

1 Einleitung

Entwicklung und Pflege von Java-Webanwendungen stellen hohe Ansprüche an alle Beteiligten: Webdesigner müssen sich täglich mit den Launen der Webbrowser bezüglich der Interpretation von HTML, CSS und JavaScript auseinandersetzen. Von den Java-Entwicklern wird erwartet, dass sie einerseits Java beherrschen, aber andererseits auch fit sind in JavaScript, HTML und mehreren XML-Dialekten. In vielen Projekten muss der Java-Entwickler sogar die Aufgaben des Webdesigns mit übernehmen.

Erschwerend kommt hinzu, dass die Basistechnologien des World Wide Web – HTTP und HTML – nie für die Umsetzung interaktiver Benutzeranwendungen gedacht waren. Seit Jahren versucht man mit Frameworks aus Open-Source-Initiativen, durch den Java Community Process sowie über kommerzielle Produkte, die Unzulänglichkeiten der Webtechnologien vor dem Entwickler zu verbergen, und verspricht ihm stattdessen mächtige und zugleich benutzerfreundliche Programmiermodelle.

Dieses Buch zeigt in der Einleitung, welche Rolle das Open-Source-Webframework Wicket dabei spielt und wie es sich gegenüber Mainstreamlösungen positioniert. Wir werden dazu einen Überblick über die wichtigsten Features von Wicket geben.

Wir wollen außerdem erläutern, was uns bewegt hat, zum Thema Wicket ein Buch zu verfassen, und welchen Leserkreis wir damit ansprechen wollen. Wir werden den Leserkreis in Zielgruppen abhängig von persönlichen Vorkenntnissen aufteilen und für jede Zielgruppe einen Leseleitfaden aufzeigen.

Am Ende dieser Einleitung sollte der Leser so viel über das Buch wissen, dass er entscheiden kann, ob sich die weitere Lektüre für ihn lohnt.

1.1 Wicket im Dschungel der Java-Webframeworks

Kaum ein Thema in der Entwicklung von Java-Webanwendungen wird zurzeit so heiß diskutiert wie die Frage, welches Webframework die beste Wahl ist. Allein unter der Webadresse <http://java-source.net/open-source/web-frameworks> werden über fünfzig Open-Source-Frameworks aufgelistet, unter denen sich der Entwickler nach Belieben bedienen kann. Daneben gibt es auch noch etliche kommerzielle Alternativen.

Warum wird so viel Energie in die Entwicklung von Webframeworks gesteckt? Gibt es keine einzelne Lösung, die leistungsfähig genug ist, um die meisten Bedürfnisse zu befriedigen, und die am besten noch standardisiert ist?

1.1.1 Historische Entwicklung

Betrachten wir die historische Entwicklung der Java-Webtechnologien, um die heutige Situation besser verstehen zu können.

Die Servlet-API

Schon lange bevor die erste Version der Java-Enterprise-Plattform definiert war, hatte Sun mit der Einführung der Servlet-API 1.0 im Jahre 1997 den Weg zur Entwicklung von dynamischen Webinhalten in Java bereitet. Das Servlet-Prinzip ist leicht zu verstehen, wenn man sich ein wenig mit den Grundlagen des HTTP-Protokolls auseinandergesetzt hat: Anfragen von Clients werden von einer Servlet-Engine entgegengenommen. Jede HTTP-Anfrage wird in ein Java-Objekt vom Typ `HttpServletRequest` verpackt und über einen URL-Abbildungsmechanismus an ein Servlet delegiert. Dabei wird die Methode `service(...)` einer `HttpServletRequest`-Instanz aufgerufen. Das Servlet führt serverseitige Logik durch und schreibt die Clientantwort (meist in HTML) in das Containerobjekt vom Typ `HttpServletResponse`. Die Servlet-Engine erzeugt aus dem Antwortcontainer die HTTP-Antwort und liefert sie an den Client zurück.

JavaServer Pages

Auch wenn das Programmiermodell eingängig ist, erregte es erst größere Aufmerksamkeit, nachdem Sun zwei Jahre später (1999) die Servlet-Spezifikation 2.1 veröffentlichte und im gleichen Zug mit *JavaServer Pages* (JSP) eine Technologie einführte, mit deren Hilfe man HTML-Seiten auf Basis von Vorlagen erzeugen konnte. Man hatte offensichtlich erkannt, dass die Generierung von Markup in Java weder zur Übersichtlichkeit noch zur Wartbarkeit des Codes beitrug. *JavaServer Pages* erlauben zur Erzeugung einer Seite die Vermischung von regulärem, statischem HTML mit dynamisch generierten Inhalten durch

Java-Code. Mit anderen Worten: Über JavaServer Pages erreicht man im Gegensatz zu Servlets eine Trennung von Darstellung und Inhalt.

