|  |  |
| --- | --- |
|  |  |
|  |  |

# *Servidores de Aplicações Java EE usando Apache Tomcat*



Sumário

[*Servidores de Aplicações Java EE usando Tomcat* 1](#_Toc374093047)

[1. Sobre Este Curso 9](#_Toc374093048)

[1.1. Objetivo deste Curso 9](#_Toc374093049)

[1.2. Quem deve fazer 9](#_Toc374093050)

[1.3. Pré-Requisitos 9](#_Toc374093051)

[1.4. Agenda 9](#_Toc374093052)

[1.5. Ambiente de Sala de Aula 10](#_Toc374093053)

[1.6. Perfil do Administrador de Tomcat 10](#_Toc374093054)

[2. Introdução ao Java EE 10](#_Toc374093055)

[2.1. Edições do Java: Java SE x Java EE 11](#_Toc374093056)

[2.1.1. Containers Java EE 11](#_Toc374093057)

[2.1.2. Padrões Web do Java EE 12](#_Toc374093058)

[2.2. Frameworks Java EE 13](#_Toc374093059)

[2.3. Java em Linux 13](#_Toc374093060)

[2.4. Tomcat x Linux 14](#_Toc374093061)

[2.5. O Projeto JPackage 14](#_Toc374093062)

[2.5.1. Convenções do JPackage 15](#_Toc374093063)

[2.6. Instalação do Java via JPackage 15](#_Toc374093064)

[2.6.1. Ambiente JPackage para aplicações instaladas manualmente 17](#_Toc374093065)

[2.7. Instalação manual do Java 17](#_Toc374093066)

[2.8. Exercícios 18](#_Toc374093067)

[Laboratório 1.1. Instalação do OpenJDK 6 18](#_Toc374093068)

[Laboratório 1.2. Instalação do Sun JDK 5 19](#_Toc374093069)

[2.9. Conclusão 19](#_Toc374093070)

[Questões de Revisão 20](#_Toc374093071)

[3. Instalação do Tomcat 20](#_Toc374093072)

[3.1. Sobre o Apache Tomcat 20](#_Toc374093073)

[3.1.1. Versões do Tomcat 21](#_Toc374093074)

[3.2. Instalação do Tomcat 21](#_Toc374093075)

[3.2.1. Instalação via JPackage 22](#_Toc374093076)

[3.2.2. Iniciando e finalizando o Tomcat como um serviço do Unix 22](#_Toc374093077)

[3.2.3. Instalação Manual do Tomcat 22](#_Toc374093078)

[3.2.4. Início e término manual do Tomcat 23](#_Toc374093079)

[3.3. Testando o Tomcat 24](#_Toc374093080)

[3.4. Se Algo Der Errado 24](#_Toc374093081)

[3.5. Documentação do Tomcat 24](#_Toc374093082)

[3.6. Exercícios 25](#_Toc374093083)

[Laboratório 1.1. Instalação e teste do Tomcat via JPackage 26](#_Toc374093084)

[Laboratório 1.2. Instalação e teste manuais do Tomcat 27](#_Toc374093085)

[Laboratório 1.3. Acesso off-line aos manuais do Tomcat 28](#_Toc374093086)

[3.7. Conclusão 28](#_Toc374093087)

[Questões de Revisão 29](#_Toc374093088)

[4. Arquitetura do Tomcat 29](#_Toc374093089)

[4.1. Componentes do Tomcat 29](#_Toc374093090)

[4.2. Layout de Instalação do Tomcat 30](#_Toc374093091)

[4.2.1. Arquivos de Configuração do Tomcat 31](#_Toc374093092)

[4.3. Sintaxe do server.xml 31](#_Toc374093093)

[4.4. O Tomcat Manager 32](#_Toc374093094)

[4.4.1. Ativação do Manager 33](#_Toc374093095)

[4.5. Exercícios 33](#_Toc374093096)

[Laboratório 1.1. Conhecendo os arquivos de configuração do Tomcat 34](#_Toc374093097)

[Laboratório 1.2. Ativação do Manager 35](#_Toc374093098)

[Laboratório 1.3. Monitoração via script 36](#_Toc374093099)

[4.6. Conclusão 36](#_Toc374093100)

[Questões de Revisão 37](#_Toc374093101)

[5. Deployment de Aplicações 37](#_Toc374093102)

[5.1. Pacotes WAR e Deployment Descriptors 37](#_Toc374093103)

[5.2. Um exemplo mínimo 38](#_Toc374093104)

[5.2.1. Compilando o exemplo 38](#_Toc374093105)

[5.2.2. Empacotando o exemplo 39](#_Toc374093106)

[5.3. Deployment de Aplicações Web 40](#_Toc374093107)

[5.4. Auto-deploy 40](#_Toc374093108)

[5.4.1. Permissões de Arquivos 40](#_Toc374093109)

[5.4.2. Hot-redepoy (atualização à quente) 41](#_Toc374093110)

[5.4.3. Limpando deployments 41](#_Toc374093111)

[5.5. Testando a aplicação recém-deployada 42](#_Toc374093112)

[5.6. Deployment via Manager 42](#_Toc374093113)

[5.6.1. Atualizando e suspendendo de aplicações 44](#_Toc374093114)

[5.7. Desligando o auto-deploy 44](#_Toc374093115)

[5.8. Exercícios 45](#_Toc374093116)

[Laboratório 1.1. Complicação, empacotamento e auto-deploy do exemplo 45](#_Toc374093117)

[Laboratório 1.2. Auto-deploy de pacotes abertos 46](#_Toc374093118)

[Laboratório 1.3. Recarga de pacotes via Manager 48](#_Toc374093119)

[Laboratório 1.4. Deployment manual, via Manager 49](#_Toc374093120)

[Laboratório 1.5. Desligar o auto-deploy 50](#_Toc374093121)

[5.9. Conclusão 50](#_Toc374093122)

[Questões de Revisão 51](#_Toc374093123)

[6. Configuração de Contextos 51](#_Toc374093124)

[6.1. O Descritor Padrão web.xml 51](#_Toc374093125)

[6.1.1. Página inicial da aplicação 52](#_Toc374093126)

[6.1.2. Entradas de ambiente JNDI 52](#_Toc374093127)

[6.2. O descritor proprietário context.xml 52](#_Toc374093128)

[6.2.1. Sobreponto entradas de ambiente 53](#_Toc374093129)

[6.3. Definição de contextos no Tomcat 53](#_Toc374093130)

[6.3.1. Nome (path) e base do contexto 54](#_Toc374093131)

[6.3.2. Contextos explícitos com auto-deploy 54](#_Toc374093132)

[6.3.3. Página inicial do Tomcat 54](#_Toc374093133)

[6.4. Instalando aplicações sem deployment 55](#_Toc374093134)

[6.5. Exercícios 55](#_Toc374093135)

[Laboratório 1.1. Entradas de ambiente 56](#_Toc374093136)

[Laboratório 1.2. Sobrepondo entradas de ambiente 57](#_Toc374093137)

[Laboratório 1.3. Múltiplas instâncias da mesma aplicação, via server.xml 58](#_Toc374093138)

[Laboratório 1.4. Múltiplas instâncias da mesma aplicação, via conf/engine/host e aplicação padrão do Tomcat 59](#_Toc374093139)

[Laboratório 1.5. Mudando a página inicial da aplicação 60](#_Toc374093140)

[6.6. Conclusão 60](#_Toc374093141)

[Questões de Revisão 61](#_Toc374093142)

[7. Configuração de Contextos 61](#_Toc374093143)

[7.1. O Que é o Ant 61](#_Toc374093144)

[7.2. Instalação via JPackage 62](#_Toc374093145)

[7.3. Instalação manual do Ant 62](#_Toc374093146)

[7.3.1. Instalação manual x JPackage 62](#_Toc374093147)

[7.4. Verificando a instalação do Ant 62](#_Toc374093148)

[7.5. Sintaxe dos buildfiles 63](#_Toc374093149)

[7.5.1. Um exemplo de buildfile 63](#_Toc374093150)

[7.5.2. Um exemplo de projeto Java com Ant 65](#_Toc374093151)

[7.6. Tasks customizados do Tomcat 65](#_Toc374093152)

[7.7. Exercícios 67](#_Toc374093153)

[Laboratório 1.1. Projeto com Ant 68](#_Toc374093154)

[Laboratório 1.2. Deployment via Ant com Manager 69](#_Toc374093155)

[7.8. Conclusão 69](#_Toc374093156)

[8. Bibliotecas 69](#_Toc374093157)

[8.1. Bibliotecas Java 69](#_Toc374093158)

[8.1.1. Onde instalar bibliotecas 69](#_Toc374093159)

[8.2. Classloaders do Java x Tomcat 70](#_Toc374093160)

[8.2.1. Configurando pastas adicionais para bibliotecas 70](#_Toc374093161)

[8.3. Exercícios 71](#_Toc374093162)

[Laboratório 1.1. Aplicação com biblioteca externa 72](#_Toc374093163)

[Laboratório 1.2. Pasta adicional de bibliotecas 74](#_Toc374093164)

[Laboratório 1.3. Versões diferentes da mesma classe 75](#_Toc374093165)

[8.4. Conclusão 76](#_Toc374093166)

[Questões de Revisão 77](#_Toc374093167)

[9. Acesso a Bancos de Dados 77](#_Toc374093168)

[9.1. Introdução ao JDBC 77](#_Toc374093169)

[9.1.1. Tipos de drivers JDBC 78](#_Toc374093170)

[9.2. DataSources Java EE 78](#_Toc374093171)

[9.2.1. DataSources e o JNDI 78](#_Toc374093172)

[9.2.2. Porque usar DataSources 79](#_Toc374093173)

[9.2.3. Pools de Conexão 79](#_Toc374093174)

[9.3. Configurando DataSources no Tomcat 80](#_Toc374093175)

[9.3.1. Links para recursos globais 81](#_Toc374093176)

[9.4. Exercícios 81](#_Toc374093177)

[Laboratório 1.1. DataSource local 82](#_Toc374093178)

[Laboratório 1.2. DataSource global 83](#_Toc374093179)

[9.5. Conclusão 83](#_Toc374093180)

[Questões de Revisão 84](#_Toc374093181)

[Questões de Revisão 85](#_Toc374093182)

[10. Autenticação HTTP 85](#_Toc374093183)

[10.1. Segurança Declarativa do Java EE 85](#_Toc374093184)

[10.1.1. Resource Collections do Java EE 85](#_Toc374093185)

[10.1.2. Segurança Programática 86](#_Toc374093186)

[10.2. Métodos de autenticação HTTP 87](#_Toc374093187)

[10.2.1. Limitações dos métodos HTTP 87](#_Toc374093188)

[10.2.2. Regras de autenticação no descritor web.xml 87](#_Toc374093189)

[10.2.3. Autenticação FORM 88](#_Toc374093190)

[10.2.4. Logout por inatividade 89](#_Toc374093191)

[10.3. A Base de Identidade default do Tomcat 89](#_Toc374093192)

[10.4. Single Sign-On 89](#_Toc374093193)

[10.5. Exercícios 90](#_Toc374093194)

[Laboratório 1.1. Aplicação com autenticação BASIC 91](#_Toc374093195)

[Laboratório 1.2. Aplicação com autenticação FORM 92](#_Toc374093196)

[Laboratório 1.3. Singe Sign-On 93](#_Toc374093197)

[10.6. Conclusão 93](#_Toc374093198)

[Questões de Revisão 94](#_Toc374093199)

[11. Bases de Identidade (Realms) do Tomcat 94](#_Toc374093200)

[11.1. Realms do Tomcat 94](#_Toc374093201)

[11.1.1. Armazenando senhas criptogradas 95](#_Toc374093202)

[11.2. Autenticando contra um Banco de Dados 96](#_Toc374093203)

[11.3. Autenticando contra um diretório LDAP 96](#_Toc374093204)

[11.4. Exercícios 97](#_Toc374093205)

[Laboratório 1.1. Armazenando Senhas Criptografadas 98](#_Toc374093206)

[Laboratório 1.2. Autenticando contra um banco de dados 99](#_Toc374093207)

[Laboratório 1.3. Autenticando contra um diretório LDAP 100](#_Toc374093208)

[11.5. Conclusão 100](#_Toc374093209)

[Questões de Revisão 101](#_Toc374093210)

[12. Logging no Tomcat 101](#_Toc374093211)

[12.1. Logging no Tomcat 101](#_Toc374093212)

[12.2. Conceitos de Logging do Java SE 101](#_Toc374093213)

[12.2.1. Porque programar para a API de logging 102](#_Toc374093214)

[12.3. O Juli do Tomcat 103](#_Toc374093215)

[12.3.1. Arquivos de log na configuração default 104](#_Toc374093216)

[12.4. Logs da aplicação 105](#_Toc374093217)

[12.4.1. Uma configuração para produção 106](#_Toc374093218)

[12.5. Logs de Acesso 107](#_Toc374093219)

[12.6. Exercícios 107](#_Toc374093220)

[Laboratório .1. Aplicação utilizando a API de logging 108](#_Toc374093221)

[Laboratório 1.2. Configuração de logging para produção 110](#_Toc374093222)

[Laboratório 1.3. Logs de acesso 111](#_Toc374093223)

[12.7. Conclusão 111](#_Toc374093224)

[Questões de Revisão 112](#_Toc374093225)

[13. Configurações de Rede 112](#_Toc374093226)

[13.1. Conectores e Válvulas do Tomcat 112](#_Toc374093227)

[13.1.1. Configuração padrão do Coyote 113](#_Toc374093228)

[13.1.2. Rodando o Tomcat na porta 80 114](#_Toc374093229)

[13.2. Firewall interno do Tomcat 114](#_Toc374093230)

[13.3. Conexões SSL com o Tomcat 115](#_Toc374093231)

[13.3.1. Gerando certificados com o keytool 115](#_Toc374093232)

[13.3.2. Segurança declarativa x SSL 117](#_Toc374093233)

[13.4. Hosts Virtuais 117](#_Toc374093234)

[13.4.1. Manager x host virtual 117](#_Toc374093235)

[13.5. Exercícios 118](#_Toc374093236)

[Laboratório 1.1. Tomcat como web server principal 119](#_Toc374093237)

[Laboratório 1.2. Restringindo acesso por IP 120](#_Toc374093238)

[Laboratório 1.3. Ativando o suporte a https: 121](#_Toc374093239)

[Laboratório 1.4. Protegendo páginas com https: 122](#_Toc374093240)

[Laboratório 1.5. Implementando um host virtual 123](#_Toc374093241)

[13.6. Conclusão 124](#_Toc374093242)

[Questões de Revisão 125](#_Toc374093243)

[14. Integração com Servidores Web Nativos 125](#_Toc374093244)

[14.1. Porque usar o Tomcat junto a um servidor web nativo 125](#_Toc374093245)

[14.2. Sobre o mod\_jk 126](#_Toc374093246)

[14.3. Instalação do mod\_jk 127](#_Toc374093247)

[14.4. Configuração do mod\_jk 128](#_Toc374093248)

[14.4.1. Dividindo tarefas entre o Tomcat e o Apache 129](#_Toc374093249)

[14.5. Integração de Segurança e Logging 130](#_Toc374093250)

[14.6. Exercícios 130](#_Toc374093251)

[Laboratório 1.1. Integração Apache com Tomcat 131](#_Toc374093252)

[Laboratório 1.2. Servindo arquivos estáticos no Apache 132](#_Toc374093253)

[14.7. Conclusão 132](#_Toc374093254)

[Questões de Revisão 133](#_Toc374093255)

[15. Clusters para Escalabilidade 133](#_Toc374093256)

[15.1. Conceitos de clusters Java EE 133](#_Toc374093257)

[15.1.1. Estado em uma aplicação web Java EE 134](#_Toc374093258)

[15.1.2. O Balanceador em um cluster Java EE 134](#_Toc374093259)

[15.1.3. Arquitetura de cluster web do Java EE 135](#_Toc374093260)

[15.2. Configurando o mod\_jk como balanceador 135](#_Toc374093261)

[15.2.1. A página de status do mod\_jk 136](#_Toc374093262)

[15.3. Configurando novas instâncias do Tomcat no mesmo servidor 137](#_Toc374093263)

[15.3.1. Vantagens de se rodar várias instâncias do Tomcat 138](#_Toc374093264)

[15.4. Exercícios 138](#_Toc374093265)

[Laboratório 1.1. Duas instâncias do Tomcat lado-a-lado 139](#_Toc374093266)

[Laboratório 1.2. Um cluster para escalabilidade 140](#_Toc374093267)

[Laboratório 1.3. Status do cluster 141](#_Toc374093268)

[Laboratório 1.4. Desligando a afinidade de sessão 142](#_Toc374093269)

[15.5. Conclusão 142](#_Toc374093270)

[Questões de Revisão 143](#_Toc374093271)

[16. Cluster para HA – Alta Disponibilidade 143](#_Toc374093272)

[16.1. Conceitos de clusters Java EE 143](#_Toc374093273)

[16.1.1. Aplicações cluster-aware 144](#_Toc374093274)

[16.1.2. Erros de aplicação a evitar 144](#_Toc374093275)

[16.2. Arquitetura de Cluster HA do Tomcat 144](#_Toc374093276)

[16.3. Configuração da Replicação de Sessão 145](#_Toc374093277)

[16.3.1. Vários clusters na mesma rede 146](#_Toc374093278)

[16.3.2. Configuração de Multicast no Linux 147](#_Toc374093279)

[16.3.3. Cluster x Manager 147](#_Toc374093280)

[16.4. Exercícios 147](#_Toc374093281)

[Laboratório 1.1. Cluster com HA 148](#_Toc374093282)

[16.5. Conclusão 149](#_Toc374093283)

[Questões de Revisão 150](#_Toc374093284)

[17. Monitoração e Tuning 150](#_Toc374093285)

[17.1. Modelo de performance para aplicações Java EE 150](#_Toc374093286)

[17.2. Monitorando o Tomcat via Manager 151](#_Toc374093287)

[17.2.1. Tuning de conector x balanceador 151](#_Toc374093288)

[17.2.2. Dimensionamento de DataSources 151](#_Toc374093289)

[17.3. Memória da JVM 151](#_Toc374093290)

[17.3.1. Passando opções para a JVM 152](#_Toc374093291)

[17.4. Tuning do Cluster 152](#_Toc374093292)

[17.5. Monitoração via JMX 153](#_Toc374093293)

[17.5.1. Usando o JConsole com o Tomcat 153](#_Toc374093294)

[17.5.2. Acesso remoto JMX 154](#_Toc374093295)

[17.5.3. JMX Proxy Servlet do Manager 154](#_Toc374093296)

[17.6. Troubleshooting 156](#_Toc374093297)

[17.6.1. OutOfMemory naPermGen 156](#_Toc374093298)

[17.6.2. datasource leaks 156](#_Toc374093299)

[17.6.3. Servidor Congelado 157](#_Toc374093300)

[17.7. Exercícios 158](#_Toc374093301)

[Laboratório 1.1. Memória com o Manager e ps 159](#_Toc374093302)

[Laboratório 1.2. Monitoração JMX via JConsole 160](#_Toc374093303)

[Laboratório 1.3. Monitoração JMX via Manager 161](#_Toc374093304)

[17.8. Conclusão 161](#_Toc374093305)

[Questões de Revisão 162](#_Toc374093306)

# Sobre Este Curso

Este capítulo apresenta o curso “Servidores de Aplicação Java EE com Tomcat”

* Objetivo
* Público-Alvo
* Pré-Requisitos
* Ambiente para Laboratórios
* Perfil do Administrador de Tomcat

## Objetivo deste Curso

Capacitar profissionais na administração e gerenciamento de servidores de aplicação Tomcat, tanto em ambiente de desenvolvimento quanto em ambiente de produção.

## Quem deve fazer

* Administradores de rede responsáveis por manter um servidor Tomcat como parte de um Portal, Intranet ou Extranet;
* Programadores e analistas de sistemas responsáveis pelo desenvolvimento de aplicações Web utilizando a plataforma Java EE;
* Administradores de rede e desenvolvedores interessados em obter conhecimentos sobre como construir, manter e otimizar uma infra-estrutura de produção baseada em servidores de aplicação Java EE.

## Pré-Requisitos

Estes conhecimentos são indispensáveis ao futuro administrador de servidores Tomcat:

* Leitura básica em Inglês Técnico;
* Conhecimentos básicos de HTML e HTTP (navegadores e servidores Web);
* Conhecimentos básicos de TCP/IP.

Também é desejável que o aluno tenha os seguintes conhecimentos:

* Administração do sistema operacional utilizado para o servidor;
* Programação na linguagem Java;
* Compilação de programas na linha de comando utilizando o JDK;
* Acesso a bancos de dados utilizando JDBC;
* Construção de servlets e páginas JSP.

A falta destes conhecimentos não irá prejudicar o aproveitamento do aluno neste curso, mas irá afetar seu desempenho profissional na área.

## Agenda

O curso é organizado em uma sucessão de tópicos conceituais e práticos que refletem na medida do possível a ordem com que um administrador típico irá encontrar cada tarefa dentro de um ambiente real de trabalho.

1. Introdução ao Java e ao Enterprise Java
2. Instalação do Tomcat
3. Deployment de Aplicação
4. Configuração de Contextos e Recursos
5. Configurações de Segurança
6. Configurações de Rede
7. Integração com servidores web nativos
8. Configuração de Clusters
9. Tuning, monitoração e troubleshooting

## Ambiente de Sala de Aula

Os laboratórios deste curso serão realizados em Linux, mais precisamente no Fedora, que é uma distribuição livre que há vários anos incorpora no suporte a aplicações Java como o Tomcat. Hoje outras distribuições populares como o Debian suportam aplicações Java como parte integrante da distribuição, mas o Fedora foi o pioneiro na inclusão de software como o Ant e o Eclipse, além do próprio Tomcat.

Então serão vistos tópicos específicos do SO Linux e da distribuição como a configuração de variáveis de ambiente e permissões de arquivos. Também serão vistos tópicos específicos para o Fedora, como a instalação de pacotes RPM, que seriam facilmente adaptados para outras distribuições por usuários com os conhecimentos necessários de sysadmin.

Apesar disso, o Tomcat e os programas-exemplo deste curso são 100% Java, de modo que é possível realizar a maior parte dos laboratórios em Windows ou Mac. A administração do Tomcat em si não depende do SO subjacente, mas algumas características de tuning, segurança e cluster dependem da interação do Tomcat com este SO. Portanto não é possível oferecer um curso realista de administração do Tomcat totalmente indiferente ao SO do servidor.

## Perfil do Administrador de Tomcat

O papel do Administrador de um Servidor de Aplicações Java EE, ou ASA (Application Server Administrador) como o Tomcat é manter o ambiente de produção para aplicações. A performance e estabilidade deste ambiente depende da qualidade e outras especificidades destas aplicações

É algo bem diferente da administração de um serviço de rede típico, por exemplo um proxy web, servidor de e-mail ou arquivos, que depende apenas do código do próprio serviço.

Então o ASA necessita conhecimentos tanto de infra-estrutura quanto de desenvolvimento, pois ele está na interseção entre estes dois universos. Ele precisa ser capaz de diferenciar problemas de configuração do servidor de aplicações de problemas de projeto ou codificação das aplicações hospedadas pelo servidor. O ASA também deve orientar os desenvolvedores no melhor aproveitamento das características do Tomcat em si e do ambiente Java EE em geral.

Na verdade, o ASA tem um perfil bastante semelhante ao de um Administrador de Banco de Dados ou DBA (DataBase Administrador) em relação ao conhecimento exigido e interação tanto com equipes de desenvolvimento quanto de infra-estrutura de SO e redes. Alguns até arriscam o prognóstico de que em um futuro próximo o ASA ocupará o lugar do DBA como profissional mais valorizado dentro do ambiente de TI.

# Introdução ao Java EE

Neste capítulo apresentamos conceitos essenciais sobre o Java EE, validamos o ambiente para a instalação do Tomcat em Linux e exercitamos o uso das ferramentas do JDK na linha de comando.

**Tópicos:**

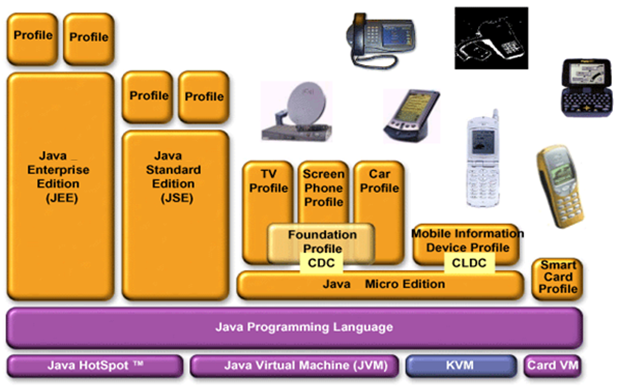
* Conceitos do Java EE
* JPackage, GCJ e Sun JDK
* Instalação do Java via gerenciador de pacotes
* Instalação Manual do Java

## Edições do Java: Java SE x Java EE

A plataforma Java é organizada em várias edições, que reúnem facilidades para ambientes computacionais específicos. Neste curso, estamos interessados em duas destas edições: Java SE e Java EE[[1]](#footnote-1).

O Java SE ou JSE (antigo J2SE) é o Java Standard Edition. Ele fornece a JVM (máquina virtual Java) e as APIs essenciais para coleções, E/S, reflexão, serialização, XML, interfaces gráficas e redes.

Os downloads e instaladores do Java SE podem fornecer o JRE (Java Runtime-Environment) que contém a JVM em si, ou o JDK (Java Development Kit) que traz o JRE e também o compilador Java. O primeiro é necessário para execução de aplicações Java entregues prontas, enquanto que o segundo é necessário para o desenvolvimento de novas aplicações Java.



**Figura 1.1** – Aplicação x Java EE x Java SE

Já o Java EE ou JEE (antigo J2EE) é o Java Enterprise Edition ou “Enterprise Java”. Ele fornece as tecnologias e APIs da plataforma Java para computação baseada em servidor, incluindo conectividade com bancos de dados, serviços de diretórios, correio eletrônico e outros serviços de rede.

O Java EE necessita de um Java SE para funcionar, ou seja, um é instalado sobre o outro. Para conveniência dos usuários, o lado cliente das tecnologias padronizadas pelo Java EE é fornecida nos instaladores do Java SE, de modo que uma instalação do Java EE é necessário apenas para o ambiente de servidor, ou então para ambiente de desenvolvimento.

A figura 1.1 ilustra o relacionamento entre uma aplicação Java EE, o próprio Java EE e o Java SE.

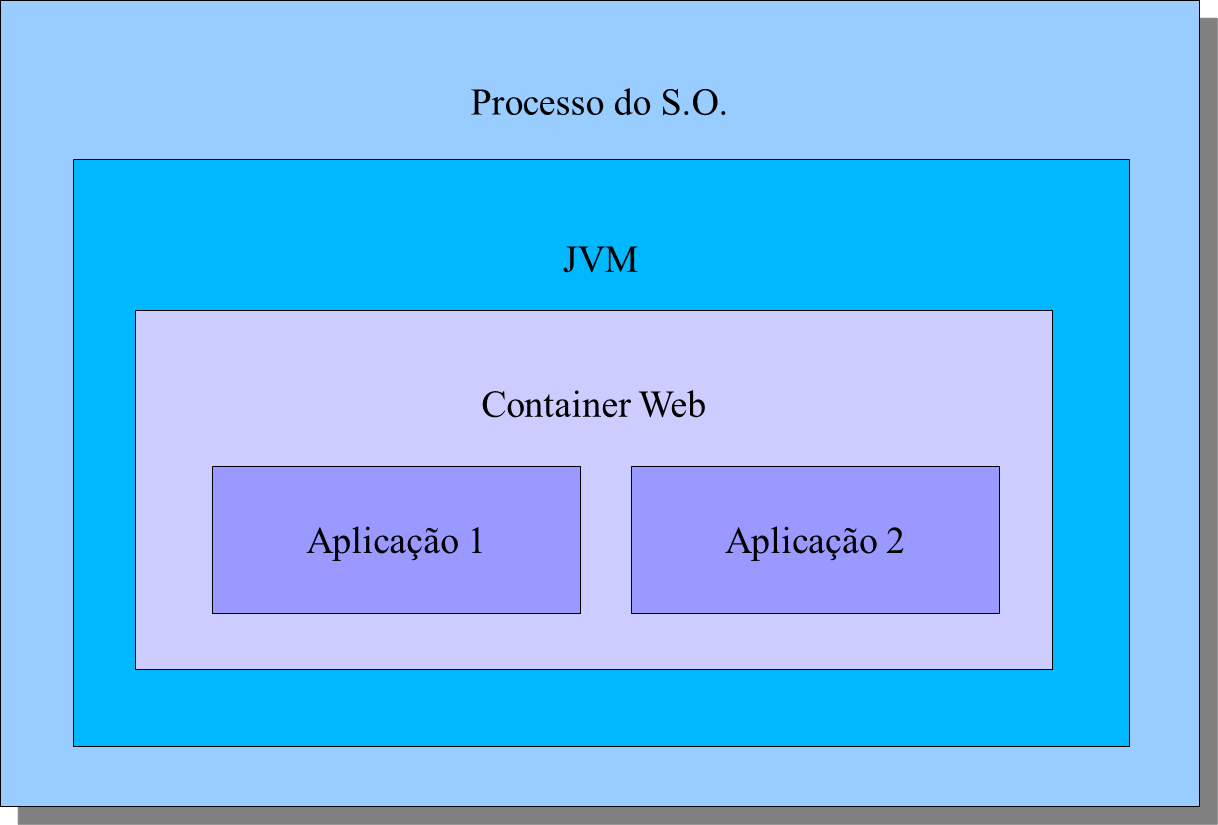
### Containers Java EE

São previstos servidores especializados para aplicações Java EE, os chamados Servidores de Aplicação. Uma aplicação Java EE não roda de forma isolada, e roda dentro do servidor de aplicações. O Tomcat é um exemplo de servidor de aplicações, embora ele não seja completo e por isso não seja certificado, coisa que não diminui em nada sua popularidade mesmo em ambientes de missão-crítica.

Um servidor de aplicações Java EE é formado por um ou mais containers:

* Container Web: hospeda aplicações web, acessadas por meio de um navegador padrão;
* Container EJB: hospeda componentes / objetos distribuídos construídos segundo o padrão EJB (Enterprise Java Beans) e compatíveis com padrões CORBA;
* Container de aplicação: hospeda aplicações cliente stand-alone, fornecendo o ambiente necessário para conexão aos serviços fornecidos por um servidor de aplicações.

Cada container fornece a interface entre os componentes de aplicação instalados ou hospedados dentro dele e os serviços do servidor de aplicações, como ilustra a figura 1.2. Muitas vezes confunde-se o container com o próprio servidor de aplicações, pois para o desenvolvedor de aplicações não há diferença concreta. Mas para o administrador é importante lembrar que o container é apenas parte de um servidor de aplicação e que sua atuação na configuração e tuning do ambiente poderá envolver outras partes do servidor.



**Figura 1.2** – Aplicação x Container x SO em tempo de execução

### Padrões Web do Java EE

Aplicações Web do Java EE utilizam uma série de APIs e componentes padronizados, a saber

* Servlets são classes Java criadas para serem gerenciadas por um container web;
* Páginas JSP são páginas HTML contendo comandos Java e tags customizadas, que são transformadas em Servlets para execução pelo Container Web;
* JSTL é um conjunto padrão de tags customizados para uso em páginas JSP;
* JSF é um framework que fornece um modelo de componentes e de eventos dentro de páginas JSP semelhante ao utilizado em aplicações gráficas tradicionais.

Dependendo da versão do Java EE suportada por um container, ele pode incorporar ou não diferentes versões e combinações destes padrões. Podem haver ainda implementações destes padrões externas ao servidor de aplicações, úteis para complementar produtos que não tragam uma implementação do padrão na sua configuração “de fábrica”.

Uma aplicação Java EE com interface web também poderá fazer uso de outras APIs padronizadas da plataforma, por exemplo:

* JTA cuida de transações distribuídas;
* JNDI permite acesso a serviços de diretório e objetos compartilhados entre aplicações e os containers;
* JDBC para acesso a bancos de dados relacionais;
* JavaMail para acesso a servidores de e-mail Internet;
* JMS para acesso a servidores de mensagens;
* JMX para gerenciamento local ou remoto;
* JAX-RPC e JAX-WS para web services.

Esta não é uma relação exaustiva, porque neste curso estamos focados apenas nos padrões da parte web, que é o foco do Tomcat. Alguns destes padrões poderão estar disponíveis no Tomcat ou no próprio Java SE, mas como já foi dito ele não fornece todos os componentes do Java EE.

## Frameworks Java EE

O desenvolvimento profissional Java EE complementa os padrões e APIs da plataforma com uma série de frameworks, a maioria deles open source:

* Struts permite a construção de aplicações segundo o modelo MVC (Model-View-Controller) onde é fornecido um Servlet controlador e o desenvolvedor cria suas páginas JSP como visões e classes JavaBeans como modelo;
* Hibernate permite o uso de um banco relacional de forma orientada a objetos;
* Spring e Wicket são outros frameworks populares entre os desenvolvedores Java EE.

