Mittel und Wege gegen Katzenjammer mit der Tomcat server.xml-Konfiguration

Kater à la Carte

Die korrekte Konfiguration eines Tomcat Web-Containers ist nicht immer offensichtlich. Eine fehlerhafte Produktionskonfiguration entscheidet nicht selten über Erfolg oder Misserfolg eines kompletten Projekts. Da die Informationen aller Konfigurationsoptionen nicht immer aktuell und eindeutig in der Dokumentation zu finden sind, ist der Quellcode oder eine „Mailing-Liste“ meist der rettende letzte Anker. Unsere Referenzkarte für den Tomcat, die Sie in diesem Heft finden, soll einen weiteren Helfer schaffen.

In der Regel wird die Konfiguration der Datei *conf/server.xml* in einem Projekt nicht geändert, da die Voreinstellungen der Distribution für die Entwicklung vollkommen ausreicht. Die einzige Änderung, welche man doch häufiger antrifft, ist die Hinzunahme der SSL- oder AJP-Connectoren. Nicht wenige dieser Konfigurationen finden sich dann auch in der Produktionsumgebung wieder, was häufig ein Problem darstellt. Ein Grund für das Nicht-Ändern der Konfiguration ist sicherlich, dass sich die Bedeutung der Elemente und Parameter nicht immer auf den ersten Blick erschließt. Weiteren Einblick bezahlt man mit wiederholtem Suchen in der Dokumentation. Für diese Situation haben wir die TomC@-Referenzkarte geschaffen. In dieser Ausgabe stellen wir die grundsätzliche Serverkonfiguration zum besseren Verständnis im Zusammenhang kurz dar. Des Weiteren haben wir im Rahmen der „Gerüchteküche“ einige Neuigkeiten über das Tomcat Release 5 zusammengestellt.

Thema des Monats: server.xml-Konfiguration

Der Tomcat-Server, wie er allgemein hin nur bezeichnet wird, bietet mehr als auf den ersten Blick zu sehen ist. Kern des Tomcats ist die so genannte Catalina-Architektur, welche das Zusammenspiel der Container-Klassen definiert und eine Standard-Implementierung enthält. Die Konfiguration des Tomcat spielt sich hauptsächlich in genau dieser Architektur ab, allerdings gibt es auch Konfigurationen, welche nur zur Konfiguration der Web-Anwendung dienen oder einen Standard für diese vorgeben.

Der Web-Container Catalina des Tomcat basiert auf einer flexiblen Architektur und dient dazu, eine Ablaufumgebung für Java-Web-Anwendung mit dem Servlet-API 2.2 und 2.3 zur Verfügung zu stellen. Betrachtet man das Feld der „Server- und Anwendungskonfiguration“, so ist es zunächst wichtig, die strikte Trennung zwischen dem Servlet-Container (*conf/server.xml*) auf der einen und der Web-Anwendung (*<webapp>/WEB-INF/web.xml*) auf der anderen Seite zu verstehen.

Der Web-Container stellt die Ablaufumgebung für eine „beliebige“ Anzahl von Web-Anwendung zur Verfügung und steht somit in der Zuständigkeit des Betreibers. Eine Web-Anwendung dagegen repräsentiert Geschäftslogik und nutzt dazu u.a. verschiedene Ressourcen, welche ihr wiederum von ihrer Ablaufumgebung – sprich dem umgebenden Web-Container – zur Verfügung gestellt werden. Der für die Web-Anwendung zuständige Anwendungsentwickler kennt (im Gegensatz zum Betreiber) in der Regel die endgültige Ablaufumgebung nicht.