Jetzt standen mit Servlets und JSP gleich zwei Ansätze zur Erstellung von Anwendungsseiten bereit. Sun hatte sich bereits in Vorabversionen der JSP-Spezifikation 1.0 darüber Gedanken gemacht, wie die Rollenverteilung zwischen diesen Technologien aussehen kann. Ein Vorschlag, den sie Model 2 nannten, ist bis heute Grundlage vieler Webframeworks (siehe Abbildung 1-1). Die Idee ist, beide Ansätze so in die Request-Verarbeitung einzubeziehen, dass jeder für sich seine Stärken ausspielen kann. Die Stärke von Servlets liegt klar in der Ausführung Businesslogik-lastiger Aufgaben. JSP-Seiten dagegen eignen sich am besten für das Rendern von Anwendungsseiten.

Die MVC-Variante
Model 2

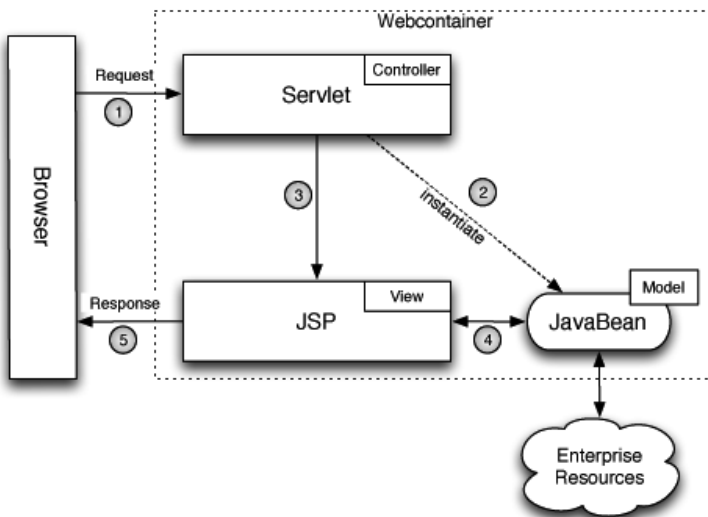


Abb. 1-1
Model 2 zur
Strukturierung der
Anwendung

Eine ausführliche Diskussion des Model-2-Anwendungsmusters kann in Abschnitt A.3.3 nachgelesen werden.

Zu Model 2 gesellte sich mit Front-Controller¹ bald ein weiteres Architekturmuster, das dabei helfen sollte, technisch motivierte Controller-Logik und die Einbindung Request-übergreifend benötigter Dienste zu zentralisieren. Beim Front-Controller-Muster werden alle Requests durch ein einzelnes Servlet – den Front-Controller – geleitet. Das Front-Controller-Servlet bindet zentrale Dienste ein und delegiert an Application-Controller-Instanzen zur Ausführung der anwendungsspezifischen Logik. Zentrale Dienste decken beispielsweise die Berei-

Front-Controller

¹<http://java.sun.com/blueprints/corej2eepatterns/Patterns/FrontController.html>

che Internationalisierung, Logging, Authentifizierung oder Autorisierung ab.

*Struts – der
Dinosaurier unter den
Webframeworks*

Front-Controller und Model 2 bilden die Grundlage für viele Webframeworks. Struts war eines der ersten Frameworks, das diese Architektur nahezu unverändert umgesetzt hat und lange Zeit als De-facto-Standard für die Entwicklung von JEE-Webanwendungen galt. Auch heute findet man in sehr vielen Firmen noch Struts-Anwendungen, die weiter gepflegt oder migriert werden müssen.

Struts wies als Framework von Anfang an gravierende Designschwächen auf, die man aber lange Zeit mangels Alternativen in Kauf nahm. In Struts2 wurden einige dieser Probleme gelöst, wodurch jetzt z. B. der Front-Controller (der als eine Interceptor-Chain realisiert wird) den eigenen Wünschen entsprechend angepasst werden kann. Jedoch kamen die lange ersehnten Verbesserungen mit Struts2 zu spät, sodass sich zwischenzeitlich Alternativen etablieren konnten, die wesentlich durchdachter waren. Erwähnenswert hierzu ist beispielsweise Spring MVC, das sich neben einer sauberen Model-2-Umsetzung auch perfekt in einen Spring-basierten Business-Layer integrieren lässt.