Estes frameworks não são parte do padrão Java EE e normalmente não são fornecidos como parte dos servidores de aplicação, mas seu uso é tão disseminado que considera-se parte obrigatória do conhecimento de um desenvolvedor Java EE. Já o ASA necessita saber como inclui-los no servidor de aplicações, e que implicações seu uso por um desenvolvedor tem sobre as configurações de segurança e perforance do servidor.

## Java em Linux

Sempre houve muita controvérsia em relação ao Java dentro da comunidade de Software Livre. Os mais radicais enxergam o Java como um produto proprietário da Sun, que portanto seria o “inimigo” assim como produtos proprietários da Microsoft e Oracle.

Por outro lado, o Java da Sun sempre foi um download gratuito e de uso irrestrito, tanto em ambientes comerciais quanto acadêmicos, o que levou muitos a utilizarem o Java da mesma forma que utilizavam PHP, MySQL e outros softwares abertos.

Entretanto, a licença de uso original do Java da Sun restringe a distribuição e modificação, impedindo sua inclusão na maioria das distribuições do Linux, mesmo aquelas que não tinham restrições a software proprietário. Por isso que até 2005 era difícil encontrar aplicações Java como parte de distribuições do Linux

A situação começou a mudar quando de implementações livres do Java, baseadas no GNU Classpath, se tornaram maduras o suficientes para permitir a inclusão de aplicações Java em distribuições como Fedora e Debian. Entre elas temos o GCJ, Kaffe, CacaoJVM e Jikes. O GCJ foi a mais popular, sendo utilizada até recentemente como o ambiente Java padrão do Fedora para a execução do próprio Tomcat, Eclipse e componentes Java do OpenOffice (como o Base e vários filtros de importação de documentos).

Em resposta à pressão dos usuários Linux (que incluíam vários dos maiores servidores rodando sites web com Java) e ao avanço das implementações livres do Java, a Sun passou a oferecer uma licença alternativa, a JDL (Java Distribution License) que permitiu a inclusão do Java da Sun no Debian e derivados.

Mas a JDL não foi aceita por outras distribuições como Fedora e Red Hat, que consideraram seus termos ambíguos e julgaram que poderiam colocar as distribuições e seus desenvolvedores sob o risco de processos pela própria Sun. Na própria comunidade Debian a JDL não foi inteiramente aceita, sendo a iniciativa e talvez o erro dos desenvolvedores do Ubuntu que colocaram o JDK da Sun nos repositórios do Debian.

Em 2007 a Sun finalmente decidiu abrir a sua implementação do Java, iniciando o projeto OpenJDK, utilizando a licença GPL+Classpath Exception, que não por acaso era a mesma utilizada pelo GCJ.

Entretanto a Sun não era proprietária de todos os componentes do seu JDK, que depende de alguns componentes binários, não-livres. Ela continuou fornecendo estes componentes proprietários sob a mesma licença de uso irrestrita e até mesmo relaxou as condições de redistribuição para os usuários do OpenJDK, mas ainda assim os termos eram inaceitáveis para sua inclusão como componente padrão da maioria das distribuições do Linux.

Aproveitando a compatibilidade de licenças entre o OpenJDK e o GCJ, alguns desenvolvedores iniciaram o processo de migrar componentes do GNU Classpath para o OpenJDK, substituindo os componentes proprietários originais. Assim nasceu o IcedTea, um Java inteiramente livre baseado no OpenJDK e no GNU Classpath.

Apesar do progresso de implementações do Java baseados no GNU Classpath, os únicos “Javas” certificados pelo JCP (Java Community Process) até recentemente eram os produtos proprietários da Sun, IBM e BEA.

Eles são fornecidos como parte de distribuições “Enterprise” do Linux, condicionadas aos contratos de suporte, como o RHEL e SuSE, que permitem o pagamento aos fornecedores originais.

Não há garantia formal de que aplicações Java do mercado funcionem corretamente com Javas não-certificados, impedindo seu uso do GCJ, OpenJDK e etc em ambientes corporativos. Felizmente, o próprio JCP flexibilizou as regras do processo de certificação e forneceu uma “bolsa” para a certificação de produtos open source.

Alguns servidores de aplicação livres, como o JBoss e o Gerônimo, já haviam usufruido desta bolsa para sua certificação Java EE. Entretanto, a certificação Java, em qualquer edição, é tudo ou nada. Então o IcedTea foi o primeiro Java livre a se tornar completo[[2]](#footnote-2) e portanto certificável.

A Sun ofereceu o patrocínio para a certificação do IcedTea, com a condição de que as distribuições binárias usassem o nome OpenJDK. Então, em julho de 2008, o OpenJDK-IcedTea incluso no Fedora 9 foi formalmente certificado como aderente padrão Java SE 6, nas plataformas Intel de 32 e 64-bits[[3]](#footnote-3).

Até o momento, não existem versões certificadas do IcedTea ou outro Java livre para SOs que não Linux e plataformas que não a Intel. Então usuários de Windows, Mac, Solaris e FreeBSD estão restritos a Javas proprietários, caso considerem a certificação como um requisito indispensável.

## Tomcat x Linux

Como já foi citado antes nesta apostila, várias distribuições do Linux hoje incorporam o Tomcat e outros softwares Java populares, por exemplo Ant, Struts e Eclipse. Na verdade, desde que o GCJ se tornou parte oficial do Fedora que esta distribuição traz um ambiente completo para o desenvolvimento e execução de aplicações web Java EE.

Mas pode acontecer do ambiente Java livre da sua distribuição não ser suficiente para executar sua aplicação, seja porque falta algum recurso previsto pelos padrões Java SE e Java EE, seja porque sua empresa exige um ambiente certificado.

Mesmo quando a distribuição inclui o OpenJDK certificado, isto pode não ser suficiente, pois o OpenJDK foi certificado no Java SE 6, e algumas aplicações e bibliotecas podem ter problemas de compatibilidade com esta versão do Java.

Este tipo de incompatibilidade é sem dúvida um bug da aplicação ou biblioteca, mas nem sempre é possível esperar pela correção, então o administrador pode se ver forçado a instalar um ambiente Java proprietário compatível (certificado) para o Java SE 5 ou mesmo para o 1.4.

Neste caso, recomendamos fortemente que o administrador utilize pacotes compatíveis com o projeto JPackage, em vez de usar diretamente os instaladores fornecidos pela Sun, IBM ou BEA.

## O Projeto JPackage

Os downloads oficiais dos fornecedores de Java certificados não são aderentes aos padrões do Linux, como o LSB (Linux Standards Base). Eles não respeitam as convenções de uso do sistema de arquivos, por exemplo instalação de arquivos de configuração em /etc e de comandos em /usr/bin. O instalador da Sun, em particular, não chega nem mesmo a configurar o ambiente para execução do Java na linha de comando[[4]](#footnote-4).

Já os downloads dos mesmos produtos, inclusos nas distribuições RHEL e SuSE, são aderentes, por seguirem as regras de empacotamentos definidos pelo Projeto JPackage. Mas, se você usa Debian, Fedora ou mesmo CentOS, e não pode usar o OpenJDK incluso nestas distribuições, está sem sorte.

Usuários de distribuições da comunidade podem seguir as instruções em jpackage.org para re-empacotar o Java da Sun e assim ter um ambiente Java ao mesmo tempo certificado e compatível com as suas distribuições.

É um processo trabalhoso, mas segui-lo traz vários benefícios:

* Arquivos de configuração, log, bibliotecas e binários são instalados nos diretórios padrão do Linux, como /etc, /var/log, /usr/lib e /usr/bin, facilitando a vida do sysadmin, que passa a lidar com o software Java da mesma forma como ele lida com o software nativo da distribuição;
* Não existem múltiplas cópias das mesmas bibliotecas, como Jakarta-Commons, simplificando a instalação de atualizações de segurança e evitando o risco de que versões antigas e inseguras de uma biblioteca continuem em uso e deixem o servidor exposto; como bônus, o servidor consome menos espaço em disco e menos memória RAM para rodar as aplicações Java;
* Utiliza o sistema alternatives para gerenciar a instalação simultânea de múltiplas versões do mesmo software.

Então é possível ter, ao mesmo tempo, o GCJ e o Sun JDK, ou várias versões do Sun JDK, instaladas em um sistema baseado no JPackage. Melhor ainda, aplicações diferentes podem ser configuradas para versões diferentes do Java, ou mesmo para produtos Java específicos, de modo que elas podem rodar ao mesmo tempo e cada qual utilizando a versão correta do Java.

Administradores e desenvolvedores Windows, que estão sempre brigando com diferentes versões do Java instalados por programas da IBM e Oracle, certamente apreciariam ter uma forma padrão de lidar com estas múltiplas instalações.

### Convenções do JPackage

O JPackage é o upstream das distribuições do Linux baseadas em pacotes RPM (Red Hat, CentOS, Fedora, SuSE, Mandriva) mas suas convenções também são utilizadas pelos empacotadores Debian – há vários membros da comunidade Debian dentro do projeto JPackage. Isto permite ao Linux ter um ambiente uniforme no suporte à aplicações Java.

As principais convenções do JPackage são:

* Bibliotecas Java são instaladas em /usr/share/java. Caso haja necessidade de bibliotecas nativas (acessadas via JNI) elas são instaladas em /usr/lib junto às demais bibliotecas nativas;
* Os arquivos jar (bibliotecas Java) são nomeados conforme o número de versão, e links simbólicos permitem a aplicações referenciar a blbioteca sem incluir a versão ou usando os nomes de versões mais antigas;
* Executáveis do JRE e JDK como java, javac, keytool, rmiregistry e etc são na verdade links simbólicos gerenciados pelo alternatives;
* As instalações dos JRE e JDKs recebem vários links simbólicos em /usr/lib/jvm, de modo que seja possível configurar o JAVA\_HOME, PATH, CLASSPATH e outras variáveis de ambiente necessárias para aplicações Java de forma específica ou genérica, ex: sun-java-1.5.0\_17 x java-1.5.0;
* Documentação JavaDoc é instalada em /usr/share/javadoc;
* Aplicações Java tem seus arquivos distribuídos pelo sistema de arquivos, segundo as convenções do LSB (/etc, /usr/bin, /var/log, etc) mas também tem um diretório em /usr/share que emula seu layout original de instalação, contendo links simbólicos para os diretórios e arquivos da aplicação. Assim aplicações e scripts que esperam o layout original de instalação da aplicação continuam funcionando.

As convenções do JPackage são em si coerentes com as convenções do LSB e outros padrões do Unix, como o Posix. A pasta /usr/share abriga arquivos de programas que sejam independentes a arquitetura de processador, o que é seguramente o caso de aplicações e bibliotecas Java.

## Instalação do Java via JPackage

O aluno pode passar pelo processo fácil ou pelo complicado. O processo fácil é usar os pacotes fornecidos pela distribuição, enquanto que o processo difícil é reempacotar o Java da Sun.

No primeiro caso, basta usar o yum para instalar os pacotes do OpenJDK:

# yum -y install java-1.6.0-openjdk java-1.6.0-openjdk-devel

Estes pacotes (e suas respectivas dependências) fornecem o JRE e JDK conforme implementados pelo OpenJDK-IcedTea certificado no Fedora. Caso se deseje também o Java Plugin (para rodar applets) e a documentação das APIs do Java SE, instale também os pacotes:

# yum -y install java-1.6.0-openjdk-plugin java-1.6.0-openjdk-javadoc

Caso você esteja utilizando RHEL e tenha habilitado na sua subscrição do RHN o canal Supplemental (ou então tenha baixado o arquivo iso com esta mídia de instalação) também poderá usar o comado yum ou então o comando rpm para instalar os pacotes do JDK proprietário da Sun, nas versões 1.4.2, 1.5.0 ou 1.6.0. Estes pacotes são compatíveis com o JPackage.

No segundo caso, deve ser seguido o roteiro fornecido em:

http://jpackage.org/installation.php

ou então usar o script shell rebuild-sun-java-rpm.sh fornecido junto aos exemplos do curso. Este script é uma evolução do script originalmente proposto em:

http://weblogs.java.net/blog/fVilaça/archive/2006/08/afraid\_of\_tryin.html

e que automatiza o processo de re-empacotamento do JDK da Sun. Depois disso, deverá usar o comando rpm para instalar os pacotes recém-gerados.

Uma vez feita a instalação do JDK desejado, seja o OpenJDK 6 ou o Sun JDK 5, deve ser utilizando o comando alternatives para garantir que a versão desejada fornece o JRE default para o sistema operacional, por exemplo:

# **alternatives --config java**

Há 2 programas que oferecem "java".

Seleção Comando

-----------------------------------------------

\*+ 1 /usr/lib/jvm/jre-1.6.0-openjdk.x86\_64/bin/java

2 /usr/lib/jvm/jre-1.5.0-gcj/bin/java

Indique para manter a seleção atual[+] ou digite o número da seleção:

O mesmo comando deve ser utilizado com o argumento “javac” de modo a configurar o JDK default. Em geral o JRE (comando java) e o JDK (comando javac) devem estar na mesma versão e fornecedor, por exemplo ambos com o OpenJDK 6 ou então ambos com o Sun JDK 5.

Em algumas versões do Debian, usar o alternatives para configurar os comandos java e javac não afeta outros comandos fornecidos pelo JRE e JDK, por exemplo rmiregistry e keytool. Neste caso, será necessário configurar cada comando individualmente usando o alternatives.

Por fim, utilize a opção de linha de comando -version com os comandos java e javac para se certificar de que a versão correta esteja valendo como default, por exemplo:

# **java -version**

java version "1.6.0\_0"

OpenJDK Runtime Environment (IcedTea6 1.6) (fedora-29.b16.fc11-x86\_64)

OpenJDK 64-Bit Server VM (build 14.0-b16, mixed mode)

### Ambiente JPackage para aplicações instaladas manualmente

Mesmo que o aluno esteja utilizando os pacotes da distribuição (ou baixando diretamente do JPackage) para a instalação do JDK, Tomcat e etc, pode ocorrer a necessidade de instalar manualmente uma aplicação Java à partir do seu download original, em geral um arquivo zip. Um exemplo seria a necessidade de usar uma versão mais nova do que a fornecida pela distribuição.

Em geral, basta descompactar o zip em um local qualquer, atribuir permissão de execução para os scripts shell e configurar a variável de ambiente JAVA\_HOME para indicar o local de instalação correta do JDK nas convenções do JPackage.

Por exemplo, caso se deseje utilizar o Java default do sistema:

$ export JAVA\_HOME=/usr/lib/jvm/java

Ou então, caso se deseje usar especificamente o Java 1.5.0 da Sun, mesmo que o Java default seja outro:

$ export JAVA\_HOME=/usr/lib/jvm/java-1.5.0-sun

Estes comandos podem ser inseridos no script profile do usuário, ou então no próprio script de inicialização da aplicação.

## Instalação manual do Java

Caso o aluno opte (ou seja obrigado) a usar o instalador fornecido pela Sun para o seu Java, seja por política da empresa ou por não estar em Linux, será necessário configurar manualmente o ambiente para a execução do Java.

Na verdade, mesmo que sejam utilizados os pacotes rpm fornecidos pela Sun para instalação em Linux, será necessário seguir este roteiro de configuração manual, pois os pacotes da Sun não configuram corretamente o ambiente do sistema operacional.

Será necessário configurar uma série de variáveis do ambiente de acordo com a sua instalação do Java:

* JAVA\_HOME para o diretório de instalação:
* PATH para incluir o diretório onde estão os comandos java e javac:
* CLASSPATH para incluir pelo menos o diretório corrente:

export JAVA\_HOME=/usr/java/jdk1.5.0\_12

export PATH=$JAVA\_HOME/bin:$PATH

export CLASSPATH=.:$CLASSPATH

Em ambiente Windows, estas configurações podem ser feitas por um arquivo bat ou então nas propriedades do Meu Computador, aba “Avançado”.

## Exercícios

## Laboratório 1.1. Instalação do OpenJDK 6

(Prática Dirigida)

**Objetivo:** Instalar e verificar o OpenJDK 6

Seguindo os passos já apresentados neste capítulo, verifique que o OpenJDK esteja instalado, e caso não esteja, providencie sua instalação usando o repositório local da sala de aula. Não faça download de pacotes e atualizações da Internet, porque você pode prejudicar o andamento das outras salas de aula!

Em seguida, utilize o comando alternatives para confirmar que o OpenJDK 6 seja o Java default, e use as opções -version do java e javac para nova confirmação.

A instalação e verificação do OpenJDK devem ser feitas como root, entretanto os próximos passos devem ser feitos com um usuário comum.

Finalmente, baixe do computador do instrutor o arquivo com os programas-exemplo do curso e os descompacte no seu diretório home. Entre em Cap1/Lab1, compile e execute o programa “hello, world” que está neste diretório, para confirmar que seu ambiente Java esteja funcional antes de passarmos para o próximo capítulo, onde será realizada a instalação do Tomcat:

$ **unzip -q exemplos.zip**

$ **cd Cap1/Lab1**

$ **ls**

Oi.java

$ **javac Oi.java**

$ **ls**

Oi.class Oi.java

$ **java Oi**

Oi do Java!

Note que, no momento da compilação, o argumento é o nome do arquivo-fonte contendo o programa Java. Mas, no momento da execução, o argumento é o nome da classe, e não o nome do arquivo compilado!

O arquivo Oi.class, que é o resultado da compilação de Oi.java, será localizado pela JVM (comando java) de acordo com a configuração do CLASSPATH, que por padrão inclui o diretório corrente.

Laboratório 1.2. Instalação do Sun JDK 5

(Prática Dirigida)

**Objetivo:** Instalar e verificar o Sun JDK 5, depois retornar para o OpenJDK 6

O instrutor irá fornecer os pacotes rpm para o Java da Sun já re-construídos segundo o roteiro do JPackage, e o aluno terá que baixar estes pacotes para seu home.

Em seguida, o aluno deverá assumir a identidade do root e realizar a instalação destes pacotes, junto com as respectivas dependências, usando o comando rpm.

Utilize o comando alternatives e a opção -version para confirmar que agora o java default do ambiente é o Java da Sun.

Retorne ao usuário comum, volte ao exemplo do exercício anterior, e realize sua compilação e execução com o Java da Sun.

Quando tudo estiver funcionado ok, volte para o root e retorne o java e javac default para o OpenJDK 6. E não esqueça de confirmar que no final estamos usando as versões corretas de cada comando.

## Conclusão

Fomos apresentados aos conceitos do Java EE e realizamos a instalação do JDK em nosso ambiente de trabalho, para receber a instalação do Tomcat no próximo capítulo.

Questões de Revisão

* JRE e JVM são sinônimos?

* Porque pode ser necessário ter no mesmo computador o OpenJDK e o Sun JDK instalados lado-a-lado?

# Instalação do Tomcat

Neste capítulo temos um primeiro contato com o servidor Tomcat, sua instalação e sua “cara” para o usuário final.

**Tópicos:**

* Apresentação do Apache Tomcat
* Instalação via pacotes ou manual do Tomcat
* Início e término do servidor Tomcat

## Sobre o Apache Tomcat

O Apache Tomcat é um servidor de aplicações Java EE “incompleto”, pois fornece apenas o Container Web. Ele nasceu pela fusão do antigo projeto JServ da Apache Software Foundation com a RI do então nascente padrão de Servlets do Java EE, que era desenvolvida pela Sun.

O Tomcat foi inicialmente parte do projeto Jakarta, que era um mega-projeto de infra-estrutura para desenvolvimento e execução de aplicações web em Java. Vários softwares livres importantes no mundo Java, como Ant, Log4J e Struts, também foram inicialmente parte do Projeto Jakarta.

Além do container web e do compilador de páginas JSP, o Tomcat fornece serviços JNDI, JMX, JAAS e Datasources, de modo que aplicações Web criadas originalmente para servidores de aplicações mais “parrudos” como o Jboss AS possam rodar sem modificações no Tomcat. Desde que elas não necessitem de outros componentes Java EE ausentes no Tomcat, como EJB e JMS.

O Tomcat é um servidor de alta performance, estabilidade e escalabilidade, fornecendo recursos avançados, como suporte nativo a clustering (desde a versão 5.0). Então, apesar de “incompleto” e por isso não-certificável no padrão Java EE, o Tomcat se tornou popular e respeitado em ambientes de missão-crítica.

A maioria dos servidores de aplicação Java EE, mesmo os proprietários, incorporam o Tomcat ou alguns de seus componentes-chave, pois a licença Apache permite esta apropriação de código livre por produtos proprietários, e não exige a menção disso.

É possível ainda reunir o Tomcat a outros projetos open source, como o ActiveMQ e o Arjuna, para fornecer capacidades Java EE ausentes nele, mas o administrador deve considerar o esforço extra de integração e configuração. Se houver realmente necessidade de um ambiente Java EE completo, porque não usar um servidor certificado como o JBoss AS, Apache Gerônimo ou Glassfish?

### Versões do Tomcat

A versão do Tomcat a ser utilizada depende da versão das especificações de Servlets e JSP a ser adotada. Consulte http://tomcat.apache.org/ para ver a relação versões do Tomcat x especificações de Servlets e JSP. A tabela 1.1 contém um resumo das versões x compatibilidade Java EE do Tomcat.

**Tabela 1.1** – Versões do Tomcat x Java EE

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tomcat | Servlets | JSP | Java EE | Java SE |
| 3.3 | 2.2 | 1.1 | 1.2 | 1.2 |
| 4.1 | 2.3 | 1.2 | 1.3 | 1.3 |
| 5.5 | 2.4 | 2.0 | 1.4 | 5 |
| 6.0 | 2.5 | 2.1 | 5 | 5 |
| 7.0 (svn trunk) | 3.0 (draft) | 2.1 | 6 | 6 |

O Tomcat 7.0 ainda não teve nenhum release, está em estágios iniciais de desenvolvimento. Já a versão 3.0 da API de Servlets ainda está em Draft, e o Java EE 6 está previsto para o final de 2009, podendo “vazar” para 2010.

Todas as versões indicadas na tabela ainda são oficialmente suportadas, de modo que a Apache Foundation tem assumido um compromisso de suporte continuado mais longo do que fornecedores como Sun, Oracle e IBM.

É claro, as séries 3.x, 4.x são suportadas somente em caso de bugs sérios e por poucos membros da comunidade. Já a série 5.5 é ativamente suportada, embora não esteja recebendo novas features. O desenvolvimento hoje foca na série 6.0 e na incipiente 7.0.

Embora este curso seja realizado com a versão 6.0 do Tomcat, a maior parte do seu conteúdo vale para a versão 5.5 e mesmo para a versão 4.1. O Tomcat é um software bastante maduro e estável sob o ponto de vista do administrador, mantendo praticamente as mesmas sintaxes e defaults nos arquivos de configuração há vários releases, e a mesma arquitetura interna (embora as implementações tenham evoluído para usufruir de novos recursos do Java SE). É o desenvolvedor quem é mais afetado pela versão do Tomcat utilizada.

Vale aqui observar que nenhuma versão do Tomcat traz suporte a JSF, embora outro projeto da Apache Software Foundation, o MyFaces, seja utilizado com freqüência para suprir esta funcionalidade.

## Instalação do Tomcat

Para a instalação do Tomcat, seu sistema deve atender aos seguintes pré-requisitos:

|  |  |
| --- | --- |
| Tomcat 5.0.x e anteriores | JDK 1.3 ou superior  O Tomcat necessita do compilador Java fornecido pelo JDK para compilar os servlets gerados pelo processamento de páginas JSP |
| Tomcat 5.5.x | JRE 1.5.0 ou superior  JRE 1.4.2 com biblioteca de compatibilidade  O Tomcat passou a incluir o compilador Java do Eclipse (ECJ), de modo que basta um JRE |
| Tomcat 6.0.x | JRE 1.5.0 ou superior  O ECJ continua incluso no Tomcat |

Então, ao contrário de outros servidores de aplicação Java EE, que necessitam de um JDK completo para funcionar, o Tomcat necessita apenas de um JRE. Entretanto, neste curso usaremos um JDK para que o aluno possa compilar e experimentar com os programas-exemplo.

### Instalação via JPackage

Se sua distribuição contém pacotes para o Tomcat, ele pode ser instalado pelo gerenciador de pacotes da forma normal para componentes da distribuição. No caso do Fedora, a instalação do Tomcat seria realizada pelo comando:

# yum -y install tomcat tomcat6-admin-webapps tomcat6-docs-webapp tomcat6-javadoc tomcat6-jsp-2.1-api tomcat6-webapps

Note que componentes opcionais do Tomcat, como a documentação e as aplicações de exemplo, são instaladas como pacotes separados.

Além disso, várias bibliotecas utilizadas ou fornecidas pelo próprio Tomcat, como o commons-logging e a API de Servlets, são fornecidas em pacotes separados, que serão automaticamente baixados e instalados pelo yum.

### Iniciando e finalizando o Tomcat como um serviço do Unix

Se você utilizou os pacotes JPackage para a instalação do Tomcat pode inicia-lo e finaliza-lo utilizando o comando service (presente nas distribuições derivadas do Red Hat) ou então chamando diretamente os scriots em /etc/init.d, por exemplo:

# **service tomcat6 start**

Starting tomcat6: [ OK ]

E para parar:

# **service tomcat6 stop**

Stopping tomcat6: [ OK ]

Acompanhe a seção “Inicialização e término manual do Tomcat” mais adiante para algumas sugestões sobre como verificar se seu servidor Tomcat está realmente no ar, e a seção seguinte sobre como testar a sua instalação do Tomcat. A maior parte do que vale para a instalação manual também vale para a instalação via JPackage, desde que se utilize como diretório de instalação do Tomcat a pasta /usr/share/tomcat6 e o usuário “tomcat”.

Entretanto, a versão JPackage do Tomcat não fornece os scripts startup e shutdown, utilizados para a inicialização manual do servidor. Em seu lugar é fornecido o comando /usr/sbin/tomcat6 que pode ser executado pelo usuário “tomcat” para obter efeito semelhante.

### Instalação Manual do Tomcat

Visite http://tomcat.apache.org e siga o link para download da versão desejada. No caso da versão 6.0, é necessário baixar a distribuição Core. Os downloads em formato zip e tar.gz trazem o mesmo conteúdo, e ambos podem ser executados tanto em Linux/Unix quanto em Windows.

A versão em formato “Windows executable” cria atalhos no menu iniciar e configura um o Tomcat para execução como serviço do Windows. Fora isto, ela é idêntica às versões em zip e tar.gz, que também podem ser configuradas para rodar como serviço no Windows.

Nenhuma das versões fornece scripts de inicialização System V (/etc/init.d) para Linux e Unix. Então o administrador terá que criar seus próprios scripts, coisa que não é necessária na versão JPackage.

Descompacte o zip do Tomcat no diretório desejado, preservando os subdiretórios contidos no zip. Uma firma fácil de se fazer isso é usando o file-roller ou então o comando unzip presentes em qualquer distribuição do Linux.

Em seguida, entre na pasta bin e dê permissão de execução para todos os shell scripts:

$ unzip apache-tomcat-6.0.20.zip

$ cd apache-tomcat-\*/bin

$ chmod a+x \*.sh

O Tomcat espera que esteja definida a variável de ambiente JAVA\_HOME, indicando o diretório de instalação do JRE (ou JDK) desejado.

Espera-se que um ambiente Java configurado para uso pela linha de comando tenha definido esta variável, como vimos no Capítulo 1. Mas o JPackage não a define, pois espera que os scripts de inicialização das aplicações utilizem os scripts do JPackage e/ou o alternatives para determinar a localização do Java Default ou de uma versão específica.

Então, caso você deseje rodar um Tomcat instalado manualmente em um JDK instalado via JPackage, é necessário configurar o JAVA\_HOME nos scripts de início do Tomcat. Para tal, edite o arquivo conf/catalina.sh, e no início do arquivo, antes da linha que “limpa” o CLASSPATH, acrescente:

export JAVA\_HOME=/usr/lib/jvm/java-1.6.0

Esta linha faz com que seja utilizado qualquer que seja o Java padrão para a versão 1.6.0, qualquer que seja ele. No nosso ambiente de sala de aula, será um OpenJDK 6.

### Início e término manual do Tomcat

Para iniciar o servidor Tomcat, entre na pasta bin da sua instalação, por exemplo ~/apache-tomcat-\*/bin. Execute o script startup:

$ cd ~/apache-tomcat-\*/bin

$ ./statup.sh

O script startup coloca o processo java (a JVM que roda o Tomcat) em background, liberando imediatamente o prompt de comandos.

Caso você deseje acompanhar as mensagens de inicialização do servidor, para ter certeza de que ele está tendo um início “limpo”, verifique o arquivo de log catalina.out:

$ tail -f ../logs/catalina.out

Também é uma boa idéia verificar a presença do processo java e via o comando netstat se as portas 8080 e 8085 estão abertas por este processo.

$ ps ax | grep java

$ netstat -anp | grep java

Para terminar o Tomcat, é possível enviar um sinal INT via kill, ou então usar o script shutdown na pasta bin do Tomcat

$ cd ~/apache-tomcat-\*/bin

$ ./shutdown.sh

Evite finalizar o processo tomcat de forma abrupta, por exemplo utilizando um sinal KILL (como o gerado por kill -9). Isto poderá deixar arquivos de trabalho corrompidos e prejudicar a próxima inicialização do Tomcat.

Se o Tomcat estiver travado, ou por algum outro motivo for finalizado sem um shutdown, por exemplo por falha na energia elétrica, será necessário limpar os diretórios de trabalho:

$ cd ~/apache-tomcat-\*/

$ rm -rf {temp,work}

## Testando o Tomcat

Para testar o seu servidor Tomcat, acesse a URL http://127.0.0.1:8080. O resultado será uma página de boas-vindas, contendo o logotipo do “tigre asiático”[[5]](#footnote-5) e links para programas exemplo e documentação.

Não será possível entrar nas aplicações administrativas, como o Manager, porque ainda não configuramos o acesso a elas. Mas há vários exemplos simples de programação de Servlets e JSP.

## Se Algo Der Errado

* Verifique os logs do Tomcat, em especial logs/catalina.out. Procure por mensagens de erro e exceções Java relacionadas com permissões de acesso a arquivos, falta de memória ou disponibilidade de endereços e portas TCP;
* Verifique se o comando java pode ser executado diretamente pela linha de comando e se ele reporta a versão correta com o argumento -version;
* Verifique se as portas 8080, 8009 e 8085 estavam livres antes do início do Tomcat utilizando o comando netstat;
* Verifique se o nome de host do computador é resolvido para um endereço IP utilizando o comando ping;
* Verifique se a estrutura de diretórios do Tomcat foi preservada depois da descompactação do arquivo zip; Ou no caso da instalação via JPackage verifique se todos os pacotes necessários foram instalados (você pode ter instalado apenas algumas bibliotecas necessárias ao desenvolvimento de aplicações Java EE e não ter instalado o servidor Tomcat propriamente dito);
* Se tudo o mais falhar, encerre todos os processos “java” ativos e reinstale o Tomcat do zero.

## Documentação do Tomcat

A documentação do Tomcat é formada por um manual de Configuração e Administração fornecido em formato HTML e pelos JavaDocs das próprias classes internas do servidor.

Os manuais estão inclusos na distribuição Core, ou então no pacote tomcat6-docs-webapp. Já os JavaDocs devem ser baixados e instalados separadamente, e geralmente não são necessários para o administrador ou desenvolvedor.

O Tomcat 6.0, ao contrário de versões anteriores, não traz mais os JavaDoc da API de Servlets e de JSP. Para obter esta documentação, é necessário baixar as especificações relevantes do JCP em www.jcp.org.

## Exercícios

Laboratório 1.1. Instalação e teste do Tomcat via JPackage

(Prática Dirigida)

**Objetivo:** Instalar, iniciar e testar o Tomcat à partir dos pacotes JPackage fornecidos pela distribuição Fedora do Linux.

Para esta instalação, utilize o repositório yum e os pacotes rpm fornecidos pelo instrutor na rede local da sala de aula. Não realize download diretamente da Internet, para não sobrecarregar o Link internet e atrapalhar o andamento de outras turmas em outras salas de aula.

Antes de instalar o Tomcat propriamente dito, confirme que você está utilizando o OpenJDK 6 em vez do Sun JDK 5. Então, utilizando o “root”, instale os pacotes do Tomcat conforme apresentado anteriormente neste capítulo, utilizando o comando yum.

Ainda utilizando o “root”, inicie o Tomcat utilizando o comando service. Utilize os comandos ps e netstat para confirmar que o Tomcat realmente está no ar e aceitando conexões TCP na porta 8080, e verifique o arquivo catalina.out não contém nenhuma mensagem de erro.

Em seguida, acesse a URL http://127.0.0.1:8080 para ver a página inicial do Tomcat. Role a janela do navegador, em busca do link para os exemplos. Dentro dos “JSP Examples”, um exemplo interessante é o “Shuffle Example”.

Finalmente, encerre o Tomcat com o comando service. Utilize novamente os comandos ps e netstat para confirmar que não existe mais processo java rodando o Tomcat e que a porta 8080 esteja livre. Volte ao arquivo de log catalina.out para se certificar de que o shutdown do Tomcat transcorreu sem erros.

