1

Server-Setup und -Konfiguration

- Herunterladen des JDK
- Einen Entwicklungsserver bereitstellen
- Den Server konfigurieren und testen
- HTML- und JSP-Seiten bereitstellen und darauf zugreifen
- Ihre Entwicklungsumgebung einrichten
- Servlets bereitstellen und darauf zugreifen
- Servlet- und JSP-Bereitstellung vereinfachen

Ehe Sie damit beginnen können, besondere Servlet- und JSP-Techniken zu erlernen, müssen Sie über die richtige Software verfügen und wissen, wie sie zu benutzen ist. Dieses einführende Kapitel erläutert, wie kostenlose Versionen der Software, die benötigt wird, um Servlets und JavaServer Pages auszuführen, erworben, konfiguriert, getestet und verwendet werden.

1.1 Download des Java Development Kit (JDK)

Sie haben das JDK vielleicht bereits installiert, aber wenn das nicht der Fall ist, sollte es Ihr erster Schritt sein. Die Version 2.3 des Servlet-APIs und Version 1.2 des JSP-APIs erfordern die Java 2 Plattform (Standard oder Enterprise-Edition). Wenn Sie keine J2EE-Funktionen wie EJB oder JNDI verwenden, empfehle ich, dass Sie die Standard-Edition verwenden, JDK 1.3 oder 1.4.

Für Solaris, Windows und Linux erhalten Sie das JDK 1.3 unter *http://java.sun.com/j2se/1.3/* und das JDK 1.4 unter *http://java.sun.com/j2se/1.4/*. Bei anderen Plattformen sollten Sie zuerst überprüfen, ob eine Java 2-Implementierung vorinstalliert ist, wie es bei Mac OS X der Fall ist. Wenn nicht, finden Sie unter *http://java.sun.com/cgi-bin/java-ports.cgi* eine Liste der Fremdanbieter von Java-Implementierungen.

1.2 Download eines Servers für Ihren Desktop

Ihr zweiter Schritt ist es, einen Server herunterzuladen, der die Java Servlet 2.3- und JSP 1.2-Spezifikationen für die Verwendung auf Ihrem Desktop implementiert. Ich habe normalerweise *zwei* Server auf meinem Desktop installiert (den kostenlosen Tomcat-Server von Apache und einen kommerziellen Server) und teste meine Anwendungen auf beiden, um zu vermeiden, dass ich aus Versehen nicht-portierbare Konstrukte verwende.

Unabhängig von dem Server, den Sie für die endgültige Bereitstellung verwenden werden, sollten Sie für die Entwicklung wenigstens über einen Server *auf Ihrem Desktop* verfügen. Auch wenn sich der Bereitstellungsserver im Büro nebenan befindet und Sie mit ihm über eine Netzwerkverbindung mit Lichtgeschwindigkeit verbunden sind, sollten Sie ihn trotzdem nicht für Ihre Entwicklung verwenden. Sogar ein Testserver in Ihrem Intranet, auf den Kunden nicht zugreifen können, ist für Entwicklungszwecke wesentlich unbequemer als ein Server direkt auf Ihrem Desktop. Die Ausführung eines Entwicklungsservers auf Ihrem Desktop vereinfacht die Entwicklung in vielerlei Hinsicht verglichen mit der Bereitstellung auf einem Remoteserver, und zwar jedes Mal, wenn Sie etwas testen möchten.

1. **Er ist schneller zu testen.** Mit einem Server auf Ihrem Desktop gibt es keine Notwendigkeit, FTP oder ein anderes Upload-Programm zu verwenden. Je schwieriger es für Sie ist, Veränderungen zu testen, desto seltener werden Sie testen. Durch seltenes Testen werden Fehler nicht ausgemerzt, was Sie auf die lange Sicht behindern wird.
2. **Er ist einfach zu debuggen.** Viele Server zeigen die Standardausgabe in einem normalen Fenster an, wenn sie auf Ihrem Desktop laufen. Das steht im Gegensatz zu Bereitstellungsservern, bei denen die Standardausgabe fast immer entweder vollständig verborgen oder nur auf dem Bildschirm des Systemadministrators zu sehen ist. So werden bei einem Desktopserver die einfachen, alten `System.out.println`-Anweisungen nützliche Hilfen zum Überwachen und Debuggen.

3. **Er kann leicht neu gestartet werden.** Sie werden feststellen, dass Sie während der Entwicklung den Server häufig neu starten müssen. Der Server liest z.B. normalerweise nur beim Hochfahren die Datei *web.xml* (siehe Kapitel 4). Also müssen Sie normalerweise den Server jedes Mal neu starten, wenn Sie *web.xml* verändern. Einige Server (z.B. ServletExec) verfügen zwar über eine interaktive Methode, *web.xml* neu zu laden, aber einige Aufgaben, wie das Löschen von Sitzungsdaten, das Zurücksetzen des `ServletContext` oder das Ersetzen von veränderten Klassendateien, die indirekt von Servlets oder JSP-Seiten verwendet werden (z.B. Beans oder Utility-Klassen), können trotzdem einen Neustart des Servers erforderlich machen. Einige ältere Server müssen außerdem neu gestartet werden, da sie das Neuladen von Servlets nicht zuverlässig implementieren. (Normalerweise instanziieren Server die Klasse, die einem Servlet entspricht, nur einmal und speichern die Instanz zwischen den Anfragen im Speicher. Beim *Servlet-Reloading* ersetzt ein Server Servlets, die sich im Speicher befinden, deren Klassendatei sich aber auf der Festplatte geändert hat, automatisch.) Außerdem empfehlen einige Bereitstellungsserver eine vollständige Deaktivierung des Neuladens von Servlets, um die Leistung zu verbessern. Es ist also viel produktiver, in einer Umgebung zu entwickeln, in der Sie den Server mit einem Mausklick neu starten können, ohne erst andere Entwickler, die den Server eventuell auch benutzen, um Erlaubnis fragen zu müssen

4. **Vergleichstests sind verlässlicher.** Es ist sogar unter den besten Umständen schwierig, akkurate Zeitmessungsergebnisse für kurz laufende Programm zu erhalten. Die Durchführung von Vergleichstests auf Systemen, die eine schwere und variierende Systemlast haben, ist aber offenkundig unzuverlässig.

5. **Er befindet sich unter Ihrer Kontrolle.** Als Entwickler sind Sie wahrscheinlich nicht der Administrator des Systems, auf dem der Test- oder Bereitstellungsserver läuft. Eventuell müssen Sie jedes Mal, wenn Sie den Server neu starten möchten, einen Systemadministrator fragen. Oder der Remoteserver ist wegen einer Systemaktualisierung gerade dann ausgeschaltet, wenn Sie an einem kritischen Punkt im Entwicklungszyklus angelangt sind. Das macht keinen Spaß.

Wenn Sie auch noch auf Ihrem Desktop den gleichen Server laufen lassen können, den Sie für die Bereitstellung verwenden, umso besser. Einer der wunderbaren Punkte von Servlets und JSP ist aber, dass Sie es nicht müssen. Sie können auf einem Server entwickeln und mit einem anderen bereitstellen. Im Folgenden finden Sie einige der beliebtesten kostenlosen Optionen für Desktop-Entwicklungsserver. In allen Fällen läuft die kostenlose Version als eigenständiger Webserver. In den meisten Fällen müssen Sie für die Bereitstellungsversion bezahlen, die in einen regulären Webserver wie Microsoft IIS, iPlanet/Netscape oder den Apache Web Server integriert werden kann. Der Leistungsunterschied zwischen der Verwendung eines der Server als Servlet- und JSP-Engine mit einem regulären Webserver und der Verwendung als vollständig eigenständiger Webserver sind aber nicht signifikant genug, als dass sie während der Entwicklung von Bedeutung wären. Unter *http://java.sun.com/products/servlet/industry.html* finden Sie eine vollständigere Liste von Servern.

- **Apache Tomcat.** Tomcat 4 ist die offizielle Referenzimplementierung der Servlet 2.3- und JSP 1.2.-Spezifikationen. Tomcat 3 ist die offizielle Referenzimplementierung für Servlets 2.2 und JSP 1.1. Beide Versionen können während der Entwicklung als eigenständige Server verwendet werden oder können für die Verwendung während der Bereitstellung an einen Standardwebser-

ver angeschlossen werden. Wie alle Apache-Produkte ist Tomcat absolut kostenlos und stellt den vollständigen Quellcode zur Verfügung. Von allen Servern scheint er auch derjenige zu sein, der am stärksten mit den neuesten Servlet- und JSP-Spezifikationen kompatibel ist. Die kommerziellen Server sind aber meist besser dokumentiert, leichter zu konfigurieren und ein wenig schneller. Unter *http://jakarta.apache.org/tomcat/* können Sie Tomcat herunterladen.

