PEARSON Education

ADDISON-WESLEY

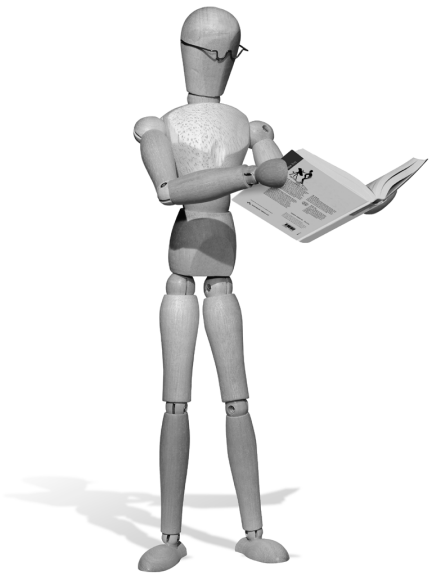
open source library

Lajos Moczar

Tomcat 5

Einsatz in Unternehmensanwendungen mit JSP und Servlets

Übersetzt von Petra Alm



ADDISON-WESLEY

An imprint of Pearson Education

München • Boston • San Francisco • Harlow, England
Don Mills, Ontario • Sydney • Mexico City
Madrid • Amsterdam

3 Administrationsgrundlagen

Was genau unter der Administration von Tomcat zu verstehen ist, lässt sich nur sehr schwer definieren. Wie bei vielen Enterprise-Anwendungen bedeutet Administration alles – vom Starten und Beenden des Programms bis hin zur Verwaltung der Ressourcen, von denen Tomcat abhängig ist. Ich werde zunächst einige Dinge definieren, die für Sie *vielleicht* zur Administration von Tomcat gehören. Bei einigen dieser Dinge handelt es sich ganz eindeutig um Administrationsaufgaben, bei anderen dagegen ist dies eher eine Frage der jeweiligen Situation. Einen Aspekt der Administration, nämlich die Konfiguration, wurde bereits im letzten Kapitel behandelt.

In diesem Kapitel beginnen wir mit den verschiedenen Möglichkeiten, wie Tomcat gestartet und ausgeführt werden kann. Anschließend werden wir uns ausführlich den Ressourcen von Tomcat widmen und diese so konfigurieren, dass sie einen Datenbankverbindungspool bilden. Diesen werden wir für eine neue JSP-Datei (JavaServer Page) verwenden, die eine Liste der in einer Datenbank gespeicherten Tomcat-Ressourcen anzeigt. Im nächsten Kapitel werde ich meine Ausführungen über die Tomcat-Administration ausdehnen und mich dem Thema Sicherheit widmen.

3.1 Einführung

In Tabelle 3.1 sind einige der Punkte aufgeführt, die meiner Erfahrung nach zur Administration von Tomcat gehören. Vermutlich werden nicht alle für Sie relevant sein, ebenso wenig wie auszuschließen ist, dass Administrationsaufgaben, die Sie durchführen, hier fehlen.

Aufgabe	Anmerkung
Installation/Upgrade	Hierbei handelt es sich ganz offensichtlich um eine Administrationsaufgabe. Umfasst eine Art Abnahmeprüfung, der neue Versionen vor dem Upgrade zu unterziehen sind. Wir haben mit der Behandlung dieses Aspekts in Kapitel 1, Tomcat-Schnellstart, begonnen und werden das Thema in Kapitel 17, Tomcat-Administration, weiter vertiefen.
Ausführung	Dazu zählen das Starten und Beenden sowie der Autostart von Tomcat. Dieser Punkt wird in diesem Kapitel kurz angesprochen und in Kapitel 17 ausführlich behandelt.

Tabelle 3.1: Aufgabenbereiche der Tomcat-Administration

Aufgabe	Anmerkung
Basiskonfiguration	Ich persönlich betrachte die Setup- oder Basiskonfiguration einer Tomcat-Installation als einen eigenständigen Aufgabenbereich, auch wenn dieser Aspekt üblicherweise dem Bereich der »Installation« zugerechnet wird. Teil dieses Aufgabenbereichs ist auch das, was ich als Produktionalisierung der Distribution bezeichne und mit der wir im letzten Kapitel begonnen haben.
Anwendungsadministration	Hierzu zählen Deployment, Upgrade sowie die allgemeine laufende Wartung der Anwendungen, die innerhalb Ihrer Tomcat-Installation vorhanden sind.
Monitoring	Hierzu zählt das Überwachen von Fehlern, Ladevorgängen und Ressourcen. Halten Sie sich an den Grundsatz: Kein Deployment ohne Monitoring.
Ressourcenadministration	Ein weiter Bereich, bei dem es um die Konfiguration der Ressourcen geht, die für Tomcat und die unter Tomcat installierten Anwendungen benötigt werden. Dies kann so einfach sein wie die Konfiguration von Tomcat für den Zugriff auf diese Ressourcen, beispielsweise eine bestehende Datenbank. Andererseits kann es sich aber auch um so komplexe Aufgaben handeln wie die Installation, Konfiguration und Einrichtung eben dieser Ressourcen.
Sicherheit	Dieser Aufgabenbereich kann eng oder weit gefasst sein. Sollten Sie bereits über ein Sicherheitskonzept und eine entsprechende Infrastruktur verfügen, kann Tomcat unter Umständen leicht in diese integriert werden. Wurden solche Vorkehrungen noch nicht getroffen, so könnte diese Aufgabe zumindest den Aufbau einer entsprechenden Basisinfrastruktur bedeuten.
Webserver-Integration	In Tomcat-Produktionsinstallationen wird Tomcat meist ein Webserver wie Apache oder IIS (Internet Information Services) vorangestellt. Dieses Thema werde ich in Kapitel 20, Tomcat-Webserver-Integration, behandeln.

Tabelle 3.1: Aufgabenbereiche der Tomcat-Administration (Forts.)

Wenden wir uns nun der Ausführung von Tomcat zu.

3.2 Tomcat 5 ausführen

Bisher habe ich Sie Tomcat durch die Eingabe von `catalina run` starten lassen. Diejenigen unter Ihnen, die bereits mit Tomcat vertraut sind, werden sich vielleicht fragen, warum ich ausgerechnet diese Methode gewählt habe. Wie bereits erläutert, ziehe ich diese Methode vor, weil ich bei eventuellen Startproblemen die entsprechenden Fehlermeldungen dann in demselben Konsolenfenster angezeigt bekomme, in dem ich auch den Startbefehl eingegeben habe. Für Entwicklungsarbeiten (oder für die Erst-

einrichtung einer Tomcat-Instanz) hat sich dies bewährt. Für den operativen Betrieb ist dagegen in der Regel eine andere Methode zu bevorzugen. Mehr dazu in Abschnitt 3.2.2.

3.2.1 Das bin-Verzeichnis von Tomcat 5

Bevor ich die verschiedenen Optionen erläutere, werfen Sie zunächst einen Blick auf den Inhalt des *bin*-Verzeichnisses, das in Abbildung 3.1 zu sehen ist.

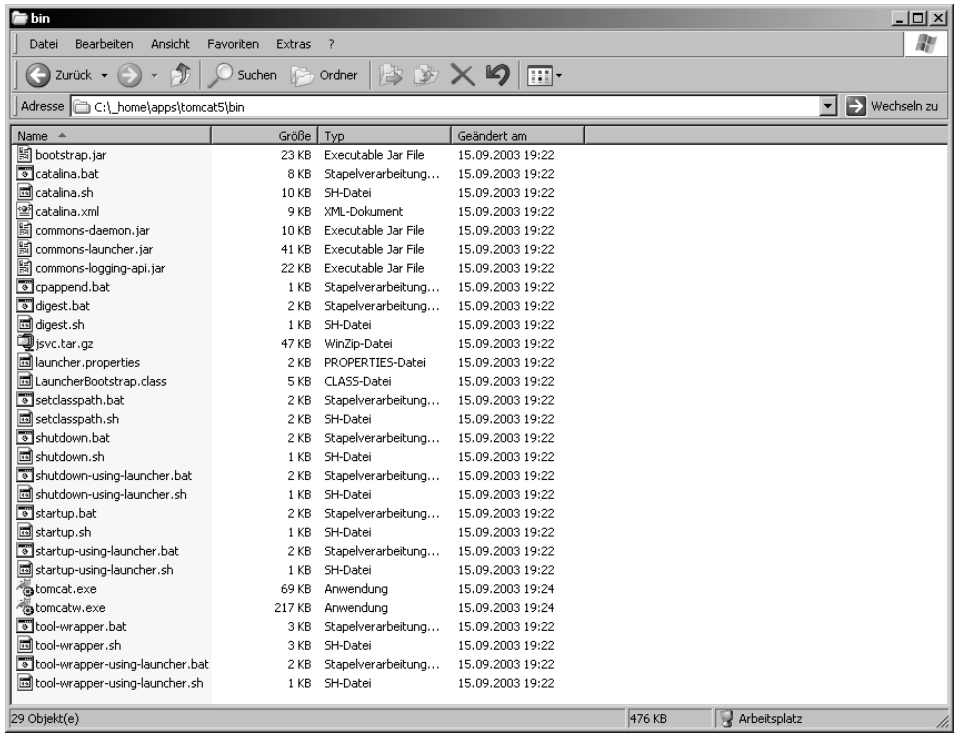


Abbildung 3.1: Inhalt des bin-Verzeichnisses von Tomcat 5

Erschreckend, nicht wahr? Nun, wenn ich einige Dateien in Gruppen zusammenfasse, sieht es schon nur noch halb so schlimm aus.