Die Catalina-Architektur ist vollständig durch Java-Interfaces definiert [1], [2]. Erst zur Laufzeit wird durch einen Bootstrap-Prozess ein Exemplar der Komponenten erzeugt und gestartet. Der Boot-

strapper-Loader lädt die Datei *${catalina.base}/conf/server.xml*, welche die Struktur für den Server vorgibt. Damit die Flexibilität der Architektur unterstützt wird, gibt es keine definierte Struktur in Form einer DTD oder eines XML Schema, sondern es wird eine dynamische XML-Form genutzt. Der Jakarta Commons Digester ist ein „XML to Java Mapper", der auf der Basis von Regeln die einzelnen XML-Elemente solcher dynamischen Formate bindet, also Java-Objekte erzeugt und Attribute mit Werten belegt [3]. Die Regeln können direkt aus der XML-Datei den Klassennamen der zu erzeugenden Klasse ableiten und die notwendigen vorhandenen Attribute binden. In einer Konfiguration können damit uneinheitlich für die gleichen XML-Elemente unterschiedlichen Java-Klassenbindungen erfolgen wie z.B. Valve, Realm oder Connector [4] (Abb. 1).

Eine Catalina-Instanz besteht aus einem Server, der aus einem oder mehreren Services, einer Definition globaler Ressourcen und beliebig vieler Listener besteht. Der Server definiert auch den Socket für das *Shutdown*-Kommando. Mit dem entsprechenden Befehl wird über diesen Service der Server kontrolliert heruntergefahren. Mittlerweile ist zusätzlich noch ein Signal-Handler in der VM registriert, der ebenfalls einen sauberen Abbruch veranlasst (ab JDK 1.3). Jeder Service besteht aus einem oder mehreren Connectoren, die wiederum verschiedene Protokolle behandeln können. Ein Connector ist dafür verantwortlich, einen Thread Pool bereitzustellen und für die Übersetzung des externen Protokolls zu sorgen. Er dient dazu, die Request- und Response-Datenstrukturen des Tomcats zu erstellen. Für die eigentliche Abarbeitung ist die Engine des Service zuständig. Momentan gibt es für den Tomcat 4 eine Reihe von verschiedenen Connectoren für HTTP1.1, HTTP1.0, SSL, AJP, WARP. Eigentlich sind bis auf die Coyote-Connectoren schon alle Entwicklungen veraltet und werden im Tomcat 5 nicht mehr vorhandenen sein. Wir haben uns deshalb in der Referenzkarte auf die Attribute der Coyote-Familie beschränkt. Ein kleiner Tipp am Rand: Die Belegung des Attributs *enableLookup* mit dem Wert *true* konfiguriert den Connector zur Auflösung der Client IP-Adresse durch den *HttpServletRequest.getRemoteHost()* basierend auf den Klasse *java.net.InetAddress*. Diese Standard JDK-Klasse stellt einen VM-globalen DNS-Cache für erfolgreiche und nicht erfolgreiche DNS-Lookups zur Verfügung. In der Voreinstellung wird eine einmalig erfolgreiche Anfrage allerdings für immer vorgehalten, nicht erfolgreiche werden nach zehn Sekunden erneut angefragt. Wer schon mal Probleme mit dieser Dynamik des Netzes hatte, sollte in den Startskripten besser die Parameter *-Dnetworkaddress.cache.ttl=360* und *-Dnetworkaddress.cache.negative.ttl=10* in die Variable *$CATALINA_OPTS* setzen. Damit sollte die Auswertung von wechselnden IP-Adressen von DynaDNS-Servern mit einem Tomcat und Web Services auch wieder zuverlässig gelingen.

Der Connector im Tomcat erstellt ein Request/Response-Pärchen und sorgt dafür, dass dieses in einem eigenen Thread innerhalb der Engine abgearbeitet werden. Die interne Datenstruktur tritt nun ihren Weg zur Bearbeitung durch das Servlet an. Dabei wird zuerst ausgewählt, welcher Host, Context (Web-Anwendung), Servlet-Filter und Servlet die Bearbeitung übernimmt. Jede Engine besteht somit aus einem oder mehreren virtuellen Hosts. Die Unterscheidung der Hosts wird anhand der im Request befindlichen Servernamen, bzw. dem URI (Context- und Servlet-Path) getroffen. Jeder Host kann durch eine Aliasliste unter verschiedenen Namen angesprochen werden. Die Web-Anwendungen haben jeweils eine eindeutige URI innerhalb ihres Hosts. Der Descriptor *WEB-INF/web.xml* der Anwendung legt fest, welches Servlet angesprochen wird und welche Servlet-Filter zwischen geschaltet werden [5].