1.1.2 Der Weg zu komponentenbasierten Frameworks

Mit wachsender UI²-Komplexität und Größe einer Webanwendung stellt man fest, dass eine Model-2-Anwendungsarchitektur häufig nicht den erwarteten Mehrwert bietet. Andere Probleme treten dann in den Vordergrund, für die Frameworks wie Struts oder Spring MVC kaum Lösungshilfen anbieten:

Model-2-Frameworks besitzen gegenüber HTTP und der Servlet-API nur eine dünne Abstraktionsschicht. Unwegsamkeiten des HTTP-Protokolls und Eigenschaften des Browsers als Clientplattform wirken sich bis in den Anwendungscode aus. Hier sind einige prominente Beispiele, die eine Behandlung im Anwendungscode erforderlich machen:

*Typische Probleme von
Webanwendungen*

- Die aktuelle Seite wird über die Browserfunktion Page-Reload neu geladen. Mit dem vorangegangenen Request wurde der Inhalt eines Formulars abgesendet. Ohne zusätzliche Maßnahmen würden die Formulardaten nochmals gesendet werden – eine unerwünschte Situation, wenn man z. B. mit dem letzten Request eine Banktransaktion abgeschlossen hat. Vergleichbare Szenarien sind die Browser-Back-Funktion oder das Öffnen eines weiteren Browserfensters.
- Das manuelle Codieren und Decodieren von URLs ist aufwendig. Fehlerhafte URLs werden oft nur zur Laufzeit erkannt.

²User-Interface

- Hauptaugenmerk fast jeder Webanwendung ist die Verwaltung des Anwendungszustandes. Der Anwendungszustand gliedert sich in Request-bezogene, benutzerbezogene oder anwendungsglobale Informationen, für deren Verwaltung die Container `HttpRequest`, `HttpSession` und `ServletContext` durch die Servlet-API bereitgestellt werden. Diese Container sind wie normale Hashmaps nicht typsicher. Das heißt, Fehler durch falsches Casten werden erst zur Laufzeit aufgedeckt. Die Anwendung ist für den Lebenszyklus der Containerobjekte selbst zuständig. Bezüglich der Sessiongröße, d. h. der Datenmenge in `HttpSession`, müssen häufig Grenzen gesetzt werden, wenn man eine große Anzahl paralleler User erlauben möchte. Mit zunehmender Größe der Benutzersession erhöht sich außerdem die zu replizierende Datenmenge beim Betrieb in einem Cluster. Die Verwaltung der Sessiondaten in `HttpSession` erfordert deshalb besondere Aufmerksamkeit durch den Anwendungsentwickler.

Komplexe Weboberflächen, die nicht nur aus einfachen Formularen, Links oder Tabellen bestehen, erfordern unverhältnismäßig hohen Entwicklungsaufwand. Diese Einschätzung teilen vor allem erfahrene Desktop-Anwendungsentwickler, die mit Frameworks wie Swing oder SWT wesentlich effizienter Benutzeroberflächen erstellen können.

Selbst wenn man den hohen Entwicklungsaufwand in Kauf nimmt, sollte man wenigstens erwarten können, dass man das Ergebnis seiner Bemühungen wiederverwenden kann – idealerweise als Blackbox verpackt mit funktionaler Aufrufsemantik. Das ist leider nicht der Fall. Eine auf die Granularität von (Web-)Seiten beschränkte Architektur wie Model 2 ist dazu wenig geeignet.

Diese und andere Erkenntnisse haben die Mitglieder des Java Community Process (JCP) dazu bewogen, über den JSR 127 mit *JavaServer Faces (JSF)* eine komponentenbasierte Architektur und API für Java-Webanwendungen zu definieren, die fortan den Standard in der Entwicklung dieser Anwendungsklasse setzen sollte.

JavaServer Faces

Als Kurzfassung lässt sich die JSF-Architektur so beschreiben: UI-Komponenten werden in einem Baum angeordnet. Benutzeraktionen werden auf serverseitige Ereignisse (Event) abgebildet und von Event-Handlern verarbeitet. Komponenten können sich selbst rendern und ihren Zustand bei Bedarf sichern und wiederherstellen. Die Request-Verarbeitung geschieht nach einem fest definierten Phasenmodell, das in jeder Phase Einhängpunkte für anwendungsspezifische Erweiterungen bietet.

Auch wenn mit jeder Spezifikation eine Referenzimplementierung von Sun zur Verfügung gestellt wurde, blieb der Ansturm der Ent-

wicklergemeinde auf JSF lange hinter den Erwartungen zurück. Selbst heute ist man mit der in der Praxis gelebten JSF-Spezifikation 1.2 als Teil von JEE 5 noch nicht an einem Punkt angekommen, an dem man von einer breiten Akzeptanz sprechen kann.

