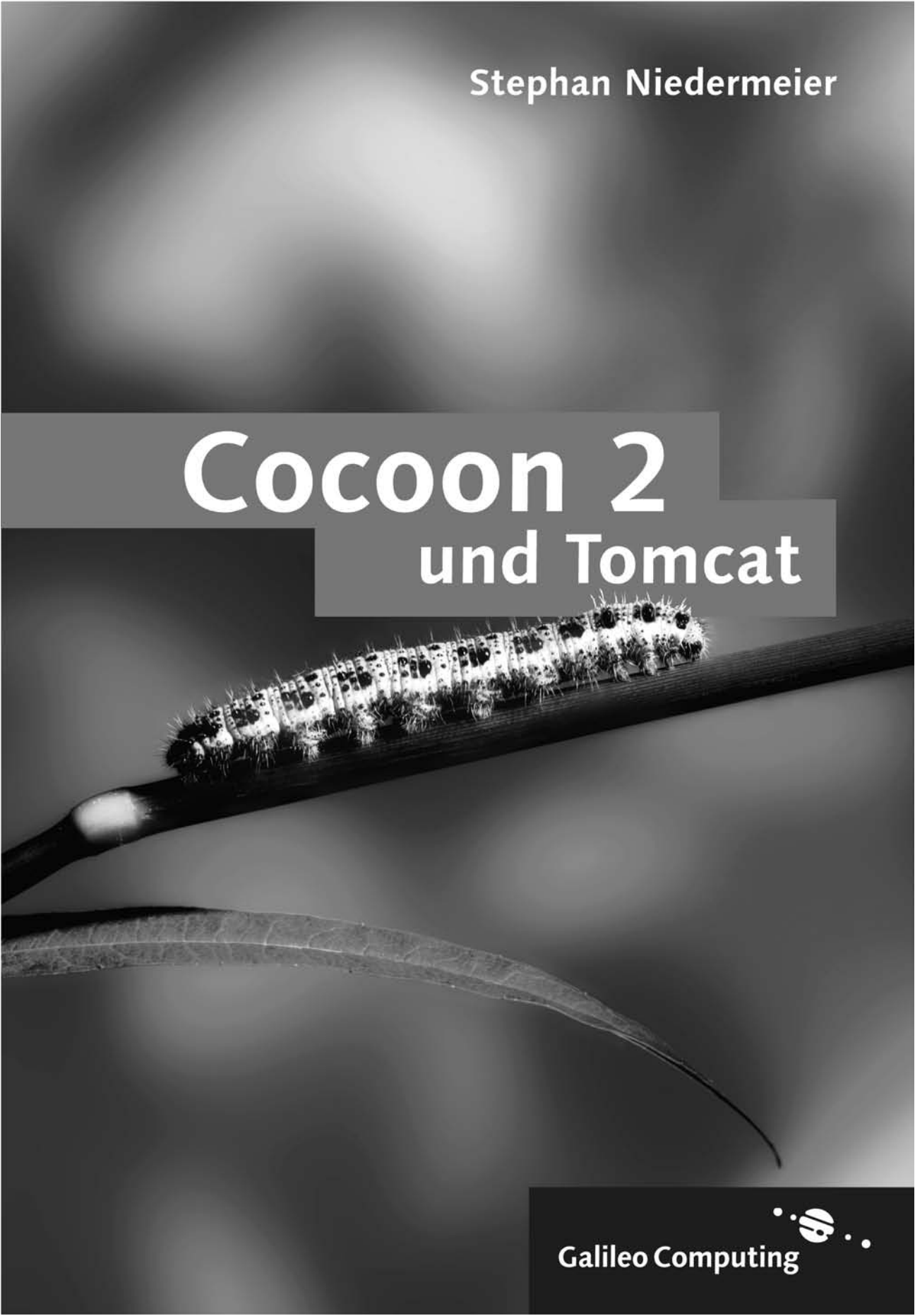
Stephan Niedermeier

Cocoon 2
und Tomcat



Galileo Computing

Liebe Leserin, lieber Leser,

ich freue mich, dass Sie sich für ein Buch von Galileo Computing entschieden haben.

Das Open Source-Projekt Apache-Cocoon braucht in seiner jetzigen Version den Vergleich mit kommerziellen Tools nicht mehr zu scheuen. Innerhalb kurzer Zeit hat es sich zu einem der mächtigsten Frameworks zur Entwicklung von Webanwendungen entwickelt. Unternehmen wie die Commerzbank oder Hewlett-Packard sowie weitere Institutionen und Firmen haben die Leistungsfähigkeit für sich entdeckt und verwenden Cocoon als XML-Publishing-Framework oder haben Teile davon in Ihre Web-Anwendungen integriert.

Ich bin froh, mit Stephan Niedermeier einen Autor gefunden zu haben, der in der Lage ist, Ihnen eine umfassende Einführung in das komplexe Zusammenspiel der Komponenten zu bieten. Mit seiner Hilfe werden Sie Ihre Lernkurve beim Einstieg in Cocoon deutlich senken. Entwicklern und Anwendern eröffnet es den Weg zum produktiven Einsatz von Cocoon. Seit längerer Zeit bietet der Autor eines der meistgelesenen deutschsprachigen Online-Tutorials zur Cocoon-Thematik auf seiner Webseite **www.logabit.com/cocoontutorial/**. Viele Erfahrungen mit eigenen Projekten sind zudem in dieses Buch eingeflossen.

Mit viel Sorgfalt haben Autor und Verlag dieses Buch produziert. Ein perfektes Buch kann es leider dennoch nicht geben. Ich freue mich daher, wenn Sie sich mit Ihren kritischen Anmerkungen an mich wenden oder den Kontakt zum Autor in seinem Forum unter **www.cocoonforum.de** suchen.

Und nun viel Freude bei der Lektüre

Stephan Mattescheck
Lektorat Galileo Computing

stephan.mattescheck@galileo-press.de
www.galileocomputing.de

Galileo Press • Gartenstraße 24 • 53229 Bonn

Auf einen Blick

Inhalt

Vorwort

Seit dem Erscheinen von Cocoon 2 im März 2001 hat es sich immer mehr von einem reinen XML-Publishing-System hin zu einem vollständigen Web-Application-Framework entwickelt, das kaum noch Wünsche offen lässt. Diese Tatsache unterstreichen Unternehmen, wie die Commerzbank oder Hewlett-Packard, sowie Institutionen wie die Raumfahrtbehörde NASA dadurch, dass sie Cocoon als Publishing-Framework verwenden oder in eigene Produkte integrieren. In diesem Buch möchte ich dieser Entwicklung Rechnung tragen.

Zu Beginn erhalten Sie eine grundlegende Einführung in das Thema XML, das für Cocoon unerlässlich ist. Von der Struktur und den Einsatzmöglichkeiten von XML abgesehen werden Sie auch erfahren, durch welche Mechanismen es möglich ist, ein XML-Dokument zu verarbeiten, zu beschränken und in andere Formate zu transformieren. Diese Abschnitte habe ich bewusst etwas umfangreicher gestaltet, um vor allem denjenigen Lesern, die erst wenig Erfahrung auf dem Gebiet XML besitzen, einen Leitfaden durch den Dschungel der XML-Techniken zu geben. Sind die hier erwähnten XML-Themen einmal verstanden, ist es ein Leichtes, sich in der Welt von XML zurechtzufinden und sein Wissen gezielt zu vertiefen. Auf der anderen Seite sind diese Abschnitte auch für erfahrene XML-Entwickler von Nutzen, da hier kompakt die wichtigsten Themen zusammengefasst sind, die für die Verwendung von Cocoon benötigt werden.

Ein weiteres großes Thema wird der Servlet-Container Tomcat bilden. Sie werden nicht nur in die Lage versetzt werden, diese Anwendung zu installieren und zu konfigurieren, sondern auch einen tiefen Einblick in dessen interne Architektur erhalten. Eine weitere Einführung in die zugehörigen Techniken, wie JSP, Servlets, Filter oder Listener, um nur einige zu nennen, rundet diesen Abschnitt ab und ermöglicht Ihnen, die Arbeitsweise von Tomcat zu verstehen und diesen Servlet-Container alleine oder in Verbindung mit Cocoon produktiv einsetzen zu können.

Nachdem Sie sich die wichtigsten Grundlagen in den vorangegangenen Kapiteln erarbeitet haben, werden Sie schrittweise an Cocoon herangeführt. Beginnend mit der Installation werden Sie erfahren, wie Sie Cocoon für einen ersten Testlauf konfigurieren können. Im Anschluss daran werde ich Ihnen die wichtigsten Konzepte von Cocoon vermitteln. Dazu zählt unter anderem die Erklärung der so genannten Sitemap, den einzelnen Sitemap-Komponenten sowie der Pipeline-Technik. In den

weiteren Kapiteln wird anschließend ein tiefer Einblick in die Cocoon-Architektur gegeben und erörtert, wie eigene Logik-Bestandteile und -Erweiterungen auf verschiedene Arten in dieses Framework integriert werden können. Eine kleine Beispiel-Applikation rundet am Ende diesen Abschnitt ab und fasst die wichtigsten Themen noch einmal kompakt zusammen.

Der umfangreiche Anhang bildet das Ende dieses Buches. Er umfasst eine nützliche Referenz zu den Konfigurationsdateien von Tomcat und eine Zusammenfassung der wichtigsten Komponenten und Spracherweiterungen von Cocoon.

Auf der beiliegenden Buch-CD finden Sie zahlreiche Tools und Anwendungen, die Sie bei der Entwicklung unterstützen. Die wichtigsten Listings sind hier ebenfalls in digitaler Form vertreten.

Dieses Buch ist so strukturiert, dass die einzelnen Kapitel auf einander aufbauend sind. Aus diesem Grund können Sie es also gemütlich von Anfang bis Ende durchlesen. Dabei werden alle relevanten Themen behandelt.

Natürlich kann Ihnen dieses Buch aber auch als generelles Nachschlagewerk zu allen wichtigen Themen rund um Cocoon dienen.

Um Ihnen in beiden Fällen die Orientierung zu erleichtern, sind bestimmte Absätze im Text durch entsprechende Symbole markiert:

Wenn ein Absatz mit diesem Symbol markiert ist, finden Sie an dieser Stelle einen besonderen Hinweis zum Thema.

Wichtige Hinweise, die Sie unbedingt beachten sollten, werden durch dieses Symbol gekennzeichnet.

Mit diesem Symbol wird darauf hingewiesen, dass an dieser Stelle ein größeres Beispiel beginnt.

Absätze, die eine Zusammenfassung wichtiger Informationen von zuvor behandelten Themen enthalten, sind mit diesem Symbol markiert.

Generell können Sie alle relevanten Listings auch auf der Buch-CD vorfinden. Falls in einem Abschnitt allerdings dieses Symbol vorkommt,

ist zusätzlich ein erweitertes Listing oder spezielle Software auf der Buch-CD verfügbar.

Ich hoffe, dass Ihnen dieses Buch einen leichten Einstieg in die Welt von Cocoon und Tomcat eröffnet und als informatives Nachschlagewerk ein ständiger und treuer Begleiter bei Ihren Projekten sein kann.

In diesem Sinne, viel Spaß mit dieser Lektüre und viel Erfolg beim Erstellen von plattformunabhängigen, skalierbaren Web-Applikationen!

München, März 2004
Stephan Niedermeier

1 Einführung in Cocoon

1 Einführung in Cocoon

In diesem Kapitel erhalten Sie einen umfassenden Einblick in Cocoon und seine Aufgabengebiete.

In diesem Kapitel möchte ich Sie an Cocoon heranführen und Ihnen einen Überblick über dessen Einsatzgebiete geben. Dabei werden wir unter anderem die wesentlichen Aspekte moderner Web-Applikationen und deren Entsprechungen in Cocoon betrachten.

1.1 Geschichte von Cocoon

Im Jahre 1998 arbeitete der italienische Student Stefano Mazzocchi an einer Möglichkeit, effektiv das Layout der Website java.apache.org von dessen Inhalt zu trennen und genauso schnell wieder verbinden zu können. Zu diesem Zeitpunkt war eine solche Vorgehensweise noch nicht sehr weit verbreitet. Stefano entschied sich, für die Trennung der beiden Bereiche XSLT einzusetzen, das zu diesem Zeitpunkt erst als Working-Draft verfügbar und von der heutigen Mächtigkeit noch weit entfernt war.

Cocoon 1.0

Ende 1998 hatte Stefano ein 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.

Nach einem Jahr Entwicklung hatte Cocoon 1.x bereits einen stattlichen Umfang erreicht. Aufgrund der Tatsache, dass zu diesem Zeitpunkt die XML-Welt 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. Der Grund lag darin, dass 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 einen Aufruf, Cocoon grundlegend umzuschreiben und die Architektur zukunftsweisend zu gestalten. Nach langen zwei Jahren Entwicklung war es endlich soweit: Cocoon 2 (C2) wurde veröffentlicht. Neben der offenen Architektur waren Per-

formance, 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 ihrer Umgebung lauffähig ist. Die ausschließliche Bindung an ein Servlet entfiel hiermit. Cocoon ist somit auch als Standalone-Applikation verwendbar und kann über ein Kommandozeilen-Tool verwaltet oder direkt in andere Java-Applikationen eingebettet werden. Trotzdem ist die am weitesten verbreitete Methode für den Einsatz von Cocoon der Betrieb als Web-Applikation innerhalb eines Servlet-Containers. Die Schnittstelle hierbei bildet in diesem Fall weiterhin ein Servlet.

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

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

Cocoon 2.1

Einen weiteren Quantensprung brachte die Cocoon-Generation ab der Version 2.1, die Mitte 2003 veröffentlicht wurde. Darin wurden einige zukunftsweisende Mechanismen implementiert, die das Erstellen von Web-Applikationen noch einfacher und übersichtlicher machen. Die einschlägigste Veränderung ist dabei sicherlich die Einführung der Continuation-Technik, die durch Flowscripts eine komfortable Alternative zu XSP und Actions bietet.

1.2 Anforderungen an Web-Applikationen

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

Bei der Entwicklung moderner Web-Applikationen müssen heutzutage viele Punkte berücksichtigt werden, die über das einfache Erzeugen einer HTML-Seite und deren Auslieferung hinausgehen. Diese Punkte

sind natürlich von Anforderung zu Anforderung verschieden, können aber generell in folgende Kategorien eingeteilt werden:

- Trennung von Layout, Inhalt und Logik
- Plattformunabhängigkeit
- Personalisierung
- Modularität und einfache Erweiterung
- Internationalisierung
- Skalierbarkeit
- verschiedene Zielplattformen (Cross-Media-Publishing)
- verschiedene Datenquellen (XML, DB, LDAP, ...)
- einfache und standardisierte Wartung

Framework

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

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, der bei der Realisierung einer Web-Applikation beachtet werden sollte und dessen Notwendigkeit in den vergangenen Jahren auch von den letzten Web-Entwicklern erkannt wurde, ist die Trennung der Layout-, Inhalt- und Logik-Schichten voneinander.

MVC

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

Model 2

MVC ist nicht gerade 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 für die Applikation zur Verfügung. Neben der Verwaltung dieser Daten ist sie in der Regel auch dafür verantwortlich, diese zu ändern.
- View
  Übernimmt die Darstellung der Daten aus dem Model für den Endanwender.
- Controller
  Der Controller 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 so genannte Beans[1] 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, die den gesamten Applikationsfluss verwaltet und View- sowie Model-Schicht miteinander verbindet.

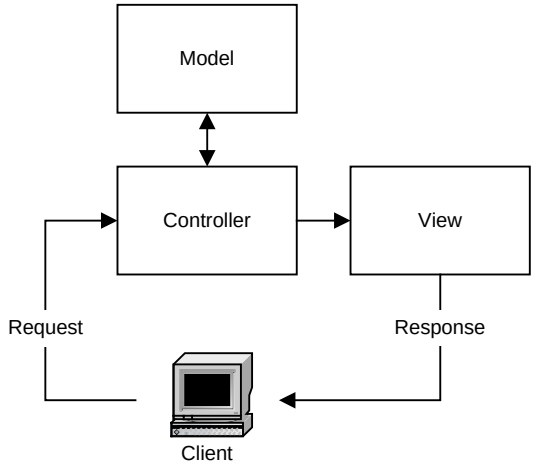


Abbildung 1.1 Das MVC-Konzept im Request-Response-Zyklus

In Cocoon wird der Controller meist von Actions, Logicsheets, Flowscripts und auch immer von der Sitemap realisiert. Die Model-Schicht

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

übernehmen häufig bestimmte Sitemap- oder eigene Avalon-Komponenten, und die View-Schicht kann durch verschiedene Transformer – wie beispielsweise dem XSLTransformer oder dem JXTemplateTransformer – erstellt werden.

Haben Sie keine Angst, falls Sie das eine oder andere der hier genannten Instrumente noch nicht kennen. Wir werden später alle ausführlich besprechen.

1.2.2 Plattformunabhängigkeit

Unter dieser Voraussetzung versteht man, dass eine Anwendung nicht an ein bestimmtes Betriebssystem gebunden ist. Dank der Tatsache, dass Cocoon vollständig in Java geschrieben ist, ist diese Voraussetzung von vorneherein gegeben. Sie können Cocoon auf jedem Betriebssystem betreiben, für das es eine aktuelle Java-Laufzeitumgebung gibt. Vor allem sind dies natürlich 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, und zum anderen, um seine »Surf-Gewohnheiten« festzuhalten und später auszuwerten.

Die Implementierung einer solchen Funktionalität ist von Ihrer eigenen Applikation abhängig. Cocoon stellt Ihnen dabei die nötigen Werkzeuge zur Verfügung, um eine solche Realisierung bewerkstelligen zu können.

1.2.4 Modularität und einfache Erweiterbarkeit

Unabhängige Bausteine, die sich unter bestimmten Bedingungen beliebig und vor allem 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 bzw. ausgetauscht werden können.

Sollte trotzdem für Ihr spezielles Problem einmal eine Komponente fehlen, können Sie relativ einfach eine neue erstellen oder eine vorhandene erweitern. Zu diesem Zweck existiert eine hervorragende Struktur

an Interfaces und Basisklassen, die eine solche Erweiterung äußerst einfach werden lassen.

1.2.5 Internationalisierung

Im Zeitalter der Globalisierung ist es auf vielen Websites unentbehrlich geworden, den Inhalt in verschiedenen Sprachen und angepasst auf einige Lokalitäten auszuliefern.

Cocoon stellt hierfür eine spezielle Komponente – den I18nTransformer – zur Verfügung. Er unterstützt Sie dabei, sowohl Texte als auch Datums-, Zahlen- 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 größeren Anfrageflut standhalten muss und deshalb die Leistungsfähigkeit entsprechend erweiterbar sein sollte. Zum anderen muss die Web-Applikation selbst mit den eigenen Anforderungen mitwachsen können, und es müssen sich möglichst viele Probleme zentral damit lösen lassen.

Vor allem durch einen umfangreichen Caching-Mechanismus und die Fähigkeit, Komponenten konfigurativ poolen zu können, ist Cocoon auch einer großen Anfrageflut durchaus gewachsen. Einige Benchmarks, die in letzter Zeit durchgeführt wurden, unterstreichen diese Tatsache.

Durch die erweiterbare Komponentenarchitektur und die Vielzahl der unterschiedlichsten Schnittstellen ist eine Erweiterung einer Cocoon-Applikation immer gewährleistet.

1.2.7 Cross-Media-Publishing

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

Angenommen, Sie möchten Ihre aktuellen Geschäftszahlen publizieren. Dies soll zum einen in Form einer HTML-Seite geschehen. Daneben werden diese Zahlen aber zusätzlich von der Buchhaltung in Form einer Excel-Datei, von verschiedenen Kunden als Offline-Version im

PDF-Format, der Druckerei im PostScript-Format und von Außendienstmitarbeitern über das WAP-Handy benötigt. Viele Frameworks würden an diesem Punkt an ihre Grenzen stoßen, da der Aufwand zum Erzeugen und Ausliefern dieser verschiedenen Formate häufig enorm ist. Dabei stellt sich auch hier wiederum das Problem, Inhalt, Layout und Logik voneinander zu trennen, um die programmierten Werkzeuge auch für andere Dokumente wieder verwenden zu können. Im Fall eines HTML-Dokuments ließe sich dieses Problem mit Sicherheit relativ einfach lösen, da es hierfür 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 FOP[2] oder für Excel-Dokumente das Apache-Jakarta-Projekt POI[3] verwendet. Somit ist sichergestellt, dass sich alle Teilbereiche unabhängig von Cocoon kontinuierlich weiterentwickeln und somit die stetigen Erweiterungen und Verbesserungen dieser Projekte auch zur Aufwertung von Cocoon selbst führen. Nicht selten ist es der Fall, dass Cocoon-Anwender und -Entwickler Vorschläge und Verbesserungen für andere Projekte bereitstellen. Somit kommt eine Symbiose zustande, von der alle beteiligten Projekte profitieren.

Die am häufigsten verwendeten Ausgabeformate, die von Cocoon standardmäßig produziert werden, lauten:

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

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

2 http://xml.apache.org/fop
3 http://jakarta.apache.org/poi

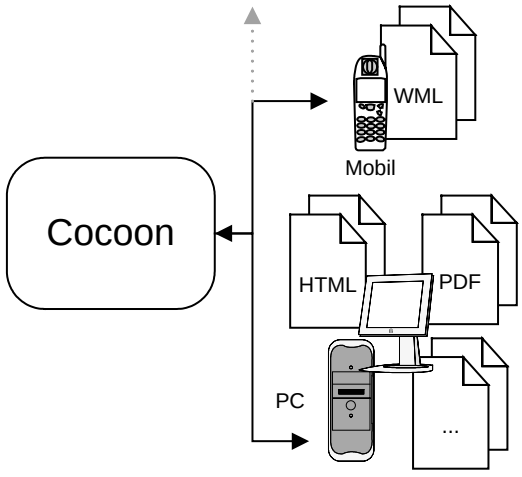


Abbildung 1.2 Cross-Media-Publishing mit Cocoon

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
- Verbindungspooling
- Überwachung der Verbindungen

JDBC

Aufgrund der Tatsache, dass Cocoon in Java programmiert ist und somit selbst erstellte Erweiterungen ebenfalls diese Sprache verwenden können, bietet sich für den Zugriff auf Datenbanken JDBC an. Dadurch ist bereits der erste Punkt »Abstraktion der Datenbankverbindung« erfüllt. Durch JDBC lassen sich nahezu alle existierenden Datenbanken einbinden. Ein Austausch kann dabei ohne jegliche Änderungen am Code der Web-Applikation erfolgen.

Datasource

Für die übrigen Punkte bringt Cocoon das Konzept der **Datasources** mit. Hierdurch können Daten-Ressourcen an einer zentralen Stelle unter einem eindeutigen Namen registriert und konfiguriert werden. Daneben kann für jede einzelne Datasource speziell ein Tuning des

standardmäßig vorhandenen Verbindungspools erfolgen, sowie das Logging entsprechend konfiguriert werden.

andere Datenquellen

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 immer in Form von XML zur Verfügung. Dieses XML-Dokument kann anschließend von weiteren Cocoon-Komponenten entsprechend verarbeitet werden. Auf diese Weise ist auch hier eine Schnittstelle vorhanden, an die unzählige Datenquellen angeschlossen werden können. Die einzige Voraussetzung hierfür ist, dass die Daten in Form von XML darstellbar sind. Für viele solcher 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 von anderen Websites in Form von XML einzubinden.

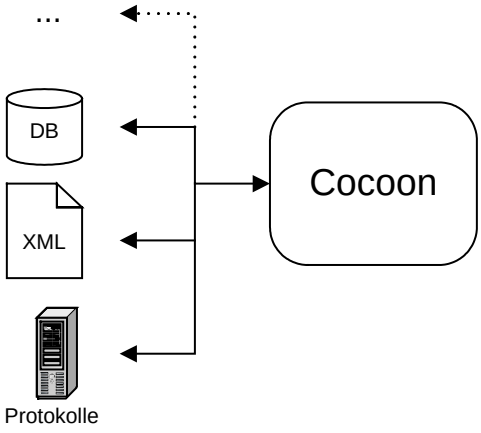


Abbildung 1.3 Cocoon besitzt fertige Schnittstellen zu den verschiedensten Datenquellen.

1.3 Ein kleiner Einblick in Cocoon

Nachdem Sie nun erfahren haben, welche Eigenschaften Web-Frameworks mitbringen sollten und wie diese in Cocoon realisiert werden, wollen wir uns in diesem Abschnitt im Konkreten ansehen, welche Stellung Cocoon in einem Server-Park besitzt und wie seine allgemeine Funktionsweise ist.

1.3.1 Cocoon ist Middleware

Three-Tier-Modell

Um eine bestmögliche Wart-, Erweiter-, und Austauschbarkeit gewährleisten zu können, ging man in der Servertechnik dazu über, die einzelnen Bereiche einer Anwendung in verschiedene Gruppen zu untergliedern und diese Gruppen auf eigene Server auszulagern. Eine solche Architektur nennt man **n-Tier-** oder **n-Schichten-Modell**. 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.

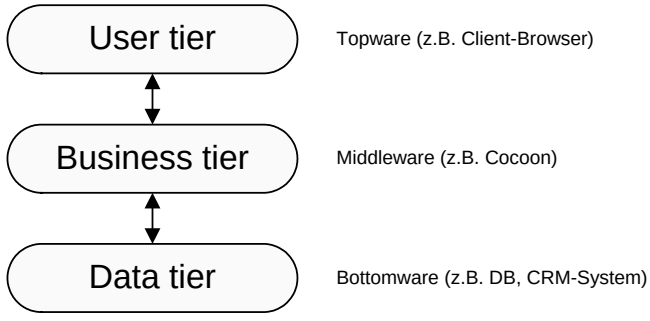


Abbildung 1.4 Die Three-Tier-Architektur

Das User Tier ist für die Darstellung der Daten für den Endanwender verantwortlich. In der Web-Entwicklung übernimmt dies in der Regel der Webbrowser des Clients. Das Business Tier ist der Logikbereich einer Anwendung. Cocoon füllt genau diese Position in einer Web-Applikation aus. Es reagiert auf Benutzereingaben, erzeugt die entsprechenden Ausgaben und kommuniziert auf der anderen Seite mit dem Data Tier, das für die Datenhaltung im allgemeinen Sinn zuständig ist. Dies können beispielsweise Datenbanken, CRM- oder Warenwirtschaftssysteme sein.

1.3.2 Cocoon basiert auf Avalon

Cocoon basiert seit der Version 2.0 vollständig auf dem Apache-Projekt Avalon[4]. Avalon ist selbst ebenfalls ein Framework, das aus JServ – einem Java-Webserver der Apache Group – hervorging. Damals versuchte man, eine allgemein gültige Vorgehensweise für die objekt- und komponentenorientierte Programmierung (COP[5]) zu schaffen. Mit der

4 http://avalon.apache.org
5 Component Oriented Programming

Zeit wurde Avalon zu einem Top-Level-Projekt der Apache Group, das inzwischen in zahlreichen großen Anwendungen Einsatz findet.

Durch die Berücksichtigung von COP haben es die Entwickler geschafft, mit Avalon eine allgemein gültige Vorschrift für die Erstellung wieder verwendbarer, austauschbarer und kombinierbarer Komponenten zu erstellen. Wenn Sie später beispielsweise etwas über Sitemap-Komponenten lernen, so sind dies gleichzeitig auch Avalon-Komponenten. Fast alle Teile von Cocoon wurden als Avalon-Komponenten realisiert und bieten deshalb ein Höchstmaß an Flexibilität. Es ist somit möglich, nahezu jeden Bereich von Cocoon den eigenen Bedürfnissen anzupassen, ohne direkten Eingriff in andere Bereiche der Anwendung vornehmen zu müssen. Durch eine Art Plug-in-Mechanismus ist ein Austausch oder eine Erweiterung einzelner Komponenten relativ einfach möglich.

Avalon selbst unterteilt sich wiederum in einige Subprojekte, von denen nur die nachfolgenden in Cocoon Verwendung finden:

- **Framework**
  Dieses Subprojekt stellt den Kern von Avalon dar. Es besteht aus einer Reihe von Interfaces und Regeln, die vorschreiben, wie Avalon-Komponenten implementiert werden müssen.
- **Excalibur**
  Das Subprojekt Excalibur stellt eine Sammlung von Standardimplementierungen (Komponenten) der verschiedenen – durch Framework – definierten Interfaces dar.
- **LogKit**
  Dieses Subprojekt stellt ein umfangreiches Logging-Framework für die Avalon-Komponenten zur Verfügung. Unter anderem abstrahiert es beispielsweise das bekannte Logging-System Log4j[6].

1.3.3 Umgebungen

Wie Sie zu Beginn dieses Kapitels bereits erfahren haben, wurde Cocoon als abstrakte Maschine entworfen. D. h., dass es – außer an Java – nicht an eine konkrekte Umgebung gebunden ist. Sie können Cocoon sowohl als Standalone-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.

6 http://logging.apache.org/log4j

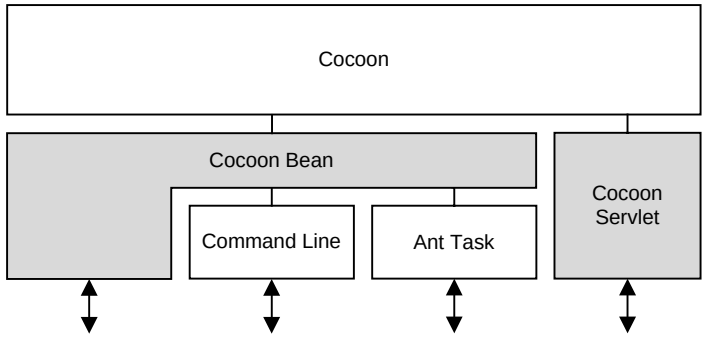


Abbildung 1.5 Die verschiedenen Möglichkeiten, Cocoon zu betreiben

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:

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.

Somit ist es beispielsweise möglich, offline statische Versionen von Cocoon-Websites zu erzeugen. Cocoon besitzt hierfür einen so genannten Crawler, der automatisch allen internen Links folgt und die dahinter liegenden Zieldokumente, wie HTML, PDF usw., statisch erzeugt und in einem vordefinierten Zielverzeichnis ablegt.

Für das Starten des CLI können Sie das Skript `cocoon.bat` für Windows bzw. `cocoon.sh` für Linux mit dem Parameter `cli` verwenden. Ein Beispiel:

```
./cocoon.sh cli <parameter>
```

Diese Dateien befinden sich im Source-Verzeichnis von Cocoon in der obersten Ebene. Sie erwarten eine kompilierte Version von Cocoon im Verzeichnis `build/webapp`. Durch den zusätzlichen Parameter `-h` können Sie sich alle verfügbaren weiteren Optionen auflisten lassen.

cli.xconf

Eine Möglichkeit, das CLI zu steuern, besteht in der Verwendung der verfügbaren Parameter. Die weitaus komfortablere Art und Weise ist jedoch, es über eine zentrale Konfigurationsdatei zu betreiben. Ein Beispiel für eine solche Konfigurationsdatei finden Sie in der Datei `cli.xconf`. Diese können Sie als Grundlage für Ihre eigene Konfiguration verwenden. Am besten, Sie erstellen sich zuvor jedoch eine Sicherheitskopie davon. Die verfügbaren Einstellungen sind allesamt ausführlich in dieser Datei beschrieben. Um das CLI mit diesen Konfigurationen zu starten, müssen Sie mit der Option `–x` den Pfad zur Datei angeben. Ein Beispiel:

```
./cocoon.sh cli -x cli.xconf
```

Anschließend startet das CLI und führt alle konfigurierten Anweisungen aus.

Ant-Task

Eine weitere, immer beliebter werdende Methode für den Offline-Betrieb von Cocoon ist der verfügbare Cocoon-Task für das Build-Tool Ant. Hiermit können Sie Cocoon direkt durch Ant ausführen und mit anderen Ant-Tasks kombinieren. Mehr Informationen hierzu erhalten Sie auf den Websites 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 und die auch den Schwerpunkt in diesem Buch bilden wird, ist die als Web-Applikation innerhalb eines Servlet-Containers. Dabei kommuniziert der Servlet-Container mit der Cocoon-Applikation über ein einzelnes Servlet, das **Cocoon-Servlet**.

Das Cocoon-Servlet »wrappt« 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 J2EE-Welt gewährleistet.

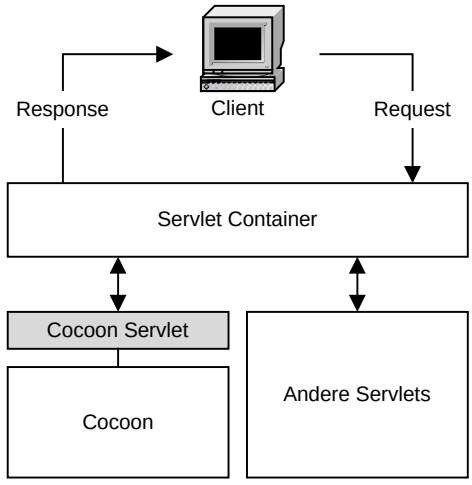


Abbildung 1.6 Die Rolle des Cocoon-Servlets

Falls Sie nicht wissen, was ein Servlet oder ein Servlet-Container ist, sollten Sie sich unbedingt auch Kapitel 6, *Tomcat*, ansehen. Dort erhalten Sie ein fundiertes Basiswissen über diese Technologien, das es Ihnen im weiteren Verlauf dieses Buches erleichtert, Cocoon richtig einzuordnen und seine Funktionsweise in einer Web-Umgebung verstehen zu können.