Zunächst sind da verschiedene jar-Dateien (oder Java-Bibliotheken), auf die Tomcat beim Starten zugreift. Diese sind in Tabelle 3.2 beschrieben.

jar-Datei	Beschreibung
bootstrap.jar	Enthält die grundlegenden Klassendateien für den Tomcat-Start.
commons-daemon.jar	Eine Reihe von Klassen aus dem Jakarta Commons Project, die benötigt werden, um Java-Programme als Daemons auszuführen.
commons-launcher.jar	Ein Java-Launcherprogramm, das aus dem Tomcat-Projekt hervorgegangen und unter das Commons-Projekt gestellt wurde.
commons-logging-api.jar	Eine Anwendungsprogrammierschnittstelle (API) für die Protokollierung, die es Tomcat ermöglicht, jedes Logging-Paket zu verwenden, das dieser API folgt.

Tabelle 3.2: jar-Dateien für den Tomcat-Start

Hinweis

Für den Fall, dass Sie Ihnen das »Jakarta Commons«-Projekt, *http://jakarta.apache.org/commons/*, kein Begriff sein sollte: Es handelt sich um ein Projekt zur Entwicklung wieder verwendbarer Java-Komponenten, aus dem bereits eine Vielzahl von Komponenten für Beans, Protokollierung, Connection Pooling, Collections usw. hervorgegangen sind. Etliche dieser Komponenten verdanken ihre Entstehung Tomcat. Seien Sie also nicht überrascht, wenn Sie auf Beispiele treffen, in denen auch Tomcat eine Rolle spielt. Andererseits haben die Entwickler von Tomcat die Commons-Komponenten für nützlich genug erachtet, um sie in ihre eigene Projektumgebung aufzunehmen. Schauen Sie sich die zur Verfügung stehenden Komponenten an! Sie können Ihnen eventuell wertvolle Zeit sparen.

Weiterhin können wir die Hauptskripte identifizieren, *catalina.bat* und *catalina.sh*, die wir bereits verwendet haben. Diese werden wiederum von einer Reihe anderer Skripte aufgerufen, die in Tabelle 3.3 aufgeführt sind.

Skript	Beschreibung
catalina.bat/.sh	Wichtigstes Kontrollskript von Tomcat
shutdown.bat/.sh	Ein Hilfsskript, das das Hauptskript mit dem `stop`-Befehl aufruft
startup.bat/.sh	Ein Hilfsskript, das das Hauptskript mit dem `start`-Befehl aufruft

Tabelle 3.3: Start-Skripte von Tomcat

Für die Installation von Tomcat als Dienst stehen zwei Windows-Exe-Dateien zur Verfügung: *tomcat.exe* und *tomcatw.exe*. Der einzige Unterschied zwischen den beiden ist, dass Letztere eine GUI-Version ausführt.

Der »Launcher« ist ein neues Konzept von Tomcat 5. Er nutzt Apache Ant, um Tomcat zu starten, zu beenden oder um verschiedene Tomcat-Tools aufzurufen, und greift hierzu auf die Dateien *LauncherBootstrap.class*, *catalina.xml*, *launcher.properties* und **-using-launcher** zu.

Tabelle 3.4 beschreibt eine Reihe weiterer Dateien – größtenteils Hilfsskripte für verschiedene Aufgaben – aus dem *bin*-Verzeichnis.

Datei	Beschreibung
cpappend.bat	Windows-Shellskript zur Erstellung einer `CLASSPATH`-Variablen
digest.bat/.sh	Skript für die Verschlüsselung von Passwörtern (siehe Kapitel 4, Abschnitt Passwörter verschlüsseln).
jsvc.tar.gz	Ein Paket für die Ausführung von Tomcat als Daemon (siehe Kapitel 17).
setclasspath.bat/.sh	Skript für die Erstellung der `CLASSPATH`-Variablen, das von *catalina.bat/.sh* aufgerufen wird.
tool-wrapper.bat/.sh *tool-wrapp-using-launcher.bat/.sh*	Skripts für die Ausführung von Tools oder anderen ausführbaren Java-Dateien, die denselben allgemeinen Classloader wie Tomcat benötigen. Sie werden beispielsweise verwendet, um das Skript *digest.bat/.sh* auszuführen.

Tabelle 3.4: Verschiedene Dateien aus dem bin-Verzeichnis

3.2.2 Optionen für die Tomcat-Ausführung

Nun komme ich zu den verschiedenen Möglichkeiten Tomcat zu starten. Grundsätzlich gibt es drei Optionen: Sie können Tomcat manuell über die Kommandozeile starten und als Anwendung laufen lassen, bis Sie diese beenden. (So haben wir es bisher gemacht.) Sie können Tomcat aber auch als Daemon (unter Linux) oder Dienst (unter Windows) starten, sodass das Programm beim Booten des Rechners automatisch startet. Die dritte Möglichkeit ist, Tomcat über den Ant-basierten Launcher zu starten. Letztere werde ich in Kapitel 17, Tomcat-Administration, ausführlich erläutern.

Tomcat 5 manuell ausführen

Wenn Sie damit zufrieden sind, Tomcat so zu starten, wie ich es Ihnen zu Beginn gezeigt habe, können Sie dies weiterhin tun. Sollten Sie jedoch die »Standardmethode« bevorzugen, so geben Sie unter Unix ein:

```
startup.sh
```

Unter Windows geben Sie ein:

```
startup
```

Beachten Sie, dass `startup` nichts anderes macht, als *catalina.bat* bzw. *catalina.sh* mit dem `start`-Argument aufzurufen.

Zum Beenden von Tomcat geben Sie unter Unix dementsprechend ein:

```
shutdown.sh
```

Und unter Windows:

```
shutdown
```

In diesem Fall ruft der Befehl *catalina.bat* bzw. *catalina.sh* mit dem `stop`-Argument auf.

Tomcat 5 automatisch ausführen

In Produktionssystemen werden Sie Tomcat mit dem Booten des Rechners starten und als Daemon (bzw. Dienst) oder automatischen, systeminternen Prozess laufen lassen wollen. Unter Unix werden Sie hierzu in der Regel ein Startskript erzeugen, das das Betriebssystem beim Booten und Herunterfahren des Rechners ausführt. Unter Windows werden Sie zu diesem Zweck Tomcat als Dienst installieren.

Essentiell für einen Daemon-Prozess ist, dass die Anwendung auf Signale des Betriebssystems reagiert und entsprechende Start- und Shutdown-Funktionen ausführt. Bei einem Unix-Daemon reagiert die Anwendung auf Signale des Betriebssystems. Bei einem Windows-Dienst implementiert die Anwendung verschiedene Methoden, die das Betriebssystem aufruft.

Ein Java-Prozess ist nicht zwangsläufig ein Daemon: Er muss vielmehr von einem Client über die `main`-Methode aufgerufen werden, die der Eintrittspunkt zur Anwendung ist. Um einen Java-Prozess als Daemon auszuführen, müssen Sie einen nativen Wrapper bereitstellen, der dem Betriebssystem eine geeignete Schnittstelle für die Ausführung als Daemon anbietet. Die Tomcat-Distribution ist mit einem solchen Wrapper (oder »Daemon-Erzeuger«) ausgestattet, der aus dem »Jakarta Commons Daemon«-Projekt hervorgegangen ist. Dieses Projekt gibt es konkret in zwei Ausführungen: `JSVC`, dem Daemon-Erzeuger für Unix, sowie `procrun` für Windows. Letzterer wird standardmäßig mit Tomcat ausgeliefert und zwar als *tomcat.exe* und *tomcatw.exe* im Verzeichnis *$CATALINA_HOME/bin*. Auf JSVC werde ich in Kapitel 17 ausführlich eingehen. Unter Unix haben Sie darüber hinaus noch die Möglichkeit, ein einfaches Startup-/Shutdown-Skript zu erzeugen (was wir im Folgenden tun werden).

