Tomcat Profi Know-how: Jasper 2

Tom C@

Das Tomcat-Projekt der Apache Jakarta Community ist sowohl Referenzimplementierung der Java Servlet- und JSP-Spezifikation als auch einer der erfolgreichsten Web-Container am Markt. Die stark wachsende Fan-Gemeinde nutzt Tomcat nicht nur für den Zweck der Ausbildung und Entwicklung, sondern vermehrt auch im produktiven Umfeld. Dieser Spagat zwischen Theorie und Praxis und die damit einhergehende Problematik war Grund genug, diese Kolumne ins Leben zu rufen.

Der Tomcat Web-Container hat sich in den letzten Jahren von einer reinen Referenzimplementierung hin zu einer durchaus etablierten Alternative zu kommerziellen Web-Containern entwickelt. Ein wichtiger Bereich, der allerdings seit jeher in diesem Projekt zu kurz kommt, ist die Dokumentation der verschiedenen Features und Konfigurationsmöglichkeiten. Dies kann sicher jeder bestätigen, der sich schon einmal außerhalb der Standardkonfigurationen bewegt hat.

Im Rahmen dieser Kolumne möchten wir aufzeigen, dass Tomcat deutlich mehr zu bieten hat als die Funktionalitäten, welche die meisten seiner User nutzen – nämlich die reine Umsetzung der Java Servlet- und JSP-Spezifikation. Im Sinne eines „Missing Manuals“ wollen wir im Laufe der Zeit all die Themen aufgreifen, die innerhalb der Tomcat-Gemeinschaft immer wieder Fragen aufwerfen und zu regen Diskussionen führen.

So – nun aber genug der Theorie ... wir wünschen viel Spaß bei beim ersten *Tom C@* Artikel...

Thema des Monats: Jasper 2 im Tomcat 4.1.24

Der Bereich Unterstützung der JSP-Entwicklung ist in der aktuellen Projektdokumentation spärlich beleuchtet. So ist die Fortentwicklung der neuen Architektur und Implementierung Jasper 2 fast unbemerkt realisiert worden. Die Interna der Jasper 2-Engine sind weitgehend undokumentiert und die Entwicklungswerkzeuge erst mit der jüngsten Version 4.1.24 erstmals ausreichend und korrekt dokumentiert. Ein detaillierter Blick lohnt sich, um die Parameter des JSPServlets in Produktionsumgebungen kennen zu lernen.

Prüfen der JSP-Syntax

Mit dem Tomcat 3 wurde begonnen einen JSP-Compiler auszuliefern, der in der Lage ist, die syntaktische Korrektheit einer JSP vor dem Einsatz im Web-Container zu prüfen. Die Verpackung, Parametrisierung und Qualität ist in den verschiedenen Releases unterschiedlich. Mit der Entwicklung des Jasper 2 ist eine Vereinheitlichung des Aufrufs und der Funktionsweise umgesetzt worden. Im aktuellen Tomcat Release 4.1.24 ist neben der weiterhin verfügbaren Java *main()*-Schnittstelle eine Ant-Task verfügbar (Listing 1) [1]. JSPs können nur sinnvoll in Zusammenhang mit der kompletten Web-Anwendungen mit dem Deskriptor *web.xml*, TagLibs, Klassen und JAR-Archiven geprüft werden. Deshalb nimmt die Jasper 2-Task mit dem Parameter *uriroot* Bezug auf die Anwendung, die als Verzeichnis-Deployment vorliegt. Gleichzeitig kann auf Wunsch direkt eine Übersetzung und eine XML-Validierung der Deskriptoren erfolgen. In der Praxis werden diese Aufgaben mit separaten Ant-Tasks meist besser gelöst. Als Besonderheit ist der Jasper Compiler in der Lage, ein *web.xml*-Fragment zu generieren, das die übersetzten Servlets im *web.xml* definiert und die entsprechenden URI-Mappings enthält. Die vollständigen Parameter und die Funktionsweise sind in Tabelle 1 dokumentiert. Mit diesen Möglichkeiten ist die Grundlage geschaffen, eine Web-Anwendung zu erstellen, welche zur Laufzeit ohne Übersetzung der JSP auskommt und damit ohne Java Compiler auf dem Server. Die Jasper 2-Task funktioniert nur vollständig, wenn die Quell- und Zielverzeichnisse vorher vorhanden sind.

