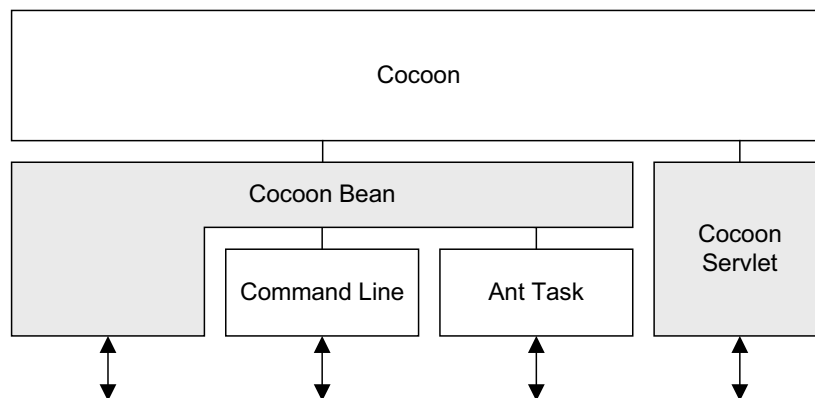


Abbildung 1.5 Die verschiedenen Möglichkeiten, Cocoon zu betreiben

Cocoon Bean

Für die Integration von Cocoon als Bestandteil einer Java-Anwendung ist das Cocoon Bean vorgesehen. Dabei handelt es sich um eine Schnittstelle in Form einer Java-Klasse, über die zwischen der Java-Anwendung und Cocoon kommuniziert werden kann. Die APIDoc zu dieser Bean gibt einen hinreichenden Überblick über die verfügbaren Methoden und ihre Bedeutungen. Sie finden sie unter der Adresse:

<http://cocoon.apache.org/2.1/apidocs/org/apache/cocoon/bean/CocoonBean.html>

Ein gutes Beispiel für die Verwendung der Cocoon Bean ist die Anwendung CLI, die durch die Klasse `org.apache.cocoon.Main` realisiert wird.

Command Line

Der Betrieb als Standalone-Anwendung erfolgt mit Hilfe des mitgelieferten *Command Line Interfaces (CLI)*. Dabei handelt es sich um eine Konsolenanwendung, über die Cocoon gesteuert werden kann. In Kapitel 17, »Cocoon offline betreiben«, erfahren Sie mehr darüber, wie CLI eingesetzt werden kann.

Ant-Task

Eine weitere, immer beliebter werdende Methode für den Offline-Betrieb von Cocoon ist der verfügbare Cocoon-Task für das Build-Tool *Ant*⁵. Hiermit können Sie Cocoon direkt durch Ant ausführen und mit anderen Ant-Tasks kombinieren. Mehr Informationen hierzu erhalten Sie auf der Website von Ant:

<http://ant.apache.org/external.html#Tasks>

oder auf der Cocoon-Website:

<http://cocoon.apache.org/2.1/userdocs/offline/ant.html>

Cocoon Servlet

Diejenige Umgebung, in der Cocoon mit Abstand am häufigsten eingesetzt wird, ist die Web-Applikation innerhalb eines Servlet-Containers. Dabei kommuniziert der Servlet-Container mit der Cocoon-Applikation über ein einzelnes Servlet.

Vereinfacht gesprochen »wrappt« das Cocoon-Servlet die notwendigen Servlet-Objekte, wie beispielsweise *Request*, *Session* oder *Response*, leitet diese an Cocoon weiter und umgekehrt. In Ihrer Cocoon-Applikation können Sie anschließend diese Objekte entsprechend verwenden. Somit ist eine nahtlose Integration in die gesamte Servlet-Welt gewährleistet.

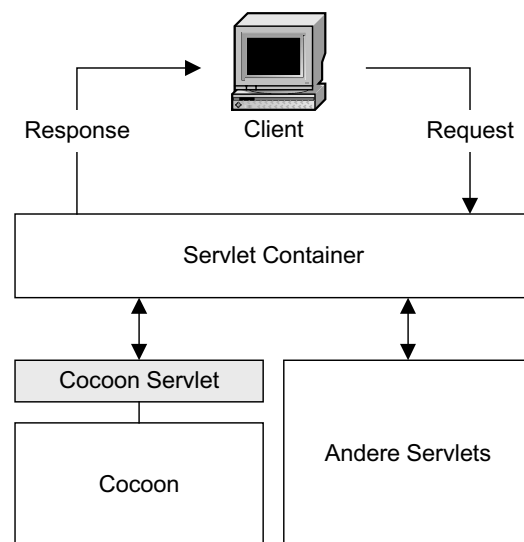


Abbildung 1.6 Die Rolle des Cocoon-Servlets

5 <http://ant.apache.org>



Falls Sie nicht wissen, was ein Servlet oder Servlet-Container ist, sollten Sie sich unbedingt auch das Kapitel 5, »Tomcat«, ansehen. Dort erhalten Sie ein fundiertes Basiswissen über diese Technologien, die es Ihnen im weiteren Verlauf dieses Buches erleichtern, Cocoon richtig einzuordnen und dessen Funktionsweise in einer Webumgebung zu verstehen.

1.3.3 Request-Response-Zyklus

Cocoon arbeitet – ähnlich wie das HTTP-Protokoll – nach dem Request-Response-Zyklus. Das heißt, an Cocoon wird zunächst ein Request (Anfrage) gesendet wird, der anschließend durch einen entsprechenden Response (Antwort) beantwortet wird. Eine Antwort könnte beispielsweise eine angefragte HTML-Seite oder eine Fehlerseite sein, falls das Dokument nicht gefunden wurde. Es ist wichtig zu verstehen, dass Cocoon – unabhängig von der verwendeten Umgebung – ohne spezielle Erweiterungen stets nur dann eine Aktion ausführt, wenn zuvor eine entsprechende Anfrage erfolgt ist.

Wird Cocoon als Web-Applikation in einem Servlet-Container betrieben, so wird der Request über einen URI – beispielsweise in Ihrem Web-Browser – dargestellt. Dieser besitzt häufig folgende Form:

```
http://localhost:8080/cocoon/index.html
```

Durch diesen URI wird zunächst das Cocoon-Servlet aufgerufen. Dieses entfernt alle nicht benötigten Angaben aus dem URI bis auf die relative Pfadangabe `index.html`. Dieser Teil wird *Sitemap-URI* genannt und wird von Cocoon anschließend dahingehend »untersucht«, ob in der aktuellen Sitemap eine Pipeline gefunden werden kann, welche auf diese Anfrage passt. Falls eine Übereinstimmung gefunden wurde, wird ein Ergebnisdokument über das Cocoon-Servlet und den Servlet-Container zurück an den Client gesendet. Andernfalls wird eine entsprechende Fehlermeldung erzeugt.

Sitemap-URI

Falls Sie sich nun fragen, was eine Sitemap ist, so kann ich Ihnen hier kurz antworten, dass dies die zentrale Datei in Cocoon ist, in der unter anderem alle Anfragen entsprechend auf sogenannte Pipelines »geroutet« werden. Im Kapitel 7, »Die Sitemap«, erfahren Sie alles, was Sie über diese wichtige Datei wissen müssen.

1.4 Die Rolle von XML in Cocoon

Cocoon wird häufig auch als XML-Publishing-System bezeichnet. Doch welche Rolle spielt hier XML?

Innerhalb von Cocoon existiert eine sogenannte *Pipeline-Technik*. Dabei handelt es sich um verschiedene Komponenten, die wie einzelne Bausteine hintereinander geschaltet werden können. Sie besitzen die Aufgabe, die Struktur und den Inhalt eines eingehenden XML-Dokuments so zu bearbeiten, dass das Zielformat (z. B. HTML oder PDF) erstellt bzw. das Zielsystem (z. B. Browser oder PDF-Viewer) die erzeugten Daten auch auswerten kann. Die einzelnen Komponenten innerhalb einer solchen Pipeline kommunizieren die Daten immer in eine Richtung in Form von XML.

Nach XML und zurück

In den vorangegangenen Abschnitten haben Sie erfahren, dass Cocoon verschiedene Schnittstellen nach außen zur Verfügung stellt, um mit anderen Systemen kommunizieren zu können. Jedes dieser Systeme schreibt dabei häufig eine eigene Art und Weise vor, wie eine solche Kommunikation mit ihm erfolgen muss. In vielen Fällen wird dies nicht mit XML realisiert. Die einzelnen Komponenten von Cocoon spielen hierbei eine Art »Übersetzer-Rolle«. Sie übersetzen die äußeren Kommunikationsdaten der einzelnen Systeme in XML und umgekehrt zurück. Somit ist innerhalb von Cocoon immer eine einheitliche Verarbeitungsweise der eingehenden Daten in Form von XML möglich.

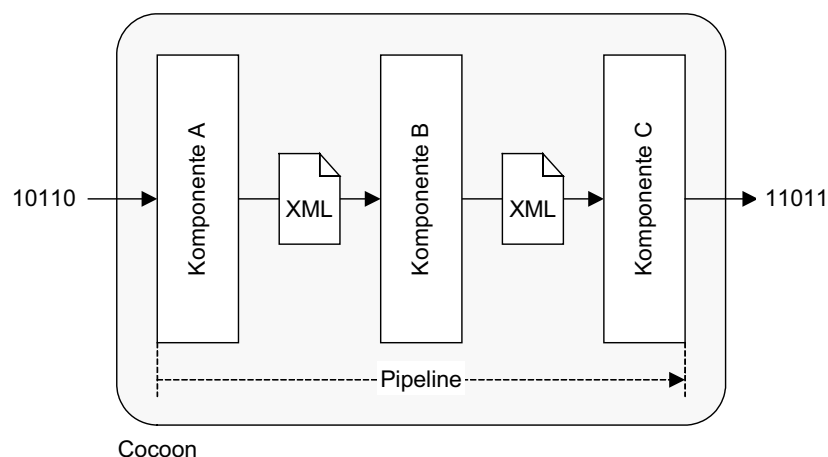


Abbildung 1.7 Übersetzung in XML und zurück mit Cocoon

Da immer mehr Anbieter von Systemen die externe Kommunikation auf Basis von XML realisieren, ist die Übersetzung von bzw. zum Zielsystem-Format in vielen Fällen nicht mehr notwendig. Die eintreffenden XML-Daten können somit direkt in Cocoon weiterverarbeitet werden ohne zuvor entsprechend konvertiert zu werden.

XSLT

XML ist ein weltweit eindeutiger Standard. Nicht zuletzt deswegen ist eine große Anzahl an Werkzeugen für die Verarbeitung von XML-Daten vorhanden. Das mit Abstand wichtigste Werkzeug innerhalb von Cocoon ist XSLT. Mit dieser Technik lassen sich XML-Dokumente nach eigenen Vorgaben verändern (transformieren) und somit an die eigenen Bedürfnisse anpassen. Als Ergebnis wird in der Regel wiederum ein XML-Dokument erzeugt. Durch die bereits vorgegebene Struktur und die Integration von XSLT lassen sich solche Transformationen äußerst komfortabel durchführen. Dabei kann ein Basisdokument in verschiedene Zielformate transformiert werden. Somit ist es lediglich notwendig, ein entsprechendes XSLT-Stylesheet zu erstellen, das die Transformation in das gewünschte Zielformat übernimmt. Alle anderen Teile der Web-Applikation bleiben in der Regel unberührt.

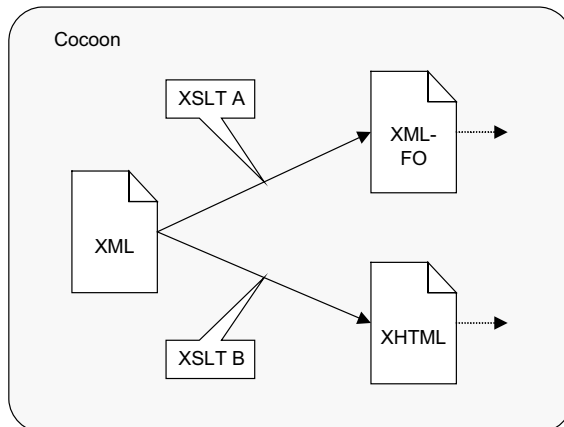


Abbildung 1.8 XSLT transformiert ein XML-Dokument in ein anderes.

Fälschlicherweise wird häufig angenommen, dass die Verwendung von Cocoon mit der Notwendigkeit verbunden ist, eigene XSLT-Stylesheets zu erzeugen und alle Transformationen mit dieser Technik durchführen zu müssen. Dies ist nicht richtig: Cocoon bietet Ihnen zwar die Möglichkeit, XSLT einzusetzen, besitzt jedoch auch zahlreiche andere Möglichkeiten, um Transformationen durchführen und damit vollständig auf XSLT verzichten zu können.

[«]

Mögliche Alternativen, die Cocoon standardmäßig unterstützt, sind beispielsweise die Verwendung von JavaServer Pages (JSP), Velocity oder des JXTemplates. Je nachdem, welche Technik für den jeweiligen Anwendungsfall am sinnvollsten erscheint, ist auch eine Kombination dieser Transformationstechniken auf einfache Weise möglich.

1.5 Zusammenfassung

In diesem Kapitel haben Sie einen Überblick in die Anwendungsgebiete und Funktionsweise von Cocoon erhalten. In den nächsten Kapiteln werden die meisten der hier angesprochenen Themen vertieft. Unter anderem erhalten Sie im nachfolgenden Kapitel einen umfassenden Einblick in XML und dessen zugehörige Techniken. Dies ist nicht nur für Einsteiger auf diesem Gebiet interessant, sondern kann auch als Nachschlagewerk für die wichtigsten XML-Themen dienen, die in den übrigen Teilen dieses Buches immer wieder auftauchen werden.

In diesem Kapitel erfahren Sie, woher Sie Cocoon beziehen, wie Sie es kompilieren und installieren können.

6 Cocoon installieren

Vielleicht haben Sie ja bereits einen Blick auf die Website der Apache Group geworfen, über die Cocoon zum Download angeboten wird und sich gewundert, wieso Sie kein aktuelles Binary¹ finden konnten. Das liegt jedoch nicht an Ihrer Suche, sondern daran, dass Cocoon ab der Version 2.1 lediglich in den Sourcen ausgeliefert wird. Auf den ersten Blick mag dies vielleicht etwas verwundern. Bei näherer Betrachtung macht es jedoch durchaus Sinn, das Framework unkompiliert auszuliefern. Der Grund hierfür sind die sogenannten *Blöcke*.

6.1 Blöcke

Das gesamte Framework Cocoon setzt sich aus dem Core² und einzelnen Blöcken zusammen. Ein Block bildet dabei eine Gruppe von Klassen, Ressourcen und Konfigurationseinstellungen für einen ganz bestimmten Anwendungsbereich. Der Block `cron`, stellt beispielsweise einen Scheduler zur Verfügung, um Tasks in regelmäßigen Intervallen auszuführen. Ein anderer Block `lucene` integriert die Suchmaschine *Lucene*³ in Cocoon. Aufgrund der Tatsache, dass für eine eigene Web-Applikation in aller Regel nur eine kleine Teilmenge der verfügbaren Blöcke benötigt wird, ist es häufig wünschenswert, eine schlanke Cocoon-Installation zu besitzen. Diese Cocoon-Installation sollte nur über diejenigen Dateien und Einträge in den verschiedenen Konfigurationsdateien verfügen, die auch wirklich benötigt werden.

Aus diesem Grund haben sich die Entwickler dafür entschieden, ab der Version 2.1 Cocoon nur noch im Sourcecode auszuliefern. Das Kompilieren muss vom Benutzer selbst durchgeführt werden. Durch eine spezielle

1 Kompilierte Version

2 Hauptkern

3 <http://lucene.apache.org>

Konfigurationsdatei kann er bestimmen, welche Blöcke bei der Kompilierung mit eingebunden werden sollen. Die für den Kompilervorgang benötigten Pakete und Ressourcen werden bereits mitgeliefert, weshalb sich ein solcher Vorgang in den meisten Fällen lediglich auf den Aufruf des entsprechenden Build-Scripts beschränkt und damit nicht wesentlich schwieriger als eine Installation über einen Wizard ist.

Nachfolgend möchte ich Ihnen eine kleine Auswahl häufig verwendeter Blöcke auflisten, um Ihnen einen Eindruck zu vermitteln, welche umfangreichen Möglichkeiten Cocoon bietet.