Laboratório 1.2. Instalação e teste manuais do Tomcat

(Prática Dirigida)

**Objetivo:** Instalar e testar uma instalação do Tomcat à partir dos arquivos zip fornecidos em tomcat.apache.org.

O instrutor irá fornecer os arquivos zip para a instalação do Tomcat, que o aluno deverá baixar para sua estação de trabalho e descompactar em seu diretório home, sem utilizar o usuário “root”.

Em seguida, utilize as instruções fornecidas anteriormente nesta apostila para iniciar o Tomcat e acesse sua página inicial. Execute um exemplo, e verifique que o Tomcat está realmente utilizando os comandos ps e netstat, tal qual foi feito no exercício anterior. E não esqueça de verificar o log catalina.out.

Cuidado para não trocar os arquivos da instalação manual com os arquivos da instalação realizada via JPackage. Evite também fazer parte do laboratório como “root”, de modo a evitar problemas com permissões de arquivos.

Quando estiver satisfeito, finalize a instância do Tomcat que você iniciou manualmente. Nos próximos laboratórios, será utilizada a instalação realizada por meio do JPackage.

Laboratório 1.3. Acesso off-line aos manuais do Tomcat

(Laboratório)

**Objetivo:** Acessar a documentação local do Tomcat, com o servidor fora do ar, para poder consultar em caso de problemas de inicialização do servidor.

Utilizando seu usuário não-administrador, e sem que haja nenhuma instância do Tomcat em execução, procure pelo arquivo RELEASE-NOTES.txt na instalação do Tomcat realizada via JPackage, e leia seu conteúdo.

O arquivo poderá ser encontrado em /usr/share/tomcat6/webapps/docs, junto com vários documentos HTML que formam o manual de Configuração e Administração do Tomcat. Aproveite e abra o arquivo index.xml e navegue um pouco neste manual.

Outra cópia deste arquivo poderá ser encontrada em /usr/share/doc/tomcat6-6.0.18/ que é o diretório padrão para instalação de documentação de programas no Linux. Note que neste diretório não estão os manuais em HTML.

## Conclusão

Agora que temos duas instalações funcionais do Tomcat, podemos estudar melhor sua arquitetura interna e como ele pode ser administrado.

Questões de Revisão

* Porque o Tomcat 5.0 em diante não necessita mais de um JDK?

* Verdadeiro ou Falso: para que seja possível acessar aplicações web escritas em Java utilizando o Tomcat, é necessário rodar também um servidor Apache HTTPd, tal qual ocorre com aplicações web escritas em PHP ou Perl.

# Arquitetura do Tomcat

Agora vamos olhar para o Tomcat “por dentro”, como um administrador deve fazer. Este capítulo é um overview geral de onde o administrador irá atuar no servidor Tomcat, sendo que os detalhes serão apresentados ao longo desta apostila.

**Tópicos:**

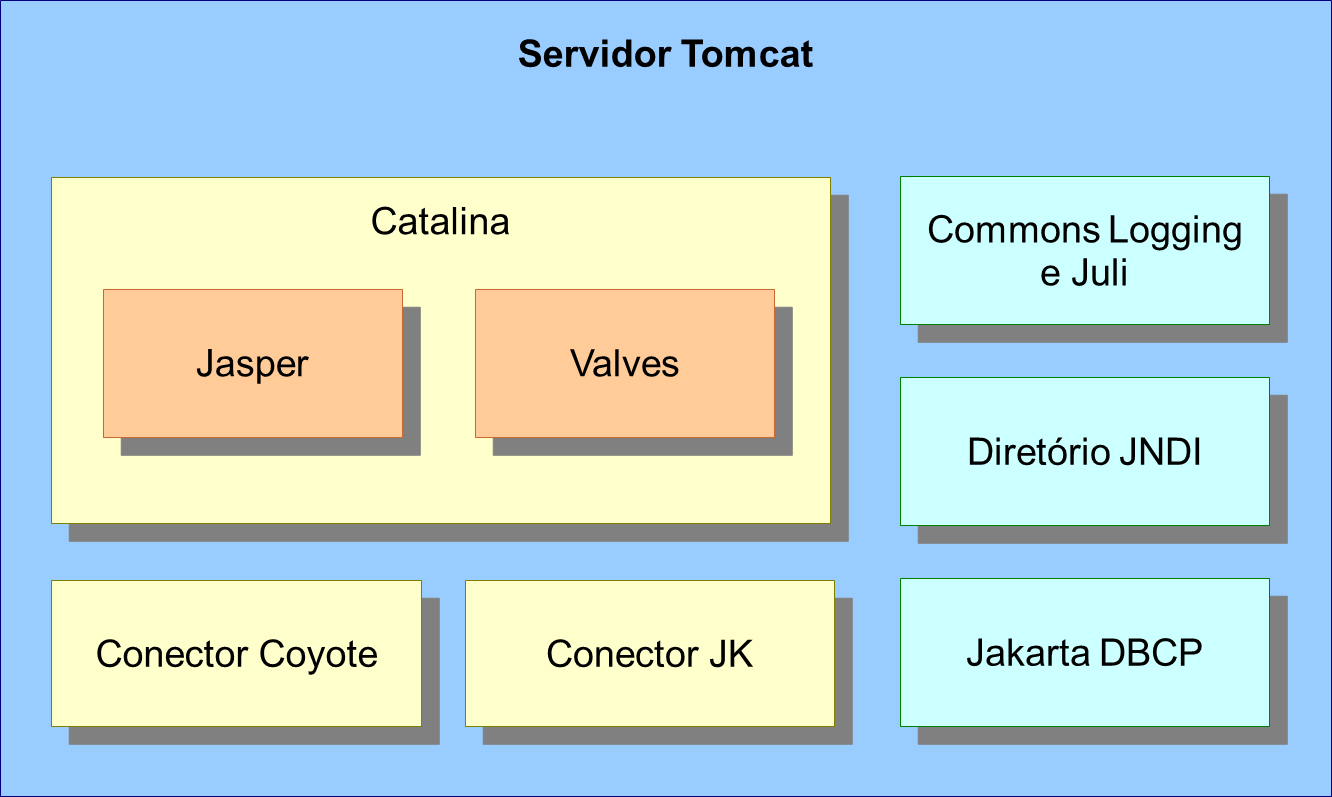
* Arquitetura do Tomcat
* Aplicações administrativas

## Componentes do Tomcat

A figura 1.1 apresenta os principais componentes do Tomcat, que são:

* Catalina – é o container web em si, que executa Servlets, Filtros e outros componentes de aplicação previstos pela API de Servlets. Este é o coração do Tomcat;
* Jasper – é o compilador de páginas JSP, implementado como um Servlet. Várias outras funcionalidades do Tomcat são implementadas como Servlets, por exemplo o acesso a arquivos estáticos dentro de uma aplicação ou o suporte a aplicações CGI e Server-Side Includes;
* Valves – são componentes utilitários que interceptam o processamento normal de requisições HTTP e implementam funções diversas como logs de acesso, auditoria, segurança ou cluster;
* Conectores Coyote e JK – recebem requisições do mundo externo, respectivamente, pelos protocolos HTTP e AJP;
* Commons Logging – é uma abstração de logging que permite plugar sob o Tomcat frameworks como o Log4J de acordo com a conveniência e preferências do administrador;
* Juli – É uma extensão para a API de Logging do JDK que visa compensar algumas de suas limitações em relações a frameworks mais sofisticados como o Log4J;
* Diretório JNDI – implementa o serviço de nomes interno do servidor de aplicação, utilizado no desenvolvimento Java EE para acess a bancos de dados, servidores de e-mail, filas de mensagens JMS e EJBs. Apesar do Tomcat não fornecer a maioria destas funcionalidades, ele está preparado para ser configurado com fábricas de conexões que permitem a ele acessar estes recursos em outros servidores de aplicações;
* Jakarta DBCP – permite ao Tomcat suportar Datasources Java EE e assim ser, no que diz respeito ao acesso a bancos de dados, tão rico em recursos quando um servidor Java EE completo, exceto pela falta de suporte a Entity Beans e JPA, que são recursos de EJB.

A figura deixa de fora, propositadamente, os componentes relacionados com o cluster Tomcat, que serão vistos nos capítulos finais desta apostila.



**Figura 1.3** – principais componentes do Tomcat

## Layout de Instalação do Tomcat

Embora o layout de diretórios da instalação do Tomcat seja bastante diferente entre a versão empacotada no padrão JPackage e a distribuição Core de apache.org, temos os mesmos arquivos, com o mesmo papel nos dois layouts. Melhor ainda, o JPackage emula o layout original utilizando links simbólicos, de modo que um administrador pode optar por seguir a visão original do Tomcat ou então a visão LSB do JPackage.

O layout original está em /usr/share/tomcat6 e o papel de cada um, além da sua localização “real” no LSB são indicados na tabela 1.1:

**Tabela 1.1** – Layout de instalação do Tomcat

|  |  |  |
| --- | --- | --- |
| **Original** | **JPackage** | **Conteúdo** |
| bin | bin | Scripts de início e término do Tomcat, startup.sh e shutdown.sh (ausentes no JPackage, substituídos pelo script /usr/sbin/tomcat)  Classes Java de inicialização do Tomcat (bootstrap e juli) |
| conf | /etc/tomcat6 | Arquivos de configuração do próprio servidor e de contextos (aplicações) web Java EE |
| lib | /usr/share/java/tomcat6 | Componentes do próprio Tomcat e APIs padrão Java EE implementadas por ele |
| logs | /var/log/tomcat6 | Arquivos de log: catalina.out e outros |
| temp | /var/cache/tomcat6/temp | Arquivos temporários diversos, por exempo upload de arquivos em formulários HTTP |
| webapps | /var/lib/tomcat6/webapps | Aplicações web Java EE instaladas por auto-deploy |
| work | /var/cache/tomcat6/work | Páginas JSP compiladas sob demanda |

Observe que os diretórios de arquivos do Tomcat 6 (na instalação JPackage) recebem o nome tomcat6 para permitir a instalação lado-a-lado de outras versões, como o Tomcat 5.0 e 5.5.

Versões anteriores do Tomcat subdividiam a pasta lib em server, common e shared. O objetivo era separar bibliotecas por classloader, de modo a isolar classes que poderiam ou não serem utilizadas por aplicações. Esta separação agora é configurada por meio do arquivo de configuração catalina.properties.

### Arquivos de Configuração do Tomcat

Dentro da pasta conf, temos os arquivos de configuração do Tomcat, que são descritos a seguir. O objetivo agora não é ser exaustivos nem substituir o manual de Administração e Configuração do Tomcat, mas apenas fornecer ao aluno uma visão rápida de onde procurar por possíveis configurações:

* catalina.policy – permissões para execução de código dentro da JVM, se o Tomcat for iniciado com um security manager ativado;
* catalina.properties – configurações gerais do container web, basicamente as configurações dos classloaders que permitem isolar aplicações entre si e em relaão ao proprio servidor Tomcat;
* context.xml – configurações de contexto default (internas ao Tomcat) para aplicações web instaladas no Tomcat;
* logging.properties – configurações de logging, por exemplo nomes dos arquivos de log e regras de retenção e rotação;
* server.xml – é o arquivo principal de configuração do Tomcat, que será examinado em detalhes mais adiante;
* tomcat6.conf (apenas JPackage) – fornece argumentos e opções de inicialização da JVM que roda o Tomcat;
* tomcat-users.xml – definições de usuários e roles para aplicações administrativas ou para uso em desenvolvimento;
* web.xml – configurações default de Servlets, filtros e listeners para aplicações web instaladas no Tomcat; cuidado ao modificar este arquivo, pois erros podem afetar funções básicas do servidor como a capacidade de executar páginas JSP.

Ainda podem haver subdiretórios na pasta conf, nomeados conforme a hierarquia Engine/Host, definida no arquivo server.xml e que será apresentada mais adiante. Os arquivos contidos nestes diretórios se limitam a configuração de contextos, ou seja, superpõem definições nos arquivo context.xml e server.xml da pasta conf.

## Sintaxe do server.xml

O principal arquivo de configuração do Tomcat é um documento XML, utilizado para construir os vários objetos Java que formam o Tomcat durante sua inicialização.

Sua estrutura é uma hierarquia descrita a seguir. Como nos outros tópicos desta seção, a idéia é apenas apresentar um overview geral do servidor, mais detalhes sobre cada elemento serão apresentados em capítulos futuros, conforme a necessidade.

* <Server>   
  representa o Tomcat como um todo, e é configurado com a porta TCP e comando para o shutdown do servidor;
  + <Listener>  
    utilizado para inicializar algum componente do Tomcat no start ou shutdown do servidor;
  + <GlobalNamingResources>  
    configura entradas de ambiente e recursos JNDI no espaço de nomes global (elementos no espaço de nomes local são inicializados dentro de elementos <Context>);
  + <Service>   
    representa um serviço oferecido pelo servidor. A única possibilidade disponível é um servidor web, embora a arquitetura do Tomcat permita a implementação de serviços adicionais;
    - <Executor>  
      pool de threads que pode ser compartilhado entre vários conectores;
    - <Conector>   
      recebe conexões de clientes remotos, que podem ser navegadores ou servidores web;
    - <Engine>   
      é o container web em si;
      * <Host>  
        Um host virtual, baseado em nome ou IP;
        + <Context>   
          Uma aplicação web instalada no Tomcat.

O elemento <Server> é especial, pois a sintaxe XML exige que existe um elemento englobando o arquivo inteiro. Já os elementos <GlobalNamingResource>, <Service>, <Engine>, <Host> e <Container> são chamados de containers[[6]](#footnote-6) e podem conter subelementos que serão descritos mais adiante.

Destes elementos, <Engine>, <Host> e <Context> formam uma hierarquia de configuração referenciada diretamente na instalação de aplicações e configurações de logging. Isto significa que muitas configurações podem afetar o servidor como um todo, apenas um dos vários hosts virtuais hospedados no mesmo servidor Tomcat, ou apenas uma única aplicação.

Os elementos a seguir podem ser inseridos como subelementos (filhos) de qualquer elemento container, e normalmente são utilizados para atender a demandas particulares de aplicações instaladas:

* <Realm>  
  fornece configurações para autenticação que pode ou não utilizar o JAAS[[7]](#footnote-7);
* <Resource>  
  define a conexão a uma fonte de dados ou serviço externo ao Tomcat, por exemplo um banco de dados relacional, servidor de e-mail, servidor de mensagens JMS ou container EJB;
* <Valve>  
  modifica o processamento de requisições, por exemplo para gerar logs de acesso ou depuração da requisição HTTP.

O manual do Tomcat inclui uma referência detalhada para todos esses elementos e também para alguns que não foram citados aqui, por que raramente são modificados pelo administrador.

O arquivo server.xml padrão de instalação do Tomcat fornece vários exemplos comentados de definições que poderiam ser inseridas, especialmente para <Conector>, <Realm>, <Valve> e <Resource>. Por isso recomendamos que o arquivo original seja salvo como referência, antes que sejam feitas modificações.

## O Tomcat Manager

O Tomcat 6.0 fornece uma única aplicação administrativa, o Manager. Ele não é um console de gerenciamento abrangente, como os fornecidos por alguns servidores de aplicação Java EE. Sua funcionalidade se restringe à monitoração do servidor e à instalação de aplicações. Ou seja, na maior parte do tempo o administrador estará editando diretamente os arquivos de configuração do Tomcat e reiniciando o servidor.

Versões anteriores do Tomcat (até a 5.5) ofereciam também uma segunda aplicação, chamada Admin. O Admin era basicamente um editor estruturado com interface web para o arquivo server.xml. Era um aplicativo bastante útil, que permitia fazer algumas configurações sem reiniciar o Tomcat, entretanto ele nunca foi reescrito para o Tomcat 6.0. Parte da funcionalidade do Admin pode ser obtida pela administração via JMX, que será vista no final desta apostila.

O Manager oferece duas interfaces web: uma mais simples, acessada pelo contexto manager (http://127.0.0.1/manager), e outra melhor acabada visualmente, acessada pela pasta html do mesmo contexto (http://127.0.0.1/manager/html).

A primeira interface existe para facilitar a escrita de scripts e programas administrativos, especialmente IDEs Java que necessitam de instalar aplicações para teste e depuração no Tomcat.

Já a segunda interface existe para operação por um administrador humano. Ambas as interfaces fornecem mais ou menos a mensa funcionalidade, com poucas exceções que serão vistas ao longo deste curso.

### Ativação do Manager

O Manager é parte da distribuição Core do Tomcat (ou do pacote tomcat6-admin-webapps) mas vem configurado com restrições de acesso, e por motivo de segurança a instalação padrão do Tomcat não define nenhum usuário com acesso de administrador.

Então é necessário editar o arquivo tomcat-users.xml para definir um usuário com o role “manager”, antes que seja possível acessar qualquer das suas funcionalidades. A listagem 1.1 apresenta um exemplo de como fazer esta edição.

**Listagem 1.1** – Definindo um usuário administrador em tomcat-users.xml

1. <?xml version='1.0' encoding='utf-8'?>
2. <tomcat-users>
3. **<user username="admin" password="senha"**
4. **roles="manager"/>**

</tomcat-users>

É possível instalar e testar aplicações no Tomcat com o Manager desabilitado (ou mesmo desinstalado) e também é possível monitorar o servidor via JMX, mas na maioria dos casos será mais cômodo usar o Manager.

Após esta modificação, assim como em qualquer outro arquivo de configuração do Tomcat, é necessário reiniciar o servidor.

## Exercícios

Laboratório 1.1. Conhecendo os arquivos de configuração do Tomcat

(Exploração)

**Objetivo:** Localizar algumas configurações importantes dentro do server.xml

Este é um exercício exploratório. Abra o arquivo de configuração server.xml do Tomcat com um editor de texto e procure os elementos que definem:

* A porta 8080 para conexões HTTP;
* O diretório webapps para a instalação de programas via auto-deploy;
* O arquivo tomcat-users.xml para cadastro de usuários administradores.

Em seguida, abra o arquivo web.xml na pasta conf do Tomcat e localise o servlet compilador de páginas JSP, o Jasper.

Laboratório 1.2. Ativação do Manager

(Prática Dirigida)

**Objetivo:** Conseguir acesso ao aplicativo Manager para administração do Tomcat

Usando as instruções já apresentadas nesta apostila, edite o arquivo tomcat-users.xml para definir um usuário administrador, cujo login seja “admin” e a senha seja “tomcat”.

Em seguira, reinicie o Tomcat e tente entrar no link para a aplicação Manager. Localize na página do Manager algumas informações úteis como:

* A versão da JVM que roda o Tomcat;
* Seu diretório de instalação;
* O thread que exibe a própria lista de aplicações instaladas dentro do Tomcat;
* A quantidade de usuários logados na aplicação Manager.

Laboratório 1.3. Monitoração via script

(Desafio)

**Objetivo:** Construir um script de monitoração do Tomcat

Utilize seus conhecimentos de shell script para criar um script que monitore continamente o servidor Tomcat, e exiba informações de memória alocada e livre, no estilo free ou vmstat do Linux.

Será necessário usar um comando como curl ou wget para acessar a página de status do Manager, e filtrar a saída com grep, sed ou awk. OU então, escrever o script usando Perl, PHP ou outra linguagem disponível no Linux.

## Conclusão

Agora que temos acesso aos arquivos e programas de administração do servidor Tomcat, o próximo passo é instalar aplicações e em seguida configurar o Tomcat de acordo com as demandas destas aplicações.

Questões de Revisão

* Qual componente do Tomcat é responsável pela sua principal finalidade, que é rodar Servlets Java EE?

* É possível usar o Manager do Tomcat para alterar parâmetros de configuração, por exemplo a porta utilizada para aceitar conexões HTTP?

# Deployment de Aplicações

Neste capítulo, aprendemos como instalar e configurar aplicações web para execução sob o servidor de aplicações Tomcat

**Tópicos:**

* Pacotes WAR
* Deployment de aplicações automático e manual
* Atualização de aplicações
* Desligando o auto-deploy

## Pacotes WAR e Deployment Descriptors

Uma aplicação web em Java deve ser empacotada em um formato chamado WAR, que corresponde a um arquivo \*.war. Um WAR do Java EE é também um JAR do Java SE, que por sua vez é um arquivo ZIP padrão.

Mais especificamente, um pacote JAR é um arquivo ZIP que contém uma pasta chamada META-INF, na qual podem ser colocados arquivos de configuração para a JVM. Irá existir pelo menos um arquivo chamado MANIFEST.MF.

Já um pacote WAR é um pacote JAR no qual é adicionado o diretório WEB-INF, contendo arquivos de configuração para o container web. Deverá existir pelo menos o descritor de deployment chamado web.xml.

Os nomes da pasta e arquivo tem que ser exatamente como indicado, pois maiúsculas e minúsculas fazem diferença[[8]](#footnote-8).

Mas, diferente de um pacote JAR, que contém apenas classes Java e recursos (arquivos de dados utilizados por estas classes, por exemplo mensagens para internacionalização) um pacote WAR contém páginas HTML estáticas, imagens PNG, páginas JSP dinâmicas e tudo o mais que forma uma aplicação web.

Na verdade, todas as classes Java dentro de um pacote WAR devem estar dentro da pasta WEB-INF, não importa se são classes Java simples, Java Beans ou Servlets propriamente ditos. Mais especificamente, dentro de WEB-INF/classes para classes isoladas (arquivos \*.class) ou então dentro de WEB-INF/lib para classes agrupadas em bibliotecas, que são pacotes JAR (arquivos \*.jar).

Ou seja, um pacote WAR é ele mesmo um JAR mas também pode conter vários outros pacotes JAR dentro dele.

Apenas pacotes WAR podem ser instalados, ou melhor, deployados pelo Container Web. Podemos dizer que o pacote WAR é o “arquivo executável” de uma aplicação web do Java EE.

## Um exemplo mínimo

Para contextualizar melhor, segue um exemplo de aplicação web Java EE, que corresponde ao primeiro laboratório deste capítulo. O exemplo ilustra várias tecnologias disponíveis para o programador de Servlets, desde os Servles básicos até as expressões EL do Java EE 1.4.

* hoje.war
* bean.jsp
* el.jsp
* hoje.jsp
* index.jsp
* WEB-INF
* classes
* exemplo
* HojeBean.class
* HojeServlet.class
* web.xml

Muitas vezes, o desenvolvedor colocar osarquivos fonte na mesma estrutura do pacote WAR, então junto aos arquivos \*.class estariam os arquivos \*.java correspondentes.

Mas é claro o container web não necessita dos fontes Java para executar as classes compiladas. Por isso a maioria dos desenvolvedores irá preferir colocar os fontes Java em uma estrutura diferente do pacote WAR, e montar o pacote reunindo o resultado da compilação das classes, as páginas e arquivos de configuração.

### Compilando o exemplo

A API de Servlets não é parte do Java EE, e sim é fornecida com o servidor de aplicações, no nosso caso o Tomcat. Então, para compilar as classes do exemplo, é necessário configurar o CLASSPATH para incluir o arquivo JAR que fornece a API de servlets.

Na instalação manual do Tomcat, o arquivo é lib/servlet-api.jar; já na instalação via JPackage, em vez de referenciarmos o arquivo dentro da instalação do Tomcat (em /usr/share/tomcat6) podemos referenciar a biblioteca compartilhada em /usr/share/java/servlet.jar.

E, se houverem várias versões da API de Servlets disponíveis no seu sistema, por exemplo porque estão instalados tanto o Tomcat 5.5 quanto o 6.0, é possível configurar o default utilizando o alternatives, tal qual foi feito no Capítulo 2 para o JDK.

Considerando que os fontes da aplicação estejam organizados de modo a espelhar a estrutura do pacote WAR, os seguintes comandos compilam as classes e colocam os bytecodes resultantes no local correto:

$ export CLASSPATH=$CLASSPATH:/usr/share/java/servlet.jar

$ cd ~/Cap1/Lab1

$ cd WEB-INF/classes

$ javac exemplo/\*.java

### Empacotando o exemplo

Embora seja possível utilizar qualquer ferramenta capaz de ler e escrever arquivos ZIP, por exemplo o 7-Zip[[9]](#footnote-9) para Windows ou o file-roller do Linux, o JDK já fornece o comando jar para a manipulação de pacotes JAR.

A sintaxe do comando jar lembra o comando tar padrão do Unix, em vez de seguir a sintaxe do Info-Zip (comandos zip e unzip do Linux) ou do PkZip original. Então, para criar o pacote WAR correspondente ao nosso exemplo, basta usar o comando:

$ $ cd ~/Cap1/

$ jar cvf hoje.war -C Lab1 .

Note que o último argumento do comando é um ponto final ou “.”, indicando o diretório corrente.

Observe as opções “cvf” no estilo tar, e o uso da opção -C para usar o diretório Lab1 (que contém a estrutura do pacote WAR desejado) sem porém incluir o próprio diretório “Lab1” na estrutura do arquivo ZIP resultante.

O próprio comando jar pode ser utilizado para verificar se o pacote WAR está com o conteúdo correto:

$ **jar tvf hoje.war**

0 Wed Mar 29 00:12:40 BRT 2006 META-INF/

44 Wed Mar 29 00:12:40 BRT 2006 META-INF/MANIFEST.MF

0 Tue Mar 28 07:23:10 BRT 2006 ./

423 Tue Mar 28 06:49:58 BRT 2006 index.jsp

166 Tue Mar 28 06:53:04 BRT 2006 el.jsp

289 Tue Mar 28 06:53:26 BRT 2006 hoje.jsp

0 Tue Mar 28 06:49:16 BRT 2006 WEB-INF/

0 Tue Mar 28 06:44:40 BRT 2006 WEB-INF/classes/

...

A saída do comando foi truncada para não enxer a página da apostila, mas você deve observar que:

* Existem as pastas META-INF e WEB-INF na raiz do pacote, e que ambas tem seus nomes inteiramente em letras maiúsculas;
* A página inicial da aplicação, index.jsp, está na raiz do pacote, e não em nenhum subdiretório;
* As classes compiladas estão em subdiretórios de WEB-INF/classes, de acordo com a estrutura de pacotes[[10]](#footnote-10) Java da aplicação.

A verificação também poderia ter sido feita usando o comando:

$ unzip -t hoje.war

Ou qualquer outra ferramenta que reconheça arquivos ZIP.

## Deployment de Aplicações Web

O Java EE usa o termo deployment para o processo de instalação de uma aplicação ou componente dentro de um servidor de aplicações. Este processo envolve garantir que todas as configurações e recursos requeridos pela aplicação estejam disponíveis no servidor onde ela é instalada, e estas “garantias” serão os temas dos próximos capíulos.

O padrão Java EE não define como exatamente é realizado o deployment, mas indica que deve ser fornecido um dos pacotes padrão, por exemplo o WAR (que é o único pacote suportado para deployment pelo Tomcat).

Há várias formas de se realizar o deployment com o Tomcat:

* Utilizando o auto-deploy;
* Utilizando o Manager;
* Editando diretamente os arquivos de configuração.

Se for usado o auto-deploy ou a edição direta dos arquivos de configuração, é possível ainda fazer o deploy de um pacote aberto, que é uma grande conveniência para o desenvolvedor mas também pode induzir a erros de conformidade com o padrão Java EE. Um pacote aberto (ou pacote explodido) é nada mais do que um subdiretório que espelha a estrutura do pacote WAR.

Mas, se for utilizado o Manager, é possível fazer deployment remoto de aplicações.

## Auto-deploy

A maneira mais fácil de fazer a instalação de uma aplicação no Tomcat é copiar seu pacote WAR (seja aberto/explodido ou fechado/compactado) para a pasta webapps. Este é o chamado auto-deploy.

Se for usado um pacote fechado, ele deve ser um arquivo com a extensão \*.war, por exemplo hoje.war mas se for um pacote aberto, o nome do diretório não deve conter a extensão, por exemplo apenas hoje.

Feito a cópia, os logs do Tomcat deverão indicar que o novo pacote foi detectado e instalado, e o novo pacote deverá então ser automaticamente listado como uma nova aplicação no Manager.

Cuidado, pois a cópia de arquivos não é uma operação atômica em nenhum sistema operacional, então é possível que o Tomcat tente processar o pacote antes da cópia ter terminado, resultando em um deployment mal-sucedido.

Para desinstalar a aplicação, basta remover o arquivo ou subdiretório de dentro da pasta webapps.

### Permissões de Arquivos

Se você está usando uma instalação manual do Tomcat, realizada à partir do ZIP “core” fornecido em tomcat.apache.org, não haverão dificuldades com o auto-deploy, porque você é o dono dos arquivos do Tomcat descompactados no seu diretório home.

Mas, se você seguiu as melhores práticas de administração de sistemas Linux e usou os pacotes JPackage, sejam eles baixados de www.jpackage.org ou incorporados à sua distribuição, seu usuário pessoal não terá acesso aos arquivos do Tomcat, incluindo aí a pasta webapps (ou melhor, /var/lib/tomcat6/webapps, ou se preferir /usr/share/tomcat6/webapps).

Observando as permissões da pasta webapps, você pode verificar que o grupo “tomcat” tem permissão de escrita:

$ **ls -ld /var/lib/tomcat6/webapps/**

drwxrwxr-x. 9 root tomcat 4096 Out 1 00:43 /var/lib/tomcat6/webapps/

Então para que seu usuário possa faze o auto-deploy de aplicações basta que ele seja membro do grupo “tomcat”. Uma forma de se fazer isso é usar, como root, o comando usermod:

# usermod -a -G tomcat aluno

Confirme a configuração usando o comando id:

# **id**

uid=500(aluno) gid=500(aluno) grupos=91(tomcat),500(aluno)

Note que um usuário logado não assume imediatamente novas atribuições de grupo, então será necessário fazer logoff e novo login, ou então usar o comando su aluno para obter um novo shell de login.

### Hot-redepoy (atualização à quente)

É possível sobreescrever um pacote dentro da pasta webapps, e assim atualizar a aplicação em execução no Tomcat. Não é necessário, em muitos casos, reiniciar o servidor: o usuário já terá acesso às modificações no seu próximo acesso!

Na verdade, o Tomcat monitora o pacote dentro da pasta webapps, e ao perceber que a data de modificação está mais recente do que a data de deployment do pacote, ele automaticamente realiza a desinstalação (undeploy) e reinstalação da aplicação. Então o efeito é exatamente o mesmo que remover e copiar novamente o pacote.

Isto permite ao desenvolvedor atualizar arquivos da aplicação in loco, isto é, dentro da própria pasta webapps, reduzindo o tempo para se testar modificações pontuais na aplicação. Por isso foi dito antes que os pacotes abertos são uma conveniência para o desenvolvedor.

Entretanto o Tomcat não irá monitorar individualmente todos os arquivos e subdiretórios de uma pacote aberto, pois isto seria muito dispendioso com aplicações grandes. Em um pacote aberto, o Tomcat monitora apenas o descritor de deployment WEB-INF/web.xml.

Portanto o desenvolvedor que modificar alguma classe ou arquivo de dados da aplicação e atualiza-la diretamente dentro do pacote explodido tem que também forçar uma atualização do web.xml. Em sistemas Linux, isto pode ser feito pelo comando touch.

A única exceção a esta regra são páginas JSP. O container web não interpreta diretamente o texto de páginas JSP, e sim os converte em um Servlet, que é compilado e executado. Este processo ocorre no primeiro acesso à página, e os acessos posteriores executam diretamente o código já compilado.

A cada novo acesso à página, o container verifica se o arquivo JSP é mais novo do que o seu Servlet compilado, e em caso afirmativo re-gera e re-compila o Servlet. Por isso modificações sobre páginas JSP em pacotes abertos tem sempre efeito imediato, sem que seja necessário re-deployar o pacote.

### Limpando deployments

Na configuração padrão, quando o Tomcat recebe um pacote fechado, ele é descompactado imediatamente e assim é gerado, dentro da pasta webapps, um pacote aberto correspondente ao mesmo pacote.

Quando o pacote fechado é removido, a aplicação é desativada (undeployment) entretanto o pacote aberto gerado pela sua descompactação pode permanecer em webapps. O resultado é que novos deployments de um pacote com o mesmo nome, por exemplo atualizações da aplicação, poderão executar a versão antiga que estava descompactada dentro de webapps.

Veremos mais adiante como lidar com este “lixo” deixado pelo deployment, ou melhor, pelo undeployment de um pacote fechado. Por enquanto, esteja atento para fazer a limpeza manual sempre que necessário![[11]](#footnote-11)

## Testando a aplicação recém-deployada

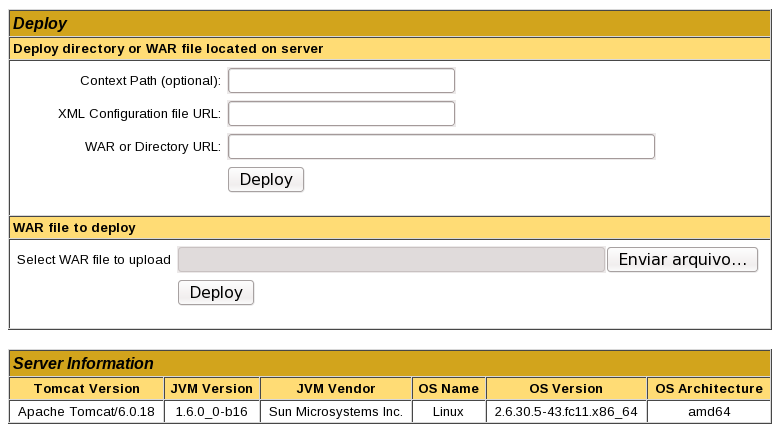
Para acessar uma aplicação, basta usar o nome do seu pacote WAR, sem a extensão, como o nome de um diretório na URL do servidor Tomcat. Por exemplo, o pacote hoje.war seria acessado como http://127.0.0.1:8080/hoje.