1.3.4 Request-Response-Zyklus

Cocoon arbeitet – ähnlich wie das HTTP-Protokoll – nach dem Request-Response-Zyklus, d. h., dass an Cocoon 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 werden konnte. Es ist wichtig zu verstehen, dass Cocoon – egal in welcher Umgebung es betrieben wird – immer 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 eine URI in Ihrem Webbrowser gestellt, wie zum Beispiel:

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

Durch diese URI wird zunächst das Cocoon-Servlet aufgerufen. Dieses entfernt alle nicht benötigten Angaben aus der URI. Letztendlich verwendet Cocoon lediglich den Pfad `index.html`, um eine Antwort zu erzeugen. Anhand dieser Pfadangabe untersucht Cocoon seine Sitemap und erzeugt – falls eine Übereinstimmung gefunden werden konnte – ein Ergebnisdokument, das über das Cocoon-Servlet und den Servlet-Container zurück an den Client gesendet wird.

Falls Sie sich nun fragen, was denn eine Sitemap ist, so kann ich Ihnen kurz antworten, dass dies die zentrale Datei in Cocoon ist, in der unter anderem alle Anfragen entsprechend geroutet werden. Am besten aber ist es, Sie lesen einfach weiter, denn spätestens in Kapitel 8, *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 eigentlich XML?

Innerhalb von Cocoon existiert eine so genannte 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 verändern, dass das Zielformat (z.B. PDF) erstellt bzw. das Zielsystem (z.B. Browser) die erzeugten XML-Daten auch auswerten kann. Die einzelnen Komponenten innerhalb einer solchen Pipeline kommunizieren die Daten immer in Form von XML untereinander.

Nach XML und zurück

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

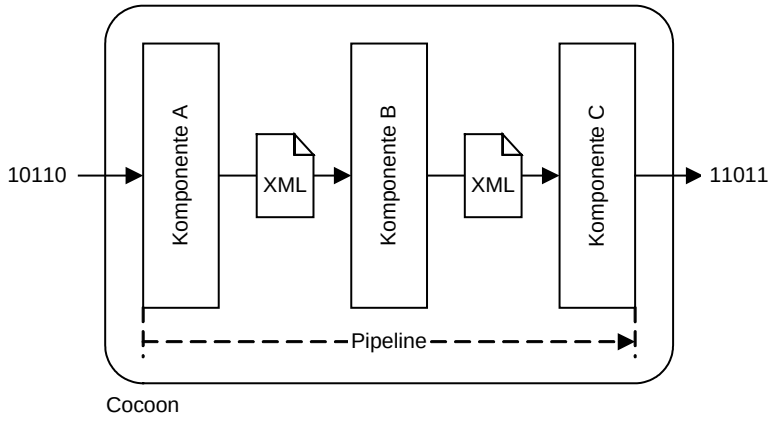


Abbildung 1.7 Übersetzung in XML und zurück mit Cocoon

Da erfreulicherweise immer mehr Anbieter von Systemen die externe Kommunikation auf XML realisieren, ist die Übersetzung von bzw. zur Zielsprache oft nicht mehr notwendig.

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 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-Dokument zu erstellen, das die Transformation in das gewünschte Zielformat übernimmt. Alle anderen Teile der Web-Applikation bleiben in der Regel unberührt.

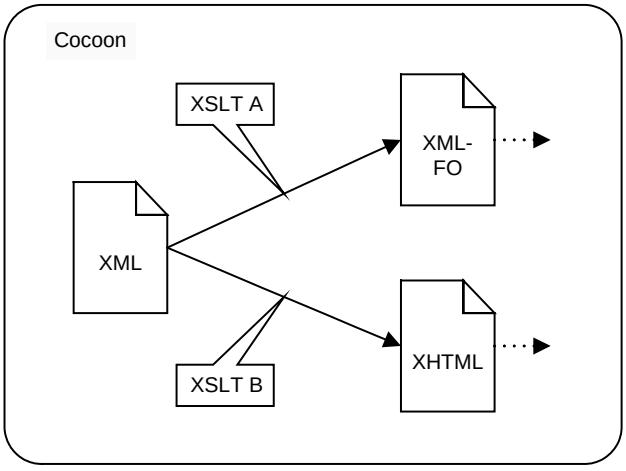


Abbildung 1.8 XSLT transformiert ein XML-Dokument in ein anderes.

1.5 Zusammenfassung

In diesem Kapitel haben Sie nun einen kleinen Einblick 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 die dazugehörenden Techniken. Dies ist nicht nur für »Newbies« auf diesem Gebiet interessant, sondern kann auch als zentrales Nachschlagewerk zu den wichtigsten XML-Themen dienen.

7 Cocoon installieren

7 Cocoon installieren

In diesem Kapitel erfahren Sie, woher Sie Cocoon beziehen, wie Sie es kompilieren und installieren können. Außerdem wird kurz das Build-Tool Ant vorgestellt.

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 Binary[1] 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 und sogar verärgern. Bei näherer Betrachtung macht es jedoch durchaus Sinn, das Framework in unkompiliertem Zustand auszuliefern. Der Grund hierfür sind die so genannten **Blocks**.

7.1 Blocks

Das gesamte Framework Cocoon setzt sich aus dem Core[2] und einzelnen Blocks zusammen. Ein Block bildet dabei eine Gruppe von Klassen, Ressourcen und Konfigurationseinstellungen für einen ganz bestimmten Anwendungsbereich. Der Block `portal-fw` stellt beispielsweise ein Framework für das Erstellen eines Web-Portals zur Verfügung. Ein anderer Block `lucene` integriert die Suchmaschine Lucene in Cocoon. Neben den notwendigen Klassen und Ressourcen wird mit einem Block in der Regel auch ein kleines Beispiel mit installiert.

Aufgrund der Tatsache, dass für eine eigene Web-Applikation in aller Regel nur eine kleine Teilmenge der verfügbaren Blöcke benötigt wird, ist es häufig gewünscht, 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 mehr im Sourcecode auszuliefern. Das Kompilieren muss vom Benutzer selbst durchgeführt werden. Durch eine spezielle Konfigurationsdatei kann er bestimmen, welche Blöcke bei der Kompilierung mit eingebunden werden sollen.

1 kompilierte Version
2 Hauptbestandteile

In der Version 2.2 wird Cocoon ein überarbeitetes Blockkonzept besitzen. Damit wird es möglich werden, die einzelne Blöcke als eine Art unabhängiger Plug-ins zu installieren oder zu deinstallieren. Eine Kompilierung wird dann aller Wahrscheinlichkeit nach entfallen.

7.2 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 dieser Website werden Ihnen neben dem aktuellen Release auch alte Versionen von Cocoon angeboten.

7.2.1 Checkout per CVS

Eine Alternative zum Download von der Website bietet Ihnen der Zugang per CVS[3]. Hiermit haben Sie die Möglichkeit, die Cocoon-Sourcen direkt aus dem Versionierungssystem zu laden. Neben dem aktuellen Release können Sie sich somit auch die neusten Alpha- und Betaversionen herunterladen und testen.

Voraussetzung hierfür ist zunächst, dass Sie einen CVS-Client installiert haben. Diese sind in der Regel kostenlos über das Internet erhältlich. Eine kleine Liste grafischer CVS-Clients:

- TortoiseCVS (Win)
  http://www.tortoisecvs.org
- WinCVS/MacCVS (Win/Mac)
  http://www.wincvs.org
- jCVS (Java)
  http://www.jcvs.org/
- SmartCVS (Java)
  http://www.smartcvs.com/
- TkCVS (Unix/Linux)
  http://www.twobarleycorns.net/tkcvs.html
- Cervisia (Linux)
  http://cervisia.sourceforge.net/

3 Concurrent Versions System: http://www.cvshome.org

Ich werde den Checkout[4] allgemein gültig anhand des Kommandozeilen-Clients von CVS erklären. Dieser Client ist Bestandteil von CVS und wird unter Linux in den meisten Fällen durch den Distributor als RPM-Package zur Verfügung gestellt. Nach der Installation ist der Client sofort einsatzbereit. Unter Windows müssen Sie sich dieses Programm separat besorgen. In WinCVS – zum Beispiel – wird die Datei `cvs.exe` mitgeliefert, die Sie verwenden können. Am besten, Sie binden sie in Ihre Windows-Path-Variable ein. Dann können Sie dieses Programm zukünftig aus jedem beliebigen Verzeichnis heraus verwenden.

Die nachfolgende Vorgehensweise ist unter Windows und Linux/Unix identisch. Nachdem Sie sich vergewissert haben, dass Sie einen funktionierenden CVS-Client installiert haben, starten Sie Ihre Konsole und wechseln Sie in das Verzeichnis, in das Sie eine Kopie der CVS-Version von Cocoon legen möchten. Als Nächstes müssen Sie sich zunächst am CVS-Server anmelden. Geben Sie hierfür folgendes Kommando ein:

```
cvs -d :pserver:anoncvs@cvs.apache.org:/home/cvspublic↵
login
```

Anschließend werden Sie nach dem Passwort gefragt. Dieses lautet `anoncvs`. Geben Sie es ein und bestätigen Sie. Als nächsten Schritt können Sie die entsprechende Cocoon-Version herunterladen. Verwenden Sie hierfür folgenden Befehl:

```
cvs -d :pserver:anoncvs@cvs.apache.org:/home/cvspublic↵
checkout cocoon-2.1
```

Sie erhalten automatisch die aktuelle Version[5] aus dem CVS. Falls Sie die Sourcen aus einem anderen Modul erhalten möchten, müssen Sie dessen Namen am Ende entsprechend ändern. Eine Übersicht über die verfügbaren CVS-Module können Sie online per ViewCVS erhalten:

http://cvs.apache.org/viewcvs.cgi/

Um beispielsweise die Entwicklerversion 2.2 von Cocoon zu erhalten, müssen Sie den Modulnamen `cocoon-2.2` verwenden.

Nach dem erfolgreichen Checkout sollten Sie sich wieder aus dem CVS-System ausloggen:

```
cvs -d :pserver:anoncvs@cvs.apache.org:/home/cvspublic↵
logout
```

4 Download aus dem CVS
5 HEAD

Anschließend befindet sich die heruntergeladene CVS-Version von Cocoon im aktuellen Arbeitsverzeichnis.

7.3 Cocoon kompilieren

Nachdem Sie sich eine aktuelle Version von Cocoon entweder per CVS oder über die entsprechende Website besorgt und in letzterem Fall entpackt haben, können Sie Cocoon nun kompilieren.

7.3.1 Ant

Der gesamte Build-Vorgang wird automatisch durch das Java-Tool Ant[6] ausgeführt. Es ist eines der bekanntesten und erfolgreichsten Projekte der Jakarta Apache Group und kann auf deren Websites kostenlos heruntergeladen werden:

http://ant.apache.org/

Ant kann durch eine XML-Datei so konfiguriert werden, dass mit nur einem Befehl eine Kette von zahlreichen Aktionen gestartet wird. Zu diesen Aktionen zählen im Beispiel von Cocoon das Kompilieren der verschiedenen Java-Klassen, das Zusammenführen der benötigten Dateien und das Vorkonfigurieren der Konfigurationsdateien, um nur einige zu nennen.

Ant wird mit Cocoon ausgeliefert.

Normalerweise erhalten Sie zusammen mit Cocoon auch eine Ant-Version ausgeliefert, die automatisch für die Kompilierung verwendet wird. Eine eigene Installation von Ant ist also in der Regel nicht erforderlich. Da es jedoch auch für spätere Zwecke hilfreich sein kann, eine eigene Ant-Installation zu besitzen, möchte ich Ihnen kurz erklären, wie Sie Ant auf Ihrem System dauerhaft installieren können.

Entpacken Sie das heruntergeladene Archiv zunächst in ein Verzeichnis Ihrer Wahl. Im Unterverzeichnis `bin` finden Sie schließlich eine Reihe von Skripten, durch die Ant gestartet werden kann. Für Windows-Systeme ist dies die Datei `ant.bat` und unter Linux das Shell-Skript `ant`.

PATH

Um von jedem beliebigen Verzeichnis heraus Ant direkt starten zu können, sollten Sie das für Ihr Betriebssystem zutreffende Skript in die Umgebungsvariable `PATH` mit aufnehmen.

build.xml

Wie bereits angesprochen, werden die Aktionen (Tasks), die Ant durchführen soll, in einer XML-Datei festgelegt. Diese Datei lautet standardmäßig `build.xml` und befindet sich in der Regel in der ersten Ebene

6 http://jakarta.apache.org/ant

des Source-Verzeichnisses der zu kompilierenden Applikation. Wenn Sie Ant ausführen möchten, wechseln Sie zunächst in dieses Verzeichnis und starten Sie Ant durch den Aufruf des Befehls `ant`. Ein Beispiel:

```
cd /home/user/mysource
ant
```

Normalerweise sucht Ant selbstständig im aktuellen Verzeichnis nach der Datei `build.xml`, liest diese ein und startet anhand der darin abgelegten Vorgaben den Build-Vorgang. Bei Cocoon sollten Sie allerdings die speziell angepasste Datei `build.bat` bzw. `build.sh` für den Startvorgang verwenden, da Sie sich dadurch das Konfigurieren einiger Klassenpfade ersparen.

7.3.2 Kompilierung starten

Um Cocoon zu kompilieren, müssen Sie zuerst in das Sourcen-Verzeichnis von Cocoon `cocoon-2.x.x` wechseln, in dem sich die Datei `build.bat` bzw. `build.sh` befindet. Starten Sie nun das für Ihr Betriebssystem zutreffende Skript. Für Windows ist dies `build.bat` und für Linux `build.sh`. In diesem Fall wird Cocoon mit der Standardkonfiguration kompiliert. Dies bedeutet, dass neben allen Blöcken mit den zugehörigen Beispielen auch die umfangreiche Apidoc erstellt wird. Das Ergebnis der Kompilierung wird im Unterverzeichnis `build` abgelegt, das zuvor automatisch erzeugt wurde, falls es noch nicht existierte.

7.3.3 Kompilierung anpassen

blocks.properties

Neben einer Standardkompilierung können Sie die Kompilierung auch an eigene Bedürfnisse anpassen. Unter anderem haben Sie beispielsweise die Möglichkeit, nur bestimmte Blocks zu integrieren. Diese Einstellungen können Sie in der Datei `blocks.properties` tätigen.

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

Je nachdem, welche Blocks 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.

build.properties

Eine weitere Konfigurationsdatei, in der Sie die Kompillierung anpassen können, ist die `build.properties`. Hier haben Sie beispielsweise die Möglichkeit, das Erstellen der Apidocs oder der Beispiele zu unterbinden.

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

7.4 Installieren

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

- cocoon-2.x.x
- webapp

Im Verzeichnis `cocoon-2.x.x` befinden sich alle kompilierten Klassen und Ressourcen, die vom aktuellen Build von Cocoon benötigt werden. In `webapp` hingegen ist Cocoon in Form einer Web-Applikation erzeugt worden. Diese Web-Applikation kann in der Regel in jedem Servlet-Container betrieben werden. In manchen Fällen kann es aber sein, dass zuvor bestimmte Anpassungen gemacht werden müssen. Das hängt davon ab, welchen Servlet-Container Sie verwenden und in welche Umgebung dieser wiederum integriert ist. Spezielle Hinweise zur Installation auf den verschiedensten Umgebungen können Sie auf folgender Website erhalten:

http://cocoon.apache.org/2.1/installing/index.html

Falls Sie jedoch Tomcat als Servlet-Container verwenden, müssen Sie im Normalfall keinerlei Anpassungen für die Installation vornehmen. Kopieren Sie sich das Verzeichnis `webapp` nach `$CATALINA_HOME/webapps` und benennen Sie es 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 folgende URI aufrufen:

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

Je nachdem, auf welchem Rechner sich Ihr Servlet-Container befindet und wie er konfiguriert ist, kann der Host-Name und die Port-Nummer

von diesem Aufruf abweichen. Im Rest dieses Buches werde ich jedoch immer die angegebene Adresse verwenden, um auf die Web-Applikation Cocoon per URI zu verweisen. Bitte denken Sie daran und passen Sie diese eventuell entsprechend an, falls Sie eine andere Konfiguration besitzen. Falls Sie nach einem Aufruf der oben genannten URI eine Willkommensseite wie die nachfolgend gezeigte erhalten, ist Cocoon korrekt installiert.

Abbildung 7.1 Die Willkommensseite der Cocoon-Installation

Beispiele

Unter dem Link `samples` können Sie zahlreiche Beispiele für die Verwendung von Cocoon und die verschiedenen Blöcke finden. Am besten Sie klicken sich nun ein wenig durch diese Beispiele und probieren das eine oder andere aus. Falls Sie an den Sourcen der einzelnen Beispiele interessiert sind, können Sie diese in folgendem Verzeichnis finden:

```
$COCOON_HOME/samples
```

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

10 Cocoon erweitern

10 Cocoon erweitern

In diesem Kapitel erfahren Sie einiges über die Architektur von Cocoon und wie Sie Cocoon durch eigene Komponenten erweitern können.

Zu den herausragendsten Eigenschaften von Cocoon gehört seine interne Architektur. Diese ermöglicht es Ihnen, relativ einfach neue Komponenten zu integrieren oder bestehende auszutauschen, ohne nur eine einzige Zeile an Programmcode schreiben zu müssen.

In diesem Kapitel wollen wir uns aber nicht nur auf das Austauschen von bestehenden Komponenten beschränken, sondern selbst welche erstellen und verwenden. Um dies umsetzen zu können, werden jedoch mindestens grundlegende Kenntnisse der Programmiersprache Java sowie der Objektorientierung vorausgesetzt. Je mehr Wissen Sie in diesen beiden Bereichen bereits besitzen, umso leichter werden Sie dieses Kapitel durcharbeiten können. An dieser Stelle soll aber ausdrücklich darauf hingewiesen werden, dass dieses Kapitel nicht unbedingt erforderlich ist, um Cocoon produktiv einsetzen zu können. Es soll Ihnen lediglich aufzeigen, welche Möglichkeiten es gibt, um Cocoon noch besser an Ihre eigenen Bedürfnisse anzupassen.

Cocoon basiert auf dem Avalon-Projekt. Dieses Projekt der Apache Group hat es sich zur Aufgabe gemacht, Richtlinien und Werkzeuge zur Verfügung zu stellen, die die Entwicklung von Applikationen standardisieren, vereinfachen und modularisieren sollen. Das Projekt unterteilt sich dabei wiederum in verschiedene Subprojekte, die jeweils ein bestimmtes Aufgabengebiet abdecken, wobei in Cocoon zur Zeit lediglich die Teile **LogKit**, **Excalibur** und **Framework** verwendet werden.

Cocoon basiert auf Avalon.

Wenn in den folgenden Abschnitten also von einer Cocoon-Architektur die Rede ist, so ist immer eine auf den Vorgaben von Avalon basierende Architektur gemeint.

10.1 Das Komponenten-Modell von Avalon

Bis jetzt haben Sie sicherlich bereits viele verschiedene Komponenten kennen gelernt. Dazu zählen zum Beispiel die Pipeline-Komponenten, wie Generatoren, Transformer oder Serializer. Alle Komponenten, die in Cocoon verwendet werden, basieren auf demselben Komponenten-

Modell von Avalon. Dieses Modell beschreibt, wie eine Komponente erzeugt, konfiguriert und verwaltet werden muss.

Der Vorteil, gekapselte Komponenten anstatt verstreuter, einzelner Logik-Klassen zu verwenden, liegt auf der Hand. Zum einen wird jede Komponente auf dieselbe Art erzeugt und konfiguriert. Somit ist die Applikation wesentlich einfacher wartbar, da jeder Entwickler denselben Richtlinien folgen muss. Daneben werden für alle Komponente die Möglichkeiten des Poolings und Loggings und anderer Erweiterungen in gleicher Weise zur Verfügung gestellt, ohne für jeden Bereich eine spezielle Lösung erstellen zu müssen. Der wichtigste und komfortabelste Punkt ist jedoch, dass eine Komponente beliebig austauschbar ist, ohne Änderungen an der Applikation selbst vornehmen zu müssen.

Avalon realisiert diese Eigenschaften vor allem unter Verwendung zweier Design-Patterns, die in der Welt der objektorientierten Programmierung äußerst häufig anzutreffen sind:

IoC und SoC

- Inversion of Control (IoC)
- Separation of Concerns (SoC)

Inversion of Control (deutsch: Umkehrung der Kontrolle) meint dabei, dass eine Komponente von einer zentralen Instanz erzeugt, konfiguriert und verwaltet werden muss. Ein häufig anzutreffendes Beispiel hierfür ist der Einsatz einer so genannten Factory. Daneben gibt Separation of Concerns (deutsch: Trennung der Zuständigkeiten) an, dass für eine einzelne Komponente genau definiert werden muss, welche Aufgabe sie hat. Ein einfaches Beispiel hierfür ist die Verwendung von Interfaces in Java. In den nachfolgenden Abschnitten werden jeweils die Abkürzungen für diese beiden Design-Patterns verwendet, wenn sie benötigt werden.

10.1.1 Komponente erstellen

Wie bereits angedeutet, muss eine Komponente bestimmten Vorgaben folgen, damit sie innerhalb von Avalon verwendet werden zu kann. Diese Vorgaben werden speziell durch das Avalon-Subprojekt »Framework« definiert. Dabei handelt es sich um eine Sammlung von verschiedenen Konzepten und Interfaces.

Im Detail gesprochen muss eine Komponente mindestens nachfolgende Bedingungen erfüllen, um innerhalb von Cocoon bzw. Avalon verwendet werden zu können:

- ein eigenes Interface implementieren, das die Funktionalitäten der Komponente beschreibt und der Komponente somit eine so genannte Rolle zuweist
- mindestenes eine funktionsfähige Implementierung des Interfaces
- Registrierung und Konfiguration der Rollen in einer entsprechenden Datei

zuweisen einer Rolle

Wie in der vorangegangenen Liste beschrieben, muss jede Komponente zunächst einer bestimmten Rolle zugewiesen werden. Diese Rolle beschreibt, welche Eigenschaften eine Komponente besitzt. Stellen Sie sich analog hierzu als Beispiel das Theaterstück Romeo und Julia vor. Die Theaterrolle Romeo kann jeder begabte Schauspieler spielen. Durch das Drehbuch wird vorgeschrieben, welche Aufgaben der Schauspieler zu erfüllen hat. Genauso ist es mit Rollen in Cocoon. Jede Komponente besitzt eine bestimmte Rolle im Theaterstück »Cocoon« und kann beliebig durch einen Schauspieler »Implementierung« ausgetauscht werden. Dabei handelt es sich um die direkte Umsetzung des Design-Patterns SoC, da jede Komponente nur für einen bestimmten, fest vorgegebenen Bereich zuständig ist.

Rollenname

Die konkrete Beschreibung einer Rolle wird durch ein Interface realisiert. In diesem Interface wird neben der eigentlichen Definition ein eindeutiger Name für die Rolle vergeben, der meistens der Klassenpfad zum jeweiligen Interface ist. Sie können für die Rolle natürlich auch einen anderen Namen verwenden. Achten Sie jedoch unbedingt darauf, dass dieser in der gesamten Cocoon-Umgebung eindeutig sein muss.

Um die Erstellung einer Komponente zu verdeutlichen, wird im Nachfolgenden parallel ein Beispiel realisiert. Dabei handelt es sich um eine einfache Komponente, die lediglich zwei Zahlen entgegennehmen und addieren kann. Im folgenden Listing sehen Sie, wie ein Interface für eine solche Komponente aussehen könnte.

Interface

```
package foo.bar;
import org.apache.avalon.framework.component.Component;

public interface MyComponent extends Component {
   String ROLE = "foo.bar.MyComponent";

   public void setValues(int a, int b);
   public int add();
}
```

Listing 10.1 Ein einfaches Interface für die Beispielkomponente

In den aktuellen Versionen von Cocoon ist das Package `component` als »deprecated« eingestuft und sollte – falls möglich – nicht mehr verwendet werden. Stattdessen sind fast alle Definitionen nun im Package `service` aufzufinden. Eine Ausnahme stellt hier das Interface `Component` dar. Da leider noch nicht alle Komponenten von Cocoon und auch Avalon auf das Service-Package umgestellt wurden, sollten Sie in Ihren Komponenten immer das Interface `Component` ableiten, um eine Abwärtskompatibilität zu gewährleisten und damit möglichen Problemen von vorneherein aus dem Weg zu gehen. Eine Aufwärtskompatibilität ist natürlich in beiden Fällen gegeben.

Neben dem Interface wird zusätzlich noch mindestens eine konkrete Implementierung benötigt, die die definierten Eigenschaften für diese Rolle realisiert. Im nachfolgenden Listing ist eine mögliche Implementierung angegeben.

konkrete Implementierung

```
package foo.bar;

public class MyComponentImpl implements MyComponent {

   private int a;
   private int b;

   public void setValues(int a, int b) {

      this.a = a;
      this.b = b;
   }

   public int add() {

      return this.a + this.b;
   }
}
```

Listing 10.2 Konkrete Implementierung des Interfaces

Bis zu diesem Punkt deutet nur wenig darauf hin, dass es sich bei der Erstellung einer Komponente um eine andere Vorgehensweise handelt, als sie auch aus der üblichen Java-Programmierung hinreichend bekannt sein dürfte. Die nächsten Schritte werden jedoch davon abweichen.

Rolle registrieren

Nachdem ein Interface mit einer konkreten Implementierung für eine Komponente erstellt wurde, muss diese Komponente unter ihrem Rollen-Namen zunächst registriert werden. Die einfachste Möglichkeit, eine Komponente in Cocoon zu registrieren, erfolgt über die Datei `cocoon.xconf`, die sich standardmäßig im Verzeichnis `$COCOON_HOME/WEB-INF` befindet. Das folgende Listing zeigt, welchen Eintrag Sie in dieser Datei machen müssen, um eine Komponente zu registrieren.

```
<?xml version="1.0" encoding="UTF-8"?>
<cocoon version="2.1">
   ...
   <component role="foo.bar.MyComponent"
      class="foo.bar.MyComponentImpl"/>
   ...
</cocoon>
```

Listing 10.3 Eine Komponente in der Datei cocoon.xconf registrieren

Durch das Element `<component/>` wird eine neue Komponente registriert. In der nachfolgenden Tabelle erhalten Sie eine Beschreibung der jeweiligen Attribute.

Attribut	Beschreibung	Pflicht
`role`	Der eindeutige Name der Rolle. Das ist in der Regel der Klassenpfad zum entsprechenden Interface für diese Rolle.	Ja
`class`	Der vollständige Klassenpfad der Klasse, die eine konkrete Implementierung des Rollen-Interfaces ist.	Ja
`logger`	Hiermit kann eine Log-Kategorie aus der Datei logikt.xconf angegeben werden, in die alle Log-Meldungen dieser Komponente geloggt werden sollen.	Nein

Tabelle 10.1 Die Attribute von <component/>

externe Rollendatei

Alternativ können Sie die Registrierung der Rolle auch in eine externe Datei auslagern. Diese Datei kann einen beliebigen Namen besitzen und sich an einem beliebigen Ort innerhalb von `$COCOON_HOME` befinden. Der Pfad zu dieser Datei wird durch das optionale Attribut `user-roles` im Wurzelelement `<cocoon/>` der Datei `cocoon.xconf` angegeben. Dieser Pfad ist relativ zu `$COCOON_HOME` und muss immer mit einem Slash / beginnen, wie im nachfolgenden Listing zu sehen ist:

```
<?xml version="1.0" encoding="UTF-8"?>
<cocoon version="2.1"
   user-roles="/WEB-INF/roles.xconf">
...
```

Listing 10.4 Konfigurieren der Rollendatei

Um in dieser Datei Komponenten zu registrieren, ist eine etwas andere Struktur notwendig, als dies in `cocoon.xconf` der Fall ist. Sehen Sie sich hierzu das folgende Listing an:

```
<?xml version="1.0" encoding="UTF-8"?>
<role-list>
   <role name="foo.bar.MyComponent "
      shorthand="my-component"
      default-class="foo.bar.MyComponentImpl"/>
</role-list>
```

Listing 10.5 Struktur der Rollendatei

Als Wurzelelement besitzt die Rollendatei das Element `<role-list/>`. Darin werden alle Komponenten jeweils durch ein Element `<role/>` registriert. In der nachfolgenden Tabelle erhalten Sie eine Beschreibung der jeweiligen Attribute von `<role/>`.

Attribut	Beschreibung	Pflicht
`name`	Der eindeutige Name der Rolle. Das ist in der Regel der Klassenpfad zum entsprechenden Interface für diese Rolle.	Ja
`shorthand`	Eine Kurzbezeichnung für `name`. Diese Bezeichnung kann in anderen Bereichen verwendet werden, um diese Komponente z. B. zu konfigurieren, wie Sie später noch sehen werden.	Ja
`default-class`	Der vollständige Klassenpfad der Klasse, die eine konkrete Implementierung des Rollen-Interfaces ist. Diese Implementierung wird standardmäßig für diese Rolle verwendet, wenn keine andere Implementierung in der Datei `cocoon.xconf` zugewiesen wurde.	Nein

Tabelle 10.2 Attribute von <role/>

Die in der Rollendatei angegebenen Komponenten werden in der Regel nur durch den Entwickler selbst konfiguriert, der eine neue Komponente zur Verfügung stellen möchte. Konfigurationen, die über das Registrieren einer Komponente hinausgehen, können in dieser Datei

nicht getätigt werden, sondern müssen wieder in der Datei `cocoon.xconf` definiert werden. Dabei wird eine Komponente immer durch ihren `shorthand` referenziert, wie Sie noch sehen werden. Somit ist gewährleistet, dass sich die Konfigurationsdaten immer an einem einheitlichen Ort befinden.

Konfiguration

Nachdem Sie in der Datei `roles.xconf` die Rolle `foo.bar.MyComponent` – wie in Listing 10.5 gezeigt – registriert haben, können Sie diese innerhalb von `cocoon.xconf` konfigurieren. Eine solche Konfiguration ist optional. Hier kann beispielsweise angegeben werden, welche Implementierung Sie für die Komponente `mycomponent` verwenden möchten. Wenn Sie hier keine Angaben zur gewünschten Implementierung machen, wird die Default-Implementierung verwendet, die zuvor in der Rollendatei durch das Attribut `default-class` angegeben wurde. Zusätzlich werden in der Datei `cocoon.xconf` in der Regel alle Parameter zur Konfiguration einer Komponente angegeben. Welche Möglichkeiten es gibt, im Speziellen Komponenten zu konfigurieren, sehen wir uns etwas später in diesem Kapitel an.

Shorthand wird zu Elementname.

Zunächst ist es wichtig zu wissen, dass Sie eine Rolle innerhalb von `cocoon.xconf` konfigurieren können, indem Sie sie über ein Element referenzieren, das den Namen des Attributs `shorthand` aus der Rollendatei erhält. In unserem Beispiel ist dies also das Element `<my-component/>`. Für dieses Beispiel weisen wir der Rolle `mycomponent` die konkrete Implementierung `MyComponentImpl` zu, die Sie bereits als Default-Implementierung registriert haben. Der entsprechende Eintrag in `cocoon.xconf` könnte demnach aussehen wie im folgenden Listing dargestellt.

```
<?xml version="1.0" encoding="UTF-8"?>
<cocoon version="2.1"
   user-roles="/WEB-INF/roles.xconf">
   ...
   <my-component class="foo.bar.MyComponentImpl"/>
   ...
</cocoon>
```

Listing 10.6 Zuweisen einer konkreten Implementierung

Neben dem Attribut `class` können Sie hier auch alle Attribute verwenden, die für das Element `<component/>` erlaubt sind, wie in Tabelle 10.2 beschrieben.

10.1.2 Der Service-Manager

Mit der Zuweisung der konkreten Implementierung in der Datei `cocoon.xconf` bzw. der Rollendatei haben Sie eine vollständige Komponente realisiert, die Sie anschließend innerhalb von Cocoon verwenden können. Um jedoch auf eine solche Komponente zugreifen und mit ihr arbeiten zu können, müssen Sie den so genannten **Service-Manager** verwenden.

Eine Komponente wird niemals direkt instanziiert, sondern immer über den Service-Manager erzeugt.

Der Service-Manager ist die zentrale Instanz, die alle verfügbaren Komponenten verwaltet. Er kann in etwa mit einer umfangreichen Factory verglichen werden, wie sie aus der Java-Programmierung bekannt sein dürfte. Der Manager implementiert dabei das gleichnamige Interface `ServiceManager`, das im folgenden Listing gezeigt wird.

```
package org.apache.avalon.framework.service;

public interface ServiceManager {
   Object lookup(String key)
      throws ServiceException;

   boolean hasService(String key);

   void release(Object object);
}
```

Listing 10.7 Das Interface ServiceManager

Bis zur Version 2.0.4 von Cocoon wurde der Component-Manager für dieselben Aufgaben verwendet, die nun der Service-Manager erfüllt. Der einzige Unterschied dieser beiden Manager besteht darin, dass der Component-Manager ausschließlich Objekte vom Typ `Component` verwalten konnte, wogegen der Service-Manager nun generell Objekte vom untersten Basistyp `Object` zurückliefert und erwartet.

Komponente anfordern

Beim Start von Cocoon werden die verschiedenen Konfigurationsdateien gelesen. Durch den Aufruf der Methode `lookup` am jeweiligen Service-Manager kann anschließend eine bestimmte Komponente erhalten werden. Dabei wird dieser Methode als Parameter der Name der Rolle übergeben, deren Implementierung gewünscht ist. In unserem Beispiel könnte die Methode mit `lookup("foo.bar.MyComponent")` aufgerufen werden. Zurückgeliefert wird in diesem Fall anschließend eine Instanz von `MyComponent`.

Die Methode `hasService` hat lediglich die Aufgabe zu überprüfen, ob eine Rolle mit dem übergebenen Namen existiert, und durch `release` wird eine zuvor erhaltene und nicht mehr benötigte Komponente wieder zurück an den Service-Manager übergeben.

Eine nicht mehr benötigte Komponente sollte am Ende der Verwendung immer durch die Methode `release` an den Service-Manager zurückgegeben werden. Dabei ist es egal, ob Zusatzfunktionalitäten, wie beispielsweise Pooling, verwendet werden oder nicht. Somit ist gewährleistet, dass eine Komponente auch nachträglich diese Funktionalitäten erhalten kann, ohne alle Teile der Applikation, die diese Komponente verwenden, umständlich erweitern zu müssen.