- **Macromedia JRun.** JRun ist eine Servlet- und JSP-Engine, die für die Entwicklung im eigenständigen Modus verwendet werden kann oder für die Bereitstellung an die gängigsten kommerziellen Webserver angeschlossen werden kann. Für Entwicklungszwecke ist er kostenlos, aber Sie müssen eine Lizenz erwerben, ehe Sie mit ihm bereitstellen. Unter Entwicklern, die nach einer einfacheren Verwaltung als Tomcat suchen, ist er sehr beliebt. Einzelheiten finden Sie unter *http://www.allaire.com/products/JRun/*.
- **New Atlanta ServletExec.** ServletExec ist eine weitere beliebte Servlet- und JSP-Engine, die im eigenständigen Modus oder, für die Bereitstellung, angeschlossen an die Microsoft IIS-, Apache- oder iPlanet/Netsacpe-Webserver verwendet werden kann. Die Version 4.0 unterstützt Servlets 2.3 und JSP 1.2. Sie können ihn kostenlos herunterladen und verwenden, aber einige der High-Performance-Fähigkeiten und Administrations-Utilities sind bis zum Erwerb einer Lizenz deaktiviert. Der ServletExec-Debugger ist die Konfiguration, die Sie bei der Verwendung als eigenständigen Desktop-Entwicklungsserver benutzen würden. Einzelheiten finden Sie unter *http://www.servletexec.com/*.
- **Caucho Resin.** Resin ist eine schnelle Servlet- und JSP-Engine mit ausgiebiger XML-Unterstützung. Sie ist für die Entwicklung und für nicht-kommerzielle Bereitstellungszwecke kostenlos. Einzelheiten finden Sie unter *http://www.caucho.com/*.
- **LiteWebServer von Gefion Software.** LWS ist ein kleiner, eigenständiger Webserver, der Servlets und JSP unterstützt. Er ist sowohl für Entwicklungs- als auch für Bereitstellungszwecke kostenlos. Aber eine Lizenz berechtigt Sie zu verstärktem Support und zum vollständigen Serverquellcode. Unter *http://www.gefionsoftware.com/LiteWebServer/* finden Sie Einzelheiten.

1.3 Ändern des Ports und Konfigurieren von anderen Servereinstellungen

Um Konflikte mit anderen Webservern zu vermeiden, die eventuell den Standard-Port (80) verwenden, benutzen die meisten in Abschnitt 1.2 aufgelisteten Server einen anderen Standard-Port. Wenn Sie aber die Server im eigenständigen Modus benutzen und keinen anderen Server permanent auf Port 80 laufen lassen, werden Sie es bequemer finden, Port 80 zu verwenden. Auf diese Art müssen Sie nicht in jedem URL, den Sie in Ihren Browser eingeben, die Port-Nummer angeben. Es gibt noch ein oder zwei andere Einstellungen, die Sie ebenfalls verändern sollten.

Die Änderung des Ports oder anderer Konfigurationseinstellungen ist ein serverspezifischer Prozess. Daher müssen Sie die Dokumentation Ihres Servers auf maßgebliche Anleitungen hin durchsehen. Ich werde aber für drei der beliebtesten kostenlosen Server hier eine kurze Zusammenfassung geben: Tomcat, JRun und ServletExec.

1.3.1 Apache Tomcat

Port-Nummer

Bei Tomcat 4 müssen Sie zum Ändern der Port-Nummer *install_dir/conf/server.xml* bearbeiten, das Attribut `port` des Elements `Connector` von 8080 in 80 ändern und den Server neu starten. Denken Sie daran, dass sich dieser Abschnitt auf die Verwendung von Tomcat auf Ihrem Desktopsystem im eigenständigen Modus bezieht, auf dem noch kein anderer Server permanent auf Port 80 läuft. Auf Unix/Linux müssen Sie über Systemadministratorrechte verfügen, um Dienste auf Port 80 oder anderen Port-Nummern unter 1024 zu starten. Auf Ihrem Desktop-Rechner haben Sie möglicherweise solche Benutzerrechte; auf Bereitstellungsservern verfügen Sie darüber jedoch nicht unbedingt.

Das ursprüngliche Element sieht ungefähr so aus:

```
<Connector
  className="org.apache.catalina.connector.http.HttpConnector"
  port="8080" ...
  ... />
```

Nach der Änderung sollte es ungefähr so aussehen:

```
<Connector
  className="org.apache.catalina.connector.http.HttpConnector"
  port="80" ...
  ... />
```

Die einfachste Methode, um den richtigen Eintrag zu finden, besteht darin, in *server.xml* nach 8080 zu suchen; es sollte nur einmal außerhalb eines Kommentars vorkommen. Stellen Sie sicher, dass Sie eine Sicherungskopie von *server.xml* erstellen, ehe Sie die Datei bearbeiten, falls Sie einen Fehler machen, durch den der Server nicht läuft. Denken Sie ebenfalls daran, dass XML die Groß-/Kleinschreibung beachtet. Sie können nicht `port` durch `Port` ersetzen oder `Connector` durch `connector`.

Bei Tomcat 3 verändern Sie die gleiche Datei (*install_dir/conf/server.xml*), aber Sie müssen etwas andere `Connector`-Elemente für verschiedene Unterveröffentlichungen von Tomcat verwenden. Bei Version 3.2 ersetzen Sie in dem folgenden Parameter-Element 8080 durch 80.

```
<Connector ...>
  <Parameter name="port" value="8080"/>
</Connector>
```

Starten Sie auch hier den Server neu, nachdem Sie die Veränderungen vorgenommen haben.

Andere Tomcat-Einstellungen

Neben dem Port sind noch drei zusätzliche Tomcat-Einstellungen wichtig: die Variable JAVA_HOME, die DOS-Speichereinstellungen und die Variable CATALINA_HOME oder TOMCAT_HOME.

Die wichtigste Tomcat-Einstellung ist die Umgebungsvariable JAVA_HOME – wird sie nicht gesetzt, wird Tomcat davon abgehalten, JSP-Seiten zu behandeln. Diese Variable sollte das Basis-JDK-Installationsverzeichnis auflisten, nicht das *bin*-Unterverzeichnis. Wenn Sie z.B. mit Windows 98/Me arbeiten und das JDK in *C:/JDK1.3* installiert haben, sollten Sie die folgende Zeile in Ihre *autoexec.bat*-Datei setzen:

```
set JAVA_HOME=C:\JDK1.3
```

Bei Windows NT/2000 sollten Sie in das Start-Menü gehen und EINSTELLUNGEN auswählen, dann SYSTEMSTEUERUNG, SYSTEM, ERWEITERT und UMGEBUNGSVARIABLEN. Dann sollten Sie den JAVA_HOME-Wert eingeben.

Bei Unix/Linux sollten Sie die folgende Codezeile in Ihre *.cshrc*-Datei setzen, wenn das JDK unter */usr/j2sdkl_3_1* installiert ist und Sie die C-Shell benutzen:

```
setenv JAVA_HOME /usr/j2sdk1_3_1
```

Statt die Umgebungsvariable JAVA_HOME global im Betriebssystem zu setzen, ziehen es einige Entwickler vor, das Startskript zu bearbeiten, um sie dort zu setzen. Wenn auch Sie diese Strategie bevorzugen, bearbeiten Sie *install_dir/bin/catalina.bat* (Tomcat 4; Windows) oder *install_dir/bin/tomcat.bat* (Tomcat 3; Windows) und ändern Sie dies:

```
if not "%JAVA_HOME%" == "" goto gotJavaHome
echo You must set JAVA_HOME to point at ...
goto cleanup
:gotJavaHome
```

in dies:

```
if not "%JAVA_HOME%" == "" goto gotJavaHome
set JAVA_HOME=C:\JDK1.3
:gotJavaHome
```

Stellen Sie sicher, dass Sie eine Sicherungskopie von *catalina.bat* oder *tomcat.bat* gemacht haben, ehe Sie Veränderungen vornehmen. Unix/Linux-Anwender nehmen ähnliche Veränderungen in *catalina.sh* oder *tomcat.sh* vor.

Wenn Sie mit Windows arbeiten, müssen Sie eventuell auch die DOS-Speichereinstellungen für die Skripte zum Starten und Herunterfahren ändern. Wenn Sie eine »Out of Environment Space«-Fehlermeldung erhalten, wenn Sie den Server starten, müssen Sie *install_dir/bin/startup.bat* mit rechts anklicken, EIGENSCHAFTEN auswählen, dann SPEICHER und den Eintrag INITIAL ENVIRONMENT von Auto in 2816 ändern. Wiederholen Sie den Prozess für *install_dir/bin/shutdown.bat*.

In einigen Fällen ist es auch hilfreich, die Umgebungsvariablen `CATALINA_HOME` (Tomcat 4) oder `TOMCAT_HOME` (Tomcat 3) zu setzen. Diese Variable identifiziert das Tomcat-Installationsverzeichnis gegenüber dem Server. Wenn Sie aber vorsichtig sind und vermeiden, die Server-Startskripte zu kopieren, und stattdessen nur Shortcuts verwenden (auf Unix/Linux »symbolische Links« genannt), müssen Sie diese Variable nicht setzen. Mehr Informationen zur Verwendung von Shortcuts finden Sie in Abschnitt 1.6.

Beachten Sie bitte, dass dieser Abschnitt die Verwendung von Tomcat als eigenständigen Server für die Servlet- und JSP-*Entwicklung* beschreibt. Um Tomcat als Servlet- und JSP-Container bereitzustellen, der in einem regulären Webserver integriert ist, ist eine vollkommen andere Konfiguration nötig. Informationen zur Verwendung von Tomcat für die Bereitstellung finden Sie unter *http://jakarta.apache.org/tomcat/tomcat-4.0-doc/*.