Este nome de diretório é chamado de contexto da aplicação web. Não é coincidência que a configuração do Tomcat, no arquivo server.xml, possui um elemento chamado de <Context>.

Mais adiante veremos como modificar configurações da aplicação web, como o nome de contexto ou a página inicial.

## Deployment via Manager

A forma manual de se fazer o deployment de uma aplicação é utilizando o Manager, o que permite tanto deployments interativos quando deployments automatizados via script.



**Figura 1.4 –** formulários HTML para deployment de aplicações via Manager

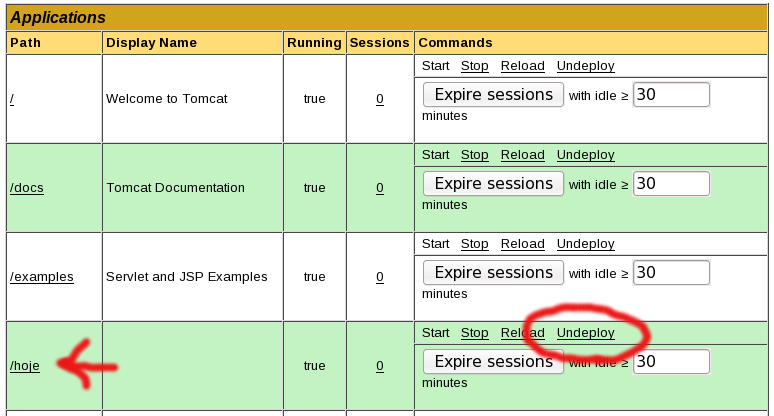
Observe que, no final da página “List Applications”, há dois formulários HTML para deployment de aplicações (figura 1.1):

* O primeiro permite deployar um pacote aberto ou fechado, que já esteja em algum sistema de arquivos visível para o servidor Tomcat, e permite ainda fornecer algumas configurações extras, que serão vistas mais à frente;
* O segundo possibilita fazer o upload de um pacote fechado da estação cliente para o servidor, e ativa-lo imediatamente.

Ou seja, o segundo formulário permite fazer deployment remoto de aplicações no servidor Tomcat. Neste caso, o pacote será salvo na própria pasta webapps, como se tivesse sofrido auto-deploy.

Um servidor Tomcat mal-configurado pode ser utilizado para propagar vermes para a sua rede local pelo deployment remoto de aplicações utilizando o Manager. Portanto, certifique-se de definir usuários administradores com uma senha forte. Além disso limite o acesso ao Manager aos endereços IP da sua rede local, ou então apenas aos endereços IP das estações dos administradores. Isto pode ser feito pelo firewall do SO (iptables no Linux) ou então utilizando as válvulas do Tomcat, que serão vistas no capítulo sobre configurações de rede do Tomcat nesta apostila.

Para realizar o undeployment de uma aplicação, utilize a página “List Applications” do Manager, e localize a aplicação desejada pelo seu path, que é o nome de contexto, e por padrão é o nome do pacote sem a extensão. Então clique no link “Undeploy”, confirme a figura 1.2.



**Figura 1.5 –** Link para undeployment de aplicações via Manager

Note que não é possível fazer o undeploy do próprio Manager, embora ele apareça na lista de aplicações deployadas.

Nunca use o “reload” do navegador com o Manager. Use sempre os links, como “List Applications”, pois caso contrário você irá re-executar a última operação, e ela pode ser potencialmente destrutiva, como o undeploy de uma aplicação.

Até onde temos conhecimento, o undeploy via Manager nunca deixa “lixo” de versões descompactadas dos pacotes fechados, como ocorre com o auto-deploy em algumas versões do Tomcat.

### Atualizando e suspendendo de aplicações

O desenvolvedor que estiver usando pacotes abertos e fez modificações in loco sobre a aplicação pode usar o Manager para forçar o redeploy de uma aplicação, mesmo que ela tenha sido originalmente deployada via auto-deploy. Basta para isto usar o link “Reload” na página “List Applications”.

Também está disponível o link “Stop”, que permite suspender temporariamente uma aplicação, impedindo que ela seja acessada, mas sem realizar o seu undeploy do servidor Tomcat. Ela será novamente disponibilizada pelo link “Start”.

Uma aplicação com erros no deployment poderá aparecer como suspensa no Manager, mas neste caso não será possível inicia-la com o link “Start”. Será necessário fazer o undeploy, gerar um novo pacote com as correções necessárias e então fazer um novo deployment. Nem sempre irá funcionar realizar as correções in loco e depois usar o link “Start”.

## Desligando o auto-deploy

O auto-deploy de pacotes WAR no Tomcat é definido ao nível de host. Na instalação padrão, podemos encontrar dentro do arquivo server.xml o elemento <Host> com a seguinte configuração:

* <Host name="localhost" **appBase="webapps"**

1. **unpackWARs="true" autoDeploy="true"**

xmlValidation="false" xmlNamespaceAware="false">

Os atributos que nos interessam estão em negrito:

* appBase – indica qual diretório será monitorado para auto-deploy de aplicações. Se for indicado um caminho relativo, o diretório será considerado dentro da instalação do Tomcat, mas é possível indicar um caminho absoluto;
* unpackWARs – diz se o Tomcat irá descompactar ou não os pacotes fechados que forem deployados. O valor padrão é “true” para contornar aplicações que esperam ter acesso direto aos seus próprios arquivos, o que é uma violação dos padrões do Java EE[[12]](#footnote-12). Ao contrário do que muitos pensam, descompactar a aplicação não melhora seu desempenho em execução;
* autoDeploy – se o valor padrão for alterado para “false”, não haverá mais auto-deploy, e todos os pacotes terão que ser instalados manualmente via o Manager, ou então serão deployados automaticamente apenas na inicialização (startup) do Tomcat.

Também pode nos interessar um atributo que não aparece na configuração padrão, mas que pode ser acrescentado caso se deseje alterar seu valor default:

* deployOnStartup – determina se pacotes na pasta indicada por appBase serão automaticamente deployados na inicialização do Tomcat.

Uma configuração adotada por muitos administradores em produção é desligar unpackWARs e também o autoDeploy. Isto poupa espaço em disco, processador e memória do servidor, além do que estes administradores se sentem mais confiantes quando novas aplicações só são ativadas por eles mesmos.

Ainda mais em versões anteriores do Tomcat, que falhavam em remover a versão aberta do pacote no undeployment, pode ser útil eliminar a duplicação de pacotes gerada por unpackWARs com o valor padrão “true”.

Já deployOnStartup pode ser modificado para “false” quando os contextos forem definidos explicitamente em vez de implicitamente pelo processo de deployment. Mais detalhe sobre esta possibilidade serão apresentados no próximo capítulo.

Se você está fazendo o deployment via Manager, é necessário manter o deployOnStartup com o valor padrão “true”, pois caso contrário as aplicações não serão ativadas no reinício do Tomcat.

## Exercícios

## Laboratório 1.1. Compilação, empacotamento e auto-deploy do exemplo

(Prática DIrigida)

**Objetivo:** colocar no ar o exemplo “hoje”, utilizando auto-deploy

Antes de mais nada, use o “root” para se incluir no grupo “tomcat”, de modo que você tenha acesso à pasta webapps do Tomcat instalado via JPackage.

Depois retorne ao seu usuário, entre na pasta Cap1/Lab1 e utilize as instruções já fornecidas neste capítulo para compilar e empacotar o exemplo.

Em seguida, copie o pacote hoje.war para a pasta webapps e confirme tanto pelos logs quanto pelo Manager que o deployment transcorreu com sucesso.

Eis um exemplo da mensagem que você deverá encontrar no catalina.out:

* 01/10/2009 00:43:31 org.apache.catalina.startup.HostConfig deployWAR

INFO: Deploying web application archive hoje.war

Observe que na pasta webapps agora existe, além do arquivo hoje.war, um subdiretório hoje.

Finalmente, acesse a aplicação pela URL http://127.0.0.1:8080/hoje. O resultado esperado é uma página como:



**Figura 1.7 –** Link para undeployment de aplicações via Manager

Uma vez que a aplicação tenha funcionado corretamente remova o arquivo hoje.war da pasta webapps.

Verifique nos logs do Tomcat que o undeployment foi bem-sucedido:

* 01/10/2009 01:27:12 org.apache.catalina.startup.HostConfig checkResources

INFO: Undeploying context [/hoje]

Verifique também que a aplicação não aparece mais na listagem do Manager, nem está mais acessível para o navegador, e confirme que não foi deixado nenhum lixo na pasta webapps. Isto é, confirme que na pasta webapps não existe mais um subdiretório hoje.

Laboratório 1.2. Auto-deploy de pacotes abertos

(Laboratório)

**Objetivo:** colocar no ar novamente o exemplo “hoje”, mas desta vez como pacote aberto, e experimentar com o hot-redeploy.

Antes de mais nada, certifique-se de ter realizado o undeploy do pacote hoje.war instalado no Tomcat no laboratório anterior.

Agora, aproveitando que a pasta Cap1/Lab1 já segue a estrutura de um pacote WAR, copie seu conteúdo integral para a pasta webapps do Tomcat, preferivelmente alterando o nome do diretório resultante. Por exemplo:

$ cd ~/Cap1

$ cp -r Lab1 /var/lib/tomcat6/webapps/hoje-aberto

Verifique que os logs do Tomcat e no Manager que a nova aplicação está disponível, então acesse a URL http://127.0.0.1:8080/hoje-aberto. O resultado deverá ser a página já conhecida de boas-vindas da aplicação.

Agora modifique a página inicial da aplicação em:

/var/lib/tomcat6/webapps/hoje-aberto/index.jsp

A modificação fica a critério do aluno (alterar o título, fonte, cores, etc). O importante é que seja possível reconhecer a alteração quando a nova página for salva e recarregada no navegador. A idéia é comprovar que alterações páginas JSP tem efeito imediato.

Agora, siga o primeiro link na página inicial, “Usando um Servlet”, e fique nesta página. Modifique o fonte Java deste Servlet dentro de webapps:

/var/lib/tomcat6/webapps/hoje-aberto/WEB-INF/classes/exemplo/HojeServlet.java

Não deverá ser difícil fazer uma modificação de texto, por exemplo, no título do Servlet, mesmo sem conhecimento prévio de Java.

Então recompile o Servlet in loco:

$ cd /var/lib/tomcat6/webapps/hoje-aberto/WEB-INF/classes

$ javac exemplo/\*.java

Caso hajam erros de compilação, certifique-se de que a variável de ambiente CLASSPATH inclui o caminho para o JAR da API de Servlets, como foi feito no laboratório anterior.

Depois de uma compilação bem-sucedida, recarregue o Serlvet no navegador. Não deverá ser observada nenhuma mudança, indicando que ainda estamos executando a versão “antiga” do Servlet, anterior à edição.

Force então o hot-redeployment da aplicação abrindo e salvando o descritor web.xml, ou então usando o comando touch:

$ touch ../web.xml

Note que o comando acima funciona porque o diretório corrente ainda é a pasta WEB-INF/classes de hoje-aberto.

O log do Tomcat deverá indicar que o pacote foi recarregado:

* 01/10/2009 01:27:12 org.apache.catalina.startup.HostConfig checkResources

1. INFO: Undeploying context [/hoje]
2. 01/10/2009 01:43:31 org.apache.catalina.core.StandardContext reload
3. INFO: Reloading this Context has started
4. 01/10/2009 01:48:13 org.apache.catalina.startup.HostConfig checkResources

INFO: Reloading context [/hoje-aberto]

E, agora sim, recarregar a página no navegador irá mostrar as modificações realizadas no Servlet.

Laboratório 1.3. Recarga de pacotes via Manager

(Laboratório)

**Objetivo:** comandar a atualização de aplicações, modificadas in loco, via o Manager

Inicie fazendo o deploy de uma nova versão “aberta” do exemplo do primeiro laboratório deste capítulo, tal qual foi feito no exercício anterior, mas utilizando um nome diferente.

Verifique que a nova versão da aplicação, por exemplo “meu-hoje” está disponível para o navegador. Crie o hábito de sempre confirmar suas operações verificando as últimas entradas no log catalina.out.

Siga agora o último link, “Usando JSP 2.0 com JavaBeans e EL” e fique nesta página.

Para que o exercício não fique muito repetitivo, vamos desta vez modificar a classe Java Bean em vez do Servlet, de modo a alterar a formatação da data exibida pela página JSP. Observe que, no exemplo original, a data aparece em formato curto, estilo “dd/mm/aaa”.

Então abra em um editor de textos o arquivo:

/var/lib/tomcat6/webapps/hoje-aberto/WEB-INF/classes/exemplo/HojeBean.java

E o altere conforme o trecho à seguir:

* public HojeBean() {

1. DateFormat df = DateFormat.getDateInstance(**DateFormat.LONG**);
2. str = df.format(new Date());

}

Note que a única mudança foi acrescentar o argumento DateFormat.LONG na chamada ao método getDateInstance().

Recompile a classe, como foi feito no exercício anterior, e não provoque o re-deploy! Observe que a página JSP continua exibindo a data no formato curto.

Agora, entre no Manager e use o link “Reload” para recarregar a aplicação. Recarregue a página JSP, e veja que agora a data é exibida com o nome do mês por extenso, em vez do número do mês.

Comprovado que o Manager pode forçar o re-deploy de uma aplicação que havia sido originalmente deployada via auto-deploy, aproveite para usar o Manager para fazer o undeploy de todas as aplicações instaladas até agora pelos exercícios deste capítulo. Vamos evitar que nossa instalação do Tomcat acumule “lixo” que possa vir a atrapalhar exercícios futuros.

Laboratório 1.4. Deployment manual, via Manager

(Laboratório)

**Objetivo:** colocar no ar novamente o exemplo “hoje”, mas utilizando o Manager.

Antes de mais nada, tenha certeza de ter desinstalado (undeployado) os pacotes instalados como parte dos laboratórios anteriores. É sempre mais fácil iniciar com uma instalação “limpa” do servidor de aplicação do que ficar lidando com possíveis conflitos entre aplicações – ou então executar a aplicação errada e por isso achar que está cometendo algum erro.

Entre no manager, e utilize o segundo formulário da página “List applications” faça manualmente o deployment do pacote hoje.war (sim, aquele mesmo do primeiro exercício deste capítulo).

Confirme que tanto o Manager quanto os logs do Tomcat acusam o deployment bem-sucedido, e então acesse a aplicação pelo seu navegador. A esta altura do campeonato, você não precisa mais que lhe digam qual a URL, certo? ;-)

Laboratório 1.5. Desligar o auto-deploy

(Desafio)

**Objetivo:** Forçar que todos os deployments sejam realizados explicitamente via Manager.

Retire todas as aplicações deployadas em exercícios anteriores, e modifique o elemento <Host> padrão no arquivo server.xml, conforme o apresentado neste capítulo para desabilitar o auto-deploy.

Será necessário realizar estas alterações como root, e então reinicie o Tomcat.

Agora experimente copiar o pacote hoje.war para a pasta webapps, e veja que o novo pacote é ignorado pelo Tomcat.

Então reinicie novamente o servidor, e veja que a aplicação “hoje” agora está disponível.

Desinstale a aplicação “hoje” pelo Manager, e veja que o efeito é imediato.

Por fim, retorne a configuração do <Host> para o padrão, habilitando novamente o auto-deploy.

## Conclusão

Aprendemos como instalar (e desinstalar) aplicações simples no Tomcat. Mas aplicações reais irão exigir customizações nas configurações da própria aplicação ou do próprio Tomcat. Estas customizações são o tema dos próximos caṕítulos.

Questões de Revisão

* É verdade que o Tomcat fornece um utilitário de linha de comando para a instalação de aplicações?

* É necessário editar algum arquivo de configuração do Tomcat para ativar aplicações empacotadas em formato WAR?

# Configuração de Contextos

Este capítulo apresenta como fornecer ao Tomcat configurações customizadas para aplicações web.

**Tópicos:**

* Descritores padrão e proprietários
* Entradas de ambiente
* Configuração de contextos explícita e implícita
* URL de acesso à aplicação
* Página inicial da aplicação

## O Descritor Padrão web.xml

O primeiro local para customizar um pacote WAR é o seu próprio descritor de deployment, WEB-INF/web.xml. Embora muitas das suas diretivas existam para atender ao desenvolvedor e não devam ser modificadas pelo administrador, outras diretivas existem para que o administrador possa customizar o comportamento da aplicação ao seu ambiente.

As configurações do web.xml são padronizadas pelo Java EE (mais especificamente pela API de Servlets) e por isso são independentes do servidor utilizado. Entretanto o padrão não define todas as configurações possíveis e necessárias para uma aplicação web, por isso cada servidor de aplicação acrescenta seus próprios mecanismos de configuração para cada pacote. No Tomcat, temos as configurações de contextos, que serão vistas ainda neste capítulo.

É comum este arquivo estar com erros de sintaxe ou com o nome errado (por exemplo, o nome da pasta WEB-INF escrito em minúsculas). Neste caso, o Tomcat irá usar a configuração padrão na pasta conf/web.xml do servidor. Então uma aplicação com erros de configuração pode ser deployada com sucesso e estar disponível para os usuários. Mas não estará com as configurações desejadas.

### Página inicial da aplicação

O descritor padrão web.xml pode conter vários elementos <welcome-file> indicando arquivos que serão retornados ao navegador em resposta a URLs que não especifiquem um arquivo em especial, mas em vez disso indiquem apenas um diretório.

O uso comum deste elemento é definir a página inicial da aplicação, que por convenção é index.jsp, mas poderia ser qualquer outra, por exemplo default.jsp imitando a convenção do IIS.

Um exemplo típico seria:

* <welcome-file-list>

1. <welcome-file>index.html</welcome-file>
2. <welcome-file>index.jsp</welcome-file>

</welcome-file-list>

Havendo vários elementos <welcome-file>, os arquivos serão procurados na ordem em que aparecem no descritor padrão, e o primeiro a ser encontrado será retornado para o usuário.

Aproveite para verificar o arquivo web.xml na pasta conf do Tomcat (ou /etc/tomcat6 na instalação JPackage) e veja porque o arquivo index.jsp funciona como paǵina inicial mesmo que o descritor web.xml do próprio pacote não tenha nenhum elemento <welcome-file>.

### Entradas de ambiente JNDI

Uma das configurações que podem ser colocadas no descritor padrão web.xml é a definição de entradas de ambiente[[13]](#footnote-13), que são parâmetros de configuração para a própria aplicação, acessados como objetos Java pelo serviço de diretório interno ao servidor de aplicação.

Muitos programadores Java utilizam arquivos de propriedades, então system properties da JVM ou mesmo tabelas de configuração em bancos de dados para parametrizações que poderiam ser realizadas pelas entradas de ambiente JNDI. Mas não vamos entrar no mérito de se isto é ou não uma boa idéia.

Um exemplo de definição de entrada de ambiente para uma aplicação que serve ao clube de futebol Flamengo do Rio de Janeiro seria:

* <env-entry>

1. <env-entry-name>saudacao</env-entry-name>
2. <env-entry-value>rubro-negras</env-entry-value>
3. <env-entry-type>java.lang.String</env-entry-type>

</env-entry>

Então, caso se deseje que a aplicação apresente uma saudação diferente (por exemplo, porque estamos fazendo a instalação para outro clube de futebol), o administrador iria alterar o arquivo web.xml dentro do pacote antes de fazer o deployment da aplicação.

## O descritor proprietário context.xml

Um pacote WAR pode conter o arquivo META-INF/context.xml, que é um descritor de deloyment proprietário para o Tomcat. Outros servidores de aplicação Java EE irão ignorar este arquivo.

Um pacote WAR pode incluir descritores proprietários para vários servidores de aplicação diferentes, de modo que ele carregue as customizações necessárias para cada ambiente. Assim uma software-house poderia fornecer um único arquivo com seu sistema já preparado para vários clientes, independente do servidor utilizado por cada um. Mas o provável é que o administrador tenha que alterar este arquivo, diretamente ou então por meio da ferramenta de deployment o seu servidor.

Note que o descritor proprietário do Tomcat está na pasta META-INF, que seria a pasta de configuração de um pacote JAR simples, em vez de na pasta WEB-INF, que seria a pasta de configuração de um pacote WAR.

### Sobreponto entradas de ambiente

Ao longo deste curso veremos várias configurações que podem ser feitas no context.xml do Tomcat, mas vamos iniciar com um exemplo simples: um valor alternativo para a entrada de ambiente.

Embora o Java EE assuma que o administrador vai modificar o descritor padrão web.xml fornecido com a aplicação, algumas empresas consideram que os fontes devam ser transferidos do ambiente de desenvolvimento para produção inalterados, e que o descritor padrão é parte destes fontes.

Por isso o Tomcat permite que o administrador use o descritor proprietário para sobrepor o valor de uma entrada de ambiente. Desta forma, ficamos com uma separação clara de responsabilidades: o desenvolvedor modificar o descritor padrão, e o administrador apenas o descritor proprietário.

Um exemplo de descritor proprietário do Tomcat para sobrepor o valor de uma entrada de ambiente seria:

* <?xml version="1.0" encoding="UTF-8"?>

1. <Context>
2. <Environment
3. name="saudacao"
4. type="java.lang.String"
5. value="coloradas"
6. override="false"/>

</Context>

Note que o valor do atributo override é o contrário do que seria intuitivo: ele diz se o valor definido no descritor padrão sobrepõe o valor definido no descritor proprietário, e não o contrário.

## Definição de contextos no Tomcat

A inclusão do descritor proprietário META-INF/context.xml no pacote WAR é apenas um exemplo do que o Tomcat chama de definição implícita de contextos. Já fomos apresentados, no capítulo 3, ao elemento <Context> no arquivo de configuração server.xml do Tomcat. Este é um exemplo de definição explícita de contextos.

Na verdade o Tomcat suporta duas formas de definição implícita de contextos, que é são:

1. Auto-deploy;
2. Deploy manual via Manager (segundo formulário).

E suporta quatro formas de definição explícita de contextos:

1. Elemento <Context> no server.xml;
2. Arquivo conf/<engine>/<host>/nome-do-contexto.xml, por exemplo conf/Catalina/localhost/minhaap.xml;
3. Deploy manual via Manager (primeiro formulário).

Na verdade, a segunda forma de definição implícita é mais ou menos equivalente à primeira, já que o pacote WAR deployado remotamente será salvo em webapps, igual aos pacotes deployados automaticamente.

De maneira similar, a terceira forma de se definir um contexto explícito é equivalente à segunda, pois o Manager no primeiro formulário irá gerar o arquivo nome-do-contexto.xml como resultado do deployment local.

### Nome (path) e base do contexto

Tanto o arquivo context.xml quanto o server.xml e também o nome-do-contexto.xml contém um elemento <Context> com basicamente a mesma sintaxe. A única diferença está nos atributos path e docBase, que podem ser ignorados dependendo da forma como o contexto foi definido.

O atributo path indica o nome do contexto na URL de acesso à aplicação. Ele é ignorado exceto quando usado o elemento <Context> é definido no arquivo server.xml. Nos demais casos, o nome do contexto é definido pelo nome do pacote WAR ou então pelo nome do arquivo XML dentro de conf/engine/host, o tal nome-do-contexto.xml.

Já o atributo docBase indica o caminho para o pacote deployado no contexto, e pode apontar tanto para um pacote fechado quanto aberto. Ele é obviamente ignorado nas definições implícitas de contextos e necessário nas definições explícitas.

Então as aplicações não necessitam estar todas juntas na pasta webapps do Tomcat. Elas podem estar espalhadas pelo servidor do modo como o administrador achar conveniente, embora no final das contas seja mais prático ficar mesmo com a pasta webapps.[[14]](#footnote-14)

Já o atributo path permite gerar uma URL de acesso a uma aplicação independente do seu nome de pacote.

Ele permite ainda definir várias aplicações independentes, possivelmente utilizando logins de acesso e bancos de dados separados, mas que compartilham o mesmo pacote WAR, o que é ótimo para provedores de hospedagem e empresas de SaS (Software as a Service).

Por exemplo, digamos que se deseje acessar o pacote WAR saudacao.war com a URL http://127.0.0.1:8080/mengo. Seria necessário criar o arquivo conf/Catalina/localhost/mengo.xml com o conteúdo:

* <?xml version="1.0" encoding="UTF-8"?>

1. <Context docBase="../saudação.war">

</Context>

O docBase padrão de um contexto é relativo ao webapps (ou melhor, ao appBase do seu <Host>) mas não pode estar dentro do próprio webapps, sendo ignorado neste caso. Por isso o exemplo coloca o pacote WAR em “../” que quer dizer no diretório pai de webapps (no caso da instalação JPackage, em /var/lib/tomcat6).

### Contextos explícitos com auto-deploy

É possível sim fornecer uma definição explícita de contexto na pasta conf/engine/host, em vez de no META-INF/context.xml do próprio pacote. Mas o nome-do-host.xml tem que ser igual ao nome do pacote.

Alguns administradores preferem fazer assim, para impedir que os programadores possam fornecer configurações de contexto customizadas em suas aplicações, obrigando-as a utilizar as configurações de segurança, acesso a banco de dados etc definidas pelo administrador.

Neste caso, é possível incluir no elemento <Host> o atributo deployXML com o valor “false” para assegurar que qualquer descritor proprietário fornecido com os pacotes WAR seja ignorado pelo Tomcat durante o auto-deploy.

### Página inicial do Tomcat

A página inicial do Tomcat, ou contexto raiz, pode ser configurada de duas formas:

1. Um pacote com o nome ROOT.war. É assim que é fornecida a página inicial default do Tomcat, com os links para documentação e o Manager;
2. Um contexto com nome (path) vazio.

É claro, não é possível usar as duas formas, pois não podem haver duas aplicações com a mesma URL de acesso. Mas o fato importante é que é possível sim ter sua aplicação web Java EE como a página inicial do seu servidor web.

## Instalando aplicações sem deployment

Considerando as maneiras de se definir contextos explicitamente, temos que é possível instalar aplicações dentro do Tomcat sem que haja necessariamente uma etapa explícita de deployment.

Esta é uma possibilidade muito interessante para bundles, onde uma aplicação é fornecida já pré-configurada junto a um servidor Tomcat, de modo que o usuário pensa apenas em instalar a aplicação, em vez de pensar em instalar o Tomcat e depois deployar a aplicação dentro dele.

O próprio Tomcat usava esta possibilidade com suas aplicações administrativas, até a versão 5.5. Em uma instalação manual, era possível verificar a existência do arquivo:

conf/Catalina/localhost/manager.xml

E que o pacote WAR do Manager estava em:

server/webapps/manager

Isto é, junto aos JARs do próprio servidor, e não junto com as aplicações normais.

No Tomcat 6.0 o Manager voltou a ser colocado dentro da pasta webapps do host default “localhost”.

## Exercícios

Laboratório 1.1. Entradas de ambiente

(Prática Dirigida)

**Objetivo:** Modificar uma parametrização de aplicação realizada sobre o descritor padrão do pacote WAR

O exemplo em Cap5/Lab1 é um simples “hello, world” mas que incorpora na sua mensagem de boas-vindas uma string configurada como entrada de ambiente no descritor web.xml.

Isto ilustra parametrizações que seriam típicas em aplicações comerciais, que devem ser configuradas para o ambiente de cada cliente.

Para que o aluno não necessite consultar as instruções no capítulo anterior, seguem os passos para compilar, empacotar e deployar o exemplo:

$ export CLASSPATH=$CLASSPATH:/usr/share/java/servlet.jar

$ cd ~/Cap5/Lab1

$ cd WEB-INF/classes/

$ javac exemplo/\*.java

$ cd ~/Cap5/

$ jar cvf saudacao.war -C Lab1 .

$ cp saudacao.war /var/lib/tomcat6/webapps

Primeiro faça o deployment da aplicação sem alterações, para ver seu comportamento com as configurações default.

Depois modifique o descritor padrão para alterar a mensagem de saudação, configurando uma saudação para uma torcida diferente, gere um novo WAR com a atualização e faça o seu redeploy, pela cópia do pacote atualizado para a pasta webapps do Tomcat.

Laboratório 1.2. Sobrepondo entradas de ambiente

(Laboratório)

**Objetivo:** Modificar uma parâmetro de aplicação (entrada de ambiente) por meio do descritor proprietário do Tomcat, sem alterar o descritor padrão.

Aproveite o mesmo exemplo do exercício anterior, mas desta vez modifique o arquivo META-INF/context.xml. Observe que ele já traz um modelo comentado para evitar que o aluno perca tempo em digitação.

Faça então um redeployment da aplicação e verifique se a mensagem realmente mudou.

Laboratório 1.3. Múltiplas instâncias da mesma aplicação, via server.xml

(Laboratório)

**Objetivo:** Rodar, lado-a-lado, duas versões da aplicação saudacao.war, customizadas para torcidas diferentes.

Cada versão da aplicação tem que usar uma URL condizente com a sua torcida, por exemplo:

* http://127.0.0.1:8080/mengo
* http://127.0.0.1:8080/fogo

Mas queremos usar o mesmo pacote WAR para ambas. Então este pacote deve ser executado à partir de um diretório que não o webapps do Tomcat, e duas definições de contexto devem apontar para este mesmo pacote.

Defina os dois contextos pela edição direta do server.xml, e confirme seu funcionamento depois de reiniciar o Tomcat.

Depois volte atrás nas edições no arquivo server.xml, deixando sua instalação do Tomcat sem nenhuma instância deployada da aplicação saudacao.war, e reinicie novamente o servidor.

Laboratório 1.4. Múltiplas instâncias da mesma aplicação, via conf/engine/host e aplicação padrão do Tomcat

(Laboratório)

**Objetivo:** Rodar, lado-a-lado, duas versões da aplicação saudacao.war, customizadas para torcidas diferentes, sendo que uma delas será a aplicação padrão do servidor.

Este exercício é uma variação da anterior, onde cada versão da aplicação tem que usar uma URL condizente com a sua torcida. Use URLs diferentes das usadas anteriormente, por exemplo:

* http://127.0.0.1:8080/flamengo
* http://127.0.0.1:8080/botafogo

Se preferir, modifique também as mensagens de saudação para configurar torcidas diferentes das usadas no último exercício.

Ainda queremos usar o mesmo pacote WAR para ambas. Então este pacote deve ser executado à partir de um diretório que não o webapps do Tomcat, e duas definições de contexto devem apontar para este mesmo pacote.

Experimente definir os contextos pela criação de arquivos como conf/Catalina/localhost/flamengo.xml e de modo similar para botafogo.xml.

Reinicie o Tomcat e confirme que as novas versões da aplicação estão no ar.

Por fim, escolha um deles e transforme na página padrão do Tomcat, acessível pela URL http://127.0.0.1:8080. Será necessário reiniciar o Tomcat mais uma vez.

Depois disso você ainda consegue acessar o Manager? Porque?

Laboratório 1.5. Mudando a página inicial da aplicação

(Laboratório)

**Objetivo:** Usar como página inicial qualquer uma que não index.jsp.

Neste exercício vamos reaproveitar o exemplo do laboratório 1 do capítulo 4, a aplcação “hoje”.

Mas, em vez de iniciar a aplicação com a página de links para os vários exemplos de programação web em Java, escolha um deles (por exemplo el.jsp) e o configure como a página inicial da aplicação, pela edição do descritor padrão web.xml.

Faça as edições necessárias e então faça o deploy (ou redeploy) da aplicação, e verifique se que a URL http://127.0.0.1:8080/hoje exibe mesmo a página desejada.

## Conclusão

Neste capítulo ganhamos familiaridade com os arquivos de configuração de aplicações web dentro do Tomcat, preparando o terreno para as configurações mais interessantes que serão vistas nos próximos capítulos, relativas a bibliotecas, bancos de dados e segurança.

Questões de Revisão

* O descritor padrão web.xml de um pacote WAR é o local onde são configurados parâmetros globais do container web, por exemplo a porta TCP para receber requisições HTTP?

* O descritor proprietário do Tomcat em META-INF/context.xml pode especificar um caminho (path) alternativo para a URL de acesso à aplicação?

# Configuração de Contextos

Este capítulo apresenta como utilizar o Apache Ant para automatizar a compilação, empacotamento e deployment de aplicações Web Java EE

**Tópicos:**

* Para que serve o Ant
* Instalação do Ant
* Alvos e tarefas
* Tarefas customizadas do Tomcat

## O Que é o Ant

O Apache Ant é uma ferramenta criada para automação de tarefas no desenvolvimento de aplicações e servidores de aplicação Java EE. Seu uso hoje é considerado obrigatório por profissionais Java, e a maioria dos IDEs populares, como Eclipse e NetBeans, delegam basante funcionalidade para o Ant.