Wie kommt das? Tatsache ist, dass JSF kein One-Stop-Shopping-System aus dem Blickwinkel vieler Projektvorhaben ist. JSF stellt in erster Linie ein Gerüst mit standardisierter API und Erweiterungspunkten dar, das in den Projekten um weitere Frameworks angereichert wird, um eine praxisgerechte Plattform zu erhalten. Die Erwartungshaltung der Technologie-Entscheider in Unternehmen gegenüber JSF ist aus unserer Erfahrung eine andere. Man erhofft sich von JSF eine fertige Lösung, die sich nicht nur auf die Bereitstellung eines Komponentenmodells konzentriert, sondern viele der täglichen Probleme angeht, so z. B. das Arbeiten in mehreren Fenstern, eine Back-Button-Unterstützung oder Bookmarks von Anwendungsseiten.

JSF soll laut JSR 127 die Entwicklung von Java-Webanwendungen deutlich erleichtern. Komponenten zu schreiben soll ebenso einfach sein, wie bestehende Komponenten wiederzuverwenden. Unserer Ansicht nach trifft es nicht zu, dass Komponenten einfach zu erstellen sind. Dagegen spricht schon, dass im Standardfall zahlreiche Artefakte erstellt bzw. gepflegt werden müssen:

- Komponenteklasse
- HTML-spezifische Komponenteklasse
- Renderer-Klasse
- faces-config.xml
- JSP-Tag-Handler-Klasse
- JSP-Tag-Library-Descriptor

JSF ist klar spezifikationsgetrieben. Das Fähnchen »Standard« wird so hoch gehalten, dass man weitverbreitete Problemfälle aus der Praxis nicht rechtzeitig wahrnimmt und im Standard vernachlässigt. Wir wollen dies an einem aktuellen Beispiel verdeutlichen. Die Aggregation einer Komponente aus bestehenden Komponenten ist einer der häufigsten Gründe, eine Komponente selbst zu schreiben. In JSF muss ein solches Aggregat mühsam in Java-Code erstellt werden. Ein Templating-Mechanismus, der statischen von dynamischem Markup trennt, bei dem die zu aggregierenden Komponenten im Markup deklariert werden, ist hier besser geeignet. In der Entwickler-Community hat man das Problem erkannt und bietet z. B. mit *Facelets* eine Lösung, die das JSP-basierte View-Handling ersetzt. Bedauerlicherweise ist die Facelet-Technologie kein JSR-Standard.

JSF ist seit JEE5 Bestandteil der Java Enterprise Edition. Verbesserungen können nur zu den JEE-Release-Zeitpunkten (etwa alle zwei Jahre) veröffentlicht werden. Das ist unschön, wenn man bedenkt, dass sich gerade im Bereich der Webtechnologien eine rasante Entwicklung vollzieht.

Auch wenn mit der kürzlich verabschiedeten Spezifikation 2.0 dringend gebrauchte Verbesserungen eingeflossen sind, bleibt JSF 2.0 für viele Anwender weiterhin Zukunftsmusik. Jetzt sind erst einmal die Application-Server-Hersteller gefragt, entsprechende Implementierungen zu liefern. Selbst dann werden längst nicht alle Anwender von einer älteren JSF-Version umsteigen können, da gerade Enterprise-Kunden oft eine konservative Haltung in Bezug auf ein Upgrade des Application Server einnehmen.

Alles in allem bietet JSF oft nicht das, was viele davon erwarten. Statt »Ease of Development« und »One-Stop-Shopping« erhält man ein Baukastensystem mit großer Lernkurve. Auf der anderen Seite muss man zugestehen, dass durch die Standardisierung einiges in die Tools und Komponenten rund um JSF investiert wird, wodurch einige seiner Schwächen in der Wahrnehmung abgemildert werden.

1.1.3 Wie Webentwicklung wieder Spaß macht

Abseits des Mainstreams wurde im Jahr 2004 von Jonathan Locke das Open-Source-Webframework Wicket initiiert. Jonathans Vision war es, ein Webframework zu entwickeln, das in der Kategorie der komponentenbasierten Frameworks neue Maßstäbe setzen sollte. Wickets Leitmotiv sollte sein, ein Rahmenwerk bereitzustellen, das intuitives Vorgehen und hohe Effizienz bei der Erstellung von Webanwendungen fördert. Bis dato war man von der Umsetzung dieser Vision durch die Mainstreamlösungen wie Struts und JSF weit entfernt. Wicket fand sehr schnell Beachtung durch die Entwickler-Community. Binnen kurzer Zeit bildete sich ein schlagkräftiges Kernteam, das bis heute zusammen mit einer Vielzahl von Comittern das Projekt vorantreibt. Wicket glänzt heute als Top-Level-Projekt unter dem Dach der Apache Software Foundation durch eine äußerst aktive Community.