1.3.2 Macromedia JRun

Wenn Sie JRun im eigenständigen Modus benutzen (im Gegensatz zur Integration mit einem Standard-Webserver), gibt es verschiedene Optionen, deren Standardwerte Sie eventuell ändern sollten. Alle können von der grafischen JRun Management Console aus gesetzt werden und/oder durch den JRun-Installationsassistenten.

Port-Nummer

Um den JRun-Port zu ändern, starten Sie den JRun Admin Server, indem Sie das entsprechende Symbol anklicken (bei Windows gehen Sie ins START-MENÜ, dann in PROGRAMME und dann zu JRUN 3.X). Klicken Sie dann die Schaltfläche JRUN MANAGEMENT CONSOLE (JMC) an oder geben Sie in einem Browser den URL *http://localhost:8000/* ein. Loggen Sie sich mit dem Benutzernamen `admin` und dem Passwort ein, das Sie angegeben haben, als Sie JRun installiert haben. Wählen Sie JRUN DEFAULT SERVER aus und dann JRUN WEB SERVER. Abbildung 1.1 zeigt das Ergebnis. Wählen Sie als Nächstes WEB SERVER PORT aus, geben Sie 80 ein und klicken Sie auf UPDATE. Das Ergebnis sehen Sie in Abbildung 1.2. Wählen Sie schließlich den JRUN DEFAULT SERVER erneut aus und klicken Sie auf die Schaltfläche RESTART SERVER.

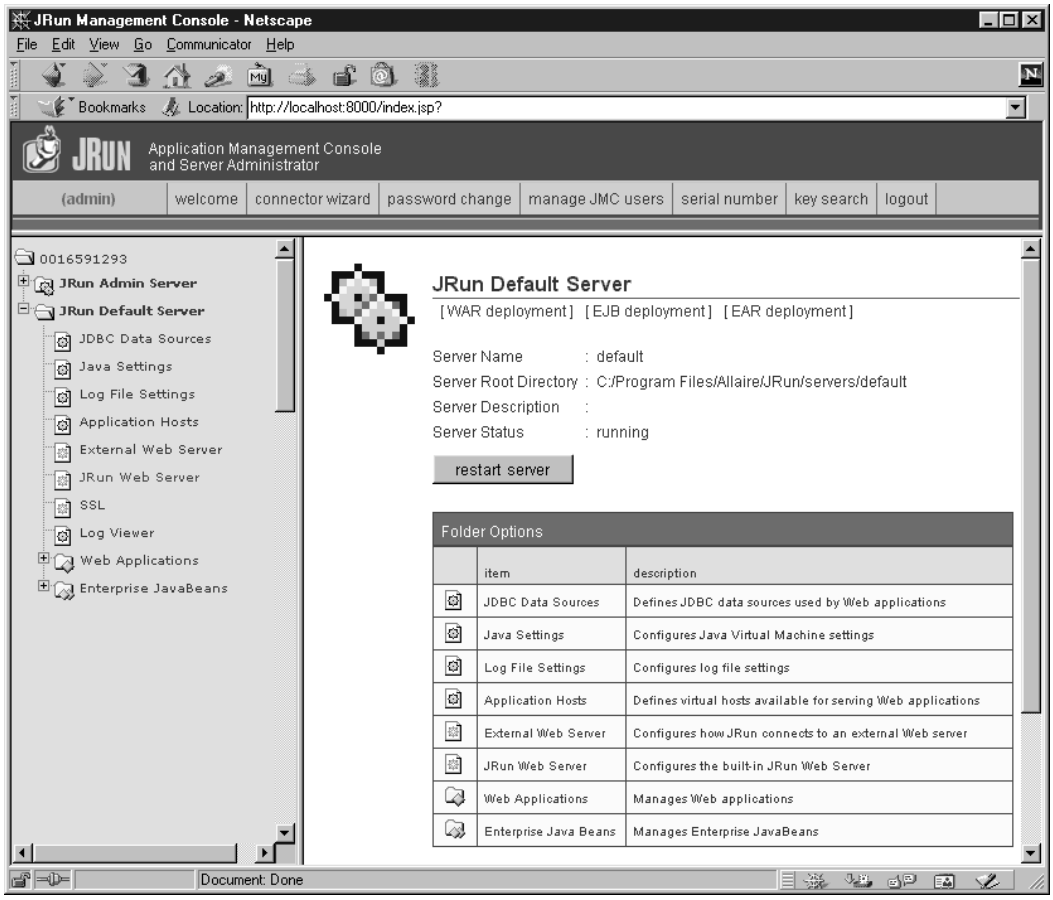

Abbildung 1.1: JMC-Konfigurationsbildschirm für den JRun Default Server

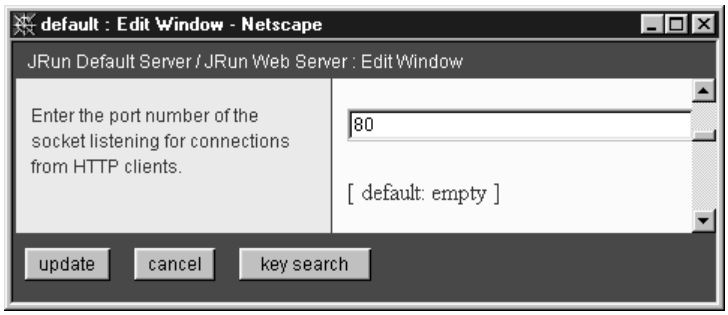

Abbildung 1.2: JRun Default Server-Pro-Konfigurationsfenster

Andere JRun-Einstellungen

Wenn Sie JRun installieren, wird der Installationsassistent Ihnen drei Fragen stellen, die besonders relevant für die Verwendung von JRun im eigenständigen Modus für Entwicklungszwecke sind. Als Erstes wird er Sie nach einer Seriennummer fragen. Sie können sie auslassen; sie ist nur für Bereitstellungsserver erforderlich. Zweitens wird er Sie fragen, ob Sie JRun als Dienst starten möchten. Sie sollten die Auswahl dieser Option aufheben; das automatische Starten von JRun ist zwar für die Bereitstellung nützlich, aber unbequem für die Entwicklung, denn das Serversymbol erscheint nicht in der Taskleiste und dadurch wird es schwieriger, den Server neu zu starten. Der Assistent sagt deutlich, dass die Verwendung von JRun als Dienst für die Bereitstellung reserviert sein sollte, aber da die Dienst-Option standardmäßig ausgewählt ist, kann sie Ihnen leicht entgehen. Schließlich werden Sie gefragt, ob Sie einen externen Webserver konfigurieren möchten. Lehnen Sie diese Option ab; Sie benötigen keinen separaten Webserver, wenn Sie JRun im eigenständigen Modus verwenden.

1.3.3 New Atlanta ServletExec

Die folgenden Einstellungen treffen auf die Verwendung des ServletExec Debugger 4.0 zu, die Version von ServletExec, die Sie für eigenständige Desktop-Entwicklung benutzen sollten (im Gegensatz zu einem regulären Webserver für die Bereitstellung).

Port-Nummer

Um die Port-Nummer von 8080 in 80 zu ändern, bearbeiten Sie *install_dir/StartSED40.bat* und fügen Sie »-port 80« am Ende der Zeile ein, die den Server startet:

```
%JAVA_HOME%\bin\java ... ServletExecDebuggerMain -port 80
```

Denken Sie daran, dass sich dieser Abschnitt auf die Verwendung von ServletExec im eigenständigen Modus auf Ihrem Desktop-System bezieht, in dem noch kein anderer Server permanent auf Port 80 läuft. Unter Unix/Linux müssen Sie über Systemadministratorrechte verfügen, um Dienste auf Port 80 oder einer anderen Port-Nummer unter 1024 zu starten. Auf Ihrem Desktoprechner verfügen Sie wahrscheinlich über solche Rechte; Sie haben sie aber nicht notwendigerweise auf Bereitstellungsservern.

Andere ServletExec-Einstellungen

ServletExec hat mit Tomcat zwei Einstellungen gemeinsam. Die eine erforderliche Einstellung ist die Umgebungsvariable `JAVA_HOME`. Wie bei Tomcat bezieht sich diese Variable auf das Basisinstallationsverzeichnis des JDK (nicht das *bin*-Unterverzeichnis). Wenn das JDK z.B. in *C:\JDK1.3* installiert ist, sollten Sie den Eintrag `JAVA_HOME` in *install_dir/StartSED40.bat* so ändern, dass er wie folgt aussieht:

```
set JAVA_HOME=C:\JDK1.3
```

Wenn Sie mit Windows arbeiten, müssen Sie, ebenso wie bei Tomcat, die DOS-Speichereinstellungen für das Startskript ändern. Wenn Sie beim Starten des Servers eine »Out of Environment Space«-Fehlernachricht erhalten, müssen Sie *install_dir/bin/StartSED40.bat* mit rechts anklicken, EIGENSCHAFTEN und SPEICHER auswählen und den Eintrag INITIAL ENVIRONMENT von Auto in 2816 ändern.