Tomcat 5 unter Unix automatisch ausführen

Um unter Unix einen Prozess automatisch ausführen zu lassen, geht man grundsätzlich so vor, das man ein Start-Skript aufsetzt und in einem passenden Verzeichnis speichert, wo es vom Betriebssystem gefunden und beim Starten ausgeführt werden

kann. Wie Sie wahrscheinlich wissen, läuft Unix auf mehreren run- oder init-Ebenen. Auf jeder Ebene werden verschiedene Dienste gestartet. Obwohl technisch gesehen zehn Ausführungsebenen zur Verfügung stehen, sind für Sie in der Regel nur von Bedeutung: Ebene 1, auf der das System im SingleUser-Modus läuft; Ebene 2, auf der das System im MultiUser-Modus mit NFS (Network File System) läuft; Ebene 3 für den vollständigen MultiUser-Modus und Ebene 5 für den vollständigen MultiUser-Modus plus X-Windows.

Um Unix mitzuteilen, welche Dienste auf welcher Ausführungsebene starten sollen, sind in einer Reihe von Verzeichnissen unter */etc/rc.d* Startup-Skripte für die verschiedenen Dienste zu finden. Abbildung 3.2 zeigt ein typisches Beispiel hierfür (auf einem Linux-Server).

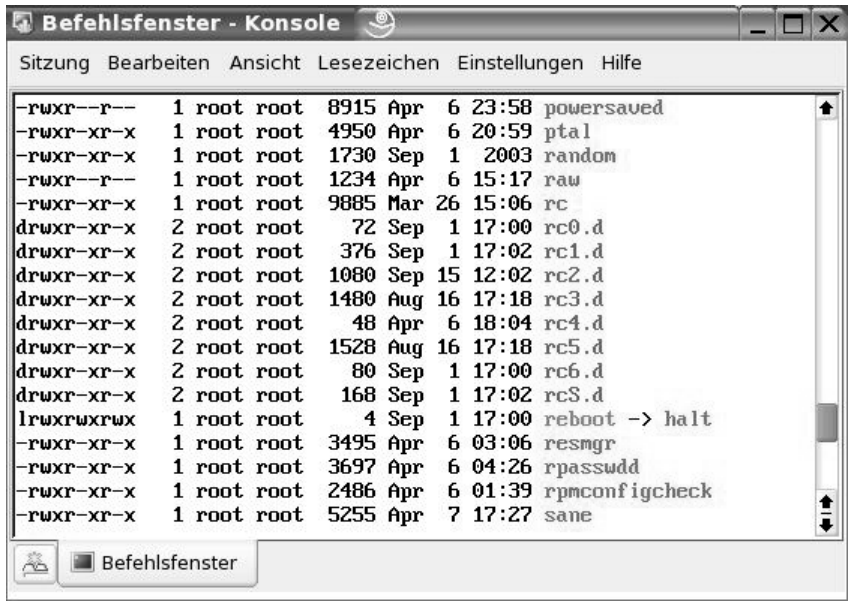


Abbildung 3.2: Inhalt von Unix /etc/rc.d

Hier sehen Sie für jede der Unix-Ausführungsebenen 0 bis 6 ein Verzeichnis *rc[n].d*, wobei `[n]` die Ausführungsebene bezeichnet. Gemäß Konvention werden die Skripte in das Verzeichnis *init.d* oder direkt in *rc.d* gestellt. Anschließend werden in den verschiedenen Ausführungsebenenverzeichnissen je nach Bedarf symbolische Verknüpfungen angelegt. Dabei können Sie auch festlegen, in welcher Reihenfolge die Skripte einer Ausführungsebene ausgeführt werden: die symbolischen Verknüpfungen werden dazu nach dem Muster `S[n][skriptname]` benannt, wobei `[n]` eine zweistellige Zahl und `[skriptname]` der eigentliche Name des Skripts im Verzeichnis `/etc/rc.d/init.d` ist, auf das verwiesen wird. Einen vergleichbaren Mechanismus gibt es für das Beenden von Diensten. Dabei führt Unix die Skripte nacheinander aus, während es die verschiedenen Ausführungsebenen von oben nach unten durchläuft. Skripte, die ausge-

führt werden müssen, um einen Dienst zu beenden, tragen die Bezeichnung `K[n][skriptname]`, wobei `[n]` eine zweistellige Zahl und `[skriptname]` der eigentliche Name des Skripts im Verzeichnis */etc/rc.d/init.d* ist, auf das verwiesen wird. Damit dies funktioniert, müssen Sie das konkrete Skript so konfigurieren, dass es sowohl für das Starten als auch für das Beenden geeignet ist. Und genau das werden Sie in einer Minute tun.

Üblicherweise startet Tomcat (oder allgemein jeder Webserver) auf Ausführungsebene 3. Ich persönlich ziehe es vor, Tomcat erst zu starten, nachdem alle Datenbankserver, von denen meine Anwendung abhängig ist, gestartet sind. Allerdings sollte der Start erfolgen, bevor Apache startet (sofern Tomcat hinter einem Apache-Server läuft). Also erstelle ich unter */etc/rc.d/init.d* zunächst eine Skriptdatei mit Namen `tomcat5`, die in etwa so aussieht wie Listing 3.1.

Listing 3.1: rc-Datei von Tomcat

```
#!/bin/sh
#
# Start-Skript für Tomcat
#

case "$1" in
  start)
        echo -n "Tomcat starten"
        JAVA_HOME="/usr/java/jdk1.3.1_08" ; export JAVA_HOME && ↩
                /usr/tomcat5/bin/startup.sh
        ;;
  stop)
        echo -n "Tomcat beenden"
        JAVA_HOME="/usr/java/jdk1.3.1_08" ; export JAVA_HOME && ↩
                /usr/tomcat5/bin/shutdown.sh
        ;;
  restart)
        $0 stop
        $0 start
        ;;
  *)
        echo "Usage: $0 {start|stop|restart}"
        exit 1
esac

exit 0
```

Dieses Skript erwartet mit einem der folgenden drei Argumente aufgerufen zu werden: `start`, `stop` oder `restart`. Alles, was ich zum Starten tun muss, ist die von Tomcat benötigte Umgebungsvariable `$JAVA_HOME` zu setzen und dann *startup.sh* im *bin*-Verzeichnis der Tomcat-Installation aufzurufen. Und das war's auch schon!

Nachdem Sie das Skript erstellt haben, müssen Sie für dieses eine symbolische Verknüpfung im entsprechenden *rc*-Verzeichnis erzeugen. Da ich für Tomcat in der Regel das Verzeichnis *rc3.d* wähle, gebe ich ein:

```
cd /etc/rc.d/rc3.d
ln -f ../init.d/tomcat5 K90tomcat5
ln -f ../init.d/tomcat5 S90tomcat5
```

Hiermit stellen Sie einen Start-Link, `S90tomcat5`, bereit, den das Betriebssystem anschließend mit dem `start`-Argument aufruft. Da Sie Tomcat sicherlich auch in gebührender Weise beenden wollen, wenn Sie den Rechner herunterfahren, müssen Sie auch einen Shutdown-Link bereitstellen, den das Betriebssystem mit dem `stop`-Argument aufruft. Beide Links verweisen auf dasselbe Skript, das Sie vorausschauend für die Bearbeitung dieser beiden Argumente geschrieben haben. Beachten Sie auch, dass ich als Sequenznummer 90 gewählt habe. Grund hierfür ist, dass Tomcat eines der letzten Programme sein soll, die auf Ausführungsebene 3 gestartet werden. Also stelle ich sicher, dass MySQL (oder welche Datenbank ich auch verwenden mag) eine niedrigere Sequenznummer hat und somit zuerst startet. Ebenso trage ich dafür Sorge, dass Apache nach Tomcat startet, indem ich ihm eine höhere Sequenznummer gebe.

In Kapitel 17 werden wir uns mit JSVC aus dem »Commons Daemon«-Projekt befassen, um Tomcat als echten Daemon laufen zu lassen.

3.2.3 Tomcat 5 als Dienst unter Windows ausführen

Unter Windows gibt es zwei Möglichkeiten, Tomcat zu installieren. Die einfachste Möglichkeit ist, die Windows-Installation von Tomcat herunterzuladen und auszuführen. Wenn Sie dies tun, wird Tomcat automatisch als Dienst installiert. Die zweite Möglichkeit ist, *tomcat.exe* unter *$CATALINA_HOME/bin* manuell auszuführen.