- ▶ **authentication-fw**
Framework für den Login- bzw. Logout von Benutzern.
- ▶ **batik**
Ermöglicht es, SVG-Dokumente in die Formate JPEG bzw. PNG zu konvertieren.
- ▶ **cron**
Ein Scheduler für die Ausführung von Tasks in beliebigen Intervallen.
- ▶ **databases**
Stellt verschiedene Komponenten für einen Zugriff auf Datenbanken zur Verfügung (z.B. ESQL-Logicsheet und SQL Transformer).
- ▶ **fop**
Ermöglicht unter anderem die Erzeugung von PDF-Dokumenten aus XSL-FO-Dokumenten.
- ▶ **jsp**
Erlaubt es, JSP-Dokumente direkt aus Cocoon heraus auszuführen.
- ▶ **mail**
Stellt verschiedene Komponenten zur Verfügung, um E-Mails zu versenden.
- ▶ **naming**
Ermöglicht den einfachen Zugriff auf einen LDAP-Server.
- ▶ **forms**
Bindet das Formular-Framework Cocoon Forms ein.
- ▶ **slide**
Integriert die WebDav-Implementierung *Slide*⁴ in Cocoon.

4 <http://jakarta.apache.org/slide>

- ▶ **velocity**
Ermöglicht es, die Template-Engine *Velocity*⁵ für die dynamische Erzeugung von Dokumenten zu verwenden.
- ▶ **faces**
Integriert *JavaServer Faces*⁶ (*JSF*) in Cocoon.

Eine vollständige Übersicht über alle verfügbaren Blöcke inklusive deren Abhängigkeiten können Sie in der Datei `blocks.properties` einsehen. Darüber hinaus bietet die Website

<http://wiki.apache.org/cocoon/BlockDescriptions>

eine Beschreibung vieler Blöcke.

6.2 Cocoon besorgen

Sie können sich eine aktuelle Version von Cocoon als gezippte Version von der folgenden Website der Apache Software Foundation laden:

<http://cocoon.apache.org/mirror.cgi>

Da Cocoon in der Programmiersprache Java geschrieben ist, gibt es für alle Betriebssysteme nur ein Archiv. Auf der obigen Website werden Ihnen neben dem aktuellsten Release auch alte Versionen von Cocoon angeboten.

Nachdem Sie sich eine aktuelle Version von Cocoon entweder von der Website oder aber von der Buch-CD besorgt haben, entpacken Sie das Archiv in ein Verzeichnis ihrer Wahl. Wenn Sie anschließend das Verzeichnis `cocoon-2.1.x` öffnen, sollten Sie in etwa folgenden Inhalt vorfinden. [o]

In den nachfolgenden Kapiteln wird häufig auf dieses Verzeichnis Bezug genommen. Aus diesem Grund werde ich zukünftig die Variable `$COCOON_HOME` dazu verwenden, um auf dieses Verzeichnis zu verweisen. Dies unabhängig davon, wo es sich befindet. [«]

⁵ <http://jakarta.apache.org/velocity>

⁶ <http://java.sun.com/javaee/jaserverfaces>

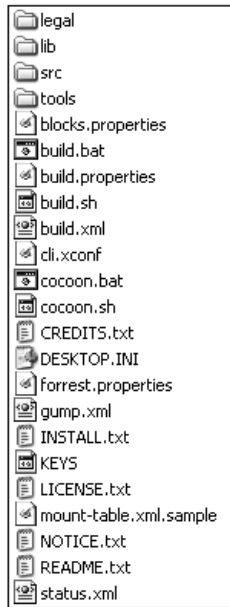


Abbildung 6.1 Der Inhalt des Cocoon-Verzeichnisses

6.3 Cocoon kompilieren

Nachdem Sie die aktuelle Version von Cocoon entpackt haben, können Sie Cocoon nun kompilieren.

6.3.1 Ant

Cocoon wird zusammen mit dem Build-Tool *Apache Ant*⁷ ausgeliefert, wodurch es nicht notwendig ist, dieses Programm von vornherein auf ihrem System installiert zu haben.

Die Konfigurationsdatei `build.xml` von Ant befindet sich ebenfalls in `$COCOON_HOME`. Rufen Sie diese Datei aber nicht direkt auf, sondern verwenden Sie die mitgelieferten Skripte `build.bat` bzw. `build.sh`. Ihre Anwendung wird noch genauer gezeigt.

6.3.2 Kompilierung starten

Um Cocoon mit den Standardeinstellungen zu kompilieren, müssen Sie zuerst in das Verzeichnis `$COCOON_HOME` wechseln. Anschließend müssen

⁷ <http://jakarta.apache.org/ant>

Sie lediglich das zu Ihrem Betriebssystem passende Skript ohne Angabe von Argumenten starten. Für Windows ist dies `build.bat` und für Linux `build.sh`. Neben allen standardmäßig aktivierten Blöcken werden auch die zugehörigen Beispiele kompiliert. Das Ergebnis der Kompilierung wird im Unterverzeichnis `$(COCOON_HOME)/build` abgelegt, das zuvor automatisch erzeugt wurde, falls es noch nicht existierte.

Um stets einen fehlerfreien Build zu erhalten, sollten Sie das Verzeichnis `build`, sofern vorhanden, vor einem Kompilervorgang entweder per Hand löschen oder das Build-Skript mit dem Argument `clean` aufrufen.

Eine Übersicht über alle möglichen Argumente erhalten Sie, indem Sie das Build-Skript mit dem Argument `-projecthelp` aufrufen.

6.3.3 Kompilierung anpassen

Neben einem »Standard-Build« können Sie die Kompilierung auch an eigene Bedürfnisse anpassen. Unter anderem haben Sie beispielsweise die Möglichkeit, nur bestimmte Blöcke einzubinden. Diese Einstellungen können Sie in der Datei `blocks.properties` bzw. `local.blocks.properties` vornehmen.

Vermeiden Sie es, die Datei `blocks.properties` direkt zu editieren, um die Standardeinstellungen ständig verfügbar zu haben. Erzeugen Sie sich stattdessen eine Kopie mit dem Namen `local.blocks.properties` im selben Verzeichnis. Ant sucht zuerst nach einer solchen Konfigurationsdatei. Falls diese nicht gefunden wird, liest es die Datei `blocks.properties` ein.

Je nachdem, welche Blöcke Sie nicht benötigen, können Sie diese vom Build-Vorgang ausschließen, indem Sie die Kommentarzeichen `#` vor den entsprechenden `exclude`-Einträgen entfernen. Achten Sie dabei auf die angegebenen Abhängigkeiten der einzelnen Blöcke untereinander.

Eine weitere Konfigurationsdatei, in der Sie die Kompilierung anpassen können, ist die Datei `build.properties`. Hier haben Sie unter anderem die Möglichkeit, das Erstellen der Apidocs und der Beispiele zu unterbinden.

Editieren Sie auch die Datei `build.properties` niemals direkt, sondern erstellen Sie sich eine Kopie mit dem Namen `local.build.properties`. Ant sucht auch hier zuerst nach einer solchen Konfigurationsdatei. Falls diese nicht gefunden wird, liest es die `build.properties` ein.

6.4 Installieren

Nach der Kompilierung werden innerhalb des Verzeichnisses `build` zwei weitere Unterverzeichnisse angelegt:

- ▶ `cocoon-2.1.x`
- ▶ `webapp`

Im Verzeichnis `cocoon-2.1.x` befinden sich unter anderem alle kompilierten Klassen und Ressourcen, die vom aktuellen Build von Cocoon benötigt werden. In `webapp` hingegen ist Cocoon in Form einer J2EE-Web-Applikation erzeugt worden. Diese Web-Applikation kann in der Regel in jedem Servlet-Container, wie z. B. Tomcat, betrieben werden. In manchen Fällen kann es aber sein, dass zuvor bestimmte Anpassungen nötig sind. Das hängt davon ab, welchen Servlet-Container Sie verwenden und in welche Umgebung dieser wiederum integriert ist. Spezielle Hinweise zur Installation auf den verschiedenen Umgebungen erhalten Sie auf folgender Website:

<http://cocoon.apache.org/2.1/installing>

6.4.1 Cocoon starten mit Jetty

Die einfachste Möglichkeit, um Cocoon als Web-Applikation zu starten, besteht darin, in das Verzeichnis `$COCOON_HOME` zu wechseln und dort entsprechend ihrem Betriebssystem das Skript `cocoon.bat` bzw. `cocoon.sh` mit dem Argument `servlet` aufzurufen:

```
>cd cocon 2.1.x
>cocoon.bat servlet
```

Durch diesen Aufruf wird der Servlet-Container *Jetty*⁸, der mit Cocoon ausgeliefert wird, gestartet. Dieser Servlet-Container ist bereits so konfiguriert, dass er die Cocoon-Web-Applikation in `$COCOON_HOME/build/webapp` startet. Anschließend können Sie durch einen Aufruf des URL `http://localhost:8888` in Ihrem Web-Browser testen, ob Cocoon einwandfrei läuft. In diesem Fall sollte Ihnen eine Willkommenseite wie in Abbildung 6.2 angezeigt werden.

Der Betrieb von Cocoon mit Jetty ist vor allem dann äußerst hilfreich, wenn Sie sich einen ersten Einblick in die Möglichkeiten von Cocoon durch Ansicht der mitgelieferten Beispiele verschaffen wollen. Ebenso,

8 <http://jetty.mortbay.org/jetty/index.html>

wenn Sie Cocoon im Offline-Modus betreiben und eine einfache Debug-Möglichkeit benötigen.

Für einen Produktivbetrieb ist dieses Vorgehen allerdings weniger zu empfehlen. In diesem Fall sollten Sie beispielsweise die Installation in Tomcat vorziehen, wie im folgenden Abschnitt erklärt wird.⁹

6.4.2 In Tomcat installieren

Falls Sie Tomcat verwenden möchten, müssen Sie im Normalfall keinerlei Anpassungen für die Installation vornehmen. Kopieren Sie sich das Verzeichnis `webapp` nach `$CATALINA_HOME/webapps` und benennen Sie es beispielsweise in `cocoon` um. Je nachdem, wie Ihr Servlet-Container konfiguriert ist, kann es sein, dass Sie ihn neu starten müssen, um die Web-Applikation zu aktivieren. Anschließend können Sie testen, ob Cocoon lauffähig ist, indem Sie in Ihrem Browser folgenden URL aufrufen:

```
http://localhost:8080/cocoon
```

Je nach Rechner, auf dem sich Ihr Servlet-Container befindet, und dessen Konfiguration können der Hostname und die Portnummer von diesem Aufruf abweichen.

Im Rest dieses Buches werde ich jedoch immer die Adresse `http://localhost:8080/cocoon` verwenden, um auf die Web-Applikation Cocoon per URL zu verweisen. Bitte denken Sie daran und passen Sie diese eventuell entsprechend an, falls Sie eine andere Konfiguration besitzen oder beispielsweise Jetty verwenden.

[«]

Falls Sie nach einem Aufruf des oben genannten URIs eine Willkommenseite wie die nachfolgend gezeigte erhalten, ist Cocoon korrekt installiert.

Unter dem Link `samples` können Sie zahlreiche Beispiele für die Verwendung von Cocoon und der verschiedenen Blöcken finden. Falls Sie an den Quellen der einzelnen Beispiele interessiert sind, können Sie diese im Verzeichnis `$COCOON/samples` finden. Voraussetzung ist allerdings, dass Sie die Einbindung von Beispielen vor dem Build von Cocoon in der Datei `local.build.properties` aktiviert haben.

Beispiele

⁹ Natürlich kommt auch Jetty für einen Produktiveinsatz in Frage. In diesem Fall sollten allerdings entsprechende Einstellungen vorgenommen werden, die Sie der offiziellen Dokumentation von Jetty entnehmen können.

6 | Cocoon installieren



Abbildung 6.2 Die Willkommenseite der Cocoon-Installation

Hinweis Im Rest dieses Buches werde ich immer die Bezeichnung `$COCOON` verwenden, um auf das Basisverzeichnis der Web-Applikation Cocoon zu verweisen, die sie soeben installiert haben.

*In diesem Kapitel erfahren Sie, wie Sie Cocoon unterstützt,
Formulare zu bearbeiten.*

11 Cocoon Forms

Interaktionen mit dem Benutzer über HTML-Formulare sind ein wesentlicher Bestandteil dynamischer Webseiten. In der Regel werden Informationen vom Benutzer über HTML-Formulare abgefragt, um diese anschließend in der zugehörigen Web-Applikation auszuwerten. Die Art und Weise, wie Informationen für den Benutzer in einem solchen HTML-Formular zur Verfügung gestellt und die eingegebenen Daten wiederum abgefragt werden, ist in den weitaus meisten Fällen gleich. Dabei wirken sich die wiederkehrenden gleichartigen Aufgaben für das Publizieren von Daten in einem Formular häufig negativ auf die Projektentwicklung aus, da diese im Verhältnis zu ihrem Nutzen relativ aufwändig sind und eine zusätzliche Fehlerquelle darstellen können.

Stellen Sie sich beispielsweise ein Mehrfachauswahlfeld vor, von dem einige Einträge vorselektiert sein sollen. Die Implementierung einer solchen Vorselektierung geschieht häufig, indem neben der Gesamtliste in der sich alle anzuzeigenden Elemente befinden, zusätzlich eine Selektionsliste angegeben wird, in der die zu selektierenden Elemente stehen. Durch eine Iteration über die Gesamtliste und einem Vergleich mit der Selektionsliste kann entschieden werden, ob das aktuelle Element im Mehrfachauswahlfeld vorselektiert werden soll. In Listing 11.1 sehen Sie ein Beispiel, wie eine solche Vorselektion in einem JX-Template realisiert werden kann.

Beispiel für
eine manuelle
Vorselektierung



```
<select name="cars">
<jx:forEach var="car" items="\${allCars}">
  <jx:forEach var="selCar" items="\${selectedCars}">
    <jx:choose>
      <jx:when test="\${car eq selCar}">
        <option selected="selected">\${car}</option>
      <jx:when>
      <jx:otherwise>
        <option>\${car}</option>
      </jx:otherwise>
    </jx:choose>
  </jx:forEach>
</select>
```

```

        </jx:choose>
    </jx:forEach>
</jx:forEach>
</select>

```

Listing 11.1 Vorselektierung in einem Mehrfachauswahlfeld

Diese einfache Aufgabe, bestimmte Werte in einer Liste vorzuselektieren, ist mit vergleichsweise viel Aufwand verbunden. Dabei wurde in diesem Beispiel noch nicht einmal auf Validierungs- und Konvertierungsmaßnahmen eingegangen. Bei größeren und komplexeren Formularen kann ein solches Vorgehen schnell zu einer mühevollen Arbeit werden, die die Layout- als auch Controller-Logik zudem unnötig verkompliziert.

11.1 Wichtige Eigenschaften von Formular-Frameworks

Aktuelle Formular-Frameworks setzen genau an dieser Stelle an. Sie stellen Mechanismen zur Verfügung, die hier genannten negativen Punkte zu umgehen und dem Entwickler einen großen Teil der ursprünglich anfallenden Arbeit abzunehmen. Dadurch wird es möglich, sich auf die eigentliche Applikationslogik zu konzentrieren. Darüber hinaus wird die Applikation selbst übersichtlicher und somit leichter wartbar. Generell erfüllen die meisten Formular-Frameworks folgende Aufgaben:

- ▶ Zusammenfassen der Formulardaten in einer zentralen Datenstruktur.
- ▶ Konvertierung der eingegebenen Formulardaten in den erwarteten Datentyp (z. B. `Date`) und zurück.
- ▶ Automatisches Vorselektieren der Eingabefelder anhand der übergebenen Daten.
- ▶ Validierung der Benutzereingaben.
- ▶ Anbieten von erweiterten Eingabefeldern (z. B. `Date-Picker`).
- ▶ Automatisches Befüllen und Laden einer benutzerdefinierten Datenstruktur (`Binding`).
- ▶ Event-Handling

Auch Cocoon stellt ein solches Formular-Framework unter dem Namen *Cocoon Forms* (kurz.: *CForms*) zur Verfügung. Dabei handelt es sich um ein noch relativ junges aber sehr mächtiges Framework, das ausnahmslos alle der oben beschriebenen Features unterstützt. Vor Cocoon Forms

existierten mehrere unterschiedliche Formular-Frameworks in Cocoon, die teilweise deutlich verschiedene Ansätze verfolgten.