1.4 Den Server testen

Ehe Sie Ihre eigenen Servlets oder JSP-Seiten ausprobieren, sollten Sie sicherstellen, dass der Server richtig installiert und konfiguriert ist. Klicken Sie bei Tomcat auf *install_dir/bin/startup.bat* (Windows) oder führen Sie *install_dir/bin/startup.sh* (Unix/Linux) aus. Gehen Sie bei JRun in das START-MENÜ und wählen Sie PROGRAMME, JRUN 3.1 aus und dann JRUN DEFAULT SERVER. Bei ServletExec klicken Sie *install_dir/bin/StartSED40.bat* an. In allen drei Fällen geben Sie den URL *http://localhost/* in Ihrem Browser ein und stellen sicher, dass Sie eine richtige Webseite erhalten, keine Fehlermeldung, die besagt, dass die Seite nicht angezeigt werden kann oder dass der Server sie nicht findet. Abbildung 1.3 bis Abbildung 1.5 zeigen typische Ergebnisse. Wenn Sie sich dazu entscheiden, die Port-Nummer nicht in 80 zu ändern (siehe Abschnitt 1.3), müssen Sie einen URL wie *http://localhost:8080/* verwenden, der die Port-Nummer enthält.

Abbildung 1.3: Anfangsseite für Tomcat 4.0

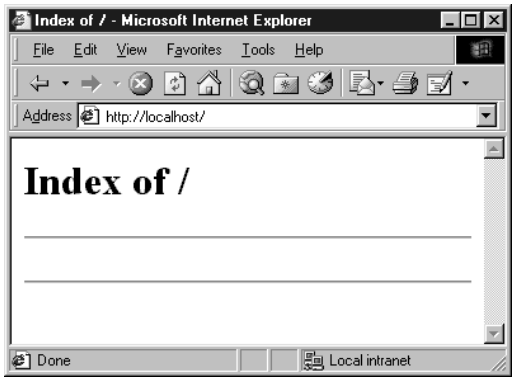

Abbildung 1.4: Anfangsseite für JRun 3.1

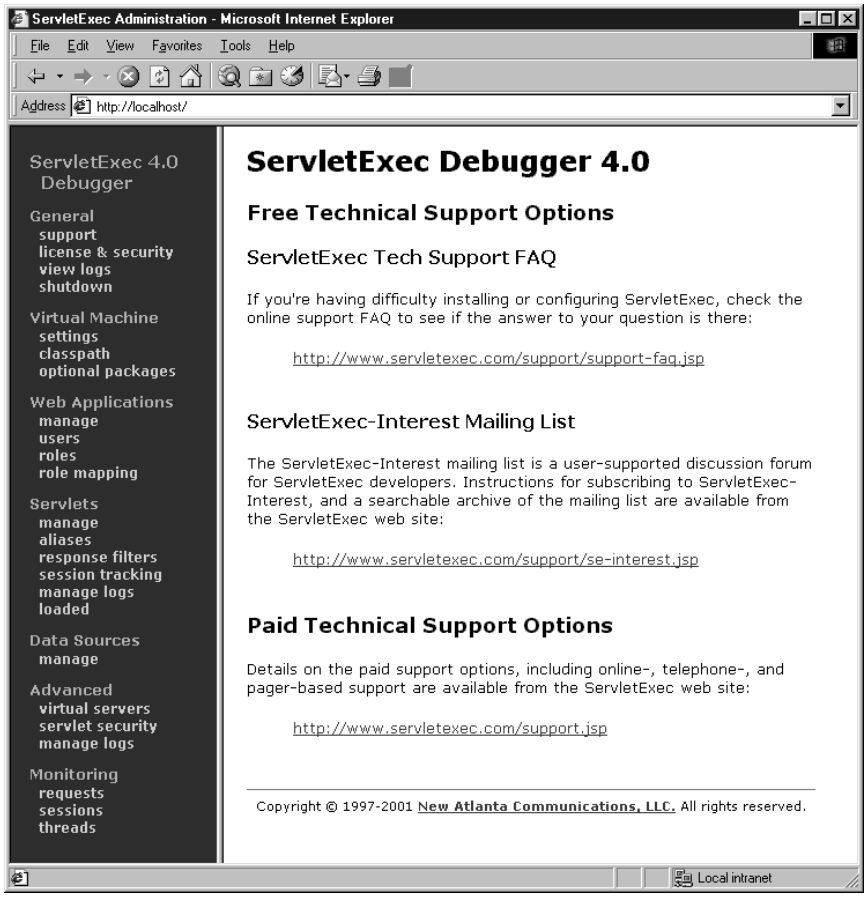

Abbildung 1.5: Anfangsseite für ServletExec 4.0

1.5 Testen Sie einige einfache HTML- und JSP-Seiten

Nachdem Sie verifiziert haben, dass der Server läuft, sollten Sie sicherstellen, dass Sie einfache HTML- und JSP-Seiten installieren und auf sie zugreifen können. Dieser Test zeigt Ihnen, wenn er erfolgreich ist, zwei wichtige Dinge. Wenn Sie erfolgreich auf eine HTML-Seite zugreifen können, zeigt das erstens, dass Sie verstehen, in welchen Verzeichnissen HTML- und JSP-Seiten gespeichert werden sollten. Wenn Sie erfolgreich auf eine neue JSP-Seite zugreifen können, zeigt das zweitens, dass der Java-Compiler (nicht nur die Java Virtual Machine) richtig konfiguriert ist. Irgendwann werden Sie sicherlich Ihre eigenen Webanwendungen erzeugen und benutzen (siehe Kapitel 4), aber zum anfänglichen Testen empfehle ich Ihnen, die Standardwebanwendung zu benutzen. Webanwendungen folgen zwar einer allgemeinen Verzeichnisstruktur, der genaue Speicherort der Standardwebanwendung ist aber serverspezifisch. Genaue Anleitungen finden Sie in der Dokumentation Ihres Servers, aber ich habe in der folgenden Liste die Speicherorte für Tomcat, JRun und ServletExec zusammengefasst. Dort, wo ich *SomeDirectory* angegeben habe, können Sie jeden beliebigen Verzeichnisnamen verwenden. (Sie dürfen aber niemals *WEB-INF* oder *META-INF* als Verzeichnisnamen benutzen. Bei der Standardwebanwendung müssen Sie außerdem Verzeichnisnamen vermeiden, die dem URL-Präfix einer anderen Webanwendung entsprechen.)

- **Tomcat-Verzeichnis:**
  install_dir/webapps/ROOT (oder *install_dir/webapps/ROOT/SomeDirectory*)
- **JRun-Verzeichnis:**
  install_dir/servers/default/default-app
  (oder *install_dir/servers/default/default-app/SomeDirectory*)
- **ServletExec-Verzeichnis:**
  install_dir/public_html[1] (oder *install_dir/public_html/SomeDirectory*)
- **Entsprechende URLs:**
  http://host/Hello.html (oder *http://host/SomeDirectory/Hello.html*) *http://host/Hello.jsp* (oder *http://host/SomeDirectory/Hello.jsp*)

Ich schlage vor, dass Sie für Ihre ersten Tests einfach *Hello.html* (Beispiel 1.1, Abbildung 1.6) und *Hello.jsp* (Beispiel 1.2, Abbildung 1.7) verwenden und es an den entsprechenden Orten speichern. Der Code dieser Dateien sowie der Code dieses Buches steht unter *http://www.moreservlets.com* und auf der Buch-CD zur Verfügung. Diese Website enthält außerdem Aktualisierungen, Ergänzungen, Informationen zu Kursen und den vollständigen Text von »Core Servlets and JavaServer Pages« (im PDF-Format). Wenn weder die HTML-Datei noch die JSP-Datei funktioniert (wenn Sie z.B. den Fehler 404 – File Not Found erhalten), verwenden Sie wahrscheinlich das falsche Verzeichnis für die Dateien. Wenn die HTML-Datei funktioniert, aber die JSP-Datei nicht, haben Sie das Basis-JDK-Verzeichnis möglicherweise falsch angegeben (z.B. mit der Variable `JAVA_HOME`).

1 Beachten Sie, dass das Verzeichnis *public_html* von ServletExec automatisch erzeugt wird, wenn Sie den Server zum ersten Mal ausführen. Sie können public_html also nicht finden, wenn Sie den Server noch nicht, wie in Abschnitt 1.4 beschrieben, getestet haben.