Verwendung des Tomcat-Installer für Windows

Gehen Sie zurück zur Binaries-Download-Seite von Tomcat (*http://jakarta.apache.org/site/binindex.cgi*). Dann suchen Sie den Abschnitt TOMCAT 5 und laden die exe-Distribution herunter. Nach dem Herunterladen starten Sie die Installation mit einem Doppelklick. Sie werden aufgefordert, den Speicherort für die Installation (standardmäßig *c:\Programme\Apache Software Foundation\Tomcat 5.0*), den Ort der Java-Installation und einige andere Angaben zu machen. Nach Abschluss der Installation steht Ihnen dann ein Dienst namens `Apache Tomcat` zur Verfügung.

Tomcat manuell als Dienst installieren

Vergewissern Sie sich zunächst, dass Sie unter SYSTEMSTEUERUNG/SYSTEM/ERWEITERT/UMGEBUNGSVARIABLEN die System-Umgebungsvariablen `%JAVA_HOME%` und `%CATALINA_HOME%` gesetzt haben. Anschließend führen Sie das Skript aus Listing 3.2 aus.

Listing 3.2: Skript für die Installation des Tomcat 5-Dienstes

```
%CATALINA_HOME%\bin\tomcat //IS//Tomcat5 --DisplayName "Tomcat5" \
--Description "Tomcat5" \
--ImagePath "%CATALINA_HOME%\bin\bootstrap.jar" \
--StartupClass org.apache.catalina.startup.Bootstrap;main;start \
--ShutdownClass org.apache.catalina.startup.Bootstrap;main;stop \
--Java "%JAVA_HOME%\jre\bin\client\jvm.dll" \
--JavaOptions -Xrs \
--StdErrorFile "%CATALINA_HOME%\logs\stderr.log" \
--StdOutputFile "%CATALINA_HOME%\logs\stdout.log"
```

Achtung

Dieses Skript ist spezifisch für JDK 1.4.2. Für JDK 1.3.x müssen Sie einen anderen Pfad für die Option `--Java` angeben, zum Beispiel: `--Java "%JAVA_HOME%\jre\bin\classic\jvm.dll" \`.

Wenn Sie diesen Befehl eingeben, schreiben Sie ihn unbedingt in eine einzige Zeile. Ich habe die Zeilenumbrüche nur eingefügt, um die Lesbarkeit zu verbessern. Der Befehl fordert Windows auf, Tomcat als Dienst mit dem Namen `Tomcat5` zu installieren. Der Dienst führt *tomcat.exe*, wobei es sich letztendlich um `procrun` aus dem »Commons Daemon«-Projekt handelt, mit verschiedenen Parametern aus.

Da der Dienst ohne Rückmeldung installiert wird, müssen Sie unter SYSTEMSTEUERUNG/VERWALTUNG/DIENSTE gehen, um zu überprüfen, ob der Dienst `Tomcat` dort, wie im Skript definiert, als `Tomcat5` aufgeführt ist.

Tomcat-Dienst ausführen

Im Dialog DIENSTE/SYSTEMSTEUERUNG/VERWALTUNG/DIENSTE können Sie einstellen, dass Tomcat beim Booten des Rechners automatisch startet. Über dieses Fenster können Sie Tomcat auch manuell starten und beenden. Wenn Sie möchten, können Sie den Dienst auch über die Kommandozeile ausführen und Tomcat starten mit:

```
net start Tomcat5
```

Beenden können Sie Tomcat durch Eingabe von:

```
net stop Tomcat5
```

Und wenn Sie den Dienst wieder entfernen wollen, geben Sie einfach ein:

```
%CATALINA_HOME%\bin\tomcat //RS/Tomcat5
```

Je nachdem, wie Sie Ihren Dienst installiert haben, kann sein Name unter Umständen von dem meines Dienstes abweichen. Wenn ich den Dienst Tomcat 5 manuell installiere, nenne ich ihn in der Regel `Tomcat5`; das Windows-Installationsprogramm nennt ihn `Apache Tomcat`.

3.3 Tomcat 5-Ressourcen

Selten lässt man Tomcat isoliert laufen. Gut, vielleicht für eine schnell zusammengebastelte Anwendung auf einem Laptop, aber in den meisten Fällen benötigt man für die Arbeit zusätzlich irgendeine Datenbankanbindung, einen E-Mail-Mechanismus und möglicherweise Hooks in einen LDAP-Server (Lightweight Directory Access Protocol) oder einen anderen Authentifizierungsmechanismus. Diese Dinge nennen wir häufig »Ressourcen«. Allerdings bezeichnet dieser Begriff noch vieles mehr. Es handelt sich in der Tat um einen häufig verwendeten Begriff, der je nach Kontext alles und nichts bedeuten kann. Als ich mich zum ersten Mal mit der Entwicklung komplexer Webanwendungen befasst habe, hat mich die Bedeutungsvielfalt dieses Begriffs sehr verwirrt. Um genau zu sein, sind nur sehr wenige der Begriffe, mit denen wir so leicht um uns werfen, wohl definiert. Aus diesem Grunde möchte ich zunächst einige Definitionen geben, die Ihnen bei der weiteren Arbeit mit diesem Buch nützlich sein werden.

3.3.1 Definitionen

Auch wenn so mancher Begriff eindeutig zu sein scheint, möchte ich zunächst eine Reihe von Definitionen geben und dabei mit dem Begriff *Webanwendung* beginnen. Was ist eine Webanwendung? Da es in diesem Buch hauptsächlich um das Schreiben und die Installation dieser Ungetüme geht, sollten wir auch eine genaue Vorstellung von dem haben, worüber wir eigentlich sprechen! Lassen Sie mich daher zunächst festhalten, dass eine Webanwendung ein Paket von Funktionen ist, deren Zweck darin besteht, für eine bestimmte Gruppe von URLs Anfragen von Web-Clients zu beantworten. Mit anderen Worten: Webanwendungen werden dafür konfiguriert, auf eine oder mehrere URLs zu reagieren und zu jeder Anfrage die passende Antwort zurückzuliefern, wobei die Antworten in der Regel aus Daten oder Inhalten bestehen (beispielsweise dem Inhalt einer Seite, auf der die von einem Unternehmen angebotenen Produkte aufgeführt sind). Die primäre Funktion einer Webanwendung besteht üblicherweise darin, irgendwelche HTML-Daten an den Web-Client (in der Regel ein Browser) zurückzuliefern. Was eine Webanwendung jedoch interessant macht, ist, dass sie außer der Bereitstellung von Inhalten noch weitere Funktionen erfüllen kann, beispielsweise Daten in eine Datenbank einspielen, die Belastung einer Kreditkarte genehmigen oder eine E-Mail verschicken.

Nachdem nun geklärt wäre, was eine Webanwendung ist, können wir untersuchen, was eine Webanwendung ausmacht. Für mich sind das grob gesehen vier Komponenten, siehe Abbildung 3.3.

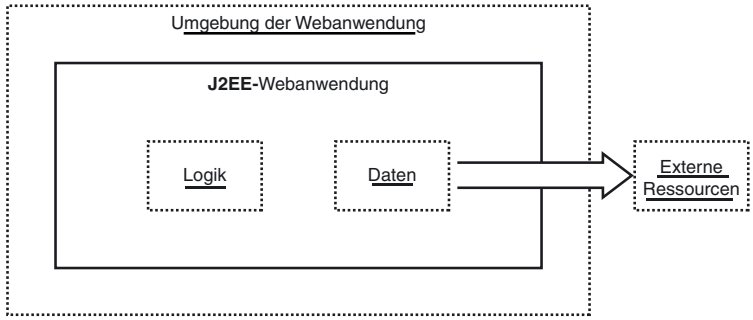


Abbildung 3.3: Komponenten einer Webanwendung

Zunächst besteht eine Webanwendung aus Daten – den Daten, die Sie in Ihre HTML-Dateien schreiben oder in Ihrer Datenbank erfassen. Zweitens verfügt sie über eine Logik, die im Code einer Programmiersprache erfasst ist und zur Erfüllung einer Aufgabe (darunter auch das Zurückliefern von Daten an den Client) ausgeführt wird. Drittens hat eine Webanwendung einen Kontext, womit die Umgebung gemeint ist, in der sie ausgeführt wird. Dieser Kontext (sprich diese Umgebung) definiert verschiedene Parameter, die die Ausführung der Anwendung beeinflussen können. Zuletzt kann die Anwendung noch über Dienste verfügen, die für ihre Funktion erforderlich sind und die sowohl innerhalb als auch außerhalb der Umgebung der Webanwendung angesiedelt sein können. Bei den Webanwendungen, die wir hier nutzen, werden J2EE-Webanwendungen genannt, weil sie ihre Inhalte mit Java erzeugen.