So gab es beispielsweise das Formular-Framework *XMLForm*, das an die W3C-Spezifikation *XForms*¹ angelehnt war, die allerdings nicht vollständig implementiert wurde. Etwas später entstand *JXForms*, das die Verarbeitung von Formularen in Cocoon schon deutlich vereinfachte, allerdings noch nicht weit genug ging.

XMLForm
JXForms

Um nun die Vorteile der jeweiligen Formular-Frameworks zu vereinigen und ein einheitliches Vorgehen festzulegen, entstand zunächst *Woody*. Dabei handelte es sich um die Alpha- und Betaversion von Cocoon Forms, in der die einzelnen Konzepte implementiert und erprobt wurden. In der Cocoon-Version 2.1.5 wurde Woody schließlich nach Cocoon Forms umbenannt und hat seitdem alle anderen Formular-Frameworks in Cocoon abgelöst. Ein Einsatz eines anderen Formular-Frameworks ist somit nicht mehr zu empfehlen und wird von Cocoon zukünftig auch nicht mehr offiziell unterstützt.

Woody wird zu
Cocoon Forms

In diesem Kapitel möchte ich Ihnen einen umfassenden Einblick in das leistungsfähige Formular-Framework Cocoon Forms geben. Dabei werden Sie alle wesentlichen Aspekte kennen lernen, die notwendig sind, um Cocoon Forms professionell einzusetzen. Da sich Cocoon Forms in einem rasanten Tempo weiterentwickelt und fast mit jedem Release von Cocoon auch neue Features in Cocoon Forms integriert werden, ist es leider nicht möglich, alle diese Features erschöpfend hier zu behandeln. Stattdessen möchte ich Ihnen hier einen relativ tiefen Einblick in die wichtigsten Aspekte von Cocoon Forms geben. Mit diesem Wissen sollten Sie anschließend in der Lage sein, weitere Features von Cocoon Forms, die hier nicht behandelt wurden, leicht zu verstehen und anzuwenden.

11.2 Grundsätzliche Funktionsweise

Von entscheidender Bedeutung, um mit Cocoon Forms arbeiten zu können, ist zunächst die Kenntnis der grundsätzlichen Funktionsweise dieses Formular-Frameworks und dessen Grundbegriffe. Ist ein solcher *Form-Publishing-Prozess* einmal verstanden, ist es leicht, Anpassungen an den verschiedenen Stellen vorzunehmen. Dabei wird vorausgesetzt, dass Sie mit den Grundlagen der Pipeline-Verarbeitung in Cocoon vertraut sind

¹ <http://www.w3.org/MarkUp/Forms>

und die grundsätzliche Funktionsweise von HTTP-Formularen verstehen. Sehen Sie sich hierzu zunächst die nachfolgende Abbildung an.

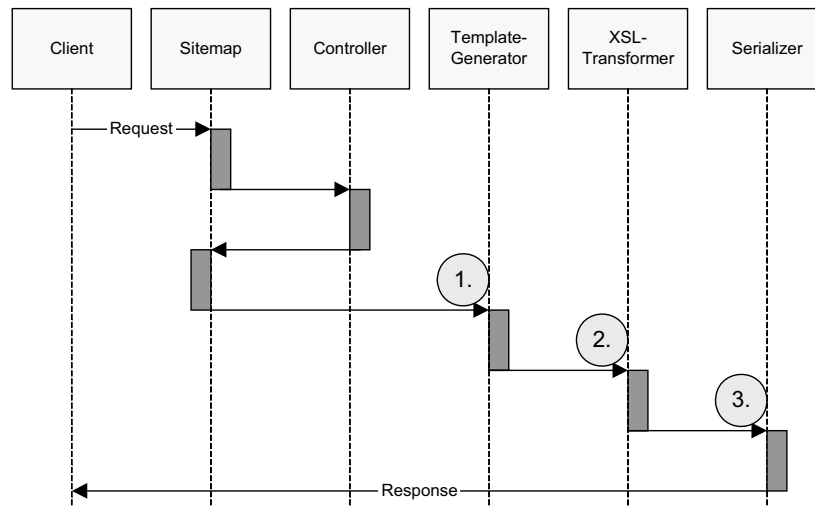
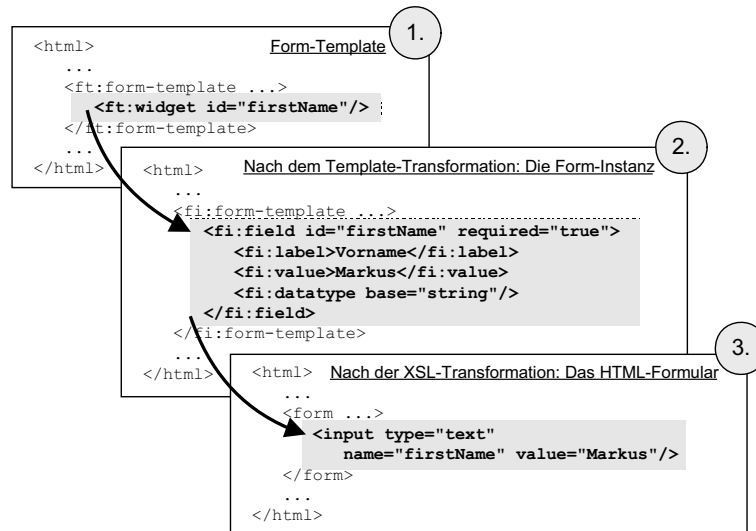


Abbildung 11.1 Die einzelnen Schritte des Form-Publishings²

In dieser Abbildung wird die grundsätzliche Funktionsweise des Publishing-Prozesses für ein Formular gezeigt. Der Client schickt zunächst eine Anfrage für eine Seite, auf der ein Formular dargestellt werden soll.

² Teile der Darstellung nach einer Idee von <http://cocoon.apache.org/2.1/userdocs/publishing/templating.html>

Diese Anfrage trifft auf der Sitemap ein und wird dort an einen entsprechend registrierten Controller delegiert.

Ein *Controller* ist in der Regel eine benutzerdefinierte Java-Klasse (z. B. Action) oder ein Flowscript. In ihm werden zum einen die Daten ermittelt, die im Formular angezeigt werden sollen und zum anderen wird darin geeignet auf Benutzereingaben reagiert.

Controller

Darüber hinaus wird im Controller ein *Form-Objekt* erstellt, indem die sogenannte *Form-Definition* eingelesen wird. Diese Form-Definition kann als eine Art zentrale Konfigurationsdatei für ein Formular angesehen werden, in der das sogenannte *Form-Model*, also die Struktur und die Eigenschaften des Formulars, festgelegt sind. Das erstellte Form-Objekt enthält zudem alle Daten, die dem Benutzer im Formular angezeigt werden sollen. Diese Daten können zuvor beispielsweise statisch direkt in der Form-Definition oder dynamisch durch den Controller über den Aufruf bestimmter Methoden gesetzt werden.³

Form-Objekt
Form-Definition
Form-Model

Als nächstes ruft der Controller explizit eine Pipeline auf, in der zunächst das sogenannte *Form-Template* verarbeitet wird. Dieses Form-Template ist in der Regel ein JX-Template und legt durch spezielle Form-Template-Elemente fest, an welchen Stellen letztendlich die einzelnen Formularfelder stehen sollen. Ist also die Form-Definition für die Abbildung der Formular-Struktur zuständig, so wird im Form-Template das Layout des Formulars in einem XML-Dokument bestimmt. In Abbildung 11.1 ist ein solches Form-Template durch **1.** dargestellt.

Form-Template

```
...
<ft:widget id="firstName"/>
...
```

Listing 11.2 Beispiel für die Positionierung eines Formularfeldes mit `<ft:widget/>` in einem Form-Template

Beim Ausführen wird dann das JX-Template mit dem darin definierten Form-Template »geparst« und eine sogenannte *Form-Instanz*⁴ erzeugt, in der unter anderem anstelle der vorbestimmten Positionen `<ft:widget/>`

Form-Instanz

³ Eine weitere Möglichkeit, Daten im Formular zu setzen, besteht in der Verwendung des Binding-Frameworks, das Sie in Abschnitt 11.10, »Binding-Framework«, kennenlernen.

⁴ Die Form-Instanz kann in gewisser Weise als XML-Repräsentation des Form-Objekts betrachtet werden; auch wenn diese beiden Begriffe grundsätzlich nicht klar voneinander getrennt sind.

für die Formularfelder spezielle Instanz-Elemente gesetzt werden. In Abbildung 11.1 durch **2.** dargestellt.

```
...
<fi:field id="firstName" required="true">
  <fi:label>Vorname</fi:label>
  <fi:value>Markus</fi:value>
  <fi:datatype type="string"/>
</fi:field>
...
```

Listing 11.3 Aus `<ft:widget/>` wird beispielsweise `<fi:field/>`

XSL-Transformation

Im nächsten Schritt der Pipeline wird das Form-Template mit der darin enthaltenen Form-Instanz mithilfe einer XSL-Transformation in das gewünschte Zielformat – in der Regel HTML – transformiert. Damit werden unter anderem die Instanz-Elemente in reguläre HTML-Elemente transformiert; in Abbildung 11.1 durch **3.** dargestellt. Dieser Zwischenschritt ist vor allem deshalb sinnvoll, da somit die vorhandene Form-Instanz in jedes beliebige Format transformiert werden kann und nicht zwingend an HTML-Formulare gebunden ist, wobei dies das mit großem Abstand häufigste Einsatzgebiet ist.

```
...
<input type="text" name="firstName" value="Markus" >
...
```

Listing 11.4 Beispiel eines aus `<fi:field/>` erzeugten HTML-Eingabefelds

Als letztes wird das so erzeugte (HTML-)Dokument mithilfe eines geeigneten Serializers an den Client versendet.

In Abbildung 11.2 wird nun zusammengefasst dargestellt, welche Komponente welches Form-Publishing-Dokument verarbeitet bzw. benötigt.

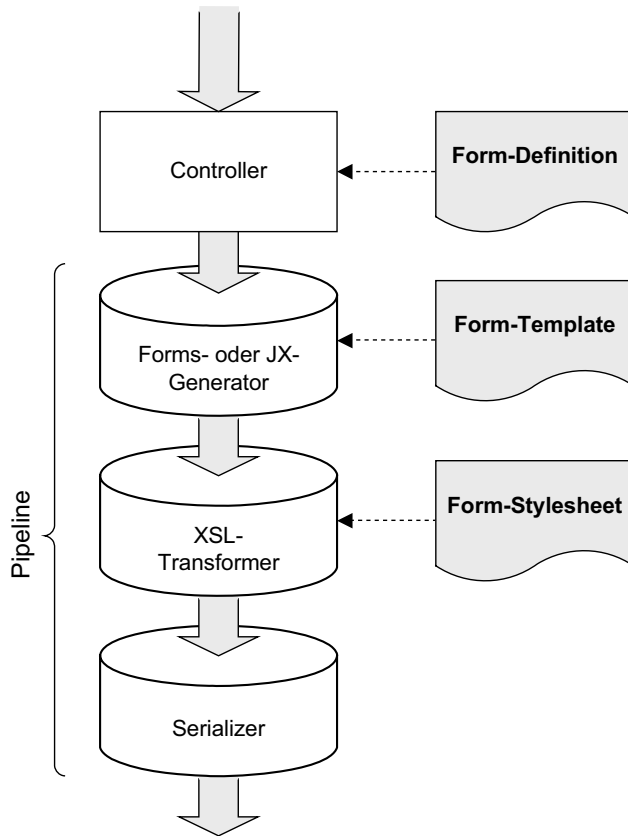


Abbildung 11.2 Die benötigten Form-Publishing-Dokumente

11.3 Ein kleines Beispiel

Um die zuvor beschriebene grundsätzliche Funktionsweise von Cocoon Forms weiter zu verdeutlichen, möchte ich Ihnen an dieser Stelle wieder ein kleines Beispiel vorstellen, das auf dem einführenden Beispiel der Cocoon Websites basiert⁵. Hier wird ein einfaches Formular durch den Benutzer ausgefüllt und mithilfe von Cocoon Forms validiert. Die hier vorgestellte Vorgehensweise ist in der Regel immer gleich und gibt Ihnen einen ersten Überblick über die grundsätzlichen Schritte für die Erstellung eines Formulars mit Cocoon Forms. Falls Ihnen das eine oder andere in diesem Ablauf noch nicht klar sein sollte, lesen Sie einfach wei-

⁵ <http://cocoon.apache.org/2.1/userdocs/basics/sample.html>

ter, da jeder einzelne Schritt in den nachfolgenden Abschnitten ausführlich erklärt wird.

Überprüfen Sie zunächst, ob Sie den Block `forms` beim Kompilieren von Cocoon eingebunden haben. Dieser enthält alle notwendigen Klassen und weiteren Dateien für die Verwendung von Cocoon Forms.

Ziel dieses Beispiels soll ein HTML-Formular sein, durch das sich ein Benutzer durch Eingabe seines Vor- und Nachnamen sowie seiner E-Mail-Adresse registrieren kann. Die einzelnen Schritte dieses Beispiels lassen sich folgendermaßen unterteilen, wobei die für Cocoon Forms relevanten Schritte fett hervorgehoben sind:

- ▶ Erstellen der Sub-Applikation
- ▶ **Erstellen der Form-Definition**
- ▶ **Erstellen des Form-Templates**
- ▶ **Erstellen des Controllers (Flowscript)**
- ▶ Erstellen der Bestätigungsseite
- ▶ **Erstellen der nötigen Sitemap-Einträge**

Das zu erzeugende Formular soll ein Aussehen erhalten, wie in Abbildung 11.3 gezeigt.



Abbildung 11.3 Das Anmelde-Formular

Nachdem alle Daten korrekt eingegeben sind, wird anschließend noch eine Bestätigungsseite ausgegeben, die folgendes Aussehen besitzt.

- [o] Dieses Beispiel finden Sie auch auf der Buch-CD unter dem Namen `forms-with-flowscript`.



Abbildung 11.4 Bestätigungsseite

11.3.1 Erstellen der Sub-Applikation

Der erste Schritt, die Erstellung einer Sub-Applikation, ist zwar nicht unbedingt für die Verwendung von Cocoon Forms notwendig, erweist sich aber vor allem dann als sinnvoll, wenn das zu erstellende Formular von verschiedenen Punkten einer Web-Applikation aus verlinkt werden soll. Darüber hinaus macht es dieses Beispiel deutlich übersichtlicher. Erstellen Sie hierfür folgende Verzeichnisstruktur innerhalb von \$COCOON:

```
forms-with-flowscript
  /flows
  /forms-configs
  /stylesheets
  /templates
```

Listing 11.5 Verzeichnisstruktur der Beispielanwendung

11.3.2 Erstellen der Form-Definition

Als nächstes wird die sogenannte Form-Definition erstellt. In dieser XML-Datei werden die einzelnen Formularfelder – in Cocoon Forms *Widgets* genannt – eines Formulars definiert und konfiguriert.

Erzeugen Sie hierfür im Verzeichnis `forms-configs` ein neues XML-Dokument mit dem Namen `register.fd`⁶ und folgendem Inhalt:

```
<?xml version="1.0"?>
<fd:form
```

⁶ Generell hat es sich als sinnvoll erwiesen, Form-Definition-Dateien mit der Endung `*.fd` oder auch `*.fd.xml` zu versehen.

```

xmlns:fd="http://apache.org/cocoon/forms/1.0#definition">

<fd:widgets>
  <fd:field id="firstName" required="true">
    <fd:label>Vorname</fd:label>
    <fd:datatype base="string"/>
    <fd:validation>
      <fd:length min="2">
        <fd:failmessage>
          Der Vorname muss aus mindestens 2
          Zeichen bestehen
        </fd:failmessage>
      </fd:length>
    </fd:validation>
  </fd:field>

  <fd:field id="lastName" required="true">
    <fd:label>Nachname</fd:label>
    <fd:datatype base="string"/>
    <fd:validation>
      <fd:length min="2">
        <fd:failmessage>
          Der Nachname muss aus mindestens 2
          Zeichen bestehen
        </fd:failmessage>
      </fd:length>
    </fd:validation>
  </fd:field>

  <fd:field id="email" required="true">
    <fd:label>Email</fd:label>
    <fd:datatype base="string"/>
    <fd:validation>
      <fd:email>
        <fd:failmessage>
          Bitte geben Sie eine gueltige
          Email-Adresse ein
        </fd:failmessage>
      </fd:email>
    </fd:validation>
  </fd:field>

</fd:widgets>
</fd:form>

```

Listing 11.6 Die Form-Definition register.fd

Wie Sie in Listing 11.6 sehen, werden in der Form-Definition die Widgets (Formularfelder) `firstName`, `lastName` und `email` definiert. Alle sind Pflichtfelder, was durch Angabe des Attributs `required="true"` erreicht wird. Darüber hinaus besitzt jedes Widget ein so genanntes Label. Dies ist der Name des Widgets, der dem Benutzer angezeigt werden soll. Der Datentyp aller Widgets ist in diesem Beispiel `string`, wodurch keine Konvertierung erforderlich wird. Zusätzlich besitzt jedes Widget noch eine Validierungsregel. In den ersten beiden Fällen `firstName` und `lastName` soll somit lediglich sichergestellt werden, dass der eingegebene Name mehr als zwei Zeichen enthält. Im Fall von `email` hingegen soll überprüft werden, ob es sich bei dem eingegebenen Wert tatsächlich um eine plausible E-Mail-Adresse handelt.