Da die einzelnen Elemente Engine, Host und Context starke Ähnlichkeit in ihren Services und Funktionsweisen haben, erben sie alle von dem Interface-Container. Ein Container kann mehrere Listener, einen Logger, ein Authentifikationsreich (Realm) und mehrere Container-Filter (Valve) enthalten. Die Listener werden im Rahmen der Initialisierung und, falls Sie das Interface *LifecycleListener* implementieren, über das Starten und Stoppen der Einheit informiert. Der Logger ist für die Steuerung potenzieller Ausgaben zuständig. Die Ausgabenvielfalt wird mit dem Attribut *debug* der Konfigurationselemente gesteuert. Welche Art von Log-Nachrichten wirklich geschrieben wird, hängt von dem Logger-Attribute *verbosity* ab. Mit Hilfe der Realms kann eine Authentifikation erfolgen. Das Servlet API definiert dabei vier verschiedenen Autorisierungsarten: HTTP BASIC und -Digest, Form Based und clientseitige Zertifikate. Das Realm stellt eine Datenquelle (Datei (*conf/ tomcat_user. xml*), Datenbank oder LDAP-Service) zur Verfügung, gegen die Login/Password bzw. Zertifikat geprüft werden. Auf allen Ebenen von Engine, Host und auch Context kann ein eigenständiges Realm definiert werden. Somit kann man für den gesamten Server, einen einzelnen Host oder eine Anwendung die Autorisierungsdatenquelle festlegen. Die Unterschiedlichkeit der Realms hat uns erst mal davon abgehalten, die Attribute auf der Referenzkarte mit aufzunehmen. Wir werden das Thema Realms aber in einer separaten Folge der Kolumne aufarbeiten.

Listing 1: Beispiel-Konfiguration für Sichtbarkeit von Parametern

```
<Host name="localhost" appBase="webapps"
                                    unpackWARs="false">
 <DefaultContext>
  <Parameter name="param1" value="host-param1"
                                    override="false" />
  <Parameter name="param2" value="host-param2"
                                     override="true" />
 </DefaultContext>

 <Context path="/contextparam1" docBase=
                        "contextparam" override="false">
  <Parameter name="param1" value="context1-param1"
                                    override="true" />
  <Parameter name="param2" value="context1-param2"
                                    override="false" />
 </Context>

 <Context path="/contextparam2" docBase=
                        "contextparam" override="true">
  <Parameter name="param2" value="context2-
                                param2" override="true" />
  <Parameter name="param3" value="context2-param3"
                                    override="false" />
 </Context>
</Host>
```

Auf dem Weg zum Servlet können Container-Filter (Valves) eingesetzt werden. Diese Filter können z.B. den beliebtem Accesslog im Common Log File Format (CLF und ELF [6]) oder in eine Datenbank schreiben, bestimmten IP-Adressen oder Hosts den Zugriff erlauben/verbieten bzw. einen vollständigen Dump des Request/Repsonse protokollieren. Interne Valves sorgen für den Durchfluss von der Engine zu Host und Context oder einer geeigneten Reaktion auf fehlerhafte Anfragen. Die Valves sind den Servlet-Filtern ähnlich. Die fachliche Logik gehört deshalb immer noch in die Servlet-Filter, da sie dort vom Web-Container unabhängig realisiert werden kann.