Nun kann ich endlich definieren, was unter einer Ressource zu verstehen ist. Leider wird der Begriff häufig für alle möglichen Komponenten einer Anwendung verwendet, zum Beispiel statische Dateien, Klassendateien, Umgebungsparameter und externe Dienste. Da ich es aber gern genau habe, werde ich den Begriff Ressource ganz streng als einen Dienst definieren, von dem eine Webanwendung abhängig ist. Um dies ganz deutlich zu machen, werde ich diese Ressourcen als Dienstressourcen bezeichnen. In diesem Sinne sind ein Datenbankserver, ein LDAP-Server, ein E-Mail-Server, ein Authentifizierungsmechanismus usw. alles Beispiele für Dienstressourcen. All diesen ist gemeinsam, dass sie außerhalb der Webanwendung angesiedelt sind.

Läuft eine J2EE-Webanwendung innerhalb eines Containers wie Tomcat, so stellt dieser verschiedene Dienste für die Webanwendung bereit. Zum einen lädt Tomcat die Webanwendung, führt sie aus, kümmert sich um die TCP/IP-Kommunikation mit Clients und übermittelt Anfragen und Antworten zwischen den Clients und der Anwendung. Darüber hinaus kann Tomcat statische Dateien (HTML-Dateien, Images, CSS-Dateien [Cascading Style Sheets] usw.) lesen und die Inhalte an den Client zurückliefern. Tomcat kann JSPs lesen, sie in Java-Dateien überführen, kompilieren und für die Anwendung ausführen. Diese Funktionen sind in der Tat sehr nützlich, befreien sie doch die Anwendungen von der Pflicht, diese Aufgaben selbst

auszuführen. Auch was das Thema Ressourcen betrifft, ist Tomcat für die Webanwendung von großem Wert, da das Programm Dienstressourcen für die Anwendung bereitstellen kann. Diese Dienste können sowohl innerhalb als auch außerhalb von Tomcat liegen; in letzterem Fall kümmert sich Tomcat um die Verbindung.

In Kapitel 21, JNDI-Ressourcen, verwalten, werde ich erläutern, wie Tomcat für verschiedene Ressourcenarten konfiguriert werden kann. In diesem Kapitel werde ich mich jedoch ausschließlich auf Datenbankressourcen konzentrieren. Im Folgenden werden wir eine JSP-Datei erstellen, die einer Datenbanktabelle einige Daten entnimmt und diese dem Client im HTML-Format präsentiert. Der Einfachheit halber werden wir uns dabei auf eine Tabelle mit zwei Spalten beschränken, in der eine Liste nützlicher, für Tomcat relevanter URLs und deren Beschreibung enthalten ist.

Vier Aufgaben liegen vor uns:

- einen Datenbankserver installieren und starten;
- eine Datenbank mit einer einzigen Tabelle erzeugen und diese mit einigen Daten füllen;
- Tomcat für die Verwaltung eines Datenbankverbindungspools konfigurieren;
- eine JSP-Datei anlegen, um mittels der von Tomcat bereitgestellten Datenbankverbindungen Zugriff auf die Tabelle zu erhalten.

In diesem Kapitel verwende ich eine MySQL-Datenbank. Sie können jedoch, falls Sie dies wünschen, jeden Datenbankserver Ihrer Wahl verwenden. Dies setzt allerdings voraus, dass Sie über geeignete Java-Treiber verfügen und die Syntax meiner Beispiele gegebenenfalls entsprechend anpassen.

3.4 Installation von MySQL

MySQL ist derzeit der beliebteste Open Source-Datenbankserver, und mittlerweile so ausgereift und ausgestattet, dass er den großen kommerziellen Paketen durchaus Konkurrenz machen kann. Für den Einstieg gibt es wahrscheinlich keinen besseren Datenbankserver als MySQL, und Beispiele für den Einsatz von MySQL in Kombination mit Java-Anwendungen wie Tomcat gibt es zuhauf.

3.4.1 MySQL herunterladen

Gehen Sie zunächst zur Hauptseite *http://www.mysql.com* und laden Sie die neueste Version (zum Zeitpunkt der Drucklegung dieses Buches war dies 4.0.16) für Ihre Plattform herunter. Auf der Seite *http://www.mysql.com/downloads/mysql-4.0.html* sind alle möglichen Optionen aufgelistet. Für meinen Windows XP-Rechner habe ich die Datei *mysql-4.0.16-win.zip* heruntergeladen und für meinen Linux-Server *mysql-standard-4.0.16-pc-linuxi686.tar.gz*.

Sie benötigen ferner die JDBC-Treiber (Java Database Connectivity) für MySQL, die separat von *http://www.mysql.com/downloads/api-jdbc-stable.html* heruntergeladen werden. Die aktuelle Version 3.0.15 steht als gzip- oder zip-Datei zur Verfügung. Wählen Sie die Variante, die für Sie einfacher ist.

3.4.2 MySQL installieren

Die Installation von MySQL ist recht einfach. Unter Windows doppelklicken Sie auf die heruntergeladene zip-Datei, um diese zu öffnen und die in ihr enthaltene Datei *setup.exe* auszuführen. Sie müssen verschiedene Angaben machen, können aber einfach die Standardvorgaben wählen, und MySQL unter *c:\mysql* installieren.

Unter Unix müssen Sie der `root`-Benutzer sein, um MySQL zu installieren. Sie können die Distribution an jedem beliebigen Speicherort entpacken. Ich wähle in der Regel dazu das Verzeichnis */opt*, doch können Sie – wie gesagt – jedes andere Verzeichnis wählen. Nach der Installation sind noch einige weitere Schritte erforderlich. Zuerst richten Sie eine Gruppe und einen Benutzer namens `mysql`ein:

```
groupadd mysql
useradd -g mysql mysql
```

Nun wechseln Sie mit `cd` in die Distribution und führen das Skript aus, das alle Systemtabellen einrichtet:

```
cd /opt/mysql-standard-4.0.16-pc-linux-i686
scripts/mysql_install_db
chown -R mysql data
chgrp -R mysql .
```

Wie gesagt – dies gilt nur für Installationen unter Unix. Beachten Sie auch, dass mein Verzeichnis der Installation der Version 4.0.16 entspricht. Passen Sie das Verzeichnis entsprechend Ihrem Speicherort und Ihrer Version an.

Was das JDBC-Treiberpaket betrifft, so müssen Sie dieses ebenfalls entpacken und nach der jar-Datei *mysql-connector-java-3.0.15-stable-bin.jar* suchen. Sie erinnern sich an meine Ausführungen zu *$CATALINA_HOME/common* und *$CATALINA_HOME/shared* im letzten Kapitel? Dort haben Sie gelernt, dass die Objekte aus diesen beiden Verzeichnissen allen Webanwendungen zur Verfügung stehen. Bei dem erstgenannten Verzeichnis gilt dies auch für Tomcat, bei letzterem hingegen nicht. Dies hier ist ein Beispiel für eine Situation, in der Sie das *common*-Verzeichnis verwenden müssen. Da Tomcat den Datenbankverbindungspool pflegen soll, müssen Sie die jar-Datei mit den JDBC-Treibern für MySQL unter *$CATALINA_HOME/common/lib* ablegen.

Bevor Sie nun MySQL starten, möchte ich noch darauf hinweisen, dass MySQL werksmäßig für die Anmeldung als Admin-Benutzer `root` ohne Passwort eingerichtet ist. Ganz offensichtlich bedarf es hier einer Überarbeitung, doch dies ist nicht Thema dieses Kapitels. Falls Sie die Anmeldung umstellen oder eine bereits vorhandene

MySQL-Installation verwenden, bei der diese Standardwerte ersetzt wurden, denken Sie daran die in den Beispielen enthaltenen Login-Daten entsprechend anzupassen.

3.4.3 MySQL unter Linux starten

MySQL unter Linux zu starten, ist nicht schwer: Gehen Sie einfach in das Distributionsverzeichnis und geben Sie folgenden Befehl ein:

```
bin/mysqld_safe --user=mysql &
```

Sie erhalten eine Reihe von Startmeldungen. Anschließend können Sie Folgendes in die Kommandozeile eintippen:

```
bin/mysql
```

Bei korrekter Eingabe wird das Kommandozeilentool in etwa folgende Meldung anzeigen:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2 to server version: 4.0.16-standard

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Jetzt können Sie loslegen.