JSPServlet-Konfiguration

Zur Laufzeit einer Web-Anwendung kommt dem JSPServlet die Aufgabe zu, dass bei der Anforderung einer JSP aus der JSP-Quelldatei ein ausführbares Servlet erzeugt wird. Die Anforderungen eines Produktions-Servers oder Entwickler-Tomcats sind verschieden: Während der Entwicklung der JSPs ist es sinnvoll, möglichst mit wenigen Server- oder Anwendungsneustarts die Änderung durchführen zu können. In dieser Phase der Entwicklung sollte ein Verzeichnis-Deployment hergestellt werden.

Die Konfiguration des Tomcat-Releases ist nur für die Entwicklungsphase vorbereitet. Jeder Zugriff auf eine JSP bewirkt einen Test, ob die JSP-Quelle schon geladen ist oder eine Aktualisierung erfolgen muss. Als Compiler wird der JDK-Javac via exter-

ner Ant-Task-Aufruf genutzt, d.h. für jede Übersetzung wird eine JVM geforkt. Wenn eine solche Standard-Konfiguration bei Ihnen auf den Produktions-Servern ist und Sie gleich mehrere JSP mit der Startseite anziehen, ist die fehlende Geschwindigkeit beim ersten Nutzerzugriff allzu erklärbar. Der externe Aufruf ist leider immer noch notwendig, da der Javac leider seit Jahren verschiedene bekannte Speicherlöcher enthält. Aus diesem Grund ist es *nur* in der Entwicklung ratsam, den Parameter *fork=false* zu setzen und damit den aufwändigen, externen Prozessaufruf für die Übersetzung zu sparen. Zusätzlich bewirkt der *fork*-Parameter, dass immer nur eine JSP-Übersetzung gleichzeitig stattfindet.

Weitere Optimierungen sind für einen ungebremsten JSP-Zugriff auf einem Produktions-Server denkbar. Durch den Parameter *development=false* werden die Prü-

Listing 1: Jasper2 Ant-Task zum Syntax Test aller JSP einer Web-Anwendung

```
<target name="jasper2" depends="prepare">
  <taskdef classname="org.apache.jasper.JspC"
                                        name="jasper2">
    <classpath id="jspc.classpath">
      <pathelement location=
                        "${java.home}/../lib/tools.jar"/>
      <fileset dir="${catalina.home}/server/lib">
       <include name="*.jar"/>
      </fileset>
      <pathelement location="${catalina.home}/
                                        common/classes"/>
      <fileset dir="${catalina.home}/common/lib">
        <include name="*.jar"/>
      </fileset>
    </classpath>
  </taskdef>
  <jasper2
    compile="false"
    validateXml="false"
    uriroot="${ROOT.base}"
    webXmlFragment="jspc/WEB-INF/generateweb.xml"
    outputDir="jspc/WEB-INF/src"/>
</target>
```

Listing 2: JSPServlet-Konfiguration in der Datei *conf/web.xml* für ungebremsten JSP-Zugriff

```
<servlet>
  <servlet-name>jsp</servlet-name>
  <servlet-class>org.apache.jasper.servlet.JspServlet
                                        </servlet-class>
  <init-param>
    <param-name>development</param-name>
    <param-value>false</param-value>
  </init-param>
  <init-param>
    <param-name>reloading</param-name>
    <param-value>false</param-value>
  </init-param>
  <init-param>
    <param-name>keepgenerated</param-name>
    <param-value>false</param-value>
  </init-param>
  <init-param>
    <param-name>classdebuginfo</param-name>
    <param-value>false</param-value>
  </init-param>
  <init-param>
    <param-name>fork</param-name>
    <param-value>true</param-value>
  </init-param>
  <init-param>
    <param-name>logVerbosityLevel</param-name>
    <param-value>ERROR</param-value>
  </init-param>
  <load-on-startup>3</load-on-startup>
</servlet>
```