In Listing 10.8 können Sie ein Beispiel sehen, wie der Service-Manager verwendet wird, um die Komponente `MyComponent` zu erhalten. Besonders hervorzuheben ist hierbei, dass die `finally`-Klausel eingesetzt wird, um die Komponente in jedem Fall an den Service-Manager zu übergeben. Dabei ist es egal, ob ein Fehler aufgetreten ist oder nicht. Bitte verwenden auch Sie in Ihren zukünftigen Programmen – falls möglich – diese Form, um eine Komponente immer korrekt zurück an den Service-Manager zu übergeben.


```
...
import foo.bar.MyComponent;
import org.apache.avalon.framework.service.↵
       ServiceManager;
import org.apache.avalon.framework.service.↵
       ServiceException;
...
public void execute() {

   MyComponent = null;

   try {

      component =
         this.manager.lookup(MyComponent.ROLE);

      component.setValues(3,4);
      System.out.println(component.add());

   } catch (ServiceException ex) {
      // Fehlerbehandlung ...
```

```
    } finally {
       this.manager.release(component);
    }
}
...
```

Listing 10.8 Verwenden des Service-Managers

Zugriff auf den Service-Manager

Wahrscheinlich stellen Sie sich an diesem Punkt die Frage, auf welche Weise denn letztendlich das Manager-Objekt erzeugt oder übergeben wird, um es in einer Klasse verwenden zu können. Diese Frage kann jedoch nur im Zusammenhang mit dem so genannten **Component-Lifecycle** von Avalon ausreichend beantwortet werden, der erst später in Abschnitt 10.2 »Komponenten erweitern« besprochen wird. Aus diesem Grund stellen wir die Lösung noch ein Weilchen hinten an.

Es soll jedoch bereits jetzt erwähnt werden, dass ein solcher Zugriff einfach durch das Implementieren des Interfaces `Serviceable` erfolgen kann. Durch die Methode `service`, die daraufhin erzeugt werden muss, wird Ihnen eine Instanz des Service-Managers übergeben.

10.1.3 Der Service-Selector

Bis jetzt wurde in den vorherigen Abschnitten immer davon ausgegangen, dass eine Rolle genau eine einzige, konkrete Implementierung besitzt, also für ein Interface genau eine Klasse existiert, die es implementiert. In der Realität ist es allerdings so, dass ein Interface häufig von vielen verschiedenen Klassen implementiert wird. Alle diese Klassen besitzen somit die Eigenschaften dieses Interfaces und stellen zusätzlich spezielle Funktionalitäten zur Verfügung. In Avalon bedeutet dies, dass einer Rolle beliebig viele Implementierungen zugewiesen und diese anschließend verwendet werden können. Das wird durch einen **Service-Selector** realisiert. Dabei kann bei der Erstellung der Komponenten exakt so vorgegangen werden wie bisher beschrieben. Lediglich die Konfiguration in der Rollendatei bzw. in `cocoon.xconf` sowie die Art des Objekts, das durch den Manager bei einer Anfrage zurückgeliefert wird, ändert sich.

Der Service-Manager liefert im Fall von mehreren Implementierungen für eine Rolle nicht etwa ein Objekt vom Typ dieser Rolle, wie es bisher üblich war, sondern ein Objekt vom Typ `ServiceSelector`, das alle zugehörigen Implementierungen der Rolle enthält. Diese können anschließend durch die Methode `select` angefordert werden. In Listing 10.9 ist das entsprechende Interface hierzu angegeben.

```
package org.apache.avalon.framework.service;

public interface ServiceSelector {

   Object select(Object policy)
      throws ServiceException;

   boolean isSelectable(Object policy);

   void release(Object object);
}
```

Listing 10.9 Das Interface Service-Selector

ExcaliburComponent-Selector

Bevor wir uns jedoch ansehen, wie der Service-Selector innerhalb einer Klasse verwendet wird, werfen wir einen Blick auf die Rollendatei. Hier müssen Sie als Erstes den entsprechenden Service-Selector registrieren, den Sie verwenden möchten. Dies geschieht durch das Attribut `default-class` innerhalb des Elements `<role/>`. Falls Sie keinen eigenen Selector besitzen oder erstellen möchten, können Sie auf den `ExcaliburComponentSelector`[1] von Avalon zurückgreifen.

Neben dem Registrieren des zu verwendenden Selectors ist es zusätzlich erforderlich, dass Sie einen Namen für diesen Selector angeben. Das geschieht – genauso wie in den einfachen Fällen – durch das Attribut `name`. Dabei verwendet man auch hier in der Regel den voll qualifizierten Namen des Interfaces, das diejenige Rolle beschreibt, die letztendlich zurückgeliefert werden soll. Zusätzlich wird allerdings noch der String »Selector« zum Schluss an den Namen angehängt, um deutlich zu machen, dass es sich hierbei um einen Selector und keine einzelne Komponente handelt.

Zum besseren Verständnis erstellen wir einen Eintrag in der Rollendatei `roles.xconf` für ein konkretes Beispiel. Dabei wird vorausgesetzt, dass ein Interface `foo.bar.AirPlaine` existiert, das die wichtigsten Eigenschaften eines jeden Flugzeugs beschreibt. Die konkreten

1 Dieser Selector implementiert zwar das »alte« Interface ComponentSelector, ist aber gleichzeitig zu dem Zeitpunkt, an dem diese Zeilen geschrieben wurden, auch die stabilste Umsetzung. Da der Service-Manager einen Selector, der das Interface ComponentSelector implementiert, automatisch in einen Selector vom Typ ServiceSelector konvertiert, spielt es nahezu überhaupt keine Rolle, ob der verwendete Selector das eine oder andere Interface implementiert. Es wird immer ein Selector vom Typ ServiceSelector zurückgeliefert. Außerdem ist durch die deklarative Zuweisung innerhalb der Rollendatei ein Austauschen jederzeit und ohne große Mühe möglich.

Implementierungen könnten somit die beiden Klassen `JumboJet` und `EuroFighter` sein. Diese beiden Klassen gehören also zur selben Rolle `foo.bar.AirPlaine`. In einem solchen Fall ist es notwendig, einen Selector zu verwenden, um innerhalb der Applikation die gewünschte Instanz zu erhalten. Der Eintrag in der Datei `roles.xconf` würde für dieses Beispiel folgendermaßen lauten:

```
...
<role
   default-class="org.apache.avalon.excalibur.↵
      component.ExcaliburComponentSelector"
   name="foo.bar.AirPlaineSelector"
   shorthand="airplaine"/>
...
```

Listing 10.10 Registrieren eines Selectors in der Rollendatei

cocoon.xconf

Nachdem in `roles.xconf` angegeben wurde, welche Rolle durch welchen Selector verwaltet und welche Kurzbezeichnung verwendet werden soll, müssen als Nächstes in der Datei `cocoon.xconf` die konkreten Implementierungen registriert werden. Dies geschieht durch das Element `<component-instance/>`, das innerhalb des Elements `<airplaine/>` platziert wird. Der Name des Elements `<airplaine/>` wird auch hier durch das Attribut `shorthand` aus der Rollendatei festgelegt.

```
...
<airplaine>
   <component-instance class="foo.bar.JumboJet"
      name="jumbojet"/>
   <component-instance class="foo.bar.EuroFighter"
      name="eurofighter"/>
</airplaine>
...
```

Listing 10.11 Zuweisen der konkreten Implementierungen

Innerhalb des Shorthand-Elements können beliebig viele konkrete Implementierungen durch das Element `<component-instance/>` registriert werden, wobei der Wert des Attributs `name` darin eindeutig sein muss. Durch das Attribut `class` wird der vollständige Klassenname der Implementierung angegeben.

Durch diesen Eintrag in der Datei `cocoon.xconf` wurden die konkreten Klassen `JumboJet` und `EuroFighter` dem Shorthand `airplaine`

und somit der Rolle `foo.bar.AirPlaineSelector` zugewiesen. Doch wie kommt man innerhalb der Applikation nun an die gewünschte Implementierung? Um es kurz zu machen: Sehen Sie sich hierzu einfach das nächste Listing an. Es zeigt den Inhalt einer Methode, die einen solchen Zugriff durchführt.

```
...
import foo.bar.AirPlaine;
import org.apache.avalon.framework.service.↵
       ServiceSelector;
import org.apache.avalon.framework.service.↵
       ServiceException;
...
public void execute() {

   ServiceSelector selector = null;
   AirPlaine jumboJet = null;

   try {

      // Den Selector erhalten
      selector =
         (ServiceSelector)this.manager.lookup(
            AirPlaine.ROLE + "Selector");

      // Die Komponente ueber den Selector erhalten
      jumboJet =
         (AirPlaine)selector.select("jumbojet");

      // Mach irgendwas ...

   } catch (ServiceException ex) {
      // Fehlerbehandlung ...
   } finally {

      selector.release(jumboJet);
      this.manager.release(selector);
   }
}
...
```

Listing 10.12 Zugriff auf eine Komponente über einen Service-Selector

Zunächst wird über den Service-Manager der Service-Selector angefordert, der unter dem Namen `foo.bar.AirPlaine` mit dem Zusatz »Selector« in der Datei `roles.xconf` registriert ist. Wie Sie bereits wissen, liefert der Service-Manager in einem solchen Fall immer eine Komponente vom Typ `ServiceSelector` zurück. Aus diesem Selector wird anschließend durch die Methode `select` genau diejenige Instanz der Komponente angefordert, die den Namen `jumbojet` in der Datei `cocoon.xconf` zugewiesen bekommen hat. Nach der Verwendung des Managers und des Selectors wird zunächst die durch den Selector erzeugte Komponente durch die Methode `release` wieder freigegeben und anschließend der Selector selbst.

10.1.4 Flowscript und der Service-Manager

Neben dem Fall, innerhalb einer Komponente auf eine weitere Komponente zuzugreifen, tritt in Cocoon häufig noch der weitere Fall auf, dass auf eine Komponente innerhalb eines Flowscripts zugegriffen werden muss. Dies geschieht dabei über das implizite Objekt `cocoon`, über das auch alle Objekte des FOMs erhalten werden können. Mehr zu diesen Objekten und zu Flowscript selbst können Sie in Kapitel 11, *Control Flow*, und in Anhang I, *FOM* nachlesen. Listing 10.13 zeigt, wie Sie eine Komponente mit dem Rollennamen `foo.bar.MyComponent` aus dem vorhergehenden Beispiel innerhalb eines Flowscripts erhalten können.

```
function main() {

   component =
      cocoon.getComponent("foo.bar.MyComponent");

   ...

   cocoon.releaseComponent(component);
}
```

Listing 10.13 Komponente innerhalb eines Flowscripts erhalten und zurückgeben

Wie in Listing 10.15 zu sehen ist, kann durch die Methode `getComponent` eine Komponente erhalten und mit `releaseComponent` wieder zurückgegeben werden.

Achten Sie darauf, dass Sie die Methode `releaseComponent` immer vor dem Aufruf von `sendPage` oder `sendPageAndWait` aufrufen. Damit wird die Komponente wieder zurückgegeben, bevor das Continuation-Objekt abgeschlossen wird und die Komponente dadurch

eventuell »verloren« gehen könnte, z.B. wenn ein Continuation-Objekt durch den Continuation-Manager zerstört wird. Zusätzlich bietet es sich an, die Arbeit mit einer Komponente innerhalb eines Flowscripts in einem try-finally-Block einzubetten und somit die Komponente auch im Fehlerfall korrekt zurückzugeben.

10.2 Komponenten erweitern

Bis jetzt wurden in diesem Kapitel nur Komponenten erstellt, die keine Eigenschaften über ihre Basisfunktionalitäten hinaus vorweisen können. Die Komponente `MyComponentImpl` aus einem der vorhergehenden Beispiele stellt genau die Funktionalitäten zur Verfügung, die durch das eigene Interface `MyComponent` definiert wurden.

Das Avalon-Framework stellt jedoch zusätzliche Funktionalitäten für eine Komponente zur Verfügung. Zu den wichtigsten zählen:

- Pooling
- Logging
- Konfiguration

Je nachdem, welche dieser Funktionalitäten eine Komponente erhalten soll, muss sie ein entsprechendes Interface implementieren.

Component-Lifecycle

Vor dem Ausliefern und nach dem Zurücklegen einer Komponente durch den Service-Manager durchläuft eine Komponente immer einen bestimmten Zyklus. Dieser Zyklus wird **Component-Lifecycle** genannt. Die zugehörigen Interfaces heißen entsprechend **Lifecycle-Interfaces**. Innerhalb eines solchen Zyklus wird die Komponente dahingehend überprüft, welches Interface aus dem Framework sie implementiert. Damit verbunden ist eine entsprechende Aktion an der Komponente. Diese Aktion ist in der Regel der Aufruf einer – durch das Interface definierten – Methode, in der bestimmte Aufgaben erfüllt werden können. Diese Aufgaben werden in den meisten Fällen durch den Entwickler der Komponente bestimmt.

Ein mögliches Beispiel ist die Konfiguration einer Komponente durch das Interface `Configurable`, das wir später noch kennen lernen. Bevor eine Komponente ausgeliefert wird, ruft der Service-Manager die Methode `configure` an der Komponente auf und übergibt ihr ein Objekt vom Typ `Configuration`. Dieses Objekt enthält alle Konfigurationsparameter, die vorher speziell gesetzt werden. Innerhalb der Methode `configure` besteht nun die Möglichkeit, diese Konfigura-

tionswerte zu validieren und entsprechend aufzubereiten, bevor die Komponente an die Applikation ausgegeben wird.

Die Überprüfung, ob eine Komponente ein bestimmtes Lifecycle-Interface implementiert, erfolgt immer in derselben vorgegebenen Reihenfolge. In Abbildung 10.1 wird diese Reihenfolge veranschaulicht. Berücksichtigt werden dabei nur Komponenten, die nach ihrer Verwendung sofort wieder zerstört werden.

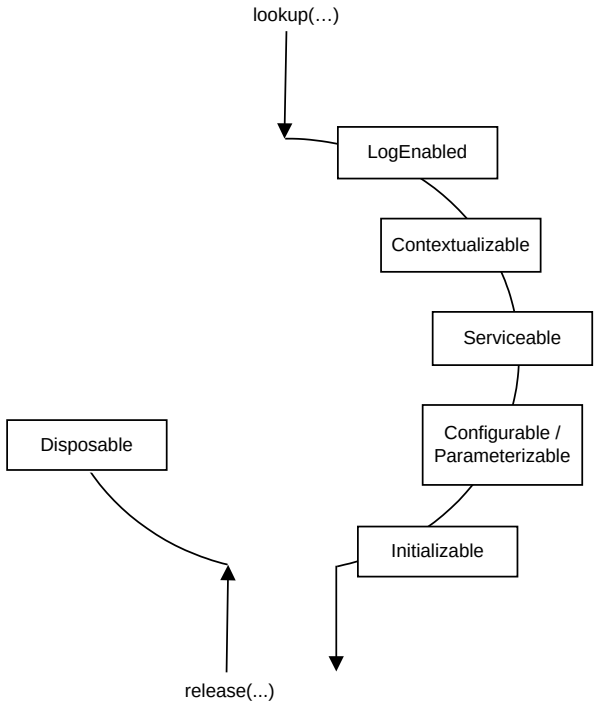


Abbildung 10.1 Die wichtigsten Lifecycle-Interfaces einer Avalon-Komponente

Neben den hier angegebenen Lifecycle-Interfaces existieren noch einige mehr, die in Cocoon jedoch in der Regel keine Verwendung finden. Falls es Sie interessiert, können Sie diese auf den Websites von Avalon nachschlagen:

http://avalon.apache.org/developing/framework.html

10.2.1 Das Interface Contextualizable

Falls eine Komponente das Lifecycle-Interface `Contextualizable` implementiert, ruft der Service-Manager beim Erstellen einer Instanz einer Komponente als Erstes die Methode `contextualize` auf. Das

Context-Objekt, das hierbei übergeben wird, enthält Informationen über die aktuelle Applikation als Name-Wert-Paare. Zwischen diesem Objekt und dem Context-Objekt, das z.B. aus der Servlet-Programmierung bekannt ist, besteht kein direkter Zusammenhang.

```
package org.apache.avalon.framework.context;

public interface Contextualizable {

   void contextualize(Context context)
      throws ContextException;
}
```

Listing 10.14 Das Interface Contextualizable

10.2.2 Zugriff auf andere Komponenten

In den vorangegangenen Beispielen wurde immer davon ausgegangen, dass eine Instanz des Service-Managers bereits innerhalb der Klasse, die Zugriff darauf benötigt, zur Verfügung steht. Dies ist natürlich nicht selbstverständlich. Komponenten, die Zugriff auf andere Komponenten und somit auf den Service-Manager benötigen, müssen zusätzlich das Lifecycle-Interface `Serviceable` implementieren.

Serivceable


```
package org.apache.avalon.framework.service;

public interface Serviceable {

   void service(ServiceManager manager)
      throws ServiceException;
}
```

Listing 10.15 Das Interface Serviceable

Dieses Interface definiert eine einfache Methode `service`, über die eine Instanz des aktuellen Service-Managers an die Komponente übergeben wird. Diese Instanz kann nun in eine Klassen-Variable abgelegt und anschließend innerhalb einer Komponente aufgerufen werden. Das folgende Listing zeigt die Implementierung des Interfaces `Serviceable` und den anschließenden Zugriff auf den Manager anhand eines Beispiels.

```
import org.apache.avalon.framework.service.↵
       ServiceManager;
import org.apache.avalon.framework.service.↵
       ServiceException;
```

```
import org.apache.avalon.framework.service.↵
       Serviceable;

public class Example implements Serviceable {

   private ServiceManager manager = null;

   public void execute() {

      MyComponent component = null;

      try {
         component =
            this.manager.lookup(MyComponent.ROLE);
         System.out.println(component.add(3,4));
      } catch (ServiceException ex) {
         // Fehlerbehandlung ...
      } finally {
         this.manager.release(component);
      }
   }

   public void service(ServiceManager manager)
      throws ServiceException {

      this.manager = manager;
   }
}
```

Listing 10.16 Eine Klasse, die das Interface Serviceable implementiert

In den Cocoon-Versionen vor 2.1 musste statt dem Interface `Serviceable` das Interface `Composable` verwendet werden. Dies erfüllte jedoch exakt dieselbe Aufgabe durch die Methode `compose`. Da die aktuelle Version von Cocoon mit dem Service-Manager arbeitet, müssen Sie das Interface `Composable` nicht weiter beachten und können stattdessen immer `Serviceable` verwenden.

10.2.3 Konfiguration mit verschachtelten Elementen

Die Konfiguration einer Komponente ist ein äußerst wichtiger Aspekt, der nicht unterschätzt werden sollte. Komponenten in Cocoon können Sie auf zweierlei Arten konfigurieren: zum einen über eindimensionale Parameter-Elemente, wie z.B.

```
<parameter name="myParam" value="myValue"/>
```

Listing 10.17 Konfigurationsparameter

oder über verschachtelte Elemente, wie z. B.:

```
<myParams>
   <foo>valueA</foo>
   <bar>valueB</bar>
</myParams>
```

Listing 10.18 Verschachtelte Konfigurationselemente

In diesem Abschnitt wollen wir zunächst den zweiten Fall genauer betrachten. Durch eine Verschachtelung von Elementen lassen sich die Konfigurationselemente gut strukturieren. Ein kleiner Nachteil dieser Art der Konfiguration ist jedoch das relativ aufwändige Auslesen von Konfigurationswerten, was jedoch mit ein wenig Übung keine größeren Probleme bereiten dürfte.

Configurable

Jede Komponente, die über eine solche Konfigurationsmöglichkeit verfügen möchte, muss das Lifecycle-Interface `Configurable` implementieren, das im Folgenden abgebildet ist.

```
package org.apache.avalon.framework.configuration;

public interface Configurable {

   void configure(Configuration configuration)
      throws ConfigurationException;
}
```

Listing 10.19 Das Interface Configurable

Konfiguration

Die Konfigurationselemente selbst werden in der Datei `cocoon.xconf` innerhalb der jeweiligen Komponente platziert. Listing 10.20 zeigt ein Beispiel, wie die Komponente `MyComponent` konfiguriert werden könnte.

```
<?xml version="1.0" encoding="UTF-8"?>
<cocoon version="2.1">
   ...
   <my-component
      class="foo.bar.MyConfigurableComponent">
      <default-values>
         <a>100</a>
         <b>300</b>
      </default-values>
```

```
    </my-component>
    ...
</cocoon>
```

Listing 10.20 Konfiguration einer Komponente über Elemente

Falls Sie dieses Beispiel selbst ausprobieren möchten, achten Sie darauf, dass Sie den Wert des Attributs `class` zu `MyConfigurableComponent` ändern müssen.

Innerhalb der Komponente kann auf die Konfigurationswerte zugegriffen werden, indem das Configuration-Objekt in der Methode `configuration` entgegengenommen und anschließend entsprechend durchlaufen wird. Im nachfolgenden Listing sehen Sie, wie die in Listing 10.20 angegebene Konfiguration ausgelesen werden kann.

```
package foo.bar;
import org.apache.avalon.framework.configuration.↵
Configurable;
import org.apache.avalon.framework.configuration.↵
ConfigurationException;
import org.apache.avalon.framework.configuration.↵
Configuration;

public class MyConfigurableComponent implements
   MyComponent, Configurable {

   private int a;
   private int b;

   public void setValues(int a, int b) {

      this.a = a;
      this.b = b;
   }

   public int add() {

      return this.a + this.b;
   }

   public void configure(Configuration config)
      throws ConfigurationException {
```

```
        Configuration def =
           config.getChild("default-values");

        Configuration a = def.getChild("a");
        Configuration b = def.getChild("b");

        this.a = a.getValueAsInteger();
        this.b = b.getValueAsInteger();
    }
}
```

Listing 10.21 Verarbeitung von Konfigurationselementen

10.2.4 Konfiguration mit Parametern

Neben der Konfiguration mit verschachtelten Elementen existiert noch ein einfachere Form der Konfigurierung: die Konfiguration mit Parametern.

Parameterizable

Um eine Komponente mit einfachen Parametern der Form `<parameter/>` konfigurieren zu können, ist die Implementierung des Lifecycle-Interfaces `Parameterizable` notwendig. In Listing 10.22 sehen Sie den Aufbau dieses Interfaces.

```
package org.apache.avalon.framework.parameters;

public interface Parameterizable {

   void parameterize(Parameters parameters)
      throws ParameterException;
}
```

Listing 10.22 Das Interface Parameterizable

Konfiguration

Innerhalb der Datei `cocoon.xconf` können durch das Element `<parameter/>` an die Komponente beliebig viele Konfigurationswerte übergeben werden. Eine solche Konfiguration könnte z.B. aussehen wie in Listing 10.23 gezeigt. Dabei wird auf das vorhergehende Beispiel `MyComponent` aufgebaut, und für die beiden Zahlen `a` und `b` werden Default-Werte angegeben.

```
<?xml version="1.0" encoding="UTF-8"?>
<cocoon version="2.1">
   ...
   <my-component
      class="foo.bar.MyParameterizableComponent">
```

```
      <parameter name="a" value="100"/>
      <parameter name="b" value="300"/>
   </my-component>
   ...
</cocoon>
```

Listing 10.23 Konfiguration einer Komponente über Parameter

Falls Sie dieses Beispiel selbst ausprobieren möchten, achten Sie darauf, dass Sie den Wert des Attributs `class` zu `MyParameterizable-Component` ändern müssen.

Implementierung

Nachdem die jeweilige Komponente zusätzlich das Interface `Parameterizable` implementiert hat, kann auf die konfigurierten Parameter zugegriffen werden, wie in Listing 10.34 gezeigt.

```
package foo.bar;

import org.apache.avalon.framework.↵
parameters.Parameterizable;
import org.apache.avalon.framework.↵
parameters.ParameterException;
import org.apache.avalon.framework.↵
parameters.Parameters;

public class MyParameterizableComponent implements
   MyComponent, Parameterizable {

   private int a;
   private int b;

   public void setValues(int a, int b) {

      this.a = a;
      this.b = b;
   }

   public int add() {

      return this.a + this.b;
   }

   public void parameterize(Parameters parameters)
      throws ParameterException {
```

```
        this.a = parameters.getParameterAsInteger("a");
        this.b = parameters.getParameterAsInteger("b");
    }
}
```

Listing 10.24 Verarbeiten von Komponentenparametern

Über die Methode `parameterize` werden durch den Service-Manager die Parameter in Form eines Objekts an die Komponente übergeben. Dieses Objekt kann anschließend äußerst komfortabel ausgelesen und somit können die jeweiligen Parameterwerte erhalten werden.

10.2.5 Initialisieren von Komponenten

Initializable

Vor dem Ausliefern einer Komponente durch den Service-Manager tritt häufig der Fall auf, dass sie zuvor initialisiert werden muss, um beispielsweise Datenbankverbindungen herzustellen oder bestimmte Vorberechnungen durchzuführen. In jedem Fall ist es aber wichtig zu wissen, dass die Methode, die innerhalb einer Komponente für die Initialisierung zuständig ist, durch den Service-Manager immer als letztes aufgerufen wird, bevor er die Komponente ausgibt. Somit ist gewährleistet, eventuelle Konfigurationswerte in den Initialisierungsprozess mit einbinden zu können. Die entsprechende Methode lautet `initialize` und wird im Interface `Initializable` definiert. Listing 10.25 zeigt dieses Interface.


```
package org.apache.avalon.framework.activity;

public interface Initializable {

    void initialize() throws Exception;
}
```

Listing 10.25 Das Interface Initializable

10.2.6 Vor dem Zerstören aufräumen

Im Normalfall wird jede Komponente, die Sie mit der Methode `release` an den Service-Manager zurückgeben, anschließend zerstört. Dabei kann es jedoch vorkommen, dass unmittelbar vor dem Zerstören noch einige Aufgaben erfüllt werden müssen. Dazu könnte zum Beispiel das Schließen von Datenbankverbindungen oder der Release von verwendeten Komponenten zählen. Diese Aufgaben können Sie innerhalb der Methode `dispose` angeben, die der Service-Manager unmit-

telbar vor dem Zerstören einer Komponente aufruft. Dies ist auch dann der Fall, wenn eine Exception aufgetreten ist. Dafür wird aber vorausgesetzt, dass eine solche Komponente das Interface `Disposable` implementiert, das in Listing 10.26 dargestellt ist.

```
package org.apache.avalon.framework.activity;

interface Disposable {

   void dispose();
}
```

Listing 10.26 Das Interface Disposable

10.2.7 Pooling

Wie Sie bereits aus den vorangegangenen Abschnitten wissen, wird bei jedem Aufruf der Methode `lookup` des Service-Managers eine Komponente zurückgeliefert. Falls nichts anderes angegeben ist, liefert der Service-Manager mit jedem Aufruf eine neue Instanz der Komponente zurück. Diese Art, eine Komponente zu erzeugen, nennt man **Single-Threaded**.

SingleThreaded

Sie können dem Service-Manager jedoch auch explizit mitteilen, dass er eine Komponente als Single-Threaded behandeln soll, indem Sie die Komponente das Interface `SingleThreaded` implementieren lassen. Dabei handelt es sich um ein **Marker-Interface**, das keinerlei Methoden definiert, sondern dem Service-Manager lediglich mitteilt, welche Eigenschaft für eine entsprechende Komponente gelten soll.

Wenn eine Komponente das Interface `SingleThreaded` implementiert, erzeugt der Servcie-Manager bei jedem Aufruf eine neue Instanz davon. Dies ist das standardmäßige Vorgehen des Service-Managers, auch wenn dieses Interface nicht implementiert wird.

Da in einer Web-Umgebung teilweise mehrere Tausend solcher Anfragen in kürzester Zeit an den Service-Manager gestellt werden können, sollte dieser Weg aber nur dann gewählt werden, wenn es sich nicht vermeiden lässt oder Sie sich sicher sind, dass die Anfragen an Ihre Applikation innerhalb eines gewissen Rahmens bleiben. Dieser Rahmen ist natürlich von den verschiedensten Faktoren abhängig, die von Fall zu Fall neu bewertet werden müssen.

ThreadSafe und Singleton

Die bessere Variante zum Erzeugen einer Komponente ist das Implementieren des Interfaces `ThreadSafe`. Implementiert eine Kompo-

nente dieses Marker-Interface, erzeugt der Service-Manager nur beim ersten Aufruf von `lookup` eine Instanz der Komponente und liefert bei allen weiteren Aufrufen dieselbe Instanz zurück. Dieses Vorgehen ist auch als **Singleton-Pattern** bekannt.

Wenn eine Komponente das Interface `ThreadSafe` implementiert, erzeugt der Service-Manager nur beim ersten Aufruf der Komponente eine Instanz davon. Bei allen weiteren Aufrufen wird anschließend immer dieselbe Instanz zurückgeliefert.

Diese Methode ist wesentlich schneller, als die Verwendung von `SingleThreaded`, da nur ein einziges Mal eine Instanz erzeugt werden muss. Leider ist auch der Einsatz des Interfaces `ThreadSafe` nur in bestimmten Fällen empfehlenswert. Der Grund hierfür liegt auf der Hand: Da nur eine Instanz der Komponente vorliegt, kann die Verwendung eines Singleton-Patterns schnell zum berüchtigten Flaschenhals werden, da alle Anfragen dieselbe Instanz einer Komponente verwenden müssen. Ist eine Komponente für eine aktuelle Anfrage gerade in Verwendung, muss eine nachfolgende Anfrage solange warten, bis die aktuelle Anfrage diese Komponente nicht mehr benötigt, und kann sie erst dann verwenden.

Poolable

Avalon hält jedoch eine äußerst effektive Lösung für dieses Problem bereit: **Objekt-Pooling**. Dabei erhält jede Anfrage eine eigene Instanz einer Komponente und kann somit parallel zu anderen Anfragen bearbeitet werden. Um diesen Mechanismus nutzen zu können, müssen Sie Ihre Komponente lediglich das Marker-Interface `Poolable` implementieren lassen.

In Kapitel 8.12.2, *Komponenten-Pool*, können Sie mehr über das Thema **Pooling** erfahren. Dort wird speziell auf die Konfiguration eines Komponenten-Pools innerhalb der Cocoon-Sitemap eingegangen. Diese Konfiguration können Sie jedoch auch in der Datei `cocoon.xconf` mit denselben Attributen im jeweiligen Komponentenelement vornehmen. Listing 10.27 zeigt, wie die Pooling-Parameter verwendet werden können.

```
...
<my-component pool-min="1" pool-max="5" pool-grow="1"/>
...
```

Listing 10.27 Konfiguration des Komponenten-Pools in der Datei cocoon.xconf

Welche Bedeutung die einzelnen Attribute zur Konfiguration des Pools besitzen, können Sie ebenfalls in Kapitel 8.12.2, *Komponenten-Pool*, nachschlagen.

Wird das Pooling einer Komponente nicht explizit durch die angegebenen Attribute vorgenommen, werden automatisch die Standardwerte verwendet. Voraussetzung ist allerdings, dass ein entsprechendes Marker-Interface wie `Poolable` oder `Recyclable` implementiert wurde.

Doch auch hier kann es einen Grund geben, diese Art der Komponentenverwaltung nicht zu verwenden. Stellen Sie sich folgende Situation vor: Eine Anfrage benötigt eine Komponente, wie etwa `MyComponent`, die als Beispiel in den vorherigen Abschnitten erstellt wurde. Im Laufe der Verarbeitung dieser Anfrage werden Werte in der Komponente gesetzt und/oder verändert. Nachdem die Verarbeitung abgeschlossen ist, wird die Komponente mit den aktuellen Einstellungen durch die Methode `release` wieder an den Service-Manager zurückgegeben. Eine andere Anfrage kann nun diese Komponente ebenfalls durch `lookup` erhalten, wobei hier immer noch die Werte der vorhergehenden Verarbeitung gesetzt sind. Dies muss nicht unbedingt problematisch sein, kann jedoch dann zu einem unvorhersehbaren Verhalten der Anwendung führen, wenn diese darauf angewiesen ist, eine vollständig initialisierte Komponente zu erhalten.

Recyclable

Um dieses Problem zu lösen, können Sie statt `Poolable` das Interface `Recyclable` verwenden. `Recyclable` ist von `Poolable` abgeleitet und definiert zusätzlich die Methode `recycle`, wie in Listing 10.28 zu sehen ist. Diese Methode wird immer dann aufgerufen, wenn eine Komponente durch `release` an den Service-Manager zurückgegeben wurde.

```
public interface Recyclable extends Poolable {
   void recycle();
}
```

Listing 10.28 Das Interface Recyclable

recycle

Wie der Name der Methode `recycle` bereits andeutet, können Sie in ihr alle Arbeiten ausführen lassen, die notwendig sind, um bei der nächsten Ausgabe eine »saubere« Komponente zu übergeben. Zu diesen Arbeiten könnten zum Beispiel das Zurücksetzen von Werten auf ihre Standardeinstellungen oder das Freigeben von Ressourcen wie etwa Datenbankverbindungen zählen.

Da eine »gepoolte« Komponente in der Regel niemals zerstört wird, kann hier das Interface `Disposable` auch nicht verwendet werden, um Aktionen auszuführen, die vor der Zerstörung erfolgen sollen. Stattdessen können Sie aber das Interface `Recyclable` verwenden und

innerhalb der Methode `recycle` alle Vorgänge ausführen, die sonst in `dispose` stehen würden.

Denken Sie daran, dass die Methode `recycle` mit jedem Release einer Komponente erneut aufgerufen wird. Die Methoden der Lifecycle-Interfaces, wie beispielsweise `contextualize`, `parameterize`, `initialize` usw., werden hingegen immer nur beim erstmaligen Anfordern der Komponente aufgerufen und dann nicht mehr.

10.2.8 Logging

Ein wichtiger Aspekt in der Programmierung ist das Loggen von Informationen, die den Zustand der Applikation beschreiben. Cocoon besitzt ein komfortables und umfangreiches Logging-System: **LogKit**. Dabei handelt es sich um ein Subprojekt von Avalon.