Wicket unterstützt einerseits das klassische Modell einer Webanwendung, bei dem die Anwendung aus Webseiten besteht, die mit jedem Benutzer-Request neu ausgeliefert werden und mit konventionellem HTML auskommen. Gleichzeitig bietet es auch das aktuelle Modell einer Rich Internet Application (RIA), das mithilfe von JavaScript clientseitig und per Ajax Desktop-ähnliche User-Interfaces ermöglicht. Das Interessante dabei ist aber, dass in Wicket beide Anwendungsarten über das gleiche Programmiermodell abgebildet werden und die

*Wicket als Top-Level-
Apache-Projekt*

*Gleichzeitige
Unterstützung
klassischer
Webanwendungen und
Rich Internet
Applications*

technische Komplexität von Rich Internet Applications komplett durch Wicket gekapselt wird.

*Leichtgewichtiges
Komponentenmodell
ähnlich zu Swing*

Im Mittelpunkt der Architektur von Wicket steht ein leichtgewichtiger Komponentenansatz. Wicket orientiert sich dabei grundsätzlich am Komponentenmodell von Swing, einem Modell zur Entwicklung von Desktop-Anwendungen in Java. Dies hat zwei entscheidende Vorteile: Man nutzt ein lange bewährtes und praxiserprobtes Modell und kann gleichzeitig Swing-erfahrenen Entwicklern den Einstieg in die Welt der Webanwendungen erleichtern. Wicket schafft es, ein zustandbehaftetes Komponentenmodell aus dem Desktop-Bereich mit den beschränkten Mitteln des Web, insbesondere dem HTTP-Protokoll, abzubilden, ohne gleichzeitig den Weg in die native Welt der Webentwicklung zu versperren.

*Virtueller
Anwendungsspeicher*

Wicket stellt jedem Benutzer der Anwendung virtuellen Anwendungsspeicher bereit, dessen Lebenszyklus durch das Framework verwaltet wird. Vorbei sind die Zeiten, in denen man Ad-hoc-Objekte in der Websession selbst verwaltete. Zustand wird in Form von Attributen an Komponenten- und Seitenklassen geheftet. Der Anwendungsentwickler muss sich keine Gedanken mehr darüber machen, ob und wie er Daten in welchen Kontexten der Servlet-API verwaltet, das Framework nimmt ihm diese Aufgabe ab.

*Java im Fokus der
Entwicklung*

Mit Wicket steht Java im Fokus der Entwicklungstätigkeit. Seiten und Komponenten werden eins zu eins durch Java-Klassen abgebildet und mit Java-Mitteln zueinander in Beziehung gesetzt. Die Anwendung ist selbst für die Instanziierung von Komponenten verantwortlich. Im Vergleich zu JSF, wo Komponenten über Tags in JSP-Seiten deklariert werden, hat dieser Ansatz entscheidende Vorteile:

- Man kann zum einen genau bestimmen, in welcher Phase der Request-Verarbeitung Komponenten erzeugt werden sollen.
- Die Komponentenschnittstellen werden nicht auf Umwegen, sondern über Konstruktoren und Java-Properties der Komponentenklassen definiert.
- Schlüsseltechniken der objektorientierten Programmierung wie Vererbung und Komposition können voll ausgenutzt werden.
- Man erfährt bereits zur Compile-Zeit, ob die Komponentenschnittstellen korrekt bedient werden (*static typing*).

Die Fokussierung auf Java hat noch den angenehmen Nebeneffekt, dass man von den Java-Refactoring-Möglichkeiten der Entwicklungsumgebung profitiert und keine zusätzliche Toolunterstützung wie Eingabehilfen oder Validierung für spezielle frameworkrelevante Artefakte benötigt.

Mit folgendem Beispiel wollen wir demonstrieren, wie angenehm der puristische Java-Ansatz für den Entwickler ist. Es zeigt, dass clientseitiges Verhalten über Java-Klassen gekapselt werden kann, so dass konventionelle Komponenten durch «Rich»-Eigenschaften ausgebaut werden können, ohne dass der Anwendungsentwickler selbst in die JavaScript-Trickkiste greifen muss.