Neben den hier verwendeten Validierungsregeln gibt es in Cocoon Forms noch weitere solcher Regeln, die z. B. die Überprüfung mithilfe eines Regulären Ausdrucks oder den Vergleich zweier Widget-Werte ermöglichen. Falls dies nicht ausreicht, ist es auch möglich, relativ einfach einen eigenen Validator zu programmieren, zu registrieren und anschließend zu verwenden, wie Sie in Abschnitt 11.8, »Validierung«, erfahren.

11.3.3 Erstellen des Form-Templates

Als nächsten Schritt ist es notwendig, das Form-Template zu erstellen. Hierbei handelt es sich in der Regel um ein reguläres JX-Template, das bestimmt, an welche Positionen die einzelnen Widgets aus der Form-Definition platziert werden sollen. Erzeugen Sie hierfür eine Datei im Verzeichnis `templates` mit dem Namen `form.jxt` und fügen Sie ihr folgenden Inhalt hinzu:

```
<html xmlns:ft="http://apache.org/cocoon/forms/1.0#template"
  xmlns:jx="http://apache.org/cocoon/templates/jx/1.0">

  <jx:import uri="resource://org/apache/cocoon/
    forms/generation/jx-macros.xml"/>

  <head>
    <title>Anmeldung</title>
  </head>
  <body>

    <ft:form-template
      action="{continuation.id}.cont" method="POST">
```

```

<table>
<tr>
  <td><ft:widget-label id="firstName"/></td>
  <td><ft:widget id="firstName"/></td>
</tr>
<tr>
  <td><ft:widget-label id="lastName"/></td>
  <td><ft:widget id="lastName"/></td>
</tr>
<tr>
  <td><ft:widget-label id="email"/></td>
  <td><ft:widget id="email"/></td>
</tr>
<tr>
  <td colspan="2">
    <input type="submit" value="Anmelden"/>
  </td>
</tr>
</table>
</ft:form-template>

</body>
</html>

```

Listing 11.7 Das Form-Template form.jxt

Wie Sie sehen, ist das JX-Template einem HTML-Dokument sehr ähnlich. Die einzige Ausnahme stellt der Formularbereich – eingeleitet durch das Element `<ft:form-template/>` – dar. Darin werden die einzelnen Widgets aus der Form-Definition über ihre eindeutige Id und den Elementen `<ft:widget/>` und `<ft:widget-label/>` referenziert. Die Auswertung dieser Elemente erfolgt durch JX-Macros, die durch die Zeile

```

<jx:import uri="resource://org/apache/cocoon/forms/
  generation/jx-macros.xml"/>

```

in das Template eingebunden werden.

- [»]** Alternativ zur Verwendung des JXTemplate Generators/Transformers und eines JX-Templates können Sie auch den Forms Generator/Transformer verwenden, der im Grunde dieselben Aufgaben erfüllt, allerdings weitaus weniger flexibel ist. Die Cocoon-Entwickler empfehlen ausdrücklich die Verwendung der JXTemplate-Lösung, weswegen in den nachfolgenden Abschnitten der Fokus hierauf gelegt wird.

11.3.4 Erstellen des Controllers (Flowscript)

Um den korrekt registrierten Benutzer weiterleiten bzw. eventuell weitere Aktionen auszuführen, erzeugen wir als nächstes einen Controller in Form eines Flowscripts. Wie bereits erwähnt, muss der Controller nicht zwangsläufig ein Flowscript sein. Es kann sich auch um eine Java-Klasse handeln, wie in Abschnitt 11.11, »Cocoon Forms ohne Flowscript«, am Beispiel einer Action gezeigt wird. Hier wird allerdings ein Flowscript verwendet, da dies der komfortablere Weg ist. Erstellen Sie deshalb im Verzeichnis `flows` eine Datei mit dem Namen `register.js` und fügen Sie ihr folgende Zeilen hinzu:

```
cocoon.load("resource://org/apache/cocoon/forms/flow/javascript/Form.js");

function doRegister() {

    // Form Objekt erzeugen
    var form = new Form("forms-configs/register.fd");

    // Form an den Benutzer "senden"
    form.showForm("registerPipeline");

    // Formulardaten "holen"
    var model = form.getModel();

    // Bestaetigung "senden"
    cocoon.sendPage("successPipeline", {"user":model});
}
```

Listing 11.8 Das Flowscript `register.js` für die Formularbearbeitung

In diesem Flowscript wird in der ersten Zeile zunächst ein Import durchgeführt, der das Flowscript `Form.js` aus dem Klassenpfad einbindet, um bestimmte Objekte und Methoden zur Verfügung zu stellen. Innerhalb der Funktion `doRegister` wird anschließend ein Form-Objekt erzeugt, indem der Pfad zur Form-Definition `register.fd` übergeben wird. Dieser Pfad wird intern durch den Source-Resolver interpretiert und kann somit alle von Cocoon unterstützten Protokollschemata beinhalten. Im Fall von `forms-configs/register.fd` wird dieser Pfad relativ zur aktuellen Sitemap interpretiert. Nachdem das Form-Objekt erzeugt wurde, wird die Funktion `showForm("registerPipeline")` aufgerufen. Dies hat zur Folge, dass die Kontrolle zurück an eine Pipeline mit dem Namen `registerPipeline` übergeben wird. Diese Pipeline ist dafür verantwort-

lich, das Formular zu generieren, das dem Benutzer angezeigt werden soll und es anschließend zu versenden. Nachdem der Benutzer das Formular korrekt ausgefüllt hat, werden seine Daten durch die Methode `getModel` entgegengenommen und an die Bestätigungsseite unter dem Schlüssel `user` übergeben, die im Anschluss darauf durch den Aufruf der Pipeline `successPipeline` erzeugt und versendet wird.

Hier wird deutlich, wie bequem die Verwendung von Flowscripts mit Continuations sein kann, da keine Überprüfung erfolgen muss, ob dies der erste oder ein folgender Aufruf des Formulars ist. Aus diesem Grund können die Daten direkt nach dem Versenden des Formulars durch `showForm` mit `getModel` »abgegriffen« werden. Der Algorithmus kann sehr einfach gehalten werden, wodurch der Kontrollfluss auf den ersten Blick ersichtlich ist.

11.3.5 Erstellen der Bestätigungsseite

Zusätzlich sollten Sie noch eine einfache Bestätigungsseite anlegen, in der dem Benutzer die erfolgreiche – fiktive – Registrierung mitgeteilt wird. Erstellen Sie hierfür innerhalb von `templates` eine neue Datei mit dem Namen `success.jxt` und fügen Sie ihr folgende Einträge hinzu:

```
<html xmlns:jx="http://apache.org/cocoon/templates/jx/1.0">
<head>
  <title>Anmeldung erfolgreich!</title>
</head>
<body>
  Herzlichen Glückwunsch
  <b>${user.firstName} ${user.lastName}</b>
  <br/>
  Ihre Daten wurden erfolgreich gespeichert!
  <br/><br/>
  <i>Vorname: ${user.firstName}</i><br/>
  <i>Nachname: ${user.lastName}</i><br/>
  <i>Email: ${user.email}</i><br/>
  <a href="../forms-with-flowscript">Nochmal</a>
</body>
</html>
```

Listing 11.9 Die Bestätigungsseite `success.jxt`

Die durch den Benutzer angegebenen Daten werden im Flowscript unter dem Schlüssel `user` an die Bestätigungsseite übergeben und können anschließend darin ausgelesen und angezeigt werden.

11.3.6 Erstellen der Sitemap-Einträge

Zuletzt müssen Sie noch die entsprechenden Einträge in der Sitemap vornehmen. Erstellen Sie hierfür innerhalb von `forms-with-flowscript` eine Datei `sitemap.xmap` und fügen Sie ihr folgende Zeilen hinzu:

```
<?xml version="1.0" encoding="UTF-8"?>

<map:sitemap
  xmlns:map="http://apache.org/cocoon/sitemap/1.0">

  <map:flow language="javascript">
    <map:script src="flows/register.js"/>
  </map:flow>

  <map:pipelines>
    <map:pipeline>

      <!-- A -->
      <map:match pattern="">
        <map:call function="doRegister"/>
      </map:match>

      <!-- B -->
      <map:match pattern="registerPipeline">
        <map:generate type="jx"
          src="templates/form.jxt"/>
        <map:transform type="xslt"
          src="stylesheets/forms-samples-styling.xsl"/>
        <map:serialize type="html"/>
      </map:match>

      <!-- C -->
      <map:match pattern="resources/*/**">
        <map:read
          src="resource://org/apache/cocoon/{1}/
            resources/{2}"/>
      </map:match>

      <!-- D -->
      <map:match pattern="*.cont">
        <map:call continuation="{1}"/>
      </map:match>

      <!-- E -->
      <map:match pattern="successPipeline">
```

```

        <map:generate type="jx" ↗
            src="templates/success.jxt"/>
        <map:serialize type="html"/>
    </map:match>

    </map:pipeline>
</map:pipelines>
</map:sitemap>

```

Listing 11.10 Die Sitemap des Beispiels forms-flowscript

Zunächst wird in der Sitemap das Flowscript `register.js` auf die übliche Weise registriert.

- A Anschließend wird mit der Pipeline A der Einstieg in die Formularverarbeitung festgelegt, indem auf die Funktion `doRegister` des Flowscripts `form.js` und somit dem Controller verwiesen wird.
- B Der Controller selbst ruft anschließend die Pipeline B auf, in der das Layout des Formulars erstellt wird. Als Erstes wird hier der JXTemplate Generator verwendet, um das Template `form.jxt` einzulesen und zu verarbeiten. Anschließend erfolgt eine XSL-Transformation mit dem Stylesheet `forms-samples-styling.xsl`, um das Form-Template in ein HTML-Formular zu transformieren. Dieses Stylesheet wird durch Cocoon Forms zur Verfügung gestellt und muss zuvor noch nach `stylesheets` kopiert werden. Sie finden dieses Stylesheet beispielsweise im gleichnamigen Beispiel auf der Buch-CD oder in der Cocoon Distribution unter folgendem Pfad:

```
src/blocks/forms/samples/resources/forms-samples-styling.xsl
```

Falls Sie das Layout oder dessen Funktionsumfang anpassen möchten, kann dieses Stylesheet als ideale Vorlage dienen.

- C Die Pipeline C stellt verschiedene Ressourcen, wie CSS-, JavaScript- und Bild-Dateien zur Verfügung, die von Cocoon Forms benötigt werden. Hierzu gehören beispielsweise CSS-Stylesheets für das Calendar-Widget oder das »Hilfe«-Icon.
- D Die Pipeline D dient lediglich für das Fortsetzen einer Continuation anhand der Id.
- E Falls der Benutzer alle Formulardaten korrekt eingegeben hat, ruft der Controller als letztes die Pipeline E auf, in der lediglich das JXTemplate `success.jxt` eingelesen und darin die entsprechenden Werte platziert werden. Anschließend wird das Resultat an den Client gesendet.

11.3.7 Ausführen des Beispiels

Um dieses Beispiel zu testen, müssen Sie zunächst Ihren Servlet-Container starten und anschließend folgenden URI in Ihren Web-Browser eingeben:

```
http://localhost:8080/cocoon/forms-with-flowscript/
```

Nachdem Sie anhand dieses Beispiels einen Einblick erhalten haben, wie Cocoon Forms eingesetzt werden, werden in den nachfolgenden Abschnitten nun die einzelnen Themen vertieft.

11.4 Form-Definition

Die Form-Definition kann als die zentrale Konfigurationsdatei zur Definition eines Formulars angesehen werden. In ihr wird das Form-Model, also die Struktur des Formulars, festgelegt. Es enthält keinerlei Präsentationsdaten. Mit Hilfe der Form-Definition kann später ein Form-Objekt erzeugt werden, das alle Daten des Formulars enthält. Die Form-Definition kann auch als eine Klasse verstanden werden und das Form-Objekt als ein Objekt davon. Auf die Eigenschaften dieses Formulars kann später über das Form-Objekt zugegriffen werden. Dies kann wahlweise in einem Flowscript oder einer Java-Klasse geschehen. Unter anderem können folgende Festlegungen in einer Form-Definition erfolgen:

- ▶ Die benötigten Widgets
- ▶ Der Datentyp eines Widgets
- ▶ Eventuelle Konvertierungsregeln
- ▶ Validierungsregeln
- ▶ Event-Definitionen

Im nachfolgenden Listing sehen Sie ein Beispiel einer Form-Definition, die ein einfaches Login-Formular beschreibt.

```
<?xml version="1.0"?>
<fd:form
  xmlns:fd="http://apache.org/cocoon/forms/1.0#definition">

  <fd:widgets>
    <fd:field id="username" required="true">
      <fd:label>Username</fd:label>
      <fd:datatype base="string"/>
    </fd:field>
```

14.9 Hibernate integrieren und verwenden

Das Thema OR-Mapping⁵ – also die Abbildung von Objekten auf eine relationale Struktur und zurück – ist aus der Entwicklung moderner, objektorientierter Web-Applikationen nicht mehr wegzudenken. Dasjenige Tool im Open Source-Bereich, welches zur Zeit die Nummer eins bezogen auf seine Verbreitung darstellt, ist *Hibernate*⁶, das von der JBoss Open Source Federation betreut wird. Um dieser Entwicklung Rechnung zu tragen, möchte ich in diesem Abschnitt darauf eingehen, wie Hibernate in Cocoon integriert und anschließend verwendet werden kann. Natürlich muss dabei ein grundlegendes Verständnis von Hibernate sowie der Erstellung eigener Komponenten in Cocoon vorausgesetzt werden.

14.9.1 Hibernate besorgen und installieren

Der erste Schritt besteht darin, eine aktuelle Version 3.x von Hibernate von der Website <http://www.hibernate.org> herunter zu laden und zu entpacken. Anschließend müssen die einzelnen Jar-Pakete dem Klassenpfad der Web-Applikation hinzugefügt werden, indem sie am einfachsten in das Verzeichnis `$COCOON/WEB-INF/lib` kopiert oder in der IDE entsprechend »gemountet« werden. Welche Jar-Pakete das sind, hängt davon ab, welche Funktionalitäten Sie in Hibernate benötigen, und kann in der offiziellen Dokumentation⁷ nachgeschlagen werden. Die Basispakete, die in der Regel immer benötigt werden, lauten:

- ▶ hibernate-x.jar
- ▶ dom4j-x.jar
- ▶ cglib-x.jar
- ▶ asm.jar
- ▶ asm-attrs.jar
- ▶ jta.jar

14.9.2 Der Hibernate-Wrapper

[O] Um Hibernate auf einfache Weise zu konfigurieren und dabei trotzdem eine nahtlose Integration in Cocoon zu gewährleisten, soll an dieser

5 Object Relational Mapping

6 <http://www.hibernate.org>

7 http://www.hibernate.org/hib_docs/v3/reference/en/html

Stelle ein Hibernate-Wrapper erstellt werden, den Sie natürlich wieder auf der Buch-CD vorfinden. Dabei handelt es sich um eine benutzerdefinierte Komponente, die im Grunde zwei wichtige Aufgaben übernimmt. Zum einen soll sie die Konfigurationen von Hibernate verwalten bzw. diese entsprechend setzen und zum anderen als zentraler Zugriffspunkt dafür sorgen, dass stets eine gültige Hibernate-Session zurückgeliefert wird, mit der anschließend gearbeitet werden kann. Diese Komponente soll Ihnen die grundsätzliche Vorgehensweise aufzeigen, wie Sie Hibernate in Ihre Cocoon-Applikation integrieren.