Konfiguration

Wie Sie das Logging verschiedener Komponenten konfigurieren können, wurde in Kapitel 8.12.13, *Logging*, erklärt. Diese Einstellungen können Sie auch für Komponenten verwenden, die in der Datei `cocoon.xconf` angegeben sind. In diesem Abschnitt interessiert uns nun, wie Sie innerhalb einer selbst erstellten Komponente die Logging-Funktionalitäten verwenden können.

LogEnabled

Zunächst muss jede Komponente, die diese Funktionalitäten bereitstellen soll, das Interface `LogEnabled` implementieren, das in Listing 10.29 gezeigt ist.

```
package org.apache.avalon.framework.logger;

public interface LogEnabled {

   void enableLogging(Logger logger);
}
```

Listing 10.29 Das Interface LogEnabled

Aufrufreihenfolge

Falls eine Komponente dieses Interface implementiert, wird die Methode `enableLogging` durch den Service-Manager aufgerufen, bevor alle anderen implementierten Lifecycle-Interfaces abgearbeitet werden. Dadurch wird es Ihnen ermöglicht, auch die Aktionen innerhalb der einzelnen Lifecycle-Methoden zu protokollieren. Sehen Sie sich hierzu eventuell nochmals Abbildung 10.1 an.

Logger

Sie erhalten dann ein Objekt vom Typ `Logger`, dessen Interface in Listing 10.30 gezeigt wird. Dieses Objekt können Sie anschließend als In-

stanz-Variable ablegen, um innerhalb der anderen Methoden Zugriff darauf zu erhalten.

```
package org.apache.avalon.framework.logger;

public interface Logger {

   void debug( String message );
   void debug( String message, Throwable throwable );
   boolean isDebugEnabled();

   void info( String message );
   void info( String message, Throwable throwable );
   boolean isInfoEnabled();

   void warn( String message );
   void warn( String message, Throwable throwable );
   boolean isWarnEnabled();

   void error( String message );
   void error( String message, Throwable throwable );
   boolean isErrorEnabled();

   void fatalError( String message );
   void fatalError( String message,
      Throwable throwable );
   boolean isFatalErrorEnabled();

   Logger getChildLogger( String name );
}
```

Listing 10.30 Das Interface Logger

Log-Levels

Ein Logger-Objekt besitzt verschiedene Methoden, um Meldungen zu registrieren. Je nachdem, welches Level in der Konfiguration eingestellt wurde, werden die Meldungen entgegengenommen oder nicht. Dabei ist zu beachten, dass es hierbei eine Reihenfolge gibt, die besagt, dass ein Level, das aktiviert ist, automatisch alle darüber liegenden Levels ebenfalls aktiviert. Diese Reihenfolge lautet:

1. `DEBUG`
   Ausgaben für den Entwickler während der Anwendungsentwicklung oder für das Debugging. In der Regel werden in diesem Level z. B. wichtige Klassen- und Methodenaufrufe geloggt.

2. INFO
   Nützliche Informationen für den Benutzer, wie z.B. Verbindungsaufbau, Statusmeldungen usw.
3. WARN
   Ein kleines Problem oder ein Konflikt ist aufgetreten, der die Funktionsweise des Systems beeinträchtigen kann.
4. ERROR
   Ein Fehler ist aufgetreten, der schnellstmöglich behoben werden muss. Er kann dazu führen, dass das System überhaupt nicht mehr oder zumindest nicht mehr korrekt arbeitet.
5. FATAL_ERROR
   Ein schwer wiegender Fehler ist aufgetreten. Das Auftreten dieses Fehlers führt dazu, dass das System nicht weiter lauffähig ist. Dieser Fehler muss umgehend behoben werden, da sonst kein weiterer Betrieb des Systems möglich ist.

Ist also beispielsweise das Level auf den Wert WARN gesetzt, werden zusätzlich auch alle Meldungen der Levels ERROR und FATAL_ERROR mit ausgegeben. Die Levels DEBUG und INFO hingegen treten nicht in Aktion.

Log-Methoden

Das Loggen selbst wird durch die entsprechende Methode debug, info, warn, error oder fatalError durchgeführt, wobei diesen Methoden entweder nur der zu registrierende Text oder zusätzlich ein Objekt vom Typ Throwable übergeben werden kann. Ein solches Objekt könnte zum Beispiel eine Exception sein.

Enabled-Methoden

Aus Performance-Gründen existiert zu jedem Level zusätzlich die Methode is[X]Enabled, wobei [X] für das jeweilige Level steht. Diese Methode liefert den boolschen Wert true zurück, wenn das entsprechende Level aktiviert ist. Dadurch kann vor der Ausgabe der Meldung überprüft werden, ob dies überhaupt notwendig ist. Stellen Sie sich hierfür das einfache Loggen der folgenden Debug-Meldung vor:

```
debug("Das ist eine Test-Meldung der Klasse: " +
   this.getClass());
```

Listing 10.31 Logging ohne Enabled-Abfrage

Da eine solche String-Konkatenation relativ aufwendig ist und vor allem mehrfach innerhalb einer Applikation auftreten kann, ist anzuraten, in ausschließlich allen Fällen die entsprechende Enabled-Funktion zu verwenden. Hiermit wird zuvor überprüft, ob das jeweilige Level

überhaupt aktiviert ist, bevor das Loggen vorgenommen wird. Denn die Abfrage eines einfaches booleschen Wertes ist weniger zeitintensiv als das Loggen auch nur eines einzigen Zeichens, das sowieso nicht ausgegeben wird. Die Log-Ausgabe aus Listing 10.31 würde mit der Enabled-Abfrage folgendes Aussehen erhalten:

```
if(isDebugEnabled())
   debug("Das ist eine Test-Meldung
      der Klasse: " + this.getClass());
```

Listing 10.32 Logging mit Enabled-Abfrage

Vor dem Loggen sollten Sie aus Performance-Gründen immer zuerst mit den enstprechenden Enabled-Funktionen überprüfen, ob das jeweilige Log-Level überhaupt aktiviert ist.

Zur Vereinfachung des Loggens stellt Avalon bereits eine Implementierung von `LogEnabled` zur Verfügung: `AbstractLogEnabled`. Falls Sie diese abstrakte Basisklasse verwenden, wird auch hier das Logger-Objekt durch die Methode `enableLogging` übergeben und kann anschließend zusätzlich innerhalb der Komponente über die Methode `getLogger` erhalten werden. In Listing 10.33 können Sie einen Ausschnitt einer Komponente sehen, die `AbstractLogEnabled` verwendet.

```
...
import org.apache.avalon.framework.logger.↵
AbstractLogEnabled;
...
public class MyComponent extends AbstractLogEnabled {

   public void myMethod(String someValue) {
      if(this.getLogger().isDebugEnabled())
         this.getLogger().debug("Aufruf der Methode
            myMethod(" + someValue +");
      ...
   }
   ...
}
```

Listing 10.33 Verwenden von AbstractLogEnabled

Das Interface `Loggable` sowie die abstrakte Klasse `AbstractLoggable` sind als »deprecated« eingestuft und sollten deshalb nicht mehr verwendet werden. Die direkten Nachfolger sind `LogEnabled` und `AbstractLogEnabled`.

10.3 Source-Resolving

Ein leidiges Problem, das vielen Entwicklern immer wieder Kopfzerbrechen bereitet, ist das Auflösen von Pfadangaben und das Erhalten von Ressourcen von verschiedenen Orten. Die Programmiersprache Java stellt in ihrer API-Sammlung zwar relativ komfortable Möglichkeiten zur Verfügung, mit der viele URIs automatisch aufgelöst und die Ressourcen mit den entsprechenden Protokollen gelesen werden können, Cocoon geht hier aber noch einen deutlichen Schritt weiter.

10.3.1 SourceResolver von Cocoon

Für die Umsetzung eines solchen Vorhabens wird ein bestimmter Bestandteil von Cocoon benötigt: der **SourceResolver**. Zugriff auf eine Instanz des SourceResolvers erhalten Sie in jeder Sitemap-Komponente über einen Methoden-Parameter, der an die Komponente übergeben wird. Voraussetzung ist natürlich, dass Sie eine Komponente erzeugen möchten, die Zugriff auf eine Ressource benötigt. Dies kann eine Datei auf dem lokalen Rechner oder einem entfernten Server sein. Vor allem die Pipeline-Komponenten Generator, Transformer und Reader benötigen solche Zugriffe häufig. In Listing 10.34 sehen Sie das Interface des SourceResolvers von Cocoon.


```
package org.apache.cocoon.environment;

import java.io.IOException;
import org.apache.cocoon.ProcessingException;
import org.xml.sax.SAXException;

public interface SourceResolver extends
   org.apache.excalibur.source.SourceResolver {

   Source resolve(String systemID)
      throws ProcessingException,
      SAXException, IOException;
}
```

Listing 10.34 Das Cocoon-Interface SourceResolver

Wenn Sie eine Instanz des SourceResolvers besitzen, können Sie mit der Methode `resolve` eine URI übergeben. Falls die angegebene Adresse aufgelöst und die darin angegebene Ressource geladen werden kann, erhalten Sie anschließend als Rückgabewert ein Objekt vom Typ `Source`. In Listing 10.35 ist das entsprechende Interface hierzu angegeben.

```
package org.apache.cocoon.environment;

import org.apache.avalon.excalibur.pool.Recyclable;
import org.apache.cocoon.ProcessingException;
import org.apache.excalibur.xml.sax.XMLizable;
import org.xml.sax.InputSource;

import java.io.IOException;
import java.io.InputStream;

public interface Source extends Recyclable, XMLizable {

   long getLastModified();

   long getContentLength();

   InputStream getInputStream()
      throws ProcessingException, IOException;

   InputSource getInputSource()
      throws ProcessingException, IOException;

   String getSystemId();
}
```

Listing 10.35 Das Cocoon-Interface Source

Wie aus Listing 10.35 zu ersehen ist, muss das Objekt `Source` verschiedene Methoden implementieren, die Zugriff auf bestimmte Attribute der Ressource erlauben. Um beispielsweise eine Ressource, die über eine URI von einem entfernten Rechner geladen wurde, lokal zu speichern, könnten Sie die Methode `getInputStream` verwenden, um auf gewohnte »Java-Weise« den Stream der Ressource in eine Datei zu schreiben.

Verwenden Sie innerhalb von eigenen Komponenten immer den SourceResolver, um Ressourcen zu erhalten. Dadurch können alle zusätzlichen Funktionalitäten, wie z.B. eigene Pseudoprotokolle in vollem Umfang verwendet werden.

In Listing 10.36 ist exemplarisch gezeigt, wie innerhalb der Methode `act` einer Action der SourceResolver verwendet wird, um eine Ressource zu laden, deren URI über einen Parameter in der Sitemap übergeben wurde.

```
...
public Map act(Redirector redirector,
   SourceResolver sourceResolver, Map map, String str,
   Parameters parameters) throws Exception {

   // Pfadangabe aus der Sitemap erhalten
   String path = parameters.getParameter("path");

   // Variable deklarieren
   Source src;

   try {

      // Ressource laden
      src = sourceResolver.resolve(path);

      // Irgendwas mit src machen ...

   } finally {

      // Ressource recyclen
      if(src != null)
         src.recycle();
   }
}
...
```

Listing 10.36 Ein Beispiel zur Verwendung des SourceResolvers

Recyclable

Da ein Source-Objekt oftmals Zugriff auf andere Avalon-Komponenten benötigt, die wieder an den Service-Manager zurückgegeben werden müssen, implementiert jede Source zusätzlich das Interface `Recyclable`. Innerhalb der Methode `recycle`, die durch dieses Interface festgelegt wird, können die verwendeten Komponenten »released« und somit zurück an den Service-Manager gegeben werden.

In Listing 10.36 sehen Sie das standardmäßige Vorgehen zur Verwendung eines Source-Objekts. Innerhalb des `finally`-Blocks wird die Methode `recycle` aufgerufen. Dadurch erfolgt dieser Aufruf in jedem Fall. Egal, ob ein Fehler aufgetreten ist oder nicht. Normalerweise würde diese Methode automatisch vom Service-Manager aufgerufen werden, wenn eine Komponente zurückgegeben wird. Im Fall von Source kommt jedoch kein solcher Manager zum Einsatz, weshalb die Methode `recycle` »von Hand« aufgerufen werden muss.

Von jedem Source-Objekt, das verwendet wurde, muss anschließend immer die Methode `recycle` »von Hand« aufgerufen werden.

Wenn Sie den vorangegangenen Abschnitt über Komponenten und den Service-Manager aufmerksam gelesen haben, stellen Sie sich hier vielleicht die Frage, weshalb für das Erhalten des SourceResolvers ein anderes Vorgehen notwendig ist als für die üblichen Cocoon-Komponenten, die allesamt über den Service-Manager verwaltet werden können. Diese Frage ist durchaus berechtigt, denn die bis hierher erklärte Vorgehensweise, um an den SourceResolver zu gelangen, ist zwar nicht falsch, widerspricht aber in einigen Punkten den Vorgaben von Avalon. Dieselbe Frage haben sich auch die Entwickler von Cocoon gestellt und in Cocoon 2.1 hierfür eine Lösung implementiert.

10.3.2 Der SourceResolver von Avalon

Der SourceResolver wurde in das Subprojekt Excalibur von Avalon ausgelagert und ist nun in gewohnter Weise als Komponente über einen Service-Manager innerhalb von Cocoon ansprechbar. Dabei wurde das Source-Interface soweit verallgemeinert, dass es nun jegliche Arten von Ressourcen repräsentieren kann. Da in Cocoon jedoch hauptsächlich mit XML-Ressourcen gearbeitet wird, wurde das Source-Interface von Avalon durch Ableitung dahingehend spezialisiert, dass es zusätzlich die Methode `getInputSource` definiert, um leicht an das Objekt `InputSource` zu gelangen, das für eine Bearbeitung mit SAX notwendig ist.

XMLizable

Des Weiteren wird das Interface `XMLizable` von Excalibur implementiert, das die Methode `toSAX` definiert. Damit jedoch auch die Architektur der bestehenden Sitemap-Komponenten, die Zugriff auf eine Ressource benötigen, beibehalten werden kann, wurde der SourceResolver von Avalon durch Ableitung um die Methode `resolve` erweitert, die von den entsprechenden Komponenten aufgerufen wird.

Im Wesentlichen ist die Art, wie Sie auf den SourceResolver zugreifen, davon abhängig, welche Art einer Ressource Sie laden und in welcher Umgebung Sie sie verarbeiten möchten. Für eine Verarbeitung von XML-Dokumenten bietet sich die Spezialisierung von Cocoon an. Für alle anderen Ressourcentypen reicht der SourceResolver von Avalon in jedem Fall. In Listing 10.37 sehen Sie das Interface für den SourceResolver aus Avalon.

```
package org.apache.excalibur.source;

import java.io.IOException;
import java.net.MalformedURLException;
import java.util.Map;

import org.apache.avalon.framework.component.Component;

public interface SourceResolver
   extends Component {

   String ROLE = SourceResolver.class.getName();
   String METHOD = "org.apache.avalon.excalibur.↵
   source.Source.uri.method";
   String URI_PARAMETERS = "org.apache.excalibur.↵
   source.Source.uri.parameters";

   Source resolveURI(String location)
       throws MalformedURLException, IOException;

   Source resolveURI(String location,
                     String base,
                     Map parameters )
       throws MalformedURLException, IOException;

   void release(Source source);
}
```

Listing 10.37 Das Excalibur-Interface SourceResolver

Beachten Sie in Listing 10.34, dass die Methode `resolve` hier ein Objekt vom Typ `org.apache.cocoon.environment.Source` zurückliefert. Demgegenüber steht die Methode `resolveURI`, die ein Objekt vom Typ `org.apache.excalibur.source.Source` zurückliefert.

```
package org.apache.excalibur.source;

import java.io.IOException;
import java.io.InputStream;

public interface Source {

   public boolean exists();
```

```
   InputStream getInputStream()
      IOException, SourceNotFoundException;

   String getURI();

   String getScheme();

   SourceValidity getValidity();

   void refresh();

   String getMimeType();

   long getContentLength();

   long getLastModified();
}
```

Listing 10.38 Das Excalibur-Interface Source

In Listing 10.39 wird wieder eine Action gezeigt, die diesmal jedoch den SourceResolver über den Service-Manager und nicht über die Methoden-Parameter erhält. Die wichtigsten Veränderungen gegenüber Listing 10.36 sind dabei besonders hervorgehoben.

```
...
import org.apache.avalon.framework.service.ServiceManager;
import org.apache.avalon.framework.service.↵
       ServiceException;
import org.apache.avalon.framework.service.Serviceable;
import org.apache.avalon.excalibur.source.Source;
...
public class MyAction extends AbstractAction
   implements Serviceable {

   private ServiceManager manager;
   ...
   public Map act(Redirector redirector,
      SourceResolver sourceResolver,
      Map map, String str, Parameters parameters)
      throws Exception {

      // Pfadangabe aus der Sitemap erhalten
      String path = parameters.getParameter("path");
```

```
        Source src = null;
        SourceResolver resolver = null;

        try {

           // Avalon-Source-Resolver erhalten
           resolver =
             (SourceResolver)this.manager.↵
             lookup(SourceResolver.ROLE);

           // Ressource laden
           src = resolver.resolveURI(path);

           // Irgendwas mit src machen ...

        } catch (ServiceException ex) {
           // Fehlerbehandlung ...
        } finally {

           // Ressource freigeben
           resolver.release(src);
        }
     }

     public void service(ServiceManager manager)

        throws ServiceException {

        this.manager = manager;
     }
}
...
```

Listing 10.39 Den SourceResolver über den Service-Manager erhalten

Der SourceResolver von Cocoon ist eine Spezialisierung des SourceResolvers von Excalibur. Er ist darauf ausgelegt, XML-Dokumente zu verarbeiten und eine Kompatibilität zu den Sitemap-Komponenten herzustellen.

10.4 Eigene Sitemap-Komponenten erstellen

Obwohl Cocoon eine Vielzahl an verschiedenen Sitemap-Komponenten zur Verfügung stellt, die so ziemlich jeden Bedarf abdecken, kann es durchaus notwendig werden, eigene Sitemap-Komponenten zu pro-

grammieren oder bestehende zu erweitern. Im Grunde ist dies relativ einfach umzusetzen, wenn man die grundlegenden Zusammenhänge und Vorgehensweisen der einzelnen Komponenten kennt.

Alle Sitemap-Komponenten basieren auf den Vorgaben von Avalon für Komponenten, d. h., dass für sie alle Regeln gelten, die in den vorhergehenden Abschnitten erläutert wurden. Cocoon erweitert einige dieser Regeln in einem bestimmten Rahmen, ohne das Konzept von Avalon zu verletzen. Die Vorgehensweise ist jedoch auch hier dieselbe: Eine Komponente ist einer bestimmten Rolle zugewiesen, darüber hinaus können die Kontrollmechanismen und Zusatzfunktionen durch das Implementieren spezieller Interfaces realisiert werden.

10.4.1 Setup von Sitemap-Komponenten

Das Interface `SitemapModelComponent` wird von den Sitemap-Komponenten vom Typ Generator, Transformer und Reader implementiert, da sie Zugriff auf Ressourcen benötigen und/oder über zusätzliche Werte konfiguriert werden müssen.

Dieses Interface definiert lediglich die Methode `setup`, die als Parameter verschiedene Werte übergeben bekommt. Die Methode `setup` wird durch den Cocoon-Prozessor unmittelbar nach den Lifecycle-Methoden von Avalon aufgerufen.

```
package org.apache.cocoon.sitemap;

import org.apache.avalon.framework.component.Component;
import org.apache.avalon.framework.parameters.Parameters;
import org.apache.cocoon.ProcessingException;
import org.apache.cocoon.environment.SourceResolver;
import org.xml.sax.SAXException;

import java.io.IOException;
import java.util.Map;

public interface SitemapModelComponent extends
   Component {

   void setup(SourceResolver resolver, Map objectModel,
      String src, Parameters par) throws
      ProcessingException, SAXException, IOException;
}
```

Listing 10.40 Das Interface SitemapModelComponent

Nachfolgend werden die einzelnen Parameter der Methode `setup` aufgelistet und erklärt:

- `SourceResolver resolver`
  Durch dieses Objekt können Ressourcen von allen Orten geladen werden, deren Protokolle Cocoon unterstützt.
- `Map objectModel`
  Diese Map erlaubt den Zugriff auf verschiedene Objekte der aktuellen Umgebung, wie beispielsweise das Request- oder Response-Objekt von Cocoon.
- `String src`
  Dieser String enthält den optionalen Wert, der im Attribut `src` des Elements in der Pipeline angegeben wurde.
- `Parameters parameters`
  Das Objekt `parameters` enthält alle Konfigurationsparameter, die innerhalb des Elements in der Sitemap angegeben wurden.

10.4.2 Zugriff auf den Output-Stream

Jede Sitemap-Komponente, die die Möglichkeit besitzen soll, den Output-Stream zu modifizieren, implementiert das Interface `SitemapOutputComponent`. Standardmäßig sind dies die Komponenten vom Typ Serializer und Reader. Alle anderen Komponenten modifizieren den Ausgabe-Stream von Cocoon entweder überhaupt nicht oder nur indirekt.

```
package org.apache.cocoon.sitemap;

import org.apache.avalon.framework.component.Component;

import java.io.IOException;
import java.io.OutputStream;

public interface SitemapOutputComponent extends
   Component {

   void setOutputStream(OutputStream out)
      throws IOException;

   String getMimeType();

   boolean shouldSetContentLength();
}
```

Listing 10.41 Das Interface SitemapOutputComponent

getMimeType Durch die Methode `getMimeType` wird – falls möglich – der Mime-Type der Ausgabe zurückgeliefert, andernfalls der Fluchtwert `null`.

shouldSetContentLength Anhand des booleschen Werts, der durch die Methode `shouldSetContentLength` zurückgeliefert wird, entscheidet Cocoon, ob die Länge der Ausgabe ermittelt und im Response gesetzt werden soll. Einige Formate benötigen eine solche Angabe, um die Ausgabe korrekt darstellen zu können.

setOutputStream Über die Methode `setOutputStream` wird durch Cocoon schließlich ein Objekt vom Typ `OutputStream` übergeben, an das weitere Ausgaben »angehängt« werden können.

10.4.3 Producer, Consumer oder Pipe?

Um in den nachfolgenden Abschnitten bestimmte Zusammenhänge verständlicher zu machen, möchte ich kurz erklären, was es mit den Interfaces `XMLProducer`, `XMLConsumer` und `XMLPipe` auf sich hat.

Wie Sie bereits wissen, gibt es innerhalb einer Pipeline verschiedene Komponenten, die jeweils eine spezielle Aufgabe besitzen. Der Generator erzeugt beispielsweise eine XML-Struktur, der Transformer wandelt eine bestehende XML-Struktur und der Serializer serialisiert eine XML-Struktur. Je nachdem, welche dieser Aufgaben eine Komponente besitzt, implementiert sie eines der genannten Interfaces. Dabei haben diese Interfaces folgende Bedeutung:

- `XMLProducer`
  Dieses Interface wird von Sitemap-Komponenten implementiert, die eine XML-Struktur erzeugen/verarbeiten und an eine andere Komponente weiterreichen können, die das Interface `XMLConsumer` implementiert.
- `XMLConsumer`
  Implementiert eine Komponente dieses Interface, ist in der Lage, SAX-Events von einem Producer entgegenzunehmen.
- `XMLPipe`
  Dieses Interface ist eine Kombination von `XMLProducer` und `XMLConsumer`. Es wird von allen Sitemap-Komponenten implementiert, die SAX-Events empfangen, verarbeiten und an eine andere Komponenten weiterreichen können.

10.4.4 Eigene Action erstellen

Zunächst wollen wir uns die Erstellung einer Sitemap-Komponente ansehen, die mit Sicherheit zu der am meisten selbst implementierten Komponente von Cocoon-Anwendern zählt: die Action. Sie gehört zwar zu den Sitemap-Komponenten, hat jedoch keinen direkten Einfluss auf die darin auftretenden SAX-Events. In ihr kann eine beliebig aufwändige Logik implementiert werden, die innerhalb einer Pipeline an einem bestimmten Punkt ausgeführt wird. Eine Action kann eher als eine umfangreichere Form eines Selectors oder Matchers verstanden werden.

das Interface Action

Jede Action muss das Interface Action aus Listing 10.42 implementieren. Dieses Interface definiert eine einzige Methode `act`. Diese Methode wird automatisch durch Cocoon aufgerufen, wenn innerhalb einer Pipeline auf ein Action-Element gestoßen wird.


```
package org.apache.cocoon.acting;

import org.apache.avalon.framework.component.Component;
import org.apache.avalon.framework.parameters.↵
Parameters;
import org.apache.cocoon.environment.Redirector;
import org.apache.cocoon.environment.SourceResolver;

import java.util.Map;

public interface Action extends Component {

   String ROLE = Action.class.getName();

   Map act(Redirector redirector, SourceResolver
      resolver, Map objectModel, String source,
      Parameters parameters) throws Exception;
}
```

Listing 10.42 Das Interface Action

die Methode act()

Der Methode `act` werden zwar – bis auf den Redirector – dieselben Parameter übergeben, wie sie auch in der Methode `setup` des Interfaces `SitemapModelComponent` vorhanden sind, eine Action implementiert jedoch dieses Interface nicht.

Map Wie Sie sicherlich aus Listing 10.42 entnehmen konnten, muss die Methode `act` ein Object vom Typ `Map` zurückliefern. Diese Map enthält Name-Wert-Paare, die zurück an die Pipeline gegeben werden und dort verwendet werden können. Sehen Sie sich das nachfolgende Listing an. Dort wird innerhalb einer Pipeline eine Action aufgerufen, die den aktuellen Wochentag bestimmt und diesen anschließend als String unter dem Namen `weekday` in der Map zurückliefert. Dieser Wert wird anschließend von einem Generator verwendet, um ein XML-Dokument einzulesen.

```
...
<map:match ...>
   <map:act type="getweekday">
      <map:generate type="file" src="{weekday}.xml"/>
      <map:serialize type="xml"/>
   </map:act>
   ...
</map:match>
...
```

Listing 10.43 Verwendung von Action-Parametern

Beachten Sie jedoch, dass der Inhalt von `<map:act/>` nur dann ausgeführt wird, wenn ein Objekt vom Typ `Map` von der Action zurückgeliefert wurde. Ob diese Map jedoch Einträge enthält oder nicht, spielt dabei keine Rolle.

Null Liefert die Methode `act` hingegen den Wert `null` zurück, wird der Inhalt von `<map:act/>` nicht ausgeführt und dieses Element übersprungen. Somit lassen sich Verzweigungen innerhalb einer Pipeline realisieren, die an komplexe Bedingungen geknüpft sein können.

Da eine Action auch eine Avalon-Komponente ist, kann sie zusätzlich alle Lifecycle-Interfaces, wie z.B. `ThreadSafe`, `LogEnabled` oder `Serviceable`, implementieren. Hierfür existieren bereits verschiedene abstrakte Basisklassen, die jeweils unterschiedliche Funktionalitäten zur Verfügung stellen. Die beiden wichtigsten lauten:

- `AbstractAction`
  Diese abstrakte Basisklasse ist von `AbstractLogEnabled` abgeleitet und stellt somit Logging-Funktionalitäten zur Verfügung.
- `AbstractConfigurableAction`
  Diese Action ist von `AbstractAction` abgeleitet und stellt somit alle Funktionalitäten zur Verfügung, die auch diese Klasse anbietet.

Darüber hinaus implementiert sie das Interface `Configurable`, womit es durch eingeschachtelte Elemente konfigurierbar wird.

Es gibt noch wesentlich mehr Basisklassen, die jeweils an spezielle Anforderungen angepasst sind. Einen vollständigen und aktuellen Überblick erhalten Sie, wenn Sie die aktuelle Apidoc[2] von Cocoon durchsehen.

Der einfachste Weg, eine eigene Action zu implementieren, ist, von einer vorhandenen Abstract-Action abzuleiten, die den eigenen Bedürfnissen am ehesten entspricht. In diesem Fall müssen Sie häufig nur noch die Methode `act` implementieren. Ein etwas anderer, aber durchaus nicht ungewöhnlicher Weg ist es, die einzelnen Interfaces selbst zu implementieren.

Eine Action muss mindestens das Interface `Action` von Cocoon implementieren. Die Methode `act` muss entweder eine Java-Map mit `0..n` Werten oder `null` zurückliefern. Der Rumpf von `<map:act/>` wird nur dann ausgeführt, wenn eine Map zurückgeliefert wurde, die auch leer sein darf. Die Keys innerhalb der Map können als Sitemap-Variablen verwendet werden, um auf den Wert in der Map zuzugreifen.

Ein kleines Beispiel

In diesem Beispiel soll eine kleine Action erstellt werden, welche dafür verwendet wird, an einem bestimmten Tag in der Woche eine andere Seite anzuzeigen als an allen üblichen Tagen. Damit lässt sich z.B. eine Hinweisseite realisieren, die dem Besucher einer Website anzeigt, dass gerade eine regelmäßige Wartung durchgeführt wird. Ob man in der Realität hierfür einen ganzen Tag ansetzt, möchte ich nicht näher kommentieren, so etwas soll es aber geben. Um in diesem Beispiel den Fokus jedoch klar auf das Erstellen einer Action zu legen, soll uns dieser einfache Anwendungsfall vorerst genügen. Falls Sie möchten, können Sie später diese Action speziell Ihren eigenen Bedürfnissen anpassen, was zugleich eine vortreffliche Übung wäre.

Wechseln Sie zunächst in das Verzeichnis `$COCOON_HOME/WEB-INF/classes`, erzeugen Sie dort eine Package-Struktur `foo.bar.cocoon.acting` und erstellen Sie darin die Action-Klasse `MaintenanceAction.java`, wie im folgenden Listing angegeben.

2 http://cocoon.apache.org/2.1/apidocs/org/apache/cocoon/acting/package-summary.html

```
package foo.bar.cocoon.acting;

import org.apache.avalon.framework.parameters.Parameters;

import org.apache.cocoon.acting.AbstractAction;
import org.apache.cocoon.environment.SourceResolver;
import org.apache.cocoon.environment.Redirector;

import java.util.Map;
import java.util.HashMap;
import java.util.Calendar;

public class MaintenanceAction extends AbstractAction {

   public Map act(Redirector redirector,
      SourceResolver sourceResolver, Map map,
      String str, Parameters parameters)
      throws Exception {

      // Konfigurations-Parameter holen
      int maintenanceDay =
         parameters.getParameterAsInteger(
         "maintenanceDay", 6);

      // Aktuelles Datum ermitteln
      Calendar now = Calendar.getInstance();

      // Aktuellen Wochentag ermitteln
      int nowDay = now.get(Calendar.DAY_OF_WEEK);

      // Ist heute maintenanceDay?
      if(nowDay == maintenanceDay)
         return new HashMap();

      return null;
   }
}
```

Listing 10.44 Die Klasse MaintenanceAction.java

Innerhalb der Klasse `MaintenanceAction` wird zunächst der konfigurierte Wert entgegengenommen, der den Service-Tag bestimmt. Dieser Wert wird aus Gründen der Übersichtlichkeit als Integer angegeben.

Dabei beginnt die Nummerierung der einzelnen Tage – wie durch die Klasse `Calendar` der Java-API vorgegeben – mit dem Sonntag, dem die Zahl `1` zugewiesen ist. Der Montag besitzt somit den Wert `2` usw.

Als Nächstes wird der aktuelle Tag ermittelt und mit dem konfigurierten Wert verglichen. Stimmen beide überein, so heißt das, dass heute der Wartungstag ist. In diesem Fall wird eine leere HashMap zurückgeliefert. Diese bewirkt, dass in der Pipeline der Rumpf von `<map:act/>` ausgeführt wird. Andernfalls wird der Wert `null` zurückgegeben und somit dieser Rumpf übersprungen. Innerhalb von `<map:act/>` kann nun die Ausgabe der Hinweisseite für die Wartung definiert werden.

Um die erstellte Action testen zu können, erstellen Sie ein Unterverzeichnis für eine Subapplikation innerhalb von `$COCOON_HOME` mit dem Namen `myaction` und fügen Sie in dieses Verzeichnis das XML-Dokument `standard.xml` ein, das geladen werden soll, wenn keine Wartung durchgeführt wird:


```
<?xml version="1.0" encoding="ISO-8859-1"?>
<text>
   Heute ist kein Wartungstag!
</text>
```

Listing 10.45 Die Datei standard.xml

An dem Tag, an dem eine Wartung durchgeführt wird, soll die Datei `maintenance.xml` angezeigt werden, die sich im selben Verzeichnis befinden muss:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<text>
   Heute ist Wartungstag!
</text>
```

Listing 10.46 Die Datei maintenance.xml

registrieren

Als letzten Schritt müssen Sie noch eine Sub-Sitemap `sitemap.xmap` innerhalb von `myaction` erzeugen, die die Action registriert und verwendet. Verwenden Sie hierfür Listing 10.47.