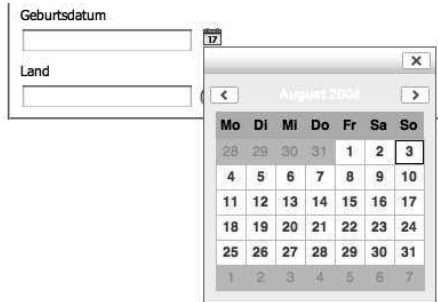


Abb. 1-2
Datumseingabe
ergänzt um
Kalender-Widget

Ein Texteingabefeld zur Datumseingabe soll um einen interaktiven Kalender ergänzt werden (siehe Abbildung 1-2). Das Kalender-Widget ist Bestandteil von Yahoos User Interface Library (YUI), einem JavaScript-Framework, das Widgets zum Entwickeln von Rich Internet Applications bereitstellt:

```
DateTextField birthday =
    new DateTextField("birthday", "dd.MM.yyyy");
birthday.add(new DatePicker());
```

Zur Eingabe des Geburtsdatums wird die Komponente `DateTextField` verwendet. Die Klasse `DatePicker` kapselt die Funktionalität des Kalender-Icons mit dem Kalender-Widget als Popup. Die Kalenderfunktionalität wird alleine durch die Codezeile `birthday.add(new DatePicker())` hinzugefügt. An keiner Stelle im Anwendungscode muss JavaScript eingebunden oder aufgerufen werden.

Über XHTML-Templates werden Layout und statischer Inhalt der Anwendungsseiten definiert. Markierungspunkte in Form von speziellen HTML-Attributen weisen das Framework an, zur Laufzeit dort Wicket-Komponenten einzufügen. XHTML-Templates beinhalten keine UI-Logik, wie man sie z. B. von JavaServer Pages her kennt, etwa zum Iterieren von Tabelleninhalten über Custom-Tags und EL-Ausdrücke. In Wicket wird UI-Logik alleine in Java abgebildet. Die Templates bleiben trotz der Markierungen für Komponenten reines (X)HTML und können von Webdesignern zu jeder Phase der Entwicklung mit ihren gewohnten Tools angepasst werden. XHTML-Templates

*XHTML-Templates
ohne UI-Logik*

*Parallele Entwicklung
von Webdesign und
UI-Logik*

unterstützen damit auf natürliche Weise die parallele Entwicklung des Webdesigns und der UI-Logik.

Die Fokussierung auf Java geht einher mit dem Ziel, auch Art und Anzahl sonstiger Artefakte zu minimieren. Nach der Idee der Wicket-Framework-Entwickler enthält eine Wicket-Anwendung neben Java im Wesentlichen folgende Artefakte:

*Minimierung von
Nicht-Java-Artefakten*

- Artefakte der Servlet-Spezifikation, z. B. web.xml
- XHTML-Templates
- CSS-Dateien
- JavaScript-Dateien
- Resource-Bundles

Auf klassische Konfigurationsdateien wie XML- und Property-Dateien zur Anwendungs- und Framework-Parametrierung wird bewusst verzichtet. Natürlich sprechen wir hier nicht von Parametern zur Konfiguration von Businessregeln oder zur Anpassung an die Betriebsumgebung. Diese haben auch unter Wicket ihre Daseinsberechtigung.

Der Trend weg vom intensiven Gebrauch von Konfigurationsdateien wurde mit dem von Ruby on Rails stammenden Paradigma »Convention over Configuration« eingeläutet. Dies wurde auch von Wicket übernommen, indem beispielsweise die Verknüpfung von zusammengehörigen Ressourcen über Namenskonvention erfolgt. Eine typische Wicket-Seite besteht z. B. aus folgenden Ressourcen:

```
LoginPage.java           // Seitenklasse
LoginPage.html           // Seitenvorlage (Template)
LoginPage.properties     // Texte und
LoginPage_de.properties // Fehlermeldungen
```

Solange alle Ressourcen im gleichen Klassenpfad liegen, kann Wicket diese zueinander in Beziehung setzen.

Ein wunder Punkt vieler Frameworkalternativen ist, dass im Fehlerfall unzureichend über die Fehlerursache berichtet wird. Anwendungsfehler, die sich mit der Seite »Error 500 – Internal Server Error« bemerkbar machen und den Entwickler zu einer aufwendigen Suche nach der Ursache verleiten, gehören mit Wicket der Vergangenheit an. Aus Wicket gemeldete Fehler bezeichnen immer Ursache mit Ortsbezug. Abbildung 1-3 zeigt z. B., wie ein Fehler in der Seitenvorlage gemeldet wird.