Beispiel 1.1: Hello.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>HTML Test</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<H1>HTML Test</H1>
Hello.
</BODY>
</HTML>
```

Abbildung 1.6: Das Ergebnis von Hello.html

Beispiel 1.2: Hello.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>JSP Test</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<H1>JSP Test</H1>
Time: <%= new java.util.Date() %>
</BODY>
</HTML>
```

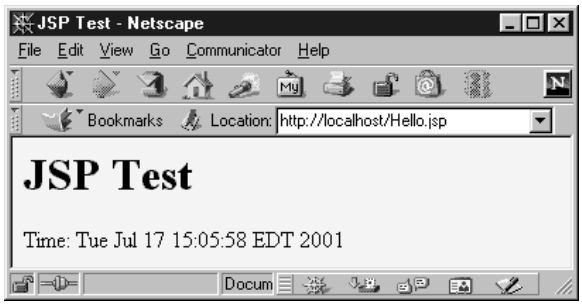

Abbildung 1.7: Das Ergebnis von Hello.jsp

1.6 Richten Sie Ihre Entwicklungsumgebung ein

Das Server-Startupskript setzt automatisch den CLASSPATH des Servers so, dass er die Standard-Servlet- und JSP-Klassen und das *WEB-INF/classes*-Verzeichnis (das kompilierte Servlets enthält) jeder Webanwendung umfasst. Aber *Sie* benötigen ähnliche Einstellungen oder Sie sind nicht in der Lage, Servlets überhaupt zu kompilieren. Dieser Abschnitt fasst die Konfiguration zusammen, die für die Servlet-Entwicklung benötigt wird.

1.6.1 Erstellen Sie ein Entwicklungsverzeichnis

Als Erstes sollten Sie ein Verzeichnis erstellen, in dem Sie die Servlets und JSP-Seiten speichern, die Sie entwickeln werden. Dieses Verzeichnis kann sich in Ihrem Heimverzeichnis befinden (z.B. *~/ServletDevel* bei Unix) oder in einem bequemen allgemeinen Speicherort (z.B. *C:\ServletDevel* unter Windows). Es sollte sich aber *nicht* im Installationsverzeichnis des Servers befinden.

Irgendwann werden Sie das Entwicklungsverzeichnis in unterschiedliche Webanwendungen aufteilen (jede mit einer allgemeinen Struktur – siehe Kapitel 4). Für das anfängliche Testen Ihrer Umgebung können Sie aber einfach Servlets entweder direkt in das Entwicklungsverzeichnis setzen (für paketlose Servlets) oder in ein Unterverzeichnis, das dem Servlet-Paketnamen entspricht. Viele Entwickler setzen einfach all ihren Code in das Bereitstellungsverzeichnis des Servers (siehe Abschnitt 1.9). Ich rate Ihnen dringend von dieser Praxis ab und empfehle Ihnen stattdessen eines der Verfahren, die in Abschnitt 1.8 beschrieben werden. Das Entwickeln im Bereitstellungsverzeichnis des Servers erscheint zwar anfangs einfacher, da kein Kopieren von Dateien erforderlich ist, auf lange Sicht verkompliziert es die Dinge aber deutlich. Das Vermischen von Speicherorten macht es schwierig, eine betriebsfähige Version von einer Version zu trennen, die Sie testen, macht es umständlich, auf mehreren Servern zu testen, und macht die Organisation viel komplizierter. Außerdem ist Ihr Desktop mit fast 100-prozentiger Sicherheit nicht der endgültige Bereitstellungsserver, sodass Sie ohnehin im Endeffekt ein gutes System für die Bereitstellung entwickeln müssen.

Hinweis

Benutzen Sie nicht das Bereitstellungsverzeichnis des Servers als Speicherort für die Entwicklung. Verwenden Sie stattdessen ein separates Entwicklungsverzeichnis.

1.6.2 Verknüpfungen zum Starten und Anhalten des Servers erstellen

Da ich den Server häufig neu starte, halte ich es für bequem, die Symbole zum Starten und Herunterfahren des Servers in meinem Hauptentwicklungsverzeichnis mit Verknüpfungen zu versehen. Sie werden es wahrscheinlich angenehm finden, es ebenso zu machen.

Gehen Sie z.B. bei Tomcat auf Windows zu *install_dir/bin*, klicken Sie *startup.bat* mit rechts an und wählen Sie KOPIEREN aus. Gehen Sie dann in Ihr Entwicklungsverzeichnis, klicken Sie das Fenster mit rechts an und wählen Sie VERKNÜPFUNG EINFÜGEN (nicht EINFÜGEN) aus. Wiederholen

Sie den Prozess für *install_dir/bin/shutdown.bat*. Auf Unix verwenden Sie `ln -s`, um eine symbolische Verknüpfung zu *startup.sh*, *tomcat.sh* (wird benötigt, auch wenn Sie diese Datei nicht direkt aufrufen) und *shutdown.sh* herzustellen.

Bei JRun unter Windows rufen Sie im START-MENÜ PROGRAMME auf, wählen dann JRUN 3.X aus, klicken das JRUN DEFAULT SERVER-Symbol mit rechts an und wählen KOPIEREN aus. Gehen Sie dann in Ihr Entwicklungsverzeichnis, klicken Sie das Fenster mit rechts an und wählen Sie VERKNÜPFUNG EINFÜGEN aus. Wiederholen Sie den Prozess für den JRun Admin Server und die JRun Management Console.

Bei ServletExec Debugger (d.h. bei einem eigenständigen Entwicklungsserver) gehen Sie zu *install_dir*, klicken *StartSED40.bat* mit rechts an und wählen KOPIEREN aus. Gehen Sie dann in Ihr Entwicklungsverzeichnis, klicken Sie das Fenster mit rechts an und wählen Sie VERKNÜPFUNG EINFÜGEN (nicht einfach EINFÜGEN) aus. Es gibt keine separate Datei zum Herunterfahren; um ServletExec anzuhalten, gehen Sie einfach zu *http://localhost/* (siehe Abbildung 1.5) und klicken Sie die Shutdown-Verknüpfung in der Kategorie GENERAL auf der linken Seite an.

1.6.3 Setzen Sie Ihren CLASSPATH

Da Servlets und JSP nicht Teil der Java 2 Plattform Standard Edition sind, müssen Sie die Servlet-Klassen für den Compiler identifizieren. Der *Server* weiß über die Servlet-Klassen bereits Bescheid, aber der *Compiler* (z.B. `javac`), den Sie für die Entwicklung benutzen, wahrscheinlich nicht. Wenn Sie also Ihren `CLASSPATH` nicht setzen, werden Sie Fehlermeldungen über unbekannte Klassen erhalten, wenn Sie versuchen, Servlets, Tag-Bibliotheken oder andere Klassen zu kompilieren, die das Servlet-API benutzen. Der genaue Speicherort der Servlet JAR-Datei variiert von Server zu Server. In den meisten Fällen können Sie nach einer Datei mit dem Namen *servlet.jar* suchen. Oder Sie lesen in der Dokumentation des Servers nach, um den Speicherort zu finden. Wenn Sie die JAR-Datei gefunden haben, fügen Sie den Speicherort zu Ihrem Entwicklungs-`CLASSPATH` hinzu. Hier sehen Sie die Speicherorte für einige gängige Entwicklungsserver:

- **Tomcat 4-Speicherort:** *install_dir/common/lib/servlet.jar*
- **Tomcat 3-Speicherort:** *install_dir/lib/servlet.jar*
- **JRun-Speicherort:** *install_dir/lib/ext/servlet.jar*
- **ServletExec-Speicherort:** *install_dir/ServletExecDebugger.jar*

Neben der Servlet-JAR-Datei müssen Sie auch Ihr Entwicklungsverzeichnis in den `CLASSPATH` setzen. Dies ist zwar für einfache paketlose Server nicht notwendig, wenn Sie aber erst einmal ein wenig Erfahrung gesammelt haben, werden Sie sicherlich Pakete verwenden. Das Kompilieren einer Datei, die sich in einem Paket befindet und die eine andere Klasse in dem gleichen Paket verwendet, macht es erforderlich, dass der `CLASSPATH` das Verzeichnis enthält, das sich ganz oben in der Pakethierarchie befindet. In diesem Fall ist es das Entwicklungsverzeichnis, das ich gerade im ersten Unterabschnitt erläutert habe. Das Vergessen dieser Einstellung ist möglicherweise *der* gängigste Fehler, der von Servlet-Programmieranfängern gemacht wird.

Hinweis

Denken Sie daran, Ihr Entwicklungsverzeichnis zu Ihrem CLASSPATH hinzuzufügen. Andernfalls werden Sie »Unresolved symbol«-Fehlermeldungen erhalten, wenn Sie versuchen, Servlets zu kompilieren, die sich in Paketen befinden und die andere Klassen im gleichen Paket nutzen.

Schließlich sollten Sie noch ».« (das aktuelle Verzeichnis) in Ihren CLASSPATH einfügen. Andernfalls können Sie nur paketlose Klassen kompilieren, die sich in der obersten Ebene des Entwicklungsverzeichnisses befinden.

Hier sehen Sie einige repräsentative Methoden für das Setzen des CLASSPATH. Sie gehen davon aus, dass Ihr Entwicklungsverzeichnis *C:\devel* (Windows) oder */usr/devel* (Unix/Linux) ist und dass Sie Tomcat 4 verwenden. Ersetzen Sie *install_dir* durch den tatsächlichen Basis-Installationsort des Servers. Stellen Sie sicher, dass Sie für die Dateinamen die richtigen Groß- oder Kleinbuchstaben verwenden. Beachten Sie, dass diese Beispiele nur ein Verfahren für das Setzen des CLASSPATH darstellen. Viele Java-integrierte Entwicklungsumgebungen haben eine globale oder projektspezifische Einstellung, die das gleiche Ergebnis erzielt. Diese Einstellungen sind aber ganz IDE-spezifisch und werden hier nicht erläutert.