Sollten Sie aus irgendeinem Grund keine derartige Meldung erhalten, suchen Sie im Verzeichnis *data* nach einer **.err*-Datei. In der Regel lässt sich dieser Datei entnehmen, warum der Server nicht starten konnte.

3.4.4 MySQL unter Windows starten

Unter Windows ist es am einfachsten, MySQL als Dienst zu installieren und diesen dann zu starten bzw. zu beenden. Gehen Sie dazu in das *bin*-Verzeichnis der MySQL-Installation, standardmäßig *c:\mysql*, und geben Sie ein:

```
mysqld-nt --install
```
[1]

Sie sollten folgende Meldung erhalten: `Service successfully installed`. Gehen Sie danach zu SYSTEMSTEUERUNG/VERWALTUNG/DIENSTE und suchen Sie dort den Dienst `mysql`, den Sie starten. Wenn Sie möchten, können Sie AUTOSTART einstellen, um den Dienst gleichzeitig mit Windows zu starten.

1 `mysqld --install` für Windows 95/98/Me

Jetzt können Sie Ihre Installation testen, indem Sie in das *bin*-Verzeichnis von MySQL zurückwechseln und eingeben:

```
mysql
```

Sie sollten dieselbe Meldung erhalten, die ich weiter oben im Abschnitt 3.4.3 aufgeführt habe.

3.4.5 Tabellen anlegen

MySQL ist standardmäßig mit einer Systemdatenbank namens `mysql` ausgestattet. Wir wollen aber unsere eigene Datenbank, `unleashed`, anlegen. Starten Sie hierzu das Client-Programm und geben Sie ein:

```
create database unleashed;
```

Nun können Sie eine Tabelle erstellen und verschiedene Tomcat-relevanten Ressourcen eintragen. Um diese Aufgabe zu vereinfachen, habe ich ein passendes SQL-Skript geschrieben, siehe Listing 3.3.

Listing 3.3: Skript zum Anlegen einer Tabelle

```
use unleashed;

create table TomcatResources (
URL  varchar(255) not null,
Titel varchar(255) not null
);

insert into TomcatResources values (
'http://www.onjava.com/pub/a/onjava/2003/06/25/tomcat_tips.html?page ↩
=2&x-showcontent=off',
'ONJava.com: Top Ten Tomcat Configuration Tips [Jun. 25, 2003]');

insert into TomcatResources values  (
'http://jakarta.apache.org/tomcat',
'The Jakarta Site - Apache Tomcat');

insert into TomcatResources values (
'http://www.moreservlets.com/Using-Tomcat-4.html',
'Apache Jakarta Tomcat 4 and 5: Configuration and Usage Tutorial');

insert into TomcatResources values (
'http://www.jguru.com/faq/Tomcat',
'Java Guru: Tomcat FAQ Home Page');
```

Um das Skript auszuführen, müssen Sie es lediglich unter einem Namen wie *unleashed.sql* speichern und folgenden Befehl abschicken:

```
mysql> source /var/data/unleashed.sql
```

Mit diesem Befehl wird Ihre Tabelle erstellt und bestückt. Über den Parameter `source` wird die Datei mit absolutem Pfad angegeben. Den Pfad müssen Sie gegebenenfalls an Ihre Verhältnisse anpassen. Wenn Sie auf Nummer Sicher gehen wollen, wechseln Sie zurück in das Kommandozeilentool, laden Sie die Datenbank und fragen Sie mit `select` die Daten aus der Tabelle ab:

```
use unleashed;
select * from TomcatResources;
```

Da Datenbank und Tabelle fertig sind, können wir sie für Tomcat erschließen.

3.5 Datenbankverbindungspools erstellen

Um Tomcat Ihrer Datenbank vorzustellen, erzeugen wir einen Datenbankverbindungspool, wobei es sich letzten Endes um eine JNDI-Ressource handelt. JNDI ist eine J2EE-Technologie, auf die ich in Kapitel 21, JNDI-Ressourcen verwalten, noch ausführlicher zu sprechen kommen werde. JNDI ist im Wesentlichen eine verzeichnisähnliche Auflistung von Ressourcen, die von einem Provider gepflegt wird und auf die Clients zugreifen können. In unserem Fall übernimmt Tomcat die Funktion des Providers und unsere Anwendung (in diesem Fall die JSP-Datei) ist der Client. JNDI vergibt Ressourcennamen, über die der Client sich Handles auf die Ressourcen beschaffen kann.

Wenn Sie unter Tomcat einen JNDI-zugänglichen Datenbankverbindungspool erstellen, müssen Sie ihn in *$CATALINA_HOME/conf/server.xml* definieren. Wie wir in Kapitel 21 sehen werden, gibt es verschiedene Ebenen, auf denen Sie Ihren Pool definieren können. Sie können ihn auf der obersten Ebene definieren und damit allen Hosts und Hostanwendungen zur Verfügung stellen. Sie können ihn aber auch auf der virtuellen Host-Ebene definieren oder auf Anwendungs- bzw. Kontextebene. Entscheidend für die Wahl der Ebene ist einzig und allein die Frage der Reichweite, sprich: Wer benötigt den Pool? Da wir gerade erst dabei sind, uns in die Thematik einzuarbeiten, definieren wir den Pool einfach auf der Anwendungsebene. Das bedeutet, dass wir die Anwendungsdefinitionsdatei *$CATALINA_HOME/conf/Catalina/localhost/unleashed.xml* bearbeiten müssen. Listing 3.4 zeigt die geänderte Datei.

Listing 3.4: Geänderte unleashed.xml

```
<Context path="/unleashed" docBase="unleashed" debug="0">

  <Logger className="org.apache.catalina.logger.FileLogger"
   prefix="localhost_unleashed_" suffix=".log"
```

```
    timestamp="false"/>

  <Resource name="jdbc/unleashed" auth="Container"
    type="javax.sql.DataSource"/>
  <ResourceParams name="jdbc/unleashed">
    <parameter><name>username</name><value>root</value></parameter>
    <parameter><name>password</name><value></value></parameter>
    <parameter><name>driverClassName</name>
     <value>org.gjt.mm.mysql.Driver</value></parameter>
    <parameter><name>url</name>
     <value>jdbc:mysql://localhost/unleashed</value></parameter>
  </ResourceParams>

</Context>
```

Dieser Code ist ein typisches Beispiel für eine JNDI-Ressourcendefinition. Zunächst müssen Sie die Ressource selbst definieren. Die Tatsache, dass das `type`-Attribut auf `javax.sql.DataSource` gesetzt ist, teilt Tomcat mit, dass es sich um einen Datenbankverbindungspool handelt. Die Wahl des Namens ist dabei völlig frei gestellt. Es ist üblich für Verbindungspools das Präfix `jdbc/` zu verwenden. Und wenn Sie an das Präfix den Namen der Datenbank anhängen, kann man auf einen Blick erkennen, auf welche Datenbank die jeweilige Ressource verweist.

Auf die Ressourcendefinition folgen mehrere Parameter, die sich auf die Ressource beziehen. Es versteht sich von selbst, dass es für jeden Ressourcentyp eigene Parameter gibt; einige davon sind obligatorisch, andere nicht. Für Datenbankpools sind mindestens die folgenden vier erforderlich: Datenbankbenutzer, Datenbankpasswort, Treiberklassenname und der JDBC-URL, der auf die Datenbank selbst verweist. Tomcat verwendet diese vier Parameter, um Verbindungen zu Ihrer Datenbank herzustellen. Der Treibername ist wichtig, weil Tomcat in seinem Klassenpfad nach diesem Objekt sucht. Wenn Sie die jar-Datei mit den JDBC-MySQL-Treibern in *$CATALINA_HOME/common/lib* ablegen, können Sie sicher sein, dass Tomcat sie findet. Probleme, die bei der Erstellung von Verbindungspools auftreten, liegen meist darin begründet, dass es für die betreffende Datenbank keine jar-Datei mit den JDBC-Treiber gibt oder dass diese nicht am richtigen Ort (das heißt nicht in Tomcats Klassenpfad) liegt. Manchmal stimmt auch einfach der Treibername nicht. Den korrekten Treiberklassennamen können Sie in der Regel der JDBC-Dokumentation zu Ihrem Datenbankserver entnehmen, ebenso wie das Format für den URL – eine weitere häufige Fehlerquelle. Wenn Sie sich Ihren URL anschauen, so werden Sie feststellen, dass die Syntax im Wesentlichen aus Host und Datenbank besteht. Sie können hinter dem Host auch eine Portnummer anhängen, sofern Ihr MySQL-Server auf einem anderen Port als dem voreingestellten Port 3306 läuft.