O Ant também é bastante útil para o administrador, e agora que será necessário usar aplicações de exemplo um pouco mais complexas – e aplicações diferentes para cada exercício, em vez de reaproveitar os dois exemplos utilizados até o momento – o uso do Ant poderá nos poupar bastante tempo de digitação.

Em uma aplicação real, a estrutura de pastas para arquivos HTML / JSP, classes Java e bibliotecas pode ser bastante complexa, inviabilizando a construção dos pacotes WAR manualmente com o comando jar.

Além disso, muitos desenvolvedores não vão querer enviar os fontes das classes Java juntamente com os binários (pacote WAR) da aplicação como fizemos até agora. Separar fontes e binários facilita em muito o controle de versões, backup e a transferência para homologação ou desenvolvimento.

O padrão de fato para estas atividades é o uso de scripts Ant, pois eles são portáveis, ao contrário de scripts shell, que rodam apenas em Unix e Linux.

## Instalação via JPackage

A instalação do Ant via JPackage é fornecida pelo Fedora e simimares, e está quebrada em uma série de pacotes, onde cada plug-in (por exemplo, administração de servidores de aplicação específicos) está em um pacote separado.

Normalmente é suficiente instalar o pacote ant via yum, que todas as dependências (e são muitas!) serão resolvidas, mas em algum cenário atípico você pode necessitar de algum plug-in que não está configurado nas dependências obrigatórias da sua distribuição.

Portanto verifique se o plug-in desejado está disponível em algum pacote amt-\*.rpm da sua distribuição (ou similar para Debian e derivados) antes de partir para a instalação manual do plug-in.

## Instalação manual do Ant

O Ant também é um projeto Apache, e pode ser baixado em http://ant.apache.org. Siga o e clique no link “Binary Distributions”

Baixe o arquivo apache-ant-1.6.5-bin.zip (ou uma versão mais recente) e descompacte na sua pasta home, ou em outra pasta à sua escolha.

Então defina a variável de ambiente ANT\_HOME apontando para o diretório de instalação do Ant. Em seguida, acrescente a pasta bin da instalação do Ant ao PATH de comandos do shell:

$ export ANT\_HOME=$HOME/apache-ant-\*

$ export PATH=$ANT\_HOME/bin:$PATH

Se preferir, coloque estes comandos no seu script de profile, para que o Ant esteja disponível em todos os shells abertos pelo seu usuário.

### Instalação manual x JPackage

Se sua distribuição do Linux já incluir uma instalação do Ant, mas você por algum motivo necessitar usar uma instalação manual (por exemplo, para usar uma vesão mais recente do que a fornecida pela distribuição) há um cuidado adicional a se tomar:

O Ant verifica a existência do arquivo de configuração /etc/ant.conf e caso ele exista é utilizada a definição do ANT\_HOME presente neste arquivo. Então será necessário edita-lo ou então remove-lo para que o sistema passe a suar a sua instalação manual do Ant.

## Verificando a instalação do Ant

Utilize a opção -version do comando ant de modo similar ao que foi feito no capítulo 1 para os comandos java e javac. Se o comando for reconhecido pelo shell, e retornar o número de versão correto, sua instalação do Ant está ok.

Também é uma boa idéia confirmar que o comando ant foi executado à partir da sua instalação manual (se for o caso) em vez de à partir do diretório de comandos do sistema operacional (como seria em uma instalação via JPackage). Por exemplo:

$ **ant -version**

Apache Ant version 1.7.1 compiled on February 23 2009

$ **which ant**

/usr/bin/ant

## Sintaxe dos buildfiles

Conceitualmente, o Ant é similar ao utilitário make utilizado no desenvolvimento em C. Entretanto, os scripts Ant (chamados de buildfiles) são bem diferentes dos makefiles por usarem sintaxe XML e serem portáveis, enquanto que um Makefile pode não funcionar corretamente mesmo em distribuições do Linux[[15]](#footnote-15).

O buildfile a ser executado é passado como argumento para a opção -f do comando ant. Se esta opção for omitida, o Ant executa por padrão o script no arquivo build.xml do diretório corrente.

O buildfile XML do Ant é organizado na hierarquia:

* Projeto (<project>)
  + Alvo (<target>)
    - Tarefa (task, como <javac>, <jar>, ou <copy>)

O Ant fornece tarefas pré-definidas para compilação, empacotamento e execução de aplicações Java, além de manipulação de arquivos, acesso a bancos de dados, sistemas de controle de versão e mesmo administração de servidores de aplicação.

O elemento <project> define o alvo padrão, que é executado caso o comando ant não indique o nome dos alvos desejados. Caso não se queira executar o alvo padrão, eles são passados como argumentos de linha de comando.

Alvos podem depender uns dos outros, de modo similar a um Makefile. Se for comandada a execução de um alvo com dependências, as dependências são executadas primeiro.

É possível implementar ambientes completos de teste automatizado e integração contínua utilizando apenas o Ant. Ou então, ele é capaz de automatizar completamente a construção de um ambiente de produção, desde a instalação do servidor de aplicações, população do banco de dados até o deployment das aplicações homologadas, obtidas à partir de um sistema de gerência de configuração.

### Um exemplo de buildfile

Na listagem 1.1 temos um exemplo de buildfile, na verdade o utilizado pelo primeiro exercício deste capítulo. O exemplo é o mesmo do capítulo 3, só que agora organizado em função do Ant.

Um buildfile normalmente inicia pela definição de variáveis (como tarefas <property>) onde podem ser configurados parâmetros como o diretório de instalação da aplicação ou caminhos para bibliotecas. Alternativamente, estes parâmetros podem ser configurados em um aquivo de propriedades Java SE chamado build.properties.

**Listagem 1.1** – buildfile do exemplo deste capítulo

* <project name="Cap6\_Lab1" default="tudo">

1. <target name="tudo" depends="limpa,compila,empacota,instala" />
3. <target name="variaveis">
4. <property name="tomcat" value="/usr/share/tomcat6" />
5. <!-- para uma instalação via JPackage -->
6. <property name="servlets" value="/usr/share/java/servlet.jar" />
7. <!-- para uma instalação manual do Tomcat
8. <property name="servlets" value="${tomcat}/common/lib/servlet-api.jar" />
9. -->
10. <property name="war" value="hoje-ant.war" />
11. </target>
13. <target name="limpa">
14. <delete>
15. <fileset dir="." includes="WEB-INF/classes/\*\*" />
16. <fileset dir="." includes="dist/\*\*" />
17. </delete>
18. </target>
20. <target name="compila" depends="variaveis">
21. <javac srcdir="src" destdir="WEB-INF/classes"
22. classpath="${servlets}"/>
23. </target>
25. <target name="empacota" depends="variaveis">
26. <jar destfile="dist/${war}">
27. <fileset dir="html" includes="\*\*" />
28. <fileset dir="." includes="WEB-INF/\*\*" />
29. </jar>
30. </target>
32. <target name="instala" depends="variaveis">
33. <copy file="dist/${war}" todir="${tomcat}/webapps" />
34. </target>
36. <target name="desinstala" depends="variaveis">
37. <delete file="${tomcat}/webapps/${war}" />
38. </target>

</project>

É comum fazer como neste exemplo, e inserir algos separados para limpeza, compilação, empacotamento e instalação da aplicação. Assim um desenvolvedor pode rodar o ant sem argumentos, e assim ter uma nova versão da aplicação compilada e empacotada “do zero”, ou então pode executar apenas os alvos realmente necessários para acelerar o processo.

Por exemplo, se houveram modificações apenas nas páginas JSP ou nos descritores de dpeloyment padrão ou proprietário, não há necessidade de recompilar a aplicação, mas apenas de reempacota-la e redeploya-la, o que seria realizado pelo comando:

$ ant empacota instala

### Um exemplo de projeto Java com Ant

O buildfile apresentado na listagem 1.1 pressupõe que seu projeto está em uma estrutura onde os fontes Java se encontram na pasta src, separados das páginas JPS, que estão na pasta html, e dos arquivos de configuração em WEB-INF e META-INF. A seguir temos o layout de diretórios do exemplo deste capítulo:

* Lab1
* build.xml
* dist
* html
* bean.jsp
* el.jsp
* ...
* src
* exemplo
* HojeServlet.java
* HojeBean.java
* WEB-INF
* classes
* web.xml

Os bytecodes são colocados pelo próprio buildfile na pasta WEB-INF/classes, e removidos pelo alvo “limpa”. Além disso, o buildfile coloca o resultado do empacotamento da aplicação (hoje-ant.war) dentro da pasta dist, e à partir daí ele pode ser instalado via auto-deploy ou pelo Manager.

Note que o projeto e seu buildfile não dependem em nada do Tomcat, a não ser por algumas referências a arquivos para a configuração do CLASSPATH e instalação da aplicação por auto-deploy. Seriam necessárias poucas mudanças para usar o mesmo buildfile com outro servidor de aplicações, e nenhuma mudança seria necessária na estrutura do projeto ou no código Java.

## Tasks customizados do Tomcat

Talvez o maior motivos da popularização do Ant dentro do mundo Java seja a facilidade com que ele pode ser extendido por plug-ins, que fornecem tarefas (tasks) customizadas.

Hoje em dia, todo software Java que seja utilizado no desenvolvimento ou administração de aplicações fornece algum plug-in para integração com o Ant. O Tomcat não é exceção.

Observe no exemplo da listagem 1.1 que o alvos de instalação e desinstalação da aplicação foram escritos utilizando tarefas de manipulação de arquivos, portanto eles utilizam o auto-deploy do Tomcat. E caso o auto-deploy esteja desligado na configuração do <Host>, ou o servidor Tomcat seja outra máquina que não a usada pelo desenvolvedor para rodar o ant?

Nestes casos podemos usar os tasks fornecidos pelo Tomcat para interação com o Manager. Eles podem realizar qualquer operação do Manager, entre elas deployment e undeployment de aplicações. A listagem 1.2 mostra quais seriam as modificações no buildfile para usar estas tasks:

**Listagem 1.2** – modificações sobre o buildfile para fazer deployment via Manager

* <project name="Cap6\_Lab2" default="tudo">

1. <target name="tudo" depends="limpa,compila,empacota,instala" />
3. <target name="variaveis">
4. ...
5. <property name="war" value="hoje-task" />
6. <property name="url" value="http://127.0.0.1:8080/manager" />
7. <property name="username" value="tomcat" />
8. <property name="password" value="tomcat" />
9. <taskdef name="deploy"
10. classpath="${tomcat}/lib/catalina-ant.jar"
11. classname="org.apache.catalina.ant.DeployTask" />
12. <taskdef name="undeploy"
13. classpath="${tomcat}/lib/catalina-ant.jar"
14. classname="org.apache.catalina.ant.UndeployTask" />
15. </target>
17. ...
18. <target name="instala" depends="variaveis">
19. <deploy url="${url}"
20. username="${username}"
21. password="${password}"
22. path="/${war}" war="dist/${war}.war"
23. />
24. </target>
26. <target name="desinstala" depends="variaveis">
27. <undeploy url="${url}"
28. username="${username}"
29. password="${password}"
30. path="/${war}"
31. />
32. </target>

</project>

No final das contas, as modificações são as esperadas:

* Definição dos tasks customizados propriamente ditos (<taskdef>);
* Definição de parâmetros necessários para o deploy (URL de acesso ao Manager, login e senha com permissões administrativos);
* E os alvos de instalação e desinstalação que passam a usar os tasks <deploy> e <undeploy> fornecidso pelo Tomcat, em vez da cópia direta de arquivos (auto-deploy).

## Exercícios

Laboratório 1.1. Projeto com Ant

(Prática Dirigida)

**Objetivo:** Re-fazer a compilação e deployment do exemplo “hoje” utilizando Ant

O exemplo deste laboratório (em Cap1/Lab1) já está organizado para usar o Ant, e já contém um buildfile pronto para uso, se você está usando um Tomcat instalado via JPackage.

Entretanto, será necessário editar o buildfile build.xml caso você esteja utilizando uma instalação manual do Tomcat. As modificações necessárias estão indicadas por comentários XML no próprio buildfile.

Então certifique-se de que seu Tomcat não está com “lixo” acumulado dos exercícios anteriores (faça o undeploy dos pacotes via Manager se for o caso) e então use o comando ant para compilar, empacotar e instalar a aplicação.

Como sempre, confirme que a instalação (deploymento) foi bem-sucedida verificando os logs do Tomcat e a lista de aplicações no Manager, e então acesse a aplicação – você já deve saber qual seria a URL de acesso.

Modifique a página inicial da aplicação, html/index.jsp, e comande o re-deployment via Ant (dica: o comando é o mesmo utilizado para o primeiro deployment).

Por fim, faça o undeploy da aplicação, também usando o Ant.

Aproveite este espaço para tomar nota das linhas de comando utilizadas em cada etapa, caso você precise relembrar:

Deployment:

Re-deployment:

Undeployment:

Laboratório 1.2. Deployment via Ant com Manager

(Laboratório)

**Objetivo:** Re-fazer a compilação e deployment do exemplo “hoje” utilizando os tasks customiuzados do Ant fornecidos pelo Tomcat

O exemplo deste laboratório (em Cap1/Lab2) já contém as modificações necessárias sobre o buildfile, mas podem ser necessárias edições adicionais caso você tenha configurado um usuário administrador para o Tomcat diferente de “tomat” com senha “tomcat”.

Então use os mesmos comandos do laboratório anterior (porém com o diretório corrente neste laboratório!) para realizar a compilação, empacotamento e instalação da aplicação via Ant.

Observe a saída diferente do Ant, indicando o uso do taks customizado e a comunicação bem-sucedida com o Manager.

E como sempre não deixe de validar o deployment da aplicação, não apenas pela sua execução mas também pelo próprio Manager e logs do Tomcat.

## Conclusão

Agora que conhecemos o Apache Ant estamos preparados para usar exemplos mais sofisticados e experimentar suas demandas específicas sobre as configurações do servidor de aplicações Tomcat.

# Bibliotecas

Aplicações reais fazem uso extenso de bibliotecas prontas e frameworks, que devem se instalados no servidor de aplicações ou inclusas como parte da aplicação.

Tópicos:

* Instalação de bibliotecas e APIs de terceiros
* Como usar várias versões da mesma biblioteca

## Bibliotecas Java

Bibliotecas Java são em geral fornecidas como pacotes JAR, que são arquivos ZIP contendo classes Java compiladas (arquivos .class). O Java SE fornece um conjunto extenso de bibliotecas junto com a JVM, e o Java EE torna obrigatório a presença de várias outras bibliotecas que implementam suas várias APIs.

O próprio Tomcat faz uso de várias bibliotecas na sua implementação, e algumas delas estão também disponíveis para a aplicação. Apensar disso, qualquer aplicação pode necessitar de bibliotecas adicionais.

### Onde instalar bibliotecas

O Tomcat traz suas bibliotecas na pasta lib. É possível colocar bibliotecas adicionais nesta pasta, e elas estarão visíveis para os componentes do próprio Tomcat e também pelas aplicações.

Já uma aplicação pode trazer bibliotecas adicionais na pasta WEB-INF/lib do seu pacote. Estas bibliotecas não serão visíveis por outras aplicações ou por componentes do Tomcat. Então é possível rodar na mesma instância do Tomcat duas aplicações usando versões diferentes de um mesmo framework, desde que ambas tragam as classes do framework dentro do seu respectivo pacote.

Também é possível poupar espaço em disco (e memória RAM) copiando classes utilizadas por várias aplicações para a pasta lib do Tomcat. Isto não impede que uma aplicação em particular use versões diferente das mesmas classes, se estas classes estiverem dentro do pacote da própria aplicação.

A ordem de busca por classes em uma aplicação web no Tomcat é:

1. Classes do Java SE;
2. Classes do Tomcat (se a aplicação foi configurada como “privilegiada”) e das APIs Java EE;
3. Classes no pacote WAR;
4. Classes na pasta lib do Tomcat.

Então um pacote pode trazer classes que sobrepõem bibliotecas fornecidas pelo Tomcat (ou instaladas em lib pelo administrador), desde que ele não tente sobrepor APIs padrão do Java SE e do Java EE.

## Classloaders do Java x Tomcat

A visibilidade (ou não) entre classes é implementada dentro da JVM por meio de classloaders. Este é um dos recursos da JVM que permite a configuração de sand boxes, ambientes restritos onde é possível executar código potencialmente inseguro. O outro recurso é o security manager, que normalmente não está habilitado no Tomcat e não será visto neste curso.[[16]](#footnote-16)

Os classloaders formam uma hierarquia onde o classloader filho primeiro delega a busca por uma classe para o classloader pai. Apenas o classloader de um pacote WAR viola esta regra, procurando primeiro as classes nele mesmo e somente depois delegando par ao classloader pai.

O Tomcat separa o classloader das classes do próprio servidor em um ramo isolado da hierarquia de classloaders, de modo que aplicações normais não tem acesso às classes do próprio Tomcat. Esta é uma medida para proteger a estabilidade e segurança do ambiente.

Entretanto, um contexto pode ser configurado como “privilegiado” (atributo privileged="true" no elemento <Context>) e assim ter acesso às classes da implementação do próprio servidor. Este é o caso do Manager e do antigo Admin do Tomcat 5.5 e anteriores.

### Configurando pastas adicionais para bibliotecas

No Tomcat 5.5 e anteriores, havia uma separação entre pastas para bibliotecas visíveis tanto aplicações quanto pelo Tomcat (common), visíveis apenas pelas aplicações (shared) ou então apenas pelo Tomcat (server). No Tomcat 6.0, esta mesma separação pode ser configurada no arquivo catalina.properties, em vez de utilizando diretórios separados.

Ou seja, o Tomcat define três classloaders próprios, além dos classloaders da JVM e de cada pacote WAR. Na prática é muito raro que o administrador necessite modificar as configurações destes classloaders, e só irá necessitar acrescentar classes ao classloader common, por exemplo drivers para banco de dados.

Em teoria, classes compartilhadas por várias aplicações, porém não utilizadas pelos componentes do próprio Tomcat, seriam separadas no classloader shared . Entretanto coloca-las no common surte o mesmo efeito na prática, pois como o Tomcat não vai usar estas classes, elas não irão atrapalhar. Por outro lado, colocar classes em shared que deveriam estar em common pode impedir o funcionamento da aplicação, por isso costuma-se acrescentar todas as bibliotecas e JARs de terceiros sempre no commons.

Por default, todas as classes em lib que não sejam de implementação do próprio Tomcat nem de APIs do Java EE são consideradas como parte do common, e o shared fica vazio. Mas alguns administradores não gostam da idéia de se misturar em um mesmo diretório arquivos que fazem parte da instalação do Tomcat e arquivos que foram acrescentados pelo administrador ou desenvolvedor.

Então é possível definir um novo diretório “lib”, que será também parte do classloader common, editando o arquivo catalina.properties como se segue:

* common.loader=${catalina.home}/lib,${catalina.home}/lib/\*.jar,  
  **${catalina.home}/meu-lib/\*.jar**

Note que todos os valores da entrada common.loader devem ser fornecidos em uma única linha, separados por vírgulas.

## Exercícios

Laboratório 1.1. Aplicação com biblioteca externa

(Prática Dirigida)

**Objetivo:** Fazer funcionar uma aplicação que necessita um componente Java Bean fornecido por uma biblioteca em separado

O exemplo deste exercício é uma versão simplificada do exemplo “hoje” utilizado nos capítulos anteriores. Ele contém uma única página JSP, que exibe a data corrente, mas que chama uma classe Java Bean para formatar a data.

O build file fornecido com o exercício gera dois pacotes: um pacote WAR contendo a aplicação, e um pacote JAR contendo o Java Bean. Em seguida realiza usando o Manager, o deployment do pacote WAR:

$ **ant**

...  
empacota:

[jar] Building jar: /home/Vilaça/lab-tomcat/Cap7/Lab1/dist/hoje-lib.war

[jar] Building jar: /home/Vilaça/lab-tomcat/Cap7/Lab1/dist/hoje-bean.jar

instala:

[deploy] OK - Deployed application at context path /hoje-lib

Entretanto, o WAR recém-deployado não leva consigo a biblioteca, o que pode ser comprovado pelo comando jar:

$ **jar tf dist/hoje-lib.war**

META-INF/

META-INF/MANIFEST.MF

index.jsp

WEB-INF/

WEB-INF/classes/

WEB-INF/classes/biblioteca/

WEB-INF/classes/exemplo/

WEB-INF/web.xml

Ao executar a aplicação, pela URL http://127.0.0.1:8080/hoje-lib, o resultado será uma mensagem de erro:

* exception

org.apache.jasper.JasperException: /index.jsp(3,0) The value for the useBean class attribute biblioteca.HojeBean is invalid.

Esta mensagem indica que o compilador de páginas JSP não foi capaz de localizar a classe HojeBean, que foi empacotada em hoje-bean.jar.

Então assuma a identidade de “root”, copie a biblioteca para a pasta lib do Tomcat, e reinicie o servidor. Retorne para seu usuário normal e recarregue a novamente a aplicação, e desta vez o resultado deverá ser a data de hoje formatada como dd/mm/aaaa.

Agora, remova a biblioteca, reinicie novamente o Tomcat e regarregue a página mais uma vez. O resultado esperado é um erro, pois a biblioteca não estará mais disponível para a aplicação, entretanto será exibida uma mensagem de erro diferente:

* exception

javax.servlet.ServletException: java.lang.NoClassDefFoundError: biblioteca/HojeBean

O erro é diferente porque a página já foi compilada antes com sucesso, então o erro ocorreu durante a execução da página.

Laboratório 1.2. Pasta adicional de bibliotecas

(Laboratório)

**Objetivo:** Configurar uma pasta de bibliotecas separada da pasta padrão do Tomcat e que possa ser acessada (escrita) por usuários comuns.

Antes de mais nada, remova a biblioteca que foi acrescentada ao Tomcat no exercício anterior.

No exercício anterior, foi necessário assumir a identidade de “root” para acrescentar uma nova biblioteca ao Tomcat. Isto não é desejável, pois é a política em muitas empresas que desenvolvedores (ou mesmo os administradores do servidor de aplicação) não tenham acesso de escrita aos diretórios de instalação de software.

Então vamos criar uma pasta chamada “meu-lib”, de forma coerente com as convenções do LSB e JPackage, na qual o usuários do grupo “tomcat” tenham permissão de escrita. Ou seja, qualquer usuário com permissão de modificar aplicações instaladas no Tomcat via auto-deploy também terá permissão de modificar as bibliotecas neste diretório.

Primeiro, edite o arquivo catalina.properties conforme as instruções já apresentadas neste capítulo. Em seguida, crie o novo diretório de bibliotecas, configure o dono e permissões, e reinicie o Tomcat:

# mkdir /var/lib/tomcat6/meu-lib

# chgrp tomcat /var/lib/tomcat6/meu-lib

# chmod g+w /var/lib/tomcat6/meu-lib

# ln -s /var/lib/tomcat6/meu-lib/ /usr/share/tomcat6/

# service tomcat6 restart

Agora, copie a biblioteca para a pasta meu-lib, e tenha certeza de que não existe uma outra cópia da biblioteca em lib. Se existir esta outra cópia, ela terá que ser removida e o Tomcat mais uma vez reiniciado.

Então acesse a aplicação e comprove que a biblioteca foi encotrada pelo Tomcat no nosso novo diretório.

Laboratório 1.3. Versões diferentes da mesma classe

(Laboratório)

**Objetivo:** Rodar uma segunda versão da aplicação “hoje” que utiliza uma versão diferente da classe Java Bean, sem afetar o funcionando da versão original da aplicação e da versão antiga da biblioteca.

Desta vez não queremos remover o lixo dos laboratórios anteriores. Muito pelo contrário, queremos que a aplicação hoje-lib.war e a biblioteca hoje-bean.jar estejam disponíveis no Tomcat antes de prosseguir.

Execute o buildfile fornecido com o exemplo, e verifique que desta vez o pacote WAR realmente contém uma biblioteca:

$ **jar tf dist/hoje-lib2.war**

META-INF/

META-INF/MANIFEST.MF

index.jsp

WEB-INF/

WEB-INF/classes/

WEB-INF/classes/biblioteca/

WEB-INF/classes/exemplo/

WEB-INF/lib/

**WEB-INF/lib/hoje-bean.jar**

WEB-INF/web.xml

Como a nova versão da aplicação já foi deployada pelo buildfile, podemos acessa-la pelo navegador e verificar que nela a data corrente é formatada junto com a hora corrente. Verifique também que a versão antiga continua funcionando, e exibindo apenas a hora.

Inspecione o JAR que foi instalado no Tomcat, e o JAR que foi inserido dentro do pacote WAR. Verifique que ambos contém a mesma classe HojeBean. Ou melhor, ambos contém o mesmo nome de classe, com diferentes implementações.

$ **jar tf WEB-INF/lib/hoje-bean.jar**

META-INF/

META-INF/MANIFEST.MF

biblioteca/

biblioteca/HojeBean.class

$ **jar tf /var/lib/tomcat6/meu-lib/hoje-bean.jar**

META-INF/

META-INF/MANIFEST.MF

biblioteca/

biblioteca/HojeBean.class

Então comprovamos que é possível rodar versões diferentes da mesma classe em aplicações diferentes dentro do Tomcat. Não há necessidade de configurar instâncias separadas do servidor para acomodar aplicações com dependências de versões diferentes da mesma biblioteca ou framework.

## Conclusão

Vimos como acrescentar bibliotecas compartilhadas por várias aplicações no Tomcat de forma organizada, sem bagunçar sua instalação do servidor. Também vimos que é possível, graças à forma como o Tomcat e o padrão Java EE fazem uso dos classloaders da JVM.

Questões de Revisão

* Classes que sejam utilizadas por componentes do Tomcat, por exemplo drivers para conexão a bancos de dados, podem ser colocadas dentro do pacote WAR da aplicação?

* Caso uma aplicação inclua no seu pacote WAR uma classe que também existe (com o mesmo nome) em algum pacote JAR na pasta lib do Tomcat, qual versão da classe será utilizada durante a execução da aplicação?

# Acesso a Bancos de Dados

Este capítulo apresenta um dos recursos fundamentais de performance e escalabilidade de um servidor Java EE, o uso de pools de conexões a um banco de dados

Tópicos:

* Introdução ao JDBC
* Configuração de DataSources
* Resources locais x Resources globais

## Introdução ao JDBC

O JDBC (Java DataBase Conectivity) é o padrão do Java EE para acesso a bancos de dados relacionais. Ele define APIs tanto para o desenvolvedor de aplicação quanto para o desenvolvedor do drive que implementa o protocolo nativo do banco de dados.

No meio cliente/servidor, APIs genéricas de acesso a banco de dados como o ODBC do Windows ficaram com uma má fama, mas na verdade os problemas eram muito mais de projeto e implementação ruim destas APIs do que do conceito de API geńerica. Então fique tranqüilo, pois o uso do JDBC será tão eficiente e performático quando as bibliotecas nativas do banco de dados desejado.

O JDBC permite acesso a recursos avançados, como prepared statements, stored procedures, catálogo e BLOBs, sem no entanto tornar o código Java amarrado ao banco de dados utilizado.

É claro, a aplicação irá conter em seu código strings de comandos SQL que não são portáveis. Este é um dos motivos de frameworks ORM como Hibernate e JPA terem se tornado tão populares. Com o framework gerando os comandos SQL otimizados para o banco desejado, a aplicação fica realmente portável para diferentes bancos sem necessidade de modificar ou recompilar o código Java.

### Tipos de drivers JDBC

A especificação JDBC define quatro tipos de bancos de dados. Alguns fornecedores podem oferecer mais do que um tipo de driver para o mesmo banco, mas em geral o que se deseja utilizar é um driver do tipo 4:

1. Tipo 1 é um conversor JDBC para ODBC, que permite o aproveitamento de drivers ODBC nativos do sistema operacional. Não deve ser utilizado exceto quando não houver outra alterativa;
2. Tipo 2 utiliza JNI (Java Native Interface) para acesso à biblioteca nativa de cliente do banco de dados. É a opção preferida dos fornecedores de BDs, pois amarra o ambiente de desenvolvimento e produção à configurações específicas do produto, mas tende a trazer problemas de estabilidade e performance, além da óbvia perda de portabilidade;
3. Tipo 3 é um gatgeway que fica entre a aplicação e o banco. O driver fala com o gateway, e este por sua vez fala com o banco, utilizando um driver nativo ou mesmo um driver JDBC de outro tipo. Foi muito popular quando os próprios fornecedores de BDs não ofereciam seus drivers, pois facilitava o desenvolvimento dos drivers alternativos. Hoje é interessante apenas em produtos de cluster de BD como o Sequoia (antigo C-JDBC, http://c-jdbc.ow2.org/).
4. Tipo 4 é um driver implementado inteiramente em Java, sem depender de nenhuma biblioteca nativa. Então o próprio driver é portável, e esta opção tende a ter a melhor performance e estabilidade dentro de um servidor de aplicações Java EE.

Desenvolvedores Java SE estão habituados a incluir o driver JDBC em suas aplicações e abrir diretamente conexões ao BD utilizando a classe DriverManager do JDBC.

Entretanto, em servidores de aplicação Java EE o recomendado é configurar um DataSource gerenciado pelo servidor de aplicações, e utilizar este objeto para abrir as conexões ao banco. Então o driver JDBC é instalado como uma biblioteca no servidor de aplicações, em vez de ser colocado no pacote da aplicação.

## DataSources Java EE

Os termos “DataSource Java EE” e “DataSource JDBC” são utilizados como sinônimos por vários produtos, e ambos se referem ao mesmo componente padrão do Java EE, que é configurado e gerenciado no servidor de aplicações como forma de disponibilizar para as aplicações o acesso a um banco de dados relacional.

Na verdade um DataSource Java EE é parte do padrão JCA (Java Connector Architecture) que prevê uma arquitetura padronizada para a implementação de conectores para diferentes tipos de EIS, que podem ser desde sistemas de mainframe até pacotes de ERPs. O JCA prevê a gerência de recursos do SO e rede, além da integração com a segurança e transações gerenciadas pelo servidor de aplicações.

O Tomcat não fornece o JCA, mas suporta os DataSources e alguns outros tipos de fábricas de conexões, por exemplo para acesso a servidores de e-mail Internet. É até possível configurar o Tomcat para suportar acesso a filas JMS e EJBs hospedados em servidores de aplicações Java EE completos, mas estes tópicos estão além do escopo desta apostila.

### DataSources e o JNDI

Uma aplicação dentro do servidor de aplicações acessa um DataSource por meio de uma busca no JNDI. Mesmo no Java EE 5, as anotações como @Resource apenas realizam a busca no JNDI.

Para os desenvolvedores que ainda usam o “modo Java SE” de se conectar a um banco de dados, a listagem 1.1 fornece um exemplo de como seria o código recomendado para uma aplicação Java EE.

**Listagem 1.1** – código Java para conextar a um BD usando um DataSource

* InitialContext ctx = new InitialContext();

1. ds = (DataSource)ctx.lookup("java:comp/env/jdbc/Contatos");

Connection con = ds.getConnection();

Onde “jdbc/Contatos” é o nome local da aplicação para o DataSource. O padrão Java EE também especifica que uma aplicação declare em seu descritor de deployment os nomes dos DataSources que serão acessados via JNDI ou anotações, confirme a listagem 1.2.

**Listagem 1.2** – declaração de um nome local para DataSource no descritor web.xml

* <resource-ref>

1. <res-ref-name>jdbc/Contatos</res-ref-name>
2. <res-ref-type>javax.sql.DataSource</res-ref-type>
3. <res-auth>Container</res-auth>

</resource-ref>

A definição no descritor web.xml define sempre um nome local, que é prefixado por “java:comp/env”. Por isso o código Java tem que incluir o prefixo, mas este é omitido no descritor.

Seria possível omitir o prefixo “java:comp/env” também no código Java e fazer a busca pelo nome global do DataSource, coisa que sem dúvida é mais simples para o desenvolvedor, porque elimina a necessidade de se declarar o nome local no descritor web.xml e também a necessidade de se mapear este local para um nome global na configuração do servidor de aplicações.

Por outro lado, utilizar nomes globais na aplicação reduz a flexibilidade do administrador, pois se duas aplicações utilizarem o mesmo nome global para referenciar bancos de dados diferentes, não será possível colocar as duas lado-a-lado na mesma instância do servidor de aplicações. Uma delas terá que ser modificada para usar um nome diferente.

O Tomcat não exige a declaração do <resource-ref> no descritor padrão, e neste caso ele viola o padrão Java EE. Muitos desenvolvedores se aproveitam deste fato e depois tem dificuldades se precisarem rodar a aplicação em outro servidor de aplicações.

### Porque usar DataSources

Conexões a bancos de dados são recursos caros computacionalmente, tanto para a aplicação cliente quanto para o próprio banco de dados. Pode ser inviável manter várias conexões abertas em aplicações com grande quantidades de usuários.

Um DataSource aproveita o fato de que, em um dado momento, a maioria dos usuários estão ociosos, então não estão utilizando suas conexões. Então seria possível utilizar uma quantidade menor de conexões, compartilhadas entre vários usuários. Para isto, o DataSource mantém um pool de conexões com o banco de dados.