*Leistungsfähige
Fehlerdiagnose*

1.1.4 Wicket-Community

Der Erfolg eines Open-Source-Projekts steht und fällt mit der dahinter stehenden Community bestehend aus den Frameworkentwicklern und

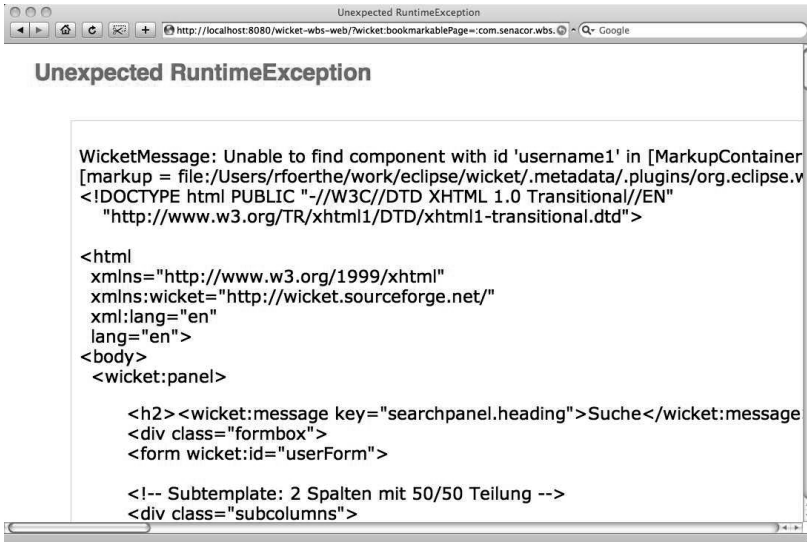


Abb. 1-3
Hilfe bei der
Fehlerdiagnose

-anwendern. Die Wicket-Community zeichnet sich durch ein sehr engagiertes Kernteam aus ca. 20 Entwicklern aus, von denen etwa die Hälfte seit Projektstart aktiv dabei sind. Zusammen mit den Wicket-Interessierten, die sich in den Wicket-Mailinglisten rege beteiligen, kann man von einer sehr aktiven Community sprechen. Das belegt zum Beispiel die Statistik des Mailinglisten-Servers <http://www.nabble.com/>, auf dem Wicket regelmäßig unter den Top 10 der aktivsten Mailinglisten im Bereich der Java-Projekte geführt wird. In der Praxis bedeutet das, dass eine Support-Anfrage in einer der Mailinglisten in der Regel innerhalb weniger Stunden kompetent beantwortet wird.

Die Wicket-Community sorgt für zuverlässige Releasezyklen. Wicket-Major-Releases erscheinen ca. alle zwei Jahre, wobei darauf geachtet wird, dass eine Migration auf das nächste Release mit vertretbarem Aufwand möglich ist. Die in anderen Open-Source-Webframeworks bisweilen anzutreffende »Innovationswut« beim Entwurf neuer Releases ist hier nicht gegeben. Das spricht auch für die Tragfähigkeit der Wicket-Architektur, die sich im Kern seit dem ersten Release im Jahr 2005 kaum verändert hat.

1.2 Warum dieses Buch?

Lohnt es sich überhaupt, ein Buch über ein Webframework zu schreiben, das nicht zu den Key Playern im Markt zählt? Wird das Interesse an Wicket nach einem kurzen Hype nicht wieder schnell abklingen? Wir sind uns sicher, dass Wicket sowohl technisch als auch als Open-

Source-Projekt mit seiner starken Community überzeugt und gerade im Vergleich zu den etablierten Frameworks wie Struts und JSF viele Vorzüge bietet. Leider hat sich in der Vergangenheit des Öfteren gezeigt, dass ein technisch überzeugendes Programmiermodell noch lange kein Garant für die Akzeptanz eines Frameworks ist. Ähnlich verhält es sich mit einer starken Community, wenn diese es nicht schafft, die Vision des Frameworks zu verbreiten.

Wir bekommen von unseren Kunden immer wieder die gleichen Fragen gestellt, wenn wir den Einsatz einer Open-Source-Technologie vorschlagen: Gibt es gute Dokumentation zu dem Thema? Gibt es genügend Know-how im Markt? Letztendlich sind es die richtigen Antworten auf diese Fragen, die oft mehr wiegen als die technische Brillanz einer Technologie.

Die als Wiki gepflegte Onlinedokumentation des Wicket-Projekts (<http://wicket.apache.org>) bietet zwar viel Information für Einsteiger und Experten. Sie ist aber weitestgehend im How-to- bzw. Referenzstil gehalten, sodass gerade für Einsteiger der »rote Faden« schwer zu erschließen ist.