Parameter	Wert	Erklärung
compile	false/true	Übersetzung nach der Generierung
validateXml	false/true	Prüfen der DTDs von web.xml und der tlds
uriroot	WAR-Verzeichnis ${ROOT.base}	Verzeichnis der deploybaren Anwendung
WebXmlFragment	${ROOT.base}/ WEB-INF/generated_web.xml	web.xml-Fragment mit JSP Servlet-Definition und Mappings
outputDir	${ROOT.base}/WEB-INF/src	Verzeichnis für die generierten Servlet Java-Quellen der JSPs
classPath	Meist leer und damit der java CLASSPATH	Systemklassenpfad für den integrierten Java-Compiler
listErrors	false/true	Anzeige der Fehler unterdrücken
package	org.apache.jsp	Hier wird der Basis Package-Pfad der generierten JSP bestimmt.
classDebugInfo	false/true	JSPC Generierung/Kompilierung mit Debug-Information oder mit Optimierung

Tab. 1: Definition der Jasper2 Ant Parameter

fungen bei jedem Zugriff unterdrückt. Mit *reloading=true* und *checkIntervall= <Sekunden>* lässt sich eine Hintergrundprüfung von Änderungen unabhängig von Anfragen aktivieren. Durch Setzen des Flags *reloading=false* wird die JSP nur noch bei der ersten Anfrage übersetzt. Schließlich bewirkt der Parameter *classdebuginfo=false,* dass ohne Debug-Informationen und mit Code-Optimierung übersetzt wird. Theoretisch kann der Compiler des Jasper 2 Ant-Projekts gewechselt und auf den schnelleren Jikes Compiler [2] umgestellt werden. Ein Test des Jikes 1.18 schlug allerdings fehl, da die Windows-Version ohne encoding-Unterstützung geliefert wird, der Jasper 2 diese Option aber einfordert. Die Debugging-Möglichkeit ist für JSP-Dateien mit dem Jasper 2 nur eingeschränkt gegeben, da der Package-Pfad in den Sourcen und im Arbeitsverzeichnis unterschiedlich ist. Diese Tatsache verhindert meist, dass der Debugger auf die generierten Quellen einen Breakpoint setzen konnte. Die erforderliche Verzeichnishierarchie des Prefix *org.apache.jsp* wird erst in der aktuellen CVS-Version des Tomcat 5 oder durch das Sysdeo-Patch angelegt [3], [4].

Profi Know-how: Nutzung von JSPs ohne Laufzeitübersetzung

Eine weitere Überlegung für die Beschleunigung einer produktiven Web-Anwendung mit JSP ist: Warum nutzen wir die übersetzten Servlets des Jasper 2 Ant-Task nicht zur Laufzeit und sparen uns die Laufzeitübersetzung gleich ganz?

Mit einem kleinen Ant-Skript haben wir den Test am Beispiel der Demo-Anwendung der Tomcat-Distribution gewagt. Das Ant-Skript generiert mit Hilfe der Japser 2 Ant-Task die Servlet-Klassen und das erforderliche Mapping für den Anwendungs-Deskriptor *WEB-INF/generaqted_web.xml*. Aus diesem *web.xml*-Fragment wird dann ein vollständiger Web-Diskriptor erzeugt und mit Hilfe der Cactus 1.5 Ant-Task *webxmlmerge* mit originaler *web.xml* der Anwendung verschmolzen [5]. Die generierten JSP-Klassen werden in das Verzeichnis *WEB-INF/classes* übersetzt. Die Ressourcen in Form von statischen HTML-Seiten, Bildern, usw. und die Java-Klassen, JAR-Archive und Taglib-Deskriptoren der ursprünglichen Anwendung werden exklusiv von den JSPs übernommen. Eine derart zusammengestellte Web-Anwendung kommt ohne lästige Wartezeiten für die Kontrolle und Zwischenverwaltung des JSPServlet aus. Wir vermuteten, dass dieses Verfahren gerade bei schwächeren Test- und Produktionsmaschinen Erfolge zeigt. Ihre Erfahrungen und Praxisberichte möchten wir im Tomcat Forum diskutieren [6]. Ein einfaches Beispiele des Deployment-Verfahrens „WithoutJSP“ finden Sie auf der Heft-CD.