```
<?xml version="1.0" encoding="UTF-8"?>
<map:sitemap
   xmlns:map="http://apache.org/cocoon/sitemap/1.0">

<map:components>
```

```
    <map:actions default="maintenance">
       <!-- Action registrieren -->
       <map:action name="maintenance"
         src="foo.bar.cocoon.acting.MaintenanceAction"/>
    </map:actions>

</map:components>

<map:pipelines>

    <map:pipeline>

        <!-- Die Action verwenden -->
       <map:match pattern="">

          <map:act name="maintenance">
             <map:parameter
                name="maintenanceDay"
                value="6"/>
             <map:read type="resource"
                src="maintenance.xml"/>
          </map:act>

          <map:read type="resource"
             src="standard.xml"/>

       </map:match>

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

</map:sitemap>
```

Listing 10.47 Die sitemap.xmap

Innerhalb der Sitemap wird die Action zunächst innerhalb von `<map:actions/>` registriert und anschließend in der Pipeline verwendet. Dabei wird im Rumpf von `<map:act/>` der Parameter `maintenanceDay` definiert, der der Action übergeben wird. Anschließend ist ein Reader platziert, der die Datei `maintenance.xml` liest und ausgibt, falls der Rumpf von `<map:act/>` ausgeführt wird. Andernfalls wird die Datei `standard.xml` durch einen weiteren Reader ausgegeben, der außerhalb des Rumpfes steht.

Es mag vielleicht ein klein wenig verwirrend sein, da der Parameter `maintenanceDay` innerhalb von `<map:act/>` angegeben wird, obwohl dieser Rumpf ja nur dann ausgeführt wird, wenn die Action die entsprechende Entscheidung getroffen hat. Die im Rumpf von `<map:act/>` angegebenen Parameter werden jedoch in jedem Fall eingelesen, egal, ob er ausgeführt wird oder nicht.

Position von Parametern

Nachdem Sie die Datei `sitemap.xmap` erstellt und die Action `MaintenanceDay.java` erfolgreich kompiliert haben, können Sie nach einem eventuellen Neustart von Tomcat über die URI

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

das Ergebnis in Ihrem Webbrowser betrachten. Je nachdem, ob es sich um den Wartungstag handelt oder nicht, wird die Hinweisseite ausgegeben oder nicht.

Für weitere Tests können Sie beispielweise den Wochentag im Kalender Ihres Betriebssystems verändern und zusätzlich an den Einstellungen in der Sitemap Veränderungen vornehmen.

10.4.5 Eigenen Generator erstellen

Ein Generator einer Pipeline ist primär dafür zuständig, XML-Strukturen zu erzeugen und anschließend anderen Komponenten, wie Transformern und Serializern, zur Verfügung zu stellen. Hierfür muss er zunächst immer das Interface `Generator` implementieren, das in Listing 10.48 gezeigt wird.

```
package org.apache.cocoon.generation;

import org.apache.cocoon.ProcessingException;
import org.apache.cocoon.sitemap.SitemapModelComponent;
import org.apache.cocoon.xml.XMLProducer;
import org.xml.sax.SAXException;

import java.io.IOException;

public interface Generator extends XMLProducer,
   SitemapModelComponent {

   String ROLE = Generator.class.getName();
```

```
    void generate() throws IOException,
    SAXException, ProcessingException;
}
```

Listing 10.48 Das Interface Generator

Neben dem Interface `SitemapModelComponent`, das andeutet, dass diese Komponente durch die Methode `setup` zusätzlich initialisiert wird, ist `Generator` auch vom Interface `XMLProducer` abgeleitet, das alle Komponenten implementieren, die zwar ein XML-Dokument erzeugen, jedoch keine SAX-Events aus einer vorhergehenden Komponente verarbeiten.

Natürlich existieren auch für das Interface `Generator` Vorimplementierungen in Form von abstrakten Basisklassen. Eine davon lautet `AbstractGenerator`. Diese Klasse stellt alle Variablen aus dem Aufruf der Methode `setup` als Instanzvariablen zur Verfügung und überschreibt die Methode `recycle`, um die verwendeten Variablen nach dem Zurückgeben an den Service-Manager zu dereferenzieren. Das Interface `Recyclable` selbst wird durch den `AbstractXMLProducer` implementiert. In Listing 10.49 sehen Sie den Inhalt von `AbstractGenerator`.

```
package org.apache.cocoon.generation;

import org.apache.avalon.framework.parameters.↵
Parameters;
import org.apache.cocoon.ProcessingException;
import org.apache.cocoon.environment.SourceResolver;
import org.apache.cocoon.xml.AbstractXMLProducer;
import org.xml.sax.SAXException;

import java.io.IOException;
import java.util.Map;

public abstract class AbstractGenerator extends
    AbstractXMLProducer implements Generator {

    protected SourceResolver resolver=null;
    protected Map objectModel=null;
    protected Parameters parameters=null;
    protected String source=null;
```

```
    public void setup(SourceResolver resolver, Map
       objectModel, String src, Parameters par) throws
       ProcessingException, SAXException, IOException {

       this.resolver=resolver;
       this.objectModel=objectModel;
       this.parameters=par;
    }

    public void recycle() {

       super.recycle();
       this.resolver = null;
       this.objectModel = null;
       this.source = null;
       this.parameters = null;
     }
}
```

Listing 10.49 Die abstrakte Klasse AbstractGenerator

Parser

Da ein Generator letztendlich immer ein XML-Dokument erzeugt und die zugehörigen SAX-Events an eine nachfolgende Komponente weitergibt, ist es notwendig, dass er Zugriff auf das Parser-Objekt von Cocoon erhält. Dieser Parser parst ein ihm übergebenes XML-Dokument, indem er die Methoden für die Bearbeitung der SAX-Events der in der Pipeline nachfolgenden Komponente aufruft. Diese nachfolgende Komponente wird dem Generator durch den Cocoon-Prozessor über die Methode `setXMLConsumer` bekannt gemacht, die durch das Interface `XMLProducer` definiert ist.

Zugriff auf den Parser erhält der Generator in der Regel nur über den Service-Manager. Aus diesem Grund muss der Generator das Interface `Serviceable` implementieren, um über die Methode `service` diesen Manager zu erhalten. Doch auch hierfür existiert ab Cocoon 2.1.3 eine abstrakte Klasse mit dem Namen `ServiceableGenerator`, die von `AbstractGenerator` abgeleitet ist und das Interface `Serviceable` implementiert. Die Instanz des Service-Managers ist somit über die Klassenvariable `manager` erreichbar.

Bis zur Version 2.1.3 war ausschließlich die Klasse `Composable Generator` für einen Generator verfügbar, die das Interface Composable implementierte, um über die Methode `compose` den Compo-

nent-Manager zu erhalten. Da in den neuen Versionen von Cocoon ausschließlich der Service-Manager Verwendung finden sollte, ist von dem Einsatz dieser abstrakten Basisklasse abzuraten.

SourceUtil

Der Aufruf des Parsers, die Registrierung des XML-Consumers und das Ausführen aller weiteren Aufgaben, um die SAX-Events aus dem Source-Objekt an die nachfolgende Komponente zu übergeben, wird durch die äußerst nützliche Helper-Klasse `SourceUtil` durchgeführt, die sich im Package `org.apache.cocoon.components.source` befindet. Dadurch wird es sehr einfach, einen Generator zu implementieren, der eine Ressource einliest, in eine XML-Struktur wandelt und anschließend an die nachfolgende Komponente in der Pipeline übergibt.

Ein kleines Beispiel

In Listing 10.50 wird eine Implementierung eines solchen einfachen Generators gezeigt, der lediglich eine Ressource in Form einer XML-Datei über den `SourceResolver` lädt und per `SourceUtil` an eine nachfolgende Komponente weitergibt.

In manchen Fällen wird innerhalb eines Generators aus einer Nicht-XML-Struktur eine XML-Struktur erzeugt. Eine solche Erzeugung können Sie durchführen, indem Sie die Instanz-Variable `contentHandler` verwenden, die automatisch zur Verfügung gestellt wird, wenn Ihr Generator von `AbstractXMLProducer` abgeleitet ist.

```
package foo.bar.cocoon.generation;

import org.apache.cocoon.generation.ServiceableGenerator;
import org.apache.cocoon.ProcessingException;
import org.apache.excalibur.source.Source;
import org.apache.cocoon.components.source.SourceUtil;

import java.io.IOException;
import org.xml.sax.SAXException;

public class MyGenerator extends ServiceableGenerator {

   public void generate() throws IOException,
      SAXException, ProcessingException {

      // Ressource laden
```

```
        Source sourceObj =
           this.resolver.resolveURI(this.source);

        // Ressource parsen und die SAX-Events
        // an den Consumer uebergeben
        SourceUtil.parse(this.manager, sourceObj,
           this.xmlConsumer);
    }
}
```

Listing 10.50 Ein äußerst einfach gehaltener Generator

Nachdem Sie die Klasse `MyGenerator` aus Listing 10.51 im zugehörigen Package `foo.bar.cocoon.generation` unter `$COCOON_HOME/WEB-INF/classes` abgelegt und kompiliert haben, können Sie den Generator testen. Erstellen Sie hierfür zunächst eine Subapplikation, indem Sie hierzu ein Verzeichnis `mygenerator` unter `$COCOON_HOME` anlegen und darin eine beliebige, gültige XML-Datei mit dem Namen `myfile.xml` erzeugen. Innerhalb von `mygenerator` müssen Sie nun nur noch eine Sub-Sitemap `sitemap.xmap` erstellen, wie im folgenden Listing gezeigt.

```
<?xml version="1.0" encoding="UTF-8"?>
<map:sitemap
   xmlns:map="http://apache.org/cocoon/sitemap/1.0">

<map:components>

   <!-- Generator registrieren -->
   <map:generators default="myGenerator">
      <map:generator name="myGenerator"
         src="foo.bar.cocoon.generation.MyGenerator"/>
   </map:generators>

</map:components>

<map:pipelines>
   <map:pipeline>

      <!-- Generator verwenden -->
      <map:match pattern="">
         <map:generate type="myGenerator"
            src="myfile.xml"/>
```

```
            <map:serialize type="xml"/>
        </map:match>

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

</map:sitemap>
```

Listing 10.51 Die Sub-Sitemap sitemap.xmap von mygenerator

Durch einen Aufruf der URI

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

können Sie sehen, dass Ihr Generator korrekt arbeitet, wenn er den Inhalt der Datei `myfile.xml` in Ihrem Webbrowser wiedergibt.

10.4.6 Eigenen Transformer erstellen

Die aufwändigste Sitemap-Komponente ist in der Regel der Transformer, da er SAX-Events von einer vorhergehenden Komponente empfangen, verarbeiten und an eine nachfolgende Komponente weitergeben kann. Doch auch hier können Sie beruhigt sein, denn das Erstellen einer solchen Komponente ist dank der zahlreichen Vorimplementierungen und guten Strukturierung einfacher als man denkt.

In der Regel transformiert ein Transformer einen SAX-Stream, indem er auf eingehende SAX-Events entsprechend reagiert, diese filtert und eventuell neue SAX-Events erzeugt. Er nimmt also aus einer Menge an XML-Elementen nur diejenigen, die er benötigt, verarbeitet diese und gibt die restlichen bzw. die neu erstellten an die nächste Komponente in der Pipeline weiter.

eigener Namensraum

Damit der Transformer entscheiden kann, welche Elemente für ihn bestimmt sind, werden diese in den meisten Fällen an einen eigenen Namensraum, der speziell für einen bestimmten Transformer steht, gebunden.

Ein Transformer implementiert zunächst immer das Interface `Transformer`, das im folgenden Listing gezeigt wird.

```
package org.apache.cocoon.transformation;

import org.apache.cocoon.sitemap.SitemapModelComponent;
import org.apache.cocoon.xml.XMLPipe;
```

```
public interface Transformer extends XMLPipe,
   SitemapModelComponent {

   String ROLE = Transformer.class.getName();

}
```

Listing 10.52 Das Interface Transformer

Wie Sie in Listing 10.52 sehen können, ist das Interface `Transformer` von den weiteren Interfaces `XMLPipe` und `SitemapModelComponet` abgeleitet. Das Interface `XMLPipe` definiert dabei, dass ein Transformer sowohl SAX-Events empfangen als auch senden kann, indem es selbst wiederum die beiden Interfaces `XMLConsumer` und `XMLProducer` erweitert. Das Interface `SitemapModelComponent` gibt an, dass es sich bei einem Transformer um eine Sitemap-Komponente handelt, die durch die Methode `setup` zusätzlich konfiguriert werden kann.

Wie bereits angedeutet, existieren natürlich auch für einen Transformer Vorimplementierungen. Die wichtigste ist sicherlich die Klasse `AbstractTransformer`, von der wiederum die meisten anderen Spezialisierungen ausgehen. Durch die Tatsache, dass diese abstrakte Klasse von `AbstractXMLPipe` abgeleitet ist, ist eine implementierende Klasse in der Lage, eine nachfolgende Komponente durch `setConsumer` zu registrieren und SAX-Events durch die entsprechenden Methoden zu bearbeiten sowie weiterzureichen. Daneben ist sie logging- und poolingfähig und implementiert somit – neben anderen – die Methoden `recycle` und `getLogger`.

Neben `AbstractTransformer` existieren noch wesentlich umfangreichere Implementierungen, wie z. B. `AbstractSAXTransformer` oder `AbstractDOMTransformer`, die je nach Anwendungsfall eine gute Basis für den eigenen Transformer bilden können. Sehen Sie hierzu in der Apidoc von Cocoon im Package `org.apache.cocoon.transformation` nach. Um die grundsätzliche Implementierung eines Transformers zu erklären, werden wir uns auf die Verwendung von `AbstractTransformer` beschränken.

Ein kleines Beispiel

Im Beispiel soll ein Transformer implementiert werden, der ein XML-Dokument entgegennimmt, in das Elemente der Form

```
<sub:substitute name="[name]"/>
```

eingebettet sein können. Wenn der Transformer auf ein solches Element stößt, ersetzt er es durch ein anderes Element. Dieses neue Element erhält als Bezeichner den Namen `[name]`. Ob ein solcher Transformer im realen Einsatz Sinn machen würde, lassen wir einfach dahin gestellt. Ziel ist es einzig und allein, die Arbeitsweise eines Transformers zu verdeutlichen. Ein XML-Dokument in der Form

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<text xmlns:sub="http://foo/bar/substitute/1.0">
   <sub:subistute name="vorname">
      Martina
   </sub:substitute>
</text>
```

würde nach der Transformation durch unseren Beispieltransformer in ein Ergebnisdokument wie das nachfolgend gezeigte gewandelt werden:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<text xmlns:sub="http://foo/bar/substitute/1.0">
   <sub:vorname>
      Martina
   </sub:vorname>
</text>
```

Um einen solchen Transformer zu erstellen, legen Sie zunächst eine neue Package-Struktur `foo.bar.cocoon.transformation` im Verzeichnis `$COCOON_HOME/WEB-INF/classes` an und erstellen Sie dort eine Klasse `SubstituteTransformer.java` mit folgendem Inhalt:

```
package foo.bar.cocoon.transformation;

import org.apache.cocoon.transformation.↵
AbstractTransformer;
import org.apache.cocoon.environment.SourceResolver;
import org.apache.avalon.framework.parameters.↵
Parameters;
import org.apache.cocoon.ProcessingException;

import org.xml.sax.SAXException;
import org.xml.sax.Attributes;
import org.xml.sax.helpers.AttributesImpl;

import java.util.Map;
```

```
import java.io.IOException;

public class SubstituteTransformer extends
   AbstractTransformer {

   /** Der zu verwendende Namensraum **/
   private final String NAMESPACE =
      "http://foo/bar/substitute/1.0";

   /** Das Element, das substituiert werden soll **/
   private final String ELEMENT_SUBSTITUE =
      "substitute";

   /** Der Attributname des Attributs 'name' **/
   private final String ATTRIBUTE_NAME = "name";

   /** Der Default-Wert, der verwendet werden soll,
    *  wenn kein Wert zugewiesen wurde **/
   private final String ATTRIBUTE_NAME_DEFAULT =
      "NONE";

   private String element = null;

   public void setup(SourceResolver sourceResolver, Map
      objectModel, String str, Parameters parameters)
      throws ProcessingException, SAXException,
      IOException {

      // Nichts machen
   }

   public void startElement(String namespaceURI,
      String localName, String qName,
      Attributes attributes) throws SAXException {

      // Nur Elemente im eigenen Namensraum ...
      if((this.NAMESPACE).equals(namespaceURI)) {

         // Handelt es sich um ein Element substitute?
         if((this.ELEMENT_SUBSTITUE).
            equals(localName)){
```

```
            // Den Wert von 'name' erhalten...
            String name =
              attributes.getValue(this.ATTRIBUTE_NAME);

            // Element setzen
            this.element = name;

            // Ein leeres Attributes-Objekt erzeugen
            Attributes attrs = new AttributesImpl();

            // Veraendertes Element an die nachfolgende
            // Komponente weitergeben
            super.startElement(namespaceURI,
               this.element, this.element, attrs);
            }

         } else {


            // Alle Elemente, die nicht dem speziellen
            // Namensraum angehoeren, unveraendert an
            // die nachfolgende Komponente
            // weiterreichen
            super.startElement(namespaceURI, localName,
               qName, attributes);
      }
   }

   public void endElement(String namespaceURI, String
      localName, String qName) throws SAXException {

      // Nur Elemente im eigenen Namensraum ...
      if((this.NAMESPACE).equals(namespaceURI)) {

         // Ein goeffnetes Element wieder schliessen
         super.endElement(namespaceURI, this.element,
            this.element);

      } else {

         // Alle Elemente, die nicht dem speziellen
         // Namensraum angehoeren, unveraendert an
```

```
            // die nachfolgende Komponente
            // weiterreichen
            super.endElement(namespaceURI, localName,
               qName);
        }
    }
}
```

Listing 10.53 Die Klasse SubstituteTransformer.java

Zunächst wird die Klasse `SubstituteTransformer` von `AbstractTransformer` abgeleitet und somit automatisch gepoolt. Danach wird der Namensraum des Transformers auf

```
http://foo/bar/substitute/1.0
```

und der Name für das zu ersetzende Element auf `substitute` festgelegt. Die wichtigsten Methoden sind `startElement` und `endElement`. Falls Sie die XML-Einführung zu Beginn dieses Buches (Kapitel 5.3, *SAX*) durchgearbeitet haben oder sich in SAX auskennen, werden Sie vielleicht bemerken, dass diese beiden Methoden genau mit denselben Parametern auch im Content-Handler von SAX verwendet werden, um auf den Start bzw. das Ende eines Elements geeignet reagieren zu können. Genau für diese Zwecke werden auch die Methoden innerhalb eines Transformers verwendet.

Diejenige Komponente, die vor dem Transformer in der Pipeline steht, parst ihr XML-Dokument, indem sie indirekt für jeden auftretenden SAX-Event (`startElement`, `endElement`, `charcters` usw.) die zugehörige Methode der nachfolgenden Komponente – unseres Transformers – aufruft. Die jeweiligen Methoden zu den entsprechenden SAX-Events sind in der Klasse `AbstractXMLPipe` vorimplementiert. Nachfolgend sehen Sie in Listing 10.54 einen kleinen Ausschnitt dieser abstrakten Basisklasse, in dem die wichtigsten Methoden gezeigt sind.

```
package org.apache.cocoon.xml;

import org.xml.sax.Attributes;
import org.xml.sax.Locator;
import org.xml.sax.SAXException;

public abstract class AbstractXMLPipe extends
   AbstractXMLProducer implements XMLPipe {
```

```
    ...

    public void startDocument() throws SAXException {
       if (contentHandler != null)
          contentHandler.startDocument();
    }

    public void endDocument() throws SAXException {
       if (contentHandler != null)
          contentHandler.endDocument();
    }

    ...

    public void startElement(String uri, String loc,
       String raw, Attributes a) throws SAXException {

       if (contentHandler != null)
          contentHandler.startElement(uri, loc, raw, a);
    }

    public void endElement(String uri, String loc,
       String raw) throws SAXException {

       if (contentHandler != null)
          contentHandler.endElement(uri, loc, raw);
    }

    public void characters(char c[], int start, int len)
       throws SAXException {

       if (contentHandler != null)
          contentHandler.characters(c, start, len);
    }

    ...
}
```

Listing 10.54 Die abstrakte Klasse AbstractXMLPipe.java

Wenn Sie sich das Interface `XMLConsumer` näher ansehen, das indirekt von `AbstractXMLPipe` implementiert wird, werden Sie entdecken, dass dieses tatsächlich das Interface `ContentHandler` von SAX ablei-

tet und somit uneingeschränkt alle Methoden zur Verfügung stellt, die auch dieses Interface definiert.

In diesem Beispiel wird beim Auftreten eines SAX-Events der Form `startElement` innerhalb dieser Methode zunächst überprüft, ob das zugehörige Element dem zuvor definierten Namensraum zugeordnet ist. Ist dies der Fall, wird als Nächstes überprüft, ob es sich um das Element mit dem Namen `substitute` handelt. Trifft auch dies zu, wird anschließend der Attributwert von `name` ausgelesen. Anschließend wird ein neues Element mit diesem Namen anstelle von `substitute` an die nachfolgende Komponente in der Pipeline weitergegeben. Alle anderen Elemente, die außerhalb des Namensraums liegen, werden unverändert an die nachfolgende Komponente weitergegeben.

Damit auch alle geöffneten Element wieder geschlossen werden, wird zu guter Letzt in der Methode `endElement` ein neu erstelltes Element geschlossen oder alle anderen Elemente, die nicht dem Namensraum zugeordnet sind, wieder unverändert weitergereicht.

Nachdem Sie die in Listing 10.53 gezeigte Klasse erstellt haben, können Sie den Transformer testen. Am einfachsten ist es, wenn Sie sich wieder ein Subprojekt `mytransformer` innerhalb von `$COCOON_HOME` erstellen und darin die Sub-Sitemap `sitemap.xmap` aus dem nachfolgenden Listing platzieren. Darin wird der Transformer registriert und anschließend innerhalb einer Pipeline verwendet.

```
<?xml version="1.0" encoding="UTF-8"?>
<map:sitemap
   xmlns:map="http://apache.org/cocoon/sitemap/1.0">

<map:components>

   <!-- Transformer registrieren -->
   <map:transformers default="substitute">
      <map:transformer name="substitute"
         src="foo.bar.cocoon.transformation.↵
         SubstituteTransformer"/>
   </map:transformers>

</map:components>

<map:pipelines>
   <map:pipeline>
```

```
   <!-- Transformer verwenden -->
   <map:match pattern="">
      <map:generate type="file"
         src="myfile.xml"/>
      <map:transform type="substitute"/>
      <map:serialize type="xml"/>
   </map:match>

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

</map:sitemap>
```

Listing 10.55 Die Sub-Sitemap von mytransformer

Als letzten Schritt benötigen Sie noch eine XML-Datei mit dem Namen `myfile.xml` innerhalb von `mytransformer`. Als Inhalt können Sie beispielsweise folgendes Listing verwenden:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<person xmlns:sub="http://foo/bar/substitute/1.0">

<sub:substitute name="vorname">
   Petra
</sub:substitute>

</person>
```

Listing 10.56 Die Datei myfile.xml

Nachdem Sie die genannten Schritte ausgeführt, Ihre Klasse kompiliert und den Servlet-Container eventuell neu gestartet haben, können Sie über die URI

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

das Ergebnis der Transformation sehen:.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<person xmlns:sub="http://foo/bar/substitute/1.0">

<sub:vorname>
   Petra
</sub:vorname>

</person>
```

Listing 10.57 Das Ergebnisdokument der Transformation

Erweitern Sie eventuell Ihren Transformer anschließend um einige Features, um ein noch besseres Verständnis für diese Komponente zu bekommen, die mit Abstand zu den wichtigsten innerhalb einer Pipeline gehört. Wie wäre es z.B. mit einem Steuerelement, das das aktuelle Datum an einer bestimmten Stelle im XML-Dokument setzt?

10.4.7 Eigenen Selector erstellen

Zu den am einfachsten zu erstellenden Sitemap-Komponenten gehört der Selector. Durch diese Komponente können Verzweigungen innerhalb einer Pipeline anhand bestimmter Bedingungen realisiert werden.

Jeder Selector muss das Interface `Selector` implementieren, das im folgenden Listing dargestellt ist.

```
package org.apache.cocoon.selection;

import org.apache.avalon.framework.component.Component;
import org.apache.avalon.framework.parameters.Parameters;

import java.util.Map;

public interface Selector extends Component {

   String ROLE = Selector.class.getName();

   boolean select (String expression, Map objectModel,
      Parameters parameters);
}
```

Listing 10.58 Das Interface Selector

Dieses Interface definiert lediglich die Methode `select`, die durch den Cocoon-Prozessor aufgerufen wird. Ihr werden verschiedene Parameter übergeben. Der wichtigste hierbei ist `expression`. Dieser String enthält einen beliebigen Ausdruck, der überprüft werden kann. Dieser Ausdruck wird durch das Attribut `test` des Elements `<map:when/>` bestimmt. Je nachdem, ob dieser Ausdruck zutrifft oder nicht, liefert die Methode `select` einen der booleschen Werte `true` oder `false` zurück. Abhängig hiervon wird der Rumpf von `<map:when/>` ausgeführt oder nicht. Für jedes Element `<map:when/>` erfolgt ein erneuter Aufruf der Methode `select` und zwar so lange, bis entweder ein Aufruf den Wert `true` zurückliefert oder alle Elemente `<map:when/>` abgearbeitet wurden und dabei `false` zurücklieferten. In diesem Fall wird der Inhalt des Elements `<map:otherwise/>` ausgeführt.

Ein kleines Beispiel

Angelehnt an das Beispiel für die Komponente Action soll hier ebenfalls eine Unterscheidung zwischen den verschiedenen Wochentagen anhand der entsprechenden Zahlen erfolgen. Anhand einer solchen Zahl kann ein bestimmter Wochentag ermittelt werden, an dem eine bestimmte Website ausgegeben werden soll. Im Unterschied zur Action wird es hier aber möglich sein, auf jeden einzelnen Tag der Woche speziell reagieren zu können.

Erstellen Sie zunächst innerhalb von `$COCOON_HOME/WEB-INF/classes` die Packagestruktur `foo.bar.cocoon.selection` und fügen Sie innerhalb von `selection` die Java-Klasse `WeekDaySelector.java` hinzu, die folgenden Inhalt besitzen muss:

```
package foo.bar.cocoon.selection;

import org.apache.cocoon.selection.Selector;
import org.apache.avalon.framework.thread.ThreadSafe;
import org.apache.avalon.framework.parameters.↵
Parameters;

import java.util.Map;
import java.util.Calendar;

public class WeekDaySelector implements Selector,
   ThreadSafe {

   public boolean select(String expression, Map map,
      Parameters parameters) {

      int expressionInt = Integer.parseInt(expression);

      // Aktuelles Datum ermitteln
      Calendar now = Calendar.getInstance();

      // Aktuellen Wochentag ermitteln
      int nowDay = now.get(Calendar.DAY_OF_WEEK);

      return (expressionInt == nowDay);
   }
}
```

Listing 10.59 Die Klasse WeekDaySelector.java

Innerhalb der Klasse `WeekDaySelector.java` wird zunächst der Wert `expression`, der über das Attribut `test` im Element `<map:select/>` übergeben wird, in einen Integer gewandelt. Anschließend wird der aktuelle Wochentag ebenfalls als Integer ermittelt. Diese beiden Integer-Werte werden am Schluss miteinander verglichen, und das Ergebnis wird zurückgeliefert. Abhängig davon, ob der heutige mit dem angegebenen Tag übereinstimmt, wird der Inhalt des entsprechenden Elements `<map:when/>` ausgeführt oder nicht. Innerhalb von `<map:when/>` kann angegeben werden, welche Aktionen erfolgen sollen, wenn der betreffende Tag auftritt.

Um Ihren Selector testen zu können, erstellen Sie am besten ein Verzeichnis `myselector` innerhalb von `$COCOON_HOME`. Erzeugen Sie darin die Datei `weekend.xml`, die nur an Wochenenden ausgegeben werden soll:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<text>
   Heute ist Wochenende und wird nix gemacht;-)
</text>
```

Listing 10.60 Die Datei weekend.xml

Die Datei `standard.xml` hingegen soll genau dann angezeigt werden, wenn es sich nicht um einen der Tage Samstag oder Sonntag handelt.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<text>
   Heute ist ein Standard-Tag.
</text>
```

Listing 10.61 Die Datei standard.xml

Nachdem Sie die XML-Dateien erzeugt haben, ist noch eine Sub-Sitemap erforderlich, die den Selector registriert und anschließend verwendet. Durch das Element `<map:when/>` wird entschieden, welche Aktion an einem bestimmten Tag ausgeführt werden soll. In dieser Sitemap werden lediglich die beiden Tage Samstag und Sonntag speziell behandelt. Sie können aber auch zwischen allen weiteren Wochentagen Unterscheidungen treffen und entsprechend reagieren. Erzeugen Sie also innerhalb von `$COCOON_HOME/myselector` eine Datei `sitemap.xmap` und fügen Sie zu ihr den Inhalt aus dem folgenden Listing hinzu.

```
<?xml version="1.0" encoding="UTF-8"?>
<map:sitemap
   xmlns:map="http://apache.org/cocoon/sitemap/1.0">

<map:components>

   <!-- Selector registrieren -->
   <map:selectors default="weekday">
      <map:selector name="weekday"
       src="foo.bar.cocoon.selection.WeekDaySelector"/>
   </map:selectors>

</map:components>

<map:pipelines>
   <map:pipeline>

   <map:match pattern="">

      <!-- Selector verwenden -->
      <map:select name="weekday">

         <!-- Samstag -->
         <map:when test="7">
            <map:read type="resource"
               src="weekend.xml"/>
         </map:when>

         <!-- Sonntag -->
         <map:when test="1">
            <map:read type="resource"
               src="weekend.xml"/>
         </map:when>

         <!-- Alle uebrigen Tage -->
         <map:otherwise>
            <map:read type="resource"
               src="standard.xml"/>
         </map:otherwise>

      </map:select>
```

```
    </map:match>
    </map:pipeline>

</map:pipelines>
</map:sitemap>
```

Listing 10.62 Die Sub-Sitemap sitemap.xmap

Nachdem Sie dann Ihre Klasse `WeekDaySelector.java` kompiliert und Tomcat eventuell neu gestartet haben, können Sie durch einen Aufruf der Adresse

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

überprüfen, ob Ihr Selector funktioniert. Auch hier soll angemerkt sein, dass Sie die anderen Tage testen können, indem Sie einfach die Datumseinstellung Ihres Betriebssystems entsprechend verändern. Zusätzlich können Sie beispielsweise weitere Elemente `<map:when/>` einfügen und so für jeden Wochentag eine spezielle Ausgabe erzeugen.

10.4.8 Eigenen Matcher erstellen

Die Erstellung eines Matchers ist in der Regel zwar äußerst selten notwendig, soll in diesem Abschnitt der Vollständigkeit halber aber trotzdem erläutert werden.

Ein Matcher kann in etwa wie ein erweiterter Selector verstanden werden. Um mit dieser Analogie zu sprechen: Die Elemente `<map:when/>` entsprechen dabei `<map:match/>`, und das Attribut `pattern` eines Matchers entspricht dem Attribut `test` bei einem Selector. Der Inhalt des Elements `<map:match/>` wird nur dann ausgeführt, wenn eine bestimmte Bedingung erfüllt ist.

Jeder Matcher muss das Interface `Matcher` implementieren, das im folgenden Listing gezeigt wird.

```
package org.apache.cocoon.matching;

import org.apache.avalon.framework.component.Component;
import org.apache.avalon.framework.parameters.Parameters;
import org.apache.cocoon.sitemap.PatternException;

import java.util.Map;
```

```
public interface Matcher extends Component {

   String ROLE = Matcher.class.getName();

   Map match (String pattern, Map objectModel,
      Parameters parameters) throws PatternException;
}
```

Listing 10.63 Das Interface Matcher

Das Interface `Matcher` definiert lediglich die Methode `match`, die durch den Cocoon-Prozessor aufgerufen wird. Als Parameter werden der Wert des Attributs `pattern` von `<map:match/>`, das Object-Model, sowie eventuelle Sitemap-Parameter übergeben. Anders als bei einem Selector liefert diese Methode jedoch keinen booleschen Wert, sondern ein Objekt vom Typ `Map` zurück. Genau wie bei den Actions kann diese Map beliebig viele Name-Wert-Paare enthalten, die innerhalb von `<map:match/>` als Sitemap-Variablen verwendet werden können. Wird stattdessen `null` zurückgeliefert, wird der Rumpf von `<map:match/>` nicht ausgeführt.

Das Attribut `pattern` enthält in der Regel einen Wildcard- oder Regulären Ausdruck, der entsprechend interpretiert wird. Speziell für diese beiden Arten existieren die abstrakten Basisklassen `AbstractRegexpMatcher` und `AbstractWildcardMatcher`, die Ihnen einige Arbeit abnehmen können.

Ein kleines Beispiel

Für dieses Beispiel soll ein Matcher erstellt werden, der dieselbe Aufgabe besitzt wie im vorangegangenen Beispiel der Selector. Die einzelnen Match-Bereiche können dabei auf einen bestimmten Wochentag reagieren. Als Basis für dieses Beispiel werden wieder die beiden Dateien `standard.xml` und `weekend.xml` aus Abschnitt 10.4.7 im Verzeichnis der Subapplikation benötigt.

Erstellen Sie zunächst in `$COCOON_HOME/WEB-INF/classes` die Packagestruktur `foo.bar.cocoon.matching`, legen Sie anschließend innerhalb von `matching` eine Java-Datei mit dem Namen `WeekDayMatcher.java` an und fügen Sie folgende Zeilen hinzu:

```
package foo.bar.cocoon.matching;

import org.apache.cocoon.matching.Matcher;
```

```
import org.apache.cocoon.sitemap.PatternException;
import org.apache.avalon.framework.parameters.↵
Parameters;
import org.apache.avalon.framework.thread.ThreadSafe;

import java.util.Map;
import java.util.Calendar;
import java.util.HashMap;

public class WeekDayMatcher implements Matcher,
   ThreadSafe {

   public Map match(String pattern, Map map,
      Parameters parameters) throws PatternException {

      int patternInt = Integer.parseInt(pattern);

      // Aktuelles Datum ermitteln
      Calendar now = Calendar.getInstance();

      // Aktuellen Wochentag ermitteln
      int nowDay = now.get(Calendar.DAY_OF_WEEK);

      if(patternInt == nowDay || patternInt == 0)
          return new HashMap();

      return null;
    }
}
```

Listing 10.64 Die Java-Klasse WeekDayMatcher.java

Innerhalb der Methode `match` wird diesmal der Wert von `pattern` in einen Integer verwandelt. Anschließend wird der aktuelle Wochentag ermittelt und mit dem Wert aus `pattern` verglichen. Nur wenn beide Werte übereinstimmen oder `pattern` den Wert `0` besitzt, wird eine leere Map zurückgeliefert und somit der entsprechende Match-Bereich ausgeführt.

Fügen Sie anschließend noch die folgende Sub-Sitemap unter dem Namen `sitemap.xmap` in das Verzeichnis `mymatcher` ein, um den Matcher zu registrieren und zu verwenden.