Outra vantagem de se usar um DataSource é que abrir e fechar conexões conforme a demanda (a cada página requisitada) é demorado, e iria prejudicar o tempo de resposta do usuário. Então a conexão é mantida aberta entre uma página e outra.

A aplicação é programada como se estivesse abrindo e fechando conexões a cada requisição HTTP, entretanto o DataSource não fecha realmente a conexão, apenas a devolve para o pool, onde ela fica disponível para uso pela próxima requisição.

### Pools de Conexão

Um pool de conexões é configurado com mínimo de conexões que são abertas na inicialização do servidor de aplicações. A idéia é poder atender imediatamente aos usuários, sem que eles tenham que aguardar pelo tempo de se abrir as primeiras conexões do pool.

O pool também é configurado com um máximo de conexões, de modo que nem o servidor de aplicações nem o banco de dados fiquem sobrecarregados com uma quantidade de conexões acima da sua capacidade de atendimento.

Caso o pool já esteja no seu máximo, as requisições ficam enfileiradas, aguardando que uma conexão esteja disponível para continuar o processamento.

Espera-se que aplicações bem escritas demorem o máximo possível a abrir conexões, ou seja, a retira-las do pool, e as feche o quanto antes, devolvendo ao pool para que possam ser usadas por outras requisições.

Por exemplo, a aplicação pode esperar para abrir a conexão somente depois que tenha completado a validação de dados de entrada do usuário, e consultar todas as informações do banco antes de começar a gerar a página HTML de resposta.

O servidor de aplicações é o responsável pelo gerenciamento do pool de conexões associado a um DataSource, decidindo quando realmente abrir e fechar conexões, ou se é necessário reciclar as conexões.

## Configurando DataSources no Tomcat

DataSources são configuradas no Tomcat por elementos <Resource> aninhados a uma definição de contexto, ou então no server.xml dentro de <GlobalNamingResources>.

Caso a definição ocorra dentro de um contexto, ela é associada a um nome local, que só é visível para o próprio contexto que definiu o DataSource.

Mas, caso a definição seja feita no server.xml como um recurso global, o mesmo DataSource (e conseqüentemente as mesmas conexões em pool) poderão ser compartilhadas por várias aplicações diferentes.

O administrador deverá pesar o potencial desperdício de conexões, caso vários contextos tenham seus próprios pools para acessar o mesmo banco de dados, contra o risco de erros em uma aplicação interferirem com o funcionamento normal das demais, caso elas compartilhem um mesmo pool global.

Esta é uma decisão que deve ser tomada caso-a-caso, de acordo com a qualidade de código e criticidade das aplicações envolvidas. Mas, qualquer que seja a decisão (pool global ou por contexto) sugerimos que os DataSources sejam definidos globalmente e que o desenvolvedor sempre faça as buscas pelos nomes locais. O motivo é dar liberdade ao administrador para configurar um cenário ou outro, sem que haja necessidade de modificações no código Java das aplicações.

A definição do DataSource, ou melhor, do elemento <Resource>, segue a mesma sintaxe qualquer que seja o local onde foi realizada. Temos um exemplo na listagem 1.3.

**Listagem 1.3** – definição de um DataSource para o Tomcat

* <Resource name="jdbc/Contatos" auth="Container"

1. type="javax.sql.DataSource"
2. maxActive="10" maxIdle="2" maxWait="-1"
3. username="tomcat" password="senha"
4. driverClassName="org.postgresql.Driver"
5. url="jdbc:postgresql://127.0.0.1/contatos"

/>

Este exemplo especifica uma conexão a um banco de dados PostgreSQL instalado no mesmo computador que roda o Tomcat. Normalmente o login e senha de acesso ao banco estão pré-fixados na definição de DataSource, o que vai contra o hábito de muitos desenvolvedores que, em ambiente cliente/servidor, repassavam para o banco de dados as credenciais do usuário.

É possível configurar o DataSource de modo que a aplicação tenha que fornecer login e senha para o banco, entretanto ter cada conexão com usuários diferentes anula as vantagens de performance escalabilidade do pool de conexões. Também complica a utilização da infra-estrutura para autenticação e autorização oferecida pelo próprio servidor de aplicações, que será vista no próximo capítulo.

Se o desenvolvedor está preocupado com os triggers de auditoria, o ambiente Java EE fornece recursos ao desenvolvedores para obter funcionalidade similar no lado da aplicação, o que tende a ser bem mais útil, por gerar informações de alto nível sobre operações de negócios, em vez de informações de baixo nível sobre alterações em tabelas.

### Links para recursos globais

Se o DataSource foi definido como um recurso global, em vez de como um recurso local no contexto da aplicação, é necessário configurar um nome local que seja um link para o nome global do recurso no JNDI.

O link é definido pelo elemento <ResourceLink>, exemplificado na listagem 1.4.

**Listagem 1.4** – definição de um link local para um DataSource global

* <ResourceLink name="jdbc/ContatosLocal"

1. global="jdbc/ContatosGlobal"
2. type="javax.sql.DataSource"

/>

O atributo name é o nome local, utilizando pela aplicação para realizar buscas, enquanto que o atributo global fornece o nome global definido no server.xml. Ambos são obrigatórios.

## Exercícios

Laboratório 1.1. DataSource local

(Prática Dirigida)

**Objetivo:** Deployar e executar com sucesso um aplicação que acessa um banco de dados

Para este exercício será utilizado um banco de dados PostgreSQL previamente configurado e populado no computador do instrutor. Tome nota dos parâmetros para acesso ao banco de dados, que serão fornecidos pelo instrutor:

Endereço IP: ...........................................

Nome do Banco: ...........................................

Login: ...........................................

Senha: ...........................................

Em primeiro lugar, é necessário instalar o driver JDBC do PostgreSQL, que é fornecido pelo Fedora no pacote postgresql-jdbc. Instale este pacote usando yum ou rpm, caso não esteja disponível na sua estação, seguindo as instruções fornecidas pelo instrutor. Assim como no capítulo 1, pedimos que os alunos não façam instalação direta nem downloads de programas da Internet, para não prejudicar outras salas de aula.

A instalação deste pacote fornece o arquivo /usr/share/java/postrgresql-jdbc.jar que é o driver em si. Temos que disponibiliza-lo para o Tomcat, o que significa copia-lo para a pasta lib. Entretanto, nós podemos fazer melhor do que copiar:

1. Use um link simbólico, para evitar duplicação de arquivos, e também para garantir que, quando o driver for atualizado pela distribuição, a versão utilizada pelo Tomcat também o seja;
2. Criamos no capítulo 7 um diretório de bibliotecas em separado do diretório default do Tomcat. Vamos usar este diretório.

Em seguida, reinice o Tomcat para que as classes do driver JDBC estejam disponíveis para o servidor.

# ln -s /usr/share/java/postgresql-jdbc.jar /usr/share/tomcat6/meu-lib/

# service tomcat6 restart

Agora sim, entre em Cap1/Lab1 mas, ANTES de executar o buildfile, modifique o arquivo META-INF/context.xml para incluir os parâmetros corretos de conexão ao banco de dados PostgreSQL no computador do instrutor. Observe que foi definido um DataSource no contexto da aplicação.

Caso você tenha deployado o exemplo sem corrigir os parâmetros de conexão na definição do DataSource, ou errou na digitação dos dados e precisa deployar o pacote corrigido, use o algo “desinstala” do buildfile.

Finalmente, execute a aplicação para ver uma relação de e-mails de personalidades do mundo Java e Open Source.

Observe que o descritor padrão web.xml traz a definição do <resource-ref> para o DataSource comentada, mesmo assim a aplicação funciona. Você pode experimentar descomentar a definição, para comprovar que sua presença não afeta a aplicação.

Depois, experimente mudar o nome da referência em <res-ref-name> para algo diferente do nome do DataSource e veja que agora a aplicação não pode mais ser deployada com sucesso.

Laboratório 1.2. DataSource global

(Laboratório)

**Objetivo:** Executar uma variação do exercício anterior, que utiliza um DataSource global do Tomcat.

Primeiro, o DataSource global tem que ser configurado no Tomcat, pela edição do arquivpo de configuração server.xml. O exercício já inclui o arquivo inserir-no-server.xml para poupar o esforço de digitação do aluno.

Reinicie o JBoss, e faça o deployment da aplicação utilizando o buildfile fornecid com o exemplo. Acesse a aplicação, para receber o erro:

* exception

javax.servlet.ServletException: javax.naming.NameNotFoundException: Name jdbc is not bound in this Context

Ele ocorre porque ainda não foi configurado o link do nome local para o nome global do DataSource. Então edite o arquivo META-INF/context.xml e veja que ele já traz a definição do link, porém comentada.

Descomente a definição e use o ant para fazer um novo deploy da aplicação. Agora sim ela deverá funcionar corretamente.

## Conclusão

Agora sim o aluno está apto a colocar aplicações “reais” em seu servidor Tomcat, pois aprendemos a configurar DataSources para acesso a bancos de dados relacioais. Também vimos como compartilhar pools de conexões entre várias aplicações.

Questões de Revisão

* Verdadeiro ou falso: um único driver JDBC permite o acesso a qualquer banco de dados relacional, porque o Java EE define um protocolo universal para BD.

* Seria possível deployar no mesmo sevidor Tomcat duas vezes a mesma aplicação, sendo que cada instância utiliza um banco de dados diferente? Este seria um cenário típico de uma empresa de SaS (Softwware as a Service) que fornece um mesmo sistema para vários clientes diferentes. Ou será que a empresa seria obrigada a configurar várias instâncias do servidor Tomcat, uma para cada banco de dados?

Questões de Revisão

* Cite alguns benefícios do uso do Ant pelo administrador de um servidor Tomcat.

* Verdadeiro ou falso: O uso do Ant para deployment de aplicações só é possível caso o auto-deploy esteja habilitado no <Host>.

# Autenticação HTTP

Este capítulo apresenta os recursos de segurança declarativa do Java EE e como eles podem ser utilizados para proteger aplicações e o próprio Tomcat

Tópicos:

* Segurança Declarativa do Java EE
* Autenticação HTTP
* Single-Sign On

## Segurança Declarativa do Java EE

O Java EE define todo um framework de autenticação, que engloba o código da aplicação, do servidor de aplicações e até fornecedores de plug-ins de segurança que podem suportar desde um servidor de diretório LDAP até hardware de biometria. Os plug-ins de segurança são definidos pela especificação JAAS.

A idéia é que o desenvolvedor não escreva nenhum código para autenticar usuários ou autorizar o acesso às páginas da aplicação. Este controle é delegado para o servidor de aplicações, que poderá faze-lo de forma mais flexível e segura.

O papel do desenvolvedor é apenas definir regras de controle de acesso a páginas (URLs) da aplicação baseadas em roles. Por isto o framework do Java EE é chamado de segurança declarativa, porque o desenvolvedor declara no descritor de deployment web.xml as regras de controle de acesso, em vez de escrever código (comandos if) para validar estas regras.

Já na configuração do servidor de aplicações (no caso do Tomcat, na configuração de contexto) é definida uma base de identidade, utilizada para validar a identidade dos usuários e obter a lista de roles que o usuário corrente possui.

### Resource Collections do Java EE

As regras de seguranças são definidas por elementos <security-contraint>, que contém um <resource-collection> descrevendo as páginas restritas pela regra e as condições para que o acesso seja permitido.

Podem haver vários <security-contraint> na mesma aplicação. Cuidado para que não haja interseção entre eles, pois neste caso o padrão Java EE não define qual regra será efetiva. Páginas que não se encaixem em nenhum <security-constraint> são consideradas públicas.

A restrição de acesso mais usual é o <auth-constraint> indicando que o acesso será permitido apenas aos roles listados pelo elemento. A listagem 1.1 fornece um exemplo de <security-contraint> pela qual as páginas páginas que se encaixarem no padrão “/checkout/\*” só serão acessíveis a usuários com o role “usuario”.

**Listagem 1.1** – exemplo de <security-constraint> no descritor web.xml

* <security-constraint>

1. <web-resource-collection>
2. <web-resource-name>Finalização da compra</web-resource-name>
3. **<url-pattern>/checkout/\*</url-pattern>**
4. </web-resource-collection>
5. <auth-constraint>
6. **<role-name>usuario</role-name>**
7. </auth-constraint>

</security-constraint>

Note que as URLs nos <url-pattern> são relativas à raiz do pacote WAR e não à raiz do servidor web. Então elas permanecem válidas independente do nome de contexto com que a aplicação for deployada.

Os recursos padrão de segurança do Java EE prevêem ainda restrição baseada em métodos HTTP, que podem ser bastante úteis para prevenir alguns tipos de ataques de CSS e de injeção de scripts, ou baseadas no uso de conexão SSL.

A listagem 1.2 fornece um exemplo de <security-contraint> pela qual as páginas que se encaixarem no padrão “/relatorios/\*” só serão acessíveis pelo método GET e nunca via POST.

**Listagem 1.2** – exemplo de <security-constraint> no descritor web.xml

* <security-constraint>

1. <web-resource-collection>
2. <web-resource-name>Relatórios sumários</web-resource-name>
3. <url-pattern>/relatorios/\*</url-pattern>
4. **<http-method>POST</http-method>**
5. </web-resource-collection>

</security-constraint>

Já a sintaxe para exigir conexões SSL será vista no capítulo sobre configurações de rede.

### Segurança Programática

O modelo de segurança declarativa do Java EE, baseado em roles e URLs não atende a totalidade das necessidades das aplicações.

Isto leva muitos desenvolvedores ao erro de desenvolverem seus próprios sistemas de autenticação e controle de acesso, que geralmente acabam sendo inflexíveis ou contendo vulnerabilidades graves.

Não há necessidade destes sistemas de segurança ad-hoc porque o Java EE prevê APIs para obter informações sobre o usuário autenticado e seus roles, o que é suficiente para a grande maioria dos casos. Então o desenvolvedor pode usar a segurança do Java EE e complementar com lógica customizada quando for necessário.

A classe ServletRequest fornece o método:

* isSecure() indica se a conexão corrente é ou não criptografada.

Já a sua subclasse HttpServletRequest acrescenta os métodos:

* getRemoteUser() retorna o nome de login autenticado pelo protocolo HTTP;
* getUserPrincipal() retorna o nome de login autenticado pelo container, seja pelos mecanismos do HTTP, pelo form-based da API de servlet ou outro mecanismo baseado em JAAS;
* isUserInRole() indica se o usuário corrente pertence ao role especificado.

Com estes poucos métodos é possível codificar regras de negócio baseadas na identidade do usuário, ou então eliminar de uma página JSP links e botões correspondentes a funcionalidades que o usuário corrente não tem acesso.

## Métodos de autenticação HTTP

Como aplicações web em geral são interativas e envolvem um cliente remoto, temos ainda dois problemas:

1. Como transferir as credenciais do usuário do cliente para o servidor;
2. Qual a aparência da interface com o usuário para requisitar estas credenciais.

O protocolo HTTP prevê três mecanismos de autenticação que cuidam desses dois problemas:

* BASIC envia a senha do usuário em claro (codificada em UUENCODE);
* DIGEST envia um hash da senha;
* CLIENT-CERT exige a instalação de um certificado digital X-400 no navegador do cliente, devidamente assinado por uma CA, e usa este certificado como prova da identidade do usuário.

A senha fornecida pelo usuário no BASIC ou DIGEST é mantida em memória pelo navegador, que a retransmite a cada página requisitada. Isto ocorre porque o protocolo HTTP é stateless (sem memória), então para o navegador web não existe um momento de login e outro de logoff. Cada página individual representa um logon e logoff.

### Limitações dos métodos HTTP

O BASIC é o único obrigatório em todos os servidores e nevagadores web, e é seguro apenas se usado em conjunto com SSL / TLS. Versões recentes do Firefox e IE suportam DIGEST, que entretanto não é tão seguro assim por si só

O CLIENT-CERT é pouco usado na prática pelas dificuldades logísticas de distribuição dos certificados digitais para os usuários e instalação nas suas estações. Ele é viável apenas para um conjunto restrito de computadores sob uma gerência centralizada, mas também é muito popular para comunicação entre programas (B2B, Business-to-Business, ou Web Services).

Em nenhum dos mecanismos previstos pelo HTTP a aplicação tem controle sobre a aparência da janela de login exibida pelo navegador. Só isto já é suficiente para inviabilizar seu uso na maioria dos sistemas corporativos.

Além disso, o fato de não existir um logout explícito impede que se implemente login por inatividade e permite várias estratégias de roubo de identidade por sites maliciosos.

### Regras de autenticação no descritor web.xml

O descritor padrão da aplicação deve relacionar todos os roles reconhecidos pela aplicação, e indica qual o mecanismo de autenticação adotado. Veja o exemplo de sintaxe na listagem 1.3.

**Listagem 1.3** – exemplo de declaração de roles e  
mecanismo de autenticação no web.xml

* <security-role>

<role-name>usuario</role-name>

</security-role>

<security-role>

<role-name>administrador</role-name>

</security-role>

1. <login-config>
2. **<auth-method>BASIC</auth-method>**
3. <realm-name>
4. Exemplo de Segurança J2EE
5. </realm-name>

</login-config>

O texto do <realm-name> normalmente é exibido pelo navegador na sua janela de login e representa o máximo de interferência da aplicação sobre a interface com o usuário para login gerada pelo navegador.

### Autenticação FORM

Como os mecanismos de autenticação previstos pelo HTTP não fornecem à aplicação controle sobre a aparência da janela de login, o Java EE fornece como alternativa um quarto mecanismo, chamado de “form-based” ou FORM.

O mecanismo “form-based” está atrelado à sessão HTTP, que é por sua vez outra abstração fornecida pelo container web e inexistente no protocolo HTTP propriamente dito. A sessão HTTP utiliza cookies para acompanhar um usuário e assim ter uma noção de entrada e saída da aplicação. O mecanismo FORM utiliza a sessão para registar a identidade do usuário corrente.

O cookie de sessão é seguro mesmo sem o uso de SSL / TLS, e resistente à replay e previsão. Mas a senha ainda é transmitida em texto claro durante o login – portanto verifique se o desenvolvedor tomou o cuidado de colocar a página de login em https!

A listagem 1.4 apresenta um exemplo da sintaxe para declarar a autenticação FORM no descritor padrão da aplicação.

**Listagem 1.4** – exemplo de declaração do mecanismo de autenticação  
FORM no web.xml

* <login-config>

1. **<auth-method>FORM</auth-method>**
2. <form-login-config>
3. <form-login-page>/login.jsp</form-login-page>
4. <form-error-page>/loginInvalido.jsp</form-error-page>
5. </form-login-config>

</login-config>

Note que agora o elemento <login-config> tem que especificar as páginas de login e de erro de login (que não devem ser páginas restritas por um <security-constraint>!)

As definições de <security-role> e <security-constraint> no web.xml permanecem inalteradas com o uso da autenticação FORM.

### Logout por inatividade

Como já foi comentado, só é possível fazer explicitamente um logout com a autenticação FORM. Basta encerrar a sessão HTTP, chamando HttpSession.invalidate().

Mas nada obriga o usuário a fazer o logout, o usuário pode simplesmente ir para outro site ou então fechar seu navegador, e o servidor web não toma conhecimento disso.

Então é importante que sessões HTTP (e conseqüentemente os logins) sejam invalidados por inatividade. O padrão pela API de Servlets é de 30 minutos, mas a listagem 1.5 apresenta um exemplo de como modificar este tempo para apenas 10 minutos.

**Listagem 1.5** – exemplo de declaração de time-out de logoff por inatividade  
para o web.xml

* <session-config>

<session-timeout>10</session-timeout>

</session-config>

O tempo limite de inatividade pode ainda ser configurado programaticamente durante a execução da aplicação pelo método HttpSession.setMaxInactiveInterval(). Então é possível, em regiões críticas de uma aplicação, diminuir este tempo e depois aumenta-lo.

## A Base de Identidade default do Tomcat

A configuração default de instalação do Tomcat já define no server.xml uma base de identidade, que armazena as definições de usuários e roles no arquivo conf/tomcat-users.xml. Nós inclusive já editamos este arquivo, no capítulo 3, quando ativamos o Manager.

É possível acrescentar quantos usuários se desejar, e cada usuário pode ter vários roles separados por vírgulas. A listagem 1.6 apresenta um exemplo de como definir usuários “chefe” e “fulano”, respectivamente com os roles “administrador” e “usuario”.

**Listagem 1.6** – exemplo de definição de usuários

* <tomcat-users>

1. <user username="tomcat" password="tomcat" roles="manager"/>
2. **<user username="chefe" password="boss" roles="administrador"/>**
3. **<user username="fulano" password="testando" roles="usuario"/>**

</tomcat-users>

O Tomcat deve ser reiniciado para que as mudanças nos usuários sejam reconhecidas. No próximo capítulo, veremos maneiras mais flexíveis de definir usuários e roles, utilizando bancos de dados e diretórios LDAP.

## Single Sign-On

Segundo o padrão Java EE, o servidor de aplicações deve tratar de forma independente a autenticação para cada aplicação / contexto. O Tomcat, em sua configuração padrão, obedece fielmente a esta regra, mesmo quando todas as aplicações estejam seja utilizado uma mesma base de identidade.

Sites mais complexos podem ser compostos por várias aplicações e neste caso seria interessante fazer o login uma única vez para todo o container web. Isto é possível se for ativada a válvula de “single sign-on”, que já aparece comentada no arquivo server.xml padrão do Tomcat.

A listagem Listagem 1.7 apresenta a sintaxe desta válvula, que deve ser acrescentada aninhada ao elemento <Host>.

**Listagem 1.7** – válvula de single sign-on

* <Valve className="org.apache.catalina.authenticator.SingleSignOn" />

Cuidado pois este não é um single sign-on de rede, mas apenas para aplicações deployadas dentro de uma mesma instância do Tomcat, ou então dentro de um mesmo cluster de servidores.

## Exercícios

Laboratório 1.1. Aplicação com autenticação BASIC

(Prática Dirigida)

**Objetivo:** Testar uma aplicação que use autenticação BASIC e configurar usuários com diferentes níveis de acesso dentro dela.

O exemplo deste exercício contém uma série de páginas JSP, organizada de tal modo que a página inicial seja pública, mas os links que partem dela sejam restritos a usuários com roles diferentes.

Eis a estrutura de páginas do exemplo:

* basic.war
* index.jsp (página pública)
* erro403.jsp (página pública)
* admin (páginas restritas ao role “administrador”)
* index.jsp
* usuarios (páginas restritas ao role “usuario”)
* index.jsp

Será utilizada a base de identidade padrão do Tomcat. Sua tarefa é criar os usuários (nome/senha) “fulano”/“testando” e “chefe”/“boss”, respectivamente com os perfis “usuario” e “administrador”, seguindo as instruções já apresentadas neste capítulo.

O exemplo já inclui um buildfile para empacotar e compilar a aplicação. Então rode o ant e acesse a aplicação como http://127.0.0.1/base.

Escolha o link “Páginas dos usuários” e faça o login como “fulano” senha “testando”. Note que o nome exibido na janela de login (“Exemplo de Segurança J2EE”) provém do elemento <login-config> no web.xml da aplicação.

O resultado será a página contendo o texto “página dos usuários”. É possível voltar para a página inicial e dela para a página de usuários sem ter que digitar novamente a senha.

Mas a tentativa de acessar a página dos administradores irá gerar uma página dizendo “Você não tem acesso à página solicitada”. Observe que a página de erro de acesso é uma página da aplicação (erro404.jsp) e que obviamente não pode estar em uma área protegida.

Para testar o usuário “chefe” e acessar a “página dos administradores”, será necessário fechar TODAS as janelas e abas do navegador. Faça isso, retorne à aplicação, escolha o link “Páginas dos Administradores” e então faça o login como “chefe” senha “boss”.

Note que o “chefe” não tem acesso às páginas dos usuários, porque ele não tem o role “usuario” -- vamos corrigir isso!

Edite o arquivo tomcat-users.xml e acrescente no usuário “boss” o role “usuário”. O arquivo deve ficar assim:

* <tomcat-users>

1. <user username="tomcat" password="tomcat" roles="manager"/>
2. <user username="chefe" password="boss" **roles="administrador,usuario"/>**
3. <user username="fulano" password="testando" roles="usuario"/>

</tomcat-users>

Reinicie o Tomcat, e confirme que agora o “chefe” pode navegar tanto nas páginas dos usuários quanto nas páginas dos administradores. Não é necessário fechar o navegador nem se logar novamente, basta reiniciar o Tomcat.

Laboratório 1.2. Aplicação com autenticação FORM

(Laboratório)

**Objetivo:** Testar uma aplicação que use autenticação FORM e configurar usuários com diferentes níveis de acesso dentro dela.

O exemplo deste exercício é nada mais do que o exemplo do exercício anterior modificado para autenticação FORM. Então não é necessário modificar as definições de usuários, basta usar o buildfile para realizar o deployment da aplicação e navegar ora como “fulano” e ora como “chefe”.

Observe que, ao contrário do exemplo anterior, a página inicial deste exemplo contém um link “logoff”. É possível usa-lo para mudar de usuário sem fechar as janelas do navegador.

Laboratório 1.3. Singe Sign-On

(Laboratório)

**Objetivo:** Configurar o Tomcat para que seja possível mudar de aplicação sem ter que fornecer login e senha novamente.

Acrescente o <Valve> de single sign-on conforme as instruções já apresentadas na apostila, e reinicie o Tomcat.

Entre primeiro em “basic”, e faça o login como “fulano”. Depois entre em “form” e veja é agora possível acessar as páginas do usuário sem passar pela página de login.

Entetanto, o fato das aplicações estarem configuradas para mecanismos de autenticação HTTP diferentes irá gerar problemas. Para mostrar isso, faça o logoff na aplicação “form”, e veja que daí em diante é necessário fazer novo login enquanto dentro desta aplicação.

Não faça o login em “form”, ou então faça novo logoff, e então mude para a aplicação “basic”. Observe que será concedido o acesso às páginas do usuário sem que o navegador exiba sua janela de login.

O motivo desta incoerência é que o navegador não toma conhecimento do logoff registrado pelo Tomcat, e mantém sua senha em memória para o BASIC. Então, ao voltar para a aplicação com autenticação BASIC, o navegador faz automaticamente um novo login, que com o single sign-on será válido também para a aplicação configurada em FORM.

Portanto, usar o single sign-on exige que todas as aplicações usem autenticação FORM.

Ao final deste exercício, não se esqueça de comentar novamente a válvula de single sign-on no server.xml.

## Conclusão

Neste capítulo conhecemos os recursos do Java EE e do HTTP para autenticação e controle de acesso, utilizando a base de identidade default do Tomcat. No próximo capítulo vamos aprender a configurar outras bases de identidade.

Questões de Revisão

* O protocolo HTTP define como realizar autenticação usando dispositivos biométricos, por exemplo scanners de digitais?

* O que acontece se a página de login configurada para o mecanismo FORM estiver dentro de um <security-constraint> que exija um determinado role?

# Bases de Identidade (Realms) do Tomcat

Neste capítulo, continuamos a apresentação dos recursos de segurança do Java EE e do Tomcat, aprendendo a configurar diferentes tipos de bases de identidade.

Tópicos:

* Armazenamento de senhas criptografadas
* Autenticação com BD
* Autenticação com LDAP

## Realms do Tomcat

Uma base de identidade no Tomcat é chamada de Realm. O Tomcat fornece vários tipos de Realms, dos quais os principais são:

1. UserDatabaseRealm (arquivo tomcat-users.xml)
2. DatasourceRealm (banco de dados relacional)
3. JNDIRealm (diretório LDAP)
4. JAASRealm (usa LoginModules padrão JAAS)

Realms podem ser configurados em qualquer nível da hierarquia Engine / Host / Context, e são definidos por elementos <Realm> no server.xml ou no descritor proprietário context.xml do pacote WAR.

Até o momento utilizamos apenas o realm default que vem pré-configurado no Tomcat. Sua definição está no server.xml aninhado ao <Engine>, então qualquer aplicação que não declare um realm diferente será automaticamente vinculada ao realm padrão. A listagem 1.1 apresenta a definição deste realm default:

**Listagem 1.1** – definição do realm default do Tomcat em server.xml

* ...

1. <Engine defaultHost="localhost" name="Catalina">
2. ...
3. **<Realm className="org.apache.catalina.realm.UserDatabaseRealm"**
4. **resourceName="UserDatabase"/>**
5. ...
6. <Host name="localhost" appBase="webapps"

...

O UserDatabaseRealm necessita de um recurso JNDI que é um banco de dados em memória, inicializado à partir de um arquivo XML. A definição deste recurso aparece antes do próprio realm no server.xml, como ilustra a listagem 1.2:

**Listagem 1.2** – definição do realm default do Tomcat em server.xml

* ...

1. <GlobalNamingResources>
2. ...
3. <Resource auth="Container"
4. name="UserDatabase"
5. type="org.apache.catalina.UserDatabase"
6. **pathname="conf/tomcat-users.xml"**
7. factory="org.apache.catalina.users.MemoryUserDatabaseFactory"/>
8. ...
9. </GlobalNamingResources>

...

Então seria possível definir outros realms, baseados em arquivos XML diferentes, mas a impossibilidade de se modificar as definições de usuários e roles neste arquivo sem reiniciar o Tomcat inviabiliza seu uso em qualquer aplicação real.

Por isso veremos neste capítulo os realms para banco de dados relacional e diretório LDAP, que são os mais usuais. Mas, antes de ver estes realms, uma feature importante disponível na maioria dos realms:

### Armazenando senhas criptogradas

As senhas no arquivo tomcat-users.xml estão armazenadas em texto claro, o que não é nada recomendado. Mesmo que o arquivo esteja protegido a nível de sistema operacional contra acesso por usuários que não o administrador, a maioria das pesoas decora um pequeno conjunto de senhas que são utilizados no seu e-mail pessoal, lojas on-line e até cartões bancários.

Então, por questões de privacidade, estas senhas tem que ser protegidas do próprio administrador. É por isso que as senhas do sistema operacional, por exemplo no arquivo /etc/shadow, são salvas em forma criptografada.

Mas a criptografia utilizada em senhas é baseada em algoritmos de hash, também chamados de algoritmos de digest, que são em essência funções matemáticas sem inversa. Caso contrário, seria fácil obter as senhas originais à partir da sua versão criptografada.[[17]](#footnote-17)

A maioria dos realms do Tomcat suporta o atributo digest, que especifica o algoritmo de hash que deve ser executado sobre a senha recebida do navegador (via BASIC ou FORM) e o resultado comparado com o texto armazenado na base de identidade.

Então, para que o arquivo tomcat-users.xml passe a armazenar senhas criptografadas segundo o popular algoritmo MD5, a definição do seu realm teria que ser editada confirme a listagem listagem 1.3.

**Listagem 1.3** – modificando o realm default para armazenar senhas criptografadas

* <Realm className="org.apache.catalina.realm.UserDatabaseRealm"

resourceName="UserDatabase" **digest="MD5"** />

O problema agora é gerar as senhas criptografadas para armazenar no arquivo tomcat-users.xml (e mais adiante no banco de dados). O Tomcat fornece uma classe utilitária chamada org.apache.catalina.realm.RealmBase que pode ser executada na linha de comando para criptografar a senha, mas a linha de comando necessária para executa-la é bem grande, e é simples fazer o mesmo chamando diretamente a classe java.security.MessageDigest padrão do Java SE, portanto será essa a abordagem nos exercícios deste capítulo.

## Autenticando contra um Banco de Dados

O DataSourceRealm obtém usuários, senhas e roles de uma DataSource JDBC e é bastante flexível. É possível configurar o nome das tabelas e das colunas consultadas, de modo que é possível utilizar praticamente qualquer banco de dados de usuários legado. Na pior das hipóteses, basta definir uma view para facilitar a obtenção das informações pelo Tomcat.

A listagem 1.3 apresenta um exemplo de realm para banco de dados. O exemplo utiliza o mesmo DataSource do capítulo sobre acesso a banco de dados, e já pressupõe que a senha seja armazenada criptografada no banco de dados.

**Listagem 1.3** – exemplo de DataSourceRealm

* <Realm className="org.apache.catalina.realm.DataSourceRealm"

1. dataSourceName="jdbc/ContatosLocal" localDataSource="true"
2. digest="MD5"
3. userTable="usuarios" userNameCol="login" userCredCol="senha"

userRoleTable="grupos" roleNameCol="role"/>

Atenção ao atributo localDataSource. Ele indica se o realm irá considerar o nome do DataSource (atributo dataSourceName) como sendo um nome local ou global no diretório JNDI do Tomcat.

## Autenticando contra um diretório LDAP

O JNDIRealm[[18]](#footnote-18) permite a autenticação contra serviços de diretório LDAP, que incluem softwares livres populares como o OpenLDA e o Fedora Directory Server, além de produtos proprietários como o Novell NDS e Microsoft Active Directory.

A tecnologia LDAP se firmou como a preferencial para bases de identidade corporativas, porque é padronizada a nível de protocolo de rede, é bem flexível na definição de schemas[[19]](#footnote-19) orientados à objetos, e especialmente pela sua performance e escalabilidade em cenários onde há mais consultas do que atualizações.