Ziel der Konfiguration des Servers ist die Web-Anwendung. Eine Web-Anwendung wird durch den Descriptor *WEB-INF/web.xml* beschrieben. Diese Beschreibung definiert die Servlets mit Mapping, Sicherheitseinschränkungen, Filtern, Listenern, Parametern und JNDI-Umgebung. Jeder Context baut im Übrigen seinen eigenen JNDI-Context auf. Gleichnamige Ressourcen können also problemlos auf unterschiedliche Datenbanken zeigen. In jedem Context kann zusätzlich ein Sessionmanager und ein Loader angegeben werden. Der Loader ist für das Laden der Klassen verantwortlich. Ein erneutes Laden der Klassen einer Web-Anwendung kann mit dem Context-Parameter *reloadable=true* erfolgen. Alle Änderungen in *WEB-INF/classes* und *WEB-INF/lib* der Anwendung führen dann zum erneuten Laden des Kontextes. Der Sessionmanager hat die Aufgabe, die Session zur Verfügung zu stellen. Der *StandardManager* speichert die aktiven Sessions nur alle gemeinsam beim Stop und lädt sie beim Starten. Die Persistent-Manager sind hier geschickter und können einzelne Sessions in Dateien oder eine Datenbank gezielt auslagern. Eine Managererweiterung von Filip Hanik sorgt sogar dafür, dass eine Session-Replikation zwischen Tomcat-Cluster-Knoten stattfindet [7], [8].

Besonders betrachtenswert ist die Initialisierung des Contexts. Hier kommen verschiedene Konfigurationseinstellungen zusammen. Grundsätzlich erfolgt die

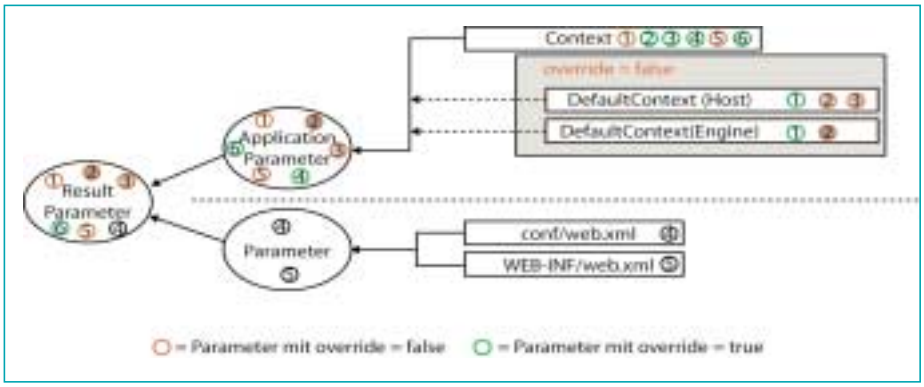

Abb. 1: Die Tomcat Server-Konfiguration

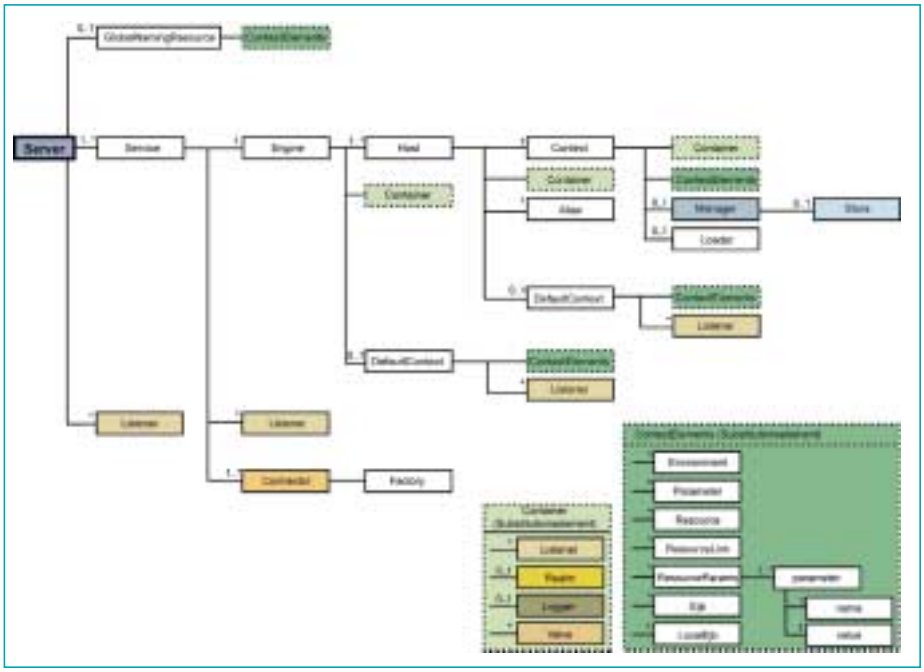

Abb. 2: Welchen Wert hat ein *<context-param>*?