```
<?xml version="1.0" encoding="UTF-8"?>
<map:sitemap
   xmlns:map="http://apache.org/cocoon/sitemap/1.0">

<map:components>

   <!-- Matcher registrieren -->
   <map:matchers default="weekday">
      <map:matcher name="weekday"
       src="foo.bar.cocoon.matching.WeekDayMatcher"/>
   </map:matchers>

</map:components>

<map:pipelines>
   <map:pipeline>

   <!-- Matcher verwenden -->

   <!-- Samstag -->
   <map:match type="weekday" pattern="7">
      <map:read type="resource"
         src="weekend.xml"/>
   </map:match>

   <!-- Sonntag -->
   <map:match type="weekday" pattern="1">
      <map:read type="resource"
         src="weekend.xml"/>
   </map:match>

   <!-- Alle uebrigen Tage -->
   <map:match type="weekday" pattern="0">
      <map:read type="resource"
         src="standard.xml"/>
   </map:match>

   </map:pipeline>

</map:pipelines>
</map:sitemap>
```

Listing 10.65 Die Sub-Sitemap sitemap.xmap von mymatcher

Beachten Sie im vorangegangenen Listing vor allem, dass durch das Attribut `type` von `<map:match/>` explizit ein bestimmter Matcher angegeben wird.

Um den Matcher in Aktion zu sehen, kompilieren Sie die Klasse `WeekDayMatcher.java`, starten Sie eventuell Tomcat neu und rufen Sie folgende URI auf:

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

10.4.9 Eigenen Reader erstellen

Reader sind Sitemap-Komponenten, die eine Ressource von einem beliebigen Ort lesen und an den Client übertragen können. Dabei wird eine solche Ressource in der Regel binär gelesen und an den Client ausgegeben.

Jeder Reader muss das Interface `Reader` implementieren. Dieses Interface ist im folgenden Listing dargestellt.


```
package org.apache.cocoon.reading;

import org.apache.cocoon.ProcessingException;
import org.apache.cocoon.sitemap.SitemapModelComponent;
import org.apache.cocoon.sitemap.SitemapOutputComponent;
import org.xml.sax.SAXException;

import java.io.IOException;

   public interface Reader extends
      SitemapModelComponent, SitemapOutputComponent {

      String ROLE = Reader.class.getName();

      void generate() throws IOException, SAXException,
         ProcessingException;

      long getLastModified();
   }
}
```

Listing 10.66 Das Interface Reader

Da ein Reader eine Sitemap-Komponente darstellt, die durch die Methode `setup` zusätzliche Werte erwartet, ist das Interface `Reader` vom

weiteren Interface `SitemapModelComponent` abgeleitet. Zusätzlich benötigt ein Reader Zugriff auf den Output-Stream von Cocoon und ist deshalb auch noch von `SitemapOutputComponent` abgeleitet. Weiter gehende Informationen zu diesen beiden Interfaces finden Sie in den Abschnitten 10.4.1, *Setup von Sitemap-Komponenten*, und 10.4.2, *Zugriff auf den Output-Stream*.

Nachdem Sie einen Reader in eine Sitemap zur Verwendung eingebunden und durch einen Aufruf gestartet haben, arbeitet der Cocoon-Prozessor alle Methoden, die ein Reader implementieren muss, immer in einer bestimmten Reihenfolge ab. Sehen Sie sich hierzu nachfolgende Abbildung an.

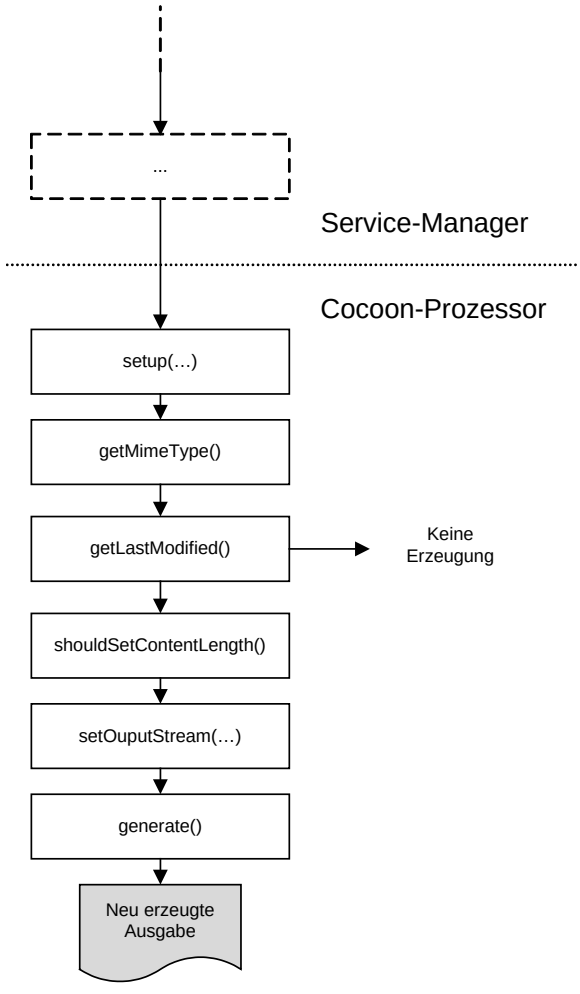


Abbildung 10.2 Die Reihenfolge der Methoden-Aufrufe eines Readers

Zunächst wird eine Reader-Instanz durch den Service-Manager an den Cocoon-Prozessor übergeben. Anschließend führt dieser zunächst die Methode `setup` aus, um dem Reader alle zusätzlich notwendigen Parameter zu übergeben. Dazu zählen z.B. Konfigurationsparameter oder der SourceResolver. Danach wird durch den Aufruf der Methode `getMimeType` versucht, den Mime-Type der Ausgabe zu erhalten. Falls der Mime-Type nicht durch den Reader bestimmt werden kann, wird versucht, das in der Sitemap für einen Reader angegebene Attribut `mime-type` auszulesen. Als Nächstes wird `getLastModified` aufgerufen. Durch diese Methode wird letztendlich bestimmt, ob die angeforderte Ausgabe neu erzeugt werden muss oder bereits beim Client in aktueller Form existiert. Falls die Ausgabe neu erzeugt werden muss, werden drei weitere Methoden aufgerufen. Zuerst wird durch den Rückgabewert von `shouldSetContentLength` ermittelt, ob die Information über die Länge der Ausgabe erforderlich ist und im Response gesetzt werden soll. Diese Eigenschaft wird deshalb zuvor abgefragt, da es relativ zeitintensiv sein kann, diese Länge zu ermitteln. Einige Formate (z.B. PDF) benötigen diese Angabe jedoch unbedingt. Durch die Methode `setOutputStream` wird als Parameter der aktuelle OutputStream an den Reader übergeben, der diesen anschließend innerhalb der Methode `generate` verwenden kann, um die Ausgabewerte darin zu platzieren. Die Methode `generate` wird als Letzte nach `setOutputStream` durch den Cocoon-Prozessor aufgerufen.

AbstractReader

Natürlich existiert auch für einen Reader eine abstrakte Basisklasse, die bereits alle notwendigen Methoden mit Standardwerten implementiert hat. Lediglich die Methode `generate` müssen Sie in diesem Fall noch erstellen. Die Rede ist dabei von der Klasse `AbstractReader`. Neben dem Interface `Reader` implementiert diese Klasse zusätzlich auch das Interface `Recyclable`, um den Reader »poolable« zu machen. Innerhalb der Methode `recycle` werden alle verwendeten Ressourcen wieder freigegeben. Da der `AbtractReader` auch noch von der Klasse `AbstractLogEnabled` abgeleitet ist, kann darin direkt auf die Instanz-Variable `m_logger` für Log-Ausgaben zugegriffen werden. In Listing 10.67 ist die abstrakte Basisklasse `AbstractReader` angegeben.

```
package org.apache.cocoon.reading;

import org.apache.avalon.excalibur.pool.Recyclable;
import org.apache.avalon.framework.logger.↵
AbstractLogEnabled;
import org.apache.avalon.framework.parameters.Parameters;
```

```
import org.apache.cocoon.ProcessingException;
import org.apache.cocoon.environment.SourceResolver;
import org.xml.sax.SAXException;

import java.io.BufferedOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.util.Map;

public abstract class AbstractReader
   extends AbstractLogEnabled
   implements Reader, Recyclable {

   protected SourceResolver resolver;
   protected Map objectModel;
   protected Parameters parameters;
   protected String source;
   protected OutputStream out;

   public void setup(SourceResolver resolver,
      Map objectModel, String src, Parameters par)
      throws ProcessingException, SAXException,
      IOException {

      this.resolver=resolver;
      this.objectModel=objectModel;
      this.source=src;
      this.parameters=par;
   }

   public void setOutputStream(OutputStream out) {
      this.out = new BufferedOutputStream(out);
   }

   public String getMimeType() {
      return null;
   }

   public long getLastModified() {
      return 0;
   }
```

```
    public void recycle() {
        this.out = null;
        this.resolver = null;
        this.source = null;
        this.parameters = null;
        this.objectModel = null;
    }

    public boolean shouldSetContentLength() {
        return false;
    }
}
```

Listing 10.67 Die Klasse AbstractReader

Ein kleines Beispiel

Als Beispiel für die Erstellung eines Readers wollen wir einen Resource-Reader implementieren, der eine beliebige Ressource über den ReSourceResolver lädt und in den Ausgabe-Stream schreibt. Ein solcher Resource-Reader existiert in umfangreicherer Form natürlich bereits schon in Cocoon. Für dieses Beispiel soll es deshalb genügen, eine eigene einfachere Variante davon zu erstellen, um lediglich das grundsätzliche Verständnis für die Implementierung dieser Komponente zu erhalten.

Erstellen Sie als Erstes innerhalb von `$COCOON_HOME/WEB-INF/classes` die Verzeichnisstruktur `foo.bar.cocoon.reading` und erzeugen Sie innerhalb von `reading` eine Java-Klasse mit dem Namen `MyResourceReader.java`, die folgenden Inhalt besitzen sollte:

```
package foo.bar.cocoon.reading;

import org.apache.cocoon.reading.AbstractReader;
import org.apache.cocoon.ProcessingException;
import org.apache.excalibur.source.Source;

import java.io.IOException;
import org.xml.sax.SAXException;
import java.io.InputStream;

public class MyResourceReader extends AbstractReader {

    public void generate() throws IOException,
```

```
        SAXException, ProcessingException {

        // Ressource laden
        Source source =
           super.resolver.resolveURI(super.source);

        // Input-Stream der Ressource holen
        InputStream inputStream =
           source.getInputStream();

        byte[] buffer = new byte[8192];
        int length = -1;

        // Input-Stream an Output-Stream anhaengen
        while(inputStream.available() > 0) {

           length = inputStream.read(buffer, 0, 8129);

           if(length > 0) {

              this.out.write(buffer, 0, length);
           }
        }

        // Fertig: Aufrauemen
        this.out.flush();
        inputStream.close();
        super.resolver.release(source);
     }
}
```

Listing 10.68 MyResourceReader.java

Wie Sie sehen, ist der Reader von `AbstractReader` abgeleitet und implementiert deshalb lediglich die Methode `generate`. Innerhalb dieser Methode wird zunächst die Ressource geladen und deren Input-Stream erzeugt. Anschließend wird dieser Input-Stream in den Output-Stream geschrieben, der durch den Cocoon-Prozessor zur Verfügung gestellt wird.

Je nachdem, welche zusätzlichen Funktionalitäten Sie in Ihrem Reader benötigen, können Sie die jeweiligen Methoden aus dem `AbstractReader` überschreiben. Dadurch kann Ihr Reader beliebig komplex

werden. Für dieses Beispiel wollen wir es jedoch bei dieser einfachsten Form belassen.

Um zu testen, ob Ihr Reader auch wirklich funktioniert, erstellen Sie zunächst wieder eine Subapplikation in Form eines Unterverzeichnisses innerhalb von `$COCOON_HOME` unter dem Namen `myreader` und fügen Sie darin eine beliebige XML-Datei mit dem Namen `file.xml` ein. Zu guter Letzt müssen Sie noch eine Sub-Sitemap erzeugen, die den Reader registriert und verwendet. Erstellen Sie hierfür eine Datei `sitemap.xmap` im Verzeichnis `myreader` mit folgendem Inhalt:

```
<?xml version="1.0" encoding="UTF-8"?>
<map:sitemap
   xmlns:map="http://apache.org/cocoon/sitemap/1.0">

<map:components>

   <!-- Reader registrieren -->
   <map:readers default="myReader">
      <map:reader name="myReader"
        src="foo.bar.cocoon.reading.MyResourceReader"/>
   </map:readers>

</map:components>

<map:pipelines>
   <map:pipeline>

      <!-- Reader verwenden -->
      <map:match pattern="">
         <map:read type="myReader"
            src="file.xml"/>
      </map:match>

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

</map:sitemap>
```

Listing 10.69 Die Sub-Sitemap sitemap.xmap

Nachdem Sie die Klasse `MyResourceReader.java` kompiliert und Tomcat eventuell neu gestartet haben, können Sie durch einen Aufruf von

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

testen, ob Ihr Reader auch wirklich funktioniert. Als Ausgabe sollte in Ihrem Webbrowser der Inhalt der Datei `file.xml` wiedergegeben werden.

10.4.10 Eigenen Serializer erstellen

Als letzte Komponente wollen wir uns ansehen, wie ein Serializer implementiert werden kann. Ein Serializer wandelt dabei die SAX-Events innerhalb einer Pipeline in ein Format, das vom Empfänger verstanden wird.

Da Sie aller Wahrscheinlichkeit nach relativ selten in die Situation kommen werden, selbst einen Serializer zu schreiben, wollen wir diesen Abschnitt auf das Wesentliche beschränken.

Jeder Serializer muss das Interface `Serializer` implementieren, das im folgenden Listing dargestellt ist.

```
package org.apache.cocoon.serialization;

import org.apache.cocoon.sitemap.SitemapOutputComponent;
import org.apache.cocoon.xml.XMLConsumer;

public interface Serializer extends XMLConsumer,
   SitemapOutputComponent {

   String ROLE = Serializer.class.getName();
}
```

Listing 10.70 Das Interface Serializer

Aus dem Interface `Serializer` ist zu erkennen, dass ein Serializer ein XML-Consumer ist, also SAX-Events empfangen kann. Daneben wird durch die Implementierung `SitemapOutputComponent` angegeben, dass ein Serializer zusätzlich den Output-Stream, der an den Client gesendet wird, verarbeitet.

Wie für alle Komponenten existiert natürlich auch für einen Serializer eine abstrakte Basisklasse `AbstractSerializer`, die die wichtigsten Implementierungen vornimmt und später bei Bedarf nur überschrieben bzw. erweitert werden muss.

Ein kleines Beispiel

In diesem Beispiel wird ein äußerst einfacher XML-Serializer erstellt, der den SAX-Stream aus der Pipeline entgegennimmt und als XML für den Client zur Verfügung stellt. Bitte beachten Sie, dass dieser Serializer sehr einfach gehalten wurde und aus diesem Grund verschiedene Funktionalitäten, wie z.B. das Behandeln von Namensräumen, außen vor lässt. Falls Sie im produktiven Einsatz einen Serializer, der einen Zeichenstrom anstatt eines binären Formats an den Client senden soll, benötigen, können Sie die abstrakte Basisklasse `AbstractText-Serializer` verwenden, die die wichtigsten Algorithmen hierfür bereits vorimplementiert hat.

Erstellen Sie im Verzeichnis `$COCOON_HOME/WEB-INF/classes` zunächst die Packagestruktur `foo.bar.cocoon.serialization` und legen Sie die in Listing 10.71 gezeigte Klasse `MyXMLSerializer.java` im Verzeichnis `serialization` ab.


```
package foo.bar.cocoon.serialization;

import org.apache.cocoon.serialization.↵
AbstractSerializer;

import java.io.OutputStream;
import java.io.IOException;

import javax.xml.transform.sax.TransformerHandler;
import javax.xml.transform.stream.StreamResult;
import org.apache.cocoon.CascadingIOException;

import javax.xml.transform.sax.SAXTransformerFactory;
import javax.xml.transform.TransformerFactory;

public class MyXMLSerializer
   extends AbstractSerializer {

   public void setOutputStream(OutputStream out)
      throws IOException {

      super.setOutputStream(out);

      try {
```

```
            SAXTransformerFactory factory =
            (SAXTransformerFactory)TransformerFactory.↵
              newInstance();
              TransformerHandler handler =
                 factory.newTransformerHandler();

              handler.setResult(new
                 StreamResult(this.output));

              // Den Handler zum Senden setzen
              this.setContentHandler(handler);

           } catch (Exception ex) {
              throw new CascadingIOException(
                 "Error setting OutputStream!", ex);
           }
        }

        public String getMimeType() {

           // Den Mime-Type festlegen
           return "text/xml";
        }
}
```

Listing 10.71 Die Klasse MyXMLSerializer.java

Um diesen Serializer zu testen, erstellen Sie wieder eine Subapplikation `myserializer` innerhalb von `$COCOON_HOME`, indem Sie darin ein Verzeichnis `myserializer` anlegen und darin wiederum eine einfache XML-Datei `myfile.xml` ohne Namensraumdefinitionen platzieren, wie z. B. im Folgenden gezeigt:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<document>
   <text>Ein Text</text>
</document>
```

Listing 10.72 Die XML-Datei myfile.xml

Als letzten Punkt müssen Sie noch eine Sub-Sitemap erstellen, in der der Serializer registriert und verwendet wird. Erstellen Sie hierfür unter `$COCOON_HOME/myserializer` die Datei `sitemap.xmap` und fügen Sie folgende Zeilen hinzu:

```
<?xml version="1.0" encoding="UTF-8"?>
<map:sitemap xmlns:map="http://apache.org/cocoon/↵
sitemap/1.0">

<map:components>

   <!-- Serializer registrieren -->
   <map:serializers default="myXMLSerializer">
      <map:serializer name="myXMLSerializer"
      src="foo.bar.cocoon.serialization.↵
      MyXMLSerializer"/>
   </map:serializers>
</map:components>

<map:pipelines>
   <map:pipeline>

   <map:match pattern="">
      <map:generate type="file"
         src="myfile.xml"/>
      <!-- Serializer verwenden -->
      <map:serialize type="myXMLSerializer"/>
   </map:match>

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

Listing 10.73 Die Sub-Sitemap von myserializer

Nachdem Sie die Klasse und die zugehörigen Dateien erstellt und Tomcat eventuell neu gestartet haben, können Sie über die URI

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

nachsehen, ob der Serializer auch ordnungsgemäß funktioniert. Dies ist dann der Fall, wenn die Datei `myfile.xml` aus Listing 10.72 in Ihrem Browser bzw. im Quelltext der Seite erscheint.

10.4.11 Caching von Sitemap-Komponenten

Cocoon besitzt einen genauso effektiven wie umfangreichen Cache-Mechanismus, um die Systemlast und die Antwortzeiten so gering wie

möglich zu halten. Was aber genau unter der Motorhaube passiert, wenn Caching verwendet wird, wollen wir uns in diesem Abschnitt etwas näher ansehen.

Wie funktioniert das Caching?

Innerhalb einer Pipeline werden der Generator und alle nachfolgenden Sitemap-Komponenten zusammengefasst, und das Resultat, das diese Komponenten produzieren, wird als so genannter **Cached-Response** gespeichert. Falls innerhalb einer solchen Pipeline jedoch nur eine einzige Komponente vorkommt, die nicht cachebar ist, wird lediglich das Resultat derjenigen Sitemap-Komponenten als Cached-Response gespeichert, die alle vom Beginn an zusammenhängend vor der ersten nicht cachebaren Komponente liegen. Auch wenn nach einer nicht cachebaren Komponente weitere Komponenten folgen, die wieder cachebar sind, so kann trotzdem kein weiterer Cached-Response hiervon erzeugt werden, da alle nachfolgenden Komponenten auf der Ausgabe der nicht cachebaren Komponente basieren. Sehen Sie sich hierzu Abbildung 10.3 an. Dort ist als Beispiel eine mögliche Pipeline dargestellt, die aus vielen cachebaren und einer einzigen nicht cachebaren Komponente besteht. Die Kernaussage ist die, dass der cachebare Pipeline-Bereich lediglich bis zum Resultat des ersten TraxTransformers reicht. Das Resultat dieser Komponente wird im Cached-Response abgelegt. Bei einem erneuten Aufruf dieser Pipeline wird dieses Resultat aus dem Cached-Response geladen und erst ab hier mit der Verarbeitung der Pipeline begonnen. Im Beispiel in Abbildung 10.3 wird also die Ausführung der beiden ersten Komponenten FileGenerator und TraxTransformer bei einem Folgeaufruf eingespart.

Behalten Sie Abbildung 10.3 immer im Kopf, wenn Sie auf cachefähige Komponenten innerhalb einer Pipeline angewiesen sind. Denn wie bereits erwähnt: Sobald eine einzige Komponente innerhalb dieser Pipeline auftritt, die nicht cachebar ist, kann die Cachefähigkeit aller nachfolgenden Komponenten nicht mehr ausgenützt werden.

Der aus der Sicht des Cachings idealste Fall ist natürlich, wenn alle Komponenten innerhalb einer Pipeline das Caching voll unterstützen. Dann kann der Cached-Response direkt an die Ausgabe weitergeleitet werden, ohne auch nur eine einzige Pipeline-Komponente ausführen zu müssen.

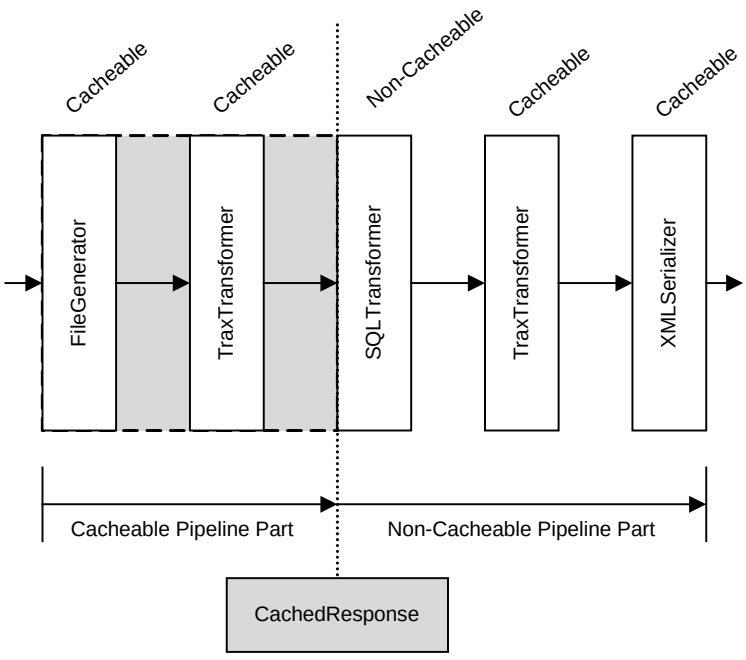


Abbildung 10.3 Unterteilung einer Pipeline in einen cachebaren und einen nicht cachebaren Bereich

Doch wann ist eine Pipeline-Komponente eigentlich cachefähig? Grob gesagt müssen vier konkrete Dinge erfüllt sein, damit eine einzelne Komponente cachefähig ist:

- Das Interface `CacheableProcessingComponent` muss durch die Komponente implementiert werden.
- Die Methode `getKey` muss einen, von `null` verschiedenen, eindeutigen Schlüssel zurückliefern.
- Die Methode `getValidity` muss ein Objekt vom Typ `SourceValidity` zurückliefern anstatt des Werts `null`.
- Das Source-Validity-Objekt muss gültig sein.

Innerhalb einer Pipeline werden alle Komponenten der Reihe nach abgefragt, ob sie Caching unterstützen. Dies wird durch das Implementieren des Interfaces `CacheableProcessingComponent` bestimmt, das in Listing 10.74 gezeigt wird. Diese Abfrage wird solange durchgeführt, bis alle Komponenten der Pipeline überprüft wurden oder auf eine Komponente gestoßen wird, die dieses Interface nicht implementiert und somit nicht cachebar ist.

```
package org.apache.cocoon.caching;

import org.apache.excalibur.source.SourceValidity;
import java.io.Serializable;

public interface CacheableProcessingComponent {

   Serializable getKey();
   SourceValidity getValidity();
}
```

Listing 10.74 Das Interface CacheableProcessingComponent

getKey Von allen hintereinander vorkommenden, cachebaren Pipeline-Komponenten wird die Methode `getKey` aufgerufen, die ein beliebiges, serialisierbares Objekt zurückliefert, das als eindeutiger Schlüssel für jede einzelne Komponente dient. Liefert diese Methode anstatt eines Schlüssels jedoch den Wert `null`, so hat dies denselben Effekt, als wenn das Interface `CacheableProcessingComponent` nicht implementiert und somit diese Komponente auch nicht cacheable ist.

Als Werte, um einen Schlüssel zu generieren, verwendet man in der Regel die URI, über die eine Ressource aufgerufen wird (z. B. der Wert des Attributs `src`) und zusätzlich eventuell vorhandene Konfigurationsparameter. Diese Werte können Sie anschließend beispielsweise in einen `StringBuffer` schreiben und zurückliefern. Der Schlüssel muss dabei nur innerhalb der entsprechenden Komponente eindeutig sein. Ein Generator und ein Transformer dürfen also ruhig den gleichen Schlüssel besitzen.

Als Nächstes geht es nun darum, festzustellen, ob

1. ein Cached-Response-Objekt mit dem angegebenen Schlüssel bereits besteht und
2. das Source-Validity-Objekt existiert und gültig ist.

Der Caching-Algorithmus sieht zunächst nach, ob ein Cached-Response-Objekt unter dem angegebenen Schlüssel bereits vorhanden ist. Ist dies nicht der Fall, wird die Cacheable-Pipeline ausgeführt, das Resultat zusammen mit den aktuellen Validity-Objekten im Cached-Response-Objekt für den nächsten Aufruf zwischengespeichert und der Caching-Algorithmus beendet. Andernfalls wird das Cached-Response-Objekt unter dem angegebenen Schlüssel geladen.

getValidity

Jede einzelne Pipeline-Komponente besitzt die Methode `getValidity`, über die ein einzelnes Objekt vom Typ `SourceValidity` zurückgeliefert wird. Wird an dieser Stelle der Wert `null` zurückgeliefert, wird diese Komponente wiederum als nicht cachebare Komponente deklariert und folgt denselben Regeln, die für Komponenten gelten, die nicht cachebar sind. Falls jedoch ein Source-Validity-Objekt zurückgeliefert wird, überprüft dieses Objekt innerhalb der Methode `isValid`, ob der Zustand der Komponente gültig und somit aktuell genug ist. Dies kann auf zwei verschiedene Wege geschehen. Entweder erfolgt diese Entscheidung innerhalb der parameterlosen Methode `isValid` oder ihr wird das zugehörige Validity-Source-Objekt der letzten Anfrage aus dem Cached-Response-Objekt übergeben. Somit kann z. B. auf Gleichheit hin geprüft werden. Eine Möglichkeit, die Gültigkeit eines Validity-Objekts zu bestimmen, wäre zum Beispiel das Vergleichen des Datums der letzten Änderung einer Datei, die von einer Komponente gelesen wird. Die wesentliche Aufgabe dieses Schritts besteht einzig und allein darin zu garantieren, dass sich die Daten, die die Komponente benötigt, seit dem letzten Aufruf nicht geändert haben.

Falls die Überprüfung der Source-Validity-Objekte der einzelnen Komponenten ergibt, dass mindestens ein solches Objekt ungültig ist, wird die gesamte cachebare Pipeline ausgeführt und das Resultat zusammen mit den einzelnen Source-Validity-Objekten im Cached-Response-Objekt für den nächsten Aufruf unter dem aktuellen Schlüssel gespeichert. Andernfalls wird das Resultat der letzten Anfrage aus dem Cache geholt und in die Pipeline »gelegt«. Komponenten, die nach dem cachebaren Teil der Pipeline folgen, können dieses Resultat anschließend weiterverarbeiten.

12 Formular-bearbeitung

12 Formularbearbeitung

In diesem Kapitel erfahren Sie, wie Cocoon Sie dabei unterstützen kann, Formulare zu bearbeiten.

Die Bereitstellung und Bearbeitung von Formularen und den zugehörigen Daten stellt in der Web-Programmierung immer wieder eine neue Herausforderung dar. Es gibt bereits viele gute Lösungsansätze, die den Ablauf einer solchen Verarbeitung strukturieren. Solche Ansätze sollten in der Regel mindestens drei konkrete Aufgaben erfüllen:

- Kapselung der Fomulardaten in einem Model
- Automatisches Vorselektieren der Formularwerte
- Rückgabe der Benutzereingaben in Form eines Models
- Validierung der Benutzereingaben

Model

Mit einem Model ist in diesem Fall eine Datenstruktur gemeint, die Informationen in einer bestimmten Art und Weise kapselt. Dies kann ein Objekt (z. B. eine JavaBean) oder ein anderes strukturiertes Element (z. B. ein XML-Dokument) sein.

Cocoon besitzt zur Zeit zwei verschiedene Form-Frameworks, die den Umgang mit Web-Formularen vereinfachen und den Entwicklungsablauf standardisieren. In diesem Kapitel wollen wir uns diese näher ansehen und anhand einfacher Beispiele ihren Einsatz verdeutlichen.

12.1 XMLForm

Im Alpha- und Beta-Stadium von Cocoon 2.1 wurde ein Formular-Framework namens XMLForm zur Verfügung gestellt. Dieses Framework wurde aber kurz vor dem Release als »deprecated« eingestuft und sollte deshalb nicht mehr verwendet werden. Das Framework, das XMLForm heute am ehesten entspricht, ist JXForms.

12.2 JXForms

JXForms kann ausschließlich im Zusammenspiel mit Flowscripts eingesetzt werden. Da es für den Zugriff auf ein Model das Apache-Paket JXPath verwendet, ist eine Bindung mit verschiedenen Model-Arten mög-

lich. Dazu zählen vor allem JavaBeans[1], XML/DOM[2], JDOM[3], DynaBeans[4] und JavaScript-Objekte.

XForms

Die Definition der Formulare erfolgt durch eine Syntax, die an XForms[5] angelehnt ist. XForms ist ein XML-Standard, der durch das W3C beschlossen wurde und (Web-)Formulare beschreibt. Diese Beschreibung ist unabhängig von der verwendeten Plattform und erlaubt eine Trennung von Aussehen und Funktionalität eines Formulars. JXForms unterstützt aber lediglich diejenigen Elemente aus W3C-XForms, die innerhalb eines HTML-Formulars benötigt werden. Teilweise kommen diese Elemente auch in etwas erweiterter Form zum Einsatz.

```
<form action="form.html">
   Vorname: <br/>
   <input type="text" name="vorname"/> <br/>
   Nachname: <br/>
   <input type="text" name="nachname"/><br/>
   Fahrzeug: <br/>
   <select name="fahrzeug">
      <option value="BMW">BMW</option>
      <option value="Mercedes">Mercedes</option>
      <option value="Audi">Audi</option>
   </select>
   <br/><br/>
   <input type="submit" value="Senden"/>
</form>
```

Listing 12.1 Ein gewöhnliches HTML-Formular

In Listing 12.1 sehen Sie ein übliches HTML-Formular, das Textfelder und ein Select-Element besitzt. Um JXForms verwenden zu können, muss dieses HTML-Formular in einer etwas anderen Art per XML beschrieben werden. Eine solche Beschreibung zeigt Listing 12.2. Es handelt sich also in beiden Fällen um eine Formulardefinition, nur mit dem Unterschied, dass im ersten Fall diese Definition per (X)HTML und im zweiten Fall per JXForms erstellt wurde.