Genau an dieser Stelle wollen wir mit unserem Buch Abhilfe schaffen. Der Einsatz des Webframeworks Wicket soll in Form einer fundierten Einführung und praktischen Darstellung vorgestellt werden. Gleichzeitig verbinden wir damit die Hoffnung, einen Beitrag zur Verbreitung dieses exzellenten Frameworks leisten zu können.

1.3 Für wen ist das Buch geschrieben?

Zum Verständnis dieses Buches ist es wichtig, dass der Leser vertiefte Kenntnisse in der objektorientierten Entwicklung mit Java besitzt. Innerhalb von Java gibt es allerdings Themen, mit denen man auch als erfahrener Java-Entwickler seltener in Berührung kommt. Bei der Entwicklung mit Wicket spielen z. B. innere Klassen und anonyme Klassen eine besondere Rolle. Wir werden deshalb auf die entsprechenden Java-Grundlagen bei Bedarf eingehen. Zusätzlich werden fundierte HTML-Kenntnisse und Grundkenntnisse in CSS und JavaScript vorausgesetzt. Zum Erwerb bzw. zur Auffrischung dieser Kenntnisse verweisen wir gerne auf <http://de.se1fhtml.org/>.

Als Hauptzielgruppe sehen wir Webentwickler, die von ihrem gewohnten Webframework auf eine leistungsfähigere Alternative umsteigen wollen. Aus eigener Erfahrung wissen wir, dass viele dieser Entwickler durch ihr bisheriges Framework leidgeprüft sind. Wir sprechen hier insbesondere von den Struts-Entwicklern. Auf der anderen Seite wollen wir mit diesem Buch auch diejenigen wachrütteln, die sich hin-

*Zielgruppe:
Java-Webentwickler*

ter dem Standpunkt verstecken, Webentwicklung sei per se kompliziert und aufwendig³.

Wir wenden uns mit diesem Buch außerdem an Swing- oder SWT⁴-Entwickler, die eine Plattform für Webanwendungen suchen, mit der sie schnell und nachhaltig produktiv sein können. Gerade die Verwandtschaft des Wicket-Programmiermodells mit Swing sollte dieser Lesergruppe entgegenkommen.

Zielgruppe: Swing- oder SWT-Entwickler

Als dritte Zielgruppe sehen wir Java-Entwickler, die noch keine Erfahrung in der Entwicklung von Clientanwendungen besitzen. In der Regel sind das Entwickler, die zumindest den JEE-Entwicklungsstack ziemlich gut kennen, was wir aber nicht zwingend voraussetzen.

Zielgruppe: erfahrene Java-Entwickler

1.4 Was kann das Buch nicht leisten?

Mit dem Ziel einer fundierten Einführung in die Programmierung mit Wicket ist es leider nicht möglich, alle Aspekte von Wicket abzudecken. Wir beschränken uns auf Themen, von denen wir der Ansicht sind, dass sie die meiste Praxisrelevanz besitzen. Sicher wird der eine oder andere eine Wicket-Komponente vermissen, die wir hier nicht behandeln können.

Des Weiteren beschränken wir uns beim Thema Backend-Integration auf die Möglichkeiten, die uns das Spring⁵-Framework hier anbietet. Einen Spring-Service-Layer als Mediator zwischen Frontend und Backend einzusetzen, ist unserer Meinung nach eine weitverbreitete Option. Dadurch ist es aus Sicht von Wicket nicht mehr relevant, ob z. B. Webservices, EJBs oder POJOs angesprochen werden.

1.5 Welche Wicket-Version wird betrachtet?

Die Ausführungen und Codebeispiele beziehen sich auf Wicket in der Version 1.4, die kurz vor Drucklegung des Buches vom Wicket-Team freigegeben wurde. Eine zuweilen anzutreffende Firmenstrategie ist es, dass Frameworks erst dann eingesetzt werden, wenn sie eine gewisse Marktreife erreicht haben. Das zu beurteilen, ist als zukünftiger Frameworkanwender allerdings nicht leicht. Man wartet deshalb vor dem Einsatz einfach noch ein paar Minor Releases ab. Sollte das in Ihrem

³Diese Aussage wird vielerorts auch vom Management getragen, wodurch oft der Antrieb in den Entwicklungsabteilungen fehlt, an dieser Situation etwas zu ändern.

⁴SWT ist ein zu Swing alternatives Widget-Toolkit, das von IBM als Basis für Eclipse entwickelt wurde.

⁵<http://www.springsource.org/>