Denken Sie auch daran, die Werte für `username` und `password` anzupassen, wenn Sie die MySQL-Anmeldung geändert haben. Wie Sie vielleicht bemerkt haben, ist mein `password`-Parameter leer. Dies liegt daran, dass der `root`-Benutzer standardmäßig kein Passwort benötigt. Es gibt noch weitere Parameter für Datenbankpools, die deren

Funktion beeinflussen – minimale und maximale Anzahl Verbindungen, Regeln für verwaiste Verbindungen usw. Mehr dazu in Kapitel 21.

3.5.1 JSP anlegen

Nur noch zwei Schritte und wir werden sehen, ob alles funktioniert. Zuerst müssen wir eine passende JSP-Datei erstellen (siehe Listing 3.5). Zweck dieser Seite ist, ein JNDI-Lookup für den Verbindungspool zu ermöglichen und eine JDBC-Verbindung herzustellen, über die wir die Tabellenzeilen einlesen können. Sollten Sie mit JDBC nicht vertraut sein, arbeiten Sie einfach mit Ausschneiden und Einfügen. Nennen Sie die erzeugte Seite *resources.jsp* und speichern Sie sie in das Verzeichnis *unleashed* unter Ihrem Deployment-Verzeichnis.

Listing 3.5: resources.jsp

```
<html>
<%@ page language="java"
    import="javax.sql.*,javax.naming.*,java.sql.*" session="false"%>

 <head>
  <style type="text/css">
   <!--
    a { text-decoration: none }
    body { font-family: verdana, helvetica, sans serif;
           font-size: 10pt; }
   -->
  </style>
 </head>

 <body>
  <center>
   <h3>Seite zum Testen des von Tomcat verwalteten Verbindungspools</h3>
  </center>

<%
    try {
        String jdbcname = "jdbc/unleashed";
        String cmd =
          "select Titel, URL from TomcatResources order by Titel";
        Context ctx = new InitialContext();
        Context envCtx = (Context)ctx.lookup("java:comp/env");
        DataSource ds = (DataSource)envCtx.lookup(jdbcname);
        Connection conn = ds.getConnection();
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(cmd);
        ResultSetMetaData rsmd = rs.getMetaData();
        int cols = rsmd.getColumnCount();
```

```
        out.println("<table border\"1\">");
        out.println("<tr bgcolor=\"lightGrey\">");

        for (int i = 1; i <= cols; i++) {
            out.println("<td align=\"center\" width=\"200\"><b>" +
                rsmd.getColumnName(i) + "</b></td>");
        }

        out.println("</tr>");

        while (rs.next()) {
            out.println("<tr>");

            for (int i = 1; i <= cols; i++) {
                out.println("<td align=\"left\">" +
                        rs.getString(i) + "</td>");
            }

            out.println("</tr>");
        }

        out.println("</table>");
        conn.close();
    } catch(Exception e) {
        out.println("Fehler: " + e);
        e.printStackTrace();
    }
%>

 </body>

</html>
```

Ein Punkt, der diese JSP-Datei von der in Kapitel 1.6 erzeugten *index.jsp* unterscheidet, ist das neu hinzugekommene `import`-Attribut in der Page-Deklaration. Ich habe `javax.naming.*` für die JNDI-Klassen, `javax.sql.*` für die JNDI-Datenquellenklassen und `java.sql.*` für die JDBC-Klassen importiert. Außerdem ist anzumerken, dass ich – statt HTML- und Java-Code zu vermischen – nur einen einzigen Code-Block erzeugt und mit der `println`-Methode des `out`-Objekts das benötigte HTML ausgegeben habe. Das `out`-Objekt ist standardmäßig für alle JSP-Dateien verfügbar. In manchen Fällen ist dies einfacher, als rohes HTML mit Java-Code zu vermischen. Allerdings müssen Sie unbedingt Doppelnennungen vermeiden, was sehr mühselig sein kann.

Die dieser Seite zugrunde liegende Logik ist, einen Handle – `InitialContext` genannt – für die JNDI-Namensumgebung zu beschaffen. Wie Sie sehen, führen wir effektiv zwei Suchen durch: eine, um den allgemeinen JNDI-Kontext, `java:comp/env`, und eine weitere, um die konkrete Ressource zu erhalten, die wir zuvor `jdbc/unleashed` genannt haben. Denken Sie daran, dass ich gesagt habe, dass ein JNDI-Lookup einen Handle

auf eine Ressource zurückliefert. In diesem Fall ist der Handle ein `javax.sql.DataSource`-Objekt, mit dessen Hilfe wir auf die in dem Objekt gekapselte Verbindung zugreifen können. Anschließend sind wir in der Lage, alle normalen JDBC-Anfragen auszuführen. Wie Sie sicherlich bemerkt haben, habe ich mich bemüht, generischen Code zu schreiben, der die auszugebenden Spaltennamen und -überschriften der zurückgelieferten Ergebnismenge entnimmt. Auf diese Weise wird die Tabelle dynamisch, entsprechend dem, was zurückgeliefert wird, erzeugt. Wenn Sie möchten, überzeugen Sie sich selbst davon und ersetzen Sie meine SQL-Anweisung durch eine Abfrage, die zu einer Ihrer eigenen Tabellen passt. (In diesem Fall müssen Sie unter Umständen auch die Ressourcendefinition ändern, um auf die neue Datenbank zu verweisen.)

3.5.2 Test

Falls Sie es noch nicht getan haben sollten, müssen Sie Tomcat jetzt starten bzw. neu starten. Anschließend können Sie *http://localhost:8080/unleashed/resources.jsp* aufrufen und die Ergebnisse der Datenbankabfrage betrachten, siehe Abbildung 3.4.

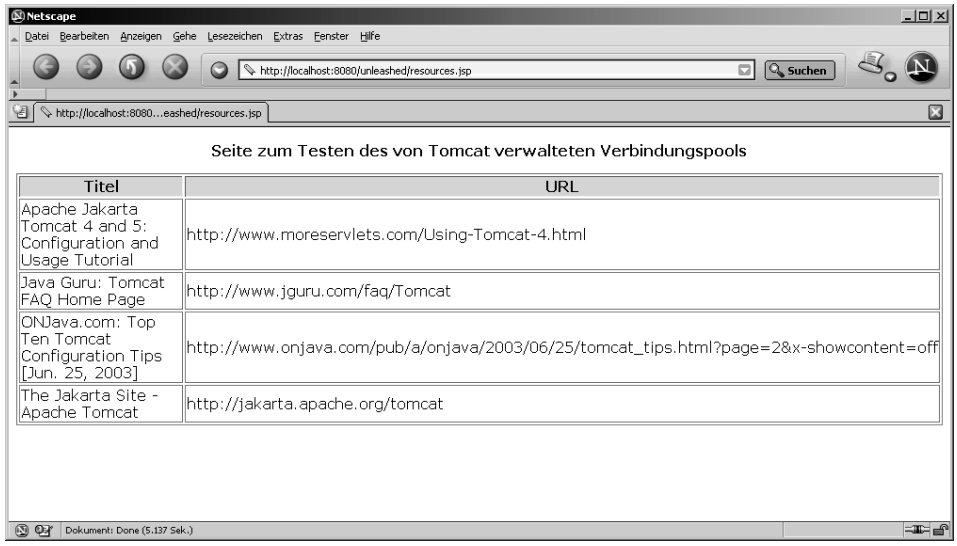


Abbildung 3.4: Test von resources.jsp

Wenn Sie die Seite nicht angezeigt bekommen, gibt es verschiedene Punkte, die Sie prüfen sollten. Zuerst sollten Sie sich vergewissern, dass *resources.jsp* im richtigen Verzeichnis liegt. Zweitens sollten Sie darauf achten, ob Sie eine Fehlermeldung, wie in Abbildung 3.5 angezeigt bekommen.

Diese Art Fehler tritt beim Zugriff auf Tomcat-Verbindungspools sehr häufig auf: Der Treiber, den Sie in Ihrer Ressourcendefinition in *server.xml* angegeben haben, kann

nicht gefunden werden. Ich schreibe meine JSP-Dateien grundsätzlich so, dass alle Ausnahmen an den Browser weitergeleitet werden. Doch nicht jeder Code ist so wartungsfreundlich implementiert. In vielen Fällen werden Sie in den Log-Dateien oder im Tomcat-Konsolenfenster nach Hinweisen suchen müssen. Meist werden Sie dabei einen umfangreichen Stack-Trace durchforsten müssen, in dem irgendwo die gesuchte Fehlermeldung versteckt ist. Wenn Sie sie gefunden haben, wissen Sie, dass sich die jar-Datei mit den JDBC-Treibern für Ihre Datenbank am falschen Ort befindet oder ganz fehlt. Beheben Sie das Problem, starten Sie Tomcat neu, und alles wird in bester Ordnung sein.