Initialisierung einer Web-Anwendung in der Reihenfolge:

- Context in der *server.xml* oder *webapps/context.xml*,
- DefaultContext des Hosts falls *context.override=false*,
- DefaultContext der Engine falls *context.override=false*,
- Default web.xml in *conf/web.xml*,
- Anwendungsdescriptor *WEB-INF/web.xml*.

Die Konfiguration der einzelnen Komponenten (Resource, EJB, Parameter, Listener) überschreiben die Einstellungen des Vorgängers, außer bei folgenden beiden Elementen in der web.xml: Parameter *<context-param>* und Environment *<env-entry>*. Der Context hat die Semantik des Attributes *override=true*, was bedeutet, dass die DefaultContext nicht in diesem Context geladen werden. Für Parameter und Environment bedeutet *override=true*, dass sie nachfolgend überschrieben werden dürfen (Abb. 2).

Es ist also möglich, die Konfiguration durch den DefaultContext in einigen Anwendung des Hosts komplett zu unterdrücken und der Anwendung zu verbieten, bestimmte Parameter durch ihren *WEB-INF/web.xml*-Descriptor zu überschreiben (Listing 1). Im Context */contextparam1* sind *param1=host-param1* und *param2=context1-param2*. Der Con-

text */contextparam2* unterdrückt den DefaultContext und nur die *param2* und *param3* sind vorhanden. Im Gegensatz zum */contextparam1* kann der *param2* aber noch von der Web-Anwendung überschrieben werden (Beispiel befindet sich auf der Heft-CD).

Gerüchteküche Tomcat 5

In der Tom C@-Gerüchteküche geht es weniger um knallharte Fakten als vielmehr um Tendenzen und Mutmaßungen zu bestimmten Themen rund um Tomcat, welche sich hauptsächlich aus den einschlägigen User Groups und den jeweils aktuellen Sourcen herleiten lassen. Unser Ziel ist es vor allem, durch das eine oder andere Gerücht eine rege Diskussion im Tomcat-Forum [9] zu entfachen.

Ohne JMX geht nix ... Das ist wohl eine der Kernaussagen, die zu dem neuen Tomcat Release 5 getroffen werden kann, wobei noch nicht ganz klar ist, ob es sich eher um ein JMX-enabled oder -based System handeln wird. Generell sollen die Elemente des Tomcat via JMX – unter Berücksichtigung der JSR 77 (J2EE Management-API) – nicht nur zu managen, sondern auch zu monitoren sein. Tomcat selbst sieht sich dabei u.a. als voll verantwortlich für die Bereiche Web-Module und Servlet an und bietet daher derzeit keine Möglichkeit, die JMX-Fähigkeit abzuschalten. Dies wiederum führt aktuell zu kleinen Streitigkeiten mit der JBoss-Group, die ebenfalls die Oberhand über die Wunderwelt des Monitorings und Managements haben wollen. Aus Sicht von JBoss sollte der umgebende Application Server die gesamte Verwaltung übernehmen. Probleme gibt es vor allem noch von daher, dass die JMX-Unterstützung im Tomcat derzeit an einigen Stellen nur unzureichend umgesetzt ist. Ebenfalls Probleme bringt der aktuelle Hybridansatz mit sich, welcher neben JMX – aus Gründen der Abwärtskompatibilität – auch noch die alten API-Aufrufe und -Konfigurationsmechanismen unterstützt.

Mittels JMX soll Tomcat noch besser als Embedded Web-Container laufen können – und das mit einer gesteigerten Performance. Über diesen Weg lassen sich Applikations-Server wie die J2EE 1.4 Referenzimplementierung oder JBoss problemlos mit einem Web-Container oder aber eigene Anwendungen mit HTTP-Support ausstatten. In der Tomcat 5-Distribution befindet sich nun sogar ein Ant-Skript, das einen vollständigen Tomcat via JMX erzeugt [1].