Wieso eine eigene Tomcat-Kolumne?

Wie mittlerweile wahrscheinlich jeder weiß, ist Tomcat *ein* wenn nicht sogar *das* tragende Projekt der seit 1999 aktiven Apache Jakarta Community. Im Laufe der letzten vier Jahre hat das Projekt einen enormen Zuwachs an Fans verzeichnen können. Mittlerweile gilt Tomcat als etabliert. Dies gilt nicht nur für die Bereiche Ausbildung und Entwicklung, sondern vor allem auch im produktiven Umfeld.

Eine Tatsache, die allerdings häufig außer Acht gelassen wird ist, dass Tomcat die offizielle Referenzimplementierung der jeweils aktuellen Java Servlet- und JSP-Spezifikation darstellt. Was zunächst nicht weiter kritisch zu sein scheint, stellt sich gerade im produktiven Umfeld häufig als ein echtes Problem dar. Jede neue Spezifikation bringt per Definition auch automatisch eine – zum Teil drastische – Änderung bzw. Erweiterung des Web-Containers mit sich. Die dicht aufeinander folgenden Release-Zyklen innerhalb des Java Community Process (JPC) im Bereich der Spezifikationen verursachen einen enormen Druck auf die Tomcat Entwicklergemeinde. Es ist somit kein Wunder, dass die Dokumentation mehr als spärlich ist und hier und da unliebsame Fehler auftauchen, die nur durch geschickte Workarounds und Patches umgangen werden können.

Genau an dieser Stelle möchten wir mit unserer Kolumne ansetzen und die verschiedenen Entwicklungen innerhalb des Tomcat-Projektes für den „normal sterblichen“ Entwickler transparenter gestalten. Wir wollen dem Leser mit Hilfe der Kolumne ein Mittel an die Hand geben, um aus Tomcat deutlich mehr herauszuholen, als nur einen standardisierte Ablaufumgebung für Web-Applikationen.

Peter Roßbach (pr@webapp.de) ist freiberuflicher Systemarchitekt und Autor der Bücher „Java Server und Servlets“ und „Tomcat 4X“. Er interessiert und engagiert sich seit 1997 für Java-basierende Server-Lösungen in Magazin- und Konferenzbeiträge. Ein weiterer Schwerpunkt seiner Tätigkeiten liegt in der Qualitätssicherung von Prozessen und Verfahren zur Testbarkeit von J2EE-Anwendungen.

Lars Röwekamp (lars.roewekamp@openknowledge.de) ist Gründer des IT Beratungsunternehmens OpenKnowledge und beschäftigt sich dort mit der eingehenden Analyse und Bewertung neuer Software- und Technologietrends. Ein besonderer Schwerpunkt seiner Arbeit liegt auf dem Bereich Enterprise Computing, wobei neben Design- und Architekturfragen insbesondere Real-Life Aspekte im Fokus seiner Betrachtung stehen. Lars Röwekamp, Autor mehrerer Fachartikel und -bücher, beschäftigt sich seit der Geburtsstunde von Java mit dieser Programmiersprache, wobei er einen Großteil seiner praktischen Erfahrungen im Rahmen großer, internationaler Projekte sammeln konnte.

Wie geht's weiter?

Die beiden Rubriken *Thema des Monats* und *Profi Know-how* wird es auch in allen folgenden Kolumnen geben. Ansonsten möchten wir die Gestaltung der Kolumne den Wünschen der Leser anpassen. Als Medium für einen Austausch zwischen Autoren und Lesern soll dabei das Tomcat Forum [6] dienen. Um es also in aller Deutlichkeit zu sagen:

Sie bestimmen Inhalt und Qualität durch Ihr Engagement und Ihre Kritik. Nutzen sie diese Chance.

Links & Literatur

[1] jakarta.apache.org/tomcat/tomcat-4.1-doc/jasper-howto.html

[2] www.sysdeo.com/eclipse/tomcatPlugin.html

[3] aktuelles Release 1.18 oss.software.ibm.com/developerworks/opensource/jikes/

[4] jakarta.apache.org/tomcat/

[5] jakarta.apache.org/cactus/integration/ant/index.html

[6] www.javamagzin.de/tomcat/