```
<jxf:form id="myForm" action="index.html">
   <jxf:input ref="/vorname">
```

1 http://java.sun.com/products/javabeans/
2 http://www.w3.org/DOM/
3 http://www.jdom.org/
4 http://jakarta.apache.org/commons/beanutils/api/org/apache/commons/beanutils/
5 http://www.w3c.org/MarkUp/Forms/

```
      <jxf:label>Vorname:</jxf:label>
   </jxf:input>
   <br/>
   <jxf:input ref="/nachname">
      <jxf:label>Nachname:</jxf:label>
   </jxf:input>
   <br/>
   <jxf:select1 ref="/auto">
      <jxf:label>Fahrzeug:</jxf:label>
      <jxf:item>
         <jxf:label>BMW</jxf:label>
         <jxf:value>BMW</jxf:value>
      </jxf:item>
      <jxf:item>
         <jxf:label>Mercedes</jxf:label>
         <jxf:value>Mercedes>/jxf:value>
      </jxf:item>
      <jxf:item>
         <jxf:label>Audi</jxf:label>
         <jxf:value>Audi</jxf:label>
      </jxf:item>
   </jxf:select>
   <br/>
   <jxf:submit>
      <jxf:label>Senden</jxf:label>
   </jxf:submit>
</jxf:form>
```

Listing 12.2 Eine Formulardefinition mit JXForms

Beide Definitionen führen zur identischen Ausgabe eines (X)HTML-Formulars, wobei das durch JXForms definierte Formular zuvor erst durch eine XSL-Transformation in ein (X)HTML-Dokument gewandelt werden muss. Die Ausgabe könnte dann in beiden Fällen etwa so aussehen, wie in der folgenden Abbildung dargestellt.

Abbildung 12.1 Die Ausgabe des Formulars

12.2.1 Datenbindung

Für die Bindung Ihrer Daten (z. B. JavaBean) mit den Formularfeldern sind in der Regel zwei Teile innerhalb von JXForms verantwortlich. Das ist zum einen das Flowscript `JXForms.js`. Dieses Skript extrahiert die Daten, die durch den Benutzer versendet wurden, aus dem Request und setzt sie im Model an die entsprechenden Positionen. Der zweite Teil ist der JXForms Generator bzw. Transformer. Dieser nimmt eine Formulardefinition entgegen und setzt im Formular die Werte aus dem Model.

Um eine Bindung zwischen den einzelnen Formularfeldern und einem Objekt zu erstellen, sind einige wichtige Dinge zu beachten:

- Das XML-Dokument, das die Formulardefinitionen enthält, muss entweder durch den JXForms Generator oder den JXForms Transformer interpretiert werden.
- Alle Formulardaten müssen in einem Objekt gekapselt sein. Dieses Objekt kann vom Typ JavaBean, XML/DOM, JDOM, DynaBean oder JavaScript sein.
- Das Model muss durch `form.setModel` registriert werden.
- Über die Sitemap muss die Flowscript-Funktion `jxforms` aufgerufen werden, die als Parameter die ID des Formulars und den im Anschluss aufzurufenden Funktionsnamen des eigentlichen Flowscripts enthält.
- Das Flowscript `JXForms.js` muss inkludiert werden.
- Ihrer Flowscript-Funktion muss als Parameter ein Form-Objekt übergeben werden können.
- Das Senden der Seite erfolgt mit `form.sendView` anstatt mit `cocoon.sendPageAndWait` oder `cocoon.sendPage`.
- Vor der Serialisierung des Formulars muss durch eine XSL-Transformation eine Umwandlung der JXForms-Elemente in die Zielelemente (z. B. XHTML-Formularelemente) erfolgen.

Innerhalb der Sitemap wird der Aufruf von JXForms konfiguriert. Hierzu werden dem bereits bekannten Aufruf `<map:call function="..."/>` des Flowscripts zwei Parameter mitgegeben: `function` und `id`. Dabei gibt `function` den Namen der Einstiegsfunktion Ihres Flowscripts und `id` den Namen des Formulars an, das verwendet werden soll. Listing 12.3 zeigt, wie ein solcher Aufruf mit Übergabe dieser Parameter erfolgen könnte.

```
...
<map:pipeline>
...
   <map:match pattern="flow">
      <map:call function="jxform">
         <map:parameter name="function" value="main"/>
         <map:parameter name="id" value="myForm"/>
      </map:call>
   </map:match>
...
</map:pipeline>
...
```

Listing 12.3 Aufruf der Flowscript-Funktion »jxforms« mit Parametern

In diesem Beispiel wird die Funktion `jxform` des Flowscripts `JXForms.js` aufgerufen, und über die Parameter `function` und `id` werden ihr die Werte `main` und `myForm` übergeben. Achten Sie darauf, dass hier zuerst die Funktion `jxforms` aufgerufen wird, die zunächst bestimmte Entscheidungen und eventuell Veränderungen am Model-Objekt vornimmt und anschließend Ihre eigene Funktion – in diesem Beispiel `main` – aufruft, die Sie in diesem Fall durch das Element

```
<map:parameter name="function" .../>
```

innerhalb von `<map:call/>` angegeben haben.

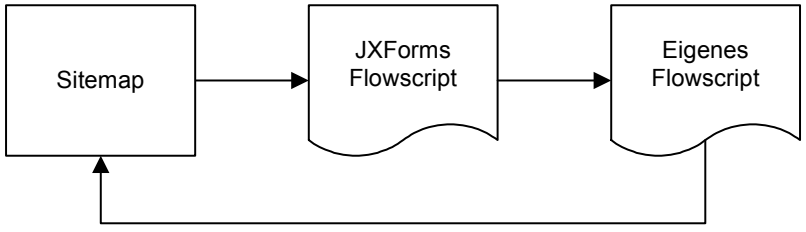


Abbildung 12.2 Reihenfolge der Flowscripts bei Verwendung von JXForms

12.2.2 Ein kleines Beispiel

Um den Einsatz von JXForms zu verdeutlichen, wollen wir an dieser Stelle ein kleines Beispiel realisieren. Ziel ist es, über insgesamt vier Instanzen von Formularen hinweg einen Wizard für das Registrieren auf einer Website zu realisieren. Die einzelnen Formulare werden nacheinander die Adresse, den Kenntnisstand und die Ausbildung der sich registrierenden Person abfragen und diese Informationen anschlie-

ßend auf einer weiteren Seite zur Vorschau anbieten. Nach dem Bestätigen werden die angegebenen Werte fiktiv gespeichert, und der Benutzer kann eine erneute Registrierung durchführen. Eine Validierung erfolgt aus Gründen der Übersichtlichkeit nicht. Abbildung 12.3 illustriert die einzelnen Seiten, die der Benutzer nacheinander angezeigt bekommt. Achten Sie hier vor allem auf die dargestellten Pfeilrichtungen. Diese deuten an, dass zwischen den einzelnen Formularen beliebig vor- und zurücknavigiert werden kann und dies ohne zusätzlichen Programmieraufwand.

Die Sourcen zu diesem Beispiel finden Sie selbstverständlich auch auf der Buch-CD.

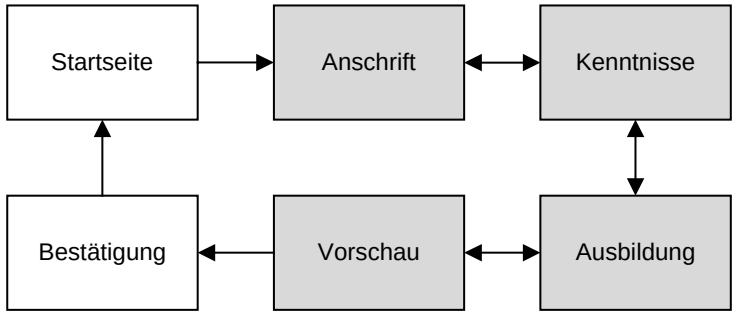


Abbildung 12.3 Seitenablauf des JXForms-Beispiels

Da auch dieses Beispiel als eigenständiges Subprojekt innerhalb von Cocoon realisiert werden soll, müssen Sie hierfür wieder ein Verzeichnis innerhalb von `$COCOON_HOME` erstellen, in das später die Sub-Sitemap gelegt wird. Dieses Verzeichnis soll den Namen `registration` erhalten und die beiden Unterverzeichnisse `documents` und `flows` besitzen. In das Verzeichnis `documents` werden später alle unsere Templates und in `flows` die Flowscripts abgelegt.

Erstellen Sie als Nächstes nacheinander die entsprechenden XML-Dateien für die einzelnen Formulare und die beiden übrigen Seiten. Legen Sie innerhalb von `documents` zunächst die Dateien `adress.xml`, `knowledge.xml`, `education.xml` und `preview.xml` an und fügen Sie die Einträge aus den nachfolgenden Listings hinzu.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<html xmlns:jxf="http://apache.org/cocoon/jxforms/1.0">
<header>
   <title>Registrierung - Ihre Adresse</title>
</header>
```

```
<body>
   <h1>Ihre Adresse</h1>
   Bitte geben Sie hier Ihre Adresse an.
   <jxf:form id="registration" action="./register">
   <table border="1">
      <tr>
         <td>Vorname</td>
         <td><jxf:input ref="firstName"/></td>
      </tr>
      <tr>
         <td>Nachname</td>
         <td><jxf:input ref="lastName"/></td>
      </tr>
      <tr>
         <td>Wohnort</td>
         <td><jxf:input ref="city"/></td>
      </tr>
      <tr>
         <td>Email</td>
         <td><jxf:input ref="email"/></td>
      </tr>
      <tr>
         <td>Geschlecht</td>
         <td>
         <jxf:select1 ref="sex">
            <jxf:item>
               <jxf:label>Weiblich</jxf:label>
               <jxf:value>Weiblich</jxf:value>
            </jxf:item>
            <jxf:item>
               <jxf:label>Männlich</jxf:label>
               <jxf:value>Männlich</jxf:value>
            </jxf:item>
         </jxf:select1>
      </td>
   </tr>
   <tr>
      <td colspan="2">
         <jxf:submit id="next" continuation="forward">
            <jxf:label>Weiter</jxf:label>
         </jxf:submit>
      </td>
```

```
    </tr>
</table>
</jxf:form>

</body>
</html>
```

Listing 12.4 adress.xml

Namensraum

Die Datei `adress.xml` beinhaltet ein Formular, in das der Benutzer seinen Namen und seine Adressdaten eingeben kann. Hierbei handelt es sich im Wesentlichen um ein einfaches (X)HTML-Dokument, in das JXForms-Elemente eingebettet wurden. Wichtig ist hierbei, dass für alle JXForms-Elemente folgender Namensraum verwendet wird, der mit dem Präfix `jxf` gebunden werden sollte:

```
http://apache.org/cocoon/jxforms/1.0
```

<jxf:form/>

Das Element `<jxf:form/>` schließt alle weiteren JXForms-Elemente ein. Mit dem Attribut `id` wird dem Formular ein eindeutiger Name zugewiesen, und `action` gibt das Ziel an, an das die Formulardaten gesendet werden sollen. Hierbei ist es wichtig, dass alle Formularinstanzen dieselbe ID erhalten und auf dieselbe URI verweisen, um die Kontrolle immer an die gleiche Stelle zurückzuübergeben.

<jxf:input/>

Die Elemente `<jxf:input/>` stehen für ein einfaches Textfeld, wobei durch das Attribut `ref` einem solchen Feld eine eindeutige Position per XPath im Model zugewiesen wird. Es wird also hiermit angegeben, wo der Wert für dieses Feld innerhalb eines Models zu finden ist. Fast jedes JXForms-Element enthält dieses Pflichtattribut. Im Beispiel von `<jxf:input ref="firstName"/>` wird in der ersten Ebene nach einem Element `firstName` gesucht und dieses durch den eingegebenen Wert gefüllt bzw. ein gesetzter Wert in das Formular geschrieben. Eine JavaBean müsste beispielsweise demnach folgende Methode besitzen:

```
bean.getFirstName()
```

<jxf:select1/>

Ein weiteres JXForms-Element ist `<jxf:select1/>`. Dieses Element steht für eine Auswahlliste, aus der nur ein einziges Element selektiert werden kann. Jedes Kindelement `<jxf:item/>` steht für einen Eintrag in dieser Liste. Mit `<jxf:name/>` und `<jxf:value/>` wird jedem Eintrag ein Name und ein Wert zugewiesen. Der Name wird in dieser Liste angezeigt, der Wert wird nach dem Selektieren übertragen.

<jxf:submit/>

Das letzte in diesem Formular auftretende JXForms-Element ist `<jxf:submit/>`. Dieses ist für die Vor- und Zurück-Naviagtion zwischen den verschiedenen Formular-Instanzen äußerst wichtig. Welche Funktion ein solcher Submit-Button erfüllt, wird durch das Attribut `continuation` angegeben. Durch den Wert `forward` wird das Flowscript weiter ausgeführt, und mit `back` wird das letzte Continuation-Objekt wieder geladen. Das Attribut `id` ist hier notwendig und weist dem Element einen beliebigen Namen zu, der in dieser Formular-Instanz eindeutig sein sollte.

Wo ist die Continuation-ID?

Falls Sie die vorhergehenden Kapitel über Control Flow genau durchgelesen haben, werden Sie sich sicherlich fragen, wie Cocoon eigentlich weiß, welches Continuation-Objekt es bei einem Versenden des Formulars starten soll. In der Action-Methode erfolgt nämlich kein Verweis auf eine Continuation-ID, sondern auf eine relative URI. JXForms übermittelt diese Continuation-ID aus dem Wert des Attributs `name` im Element `<jxf:submit/>`. Nachdem das Formular durch einen entsprechenden JXForms Generator bzw. Transformer verarbeitet und durch ein XSLT-Stylesheet in XHTML umgewandelt wurde, findet man in diesem Attribut einen langen String, der durch Doppelpunkte getrennt ist. Dieser könnte zum Beispiel so aussehen:

```
cocoon-action-
2a676857766b53543d3d401e242a7e484b1c344e:null:next
```

Dieser String enthält neben einigen anderen Informationen eine Continuation-ID. Welche Continuation durch diese ID angesprochen wird, wird durch das Attribut `continuation` in der Definition bestimmt. Wurde das Attribut `continuation="back"` angegeben, enthält er die ID der vorhergehenden Continuation. Im Fall von `continuation="forward"` und bei keiner Angabe dieses Attributs, enthält er die ID der nachfolgenden Continuation. Dieser String wird an das JXForms-Flowscript übermittelt, das die Continuation-ID daraus extrahiert und das zugehörige Continuation-Objekt lädt.

Um eine Vor- und Zurück-Navigation mit JXForms zu erzeugen, ist es notwendig, dass alle versendeten Formulare durch die Funktion `jxforms` des Flowscripts `JXForms.js` verarbeitet werden. Des Weiteren ist für die Navigation in die entsprechende Richtung ein Submit-Button mit dem Attribut `continuation="forward"` bzw. `continuation="back"` zu erzeugen.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<html xmlns:jxf="http://apache.org/cocoon/jxforms/1.0">
<header>
   <title>Registrierung - Ihre Kenntnisse</title>
</header>
<body>
   <h1>Ihre Kenntnisse</h1>
   Bitte wählen Sie diejenigen Bereiche,<br/>
   in denen Sie am meisten Kenntnisse besitzen.
   <jxf:form id="registration" action="./register">
   <table border="1">
   <tr>
      <td>Kenntnisse</td>
      <td>
        <jxf:select ref="knowledge">
          <jxf:item>
            <jxf:label>Programmierung</jxf:label>
            <jxf:value>Programmierung</jxf:value>
          </jxf:item>
          <jxf:item>
            <jxf:label>IT-Technik</jxf:label>
            <jxf:value>IT-Technik</jxf:value>
          </jxf:item>
          <jxf:item>
            <jxf:label>Software-Design</jxf:label>
            <jxf:value>Software-Design</jxf:value>
          </jxf:item>
          <jxf:item>
            <jxf:label>Support</jxf:label>
            <jxf:value>Support</jxf:value>
          </jxf:item>
        </jxf:select>
    </td>
    </tr>
    <tr>
      <td colspan="2">
        <jxf:submit id="prev" continuation="back">
          <jxf:label>Zurück</jxf:label>
        </jxf:submit>

        <jxf:submit id="next" continuation="forward">
          <jxf:label>Weiter</jxf:label>
```

```
        </jxf:submit>
      </td>
    </tr>
  </table>
  </jxf:form>
</body>
</html>
```

Listing 12.5 knowledge.xml

<jxf:select/>

Das nächste Dokument erfragt vom Benutzer Informationen über seinen Wissenstand im Bereich »Computer und Software«. Diese Angaben kann er durch das Selektieren einzelner vorgegebenen Antworten machen. Dabei wird hier durch das Element `<jxf:select/>` eine Auswahlliste angegeben, aus der mehrere Elemente markiert werden können.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<html xmlns:jxf="http://apache.org/cocoon/jxforms/1.0">
<header>
  <title>Registrierung - Ihre Ausbildung</title>
</header>
<body>
  <h1>Ihre Ausbildung</h1>
  Bitte geben Sie hier Ihre Vorbildung an.
  <jxf:form id="registration" action="./register">
  <table border="1">
    <tr>
      <td>Abgeschlossene Studiengänge</td>
      <td>
        <jxf:select ref="education">
          <jxf:item>
            <jxf:label>Informatik</jxf:label>
            <jxf:value>Informatik</jxf:value>
          </jxf:item>
          <jxf:item>
            <jxf:label>Elektrotechnik</jxf:label>
            <jxf:value>Elektrotechnik</jxf:value>
          </jxf:item>
          <jxf:item>
            <jxf:label>Physik</jxf:label>
            <jxf:value>Physik</jxf:value>
          </jxf:item>
          <jxf:item>
```

```
            <jxf:label>BWL</jxf:label>
            <jxf:value>BWL</jxf:value>
          </jxf:item>
        </jxf:select>
      </td>
    </tr>
    <tr>
      <td>Zusätzliche Anmerkungen</td>
      <td><jxf:textarea ref="education_info"/></td>
    </tr>
    <tr>
      <td colspan="2">
        <jxf:submit id="prev" continuation="back">
          <jxf:label>Zurück</jxf:label>
        </jxf:submit>

        <jxf:submit id="next" continuation="forward">
          <jxf:label>Weiter</jxf:label>
        </jxf:submit>
      </td>
    </tr>
  </table>
  </jxf:form>
</body>
</html>
```

Listing 12.6 education.xml

<jxf:textarea/>

Im letzten Formular `education.xml` – in das Werte eingegeben werden können – wird vom Benutzer seine Ausbildung abgefragt. Dies erfolgt wieder durch eine Multiselect-Liste, die aus dem JXForms-Element `<jxf:select/>` erzeugt wird. Zusätzlich kann er Anmerkungen zu seiner Ausbildung in einem Textbereich machen. Dieser Bereich wird durch das Element `<jxf:textarea/>` bestimmt.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<html xmlns:jxf="http://apache.org/cocoon/jxforms/1.0">
<header>
  <title>Registrierung - Vorschau</title>
</header>
<body>
  <h1>Vorschau</h1>
  Bitte übeprüfen Sie nun, ob alle <br/>
  gemachten Angaben korrekt sind.
```

```
<jxf:form id="registration" action="./register">
<table border="1">
  <tr>
    <td>Vorname</td>
    <td><jxf:output ref="firstName"/></td>
  </tr>
  <tr>
    <td>Nachname</td>
    <td><jxf:output ref="lastName"/></td>
  </tr>
  <tr>
    <td>Wohnort</td>
    <td><jxf:output ref="city"/></td>
  </tr>
  <tr>
    <td>Email</td>
    <td><jxf:output ref="email"/></td>
  </tr>
  <tr>
    <td>Geschlecht</td>
    <td><jxf:output ref="sex"/></td>
  </tr>
  <tr>
    <td>Kenntnisse</td>
    <td>
      <jxf:repeat nodeset="/knowledge">
        <jxf:output ref="."/><br/>
      </jxf:repeat>
    </td>
  </tr>
  <tr>
    <td>Ausblidung</td>
    <td>
      <jxf:repeat nodeset="/education">
        <jxf:output ref="."/><br/>
      </jxf:repeat>
    </td>
  </tr>
  <tr>
    <td>Zusätzliche Informationen zur Ausbildung</td>
    <td><jxf:output ref="education_info"/></td>
  </tr>
```

```
    <tr>
      <td colspan="2">
        <jxf:submit id="prev" continuation="back">
          <jxf:label>Zurück</jxf:label>
        </jxf:submit>

        <jxf:submit id="next" continuation="forward">
          <jxf:label>Registrieren</jxf:label>
        </jxf:submit>
      </td>
    </tr>
  </table>
  </jxf:form>
</body>
</html>
```

Listing 12.7 preview.xml

Die letzte Formulardefinition ist die Datei `preview.xml`. In dieses Formular können keine Informationen eingegeben werden. Hier werden lediglich die Werte aus den anderen Formularen wieder ausgegeben.

<jxf:output/> Das Element `<jxf:output/>` wird dabei verwendet, um einzelne Werte aus einem Model auszugeben. Dabei wird mit dem Attribut `ref` genau der Knoten im Model bestimmt, der ausgegeben werden soll.

<jxf:repeat/> Neben der Ausgabe von einzelnen Werten ist es natürlich auch wichtig, eine Liste von Werten ausgeben zu können. In unserem Beispiel befinden sich im Model zwei solcher Listen: `knowledge` und `education`. Über beide kann mit dem Element `<jxf:repeat/>` iteriert werden. Dabei bestimmt das Attribut `nodeset` die Knotenmenge, die zur Ausgabe selektiert werden soll. Innerhalb des Elements `<jxf:repeat/>` wird durch `<jxf:ouput ref="."/>` der aktuelle Wert während des Durchlaufs ausgegeben.

Als letzte Templates fügen Sie zu Ihrem Verzeichnis `documents` bitte noch die beiden Dateien `start.xml` und `end.xml` hinzu. Diese Seiten dienen lediglich dazu, den Start- und Endpunkt des Wizards anzuzeigen.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<html>
<header>
  <title>Registrierung</title>
</header>
```

```
<body>
  <h1>Startseite</h1>
  Herzlich willkommen auf diesen Seiten.<br/>
  Hier können Sie sich registrieren:
  <a href="./register">Start</a>
</body>
</html>
```

Listing 12.8 start.xml

Das Template `start.xml` wird ausgegeben, wenn der Benutzer die Registrierung betritt. Auf dieser Seite hat er dann die Möglichkeit, den Wizard zu starten. Der Link zum Starten führt dabei zu einer URI, hinter der in der Sitemap ein JXForms-Flowscript hinterlegt ist. Dieses Skript startet anschließend den Wizard.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<html>
<header>
  <title>Registrierung</title>
</header>
<body>
  <h1>Vielen Dank.</h1>
  Ihre Registrierung wurde gespeichert.<br/>
  <a href="./">Startseite</a>
</body>
</html>
```

Listing 12.9 end.xml

Nachdem die Daten der Registrierung entgegengenommen und fiktiv gespeichert wurden, wird durch dieses Template eine Bestätigung ausgegeben, dass dies erfolgreich durchgeführt wurde.

Flowscript

Da Sie nun die notwendigen Formulardefinitionen und Templates erzeugt haben, sehen wir uns an, wie das Flowscript aussieht, das das Daten-Model erzeugt, die Reihenfolge der Formular-Instanzen angibt und das fiktive Speichern vornimmt. Werfen Sie hierfür einen Blick auf das folgende Listing:

```
cocoon.load("resource://org/apache/cocoon/components/↵
jxforms/flow/javascript/JXForm.js");

function main(form) {
```

```
  // Model erstellen
  var model = {
    "firstName": "",
    "lastName": "",
    "city": "",
    "email": "",
    "sex": "",
    "education": new Array(),
    "education_info": "",
    "knowledge": new Array()
  };

  // Model registrieren
  form.setModel(model);

  // Adress-Seite senden
  form.sendView("adress.xml");

  // Kenntniss-Seite senden
  form.sendView("knowledge.xml");

  // Ausbildungs-Seite senden
  form.sendView("education.xml");

  // Vorschau-Seite senden
  form.sendView("preview.xml");

  // Hier wuerde das Speichern der Daten erfolgen

  // Ende-Seite senden
  form.sendView("end");
}
```

Listing 12.10 register.js

Im Flowscript `register.js` erfolgen fünf wichtige Abläufe:

- Einbinden des JXForms-Flowscripts `JXForms.js` durch den Aufruf `cocoon.load(...)`
- Erstellen des Daten-Models
- Registrieren des Daten-Models
- Senden der Formulare in der gewünschten Reihenfolge
- Fiktives Speichern der Daten

Nachdem durch die Sitemap die Funktion `jxforms` des eingebundenen Flowscripts `JXForms.js` aufgerufen wurde, werden hier zwei wichtige Aufgaben ausgeführt: Zum einen wird entschieden, ob eine existierende Continuation geladen werden soll und wenn ja, welche, und zum anderen werden die Werte existierender und übereinstimmender Request-Parameter in das Model kopiert, falls ein solches bereits vorhanden ist. Das Model wird innerhalb des Flowscripts erstellt. In diesem Beispiel handelt es sich um ein JavaScript-Objekt. Es könnte jedoch genauso ein DOM-, JavaBean- oder anderes Objekt sein.

Nach der Ausführung der entsprechenden Aufgaben ruft das Skript `JXForms.js` seinerseits die »Start-Funktion« des Skripts `register.js` auf. Der Name der aufzurufenden Methode `main` wird dabei in der Sitemap durch folgenden Sitemap-Parameter angegeben: Sitemap

```
<map:paramater name="function" value="main" />
```

Als zweiten Parameter erhält die Funktion `jxforms` zusätzlich den Namen der Formular-Instanz durch folgenden Sitemap-Parameter übergeben:

```
<map:paramater name="id" value="registration"/>
```

Sehen Sie sich hierzu das folgende Listing an, das die erforderliche Sitemap zeigt.

```
<?xml version="1.0"?>
<map:sitemap↵
     xmlns:map="http://apache.org/cocoon/sitemap/1.0">

   <map:components>
   <map:generators>
      <map:generator name="jxforms"
         src="org.apache.cocoon.↵
         generation.JXFormsGenerator"/>
      <map:generator name="jxt"
         src="org.apache.cocoon.↵
         generation.JXTemplateGenerator"/>
   </map:generators>
   </map:components>

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

```
  <map:pipelines>
    <map:pipeline>

    <map:match pattern="">
      <map:generate type="jxt"
        src="documents/start.xml"/>
      <map:serialize type="html"/>
    </map:match>

    <map:match pattern="end">
      <map:generate type="jxt"
        src="documents/end.xml"/>
      <map:serialize type="html"/>
    </map:match>

    <map:match pattern="register">
      <map:call function="jxform">
        <map:parameter name="function"
          value="main"/>
      <map:parameter name="id"
          value="registration"/>
      </map:call>
    </map:match>

    </map:pipeline>
    <map:pipeline internal-only="true">

    <map:match pattern="*.xml">
      <map:generate type="jxforms"
       src="documents/{1}.xml"/>
       <map:transform type="xslt"
       src="stylesheets/jxforms2html.xsl"/>
      <map:serialize type="html"/>
    </map:match>

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

</map:sitemap>
```

Listing 12.11 Die Sub-Sitemap

Wie Sie in Listing 12.11 sehen können, wird hier das XSLT-Stylesheet `jxforms2html.xsl` verwendet, um die JXForms-Elemente in

(X)HTML-Elemente zu wandeln. Dieses Stylesheet können Sie im Samples-Verzeichnis von Cocoon finden:

```
$COCOON_HOME/samples/jxforms/stylesheets
```

Kopieren Sie sich bitte das gesamte Verzeichnis `stylesheets` in Ihre Subapplikation `registration`.

Als letzten Schritt legen Sie bitte im Verzeichnis `registration` eine Sub-Sitemap `sitemap.xmap` an und fügen Sie zu dieser Datei die in Listing 12.11 angegebenen Zeilen hinzu.

Nachdem Sie auch diese Datei angelegt haben, können Sie mit einem Aufruf der URI

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

das Beispiel starten und testen. Beachten Sie vor allem, dass vollkommen egal ist, ob Sie mit den Formular- oder mit den Browser-Buttons vorwärts oder zurück navigieren. Es erscheint immer der korrekte Zustand des Formulars.

12.2.3 Deklarative Validierung

Eine deklarative Validierung, in der die Formulareingaben mit Hilfe von Schematron[6] geprüft werden, ist in JXForms ebenfalls vorgesehen. Hierfür müssen beim Aufruf der Funktion `jxforms` durch `<map:call/>` zusätzlich noch der Namensraum

```
http://www.ascc.net/xml/schematron
```

und der Ort, an dem die Schema-Instanz vorliegt, angegeben werden. Das folgende Listing zeigt, wie eine Schematron-Datei eingebunden wird.

```
<map:match pattern="*.js">
   <map:call function="jxform">
      <map:parameter name="function" value="*.js"/>
      <map:parameter name="id" value="myForm"/>
      <map:parameter name="validator-schema-namespace"
         value="http://www.ascc.net/xml/schematron"/>
      <map:parameter name="validator-schema"
         value="[Pfad-zur-Schema-Datei]"/>
   </map:call>
</map:match>
```

Listing 12.12 Einbinden einer Schematron-Validierung

6 http://www.ascc.net/xml/schematron/

Was ist Schematron?

Mit Schematron lassen sich XML-Dokumente – ähnlich wie mit XML Schema – formal beschreiben. Dadurch können Dokumente beschränkt werden. Innerhalb Ihres Schematron-Dokuments können Sie XPath-Ausdrücke verwenden, um Ihre Formularelemente aus dem JXForms-Model zu referenzieren. Da JXForms das Projekt JXPath verwendet, um Schematron zu integrieren, können natürlich auch alle unterstützten Objekte durch Schematron validiert werden (JavaBeans, DOM-Objekte usw.).

Da Schematron zwar von JXForms verwendet wird, aber im Großen und Ganzen eine eigene, umfangreiche Sprache darstellt, möchte ich an dieser Stelle auf Ihr XML-Buch oder die Website

http://www.ascc.net/xml/resource/schematron/Schematron2000.html

verweisen, wenn Sie mehr über Schematron und dessen »Assertion-Language« wissen möchten. Im Verzeichnis `samples/jxforms/wizard` von Cocoon können Sie übrigens ein Beispiel für eine solche Schematron-Datei vorfinden, vorausgesetzt natürlich, Sie haben den JXForms-Block in Cocoon kompiliert und die Samples aktiviert.

12.2.4 JXForms-Elemente

Welche JXForms-Elemente Ihnen für bestimmte Aufgaben innerhalb Ihrer Formulardefinition zur Verfügung stehen, werden wir uns in diesem Abschnitt ansehen. Eine vollständige Übersicht über alle XForms-Elemente können Sie auf den Seiten des W3C erhalten:

http://www.w3.org/TR/xforms/slice8.html

Achten Sie jedoch unbedingt darauf, dass nicht alle diese Elemente und deren Attribute von JXForms unterstützt werden, ebenso nicht die eigenen, die zusätzlich hinzugefügt wurden. Alle verwendeten JX-Forms-Elemente müssen dem folgenden Namensraum zugeordnet werden und sollten durch das Präfix `jxt` gebunden werden:

```
http://apache.org/cocoon/jxforms/1.0
```

Zusätzlich können Sie in den meisten JXForms-Elementen eigene Attribute angeben, die unverändert in das Ergebnisdokument übernommen werden bzw. durch ein von Ihnen erstelltes XSLT-Stylesheet verarbeitet werden können.

<jxf:form/>

Dieses Element beinhaltet alle anderen Formularelemente und stellt somit das Wurzelelement eines jeden JXForms dar. Innerhalb von W3C XForms gibt es kein direkt vergleichbares Element.

Attribut	Beschreibung	Pflicht
`id`	Ein eindeutiger Bezeichner des Formulars. Dieser Wert muss mit dem in der Sitemap angegebenen Parameter `id` übereinstimmen. Dadurch wird das Formular eindeutig referenziert.	Ja
`view`	Der Attributwert muss mit dem Attributwert von `id` des Elements `<phase/>` innerhalb einer Schematron-Validierung übereinstimmen, um Validierungsregeln für dieses Formular definieren zu können.	Nein
`action`	Mit diesem Attribut wird eine relative URI angegeben, an die das Formular gesendet werden soll.	Nein

Tabelle 12.1 Attribute von <jxf:form/>

Nachfolgend können Sie ein Beispiel für die Verwendung des Elements `<jxf:form/>` sehen:

```
<jxf:form id="contact" view="valuesCorrect"
   action="send.html">
   ...
</jxf:form>
```

Listing 12.13 Beispiel für das Element <jxf:form/>

<jx:label/>

Der Inhalt dieses Elements weist dem Formularelement einen beschreibenden Namen zu. Jedes Formularelement muss ein solches Label-Element enthalten. Es basiert auf dem gleichnamigen Element aus XForms.

Attribut	Beschreibung	Pflicht
`ref`	Dieses Attribut kann einen beliebigen XPath-Ausdruck enthalten, der dieses Element an ein bestimmtes Model-Feld und somit einen Rückgabewert bindet. Dies ist ein alternativer Weg, den Inhalt von `<jxf:label/>` zu bestimmen. Enthält das Element `<jxf:label/>` einen Wert und ist gleichzeitig das Attribut `ref` angegeben, wird der Rückgabewert von `ref` als Wert für das Element verwendet und der Inhalt von `<jxf:label/>` nicht beachtet.	Nein

Tabelle 12.2 Attribute von <jxf:label/>

Nachfolgend sehen Sie ein Beispiel für die Verwendung des Elements <jxf:label/>:

```
<jxf:input ref="/productNumber">
   <jxf:label>Artikelnummer</jxf:label>
</jxf:input>
```

Listing 12.14 Beispiel für das Element <jxf:label/>

<jxf:help/>

Dieses Element basiert auf dem gleichnamigen W3C-XForms-Element und kann optional in jedem Formularelement angegeben werden, um dem Benutzer einen kontextsensitiven Hilfetext anzuzeigen. Es enthält keine Attribute. Im Folgenden können Sie ein Beispiel für die Verwendung des Elements <jxf:help/> sehen:

```
<jxf:input ref="/productNumber">
   <jxf:label>Artikelnummer</jxf:label>
   <jxf:help>
      Bitte geben Sie Ihre Art.-Nr. an.
   </jxf:help>
</jxf:input>
```

Listing 12.15 Beispiel für das Element <jxf:help/>

<jxf:hint/>

Hiermit kann eine etwas ausführlichere Beschreibung für ein Formularelement angegeben werden, die zum Beispiel per Mouse-Over angezeigt wird. Es kann mit einem Tool-Tip-Text verglichen werden und entspricht dem gleichnamigen Element aus den W3C-XForms. Dieses Element enthält keine Attribute. Die Verwendung des Elements <jxf:hint/> sehen Sie in folgendem Beispiel:

```
<jxf:input ref="/productNumber">
   <jxf:label>Artikelnummer</jxf:label>
   <jxf:help>
      Bitte geben Sie Ihre Art.-Nr. an.
   </jxf:help>
   <jxf:hint>Feld für Artikelnummer</jxf:hint>
</jxf:input>
```

Listing 12.16 Beispiel für das Element <jxf:hint/>

<jxf:value/>

Der Wert innerhalb des Elements <jxf:value/> wird dem entsprechenden Formularelement zugewiesen. Es entspricht dem gleichnamigen W3C-XForms-Element und ist nur innerhalb von Formularelementen gültig.