Assim como no banco de dados, é necessário configurar o realm para saber como extrair as informações de identidade e roles de dentro do diretório. Não faz parte deste curso um escopo à tecnologia LDAP – mas a Trainning fornece cursos especializados na configuração e administração de servidores OpenLDAP.

É possível ainda configurar o JNDIRealm para fazer o bind no diretório LDAP, o que é essencial quando se usa LDAP associado ao Keberos, ou então ele mesmo consultar a senha no diretório e fazer a comparação com a senha informada pelo navegador. Apenas neste segundo caso, o atributo digest se aplica.

A listagem 1.4 ilustra um exemplo de realm que seria adequado para o OpenLDAP em configuração padrão, utilizando o schema inetOrgPerson. O Tomcat tentará se conectar (bind) ao OpenLDAP usando o login do usuário como o atributo uid do diretório.

Se o dn do usuário for encontrado no atributo member de algum objetos dentro da ou “grupos”, este objeto é considerado como sendo um role do usuário

**Listagem 1.4** – exempo de realm para o schema default do OpenLDAP

* <Realm

1. className="org.apache.catalina.realm.JNDIRealm"
2. connectionURL="ldap://localhost:389"
3. userBase="ou=people,dc=example,dc=com"
4. userSearch="(uid={0})"
5. roleBase="ou=groups,dc=example,dc=com"
6. roleName="cn" roleSearch="(member={0})"

/>

Consulte o manual do Tomcat para informações sobre atributos adicionais do JNDIRealm, por exemplo para configurar conexões TLS.

## Exercícios

Laboratório 1.1. Armazenando Senhas Criptografadas

(Prática Dirigida)

**Objetivo:** Eliminar as senhas em texto claro do arquivo tomcat-users.xml.

Primeiro, temos que gerar as versões criptogradas das senhas, e para facilitar a tarefa o exemplo fornece o programa CriptografaSenha.java, que deve ser compilado (vimos como no capítulo 1) pelo aluno.

O seguinte roteiro gera as senhas dos usuários igual estavam em texto claro. Sugerimos que o aluno experimente senhas diferentes para cada usuário:

$ **cd ~/Cap1/Lab1**

$ **javac CriptografaSenha.java**

$ **java CriptografaSenha MD5 testando**

caa9c8f8620cbb30679026bb6427e11f

Já a entrada deste usuário em tomcat-users.xml ficaria assim:

* <user username="fulano"

1. **password="caa9c8f8620cbb30679026bb6427e11f"**

roles="usuario"/>

Depois que todas as senhas tenham sido encriptadas (não esqueça da senha de acesso a Manager!), modifique o server.xml para acrescentar o atributo digest ao realm default, conforme instruções já apresentadas neste capítulo, e então reinicie o Tomcat.

Você pode usar tanto o Manager quanto o exemplo “form” do capítulo anterior para validar a configuração do Tomcat.

Laboratório 1.2. Autenticando contra um banco de dados

(Laboratório)

**Objetivo:** Autenticar usuários contra um banco de dados relacional

Este exercício traz um exemplo que é uma simples adaptação do exemplo “form”, porém com seu contexto configurado para usar um DataSourceRealm. Então ele não irá afetar o acesso ao Manager nem sofrerá interferência dos outros exemplos deste capítulo.

Será reaproveitado o mesmo banco de dados (e DataSource global) do capítulo 8, que já deverá estar populado com dados de usuários e roles.

O banco de dados contém os usuários “sicrano”/“teste123” e “gerente”/“123456” que devem ser utilizados em lugar de “fulano” e “chefe”.

Então use o buildfile para fazer o deploy da aplicação, e valide seu funcionamento. Experimente depois retirar o DataSource global do Tomcat (e também o link no contexto da aplicação) para confirmar que, sem acesso ao banco de dados, a aplicação não funciona.

Laboratório 1.3. Autenticando contra um diretório LDAP

(Laboratório)

**Objetivo:** Autenticar usuários contra um servidor OpenLDAP

Este exercício traz uma nova adaptação do exemplo “form”, porém com seu contexto configurado para usar um JNDIRealm. Então ele não irá afetar o acesso ao Manager nem sofrerá interferência dos outros exemplos deste capítulo.

O diretório do exemplo também traz o arquivo LDIF utilizado para popular o servidor OpenLDAP no computador do instrutor, e scripts para busca no diretório, que podem ser examinados pelos alunos com conhecimento prévio da tecnologia LDAP.

Será necessário modificar a definição do realm no descritor propretário da aplicação para indicar os parâmetros corretos de conexão ao servidor OpenLDAP da sala de aula.

Depois basta usar o ant para deployar a aplicação e então a teste com os usuários “fulano”/“detal” e “admin”/“senha”

## Conclusão

Agora sim temos o conhecimento necessário para configura autenticação em aplicações reais utilizando o Tomcat junto à bancos de dados ou diretórios LDAP. Mas ainda voltaremos em capítulos futuros ao tópico geral de segurança, quando veremos configurações de firewal e de SSL com o Tomcat.

Questões de Revisão

* O Tomcat impõe uma estrutura de tabelas para autenticar usuários em um banco de dados relacional?

* Qual a diferença entre o conceito de “roles” adotado pelo Java EE e o conceito de “grupo” normalmente utilizado em permissões do sistema de arquivos?

# Logging no Tomcat

Este capítulo apresenta as configurações de logging do servidor de aplicações.

Tópicos:

* Configurações de logging da JVM
* Logs de acesso

## Logging no Tomcat

A manutenção de um registro de atividades em qualquer tipo de servidor é fundamental para identificar e isolar problemas de segurança, instabilidade ou performance. Por outro lado, logs de servidores de aplicações podem crescer rapidamente e se tornarem um gargalo de performance.

O log do servidor de aplicações exige um cuidado maior do que os logs de outros tipos de servidores de rede, pois misturam informações referentes a vários usuários e aplicações distintos, além das informações referentes ao próprio servidor de aplicações.

O Tomcat utiliza uma versão customizada da biblioteca Commons Logging do Projeto Jakarta (http://jakarta.apache.org/commons/logging) que vem atrelada à API de logging padrão do Java SE. Mas ele pode ser configurado, pela adição de uma nova customização do Commons Logging, para usar o (mais) popular Log4J.

Neste curso nos focaremos na configuração de log default do Tomcat, os interessados em mudar para o Log4J devem consultar os manuais do Tomcat para ver o procedimento.

O fato é que ambas as APIs de logging oferecem grande flexibilidade em relação à configuração de filtros, nível de detalhes e separação das informações de log em arquivos separados ou mesmo seu envio para servidores remotos. Se o aluno não está habituado ao uso e configuração de algum sistema de logging de servidor, como o syslog do Unix, vale à pena estudar com cuidado a documentação das duas bibliotecas de logging

## Conceitos de Logging do Java SE

A API de logging do Java SE e sua configuração são baseadas em um pequeno conjunto de conceitos:

* Categoria (category) identifica o componente que gera a mensagem de log, de modo que seja possível registrar as mensagens de alguns componentes mas não de outros. Por convenção utiliza-se o nome completo de uma classe como categoria, o que facilita a construção de filtros para separar mensagens provenientes de diferentes bibliotecas ou aplicações;
* Nível (level ou severity) identifica a criticidade da mensagem e o seu nível de detalhamento. Níveis mais altos registram menos informações nos log, e níveis mais baixos registram mais. A idéia é que níveis mais altos sejam reservados para mensagens de erro fatais, enquanto que níveis mais baixos sejam utilizados para informações de depuração;
* Saída (handler ou appender) indica o local onde o log será registrado, em geral um arquivo texto, mas pode ser o console (saída padrão), um servidor syslog remoto, ou até mesmo um endereço de e-mail;
* Formatador (formatter ou layout) diz como devem ser formatadas as mensagens de logging, em relação à origem, data e hora, severidade e etc. Diferentes formatadores podem usar formato CSV (comma-separated values, ou texto separado por vírgulas) ou XML.

A configuração de logging envolve associar saídas para cada categoria, e níveis para cada categoria e saída. Com estas combinações é possível gerar logs mais detalhados ou mais suscintos.

Várias categorias podem ser vinculada a uma mesma saída, e uma mesma categoria pode ser vinculada a várias saídas diferentes. Mas Cada saída é associada a um único formatador.

O nível pode ser indicado de forma independente para as saídas e categorias, de modo a controlar o nível de detalhamento em cada arquivo de log.

O nível default é INFO pode ser mudado, por exemplo, para:

* SEVERE (menos detalhes, registra apenas erros críticos)
* FINEST (mais detalhes, exibe informações sobre o funcionamento interno do Tomcat)
* ALL (registra todas as mensagens, sem descartar nada)
* OFF (não registra nada)

### Porque programar para a API de logging

A geração de logs pelo servidor de aplicações já obrigatória para permitir a depuração de configurações do próprio servidor, mas a grande utilidade dos logs depende do seu uso pela aplicação.

Em um ambiente multiusuário, surgem problemas que dependem da interação entre vários usuários e threads concorrentes, problemas que não podem ser reproduzidos facilmente no ambiente do desenvolvedor e que são virtualmente impossíveis de acompanhar em um depurador passo-a-passo.

Na verdade os problemas de estabilidade em um servidor de aplicações serão na maioria das vezes causados por código de aplicações hospedadas no mesmo. Então, se estas aplicações não gerarem mensagens de log úteis, identificar a causa dos problemas se torna um jogo de adivinhação.

Espera-se que as aplicações utilizem a API de logging em vez de “prints depurativos”, pois assim o administrador pode decidir a qualquer momento gerar logs mais ou, menos detalhados, para cada aplicação em particular, conforme a necessidade. Assim é possível chegar a um balanço entre o registro de informações para depuração e o peso deste registro sobre um ambiente de teste ou mesmo de produção.

Em especial, uma aplicação deve usar a API de logging para gerar mensagens de erro, em vez de chamar diretamente métodos como printStackTrace de Trowable (que é a superclasse de todas as exceções e erros do Java). Se for usada a API de logging, é possível relacionar claramente cada erro com suas causas. Se a aplicação gerar mensagens de erro na saída padrão (ou mesmo na saída de erros) não é possível delimitar de forma automática o início e fim de cada stack trace.

Consulte os JavaDocs do Java SE, em especial as classes java.util.logging.Logger e java.util.logging.Level para saber sobre a programação da API de logging do JavaSE. A listagem 1.1 exibe um exemplo do uso de diferentes níveis de logging em uma aplicação, de modo a contextualizar melhor este tópico.

**Listagem 1.1** – exemplo de aplicação fazendo uso de logging

* public void doGet(

1. HttpServletRequest request, HttpServletResponse response)
2. throws ServletException, IOException {
3. **log.info("Requsitada lista de contatos");**
5. DataSource ds = null;
6. try {
7. **log.fine("Buscando datasource");**
8. InitialContext ctx = new InitialContext();
9. ds = (DataSource)ctx.lookup(
10. "java:comp/env/jdbc/ContatosLocal");
11. }
12. catch (NamingException ex) {
13. **log.log(Level.SEVERE, "Erro na busca JNDI", ex);**
14. throw new ServletException(ex);

}

Observe o uso dos níveis INFO, FINE e SEVERE: o primeiro reporta eventos normais porém significativos em uma aplicação; o segundo exibe mensagens de depuração ou de trace; e o terceiro exibe erros dos quais a aplicação não consegue se recuperar sozinha.

É claro, há muito mais a se falar em termos de melhores práticas de logging tanto para o desenvolvedor quanto para o administrador, mas esperamos que esta breve introdução tenha sido suficiente para demonstrar a importância e utilidade prática do tema.

## O Juli do Tomcat

O Tomcat na verdade utiliza uma extensão da API de logging padrão do Java EE, chamada Juli. Esta extensão acrescenta funcionalidades importantes como rotação dos arquivos de log (para que eles não cresçam indefinidamente) e menos restrições para a nomeação das categorias.

Fora estas capacidades extras, o Juli é utilizado pelas aplicações e configurado exatamente da mesma forma que a API de logging padrão do Java SE. A configuração padrão para a Logging API está na pasta conf, no arquivo logging.properties.

Recomendamos que aplicações não tragam suas próprias configurações de logging dentro dos seus pacotes WAR, mas que utilizem as configurações no próprio arquivo logging.properties do Tomcat. Assim o administrador pode variar estas configurações sem ter que mexer “dentro” dos pacotes e fazer re-deployment dos mesmos.

Também recomendamos que os pacotes não tragam classes (JARs) de nenhuma biblioteca de logging. O ideal é que elas utilizem o Commons Logging, de modo a funcionar sem modificações nem recompilação em servidores que utilizem a API de logging do Java SE quanto em servidores que utilizem o Log4J,

A listagem 1.2 apresenta trechos da configuração de log default do Tomcat, para que possamos discutir a sua sintaxe:

**Listagem 1.2** – Trechos da configuração de logging default do Tomcat

* handlers = 1catalina.org.apache.juli.FileHandler, **2localhost.org.apache.juli.FileHandler**, 3manager.org.apache.juli.FileHandler, 4admin.org.apache.juli.FileHandler, 5host-manager.org.apache.juli.FileHandler, java.util.logging.ConsoleHandler

1. .handlers = 1catalina.org.apache.juli.FileHandler, java.util.logging.ConsoleHandler
2. ...
3. 2**localhost.org.apache.juli.FileHandler**.level = FINE
4. **2localhost.org.apache.juli.FileHandler**.directory = ${catalina.base}/logs
5. **2localhost.org.apache.juli.FileHandler**.prefix = localhost.
6. ...
7. org.apache.catalina.core.ContainerBase.[Catalina].[localhost].level = INFO
8. org.apache.catalina.core.ContainerBase.[Catalina].[localhost].handlers = **2localhost.org.apache.juli.FileHandler**

...

A primeira linha contém uma relação de todos as saídas a serem configuradas no arquivo. Cada saída é nomeada por um prefixo iniciado por um dígitos numéricos, seguido do nome completo da classe que implementa o tipo de saída.

Observe que são usados dois tipos de saída: org.apache.juli.FileHandler, que é uma saída em arquivo texto que é rotacionado diariamente, e java.util.logging.ConsoleHandler . Já 2localhost.org.apache.juli.FileHandler é uma das saídas para arquivo que foram configuradas.

A linha iniciada por .handlers define quais são as saídas para a categoria raiz, cujo nome é vazio. Estas saídas receberão as mensagens de todas as categorias que não tenham suas próprias saídas configuradas explicitamente.

O nome completo da saída é utilizado mais adiante para configurar propriedades da própria saída, como level (que é o nível de mensagens a serem filtradas), directory (o local onde o arquivo será gerado) e prefix (o prefixo do nome do arquivo, que será acrescentado da data corrente em formato “aaaa-mm-dd”).

Por fim, categorias específicas são associadas a seus próprios níveis (level) e saídas (handler).

### Arquivos de log na configuração default

Além da convenção padrão de usar o nome de classes da aplicação (org.apache.catalina.core.ContainerBase), o Tomcat acrescenta em suas próprias categorias a hierarquia de configuração [Engine].[Host].[Context], por exemplo [Catalina].[localhost].[/manager].

Desta forma o Tomcat pode gerar logs separados para cada container web (normalmente haverá apenas um), cada host (que podem haver vários, como veremos no capítulo sobre configurações de rede) e cada contexto de aplicação.

As categorias [Engine].[Host].[Context] são úteis para a depuração do próprio Tomcat e das suas configurações. Gerar um arquivo de log separado para um contexto afeta as mensagens de log geradas pela aplicação deployada neste contexto.

A configuração padrão gera todos os arquivos na pasta logs (${catalina.base} é o diretório de instalação do Tomcat, ou sua emulação pelo JPackage). Estes arquivos são:

* catalina.<data>.log contém as mensagens de todas as categorias que não definiram suas próprias saídas;
* localhost.<data>.log contém mensagens específicas para o host “localhost”;
* manager.<data>.log contém as mensagens da aplicação de administação do Tomcat, o Manager.

Todos esses arquivos, exceto o primeiro, servem principalmente para a depuração do próprio Tomcat e provavelmente serão de pouca utilidade para o administrador, que pode pensar em elimina-los da sua configuração de produção.

Além disso, o script de início do Tomcat gera na pasta logs o arquivo catalina.out contendo o redirecionamento das saída padrão e de erros da JVM. Assim mensagens geradas de forma incorreta pelas aplicações (por exemplo, chamando System.out ou Exception.printStackTrace) não serão perdidas.

O arquivo catalina.out também irá conter as mesmas mensagens de log que catalina.<data>.log já que a categoria raiz também foi vinculada à saída java.util.logging.ConsoleHandler. Entretanto as mensagens de erro geradas de forma incorreta não estarão no arquivo de log, apenas na saída padrão.

O arquivo catalina.out não está sujeito à rotação, pois ele não é gerenciado pelo logging do Java SE. O administrador terá que tomar cuidado para que este arquivo não cresça indefinidamente.

## Logs da aplicação

É fácil adaptar o modelo fornecido pelo logging.properties para gerar um arquivo de log contendo somente as mensagens geradas por uma uma aplicação específica.

O processo consiste em:

1. Acrescentar uma nova saída na relação no início do arquivo;
2. Configurar esta saída, informando o nível de severidade das mensagens registradas e o prefixo do nome do arquivo
3. Vincular a saída à categoria “raiz” da aplicação, indicado pelo seu maior prefixo comum para nomes de pacotes Java.

A listagem 1.3 apresenta os trechos a serem modificados (a maioria deles acrescentados) na configuração de log default do Tomcat.

**Listagem 1.3** – exemplo de geração de logs separados para uma aplicação

* handlers = 1catalina.org.apache.juli.FileHandler, 2localhost.org.apache.juli.FileHandler, 3manager.org.apache.juli.FileHandler, 4admin.org.apache.juli.FileHandler, 5host-manager.org.apache.juli.FileHandler, **6contatos.org.apache.juli.FileHandler**, java.util.logging.ConsoleHandler

1. ...
2. **6contatos.org.apache.juli.FileHandler**.level = FINE
3. **6contatos.org.apache.juli.FileHandler**.directory = ${catalina.base}/logs
4. **6contatos.org.apache.juli.FileHandler**.prefix = contatos.
5. ...
6. exemplo.level = FINE

exemplo.handlers = **6contatos.org.apache.juli.FileHandler**

O resultado destas modificações é a geração do arquivo exemplo.<data>.log contendo apenas as mensagens para todas as aplicações de exemplo desta apostila, pois todas estão no pacote “exemplo” (veja no código-fonte das classes Java a declaração “package exemplo;”).

Além disso, as mensagens da aplicação (ou no caso, das nossas aplicações de exemplo) estão sendo geradas com o nível FINE, ou seja, mensagens de depuração.

### Uma configuração para produção

Em ambiente de produção, mesmo as mensagens de nível INFO (que é o default do Tomcat) podem gerar um volume de E/S de disco suficiente para comprometer a performance do ambiente, e talvez gerar até Gibabytes de logs por dia.

Por isso sugerimos que o nível default em produção seja WARNING, ou mesmo SEVERE, configurando níveis mais baixo apenas para aplicações específicas – ou preferencialmente para módulos isolados destas aplicações – quando houver necessidade de informações de depuração.

Então na listagem 1.4 temos um exemplo completo do arquivo logging.properties seguindo estas recomendações.

**Listagem 1.4** – sugestão de configuração de logging para produção

* handlers = 1catalina.org.apache.juli.FileHandler,6contatos.org.apache.juli.FileHandler, java.util.logging.ConsoleHandler

1. .handlers = 1catalina.org.apache.juli.FileHandler, java.util.logging.ConsoleHandler
2. 1catalina.org.apache.juli.FileHandler.level = WARNING
3. 1catalina.org.apache.juli.FileHandler.directory = ${catalina.base}/logs
4. 1catalina.org.apache.juli.FileHandler.prefix = catalina.
5. 6contatos.org.apache.juli.FileHandler.level = INFO
6. 6contatos.org.apache.juli.FileHandler.directory = ${catalina.base}/logs
7. 6contatos.org.apache.juli.FileHandler.prefix = contatos.
8. java.util.logging.ConsoleHandler.level = WARNING
9. java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter
10. exemplo.level = INFO

exemplo.handlers = 6contatos.org.apache.juli.FileHandler,1catalina.org.apache.juli.FileHandler

Observe que temos apenas três saídas, o arquivo catalina-<data>.out, o console e o arquivo da aplicação contatos-<data>.log. Caso houvessem mais aplicações no servidor, provavelmente seriam configuradas saídas adicionais para cada uma.

Note ainda que a categoria da aplicação (exemplo) agora está configurada para duas saídas ao mesmo tempo, o arquivo default to Tomcat e também o específico da aplicação.

Mas, de acordo com os níveis configurados para cada categoria e saída, o arquivo default do Tomcat irá conter apenas mensagens de erro, enquanto que as mensagens informativas estarão apenas no arquivo específico da aplicação.

É bem útil para o administrador ter um único log com todos os erros, de modo que ele não perca tempo no dia-a-dia com mensagens de depuração. Mas, caso surja uma mensagem de erro, ele tem o log da aplicação com mais detalhes para depurar o problema.

Esta sugestão de configuração pode não agradar aos adeptos do auto-deploy, pois ela não irá gerar as mensagens confirmando o deployment e undeployment bem-sucedido das aplicações, que são gerada no nível INFO.

Por isso na listagem 1.5 indicando como modificar a configuração de modo que as mensagens de deployment sejam apresentadas, mas não outras mensagens informativas geradas pelo Tomcat durante sua execução normal.

**Listagem 1.5** – modificando a sugestão para produção de modo a preservar as mensagens de log no deployment e undeployment via auto-deploy

* ...

1. 1catalina.org.apache.juli.FileHandler.level = **INFO**
2. ...
3. java.util.logging.ConsoleHandler.level = **INFO**
4. ...
5. **.level = WARNING**
6. **org.apache.catalina.startup.HostConfig.level = INFO**

...

Note que foi necessário aumentar o nível das saídas para INFO, e então definir explicitamente o nível para a categoria padrão como WARNING (ou SEVERE). Em seguida, o nível da categoria org.apache.catalina.startup.HostConfig, que é quem processa o auto-deploy de aplicações, foi aumentado para INFO.

## Logs de Acesso

Os logs de acesso formam um conjunto diferente de logs, que não são tratados pelo Commons Logging mas sim por uma válvula do Tomcat.

Logs de acesso registram páginas visitadas pelos usuários, permitindo gerar estatísticas como páginas mais visitadas, volume de E/S e tempo médio de resposta para o site ou aplicações isoladas.

A configuração padrão da válvula (que está comentada no arquivo server.xml e ilustrada na listagem 1.6) gera um log no mesmo formato do gerado pelo Apache HTTPd, pronto para ser processado por ferramentas como Webalizer e Awstats.

**Listagem .6** – ativação da válvula de log de acesso no server.xml

* <Host name="localhost" appBase="webapps"

1. ...
2. <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
3. prefix="localhost\_access\_log." suffix=".txt" pattern="common"

resolveHosts="false"/>

Observe que o diretório logs é relativo ao diretório de instalação do Tomcat, não é necessário prefixar com ${catalina.base} como tem que ser feito nas configurações do logging.properties.

Esta configuração pode ser modificada para incluir informações úteis para aplicações web que não estão normalmente presentes no log padrão do Apache, por exemplo tempo de execução da página e valores de atributos da sessão HTTP. Consulte o manual do Tomcat para mais informações sobre estes ajustes.

## Exercícios

Laboratório .1. Aplicação utilizando a API de logging

(Prática Dirigida)

**Objetivo:** Observar um modelo de como aplicações deveriam gerar mensagens de erros e de depuração, e configurar o nível de detalhamento destas mensagens nos logs do Tomcat

O exemplo deste exercício é uma variação da aplicação “contatos” que utiliza a API de logging do Java SE. Examine primeiro o código-fonte do Servlet em src/exemplo/ContatosServlet.java e depois use o buildfile já incluso para deployar a aplicação.

Acesse a página de lisgtagem de contatos, e em seguida examine cada arquivo de log do Tomcat, para ver em quais apareceram mensagens de log da aplicação. Registre à seguir quais arquivos tinham mensagens específicas da aplicação:

O instrutor irá então derrubar o servidor de banco de dados, de modo que os logs do Tomcat deverão exibir mensagens de erro da aplicação, por exemplo:

* 03/09/2009 15:16:24 exemplo.ContatosServlet doGet

1. INFO: Requsitada lista de contatos
2. 03/09/2009 15:16:24 exemplo.ContatosServlet doGet
3. SEVERE: Erro de banco na consulta

org.apache.tomcat.dbcp.dbcp.SQLNestedException: Cannot create PoolableConnectionFactory (Conexão negada. Verifique se o nome da máquina e a porta estão corretos e se o postmaster está aceitando conexões TCP/IP.)

Observe como ficou fácil identificar a aplicação que gerou o erro, e qual a origem do erro.

Agora edite as configurações de log do Tomcat para que a aplicação passe a gerar mensagens de depuração, acrescentando a linha:

exemplo.level = FINE

Reinicie o Tomcat e acesse novamente a aplicação. Agora, haverão nos logs mensagens como:

* 03/09/2009 15:27:37 exemplo.ContatosServlet doGet

1. INFO: Requsitada lista de contatos
2. 03/09/2009 15:27:37 exemplo.ContatosServlet doGet
3. FINE: Buscando datasource
4. 03/09/2009 15:27:37 exemplo.ContatosServlet doGet
5. FINE: Conectando ao banco
6. 03/09/2009 15:27:37 exemplo.ContatosServlet doGet
7. FINE: Executando consulta SQL
8. 03/09/2009 15:27:37 exemplo.ContatosServlet doGet
9. FINE: Percorrendo resultado
10. 03/09/2009 15:27:37 exemplo.ContatosServlet doGet
11. FINE: Fechando conexão ao banco
12. 03/09/2009 15:27:37 exemplo.ContatosServlet doGet

FINE: Exibindo página de contatos

Observe que os níveis são cumulativos, então o nível FINE inclui também as mensagens dos níveis INFO e SEVeRE.

Ainda é possível exibir mensagens mais detalhadas da aplicação? Sim, temos mensagens em FINEST, no qual a aplicação diz não só qual caminho de código foi executado, mas também os valores de variáveis, argumentos e resultados do banco de dados.

Então configure o Tomcat para gerar mensagens deste nível apenas para a aplicação de exemplo, ou melhor, para todas as aplicações de exemplo. O exemplo deste capítulo é o único no curso que gera logs, mas como todos os exemplos foram colocados no mesmo pacote da linguagem Java, todos entram na mesma categoria de logging (a não ser que você queira configurar logging para classes individuais, o que é permitido porém muito trabalhoso para ser viável em uma aplicação real).

Laboratório 1.2. Configuração de logging para produção

(Laboratório)

**Objetivo:** Gerar logs contendo apenas mensagens de erro para o Tomcat e mensagens informativas para as aplicações, em arquivos separados.

A idéia é implementar a configuração sugerida neste capítulo. O diretório de exemplo deste laboratório (Cap1/Lab2) já contém um modelo para o arquivo logging.properties, de modo a poupar tempo de edição dos alunos.

Entretanto, o modelo está configurado para depuração da aplicação, mas queremos um log mais suscinto, contendo apenas mensagens informativas.

Então edite o modelo e coloque no local apropriado para o Tomcat, não esquecendo de salvar o arquivo de configuração original para o caso de você precisar voltar atrás.

Em seguida, reinicie o Tomcat e rode a aplicação, comprovando que os logs estão sendo gerados apenas com o nível de detalhe desejado, que é:

* WARNING para o Tomcat;
* INFO para a plicação.

No final deste exercício, retorne a configuração para o nível default INFO, pois queremos ver as mensagens de log informativas do Tomcat nos próximos capítulos.

Laboratório 1.3. Logs de acesso

(Laboratório)

**Objetivo:** Gerar logs de acesso a páginas no servidor Tomcat.

Ative a válvula de log de acessos, conforme as instruções já apresentadas neste capítulo, e reinicie o Tomcat. Em seguida, navegue pelas páginas de alguma aplicação exemplo, e também pelo Manager. Depois verifique se o log de acessos contém a relação correta de páginas visitadas.

Não remova esta configuração, pois nos próximos capítulos vamos precisar dos logs de acesso gerados pelo Tomcat.

## Conclusão

A geração de logs pelo servidor e pelas próprias aplicações são frequentemente a última preocupação de administradores e desenvolvedores – até o momento em que algo sai errado, e as informações do log são necessárias. Mas os logs não são muito úteis se sua configuração for realizada apenas depois dos problemas se iniciarem, nem se as aplicações não fizerem uso das APIs especializadas.

Por isso vale à pena ser pró-ativo, e esta é uma das áreas (junto com o uso e configuração de DataSources no JNDI) onde é crucial a interação entre administradores e desenvolvedores.

Já o log de acessos costuma ser útil por si só, e é bem leve, então vale à pena deixa-lo ativo sempre.

Questões de Revisão

* Mensagens de erro geradas manualmente pela aplicação, chamando o método println de System.out ou System.err são registradas no arquivo de log catalina-<data>.log do Tomcat?

* É possível extrair informações de uso de processador do log de acessos do Tomcat?

# Configurações de Rede

Neste capítulo seremos apresentados às configurações de rede do Tomcat, incluindo hosts virtuais e recursos de segurança de rede embutidos no servidor de aplicações

Tópicos:

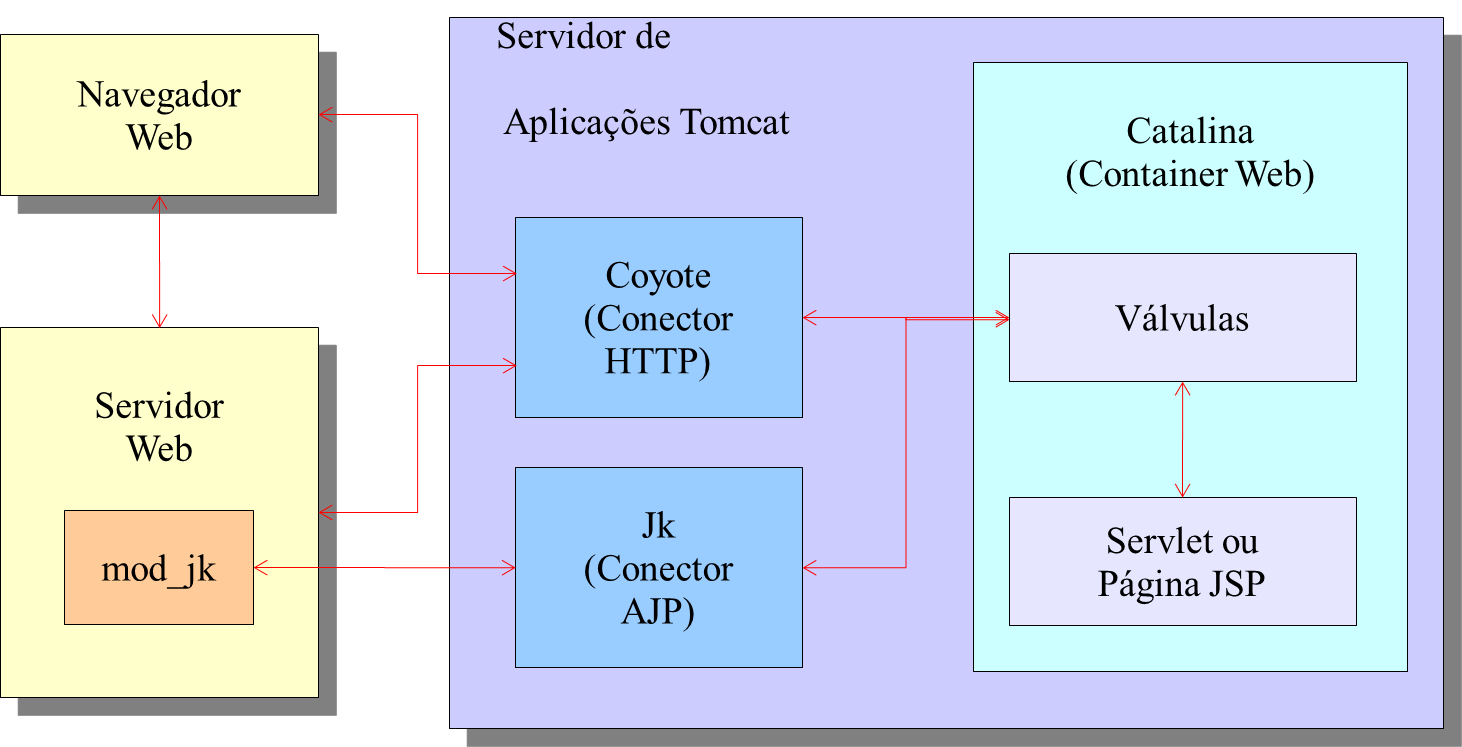
* Conectores do Tomcat
* Suporte a SSL
* Configuração de hosts virtuais
* Válvulas de firewall

## Conectores e Válvulas do Tomcat

Os conectores são os componentes do Tomcat responsáveis por receber conexões de redes, processar o protocolo utilizado na conexão e repassar as requisições para o container web (Catalina).

Já as válvulas são componentes configurados no container web para interceptar e pré (ou pós) processar as requisições recebidas de qualquer conector. Uma válvula pode cumprir funções de depuração auditoria ou implementar funcinalidades específicas como o single sign-on já visto no capítulo sobre autenticação.