Um dies zu erreichen, gehen wir auf die bereits bekannte Art und Weise vor, eine benutzerdefinierte Komponente in Cocoon zu erstellen. Zunächst wird dazu ein entsprechendes Interface wie im nachfolgenden Listing definiert.

Interface

```
package de.cocoonbuch.hibernate;

import org.hibernate.Session;

public interface HibernateWrapper {

    public static final String ROLE =
        HibernateWrapper.class.getName();

    public Session createSession()
        throws HibernateWrapperException;
}
```

Listing 14.31 Das Interface des Hibernate-Wrappers

Dieses Interface zeigt bereits, dass der Hibernate-Wrapper im Grunde relativ einfach aufgebaut ist. Er verfügt lediglich über die öffentliche Methode `createSession`, über die eine gewöhnliche Hibernate-Session zurückgeliefert wird. In Ihrer eigenen Applikation müssen Sie später keine weiteren Anpassungen an Hibernate vornehmen. Die Verwaltung der Hibernate-Konfiguration und der `SessionFactory` wird vollständig vom Hibernate-Wrapper übernommen. Falls Sie Zugriff auf die `SessionFactory` benötigen, können Sie den Wrapper später beliebig erweitern.

Die Implementierung `HibernateWrapperImpl` des Interfaces ist zwar ebenfalls relativ einfach gehalten, aber leider etwas umfangreicher. Deshalb möchte ich Sie hierfür auf die Quellen auf der Buch-CD verweisen. Für die Implementierung anzumerken ist, dass der `Datasource-Pool` von Cocoon bzw. Avalon-Excalibur verwendet wird, um eine Datenbank-Ver-

Implementierung



bindung zu laden, anstelle des von Hibernate zur Verfügung gestellten. Dieser ist laut Hibernate-Entwickler nicht für eine Produktivumgebung gedacht. Darüber hinaus sorgt der Hibernate-Wrapper dafür, dass die Hibernate-Konfiguration nicht mit jedem Zugriff neu erzeugt wird. An dieser Stelle lassen sich sicherlich noch einige Optimierungen durchführen. Falls Sie beispielsweise eine massive Last des Servers erwarten, kann es unter Umständen sinnvoll sein, diese Implementierung entsprechend zu erweitern und beispielsweise »poolable« zu machen wie in Kapitel 9, »Cocoon erweitern«, beschrieben.

Installieren und Registrieren

Nachdem Sie die kompilierten Klassen und Interfaces des Hibernate-Wrappers nach `$COCOON/WEB-INF/classes`, oder das auf der Buch-CD bereitgestellte Jar-Package `hibernate-wrapper.jar` nach `$COCOON/WEB-INF/lib` kopiert haben, müssen Sie diese Komponente noch in der Datei `cocoon.xconf` registrieren:

```
...
<component
  role="de.cocoonbuch.hibernate.HibernateWrapper"
  class="de.cocoonbuch.hibernate.HibernateWrapperImpl">
  <parameter name="datasource" value="mysql-pool"/>
</component>
...
```

Listing 14.32 Registrieren des Hibernate-Wrappers

Datasource Durch den Konfigurationsparameter `datasource` müssen Sie anschließend noch auf die Datasource verweisen, die verwendet werden soll. In Listing 14.32 ist dies beispielsweise wieder `mysql-pool`. Falls noch nicht geschehen, registrieren Sie anschließend eine Datasource mit diesem Namen in `cocoon.xconf` wie z. B. in Listing 14.2 gezeigt.

[>>] Achten Sie darauf, dass Sie den Hibernate-Dialect in der Hibernate-Konfiguration entsprechend der hier verwendeten Datenbank anpassen!

Der nächste Schritt unterscheidet sich nicht von einer »normalen« Verwendung von Hibernate, indem die Hibernate-Konfiguration `hibernate.cfg.xml` innerhalb von `$COCOON/WEB-INF/classes` erstellt und ihr beispielsweise die im nachfolgenden Listing gezeigten Zeilen hinzugefügt werden.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
```

```

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/↵
hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

    <!-- SQL dialect -->
    <property name="dialect">
        org.hibernate.dialect.MySQLDialect
    </property>

    <!-- Mapping files -->
    <mapping
        resource="de/cocoonbuch/hibernate/testing/User.hbm.xml"/>

</session-factory>
</hibernate-configuration>
    
```

Listing 14.33 Die Hibernate-Konfiguration hibernate.cfg.xml

Wie Sie sicherlich bereits bemerkt haben, wird in diesem Beispiel eine MySQL-Datenbank in Verbindung mit Hibernate verwendet. Falls Sie dieses Beispiel selbst ausprobieren möchten, sollten Sie entsprechend ein Schema mit dem Namen `userdb` in MySQL anlegen und diesem die Tabelle `User` mit den drei Spalten `id`, `firstName` und `lastName` hinzufügen. Das folgende Statement zeigt dies.

Tabellenstruktur

```

CREATE TABLE `userdb` (
  `id` BIGINT( 20 ) NOT NULL AUTO_INCREMENT PRIMARY KEY ,
  `firstName` VARCHAR( 200 ) NOT NULL ,
  `lastName` VARCHAR( 200 ) NOT NULL
);
    
```

Listing 14.34 Die Struktur des Beispiel-Schemas userdb

Um den Hibernate-Wrapper zu testen, können Sie die Sub-Applikation `hibernate` verwenden, die sich ebenfalls auf der Buch-CD befindet und gut kommentiert ist. Hier wird die Action `TestHibernateAction` verwendet. Sie demonstriert, wie ein Zugriff auf den Hibernate-Wrapper erfolgen kann, indem sowohl das Speichern, das Abfragen als auch das Löschen mit Hibernate gezeigt werden. Im nachfolgenden Listing können Sie einen Ausschnitt aus dieser Action sehen. Der Einsatz in einem Flowscript erfolgt ähnlich.

```

...
try {
    // HibernateWrapper laden und Session erzeugen
    hibernateWrapper = (HibernateWrapper)
        this.manager.lookup(HibernateWrapper.ROLE);
    session = hibernateWrapper.createSession();
    session.beginTransaction();

    // Hibernate-Aktionen ausführen...
    List userList =
        session.createQuery("from User").list();

    // userList fuer das Template bereitstellen
    request.setAttribute("userList", userList);

    // Anfrage ausführen
    session.getTransaction().commit();

    // Session noch nicht schließen => Lazy Loading!
    // Siehe CloseHibernateSessionFilter.
    request.setAttribute(
        CloseHibernateSessionFilter.ATTR_HIBERNATE_SESSION,
        session);

} catch (Exception e) {
    // Im Fehlerfall Session selbst schließen
    if(session != null) {
        session.flush();
        session.connection().close();
        session.close();
    }
    throw e;
} finally {
    // HibernateWrapper wieder freigeben
    this.manager.release(hibernateWrapper);
}
...

```

Listing 14.35 Verwenden des Hibernate-Wrappers

Mapping Auch die Mapping-Dateien für Hibernate und das zu »mappende« Objekt `User` sind aus den Quellen auf der Buch-CD ersichtlich und sollen an dieser Stelle nicht weiter ausgeführt werden.

14.9.3 Der CloseHibernateSessionFilter

Wenn Sie sich die Action `TestHibernateAction` in Listing 14.35 genauer ansehen, werden Sie feststellen, dass bei einer normalen Ausführung der Action die verwendete Hibernate-Session nicht geschlossen, sondern stattdessen in den aktuellen Request gelegt wird. Der nachfolgende Ausschnitt aus dieser Klasse zeigt dies nochmals.

```
...
request.setAttribute(
    CloseHibernateSessionFilter.ATTR_HIBERNATE_SESSION,
    session);
...
```

Listing 14.36 Hibernate-Session wird in den Request gelegt, anstatt geschlossen.

Dieses Vorgehen ist vor allem dann äußerst wichtig, wenn Sie Hibernate mit dem sogenannten »Lazy-Loading« verwenden. In diesem Fall ist Hibernate so konfiguriert,⁸ dass es bei einer Anfrage einer größeren Mengen von Objekten (z. B. einer Liste) diese nicht alle auf einmal lädt, sondern immer nach und nach in Paketen aus der Datenbank »nachlädt«. Wird beispielsweise über eine solche Liste iteriert, werden die Objekte erst dann aus der Datenbank geladen und erzeugt, wenn der Iterator am entsprechenden Objekt-Paket angelangt ist oder auf das Objekt direkt zugegriffen wird.

Lazy Loading

Stellen Sie sich nun vor, Sie möchten eine relativ große Liste von User-Objekten, wie im vorhergehenden Beispiel angesprochen, per Hibernate aus der Datenbank laden und anschließend auf einer HTML-Seite anzeigen. Die grundsätzliche Vorgehensweise wäre, die Liste mit den User-Objekten z. B. in den Request-Scope zu legen und nach der Action in der Sitemap beispielsweise einen JXTemplate Transformer zu platzieren, der diese Liste aus dem Request durchläuft und die einzelnen User-Objekte in einer Tabelle darstellt. Wenn Sie nun aber die Hibernate-Session in Ihrem Controller (z. B. Action oder Flowscript) schließen, bevor das Template die Liste von User-Objekten vollständig ausgelesen hat, erhalten Sie eine Fehlermeldung. Der Grund für diese Fehlermeldung liegt darin, dass Hibernate keine Möglichkeit hat, weitere Objekte für die Liste »dynamisch« aus der Datenbank nachzuladen, da die Hibernate-Session bereits geschlossen ist.

Problem

⁸ Dies ist die Default-Einstellung von Hibernate. Sie lässt sich deaktivieren.

Lösung Um dieses Problem zu umgehen, ist es notwendig, die Hibernate-Session solange geöffnet zu halten, bis das Template vollständig abgearbeitet wurde. Für diesen Zweck hat sich folgendes Vorgehen bewährt: Die Hibernate-Session wird nicht im Controller geschlossen, sondern in den Request als Attribut gelegt wie in Listing 14.36 gezeigt. Zusätzlich wird ein Servlet-Filter eingebunden, der am Ende des gesamten Requests, also nachdem auch das Template verarbeitet wurde, diese Hibernate-Session aus dem Request lädt und anschließend schließt. In Kapitel 5, »Tomcat«, haben Sie bereits erfahren, wie Sie ihren eigenen Servlet-Filter erstellen und registrieren. In Listing 14.37 sehen Sie einen Ausschnitt des Servlet-Filters `CloseHibernateSessionFilter`, der sich ebenfalls auf der Buch-CD befindet und lediglich die Hibernate-Session aus dem Request extrahiert und, falls vorhanden, schließt.

```
...
Session session =
    (Session)req.getAttribute(ATTR_HIBERNATE_SESSION);

if(session != null) {
    try {
        session.flush();
        session.connection().close();
        session.close();
    } catch (Exception e) {
        // Nichts machen
    }
}
...
```

Listing 14.37 Schließen der Hibernate-Session über einen Servlet-Filter

Um den Servlet-Filter verwenden zu können, müssen Sie diesen zusätzlich noch in der Datei `$COCOON/WEB-INF/web.xml`, am besten direkt unter dem Start-Tag `<web-app>`, registrieren.

```
...
<filter>
    <filter-name>CloseHibernateSession</filter-name>
    <filter-class>
        de.cocoonbuch.hibernate.CloseHibernateSessionFilter
    </filter-class>
</filter>