Attribut	Beschreibung	Pflicht
ref	Dieses Attribut kann einen beliebigen XPath-Ausdruck enthalten, der dieses Element an ein bestimmtes Model-Feld und somit einen Rückgabewert bindet. Dies ist ein alternativer Weg, den Inhalt von <jxf:value/> zu bestimmen. Enthält das Element <jxf:value/> einen Wert und ist gleichzeitig das Attribut ref angegeben, wird der Rückgabewert von ref als Wert für das Element verwendet und der Inhalt von <jxf:value/> nicht beachtet.	Nein

Tabelle 12.3 Attribute von <jxf:value/>

Nachfolgend sehen Sie ein Beispiel für die Verwendung des Elements <jxf:value/>:

```
<jxf:input ref="/productNumber">
   <jxf:label>Artikelnummer</jxf:label>
   <jxf:help>
   Bitte geben Sie Ihre Art.-Nr. an.
   </jxf:help>
   <jxf:hint>Feld für Artikelnummer</jxf:hint>
   <jxf:value>12345</jxf:value>
</jxf:input>
```

Listing 12.17 Beispeil von <jxf:value/>

<jxf:item/>

Dieses Element ist nur innerhalb von <jxf:select/> und <jxf:select1/> gültig und enthält die weiteren Elemente <jxf:value/> und <jxf:label/>. Es steht für einen Eintrag innerhalb einer Auswahlliste, enthält keine Attribute und entspricht dem gleichnamigen W3C-XForms-Element.

Dieses Element wird meistens für die Erzeugung des HTML-Elements <option value=""></option> innerhalb von <select/> verwendet. Im Folgenden sehen Sie ein Beispiel für die Verwendung des Elements <jxf:ietm/>:

```
<jxf:select1 ref="/productCategory">
   <jxf:label>Produkt-Kategorie</jxf:label>
      <jxf:item>
         <jxf:label>Lebensmittel</jxf:label>
         <jxf:value>1</jxf:value>
      </jxf:item>
      <jxf:item>
      <jxf:label>Getränke</jxf:label>
         <jxf:value>2</jxf:value>
      </jxf:item>
</jxf:select1>
```

Listing 12.18 Beispiel für <jxf:item/>

Dieses Gesamtkonstrukt könnte nach der Transformation z.B. folgendem Auswahlelement entsprechen:

Abbildung 12.4 Abbildung 12.5:Items innerhalb eines Drop-down-Menüs

<jxf:input/>

Mit diesem Element kann ein Eingabefeld für Freitext oder andere Datentypen angegeben werden. Es entspricht dem gleichnamigen W3C-XForms-Element. Dieses Element wird meistens für die Erzeugung der HTML-Elemente `<input type="text">`, `<input type="checkbox">` oder `<input type="radio">` verwendet. Die Entscheidung darüber, welches dieser HTML-Elemente verwendet wird, wird anhand des Datentyps des Kindelements `<jxf:value/>` gefällt.

Attribut	Beschreibung	Pflicht
`ref`	Dieses Attribut kann einen beliebigen XPath-Ausdruck enthalten, der dieses Element an ein bestimmtes Model-Feld und somit einen Rückgabewert bindet.	Ja
`class`	Dieses Attribut kann verwendet werden, um das Element über ein Stylesheet zu formatieren.	Nein

Tabelle 12.4 Attribute von <jxf:input/>

Es folgen einige Beispiele für die Verwendung des Elements `<jxf:input/>`:

```
<jxf:input ref="/productNumber">
   <jxf:label>Artikelnummer</jxf:label>
   <jxf:value>1234</jxf:value>
</jxf:input>
```

Listing 12.19 Ein Beispiel für <jxf:input/>

Nach der Transformation könnte dieses Element folgendem HTML-Eingabefeld mit gesetztem Wert entsprechen:

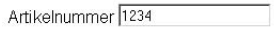


Abbildung 12.5 Ein Textfeld

```
<jxf:input type="/noNews">
   <jxf:label>News per Email erhalten</jxf:label>
   <jxf:value>true</jxf:value>
</jxf:input>
```

Listing 12.20 Ein Beispiel für <jxf:input/>

Nach der Transformation könnte dieses Element folgendem HTML-Eingabefeld entsprechen, da es sich bei `<jxf:value/>` um einen booleschen Wert handelt:

Abbildung 12.6 Eine Checkbox

<jxf:secret/>

Dieses Element entspricht einem Eingabefeld für Werte, die nicht angezeigt werden sollen, wie zum Beispiel Passwörter. Das zugehörige W3C-XForms-Element ist das gleichnamige Element.

Dieses Element wird meistens für die Erzeugung des HTML-Elements `<input type="password">` verwendet.

Attribut	Beschreibung	Pflicht
`ref`	Dieses Attribut kann einen beliebigen XPath-Ausdruck enthalten, der dieses Element an ein bestimmtes Model-Feld und somit einen Rückgabewert bindet.	Ja
`class`	Dieses Attribut kann verwendet werden, um das Element über ein Stylesheet zu formatieren.	Nein

Tabelle 12.5 Attribute von <jxf:secret/>

Nachfolgend können Sie ein Beispiel für die Verwendung des Elements `<jxf:secret/>` sehen:

```
<jxf:secret ref="/password">
   <jxf:label>Passwort</jxf:label>
   <jxf:value>myPassword</jxf:label>
</jxf:secret>
```

Listing 12.21 Ein Beispiel für <jxf:secret/>

Nach der Transformation könnte dieses Element folgendem HTML-Eingabefeld entsprechen:

Abbildung 12.7 Ein Passwort-Feld

<jxf:textarea/>

Dieses Element basiert auf dem gleichnamigen W3C-XForms-Element und steht für einen editierbaren Textbereich. Es wird meistens für die Erzeugung des HTML-Elements `<textarea/>` verwendet.

Attribut	Beschreibung	Pflicht
`ref`	Dieses Attribut kann einen beliebigen XPath-Ausdruck enthalten, der dieses Element an ein bestimmtes Model-Feld und somit einen Rückgabewert bindet.	Ja
`class`	Dieses Attribut kann verwendet werden, um das Element über ein Stylesheet zu formatieren.	Nein

Tabelle 12.6 Attribute von <jxf:textarea/>

Im Folgenden sehen Sie ein Beispiel für die Verwendung des Elements `<jxf:textarea/>`:

```
<jxf:textarea ref="/text" class="text">
   <jxf:label>Beschreibung</jxf:label>
   <jxf:value>Das ist eine Beschreibung</jxf:label>
</jxf:textarea>
```

Listing 12.22 Ein Beispiel für <jxf:textarea/>

Nach der Transformation könnte dieses Element folgendem HTML-Eingabefeld mit gesetztem Wert entsprechen:

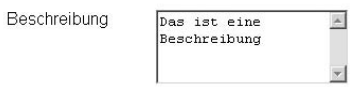


Abbildung 12.8 Eine Textarea

<jxf:output/>

Hiermit wird eine einfache Ausgabe von Model-Werten erzeugt, die nicht editierbar ist. Dieses Element basiert auf dem gleichnamigen W3C-XForms-Element.

Attribut	Beschreibung	Pflicht
`ref`	Dieses Attribut kann einen beliebigen XPath-Ausdruck enthalten, der dieses Element an ein bestimmtes Model-Feld und somit einen Rückgabewert bindet.	Ja
`class`	Dieses Attribut kann verwendet werden, um das Element über ein Stylesheet zu formatieren.	Nein
`value`	Dieses Attribut kann einen Wert enthalten, der ausgegeben werden soll. Wenn dieses Attribut angegeben wird, kann das Attribut `ref` nicht mehr verwendet werden.	(Ja)

Tabelle 12.7 Attribute von <jxf:output/>

Nachfolgend können Sie ein Beispiel für die Verwendung des Elements `<jxf:output/>` sehen:

```
Im Preis von <jxf:output ref="/price"/> ist die
gesetzliche Mehrwertsteuer bereits inbegriffen.
```

Nach der Transformation könnte dieses Element einer HTML-Ausgabe entsprechen, wie in Abbildung 12.10 dargestellt.

Abbildung 12.9 Die HTML-Ausgabe

<jxf:select/>

Das Element `<jxf:select/>` basiert auf dem gleichnamigen W3C-XForms-Element und steht für eine Auswahlliste, aus der beliebig viele Elemente ausgewählt werden dürfen (Multiselect). Die einzelnen Auswahlelemente werden durch das Kindelement `<jxf:item/>` bestimmt. Es wird meistens für die Erzeugung des (X)HTML-Elements `<select mutliple="multiple">` verwendet.

Attribut	Beschreibung	Pflicht
`ref`	Dieses Attribut kann einen beliebigen XPath-Ausdruck enthalten, der dieses Element an ein bestimmtes Model-Feld und somit einen Rückgabewert bindet.	Ja
`appearance`	Dieses Attribut kann einen der Werte `compact` oder `full` enthalten, je nachdem, ob die Listenelemente in einer Multiselect- oder einer Checkbox-Darstellung angezeigt werden sollen. Der Standardwert ist `compact`.	Nein
`class`	Dieses Attribut kann verwendet werden, um das Element über ein Stylesheet zu formatieren.	Nein

Tabelle 12.8 Attribute von <jxf:select/>

Es folgt ein Beispiel für die Verwendung des Elements `<jxf:select/>`:

```
<jxf:select ref="/searchCategories">
   <jxf:label>Kategorie</jxf:label>
   <jxf:item>
      <jxf:label>Bücher</jxf:label>
      <jxf:value>1</jxf:value>
   </jxf:item>
   <jxf:item>
      <jxf:label>Musik</jxf:label>
      <jxf:value>2</jxf:value>
   </jxf:item>
   <jxf:item>
      <jxf:label>Elektronik</jxf:label>
      <jxf:value>3</jxf:value>
   </jxf:item>
   <jxf:item>
      <jxf:label>Software</jxf:label>
      <jxf:value>4</jxf:value>
   </jxf:item>
</jxf:select>
```

Listing 12.23 Ein Beispiel für <jxt:select/>

Nach der Transformation könnte dieses Element einer selektierten HTML-Auswahlliste entsprechen, wie in Abbildung 12.10 gezeigt. Diese Ansicht wird durch den Wert `compact` im Attribut `appearance` festgelegt.

Abbildung 12.10 Eine Multiselect-Auswahlliste

Eine andere denkbare Darstellung von `<jxf:select/>` wäre eine Auswahlliste, die durch Checkboxen realisiert ist, wie in Abbildung 12.11 dargestellt. Diese Ansicht wird durch den Wert `full` im Attribut `appearance` festgelegt.

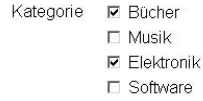


Abbildung 12.11 Eine Checkbox-Auswahlliste

<jxf:select1/>

Mit diesem Element wird ähnlich dem Element `<jxf:select/>` eine Auswahlliste definiert, aus der jedoch nur ein einziges Element ausgewählt werden kann (Singleselect). Das zugehörige W3C XForms Element lautet ebenfalls `<select1/>`. Die einzelnen Auswahlelemente werden durch das Kindelement `<jxf:item/>` bestimmt. Es wird meistens für die Erzeugung des (X)HTML-Elements `<select/>` verwendet.

Attribut	Beschreibung	Pflicht
`ref`	Dieses Attribut kann einen beliebigen XPath-Ausdruck enthalten, der dieses Element an ein bestimmtes Model-Feld und somit einen Rückgabewert bindet.	Ja
`appearance`	Dieses Attribut kann einen der Werte `compact` oder `full` enthalten, je nachdem, ob die Listenelemente in einer Drop-down-Box- oder Radio-Button-Darstellung angezeigt werden sollen. Der Standardwert ist `compact`.	Nein
`class`	Dieses Attribut kann verwendet werden, um das Element über ein Stylesheet zu formatieren.	Nein

Tabelle 12.9 Attribute von <jxf:select1/>

Nachfolgend können Sie ein Beispiel für die Verwendung des Elements `<jxf:select1/>` sehen:

```
<jxf:select1 ref="/searchCategories">
   <jxf:label>Kategorie</jxf:label>
   <jxf:item>
```

```
      <jxf:label>Bücher</jxf:label>
      <jxf:value>1</jxf:value>
   </jxf:item>
   <jxf:item>
      <jxf:label>Musik</jxf:label>
      <jxf:value>2</jxf:value>
   </jxf:item>
   <jxf:item>
      <jxf:label>Elektronik</jxf:label>
      <jxf:value>3</jxf:value>
   </jxf:item>
   <jxf:item>
      <jxf:label>Software</jxf:label>
      <jxf:value>4</jxf:value>
   </jxf:item>
</jxf:select1>
```

Listing 12.24 Ein Beispiel für <jxf:select1/>

Nach der Transformation könnte dieses Konstrukt einer HTML-Auswahlliste wie in Abbildung 12.12 entsprechen. Diese Ansicht wird durch den Wert `compact` im Attribut `appearance` festgelegt.

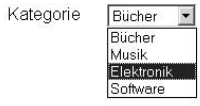


Abbildung 12.12 Eine Single-Auswahl-Box

Eine andere denkbare Darstellung von `<jxf:select1/>` wäre eine Auswahlliste, die durch Radio-Buttons realisiert ist, wie es Abbildung 12.13 zeigt. Diese Ansicht wird durch den Wert `full` im Attribut `appearance` festgelegt.

Abbildung 12.13 Eine Radio-Button-Liste

<jxf:submit/>

Dieses Element weicht von der Definition des gleichnamigen W3C-XForms-Element ab. Es bestimmt einen Submit-Button, um Werte zu

versenden. Es wird meistens für die Erzeugung des HTML-Elements `<input type="submit"/>` verwendet.

Attribut	Beschreibung	Pflicht
`id`	Gibt diesem Submit-Button eine eindeutige, beliebige Id.	Ja
`class`	Dieses Attribut kann verwendet werden, um das Element über ein Stylesheet zu formatieren.	Nein
`continuation`	Wird dieses Attribut angegeben, muss es entweder als Wert `forward` oder `back` besitzen. Hiermit wird JXForms angewiesen, entweder das Flowscript weiter abzuarbeiten oder eine Continuation zurückzuspringen. Dieses Attribut ist äußerst nützlich, wenn Sie Formulare über mehrere Instanzen hinweg anzeigen möchten (z.B. bei Wizards).	Nein

Tabelle 12.10 Attribute von <jxf:submit/>

Im Folgenden sehen Sie ein Beispiel für die Verwendung des Elements `<jxf:submit/>`:

```
<jxf:submit id="send" continuation="next">
   <jxf:label>Senden</jxf:label>
</jxf:submit>
```

Listing 12.25 Ein Beispiel für <jxf:submit/>

Nach der Transformation könnte dieses Element folgendem HTML-Button entsprechen:

Abbildung 12.14 Ein Submit-Button

<jxf:group/>

Mit `<jxf:group/>` kann eine beliebige Anzahl von Formularelementen zu einer Gruppe zusammengefasst werden. Innerhalb von W3C-XForms existiert ein gleichnamiges Element, das dieselbe Bedeutung hat und in derselben Weise angewendet wird.

Attribut	Beschreibung	Pflicht
`ref`	Der Wert dieses Attributs entspricht einem XPath-Ausdruck, der einen Kontext-Knoten bestimmt. Die Kindelemente dieses Knotens sind die enthaltenen, gruppierten Formularelemente.	Ja

Tabelle 12.11 Attribute von <jxf:group/>

Attribut	Beschreibung	Pflicht
`class`	Dieses Attribut kann verwendet werden, um das Element über ein Stylesheet zu formatieren.	Nein

Tabelle 12.11 Attribute von <jxf:group/> (Forts.)

Hier ein Beispiel für die Verwendung des Elements `<jxf:group/>`:

```
<jxf:group ref="adress">
   <jxf:label>Adresse</jxf:label>
   <jxf:input ref="/firstName">
      <jxf:label>Vorname</jxf:label>
   </jxf:input>
   <jxf:input ref="/lastName">
      <jxf:label>Nachname</jxf:label>
   </jxf:input>
</jxf:group>
<jxf:group ref="product">
   <jxf:label>Produkt</jxf:label>
   <jxf:input type="/number">
      <jxf:label>Nummer</jxf:label>
   </jxf:input>
   <jxf:input ref="/name">
      <jxf:label>Name</jxf:label>
   </jxf:input>
</jxf:group>
```

Listing 12.26 Ein Beispiel für <jxf:group/>

Nach der Transformation könnte diese Definition einer HTML-Ausgabe entsprechen, wie sie in Abbildung 12.15 gezeigt wird.

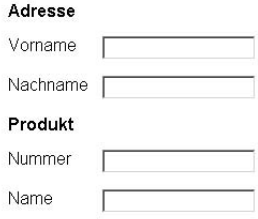


Abbildung 12.15 Ein gruppiertes Formular

<jxf:repeat/>

Dieses Element iteriert über eine angegebene Knotenmenge und schreibt mit jedem Durchgang ihren Inhalt in das Ergebnisdokument. Es basiert auf dem gleichnamigen W3C-XForms-Element.

Attribut	Beschreibung	Pflicht
`nodeset`	Ein XPath-Ausruck, der eine eindeutige Knotenmenge zurückliefert, über die iteriert werden kann.	Ja

Tabelle 12.12 Attribute von <jxf:repeat/>

Nachfolgend sehen Sie ein Beispiel für die Verwendung des Elements `<jxf:repeat/>`:

```
<jxf:repeat nodeset="/products/product">
   <jxf:input ref=".">
      <jxf:label ref="./label"/>
      <jxf:value ref="./value"/>
   </jxf:input>
</jxf:repeat>
```

Listing 12.27 Ein Beispiel für <jxf:repeat/>

<jxf:itemset/>

Das Element `<jxf:itemset/>` basiert auf dem gleichnamigen W3C-XForms-Element. Es wird verwendet, um innerhalb von `<jxf:select/>` und `<jxf:select1/>` dynamisch die Elemente `<jxf:item/>` aus einer Knotenmenge zu erzeugen.

Attribut	Beschreibung	Pflicht
`nodeset`	Ein XPath-Ausruck, der eine eindeutige Knotenmenge zurückliefert, über die iteriert werden kann.	Ja

Tabelle 12.13 Attribute von <jxf:itemset/>

Es folgt ein Beispiel für die Verwendung des Elements `<jxf:itemset/>`:

```
<jxf:select ref="selection">
   <jxf:label>Kategorien</jxf:label>
   <jxf:itemset nodeset="/product/colours">
      <jxf:label ref="./label "/>
      <jxf:value ref="./value "/>
```

```
   </jxf:itemset>
</jxf:select>
```

Listing 12.28 Ein Beispiel für <jxf:itemset/>

<jxf:violations/>

Hiermit wird innerhalb des JXForms ein Set an Fehlermeldungen definiert, das alle Fehlermeldungen enthält, die sich auf ein Formularelement beziehen. Jede einzelne Fehlermeldung ist wiederum in ein Element `<jxf:violation/>` eingefasst. Innerhalb von W3C-XForms gibt es keine Entsprechung für dieses Element.

Attribut	Beschreibung	Pflicht
`class`	Dieses Attribut kann verwendet werden, um das Element über ein Stylesheet zu formatieren.	Nein

Tabelle 12.14 Attribute von <jxf:violations/>

<jxf :violation/>

Ein Element `<jxf:violation/>` entspricht einer einzelnen Fehlermeldung innerhalb von `<jxf:violations/>`. Innerhalb von W3C-XForms gibt es keine Entsprechung für dieses Element. Angenommen, Sie besitzen eine Schematron-Validierung, die eine Überprüfung des Wertebereichs eines eingegebenen Wertes vornimmt, wie die nachfolgend gezeigte:

```
...
<rule context="/number">
   <assert test="number() &gt; 10">
   Die angegebene Nummer muß größer als 10 sein.
   </assert>
</rule>
...
```

Listing 12.29 Schematron-Validierung

Des Weiteren haben Sie ein JXForms-Element definiert, das ein Element `<jxf:violations/>` wie das folgende enthält:

```
...
<jxf:input ref="/number">
   <jxf:label>Nummer</jxf:label>

   <jxf:violations class="error"/>
```

```
</jxf:input>
...
```

Listing 12.30 Ein JXForms-Element

Das Formular würde somit nach der Bearbeitung durch den JXForms Generator in folgende XML-Struktur umgewandelt werden, falls der eingegebene Wert nicht gültig ist:

```
...
<jxf:input ref="/number">
   <jxf:label>Nummer</jxf:label>
   <jxf:violations class="error">
      <jxf:violation>
      Die angegebene Nummer muß größer als 10 sein.
      </jxf:violation>
   </jxf:violations>
</jxf:input>
...
```

Listing 12.31 Ergebnis-XML im Fehlerfall

Somit ist eine eindeutige Bestimmung des Formularelements möglich, in das der fehlerhafte Wert eingegeben wurde.

12.3 Woody

Das mit Abstand leistungsfähigste Formular-Framework, das zur Zeit in Cocoon verfügbar ist, ist Woody. Es bietet neben allen Eigenschaften, die auch JXForms bereitstellt, eine Vielzahl weiterer komfortabler Möglichkeiten, die wir uns in diesem Abschnitt etwas genauer ansehen möchten. Auf den ersten Blick scheint mit dem Leistungsumfang auch die Komplexität dieses Frameworks gegenüber JXForms enorm zugenommen zu haben. Ich kann Sie aber mit gutem Gewissen beruhigen: Woody ist nicht wesentlich aufwendiger zu verwalten als andere Formular-Frameworks. Sobald Sie ein erstes eigenes kleines Beispiel durchgearbeitet haben, werden Sie sehen, wie einfach der Einsatz von Woody sein und wie viel Arbeit es Ihnen abnehmen kann.

Cocoon Forms

Woody wird häufig auch als *Cocoon Forms* bezeichnet. Unter diesem Namen wird es auch in naher Zukunft das offizielle Standard-Framework für die Formularverarbeitung unter Cocoon werden.

Woody ist Beta!

Beachten Sie, dass sich Woody zur Zeit[7] immer noch im Beta-Stadium befindet. Aus diesem Grund sollten Sie darauf gefasst sein, dass sich im

7 Zum Zeitpunkt, als an diesem Kapitel geschrieben wurde

M Anhang: Wichtige Quellen

M.1 Cocoon

- **http://cocoon.apache.org**
  Die offizielle Website des Cocoon-Projekts. Hier finden Sie einige der wichtigsten Informationen rund um Cocoon. Außerdem können Sie über diese Website die aktuelle wie auch ältere Cocoon-Versionen herunterladen.
- **http://cocoon.apache.org/2.1/userdocs/index.html**
  Der User-Guide der Cocoon-Websites. Dieser Teil enthält wichtige Informationen für alle Anwender von Cocoon. Neben einer tiefer gehenden Erklärung der wichtigsten Sitemap-Komponenten erhalten Sie hier Informationen über Control Flow, XSP und die Offline-Generierung. Im Bereich »Concepts« wird auf wichtige Themen von Cocoon speziell eingegangen.
- **http://cocoon.apache.org/2.1/apidocs/**
  Unter dieser URI finden Sie immer die aktuelle Apidoc zu allen Java-Klassen von Cocoon.
- **http://wiki.cocoondev.org**
  Bei dieser Website handelt es sich um eine Wiki-Page, d.h., jeder darf aktiv an dieser Seite mitgestalten. Diese Website enthält in der Regel sehr aktuelle Informationen und wird von den Cocoonern dazu benützt, Informationen und Anleitungen zeitnah zu veröffentlichen. Diese Website sollte zu den ersten Anlaufstellen gehören, wenn Sie weiter führende Fragen zu Cocoon haben.
- **http://www.cocoonforum.de**
  Bei dieser Website handelt es sich um ein deutschsprachiges Forum rund um das Thema Cocoon und alles, was dazu gehört. Diese Seite wurde von mir mit dem Erscheinen dieses Buches ins Leben gerufen.
- **http://www.logabit.com/cocoontutorial**
  Ein deutschsprachiges Cocoon-Tutorial, das von mir vor diesem Buch erstellt wurde. Von Zeit zu Zeit werde ich dieses Tutorial aktualisieren.
- **users@cocoon.apache.org**
  Hierbei handelt es sich um die offizielle Mailing-List für Cocoon-Anwender. Wenn Sie Fragen zur Anwendung von Cocoon haben, können Sie diese hier stellen. Die Liste ist relativ stark frequentiert.

Um sich anzumelden, schicken Sie einfach eine leere E-Mail an folgende Adresse:
users-subscribe@cocoon.apache.org
Um sich von der Liste wieder abzumelden, schicken Sie eine leere E-Mail an diese Adresse:
users-unsubscribe@cocoon.apache.org

- **dev@cocoon.apache.org**
  Diese Mailing-List wird hauptsächlich von den Cocoon-Entwicklern zur Kommunikation und Diskussion verwendet. Sie können hier Verbesserungsvorschläge einbringen oder selbst an bestimmten Diskussionen teilnehmen.
  Um sich anzumelden, schicken Sie einfach eine leere E-Mail an folgende Adresse:
  dev-subscribe@cocoon.apache.org
  Um sich von der Liste wieder abzumelden, schicken Sie eine leere E-Mail an diese Adresse:
  dev-unsubscribe@cocoon.apache.org

M.2 Tomcat

- **http://jakarta.apache.org/tomcat/index.html**
  Die offizielle Website des Tomcat-Projekts. Hier finden Sie zahlreiche Informationen rund um Tomcat. Neben aktuellen Tomcat-Versionen können Sie hier auch ältere Versionen des Servlet-Containers bekommen.
- **http://www.jsp-develop.de**
  Eine deutschsprachige Website rund um das Thema JSP & Co. Im integrierten Forum können Sie Fragen zu Servlets, JSP und den Servlet-Containern stellen, die relativ schnell beantwortet werden.
- **tomcat-user@jakarta.apache.org**
  In dieser stark frequentierten Mailing-List treffen sich Tomcat-Anwender, um Erfahrungen auszutauschen und Problemlösungen zu erhalten.
  Um sich anzumelden, schicken Sie einfach eine leere E-Mail an folgende Adresse:
  tomcat-user-subscribe@jakarta.apache.org
  Um sich von der Liste wieder abzumelden, schicken Sie eine leere E-Mail an diese Adresse:
  tomcat-user-unsubscribe@jakarta.apache.org

- **tomcat-dev@jakarta.apache.org**
  Diese Mailing-List wird hauptsächlich von den Tomcat-Entwicklern zur Kommunikation und Diskussion verwendet. Sie können hier Verbesserungsvorschläge einbringen oder selbst an bestimmten Diskussionen teilnehmen.
  Um sich anzumelden, schicken Sie einfach eine leere E-Mail an folgende Adresse:
  tomcat-dev-subscribe@jakarta.apache.org
  Um sich von der Liste wieder abzumelden, schicken Sie eine leere E-Mail an diese Adresse:
  tomcat-dev-unsubscribe@jakarta.apache.org

M.3 Avalon

- **http://avalon.apache.org**
  Die offizielle Website des Avalon-Projekts. Auf dieser Website finden Sie zahlreiche Informationen zu Avalon selbst sowie zu seinen Subprojekten, wie beispielsweise Excalibur oder LogKit. Ein Download der jeweiligen Sourcen ist ebenso verfügbar.
- **http://avalon.apache.org/framework/api/index.html**
  Die Apidoc des Avalon-Frameworks. Diese Dokumentation wird häufig dann benötigt, wenn in Cocoon eigene Avalon-Komponenten integriert werden sollen.
- **users@avalon.apache.org**
  In dieser Mailing-List treffen sich Avalon-Anwender, um Erfahrungen auszutauschen und Problemlösungen zu erhalten.
  Um sich anzumelden, schicken Sie einfach eine leere E-Mail an folgende Adresse:
  users-subscribe@avalon.apache.org
  Um sich von der Liste wieder abzumelden, schicken Sie eine leere E-Mail an diese Adresse:
  users-unsubscribe@avalon.apache.org
- **dev@ avalon.apache.org**
  Diese Mailing-List wird hauptsächlich von den Avalon-Entwicklern zur Kommunikation und Diskussion verwendet. Sie können hier Verbesserungsvorschläge einbringen oder selbst an bestimmten Diskussionen teilnehmen.
  Um sich anzumelden, schicken Sie einfach eine leere E-Mail an folgende Adresse:
  dev-subscribe@avalon.apache.org

Um sich von der Liste wieder abzumelden, schicken Sie eine leere E-Mail an diese Adresse:
dev-unsubscribe@avalon.apache.org

M.4 XML & Co.

- **http://www.w3.org**
  Dies ist die offizielle Website des W3-Konsortiums. Sie sollte zu den ersten Anlaufstellen für das Nachschlagen von bestimmten Themen rund um XML gehören.
- **http://www.w3schools.com**
  Diese Website enthält zahlreiche vortreffliche Tutorials zu den Themen XML, XSL, XSL-FO, XPath und vielen mehr.
- **http://www.saxproject.org**
  Die offizielle Website von SAX. Hier erhalten Sie zahlreiche Informationen rund um die »Simple API for XML«.
- **http://xml.apache.org**
  Die offizielle Website des Apache-XML-Projects. Auf dieser Website finden Sie viele Programme für die Verarbeitung von XML. Unter anderem werden hier auch der XML-Parser Xerces und der XSL-Prozessor Xalan zum Download angeboten.
- **http://saxon.sourceforge.net**
  Die offizielle Website des XSL-Prozessors SAXON. Hier erhalten Sie umfangreiche Informationen zu SAXON und können das Programm herunterladen.
- **http://www.zvon.org/index.php?nav_id=references&mime=html**
  Auf dieser Website finden Sie zahlreiche Referenzen und Tutorials zu den verschiedensten Themen. Vor allem die Referenzen zu XSLT, XSL-FO oder Regulären Ausdrücken sollten Sie unbedingt in Ihre Bookmarks aufnehmen.

M.5 Mail-Archive

- **http://www.mail-archive.com**
  Auf dieser Website finden Sie ein durchsuchbares Mail-Archiv, das unter anderem fast alle Mailing-Lists der Apache Software Foundation regelmäßig archiviert.
- **http://marc.theaimsgroup.com**
  Auch diese Website archiviert zahlreiche Mailing-Lists, die anschließend komfortabel durchsucht werden können.

M.6 Sonstige

- **http://www.cvshome.org**
  Die offizielle Website des CVS-Projekts. Hier können Sie unter anderem eine umfangreiche Dokumentation zu diesem System sowie Links zu verschiedenen CVS-Tools bekommen.
- **http://www.wikipedia.de**
  Dieses Internet-Lexikon enthält Antworten auf zahlreiche Fragen, auch aus dem EDV-Bereich.

N Inhalt der CD-ROM

Cocoon 2.0.4 und 2.1.4

Neben der Source-Version 2.1.4 befinden sich auf der Buch-CD zusätzlich eine vorkompillierte Version mit allen Blocks und Beispielen sowie die ältere Version 2.0.4, in der das Konzept des Control Flow noch nicht verfügbar war, die aber noch relativ weit verbreitet ist.

http://cocoon.apache.org

Tomcat 4.1.30 und 5.0.19

Der Servlet-Container der Apache Jakarta Group in den Versionen 4 und 5. Die Version 4.x implementiert die Servlet 2.3 und JSP 1.2 Spezifikationen, Version 5 hingegen die Servlet 2.4 und JSP 2.0 Spezifikationen. Auf der Buch-CD befinden sich sowohl die gepackten Zip- und Tar-Versionen als auch eine EXE-Version für eine grafische Installation unter Windows.

http://jakarta.apache.org/tomcat/

TortoiseCVS 1.5.4

Ein komfortabler CVS-Client für Windows, der sich über das Kontextmenü bedienen lässt.

http://www.tortoisecvs.org

WinCVS 1.2

Ein weit verbreiteter, grafischer CVS-Client für Windows.

http://www.wincvs.org

Apache HTTP-Server 1.3.27 und 2.0.48

Der am weitesten verbreitete HTTP-Server der Welt. Auf der Buch-CD befinden sich Versionen sowohl für Linux, als auch für Windows, sowie entsprechende Jk-Module für die Kommunikation mit dem Servlet-Container Tomcat.

http://httpd.apache.org/

MySQL 4.0

Eine weit verbreitete, schnelle Datenbank. Inklusive eines JDBC-Treibers für Java.

http://www.mysql.com

Xindice 1.0

Eine XML-Datenbank der Apache XML Group.

http://xml.apache.org/xindice/

jEdit 4.1

Ein komfortabler, auf Java basierender Editor.

http://www.jedit.org

syn 2.1

Ein kleiner, aber äußerst komfortabler Editor für das Betriebssystem Windows.

http://syn.sourceforge.net/

Eclipse 2.1.2

Eine umfangreiche und komfortable Entwicklungsumgebung für Windows und Linux mit einem vorbildlichen Plug-In-Mechanismus.

http://www.eclipse.org

NetBeans 3.5.1

Eine vollständige und komfortable Entwicklungsumgebung mit sehr guter Unterstützung von J2EE- und grafischen Anwendungen.

http://www.netbeans.org

Java 2 Software Developer Kit (J2SDK) 1.4.2

Das, von der Firma Sun Microsystems zur Verfügung gestellte Entwickler-Kit. Es umfasst neben der Java Virtual Machine den Java-Compiler Javac, eine umfangreiche Sammlung an verschiedenen Apis und andere nützliche Tools für die Entwicklung von Java-Anwendungen.

http://java.sun.com/

XAMPP 1.3

Ein Komplettpaket, das den Apache HTTP-Server, MySQL, PHP, Perl uvm. enthält. Ideal für die Entwicklung: einfach entpacken, starten und loslegen

http://www.apachefriends.org

FOP 0.20.5

Der Formatting Objects Processor (FOP) ist ein in Java geschriebenes Tool der Apache XML Group, um XML in Formate, wie beispielsweise PDF, PS, PCL oder SVG zu wandeln.

http://xml.apache.org/fop/

SAXON 6.5.3 und 7.9

Ein weit verbreiteter Open Source-XSLT-Prozessor von Michael Kay für Java.

http://saxon.sourceforge.net/

Xalan 2.6.0

Ein XSLT-Prozessor der Apache XML Group für Java.

http://xml.apache.org/xalan-j/

Xerces 2.6.2

Einer der bekanntesten XML-Parser für Java. Er wird von der Apache XML Group betreut.

http://xml.apache.org/xerces2-j

Index

C

D

G

H

K

L

M

N

O

P

R

S

T

U

V

W

X

Z