- **Windows 98/Me:** Setzen Sie Folgendes in Ihre *autoexec.bat*. (Beachten Sie, dass alles in eine Reihe geschrieben wird, ohne Leerstellen – hier wurde es für die Lesbarkeit umbrochen.)

```
set CLASSPATH=.;
              C:\devel;
              install_dir\common\lib\servlet.jar
```

- **Windows NT/2000:** Gehen Sie in das START-Menü und wählen Sie EINSTELLUNGEN aus, dann SYSTEMSTEUERUNG, SYSTEM und UMGEBUNG. Geben Sie dann den CLASSPATH-Wert des vorherigen Punkts ein.
- **Unix/Linux (C shell):** Setzen Sie Folgendes in Ihre *.cshrc*. (In der tatsächlichen Datei gehört der Code in eine einzige Zeile, ohne Leerstellen.)

```
setenv CLASSPATH .:
                 /usr/devel:
                 install_dir/common/lib/servlet.jar
```

1.6.4 Setzen Sie für die Servlet- und JSP API-Dokumentation ein Lesezeichen oder installieren Sie sie

So wie kein ernsthafter Programmierer allgemein dienliche Java-Anwendungen ohne Zugriff auf die JDK 1.3 oder 1.4 API-Dokumentation entwickeln sollte (im Javadoc-Format), so sollte auch kein ernsthafter Programmierer Servlets oder JSP-Seiten ohne Zugriff auf das API für Klassen in den `javax.servlet`-Paketen entwickeln. Hier sehen Sie eine Zusammenfassung, wo Sie das API finden:

- *http://java.sun.com/products/jsp/download.html*
  Auf dieser Site können Sie die Javadoc-Dateien für das Servlet 2.3- und JSP 1.2-API oder für das Servlet 2.2- und JSP 1.1-API herunterladen. Sie werden dieses API möglicherweise so nützlich finden, dass Sie eine lokale Kopie besitzen möchten, statt es online durchzublättern. Einige Server bündeln diese Dokumentation aber, was Sie vor dem Download überprüfen sollten.
- *http://java.sun.com/products/servlet/2.3/javadoc/*
  Auf dieser Site können Sie das Servlet 2.3-API online durchblättern.
- *http://java.sun.com/products/servlet/2.2/javadoc/*
  Auf dieser Site können Sie das Servlet 2.2- und JSP 1.1-API online durchblättern.
- *http://java.sun.com/j2ee/j2sdkee/techdocs/api/*
  Unter dieser Adresse können Sie das vollständige API für die Java 2 Plattform, Enterprise Edition (J2EE) durchblättern, zu der auch die Servlet 2.2- und JSP 1.1-Pakete gehören.

1.7 Kompilieren und testen Sie einige einfache Servlets

OK, Ihre Umgebung ist eingerichtet. Zumindest *denken* Sie, dass es so ist. Es wäre schön, wenn Sie diese Hypothese bestätigen könnten. Im Folgenden finden Sie drei Tests, die Ihnen dabei helfen, das zu verifizieren.

1.7.1 Test 1: Ein Servlet, das keine Pakete verwendet

Das erste Servlet, das Sie ausprobieren können, ist ganz elementar: keine Pakete, keine Zusatz(Hilfs)-Klassen, nur einfache HTML-Ausgabe. Statt Ihr eigenes Test-Servlet zu schreiben, können Sie einfach *HelloServlet.java* (Beispiel 1.3) aus dem Quellcodearchiv des Buches unter *http://www.moreservlets.com* oder der Buch-CD nehmen. Wenn Sie Kompilierungsfehler bekommen, gehen Sie zurück und überprüfen Sie die `CLASSPATH`-Einstellungen (Abschnitt 1.6) – wahrscheinlich haben Sie sich bei der Auflistung des Speicherorts der JAR-Datei geirrt, die die Servlet-Klassen (z.B. *servlet.jar*) enthält. Wenn Sie erst einmal *HelloServlet.java* kompilieren, setzen Sie *HelloServlet.class* in den entsprechenden Speicherort (normalerweise das Verzeichnis *WEB-INF/classes* der Standard-Webanwendung Ihres Servers). Überprüfen Sie diesen Speicherort in der Dokumentation Ihres Servers oder sehen Sie in der folgenden Liste nach, die eine Zusammenfassung der Speicherorte enthält, die von Tomcat, JRun und ServletExec verwendet werden. Greifen Sie dann auf das Servlet mit dem URL *http://localhost/servlet/HelloServlet* zu (oder auch *http://localhost:8080/servlet/HelloServlet*, wenn Sie sich dazu entschieden haben, die Port-Nummer nicht zu ändern, wie es in Abschnitt 1.3 beschrieben wurde). Sie sollten eine ähnliche Ausgabe wie in

Abbildung 1.8 erhalten. Wenn dieser URL fehlschlägt, aber der Test des Servers selbst (Abschnitt 1.4) erfolgreich ist, haben Sie möglicherweise die Klassendatei in das falsche Verzeichnis gesetzt.

- **Tomcat-Verzeichnis:** *install_dir/webapps/ROOT/WEB-INF/classes*
- **JRun-Verzeichnis:** *install_dir/servers/default/default-app/WEB-INF/classes*
- **ServletExec-Verzeichnis:** *install_dir/Servlets*
- **Entsprechender URL:** *http://host/servlet/HelloServlet*

Abbildung 1.8: Ergebnis von HelloServlet

Beispiel 1.3: HelloServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/** Einfaches Servlet, mit dem der Server getestet wird. */

public class HelloServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String docType =
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\">\n";
    out.println(docType +
                "<HTML>\n" +
                "<HEAD><TITLE>Hello</TITLE></HEAD>\n" +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1>Hello</H1>\n" +
                "</BODY></HTML>");
  }
}
```

1.7.2 Test 2: Ein Servlet, das Pakete verwendet

Das zweite Servlet, das Sie ausprobieren können, verwendet Pakete, aber keine Hilfsklassen. Sie können *HelloServlet2.java* (Beispiel 1.4) aus dem Quellcodearchiv dieses Buches unter *http://www.moreservlets.com* oder der Buch-CD verwenden. Da sich dieses Servlet in dem `moreservlets`-Paket befindet, sollte es in das *moreservlets*-Verzeichnis kommen, sowohl während der Entwicklung als auch, wenn es für den Server bereitgestellt wird. Wenn Sie Kompilierfehler bekommen, gehen Sie zurück und überprüfen Sie Ihre `CLASSPATH`-Einstellungen (Abschnitt 1.6) – wahrscheinlich haben Sie ».« (das aktuelle Verzeichnis) nicht angegeben. Wenn Sie *HelloServlet2.java* kompilieren, setzen Sie *HelloServlet2.class* in das *moreservlets*-Unterverzeichnis von dem Verzeichnis, das der Server für Servlets verwendet, die sich nicht in benutzerdefinierten Webanwendungen befinden (normalerweise das *WEB-INF/classes*-Verzeichnis der Standardwebanwendung). Sehen Sie wegen dieses Speicherorts in der Dokumentation Ihres Servers nach oder in der folgenden Liste, die eine Zusammenfassung der Speicherorte für Tomcat, JRun und ServletExec enthält. Vorerst können Sie einfach die Klassendatei vom Entwicklungsverzeichnis in das Bereitstellungsverzeichnis kopieren, aber Abschnitt 1.8 wird einige Optionen für die Vereinfachung des Prozesses bieten.

Wenn Sie erst einmal das Servlet in das richtige Verzeichnis gesetzt haben, greifen Sie mit dem URL *http://localhost/servlet/moreservlets.HelloServlet2* darauf zu. Sie sollten eine ähnliche Ausgabe wie in Abbildung 1.9 erhalten. Wenn der Test fehlschlägt, haben Sie wahrscheinlich entweder den URL falsch eingegeben (z.B. einen Schrägstrich statt einen Punkt nach dem Paketnamen gesetzt) oder *HelloServlet2.class* an den falschen Speicherort gesetzt (z.B. direkt in das *WEB-INF/classes*-Verzeichnis statt in das *moreservlets*-Unterverzeichnis).

- **Tomcat-Verzeichnis:** *install_dir/webapps/ROOT/WEB-INF/classes/moreservlets*
- **JRun-Verzeichnis:** *install_dir/servers/default/default-app/WEB-INF/classes/moreservlets*
- **ServletExec-Verzeichnis:** *install_dir/Servlets/moreservlets*
- **Entsprechender URL:** *http://host/servlet/moreservlets.HelloServlet2*

Beispiel 1.4: moreservlets/HelloServlet2.java

```
package moreservlets;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/** Einfaches Servlet, mit dem die Verwendung von Paketen getestet wird. */

public class HelloServlet2 extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
```