<filter-mapping>
```

```

    <filter-name>CloseHibernateSession</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
...

```

Listing 14.38 Registrieren des CloseHibernateSessionFilters

14.10 XML-Datenbanksysteme

Bis zu diesem Abschnitt haben Sie bereits eine Menge darüber erfahren, wie Sie mit Cocoon auf relationale Datenbanksysteme zugreifen. Sogar die Abbildung von Objekten in diese Art von Datenbank durch das Tool Hibernate wurde behandelt. Was aber bei einem XML Publishing-System in keinem Fall fehlen darf, ist die Anbindung an XML-Datenbanksysteme als dritter Anwendungsfall.

14.10.1 XML-enabled und XML-native

XML-Datenbanksysteme besitzen im Vergleich zu relationalen Datenbanksystemen nicht den Stellenwert, den man von dem XML-Hype der vergangenen Jahre erwartet hätte. Dies liegt allerdings zum Großteil nicht an der Technik selbst, sondern vielmehr an den hohen Kosten, die ein Umstieg von einem relationalen auf ein XML-Datenbanksystem mit sich bringt. Darüber hinaus haben sich relationale Datenbanksysteme bereits etabliert und leisten – trotz einiger Nachteile gegenüber objekt-orientierten oder XML-Datenbanken – hervorragende Arbeit. Viele Anbieter von relationalen Datenbanksystemen (wie z. B. Oracle) erkannten allerdings den Bedarf nach einer einfachen Schnittstelle, um XML in einer relationalen Datenbank abzulegen, und bieten daher entsprechende Erweiterungen für ihre Systeme an.

Relationale Datenbanken,⁹ die XML mithilfe entsprechender proprietärer Erweiterungen speichern und lesen können, werden *XML-enabled* Datenbanken genannt.

XML-enabled



Neben der Möglichkeit, XML in eine relationale Datenbank zu »map-pen«, gibt es noch einen zweiten Ansatz: Native XML-Datenbanken (*XML-native*).

⁹ Aber auch andere Datenbanktypen.

Index

\$CATALINA_HOME 125
\$COCOON 196
\$COCOON_HOME 191
\$CONTEXT 145
(X)HTML-Dokumente 64
{0} 219

A

AbstractAction 327
AbstractConfigurableAction 327
AbstractGenerator 332
AbstractJavaSelectionList 441
AbstractLogEnabled 305
AbstractLoggable 305
AbstractReader 350
AbstractSerializer 353
AbstractTransformer 337
Action 232, 325
 AbstractAction 327
 AbstractConfigurableAction 327
 act 326
 Action 325
 Anhang 661
 erstellen 325
 erstellen, Beispiel 327
 Kommunikation 234
 Map zurück liefern 326
 Null zurück liefern 326
 registrieren 233, 329
 Request, Zugriff auf 326
 sendmail 661
 Sendmail Action 661
 session 664
 Session Action 664
 verwenden 233
ActionEvent 494
Actions 661
Action-Set 238
 Parameter 240
 selektieren 241
 verwenden 239
Action-Variable 233
Action-Widgets 465
AE 278

Aggregatefield-Widget 457
Aggregation 248
 definieren 250
Ajax 448, 481
AjaxRequestSelector 451, 482
AJP 121
Anforderungen 29
Ant 192, 803
Ant Task 38
Apache Forrest 808
Apache JServ Protocol 121
Apache Lenya 809
Apache Software Foundation 803
Apache Xindice 573
Apidoc 803
Apples 405
 AppleController 405
 AppleRequest 405
 AppleResponse 405
 Beispiel 407
 process 405
 registrieren 406
 sendPage 406
ASCII 46
ASF 803
Assoziatives Array 803
Asterisk 215
Augment Transformer 224, 719
Autoincrement-Modul 275
Avalon 277
Avalon-Excalibur 278

B

BaseLink Module 785
 {baselink:SitemapBaseLink} 785
 baselink 785
Basisname 584
Batik 811
Bean 500
Bean Scripting Framework 683
Binding-Framework 498
 AbstractCustomBinding 504
 Aggregatefield-Widget 505
 Bean 500

Index

- Benutzerdefinierte Regeln* 504
 - Binding in Java* 512
 - Binding-Datei* 501
 - Binding-Manager* 512
 - createBinding* 505, 513
 - doLoad* 504
 - doSave* 504
 - Einfachauswahl* 502
 - fb:context* 502
 - fb:custom* 504
 - fb:javascript* 504
 - fb:load-form* 504
 - fb:multi-value* 503
 - fb:save-form* 504
 - fb:value* 502
 - Flowscript* 505
 - Java-Bean* 500
 - JXPath* 500
 - Konfigurationsdatei* 501
 - konfigurieren* 501
 - load* 505
 - loadFormFromModel* 513
 - Mapping-Regeln* 501
 - Mehrfachauswahl* 503
 - Namensraum* 501
 - Objekte* 500
 - parent-path* 503
 - path* 502
 - Properties-Schreibweise* 502
 - Repeater-Widget* 505
 - row-path* 503
 - save* 505
 - saveFormToModel* 513
 - BizData 386
 - Black-Box-Test 360
 - Block 189
 - authentication-fw* 190
 - batik* 190
 - cron* 190
 - databases* 190
 - faces* 191
 - fop* 190
 - forms* 190
 - jsp* 190
 - mail* 190
 - naming* 190
 - slide* 190
 - velocity* 191
 - Block ausschließen 193
 - blocks.properties 193
 - Booleanfield-Widget 428, 435
 - Browser Selector 231, 701
 - BrowserUpdate Transformer 451, 482
 - BSF 683
 - Buglet 809
 - build.properties 193
 - Built-In-Logicsheet 522
 - action* 522
 - input* 522
 - log* 522
 - util* 522
 - xsp-cookie* 522
 - xsp-formval* 522
 - xsp-request* 522
 - xsp-response* 522
 - xsp-session* 522
 - Business tier 36
- ## C
-
- Cache Tag Library 168
 - Cached-Response 356
 - Caching 262, 356
 - CacheableProcessingComponent* 357
 - getKey* 357
 - Source-Validity-Objekt* 357
 - Catalina 123
 - CDATA 55
 - Checkout 803
 - CInclude Transformer 224, 249, 720
 - Class-Widget 459
 - CLI 639
 - cli.xconf 642
 - CLI-Konfigurationsdatei 642
 - CloseHibernateSessionFilter 569
 - Cocoon
 - Apidocs* 807
 - besorgen* 191
 - Block-Beispiele* 195
 - downloaden* 191
 - installieren* 189, 194
 - kompilieren* 192
 - Mailinglist* 808
 - starten* 194
 - Userdocs* 807
 - Website* 807
 - Wiki* 807
 - Zones* 807

- cocoon 747
 - createObject* 749
 - disposeObject* 749
 - Funktionen* 749
 - getComponent* 750
 - load* 750
 - parameters* 748
 - processPipelineTo* 750
 - Properties* 748
 - redirectTo* 751
 - releaseComponent* 751
 - sendPage* 752
 - sendPageAndWait* 752
 - sendStatus* 753
- Cocoon 1.x 27
- Cocoon 2 28
- Cocoon 2.1.x 28
- Cocoon Ant Task 639
- Cocoon Bean 37, 639
- Cocoon erweitern 277
- Cocoon Forms 411
 - \${treeNode.node}* 450
 - \${treeNode.path}* 452
 - \${treeNode}* 450
 - \${widget.fullName}* 452
 - \${widget}* 450
 - AbstractFormHandler* 497
 - AbstractFormsAction* 507
 - AbstractJavaSelectionList* 441
 - ActionEvent* 494
 - ActionListener* 467
 - Action-Widget* 465
 - Action-Widgets* 465
 - active* 432
 - addActionListener* 496
 - addItem* 442
 - addMessage* 463
 - add-row* 466
 - Aggregatefield-Widget* 457
 - Ajax* 448, 481
 - AjaxRequestSelector* 451, 482
 - Assert-Validator* 484
 - attribute-name* 510
 - Automatisches Layout* 477
 - Basis-Datentypen* 433
 - Baumstruktur darstellen* 444
 - Beispiel* 417
 - Binding-Framework* 498
 - Binding-Manager* 512
 - Booleanfield-Widget* 428, 435
 - BrowserUpdate Transformer* 451, 482
 - Buttons* 476
 - Checkbox* 435
 - Checkbox-Liste* 475
 - choice* 480
 - Class-Widget* 459
 - collapse* 452
 - Column-Layout* 478
 - Columns-Layout* 478
 - Combine* 459
 - command* 465, 466
 - Container Widget* 431
 - Controller* 415, 423
 - Convertor* 433
 - Convertor-Typ bestimmen* 434
 - createForm* 507
 - Datentypen* 432
 - Default Selection-List* 438
 - DefaultTreeModel* 445
 - DefaultTreeNode* 445
 - delete-rows* 466
 - disabled* 432
 - Doppel-Liste* 476
 - Dynamic Selection List* 439
 - Einfachauswahl* 436
 - Einfache Widgets* 434
 - Email-Validator* 484
 - Enum Selection-List* 443
 - Enumeration-Pattern* 443
 - Errors auflisten* 481
 - Event-Element* 495
 - Event-Handling* 494
 - expand* 452
 - expression* 459
 - Failmessage* 483
 - fd:action* 465
 - fd:aggregatefield* 458
 - fd:assert* 484
 - fd:attribute* 430
 - fd:attributes* 430
 - fd:booleanfield* 435
 - fd:class* 460
 - fd:combine* 459
 - fd:convertor* 433
 - fd:datatype* 432
 - fd:dirset* 448
 - fd:email* 484
 - fd:exclude* 448

Index

- fd:failmessage* 484, 492
- fd:field* 434, 436
- fd:fileset* 448
- fd:form* 428
- fd:group* 461
- fd:help* 430
- fd:hint* 430
- fd:imagemap* 467
- fd:imageuri* 467
- fd:include* 448
- fd:initial-value* 436
- fd:item* 438
- fd:java* 488, 496
- fd:javascript* 491
- fd:label* 430, 438
- fd:length* 485
- fd:map* 458
- fd:messages* 463
- fd:mod10* 485
- fd:multivaluefield* 437
- fd:new* 460
- fd:on-...* 495
- fd:range* 486
- fd:regexp* 487
- fd:repeater* 464
- fd:selection-list* 436, 437
- fd:split* 458
- fd:submit* 466
- fd:true-param-value* 436
- fd:upload* 468
- fd:validation* 483
- fd:validation-errors* 481
- fd:value-count* 487
- fd:widgets* 428, 429
- Fehlerschlüssel* 492
- fi:available-label* 476
- fi:group* 478, 479
- fi:items* 478, 480
- fi:selected-label* 476
- fi:state* 480
- fi:styling* 471
- Field-Widget* 428
- FileSource* 451
- Flow-Context* 440
- Flow-JXPath Selection-List* 439
- Flowscript* 423
- Form.js* 423
- formatting* 434
- FormContext* 508
- Form-Definition* 427
- Form-Handler* 496
- FormHandler* 497
- Form-Manager* 507
- Form-Model* 427
- Form-Objekt erstellen* 423
- forms-samples-styling.xsl* 471
- Form-Template* 469
- ft:class* 460
- ft:form-template* 422, 470
- ft:group* 461
- ft:new* 461
- ft:tree* 449
- ft:tree-children* 450
- ft:tree-nodes* 450
- ft:widget* 470
- ft:widget-label* 470
- GET* 511
- getActionCommand* 465
- getAttribute* 430
- getChildren* 431
- getId* 431
- getModel* 423
- getValue* 431
- Group-Widget* 461
- Gruppierende Widgets* 457
- handleEvent* 497
- HandleSubmitAction* 510
- HtmlArea* 473
- I18n* 430
- Image-Button* 476
- ImageMapEvent* 467
- Imagemap-Widget* 467
- initial-size* 464
- Instanz-Element* 416, 470
- Interface Widget* 431
- Internationalisieren* 492
- Internationalisierung* 430
- invisible* 432
- Items-Provider* 437
- Java Selection-List* 441
- Java-Eventt-Listener* 495
- JavaScript-Event-Listener* 497
- JavaScript-Validator* 491
- Java-Validator* 488
- JX-Macros* 422
- JX-Template* 421
- Karteireiter* 479
- Kreditkartennummer validieren* 485

- label-path* 441
- Length-Validator* 485
- Link-Button* 477
- Listenbox* 475
- Listen-Widgets* 436
- list-orientation* 475
- list-path* 440
- lookupWidget* 431
- MakeFormAction* 510
- max-size* 464
- Mehrfachauswahl* 437
- Messages-Widget* 463
- millis* 434
- mime-types* 468
- min-size* 464
- Mod10-Validator* 485
- Multivalue-Widget* 428
- number-of-rows* 466
- Ohne Flowscript* 506
- output* 432
- Part* 468
- Passwort-Feld* 472
- plain* 434
- POST* 512
- ProcessingPhaseEvent* 494
- Radio-Liste* 474
- Range-Validator* 486
- Regex-Validator* 487
- repeater* 466
- Repeater Action* 466
- Repeater-Widget* 464
- Ressourcen publizieren* 426
- root-visible* 446
- Row Action* 466
- Row-Layout* 478
- Rows-Layout* 478
- select* 452, 466
- Selection-List* 437
- setState* 432
- setValidationError* 490
- setValue* 431
- showForm* 423
- Sitemap* 425
- SourceTreeModel* 447
- Split* 458
- state* 432
- Submit-Widget* 465
- Tabelle darstellen* 464
- Tabs* 479
- tabs* 480
- Template* 469
- Textarea* 473
- toggle-collapse* 452
- toggle-select* 452
- TreeAction* 453
- Tree-Ereignisse* 452
- Tree-Listener* 455
- TreeModel* 444
- TreeSelection-Listener* 456
- TreeToggleCollapse* 453
- TreeWalker* 450
- Tree-Widget* 428, 444
- Union-Widget* 461, 462
- unselect* 452
- Upload-Widget* 429, 468
- ValidationError* 489
- ValidationErrorAware* 490
- Validation-Errors auflisten* 481
- Validierung* 483
- ValueChangedEvent* 494
- Value-Count-Validator* 487
- value-path* 441
- Widget-Definition* 429
- Widget-Id* 428, 429
- Widget-Objekt* 430
- Widgets* 428
- WidgetState* 432
- WidgetValidator* 488
- Widget-Zustand Active* 431
- Widget-Zustand Disabled* 431
- Widget-Zustand Invisible* 432
- Widget-Zustand Output* 432
- Widget-Zustände* 431
- XML* 500
- XSL-Transformation* 416
- Cocoon Servlet 38
- Cocoon starten 194
- cocooncenter.org 808
- cocoonforum.de 808
- Cocoon-Include 720
- Command Line Tool 639
- Command Line Tool (CLI) 37
- component 281
- Component-Lifecycle 287, 288
- Component-Manager 285
- Composable 293
- Configurable 294
- Connection-Pool 539

Index

- Consumer 324
- Container-Widget 431
- ContentHandler 115
- Context
 - Zugriff aus Komponente* 290
- context 763
 - Funktionen* 764
 - getAttribute* 764
 - getInitParameter* 764
 - removeAttribute* 764
 - setAttribute* 764
- ContextHelper 290
- Contextualizable 289
- Continuation 374
 - konfigurieren* 376
 - laden* 375
- Control Flow 369
 - Komponenten* 388
- cookie 765
 - Funktionen* 765
 - getComment* 765
 - getDomain* 765
 - getName* 765
 - getPath* 765
 - getSecure* 766
 - getValue* 766
 - getVersion* 766
 - setComment* 766
 - setDomain* 766
 - setPath* 766
 - setSecure* 767
 - setValue* 767
 - setVersion* 767
- Cross Media Publishing 32
- CSV Generator 667
- CVS 803

- D**

- Daisy 809
- DAO 546
- Data Access Object 546
- Data tier 36
- Database Action 234
- Database-Modul 275
- Datasource 538, 540
 - auto-commit* 540
 - dburl* 540
 - driver* 540
 - getConnection* 543
 - JDBC* 538
 - JNDI* 561
 - password* 540
 - pool-controller* 539
 - registrieren* 538
- DataSourceComponent 543
- Date Input Module 272
- DateInput Module 786
 - {date:date}* 786
 - date* 786
- Datei-Upload 529
- Datenbankzugriffe 537
- Datenquellen 34
- DateTime Tag-Library 168
- DBTags Tag-Library 168
- DEBUG 303
- Debuggen
 - View* 243
- Default Deployment Descriptor 149
- Default Selection-List 438
- Default-Komponente 210
- Defaults Module 273, 787
 - {defaults:skin}* 787
 - defaults* 787
- DefaultTreeModel 445
- DefaultTreeNode 445
- Dependency Injection 279
- Deployment Descriptor 149
- Directory Generator 219, 221, 669
- Directory ZIP Archiver 229
- Disposable 298
- Document Object Model 101
- DOM 101
 - Apidoc* 106
 - Attribute* 107
 - ATTRIBUTE_NODE* 109
 - CDATA_SECTION_NODE* 109
 - COMMENT_NODE* 109
 - Document* 106
 - DOCUMENT_NODE* 109
 - Element* 106
 - ELEMENT_NODE* 109
 - Java-Binding* 106
 - Knotentyp* 108
 - Nachteile* 101
 - NamedNodeMap* 106
 - Node* 106
 - NodeList* 106

Parser 102
Rekursion 108
SAXException 105
startElement 115
TEXT_NODE 109
verzeigerte Objekte 101
Vorteile 101
 DOM LS 105
 dom4j 315, 811
 DOM-Parser 102
 DTD 64
 67, 68, 69, 70
 * 67
 + 67
 ? 67
 Attribut 68
 einbetten 71
 einbinden 70
 Element 67
 Entity 69
 Gemischter Inhalt 68
 IMPLIED 68
 Import 70
 PCDATA 68
 referenzieren 70
 REQUIRED 68
 Dummy-Objekt 361
 Dynamischer Generator 517

E

Eclipse 100
 Einfache Widgets 434
 eingebettete Anweisungen 225
 EncodeURL Transformer 224, 724
 Encoding 46
 Entity Referenzen 54
 Entwicklungsumgebung 99
 ERROR 304