Weniger ein Gerücht als vielmehr eine Tatsache ist die vollständige Unterstützung der JavaServer Pages 2.0 sowie der Java Servlet 2.4-Spezifikation, bzw. deren mit MANDATORY gekennzeichnete Pflichtanteil.

Tomcat via JMX monitoren

Bisherige Problemfelder, wie zum Beispiel das Protokoll des AJP-Connectors, welches für die direkte Kommunikation zwischen Tomcat und dem Apache Webserver verwendet werden kann, sollen in Zukunft mit einer deutlich höheren Stabilität aufwarten. Die Coyote-Familie wird erwachsen.

Ebenfalls einem gründlichen Refactoring wurde auch der Application Deployer unterzogen, welcher u.a. durch einen stand-alone Application Deployer erweitert wurde. Dieser ermöglicht die Validierung und Kompilierung einer Web-Applikation vor ihrem tatsächlichen Deployment.

Schneller soll der neue Tomcat natürlich auch werden. Dies ist sicher ein Tribut an die Tatsache, dass Tomcat heute deutlich über eine Referenzimplementierung hinausgeht und immer häufiger auch in produktiven Umfeldern zu finden ist.

Mit einem Final Release des neuen Tomcat kann so gegen Ende des dritten bzw. Anfang des vierten Quartals 2003 gerechnet werden. Das tatsächliche Datum hängt dabei weniger von der Tomcat Developer Group ab, als viel mehr von der Finalisierung der JSP 2.0- und der Servlet 2.4-Spezifikation.

Neben allen technischen Neuerungen und Verbesserungen soll auch der Bereich der Dokumentation deutlich aufgebessert werden ... aber dies ist wohl wirklich nur ein Gerücht ;-)

Hier noch ein kleiner Tipp am Rande. Möchte man sich von Zeit zu Zeit über die neuesten Entwicklungen rund um Tomcat informieren, lohnt sich immer ein Blick in die Mailing-Listen des Tomcats [10]. Durch Eingabe des Suchbegriffs „ANN“ erhält man stets die neuesten Announcements.

Ein manifestierendes Gerücht in eigener Sache ist, dass wir begonnen haben, ein Buch über den Tomcat5 zu schreiben und für Anregungen dankbar sind (Q2/2004) [11].

Und beim nächsten Mal ...

Als Server zu Entwicklungszwecken wird der Tomcat heute mehr als häufig eingesetzt, aber mal ehrlich: Wer macht sich die Mühe, darüber nachzudenken, wie man sich die Entwicklung mit Tomcat noch vereinfachen kann? Um hierzu ein paar Anregungen zu geben, werden wir in der nächsten Ausgabe als Schwerpunkt-Thema das Deployment des Tomcats über Ant, sowie die Manager- und Admin-Anwendung näher beleuchten. Hoffentlich mit dem kleinem „Aha“-Effekt.

Links & Literatur

[1] jakarta.apache.org/tomcat/

[2] Peter Roßbach (Hrsg.); Andreas Holubek, Thomas Pöschmann, Lars Röwekamp, Peter Tabatt „Tomcat 4X: Die neue Architektur und moderne Konzepte für Web-Anwendungen im Detail“; Software & Support Verlag, 2002

[3] jakarta.apache.org/commons/digester.html
www.onjava.com/pub/a/onjava/2002/10/23/digester.html
www.javaworld.com/javaworld/jw-10-2002/jw-1025-opensourceprofile.html

[4] Tom C@ - Die Referenzkarte, *Java Magazin* 9.2003, neue Versionen: tomcat.objektpark.org/

[5] java.sun.com/products/servlet/download.html

[6] httpd.apache.org/docs/logs.html Extended Log File Format: www.w3.org/TR/WD-logfile.html

[7] www.filip.net/tomcat/

[8] Jason Brittain; Ian F. Darwin; „Tomcat: The Definitive Guide“; O'Reilly, 2003

[9] www.javamagazin.de/tomcat/

[10] www.mail-archive.com/tomcat-user@jakarta.apache.org/
www.mail-archive.com/tomcat-dev@jakarta.apache.org/

[11] tomcat.objektpark.org/