```
    PrintWriter out = response.getWriter();
    String docType =
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\">\n";
    out.println(docType +
                "<HTML>\n" +
                "<HEAD><TITLE>Hello (2)</TITLE></HEAD>\n" +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1>Hello (2)</H1>\n" +
                "</BODY></HTML>");
  }
}
```

Abbildung 1.9: Das Ergebnis von HelloServlet2

1.7.3 Test 3: Ein Servlet, das Pakete und Hilfsprogramme verwendet

Das letzte Servlet, das Sie testen sollten, um die Konfiguration Ihres Servers und der Entwicklungsumgebung zu überprüfen, verwendet sowohl Pakete als auch Hilfsklassen. Beispiel 1.5 enthält *HelloServlet3.java*, ein Servlet, das die Klasse `ServletUtilities` (Beispiel 1.6) verwendet, um die Generierung der `DOCTYPE`- (gibt die HTML-Version an – nützlich bei der Verwendung von HTML-Validatoren) und `HEAD`- (gibt den Titel an) Abschnitte der HTML-Seite zu vereinfachen. Diese beiden Teile der Seite sind nützlich (im Grunde sogar technisch erforderlich), die Generierung mit `println`-Servlet-Anweisungen ist aber mühselig. Sie finden auch hier den Quellcode unter *http://www.moreservlets.com* oder auf der Buch-CD.

Da sich sowohl das Servlet als auch die Hilfsklasse im `moreservlets`-Paket befindet, sollten sie in das *moreservlets*-Verzeichnis gesetzt werden. Wenn Sie Kompilierungsfehler bekommen, müssen Sie zurückgehen und Ihre `CLASSPATH`-Einstellungen (Abschnitt 1.6) überprüfen – wahrscheinlich haben Sie vergessen, die oberste Ebene des Entwicklungsverzeichnisses einzuschließen. Ich habe es bereits gesagt, aber ich sage es noch einmal: Ihr `CLASSPATH` muss das Verzeichnis der obersten Ebene Ihrer Pakethierarchie enthalten, ehe Sie eine gepackte Klasse kompilieren können, die eine weitere Klasse des gleichen Pakets benutzt. Diese Anforderung gilt nicht speziell für Servlets, so

arbeiten Pakete auf der Java-Plattform im Allgemeinen. Trotzdem sind sich viele Servlet-Entwickler dieser Tatsache nicht bewusst und es ist einer der (vielleicht sogar *der*) gängigsten Fehler, den Entwickleranfänger machen.

Hinweis

Ihr `CLASSPATH` muss das Top Level-Entwicklungsverzeichnis enthalten. Andernfalls können Sie keine Servlets kompilieren, die sich in Paketen befinden, und die außerdem Klassen aus dem gleichen Paket benutzen.

Wenn Sie *HelloServlet3.java* kompilieren (wodurch automatisch *ServletUtilities.java* kompiliert wird), setzen Sie *HelloServlet3.class* und *ServletUtilities.class* in das *moreservlets*-Verzeichnis des Verzeichnisses, das der Server für Servlets verwendet, die sich nicht in benutzerdefinierten Webanwendungen befinden (normalerweise das *WEB-INF/classes*-Verzeichnis der Standardwebanwendung). Sehen Sie wegen dieses Speicherorts in der Dokumentation Ihres Servers nach oder in der folgenden Liste, die eine Zusammenfassung der Speicherorte für Tomcat, JRun und ServletExec enthält. Greifen Sie dann auf das Servlet mit dem URL *http://localhost/servlet/moreservlets.HelloServlet3* zu. Sie sollten eine ähnliche Ausgabe wie in Abbildung 1.10 erhalten.

- **Tomcat-Verzeichnis:** *install_dir/webapps/ROOT/WEB-INF/classes/moreservlets*
- **JRun-Verzeichnis:** *install_dir/servers/default/default-app/WEB-INF/classes/moreservlets*
- **ServletExec-Verzeichnis:** *install_dir/Servlets/moreservlets*
- **Entsprechender URL:** *http://host/servlet/moreservlets.HelloServlet3*

Beispiel 1.5: moreservlets/HelloServlet3.java

```
package moreservlets;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/** Einfaches Servlet, mit dem die Verwendung von Paketen
 *  und Hilfsprogrammen aus dem gleichen Paket getestet wird */

public class HelloServlet3 extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
```

```
    PrintWriter out = response.getWriter();
    String title = "Hello (3)";
    out.println(ServletUtilities.headWithTitle(title) +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1>" + title + "</H1>\n" +
                "</BODY></HTML>");
  }
}
```

Beispiel 1.6: moreservlets/servletUtilities.java

```
package moreservlets;

import javax.servlet.*;
import javax.servlet.http.*;

/** Einige einfache Zeitsparer. Die meisten sind statische Methoden. */

public class ServletUtilities {
  public static final String DOCTYPE =
    "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
    "Transitional//EN\">";

  public static String headWithTitle(String title) {
    return(DOCTYPE + "\n" +
           "<HTML>\n" +
           "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n");
  }

  ...
}
```

Abbildung 1.10: Ergebnis von HelloServlet3

1.8 Eine vereinfachte Bereitstellungsmethode einrichten

Okay, Sie haben also ein Entwicklungsverzeichnis. Sie können Servlets mit oder ohne Pakete kompilieren. Sie wissen, in welches Verzeichnis die Servlet-Klassen gehören. Sie kennen den URL, mit dem auf sie zugegriffen werden sollte (zumindest den Standard-URL; in Abschnitt 5.3 werden Sie sehen, wie diese Adresse angepasst wird). Aber wie verschieben Sie die Klassendateien vom Entwicklungsverzeichnis in das Bereitstellungsverzeichnis? Wenn Sie jedes Mal jede einzelne per Hand kopieren müssen, ist das mühselig und fehleranfällig. Wenn Sie erst einmal Webanwendungen verwenden (siehe Kapitel 4), wird das Kopieren einzelner Dateien sogar noch umständlicher.

Es gibt verschiedene Optionen, um diesen Prozess zu vereinfachen. Hier sehen Sie einige der beliebtesten. Wenn Sie gerade erst mit Servlets und JSP beginnen, möchten Sie vielleicht mit der ersten Option beginnen und sie benutzen, bis Sie mit dem Entwicklungsprozess vertraut sind. Beachten Sie, dass ich *nicht* die Option aufliste, Ihren Code direkt in das Bereitstellungsverzeichnis des Servers zu setzen. Dies ist zwar eine der am häufigsten verwendeten Möglichkeiten unter Anfängern, es lässt sich aber so schlecht auf fortgeschrittenere Aufgaben skalieren, dass ich Ihnen empfehle, von Anfang an die Hände davon zu lassen.

1. Kopieren Sie sie in eine Verknüpfung oder einen symbolischen Link.
2. Verwenden Sie die Option `-d` von `javac`.
3. Lassen Sie Ihre IDE die Bereitstellung übernehmen.
4. Verwenden Sie *ant* oder ein ähnliches Tool.

Einzelheiten zu diesen vier Optionen finden Sie in den folgenden Unterabschnitten.

1.8.1 Kopie in eine Verknüpfung oder einen symbolischen Link einfügen

Gehen Sie unter Windows zur Standardwebanwendung des Servers, klicken Sie das *classes*-Verzeichnis mit rechts an und wählen Sie KOPIEREN aus. Gehen Sie dann in Ihr Entwicklungsverzeichnis, klicken Sie es mit rechts an und wählen Sie VERKNÜPFUNG EINFÜGEN aus (nicht nur EINFÜGEN). Immer wenn Sie nun ein paketloses Servlet kompilieren, ziehen Sie einfach die Klassendateien auf die Verknüpfung. Wenn Sie in Paketen entwickeln, verwenden Sie die rechte Maustaste, um das gesamte Verzeichnis (z. B. das *moreservlets*-Verzeichnis) auf die Verknüpfung zu ziehen, lassen Sie die Maustaste los und wählen Sie KOPIEREN aus. Bei Unix/Linux können Sie symbolische Links benutzen (die mit `ln -s` erzeugt werden), ähnlich wie für Windows-Verknüpfungen.

Ein Vorteil dieses Verfahrens besteht darin, dass es einfach ist. Es ist also gut für Anfänger, die sich auf das Erlernen von Servlets und JSP konzentrieren möchten, nicht auf Bereitstellungs-Tools. Ein weiterer Vorteil besteht darin, dass eine Variation angewendet wird, wenn Sie erst einmal Ihre eigenen Webanwendungen benutzen (siehe Kapitel 4). Erstellen Sie einfach eine Verknüpfung zum Webanwendungsverzeichnis (eine Ebene weiter oben als die Standardwebanwendung) und kopieren Sie jedes Mal die gesamte Webanwendung, indem Sie mit der rechten Maustaste das Verzeichnis, das Ihre Webanwendung enthält, auf diese Verknüpfung ziehen und KOPIEREN auswählen.

Ein Nachteil dieses Verfahrens besteht darin, dass es ein wiederholtes Kopieren erfordert, wenn Sie mehrere Server verwenden. Ich habe z.B. immer mindestens zwei unterschiedliche Server auf meinem Entwicklungssystem und teste meinen Code regelmäßig auf beiden Servern. Ein zweiter Nachteil besteht darin, dass bei diesem Verfahren sowohl die Java-Quellcodedateien als auch die Klassendateien auf den Server kopiert werden, obwohl nur die Klassendateien benötigt werden. Das mag auf Ihrem Desktop-Server keinen Unterschied machen, aber wenn es um einen »richtigen« Bereitstellungsserver geht, sollten Sie die Quellcodedateien nicht einschließen.

1.8.2 Verwenden Sie die Option –d von javac

Standardmäßig platziert der Java-Compiler (`javac`) Klassendateien im gleichen Verzeichnis wie die Quellcodedateien, von denen sie kommen. `javac` verfügt aber über eine Option (`-d`), mit der Sie einen anderen Speicherort für die Klassendateien angeben können. Sie müssen nur das Toplevel-Verzeichnis für Klassendateien angeben – `javac` wird gepackte Klassen automatisch in Unterverzeichnisse setzen, die den Paketnamen entsprechen. Bei Tomcat könnte ich z.B. das Servlet `HelloServlet2` (Beispiel 1.4, Abschnitt 1.7) wie folgt kompilieren (der Zeilenumbruch wurde nur der Klarheit wegen eingefügt, in Wirklichkeit wird er ausgelassen).