 Erweiterbarkeit 31
 ESQ-Logicsheet 554
 Event-Handling 494
 Excalibur 277
 Apidocs 807
 Website 807
 Excel 228
 Excel Generator 674
 Exception Selector 265
 exclude 193

eXist 573, 811
 Extensible HyperText Markup Language 62
 Extensible Markup Language 44
 Extensible Server Pages 515, 773
 Extensible Stylesheet Language 73, 80

F

Factory-Pattern 279
 FATAL_ERROR 304
 Fehlerbehandlung 263
 Exception Selector 265
 Reihenfolge 265
 Status Codes 268
 XPath Exception Selector 266
 Field-Widget 428
 File Generator 221, 673
 Filter 155
 Filter Transformer 726
 Filter-Chain 155
 Flow Object Model 384, 747
 FlowAttribute Module 787
 flow-attribute 788
 Flow-Context 440
 Flow-JXPath Selection-List 439
 Flowsript 378
 AE-Komponenten 381
 Bean 386
 Beispiel 396
 BizData 385, 386
 fortsetzen 384
 getComponent 288, 381
 Getter-Zugriff 381
 importClass 381
 importPackage 381
 JavaScript, warum 379
 JXTemplate Generator 392
 JXTemplate Transformer 392
 Komponenten 388
 Model 386
 Namensraum 383
 Properties 381
 registrieren 382
 releaseComponent 288, 382
 sendPage 386
 sendPageAndWait 385
 Service-Manager 287
 Setter-Zugriff 381

Index

- starten* 383
- Velocity Generator* 389
- View-Komponenten* 388
- Zugriff auf DB* 546
- FOM 384, 747
 - Anhang* 747
 - cocoon* 384
 - context* 385
 - cookie* 385
 - log* 385
 - request* 384
 - response* 385
 - session* 385
 - WebContinuation* 385
- FOP 93, 708, 811
- FOP Serializer 227
- FO-Prozessor 89, 93
- Formatting Objects 88
- Form-Definition 415, 419, 427
 - Namensraum* 428
 - required* 430
- Form-Instanz 415
- Form-Manager 507
- Form-Model 415, 427
- Form-Objekt 415
- Form-Publishing-Dokumente 417
- Form-Publishing-Prozess 413
- Forms Generator 422
- Forms Transformer 422
- forms-samples-styling.xsl* 471
- Form-Template 415, 421
 - Widget-Definition* 429
- Formular-Frameworks 412
- Fragment Extractor Generator 221

G

- Generator 219
 - AbstractGenerator* 332
 - Anhang* 667
 - csv* 667
 - CSV Generator* 667
 - Datei parsen* 335
 - directory* 669
 - Directory Generator* 669
 - erstellen, Beispiel* 332
 - erstellen* 331
 - file* 673
 - File Generator* 673

- Generator* 331
- HSSF Generator* 674
- html* 675
- HTML Generator* 675
- Image Directory Generator* 676
- imagedirectory* 676
- jsp* 677
- JSP Generator* 677
- jx* 678
- JXTemplate Generator* 678
- linkStatus* 679
- LinkStatus Generator* 679
- Parser* 332
- registrieren* 220
- request* 680
- Request Generator* 680
- script* 682
- Script Generator* 682
- search* 684
- Search Generator* 684
- Server Pages Generator* 688
- serverpages* 688
- stream* 689
- Stream Generator* 689
- velocity* 690
- Velocity Generator* 690
- verwenden* 220
- xls* 674
- XPath Directory Generator* 690
- xpathdirectory* 690
- Generatoren 667
- Generierung 199
- getMimeType* 324
- Global Input Module 272
- GlobalInput Module 788
 - {global:document}* 789
 - global* 789
 - map:component-configurations* 789
- global-variables 789
- Glossar 803
- Group-Widget 461
- Gruppierende Widgets 457
- Gruppierung 198

H

- HandleSubmitAction 510
- Header Selector 231
- HeaderAttribute Module 790

{request-header:user-agent} 790
accept 790
accept-encoding 790
accept-language 790
cache-control 790
connection 790
host 790
request-header 790
user-agent 790
 Helper(-Klasse) 803
 Hibernate 564
 besorgen 564
 CloseHibernateSessionFilter 569
 installieren 564
 Lazy Loading 569
 Mapping 568
 Hibernate-Wrapper 564
 Host -Request-Header 231
 Host Selector 231, 703
 HSSF Generator 674
 HSSF Serializer 228
 HTML 64
 HTML Generator 222, 675
 HTML Serializer 227, 705
 HtmlUnit 360
 HttpContext 291
 HttpSessionBindingListener 160
 HttpSessionListener 160
 sessionCreated 160
 sessionDestroyed 160
 HttpUnit 360
 HTTP-User-Agent-Header 231

I

I18n 581
 Parameter übersetzen 593
 I18N Tag-Library 168
 I18n Transformer 224, 581, 728
 DateFormat 595
 DecimalFormat 597
 i18n:attr 594
 i18n:catalogue 593
 i18n:date 595
 i18n:date-time 595
 i18n:key 590
 i18n:number 596
 i18n:param 591
 i18n:text 590

i18n:time 595
 i18n:type 597
 Namensraum 588
 NumberFormat 597
 Parameter verwenden 591
 registrieren 582
 SimpleDateFormat 595
 verwenden 583
 IANA 165
 IDE 99
 Image Directory Generator 676
 Image Reader 229
 ImageMapEvent 467
 Imagemap-Widget 467
 Implizites Objekt 803
 INFO 304
 Initializable 298
 inkludieren 720, 741
 Inkrementierung 804
 Input-Modul 272
 Anhang 785
 Input-Module 785
 Installieren von Cocoon 194
 Instanz 804
 internal-only 202
 Internationalisierung 32, 581
 Inversion of Control 278
 Invoker-Servlet 132
 IoC 278
 ISO-8859-1 46
 ISO-Zeichensatz 46
 Items-Provider 437
 Iteration 804

J

j_password 176
 j_security_check 176
 j_username 176
 J2EE 804
 J2EE-Connection 561
 J2ME 804
 J2SDK 804
 Jakarta 122, 804
 Jasper 134
 Java 804
 Java Expression Language 392
 Java Selection-List 441
 Java Virtual Machine 804

Index

Java Web Server 121
JAVA_HOME 124
Javaflow 379, 402
 AbstractContinuable 402
JavaScript 379
JavaServer Pages 133
JAXP 105
JDBC 804
JDBC-Treiber 537
 installieren 537
 load-class 538
JEdit 99
Jetty 194
Jexl 392, 811
JNDI-Datasource 561
JNDI-Resource 563
JPath Logicsheet 395
 Elemente 396
 Namensraum 395
JPEG 709
JRE installieren 123
JServ 121
JSP 133
 %> 135
 <% 135
 Beispiel 135
 registrieren 162
JSP 2.0 123
JSP Generator 222, 677
JSP Reader 229, 697
JSP-Compiler 134
jsp-develop.de 809
JspServlet 134
JTidy 675
JUnit 361
JXForms 413
JXPath 79, 392, 810
JXTemplate 392
 Continuation-Id 394
 implizite Objekte 394
 Iteration 393
 jsx:forEach 393
 Sprachelemente 395
 XPath 393
JXTemplate Generator 222, 678
JXTemplate Transformer 224, 678

K

Karteireiter 479
Kompilieren 804
Kompilieren von Cocoon 192
Komponente
 Component-Lifecycle 288
 erstellen 279
 erweitern 288
 initialisieren 298
 konfigurieren mit Elementen 293
 konfigurieren mit Parameter 296
 loggen 259
 loggen, Meldungen 302
 poolen 299
 registrieren 281
 zugreifen auf andere Komponente 292
 Zugriff auf DB 542
 Zusatzfunktionalität 288
Komponente testen 365
Komponenten-Modell 278
Konfiguration 258
 Root-Sitemap 258
Konkatenation 804

L

L10n 581
Label 245
Lifecycle-Interface
 Composable 293
 Configurable 294
 Contextualizable 289
 Disposable 298
 Initializable 298
 LogEnabled 302
 Loggable 305
 Parameterizable 296
 Serviceable 292
Links überprüfen 679
LinkStatus Generator 221, 679
Listen-Widgets 436
local.build.properties 193
Locale 585
Locale Action 598
 registrieren 599
 verwenden 600
log 767
 debug 768

- error* 768
 - Funktionen* 768
 - info* 768
 - isDebugEnabled* 768
 - isErrorEnabled* 768
 - isInfoEnabled* 769
 - isWarnEnabled* 769
 - warn* 769
 - Log Transformer 224, 731
 - Log4j verwenden 306
 - LogEnabled 302
 - Loggable 305
 - Logging 259, 302
 - AbstractLogEnabled* 305
 - AbstractLoggable* 305
 - Enabled-Methode* 304
 - LogEnabled* 302
 - Loggable* 305
 - Performance steigern* 305
 - Logicsheet 522
 - action* 522
 - eigenes erstellen* 525
 - input* 522
 - log* 522
 - registrieren* 523
 - util* 522
 - verwenden* 523
 - xsp-cookie* 522
 - xsp-formval* 522
 - xsp-request* 522
 - xsp-response* 522
 - xsp-session* 522
 - Log-Kategorie 260
 - LogKit 259
 - Log-Level 260, 303
 - DEBUG* 303
 - ERROR* 304
 - FATAL_ERROR* 304
 - INFO* 304
 - WARN* 304
 - Lokalisierung 581
 - Lucene 684, 811
- M**
-
- Mail-Archive 812
 - Mailer Tag-Library 168
 - MakeFormAction 510
 - map:act 233
 - map:action 233
 - map:actions 233
 - map:action-set 198
 - map:component-configurations 789
 - map:components 198
 - map:flow 198
 - map:generate 220
 - map:generator 220
 - map:generators 220
 - map:match 213
 - map:matcher 214
 - map:matchers 214
 - map:mount 255
 - map:pipe 262
 - map:pipeline 213
 - map:pipelines 198
 - map:read 229
 - map:reader 229
 - map:readers 229
 - map:resources 198
 - map:serialize 227
 - map:serializer 226
 - map:serializers 226
 - map:transform 223
 - map:transformer 223
 - map:transformers 223
 - map:views 198
 - Marker-Interface 299
 - Poolable* 300
 - SingleThreaded* 299
 - ThreadSafe* 300
 - Matcher 212, 345, 695
 - Anhang* 695
 - erstellen* 344
 - erstellen, Beispiel* 345
 - Matcher* 345
 - Wildcard* 215
 - wildcard* 695
 - WildcardURI Matcher* 695
 - Matching 199
 - Mazzocchi, Stefano 27
 - McClanahan Craig 123
 - Message-Katalog 584
 - Messages-Widget 463
 - Middleware 36
 - Mock 361
 - Model View Controller 30
 - Modul 271
 - autoincrement-modules* 272

Index

- input-modules* 271
- output-modules* 271
- registrieren* 271
- Modularität 31
- mounten
 - Sub-Sitemap* 255
- Mozilla 379
- MySQL 541
- Multichannel Publishing 32
- Multivalue-Widget 428
- MVC 30
- MVC Model 2 30
- MySQL 540

N

- Namensraum 57
 - Bindung* 57
 - Präfix* 57
 - qualifizierter Name* 57
 - URI* 57
- Namespace 57
- NetBeans 100, 170
- News-Portal 603
 - article2fo.xsl* 628
 - article2html.xsl* 626
 - Benutzerverwaltung* 607
 - createUser* 613
 - error.jxt* 616
 - Flowscript* 611
 - internal-only* 620
 - JXTemplates* 615
 - listUsers* 614
 - list-users.jxt* 618
 - Login* 631
 - login.jxt* 632
 - loginManager.js* 633
 - Manager registrieren* 610
 - News* 620
 - news2html.xsl* 623
 - Pipeline schützen* 634
 - register.jxt* 616
 - register-ok.jxt* 617
 - Sitemap* 619
 - Struktur* 606
 - User* 607
 - UserManager* 608
- Non Caching Pipeline 263
- noncaching 263

O

- Object-Model 323
 - ObjectModelHelper* 323
- ObjectModelHelper 323
- Offline-Betrieb 639
- Oracle 540
- OR-Mapping 564
- ORO 487
- Output-Modul 275
 - commit* 275
 - rollback* 275
 - Transaktion* 275
- Output-Stream
 - Zugriff auf den* 323

P

- Parameter
 - Gruppierungen* 207
 - Sitemap* 206
- parameter 296
- Parameter Selector 231
- Parameterizable 296
- Part 531
- PartInMemory 531
- PartOnDisk 531
- pattern 213, 215
- PDF 804
- PDF erstellen 94, 628
- PDF Serializer 227, 707
- PDF Serializer → FOP Serializer
- Persistenz 804
- Personalisierung 31
- Pfadangabe 203
- PI 805
- Pipe 324
- Pipeline 198
 - intern* 201
 - öffentlich* 201
- Pipeline-Key 218
- Pipeline-Pfad 213
- Pipeline-Variable 218
- Plattformunabhängigkeit 31
- PNG 710
- Pointer 374
- Poolable 300
- Pooling 299
 - Poolable* 300

Recyclable 301
 Portal 603
 PostgreSQL 540
 PostScript erstellen 628
 präsentierendes Format 89
 Processing Instruction 805
 ProcessingPhaseEvent 494
 Producer 324
 Programmers Notepad 2 100
 PropertiesFile Module 791
 file 791
 my-properties 791
 Properties-Schreibweise 502
 Property-Datei 805
 Protokoll 203
 cocoon:/ 205
 cocoon:// 205
 context:// 205, 206
 resource:// 205, 206
 Protokollschema 203
 PS Serializer 708
 Pseudo-Protokoll 205
 xmldb 575

R

Random Number Module 273
 RandomNumber Module 792
 {random:random} 792
 max 792
 min 792
 random 792
 RawRequestParameter Module 793
 {raw-request-param:bar} 793
 raw-request-param 793
 Read DOM Session Transformer 732
 Reader 228, 348
 AbstractReader 350
 Anhang 697
 erstellen 348
 erstellen, Beispiel 350
 jsp 697
 JSP Reader 697
 Reader 348
 registrieren 229
 resource 698
 Resource Reader 698
 verwenden 229
 Readers 697
 RealPath Module 794
 {realpath:page.xml} 794
 realpath 794
 Recyclable 301
 Redirect 242
 Parameter 238
 verwenden 242
 RegExp 671
 Registrieren
 Sitemap-Komponente 209
 Rekursion 805
 relative URLs wandeln 719
 Repeater Action 466
 Repeater-Widget 464
 Request 805
 Zugriff aus Komponente 290
 Zugriff aus Sitemap-Komponente 323
 request 753
 Funktionen 753
 get 754
 getAttribute 754
 getAttributeNames 754
 getAuthType 755
 getCharacterEncoding 755
 getContentLength 755
 getContentType 755
 getCookies 755
 getHeader 755
 getHeaderNames 756
 getHeaders 756
 getLocale 756
 getLocales 756
 getParameter 757
 getParameterValues 757
 getProtocol 757
 getRemoteAddr 758
 getRemoteUser 758
 getScheme 758
 getServerName 758
 getServerPort 758
 getUserPrincipal 758
 isSecure 753
 isUserInRole 754
 removeAttribute 759
 setAttribute 759
 setCharacterEncoding 759
 Request Attribute Map 275
 Request Attribute Module 273
 Request Attribute Output Module 275

Index

- Request Generator 221, 680
- Request Module 273, 795
 - {request:userPrincipal/name}* 796
 - request* 796
- Request Parameter Module 273
- Request Selector 231
- RequestAttribute Module 794
 - {request-attr:bar}* 795
 - request-attr* 795
- RequestParamer Module 796
 - {request-param:bar}* 797
 - request-param* 797
- Request-Response-Zyklus 39
- Request-Response-Zyklus 212
- resolveURI 310
- Resource 236
 - aufrufen* 237
 - erstellen* 236
- Resource Reader 229, 698
- Response 805
- response 759
 - addCookie* 759
 - addHeader* 760
 - containsHeader* 760
 - createCookie* 760
 - Funktionen* 759
 - setHeader* 760
 - setStatus* 761
- Rhino 379, 811
- role 283
- role-list 283
- Rolle 282
 - Datei, extern* 282
 - Shorthand* 284
- Rolle von Cocoon 40
- Root-Sitemap 254
 - konfigurieren* 258
- S**

- SAX 112
 - Apidoc* 116
 - Callback-Methode* 115
 - characters* 116
 - ContentHandler* 115
 - DefaultHandler* 116
 - endDocument* 116
 - endElement* 116
 - ereignisorientiert* 112
 - Error-Handler* 116
 - getElementsByTagName* 112
 - Nachteile* 113
 - Parser* 113
 - SAX-Event* 114
 - sequentiell* 112
 - startDocument* 115
 - Vorteile* 113
- SAX-Events
 - filtern* 726
 - loggen* 731
- Saxon 811
- SAX-Parser 113
- saxproject.org 810
- Scope 805
- Script Action 235
- Script Generator 682, 683
 - JavaScript* 683
 - REXX* 683
 - XSLT* 683
- Search Generator 684
- Selection-List 437
- Selector 229, 340
 - Anhang* 701
 - browser* 701
 - Browser Selector* 701
 - erstellen* 340
 - erstellen, Beispiel* 341
 - host* 703
 - Host Selector* 703
 - Selector* 340
- Selectoren 701
- Sendmail Action 234, 661
- Separation of Concerns 279
- Serialisierung 200
- Serializer 226, 353, 705
 - AbstractSerializer* 353
 - Anhang* 705
 - erstellen* 353
 - erstellen, Beispiel* 353
 - fo2pdf* 707
 - FOP Serializer* 708
 - html* 705
 - mime-type* 227
 - PDF Serializer* 707
 - registrieren* 226
 - Serializer* 353
 - SVG Serializer* 709, 710, 712
 - SVG/JPEG Serializer* 709

- SVG/PNG Serializer 710
- SVG/TIFF Serializer 712
- SVG/XML Serializer 714
- svg2jpeg 709
- svg2png 710
- svg2tiff 712
- svgxml 714