Abbildung 3.5: Fehlermeldung von resources.jsp

3.6 Tomcat 4 ausführen

Das meiste von dem, was ich über das Starten und Beenden von Tomcat 5 gesagt habe, gilt auch für Tomcat 4. Der wichtigste Unterschied ist, dass es bei Tomcat 4 keine Launcher- oder JSVC-Daemon-Optionen gibt.

3.6.1 Die bin-Dateien von Tomcat 4

Abbildung 3.6 zeigt den Inhalt des *bin*-Verzeichnisses von Tomcat 4.

Wie Sie sehen, gibt es keine großen Unterschiede zu Tomcat 5. Es gibt jedoch keine Launcher- oder jar-Dateien mit JSVC-Treibern. Dafür enthält die Tomcat 4-Distribution Skripte, mit denen der JSP-Compiler (Jasper) manuell ausgeführt werden kann.

3.6.2 Optionen für die Tomcat 4-Ausführung

Auch hier gilt das meiste von dem, was ich bereits zu Tomcat 5 erläutert habe. Unter Unix können Sie Ihre *rc*-Datei genau in der beschriebenen Weise erstellen (natürlich müssen die Pfade geändert werden). Was die Installation von Tomcat 4 als Dienst unter Windows betrifft, so ist die Syntax unterschiedlich, da Tomcat 4 `procrun` aus dem »Commons Daemon«-Projekt nicht verwendet. Wenn Sie den Windows-Installer verwenden, so können Sie mit diesem Tomcat als Dienst installieren – und das war's. Wenn Sie Tomcat manuell installieren wollen, müssen Sie das in Listing 3.6 aufgeführte Skript verwenden.

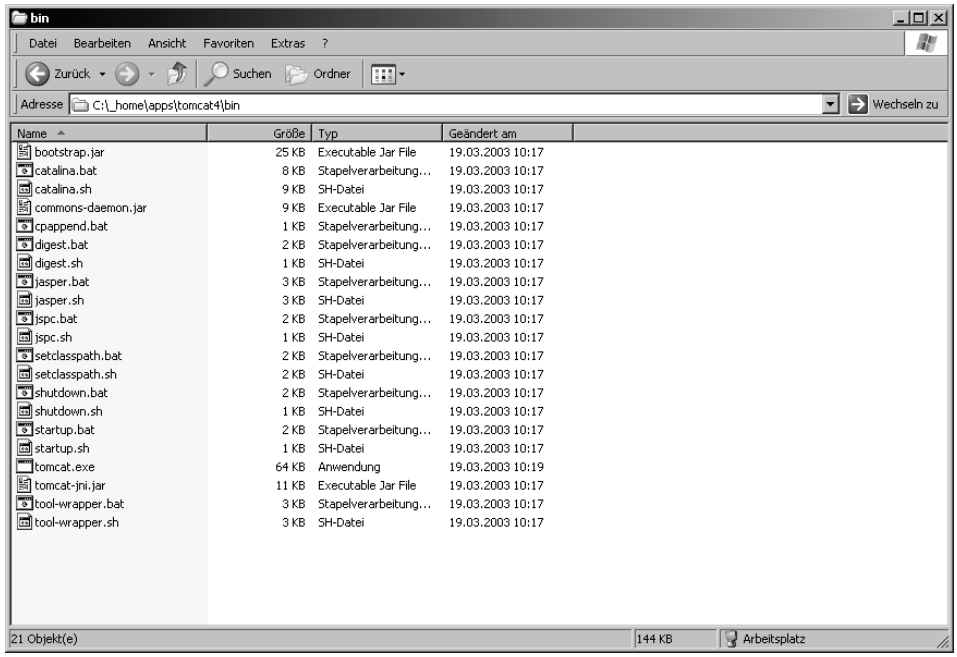


Abbildung 3.6: Inhalt des bin-Verzeichnisses von Tomcat 4

Listing 3.6: Skript für die Installation von Tomcat 4 als Dienst

```
"%CATALINA_HOME%\bin\tomcat.exe"
-install "Tomcat4"
"%JAVA_HOME%\jre\bin\client\jvm.dll"
-Djava.class.path="%CATALINA_HOME%\bin\bootstrap.jar; ↩
 %JAVA_HOME%\lib\tools.jar"
-Dcatalina.home="%CATALINA_HOME%"
-Xrs
-start org.apache.catalina.startup.Bootstrap -params start
-stop org.apache.catalina.startup.Bootstrap -params stop
-out "%CATALINA_HOME%\logs\stderr.log"
```

Achtung

Für das JDK 1.3.x lautet der Pfad *jvm.dll*: *%JAVA_HOME%\jre\bin\classic\jvm.dll*.

Wenn Sie diesen Befehl eingeben, erhalten Sie die Meldung:

```
Der Dienst wurde erfolgreich installiert.
```

Zur Überprüfung können Sie unter SYSTEMSTEUERUNG/VERWALTUNG/DIENSTE gehen and nachsehen, ob der Dienst, wie im Skript definiert, als `Tomcat4` gelistet ist.

Im DIENSTE-Dialogfenster können Sie einstellen, dass Tomcat beim Booten des Rechners automatisch gestartet wird. Darüber hinaus können Sie Tomcat über dieses Fenster auch manuell starten und beenden. Sollten Sie bevorzugen, den Dienst über das Kommandozeilentool auszuführen, können Sie Tomcat starten, indem Sie eingeben:

```
net start Tomcat4
```

Sie beenden Tomcat mit:

```
net stop Tomcat4
```

Und wenn Sie den Dienst entfernen wollen, geben Sie einfach ein:

```
%CATALINA_HOME%\bin\tomcat -uninstall Tomcat4
```

Je nachdem wie Sie den Dienst installiert haben, kann sein Name von dem meines Dienstes abweichen. Wenn ich den Dienst manuell installiere, nenne ich ihn in der Regel `Tomcat4`. Der Windows-Installer nennt ihn `Apache Tomcat`.

3.7 Tomcat 4-Ressourcen

Es wird Sie freuen, dass das kleine Verbindungspool-Beispiel (unter Verwendung von MySQL und JNDI) in gleicher Weise unter Tomcat 4 ausgeführt werden kann. Sollten Sie MySQL bislang nicht installiert haben, so tun Sie es nun und entpacken Sie die JDBC-Distribution. Kopieren Sie die jar-Datei mit den JDBC-Treibern für MySQL nach *$CATALINA_HOME/common/lib*.

Unter der Voraussetzung, dass Sie unter Tomcat 4 eine Kontextdeskriptordatei für die Unleashed-Anwendung erstellt haben, können Sie diese nun in derselben Weise bear-

beiten wie die Datei *unleashed.xml* von Tomcat 5. Vergessen Sie nicht die beiden Blöcke `<Resource>` und `<ResourceParams>` in die Datei einzufügen.

Zu guter Letzt setzen Sie die JSP-Datei auf – sofern Sie dies nicht bereits getan haben. Starten Sie Tomcat 4 und richten Sie Ihren Browser auf *http://localhost:8080/unleashed/resources.jsp*. Auch hier gilt: Sollten Sie eine Fehlermeldung erhalten, prüfen Sie Position und Namen Ihrer Dateien.

3.8 Abschluss und Ausblick

Blicken wir zurück auf das, was wir geleistet haben. Wir haben unsere produktionalisierte Tomcat-Installation um Autostart-Funktionen erweitert, sodass Tomcat beim Booten des Rechners automatisch gestartet wird. Wir haben definiert, was Webanwendungsressourcen sind und welche Aufgaben bei der Verwaltung von Ressourcen anfallen. Wir haben eine Datenbank installiert und konfiguriert, passende JNDI-Datenquellen eingerichtet und eine Anwendung (bestehend aus einer einzelnen JSP-Datei) geschrieben, die auf die Datenbank zugreift. Wenn Sie in einem größeren IT-Unternehmen tätig sind, gibt es vermutlich eine oder mehrere Personen, die für die Datenbank zuständig sind, sodass Ihre Aufgaben als Tomcat-Administrator dort etwas weniger umfangreich sein werden.

Im nächsten Kapitel komme ich zu einem sehr wichtigen Punkt, und zwar zur Frage der Sicherheit. Wir werden Ihrer Tomcat-Installation einige grundlegende Authentifizierungs- und Schutzmechanismen hinzufügen. Danach kommen wir zum Thema Anwendungsentwicklung, in dessen Rahmen ich verschiedene Methoden und Techniken für die Erstellung von Tomcat-Anwendungen beschreiben werde.