```
javac -d install_dir/webapps/ROOT/WEB-INF/classes
        HelloServlet2.java
```

Sie könnten sogar eine Windows-Batchdatei oder ein Unix-Skript oder -Alias erstellen, das einen Befehl wie `servletc` auf `javac -d install_dir/.../classes` erweitert. Unter *http://java.sun.com/j2se/1.3/docs/tooldocs/win32/javac.html* finden Sie weitere Einzelheiten zu `–d` und anderen `javac`-Optionen.

Ein Vorteil dieses Verfahrens besteht darin, dass ein manuelles Kopieren der Klassendateien nicht erforderlich ist. Außerdem kann genau der gleiche Befehl für Klassen in unterschiedlichen Paketen verwendet werden, da `javac` die Klassendateien automatisch in ein Unterverzeichnis setzt, das dem Paket entspricht.

1.8.3 Lassen Sie Ihre IDE die Bereitstellung übernehmen

Die meisten Entwicklungsumgebungen, die Servlets und JSP verstehen (z.B. IBM WebSphere Studio, Macromedia JRun Studio, Borland JBuilder), verfügen über Optionen, mit denen Sie der IDE mitteilen können, wo Klassendateien für Ihr Projekt bereitgestellt werden sollen. Wenn Sie die IDE dann anweisen, das Projekt zu erstellen, werden die Klassendateien automatisch am richtigen Speicherort bereitgestellt (paketspezifische Unterverzeichnisse und alles).

Ein Vorteil dieses Verfahrens, zumindest bei einigen IDEs, besteht darin, dass es HTML- und JSP-Seiten bereitstellen kann und sogar ganze Webanwendungen, nicht nur Java-Klassendateien. Ein Nachteil besteht darin, dass es sich um eine IDE-spezifische Technik handelt und dass sie daher nicht über alle Systeme portierbar ist.

1.8.4 Verwenden Sie ant oder ein ähnliches Tool

ant, das von Apache entwickelt wurde, ist ein Tool ähnlich wie das Hilfsprogramm *make* von Unix. *ant* ist aber in der Programmiersprache Java geschrieben (und ist daher portierbar) und wird als einfacher und mächtiger als *make* angepriesen. Viele Servlet- und JSP-Entwickler verwenden *ant* für die Kompilierung und Bereitstellung. Die Verwendung von *ant* ist besonders unter Tomcat-Anwendern beliebt und bei denjenigen, die Webanwendungen (siehe Kapitel 4) entwickeln.

Allgemeine Informationen zur Verwendung von *ant* finden Sie unter *http://jakarta.apache.org/ant/manual/*. Eine spezielle Anleitung für die Verwendung von *ant* mit Tomcat finden Sie unter *http://jakarta.apache.org/tomcat/tomcat-4.0-doc/appdev/processes.html*.

Der Hauptvorteil dieses Verfahrens liegt in der Flexibilität: *ant* ist mächtig genug, um alles zu behandeln, von der Kompilierung von Java-Quellcode bis hin zum Kopieren von Dateien, um WAR-Dateien zu erzeugen (Abschnitt 4.3). Der Nachteil von *ant* besteht darin, dass Sie erst einmal lernen müssen, wie es verwendet wird; bei *ant* müssen Sie mehr lernen als bei den anderen Techniken in diesem Abschnitt.

1.9 Bereitstellungsverzeichnisse für Standardwebanwendungen: Zusammenfassung

Die folgenden Unterabschnitte fassen zusammen, wie HTML-Dateien, JSP-Seiten, Servlets und Hilfsklassen in Tomcat, JRun und ServletExec bereitgestellt werden und wie auf sie zugegriffen wird. Die Zusammenfassung geht davon aus, dass Sie Dateien in der Standardwebanwendung bereitstellen, die Port-Nummer auf 80 geändert haben (siehe Abschnitt 1.3) und auf Servlets über den Standard-URL zugreifen (d.h. *http://host/servlet/ServletName*). In späteren Kapiteln wird erläutert, wie benutzerdefinierte Webanwendungen bereitgestellt werden und wie die URLs angepasst werden. Aber Sie werden wahrscheinlich mit den Standardeinstellungen anfangen wollen, um zu bestätigen, dass alles richtig funktioniert. Der Anhang enthält eine vereinheitlichte Zusammenfassung der Verzeichnisse, die von Tomcat, JRun und ServletExec benutzt werden, sowohl für die Standardwebanwendungen als auch für die benutzerdefinierten Webanwendungen.

Wenn Sie mit einem Server auf Ihrem Desktop arbeiten, können Sie *localhost* für den *host*-Abschnitt jedes URLs in diesem Abschnitt verwenden.

1.9.1 Tomcat

HTML und JSP-Seiten

- **Haupt-Speicherort:** *install_dir/webapps/ROOT*
- **Entsprechende URLs:** *http://host/SomeFile.html http://host/SomeFile.jsp*
- **Speziellerer Speicherort (beliebiges Unterverzeichnis):** *install_dir/webapps/ROOT/SomeDirectory*
- **Entsprechende URLs:** *http://host/SomeDirectory/SomeFile.html http://host/SomeDirectory/SomeFile.jsp*

Individuelle Servlet- und Hilfsklassendateien

- **Haupt-Speicherort (Klassen ohne Pakete):** *install_dir/webapps/ROOT/WEB-INF/classes*
- **Entsprechende URLs (Servlets):** *http://host/servlet/ServletName*
- **Speziellerer Speicherort (Klassen in Paketen):**
  install_dir/webapps/ROOT/WEB-INF/classes/packageName
- **Entsprechende URLs (Servlets in Paketen):** *http://host/servlet/packageName.ServletName*

In JAR-Dateien gebündelte Servlet- und Hilfsklassendateien

- Speicherort: *install_dir/webapps/ROOT/WEB-INF/lib*
- **Entsprechende URLs (Servlets):**
  http://host/servlet/ServletName
  http://host/servlet/packageName.ServletName

1.9.2 JRun

HTML und JSP-Seiten

- **Haupt-Speicherort:** *install_dir/servers/default/default-app*
- **Entsprechende URLs:**
  http://host/SomeFile.html
  http://host/SomeFile.jsp
- **Speziellerer Speicherort (beliebiges Unterverzeichnis):**
  install_dir/servers/default/default-app/SomeDirectory
- **Entsprechende URLs:**
  http://host/SomeDirectory/SomeFile.html
  http://host/SomeDirectory/SomeFile.jsp

Individuelle Servlet- und Hilfsklassendateien

- **Haupt-Speicherort (Klassen ohne Paket):**
  install_dir/servers/default/default-app/WEB-INF/classes
- **Entsprechende URL (Servlets):** *http://host/servlet/ServletName*
- **Speziellerer Speicherort (Klassen in Paketen):**
  install_dir/servers/default/default-app/WEB-INF/classes/packageName
- **Entsprechende URL (Servlets in Paketen):**
  http://host/servlet/packageName.ServletName

In JAR-Dateien gebündelte Servlet- und Hilfsklassendateien

- **Speicherort:** *install_dir/servers/default/default-app/WEB-INF/lib*
- **Entsprechende URLs (Servlets):**
  http://host/servlet/ServletName
  http://host/servlet/packageName.ServletName

1.9.3 ServletExec

HTML und JSP-Seiten

- **Haupt-Speicherort.** *install_dir/public_html*
- **Entsprechende URL:** *http://host/SomeFile.html http://host/SomeFile.jsp*
- **Speziellerer Speicherort (beliebiges Unterverzeichnis):**
  install_dir/public_html/SomeDirectory
- **Entsprechende URLs:**
  http://host/SomeDirectory/SomeFile.html
  http://host/SomeDirectory/SomeFile.jsp

Individuelle Servlet- und Hilfsklassendateien

- **Haupt-Speicherort (Klassen ohne Pakete):** *install_dir/Servlets*
- **Entsprechende URL (Servlets):** *http://host/servlet/ServletName*
- **Speziellerer Speicherort (Klassen in Paketen):** *install_dir/Servlets/packageName*
- **Entsprechende URL (Servlets in Paketen):** *http://host/servlet/packageName.ServletName*

In JAR-Dateien gebündelte Servlet- und Hilfsklassendateien

- **Speicherort:** *install_dir/Servlets*
- **Entsprechende URLs (Servlets):**
  http://host/servlet/ServletName
  http://host/servlet/packageName.ServletName