- verwenden 227
- xhtml 715
- XHTML Serializer 715
- XML Serializer 717
- Server Pages Generator 221, 688
- server.xml 645
 - Connector 646
 - Context 648
 - Engine 647
 - Host 648
 - Server 646
 - Service 646
- ServerPages Generator 516
 - registrieren 516
- Service Selector 306
- Serviceable 292
- Service-Manager 285
 - finally 286
 - Flowscript 287
 - hasService 286
 - lookup 286
 - release 286
- Servlet 127
 - Apidoc 129
 - Beispiel 129
 - destroy 128
 - doGet 128
 - doPost 128
 - doPut 128
 - GenericServlet 128
 - HttpServlet 128
 - init 128
 - Lebenszyklus 128
 - mappen 131
 - registrieren 131, 162
 - service 128
 - servlet.jar 129
 - ServletContext 154
- Servlet Developer Kit 121
- Servlet-API 2.4 123
- Servlet-Container 121, 127
- ServletContext 292
 - ServletContextListener 160
 - Sesion Module
 - {session.id} 799
 - Session
 - Zugriff aus Komponente 291
 - session 761
 - Funktionen 761
 - getAttribute 761
 - getCreationTime 761
 - getId 762
 - getLastAccessedTime 762
 - getMaxInactiveInterval 762
 - invalidate 762
 - isNew 762
 - removeAttribute 763
 - setAttribute 763
 - setMaxInactiveInterval 763
 - Session Action 235, 664
 - Session Attribute Module 273
 - Session Attribute Output Module 275
 - Session Module 273, 798
 - session 799
 - Session Selector 231
 - SessionAttribute Module 797
 - session-attr 798
 - setOutputStream 324
 - SGML 44, 805
 - shouldSetContentLength 324
 - Simple Api for XML 112
 - SingleThreaded 299
 - Singleton 300
 - Sitemap
 - If-Konstrukt 232
 - Root-Sitemap 258
 - Sub-Sitemap 255
 - Switch-Konstrukt 229
 - Verzweigung 229
 - sitemap.xmap 197
 - SitemapComponentTestCase 362
 - Sitemap-Komponente 208
 - Action 325
 - Action erstellen 325
 - cachen 356
 - Consumer 324
 - erstellen 322
 - Generator 331
 - Generator erstellen 331
 - getMimeType 324
 - konfigurieren 235

Index

- loggen* 259
- Matcher* 345
- Matcher erstellen* 344
- Pipe* 324
- Producer* 324
- Reader* 348
- Reader erstellen* 348
- registrieren* 209
- Request, Zugriff auf* 323
- Selector* 340
- Selector erstellen* 340
- Serializer* 353
- Serializer erstellen* 353
- setOutputStream* 324
- Setup* 322
- shouldSetContentLength* 324
- SitemapModelComponent* 322
- SitemapOutputComponent* 323
- Transformer* 336
- Transformer erstellen* 336
- Typ* 208
- verwenden* 211
- XMLConsumer* 324
- XMLPipe* 325
- XMLProducer* 324
- SitemapModelComponent* 322
- SitemapOutputComponent* 323
- Sitemap-Parameter* 206
- Sitemap-Resource* 236
- Sitemap-URI* 39, 213
- Sitemap-Variablen* 207
- Skalierbarkeit* 32
- SoC* 279
- Source* 311
- Source implementieren* 315
- Source Writing Transformer* 224
- Source-Factory* 319
- Source-Resolver* 310
 - Source* 312
 - Source, benutzerdefiniert* 313
- Source-Resolving* 309
- SourceUtil* 312, 335, 336
- SourceWriting Transformer* 733
- spreadcococon.org* 808
- SQL Transformer* 224, 547, 735
- Standalone-Deployer* 123
- Standard Generalized Markup Language*
 - 44
- Standard-Actions* 234
- Standard-Entities* 55
- Standard-Generatoren* 221
- Standard-Protokoll* 204
- Standard-Reader* 229
- Standard-Selectoren* 231
- Standard-Serializer* 227
- Standard-Transformer* 223
- Status Code* 268
- Stream Generator* 689
- String nach XML* 335
- String parsen* 335
- StringXMLizable* 336
- Sub-Applikation* 254
- Submit-Widget* 465
- Sub-Sitemap* 254
 - Bäume* 256
 - erzeugen* 257
 - minimal* 257
 - mounten* 255
 - mounten, automatisch* 256
- Subversion* 805
- SVG* 714
- SVG Serializer* 227
- SVG/JPEG Serializer* 709
- SVG/PNG Serializer* 710
- SVG/TIFF Serializer* 712
- SVG/XML Serializer* 714
- System Property Module* 273
- SystemProperty Module* 799
 - {system-property:os.name}* 801
 - file.separator* 800
 - java.class.path* 800
 - java.home* 799
 - java.vendor* 799
 - java.version* 799
 - java.vm.vendor* 800
 - java.vm.version* 799
 - line.separator* 800
 - os.name* 800
 - os.version* 800
 - system-property* 800
 - user.dir* 800
 - user.home* 800
 - user.name* 800

T

-
- Tabs 479
 - Tag 47
 - Tag Library Descriptor 169
 - Taglib 167
 - Direktive* 170
 - Tag-Library 167
 - Tamino XML Server 573
 - tee 738
 - Tee Transformer 224, 738
 - Template 805
 - testen 360
 - Testkonfiguration 363
 - ThreadSafe 300
 - Three-Tier-Model 36
 - TIFF 712
 - TLD 169
 - Tomcat 121, 645
 - Access Log Valve* 137
 - Admin-Tool* 142
 - AJP Connector* 140
 - Anhang* 645
 - Architektur* 136
 - Auto-Reload* 132
 - besorgen* 123
 - bin* 126
 - Catalina* 123
 - classes* 146
 - common* 126
 - conf* 126
 - Connector* 140
 - Connector-Typen* 140
 - Container* 136
 - Context* 142
 - DataSourceRealm* 179
 - DefaultServlet* 145
 - destroy* 157
 - Digest-Tool* 181
 - doFilter* 157
 - Engine* 141
 - EXE-Version* 124
 - Fehlerbehandlung* 167
 - Fehlerseiten* 167
 - Filter* 155
 - Filter-Beispiel* 156
 - FilterConfig* 157
 - Geschichte* 121
 - Host* 142
 - HTTP Connector* 140
 - installieren* 123
 - Invoker-Servlet* 132
 - JDBCRealm* 182
 - JNDIRealm* 179
 - JNDI-Resource registrieren* 563
 - Konfiguration* 138
 - Lade-Reihenfolge* 146
 - lib* 146
 - Listener* 160
 - logs* 127
 - Mailinglist* 809
 - Manager* 186
 - MemoryRealm* 179
 - Mime-Types registrieren* 165
 - Nested Components* 137
 - Realm* 177
 - Referenzimplementierung* 122
 - reloadable* 132
 - Server* 139
 - server* 127
 - server.xml* 138
 - Service* 139
 - Sessions konfigurieren* 165
 - shared* 127
 - shutdown.bat* 126
 - shutdown.sh* 126
 - starten und stoppen* 125
 - startup.bat* 126
 - startup.sh* 126
 - Tag-Library* 167
 - temp* 127
 - tomcat-users.xml* 179
 - Verteilte Anwendung* 153
 - Verzeichnis-Struktur* 126
 - virtueller Host* 142
 - web.xml* 149
 - webapps* 127
 - WEB-INF* 145
 - Website* 809
 - Wiki* 809
 - Willkommensdateien* 166
 - work* 127
 - Zugriffsrechte* 171
 - Tomcat 3.0 122
 - Tomcat 3.3 122
 - Tomcat 4.0 123
 - Tomcat 5.0 123

Index

Tomcat-Manager 186
 Benutzer einrichten 186
 HTML-Version 187
 URI-Kommandos 187
Transformation 199
Transformer 222, 336, 719
 AbstractDOMTransformer 337
 AbstractSAXTransformer 337
 AbstractTransformer 337
 Anhang 719
 augment 719
 characters 339
 cinclude 720
 CInclude Transformer 720
 endElement 339
 erstellen 336
 erstellen, Beispiel 337
 filter 726
 Filter Transformer 726
 i18n 728
 I18n Transformer 728
 Read DOM Session Transformer 732
 readDOMsession 732
 SourceWriting Transformer 733
 startElement 339
 Tee Transformer 738
 Transformer 336
 Write DOM Session Transformer 739
 writeDOMsession 739
 write-source 733
 xinclude 741
 XInclude Transformer 741
Transformer registrieren 223
Transformer verwenden 223
Transformer
 Augment Transformer 719
 encodeURL 724
 EncodeURL Transformer 724
 log 731
 Log Transformer 731
 sql 735
 SQL Transformer 735
 xslt 742
 XSLT Transformer 742
transformieren 79
TreeModel 444
Tree-Widget 428, 444
Trest-Framework 360

U

übersetzen
 Abschnitte 590
 Attribute 594
 Datumsformate 595
 Sätze 590
 Währungsformate 596
 Wörter 588
 Zahlenformate 596
 Zeitformate 595
Umgebungen 37
Umleitung 242
Unicode 46
Union-Widget 461, 462
Upload 529
 autosave-uploads 530
 enable-upload 529
 enable-uploads 529
 multipart/form-data 529
 overwrite-uploads 530
 Part 531
 PartInMemory 531
 PartOnDisk 531
 upload-directory 529
 upload-max-size 530
 web.xml 529
Upload-Widget 429, 468
URI 203, 805
URL 203, 805
URLs encoden 724
URN 203
US-ASCII 46
User tier 36
UTF-16 46
UTF-8 46

V

Validation-Errors auflisten 481
validieren 805
Validierung 483
ValueChangedEvent 494
Variable
 Sitemap 206
Variant 585
Velocity 690
Velocity Generator 222, 389, 690
 #foreach 390

context 392
implizite Objekte 391
parameters 392
request 391
response 391
session 391
 Velocity Template Language 392
 View 243
 aufrufen 248
 erstellen 244
 global konfigurieren 247
 Label platzieren 245
 Label, mehrere 247
 registrieren 245
 unterstützende Elemente 247
 View-Komponenten 388
 View-Pipeline 244
 virtuelle Pfadangabe 213
 VM 804
 VTL 392

W

w3.org 810
 W3C 805
 w3schools.com 810
 War-Archiv 147
 packen 147
 War-Datei 147
 WARN 304
 Web Continuation
 Properties 769
 Web Tools Platform 100
 web.xml 149, 649
 auth-constraint 173
 auth-method 174
 BASIC 174
 CLIENT-CERT 175
 CONFIDENTIAL 174
 context-param 154, 651
 description 153, 650
 DIGEST 175
 display-name 153, 650
 distributable 154, 651
 ejb-local-ref 657
 ejb-ref 656
 env-entry 656
 error-page 167, 654
 filter 155, 651
 filter-class 156
 filter-mapping 155, 652
 filter-name 156
 FORM 175
 form-error-page 176
 form-login-config 174, 175
 form-login-page 175
 http-method 173
 icon 651
 init-param 164
 INTEGRAL 174
 jsp-config 169, 655
 jsp-file 163
 listener 652
 load-on-startup 164
 locale-encoding-mapping-list 659
 login-config 174, 655
 message-destination 658
 message-destination-ref 658
 mime-mapping 165, 654
 NONE 174
 param-name 154, 164
 param-value 154, 164
 realm-name 174, 175
 Reihenfolge d. Elemente 152
 resource-env-ref 658
 resource-ref 657
 role-name 173
 security-constraint 172, 173, 655
 security-role 177, 656
 service-ref 657
 servlet 162, 652
 servlet-class 163
 servlet-mapping 162, 653
 session-config 165, 653
 session-timeout 165
 taglib 169
 taglib-location 169
 taglib-uri 169
 transport-guarantee 174
 url-pattern 156, 173
 user-data-constraint 174
 web-app 650
 web-resource-collection 172
 web-resource-name 172
 welcome-file-list 166, 654
 Web-Applikation 144
 konfigurieren 154
 Verzeichnisstruktur 145

Index

Web-Continuation 374
 Funktionen 770
WebContinuation 769
 continuation 769
 getChildren 770
 getParent 770
 id 770
 invalidate 770
WEB-INF 145
White-Box-Test 360
Widget-Definition 429
Widget-Objekt 430
Widgets 419, 428
Wildcard 215
Wildcard für Pfadangabe 216
Wildcard URI Matcher 212, 214
Wildcards kombinieren 216
WildcardURI Matcher 695
Woody 413
Write DOM Session Transformer 739
WTP 100

X

Xalan 811
Xerces 102, 113, 811
 Apidoc 106
 installieren 103
XForms 413
XHTML 62
XHTML Serializer 227, 715
XInclude Transformer 224, 249, 741
XInclude-Spezifikation 741
Xindice 811
XLS 228
XML 44
 & 55
 ' 55
 > 55
 < 55
 " 55
 Attribut 49
 Attribute und Namensraum 58
 CDATA 55
 Child 53
 Default-Namensraum 59
 Definitionsformate 65
 Deklaration 45, 53
 DTD 64
 Element 48
 Element-Varianten 48
 Elternelement 53
 Encoding 46
 End-Tag 48
 Entity-Referenzen 54
 Geschwisterelement 53
 Gültigkeit 49
 Kindelement 53
 Kommentare 56
 Kurzform v. Element 48
 Mehrere Namensräume 59
 Namensraum 57
 Namespace 57
 öffnendes Tag 48
 Parent 53
 parsen 45
 Parser 50
 Präfix 58
 Reservierte Zeichen 54
 Root 53
 Root-Element 50
 schließendes Tag 48
 Sibling 53
 Standard-Entities 55
 Start-Tag 48
 Tag 47
 transformieren 79
 validierender Parser 50
 Vorschrift 50
 wellformtness 49
 Wohlgeformtheit 49
 Wurzelement 50, 53
 XHTML 62
 xmlns 58
XML Beschränken 64
XML Dokument 45
XML File Module 273
XML Schema 66
XML Serializer 227, 717
xml.apache.org 810
XML:DB 575
XML4J 102
XMLConsumer 324
XML-Datenbanksysteme 571
 Apache Xindice 573
 Collection 572
 Dokument 572
 eXist 573

- Struktur 572
- Tamino XML Server 573
- XML:DB 575
- XML-Dokumenttypen 573
- XML-enabled 571
- XML-native 572
- Zugriff auf 574
- xmlldb 575
- XMLDB Transformer 577
 - collection 579
 - oid 579
 - registrieren 577
 - type 579
 - verwenden 578
- XML-Dokumenttypen 573
 - datenzentriert 573
 - dokumentenorientiert 574
- XMLFile Module 801
 - {myxml:document/title} 802
 - cacheable 801
 - reloadable 801
- XMLForm 413
- XML-Name 47
- XML-Parser 805
- XMLPipe 325
- XMLProducer 324
- XMLUnit 360
- XPath 74
 - * 76
 - @ 76
 - @* 76
 - Attributknoten 76
 - comment() 76
 - compare() 78
 - concat() 78
 - contains() 79
 - count() 78
 - Elementknoten 75
 - Funktionen 78
 - Knotentypen 75
 - Kommentarknoten 76
 - Lokalisierungspfad 75
 - lower-case() 79
 - name() 79
 - node() 76
 - not() 79
 - position() 78
 - Prädikat 77
 - Processing Instruction 76
 - processing-instruction() 76
 - round() 78
 - starts-with() 79
 - string-length() 79
 - text() 76
 - Textknoten 76
 - upper-case() 79
 - Wildcard 76
- XPath Directory Generator 221, 690
- XPath Exception Selector 266
- xReporter 459, 809
- XSL 73
- XSL-FO 88
 - fo 89
 - fo:block 93
 - fo:flow 93
 - fo:layout-master-set 91
 - fo:list-block 93
 - fo:page-sequence 92
 - fo:region-after 91
 - fo:region-before 91
 - fo:region-body 91
 - fo:region-end 91
 - fo:region-start 91
 - fo:root 89
 - fo:simple-page-master 91
 - fo:table 93
 - fo:table-and-caption 93
 - FO-Prozessor 89, 93
 - master-name 91
 - Namensraum 89
 - page-reference 93
 - Präfix 89
 - präsentierendes Format 89
 - Regionen 92
 - Seitenrand 92
 - XSL-FO-Dokument 89
- XSLT 41, 73, 79
 - Binding shadow 87
 - Kontextknoten 83
 - Kopieren, Teilbaum 84
 - Namensraum 82
 - Parameter 85
 - Parameter, Template 87
 - Präfix 82
 - Prozessor 80
 - Quelldokument 80
 - Stylesheet 80
 - Variable 85

Index

- Version* 82
 - Wurzelement* 82
 - xsl:apply-templates* 84
 - xsl:call-template* 84
 - xsl:copy-of* 84
 - xsl:for-each* 85
 - xsl:if* 85
 - xsl:sort* 84
 - xsl:template* 82
 - xsl:value-of* 83
 - Zieldokument* 80
 - XSLT Transformer 223, 742
 - XSLT-Instruktionen 83
 - XSLT-Prozessor 80
 - XSL-Transformation
 - nach FO* 628
 - nach HTML* 623
 - XSLT-Stylesheet 80
 - XSP 515, 773
 - andere Programmiersprache* 520
 - Anhang* 773
 - Beispiel* 517
 - context* 775
 - Elementübersicht* 775
 - Implizite Objekte* 774
 - Importierte Klassen* 773
 - Java* 520
 - JavaScript* 520
 - Logicsheet* 522
 - Namensraum* 516
 - objectModel* 775
 - parameters* 775
 - Python* 520
 - request* 775
 - resolver* 775
 - response* 775
 - ServerPages Generator* 516
 - session* 775
 - Top-Level-Element* 518, 775
 - Vergleichsoperatoren* 520
 - xsp:attribute* 781
 - xsp:comment* 783
 - xsp:content* 782
 - xsp:element* 780
 - xsp:exit-page* 778
 - xsp:expr* 779
 - xsp:include* 777
 - xsp:init-page* 777
 - xsp:logic* 778
 - xsp:page* 776
 - xsp:param* 783
 - xsp:pi* 782
 - xsp:structure* 776
 - XSP Generator 688
 - xtest 363
 - Vorlage* 364
-
- ## Z
- Zeichensatz 46
 - Zurück-Problem 377
 - Zustandsänderung 369
 - Zustandsautomat 369
 - Zustandsdiagramm 369
 - zvon.org 810