A figura 1.1 apresenta o fluxo de tratamento de uma requisição HTTP pelo Tomcat, mostrando em que momento os conectores e válvulas participam do processo.



**Figura 1.8** – fluxo de processamento de uma requisição HTTP pelo Tomcat

O Tomcat pode ser utilizado diretamente como um servidor web, porque ele possui um conector que entende diretamente o protocolo HTTP. Este conector suporta, com a ajuda do próprio container web, recursos como execução de scripts CGI e Server-Side Includes, além de embutir um cache para arquivos estáticos. Ou seja, para muitas empresas o Tomcat substituiria perfeitamente bem um Apache HTTPd ou IIS.

O conector que reconhece o HTTP é chamado de Coyote, e pode ser configurado para aceitar ou HTTP simples ou então HTTPS, que é o HTTP sob SSL. Em sites que usem SSL, serão configuradas duas instâncias do Coyote, uma para URLs http: e outra para URLs https:.

Quando existe um outro servidor web entre o Tomcat e o navegador do usuário, o Conector Jk permite comunicação otimizada usando o protocolo AJP. É possível integrar o Tomcat a um servidor web nativo usando o HTTP simples, entretanto o uso do AJP traz grandes melhorias de performance e reduz a utilização de rede ente os dois servidores.

O uso do JK e cenários de integração entre o Tomcat e servidores web nativos serão o tema do próximo capítulo. Por enquanto vamos continuar usando o Tomcat como o servidor web visto pelo navegador do usuário.

### Configuração padrão do Coyote

Um elemento <Conector> com configuração mínima, por exemplo o primeiro conector presente na configuração default do Tomcat no arquivo server.xml, aninhado ao elemento <Service>, define um conector Coyote que processa o HTTP simples. A listagem 1.1 apresenta este conector.

**Listagem 1.1** – conectores presentes na configuração default do Tomcat

1. ...
2. <Service name="Catalina">
3. ...
4. **<Connector port="8080" protocol="HTTP/1.1"**
5. **connectionTimeout="20000"**
6. **redirectPort="8443" />**
7. ...
8. <Connector port="8009" protocol="AJP/1.3"
9. redirectPort="8443" />
10. ...
11. <Engine name="Catalina" defaultHost="localhost">

...

Cada tipo de conector do Tomcat suporta vários atributos que não aparecem na configuração default mas podem ser acrescentados conforme a necessidade. Recomendamos que os alunos dêm uma boa lida no manual do Tomcat para ver as possibilidades existentes, mesmo que as mais usuais sejam abordados neste e nos próximos capítulos desta apostila.

Inicialmente, os atributos importantes são port e protocol. O primeiro indica a porta TCP na qual o conector aceitará conexões, e o segundo indica o protocolo que será aceito pelo conector, o que na prática indica o tipo de conector a ser instanciado na inicialização do Tomcat.

### Rodando o Tomcat na porta 80

Seria fácil mudar a configuração do Coyote para que ele aceite conexões na porta 80, mas isto não é recomendado em sistemas Unix. O motivo é que as portas baixas (abaixo de 1024) são restritas ao root, então o Tomcat instalado via JPackage ou a instalação manual realizada no home do aluno não poderiam abrir a porta 80.

Seria possível colocar um Tomcat nos diretórios de sistema e o roda-lo como o usuário root, mas isto violaria uma recomendação de segurança do ambiente Unix: nunca aceitar conexões remotas como root.

A solução é configurar o firewall interno do sistema operacional (o iptables do Linux) para redirecionar a porta 80 para a porta do Tomcat, seja a 8080 ou outra, utilizando o recurso de DNAT (Destination Network Address Translation). Os comandos para esta configuração seriam, conforme o FAQ do Tomcat:

# iptables -A FORWARD -p tcp --destination-port 80 -j ACCEPT

# iptables -t nat -A PREROUTING -j REDIRECT -p tcp --destination-port 80  
--to-ports 8080

Outras alternativas seriam fazer o DNAT no firewall de borda da empresa; usar algum redirecionador de portas; ou usar um servidor web como front-end. Esta última opção será vista no próximo capítulo.

## Firewall interno do Tomcat

As válvulas RemoteAddrValve e RemoteHostValve permitem definir regras de firewall em um <Engine>, <Host> ou contexto. A funcionalidade é semelhante a outros firewalls de aplicação, como o tcpwrapper do Unix, e é um recurso útil pra implementar “segurança em profundidade”[[20]](#footnote-20).

Os especialistas em segurança irão alertar que segurança baseada em nome de host é questionável, porque é fácil fornecer mapeamentos faltos para um servidor DNS (por isso certificados SSL reais são assinados por um terceiro).

Já a segurança baseada em IP tem pouca efetividade pois a maioria dos usuários estará em uma rede privativa, acessando a Internet por meio de um roteador com NAT, portanto o servidor enxerga um único IP para dezenas ou mesmo centenas de usuários diferentes.

Ainda assim a filtragem por IP tem sua utilidade, pois permite identificar a empresa ou provedor de origem de um usuário, especialmente em ambientes de B2B.

A listagem 1.2 demonstra como autorizar apenas usuários provenientes do domínio empresa.com.br.

**Listagem 1.2** – aceitando apenas acesos com origem em um certo domínio DNS

1. <Valve className="org.apache.catalina.valves.RemoteHostValve"

allow=".\*.empresa\.com\.br"/>

Já a listagem 1.3 demonstra como autorizar apenas acessos provenientes da rede local da empresa, que estaria na faixa privativa 192.168.0.0/24.

**Listagem 1.3** – aceitando apenas acesos originados em uma subnet IP

1. <Valve className="org.apache.catalina.valves.RemoteAddrValve"

allow="192\.168\.0\..\*"/>

Note que os valores dos atributos allow e deny das válvulas de firewall do Tomcat utilizam expressões regulares padrão do Java SE, que seguem a mesma sintaxe do egrep e Perl. Então há algumas surpresas para o não-iniciado, como o fato de que “.” significa “qualquer coisa” e por isso teve que ser escapado com “\”.

Mas podem ser especificadas várias expressões, separadas por vírgulas.

## Conexões SSL com o Tomcat

O Tomcat suporta, por meio do JSSE (Java Secure Sockets Extensions, padrão no Java SE desde a versão 1.4) conexões encriptadas nos padrões SSL e TLS, ou seja, conexões https:.

Em alguns casos, prefere-se que a criptografia seja realizada por um servidor web nativo atuando como front-end, ou por um “acelerador web” (proxy reverso) devido à possibilidade de se usar hardware acelerador especializado.

Ou então, se houver um servidor web atuando como front-end para o Tomcat, por exemplo como banceador de rede para um cluster, este servidor tem que implementar o processamento do SSL e não será necessário configurar o conector SSL do Tomcat. Esta configuração será vista no próximo capítulo.

O conector SSL está comentado no server.xml padrão do Tomcat, basta descomenta-lo. Mas é necessário fornecer um certificado digital para que o conector funcione. A listagem 1.4 mostra as alterações necessárias para isto.

**Listagem 1.4** – confirando um certificado digital no conector SSL do Tomcat

1. <Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
2. scheme="https" secure="true"
3. **keystorePass="secreto"**
4. **keystoreFile="conf/keystore"**

clientAuth="false" sslProtocol="TLS" />

O certificado pode ser gerado e assinado por qualquer ferramenta que suporte os padrões Internet para PKI (Public Key Infrastructure) como o OpenSSL ou o EJBCA. Mas para conveniência do desenvolvedor e também do administrador o JDK fornece o keytool.

### Gerando certificados com o keytool

O keytool é uma ferramenta para manutenção de certificados digitais, mantidos em um arquivo chamado keystore. Ele é capaz de criar keystores, importar e exportar certificados (junto ou em separado das respectivas chaves privadas) e assinar certificados. O keytool fornece funcionalidade suficiente para uma CA (Certification Authority ou Autoridade Certificadora) básica.

Sites de produção utilizam um certificado adquirido em uma empresa reconhecida por padrão pelos navegadores, como a Verisign. Entretanto, para testes e Intranets, é possível gerar um certificado “auto-assinado”, que traz as mesmas garantias de segurança contra interceptação de tráfego que um certificado “oficial”.

O que um certificado “oficial” garante, e que o certificado auto-assinado não consegue garantir, é que o usuário está acessando o site real e não uma imitação. Por isso o navegador SEMPRE irá alertar o usuário quanto a certificados auto-assinados

O keystore e o certificado auto-assinado podem ser gerados com uma única execução do keytool. As informações de cadastro armazenadas no certificado só são importantes em um certificado real, mas é importante que a senha do keystore e a senha do certificado sejam as mesmas. Eis um exemplo de execução do keytool, gerando o arquivo keystore no diretório corrente:

$ **keytool -keystore keystore -genkey -alias tomcat -keyalg RSA**

Enter keystore password:

Re-enter new password:

What is your first and last name?

[Unknown]: **Miguel Vilaça**

What is the name of your organizational unit?

[Unknown]: **Treinamento**

What is the name of your organization?

[Unknown]: **Trainning**

What is the name of your City or Locality?

[Unknown]: **Sao Paulo**

What is the name of your State or Province?

[Unknown]: **SP**

What is the two-letter country code for this unit?

[Unknown]: **br**

Is CN=Miguel Vilaça, OU=Treinamento, O=Trainning, L=Sao Paulo, ST=SP, C=br correct?

[no]: **yes**

Enter key password for <tomcat>

(RETURN if same as keystore password):

Nos dois primeiros prompts pela senha, foi digitada a mesma senha (lembre-se de qual foi, você vai precisar coloca-la na configuração do conector SSL!). Mas no segundo prompt foi digitado apenas [Enter].

Consulte os manuais do Tomcat e do Java SE para o procedimento caso você queira utilizar um certificado “oficial”, em um servidor de produção. Em resumo, você terá que:

1. Gerar um CSR (Certificate Sign Request) usando keytool, onde o prompt “First and last name” deve ser respondido com o nome DNS público do seu servidor;
2. Enviar o CSR para uma autoridade certificadora reconhecida, junto com documentação da sua empresa e o comprovante de pagamento de uma taxa de serviço particular para cada empresa que presta este serviço;
3. A autoridade certificadora devolve para você o certificado assinado por ela, que deve ser importado para um keystore e configurado no conector SSL do Tomcat.

### Segurança declarativa x SSL

É possível definir uma restrição de acesso que obrigue o uso de SSL para acessar certas páginas, ou mesmo para uma aplicação inteira. Basta definir um <web-resource-collection> incluindo o <transport-guarantee> com o valor CONFIDENTIAL. A listagem 1.5 apresenta um exemplo desta configuração.

**Listagem 1.5** – exigindo conexões SSL para acesso a uma página, no descritor web.xml

1. <security-constraint>

<web-resource-collection>

1. ...

</web-resource-collection>

1. **<user-data-constraint>**
2. **<transport-guarantee>**
3. **CONFIDENTIAL**
4. **</transport-guarantee>**
5. **</user-data-constraint>**

</security-constraint>

Esta restrição pode ser usada sozinha ou então junto com a restrição <auth-constraint> que já vimos no capítulo sobre autenticação.

Pelo padrão Java EE, qualquer tentativa de acessar as páginas da coleção em conexões não-encriptadas irá fazer com que o Tomcat redirecione o usuário para uma conexão SSL.

Por isso os conectores não-SSL tem o atributo redirectPort, presente na listagem 1.1 apresentada antes neste capítulo, pois o container web precisa saber qual porta colocar na nova URL com https:. É claro, o valor do atributo tem que concordar com o valor da porta TCP utilizada pelo conector SSL.

Infelizmente não é possível usar este artifício para garantir que a página de login FORM seja encriptada, porque o navegador nunca referencia diretamente esta página.

## Hosts Virtuais

Hosts virtuais são um artifício padrão de provedores de hospedagem e parte do padrão HTTP. A idéia é que um único servidor web possa responder a requisições cujas URLs informem nomes de host diferentes.

Para o usuário, eles são indistinguíveis de hosts “reais”, hospedados por servidores fisicamente independentes. Daí o “virtual” no nome.

É possível configurar dois tipos de hosts virtuais: baseados em endereços IP diferntes, ou baseados apenas no nome do host (os vários nomes respondem ao mesmo IP). A segunda forma é a preferida, por não desperdiçar endereços IP, especialmente quando estamos falando de servidores visíveis para a Internet.

As duas formas de se configurar hosts virtuais são indiferentes para o Tomcat. Em ambos os casos, basta acrescentar um novo elemento <Host> ao server.xml, usando como nome (name) o nome DNS desejado, e decidir que aplicações serão configuradas nele.

Cada host virtual tem a sua própria pasta para auto-deploy, e tem configurações diferentes quando à ativação ou não desta funcionalidade. É possível gerar logs de depuração e de acesso separados para cada host, ou definir outras configurações em separado como realms, válvulas e DataSources. O formato XML utilizado para o server.xml fornece esta flexibilidade.

### Manager x host virtual

A única limitação dos hosts virtuais no Tomcat é que cada um tem que ser configurado com sua própria instância do Manager. Mas todas as instâncias podem usar o mesmo pacote WAR para a aplicação.

Como foi visto no capítulo 4, há várias formas de se instalar uma aplicação em um host, seja esta aplicação privilegiada como Manager ou qualquer outra. Entre elas:

* Acrescentar um novo elemento <Context> apontando para o WAR do Manager, aninhado na definição do host virtual em server.xml;
* Criar a pasta conf/Catalina/hostvirtual e copiar para ela o arquivo conf/Catalina/localhost/manager.xml (instalação manual do Tomcat), ou então criar dentro dela um novo arquivo manager.xml apontando para o WAR correto (instalação via JPackage).

## Exercícios

Laboratório 1.1. Tomcat como web server principal

(Prática Dirigida)

**Objetivo:** Rodar o Tomcat na porta 80

Não vamos neste momento “brigar” com o firewall do Linux. Em vez disso, vamos configurar o conector HTTP para a porta 80 e iniciar o Tomcat manualmente. Então teremos que rodar inteiramente como root.

Primeiro, derrube o Tomcat usando o comando service. Em seguida, altere o server.xml. Por fim, inicie o tomcat manualmente com o comando:

# tomcat6 start

Teste a alteração acessando o Manager em uma URL sem incluir a porta 8080, ou seja, usando a porta padrão do HTTP que é 80.

Ao final do exercício, desfaça todas as configurações, e finalize o Tomcat iniciado manualmente. Pois no próximo capítulo vamos precisar rodar um Apache HTTPd na porta 80, então não queremos conflito de porta entre ele e o Tomcat.

Laboratório 1.2. Restringindo acesso por IP

(Laboratório)

**Objetivo:** Bloquear o acesso de um computador ao servidor Tomcat, sem usar o firewall do sistema operacional nem um firewall de rede.

Este laboratório deve ser realizado em duplas. Inicie escolhendo um parceiro (seu vizinho na sala de aula) e verifique o endereço IP dele. Informe a ele o seu endereço IP.

Este exercício traz um exemplo que é o velho conhecido “hoje”, porém já incluindo um descritor proprietário META-INF/context.xml onde está um modelo comentado da válvula de firewall.

Use o buildfile fornecido no exemplo para deployar a aplicação, e confirme que seu colega consegue acessar a aplicação. Caso ele não tenha sucesso, verifique se existe alguma restrição de firewall ou outro problema de conectividade de rede para prosseguir. Peça ajuda ao instrutor!

Uma dica é desligar o firewall da sua estação com o comando iptables -F.

Agora que seu colega consegue acessar, modifique a aplicação de modo a ativar a válvula configurado para negar o endereço dele. Use o ant para re-deployar a aplicação com o contexto atualizado (ant desinstala empacota instala).

Se você fez tudo direito, agora seu colega não vai mais conseguir acessar a aplicação.

Laboratório 1.3. Ativando o suporte a https:

(Laboratório)

**Objetivo:** Permitir que o Tomcat aceite conexões https:, usando um certificado auto-assinado.

Siga as instruções já fornecidas neste capítulo para criar o certificado auto-assinado no seu diretório corrente, e em seguida copie o arquivo para o diretório conf do Tomcat.

Depois edite o server.xml para habilitar o conector SSL e indicar o caminho e senha para o keystore. Reinicie o Tomcat, e verifique nos logs se o servidor iniciou corretamente.

Experimente acessar qualquer aplicação via SSL, por exemplo o Manager. A URL seria https://127.0.0.1:8443/manager/html. É importante que a URL indique tanto o prefixo correto (https:) quanto a porta correta (8443) para que a conexão SSL funcione.

O navegador irá reclamar duas vezes do certificado auto-assinado. Uma vez porque não foi capaz de validar a assinatura; outra vez porque o nome do registrado no certificado é diferente do nome de host que está sendo acessado. Responda ao navegador que quer acessar assim mesmo.

Laboratório 1.4. Protegendo páginas com https:

(Laboratório)

**Objetivo:** Assegurar que páginas sensitivas de uma aplicação só estejam disponíveis via https:

Conseguindo o acesso criptografado ao Manager, modifique o seu descritor web.xml para exigir conexões SSL. Afinal o Manager é uma aplicação sensitiva, não queremos expor senhas de administração do Tomcat para sniffers de rede.

Use o modelo já fornecido neste capítulo para configurar a restrição. Cuidado, pois o Manager já possui regras de acesso para todas as suas páginas, então você tem que ACRESCENTAR a restrição CONFIDENTIAL dentro do <security-constraint> já existente no web.xml do Manager.

Como o Manager não permite fazer undeploy ou reload dele mesmo, será necessário reiniciar o Tomcat.

Agora entre no Manager com uma URL normal, sem SSL. O próprio Tomcat deverá gerar um redirect e colocar o usuário automaticamente em https: e com a porta correta.

Em um servidor real, provavelmente você deixaria o Manager assim (modificando os buildfiles das aplicações para acessarem o Manager via SSL) e além disso incluiria uma válvula para restringir ainda mais o acesso. Mas como estamos em sala de aula, você vai desfazer as modificações, liberando novamente o Manager para acesso sem SSL.

Laboratório 1.5. Implementando um host virtual

(Laboratório)

**Objetivo:** Configurar um segundo host, um host virtual baseado no nome, dentro do mesmo servidor Tomcat

Para que nosso host virtual funcione, primeiro precisamos de um nome de host que seja resolvido com sucesso na estação do aluno, para ela mesma. Em vez de nos dar ao trabalho de fazer isso cadastrando o nome algum servidor DNS, vamos usar a resolução local de nomes do Unix, o arquivo /etc/hosts. Acrescente a linha:

1. 127.0.0.1 hostvirtual

Confirme que o novo nome de host funciona com o comando ping.

Seguindo a estrutura do JPackage, crie a pasta /var/lib/tomcat6/maiswebapps, que deve pertencer ao grupo “tomcat” e ter permissão de escrita para o grupo. Os passos são basicamente os mesmos que utilizamos no capítulo 7 quando configuramos o diretório meu-lib. Lembre-se também do link simbólico em /usr/share/tomcat6!

Depois edite o arquivo server.xml e acrescente a seguinte definição de host, mas dentro do único elemento <Engine> e ao lado (no mesmo nível) do elemento <Host name=”localhost”>, e reinicie o Tomcat:

1. <Host name="hostvirtual"
2. appBase="maiswebapps"

autoDeploy="true" />

Para que seja possível instalar alguma coisa neste host, por enquanto é necessário usar o auto-deploy. Então peque qualquer exemplo deste curso que não tenha dependências em outras configurações do Tomcat, por exemplo o exemplo do laboratório 2 deste capítulo.

Feita a cópia, deve aparecer uma mensagem no log do Tomcat confirmando deployment bem-sucedido, e a URL http://hostvirtual:8080/hoje-valve deverá funcionar.

O novo host ainda não tem Manager, e você pode comprovar que o Manager só afeta seu próprio host fazendo o undeploy da aplicação “hoje-valve” no host “localhost”. Ela continuará funcionando no host “hostvirtual”.

Então, para ter realmente o novo host totalmente funcional no Tomcat, temos que colocar um Manager nele. Vamos definir um contexto explícito (capítulo 6) apontando para o pacote do Manager que já está em webapps, que é o appBase do host “localhost”. Os seguintes passos fazem o trabalho na instalação via Jpackage:

# mkdir /etc/tomcat6/Catalina/hostvirtual

# cp /var/lib/tomcat6/webapps/manager/META-INF/context.xml /etc/tomcat6/Catalina/hostvirtual/manager.xml

O arquivo manager.xml criado pela cópia do context.xml do pacote WAR (aberto) do Manager deve ser editado como se segue:

1. <Context antiResourceLocking="false" privileged="true"

**docBase="../webapps/manager"** />

O caminho relativo para o docBase funciona desde que webapps e maiswebapps sejam diretórios “irmãos”. Mas também seria possível indicar o caminho absoluto, no caso /var/lib/tomcat6/webapps/manager.

Acesse esta aplicação no host virtual usando uma URL como http://hostvirtual:8080/manager/html e verifique que a relação de aplicações deployadas inclui somente o próprio Manager e o “hoje-valve”. Afinal as nem aplicações de exemplo do próprio Tomcat, nem a página inicial (/) e nem os exemplos deste curso que você esqueceu de desinstalar não estão no novo host virtual.

## Conclusão

Com os recursos de firewall e SSL, encerramos o tópico de “segurança” no Tomcat, e você tem condições de colocar seu servidor em produção sem medo, exceto pelas vulnerabilidades embutidas nas próprias aplicações. Em Java são menos do que normalmente haveriam em aplicações ASP, PHP ou Perl, mas existem! Consultem o site OWASP.org para saber sobre os riscos e como preveni-los.

Questões de Revisão

* O Tomcat é capaz de operar com certificados SSL gerados por uma autoridade certificadora interna, ou exige certificados assinados pelas CA's oficiais, como a Verisign?

* Por que, no último exercício deste capítulo, foi possível acessar o Manager em “hostvirtual” com o mesmo login e senha utilizado para o localhost?

# Integração com Servidores Web Nativos

Neste capítulo somos apresentados aos recursos de integração do Tomcat com servidores web nativos, especialmente o Apache, e vemos como estes recursos podem ser utilizados para melhorar a integração ou segurança do ambiente Java EE

Tópicos:

* Integração com Apache Httpd
* Instalação do mod\_jk
* Dividindo tarefas entre o Apache e o Tomcat

## Porque usar o Tomcat junto a um servidor web nativo

Há várias situações onde pode ser vantajoso rodar o Tomcat integrado a um sevidor web nativo, por exemplo:

* Usar aplicações Java EE no mesmo site que já contém partes usando outras tecnologias, por exemplo CGI, PHP, ASP ou Cold Fusion;
* Aplicações que demandam especializados para o servidores web nativo, como streaming multimídia e SIP (padrão Internet para VoIP e URA[[21]](#footnote-21)) – mas note que a maioria destas aplicações também possuem equivalentes 100% Java;
* Otimização de performance usando proxies reversos com cache de páginas;
* Colocar o servidor nativo na DMZ, onde há maior exposição a ataques, e manter o servidor de aplicações dentro da rede local da empresa, protegido da DMZ por um firewall;
* Usar o servidor nativo como distribuidores de carga em um cluster.

Antigamente também era comum rodar o Tomcat integrado a servidores nativos porque esperava-se que servidores web nativos tenham melhor performance que o Tomcat em situações onde o volume de E/S de rede ou disco fosse mais intenso, por exemplo:

* Grandes quantidades de usuários simultâneos, navegando em sites na sua maior parte estáticos;
* Download de arquivos volumosos, como músicas, vídeos e relatórios em PDF;
* Grandes quantidades de conexões encriptadas.

Mas de todas elas apenas a última ainda representa um “ganho” garantido, isso se forem usadosde aceleradores de hardware para criptografia SSL. Nos demais casos, versões recentes da JVM e do Tomcat tem performance e escalabilidade comparáveis ou mesmo superiores à de servidores web nativos. Consulte, por exemplo, os benchmarks publicados na 2ª edição do livro de Administração do Tomcat publicado pela editora O'Reilly.

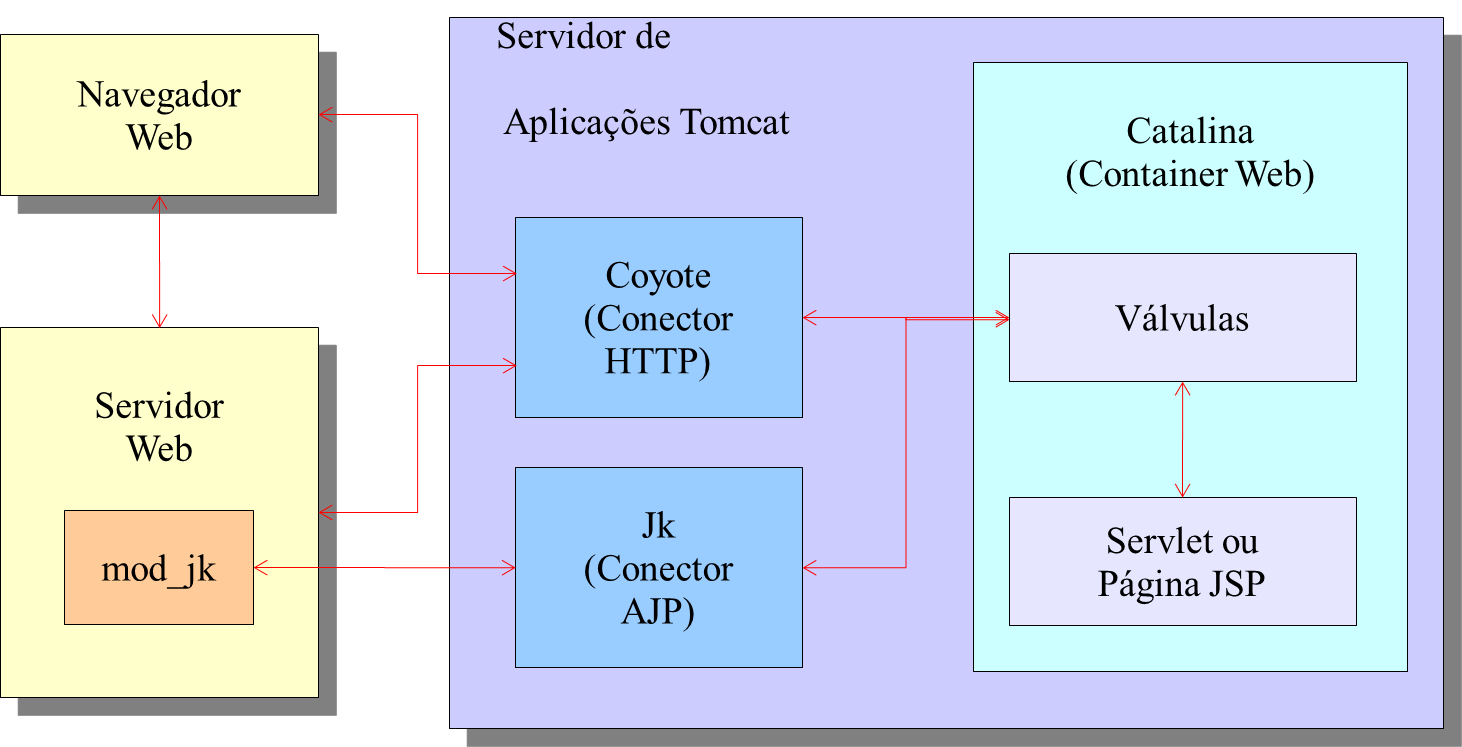
Um mito comum é que o Java é interpretado, por isso aplicações Java seriam necessariamente mais lentas do que aplicações equivalentes escritas em C. Mas na verdade o Java é compilado, e roda com código de máquina nativo do processador. A diferença é que linguagens como C realizam a compilação antecipada (AoT – Ahead of Time) enquanto que JVMs realizam a compilação apenas durante a execução da aplicação (JIT – Just In Time).

É por isso que aplicações Java demandam mais memória do que aplicações nativas, e demoram mais a iniciar. Estes são problemas legítimos para a popularização do Java em aplicações Desktop, onde o que se vê são aplicações para nichos especializados como o IDE Eclipse. Mas não são problemas para o ambiente de servidor, onde aplicações permanecem rodando sem reinício por dias ou mesmo semanas; e o consumo de memória “extra” da JVM é insignificante perto do consumo de memória para dados das aplicações.

## Sobre o mod\_jk

O mod\_jk é um plug-in de servidor web, mantido pela mesma comunidade que desenvolve o Tomcat, que acrescenta o suporte ao protocolo AJP. Ele é a ponta “cliente” para o conector Jk, que faz a ponta “servidora” do AJP.

A figura 1.1, que já foi apresentada no capítulo anterior, apresenta o fluxo de tratamento de uma requisição HTTP pelo Tomcat, mostra graficamente o papel do mod\_jk (ou do mod\_proxy\_ajp) em relação ao Conector Jk. Existe também uma seta diretamente do servidor web para o conector Coyote porque também é possível fazer esta integração por redirecionamento de URLs, utilizando HTTP, mas a performance não é tão boa quanto com o AJP.



**Figura 1.9** – fluxo de processamento de uma requisição HTTP pelo Tomcat

Também é possível usar o mod\_proxy\_ajp do Apache, que acrescenta ao mod\_proxy padrão o suporte a AJP. Os recursos são basicamente os mesmos do mod\_jk, entretanto o segundo tem a vantagem de poder ser compilado também como plug-in para os servidores web IIS da Microsoft e iPlanet da Sun (antigo Netscape Enterprise Server).

A desvantagem do mod\_jk é que ele não está incluso na maioria das distribuições do Linux (embora esteja normalmente presente nas distribuições comerciais como RHEL e SuSE). O motivo é que as distribuições consideram redundante oferecer o mod\_jk quando o Apache já traz o mod\_proxy\_ajp.

Mas o mod\_proxy\_ajp utiliza o mod\_jk como upstream, então novos recursos estarão implementados primeiro no mod\_jk.

Existe um mod\_jk2 que entretanto foi descontinuado já há alguns anos, portanto o mod\_jk 1.2.x é a série mais recente do mod\_jk!

## Instalação do mod\_jk

Como é improvável que o mod\_jk esteja incluso como um pacote padrão na sua distribuição do Linux, e também é improvável que você consiga baixar um mod\_jk binário para a sua combinação exata de arquitetura de processador, versão da biblioteca C (do sistema operacional) e versão do Apache HTTPd, o jeito é compilar o mod\_jk à partir dos contes, que podem ser baixados no link “Tomcat Connectors” da página oficial do Tomcat em http://tomcat.apache.org.

O arquivo a ser baixado é o tomcat-connectors-1.2.23-src.tar.gz ou mais recente. Este mesmo conjunto de fontes serve para qualquer versão do Apache, e também para outros servidores.[[22]](#footnote-22)

O pré-requisito para a compilação do mod\_jk é a disponibilidade do apxs, que é parte do Apache, Em sistemas Linux ele costuma estar em /usr/sbin/apxs, mas só é instalado como parte do pacote de desenvolvimento do Apache, que é http-devel no Fedora e assemelhados.

É claro, devem estar disponíveis as dependências do próprio http-devel, por exemplo apr-devel e apr-util-devel (o comando yum cuida de localizar e instalar todas essas dependências). Também devem estar disponíveis os comandos para desenvolvimento em linguagem C, pelo menos o próprio compilador C (o pacote gcc), o GNU Make (pacote make) e os GNU Bintuils (pacote binutils).

Note que em nenhum momento pedimos pelos fontes do próprio Apache, que seri no Fedora um pacote srpm. O Apache em si não necessita ser recompilado para a compilação de um módulo de extensão, então basta compilar o próprio mod\_jk!

Com todas as dependências satisfeitas, descompacte então os fontes do mod\_jk em sua pasta home. Entre na pasta native e dentro dela rode o utilitário configure. Em seguida, compile tudo com make, e apenas então mude para o “root” e rode make install para inserir o binário do mod\_jk no Apache da sua distribuição do Linux.

$ cd $HOME

$ tar xzf tomcat-connectors-\*.tar.gz

$ cd tomcat-connectors-\*-src

$ cd native

$ ./configure --with-apxs=/usr/sbin/apxs

$ make

$ su

# make install

Ainda é necessário configurar o mod\_jk antes que possamos reiniciar o Apache e testar a conectivdade dele com o Tomcat.

## Configuração do mod\_jk

A configuração do mod\_jk é realizada em três partes:

1. Configuração do módulo para o Apache em si, nos arquivos de configuração do Apache HTTPd;
2. Configuração de conexões com instâncias do Tomcat, no arquivo worker.properties;
3. (Opcional) configuração de URLs mapeadas para o Tomcat, no arquivo uriworkermap.properties.

Para realizar a primeira parte, podemos ou editar diretamente o httpd.conf ou então criar um novo arquivo na pasta /etc/httpd/conf.d. A segunda opção é considerada melhor prática, mas poderá não estar configurada por padrão na instalação do Apache que acompanha sua distribuição do Linux ou outro SO.

A listagem 1.1 exibe o conteúdo do arquivo /etc/httpd/conf.d/jk.conf. O mesmo conteúdo deverá ser colocado no arquivo de configuração do Apache que você preferir utilizar para a configuração de módulos de extensão.

**Listagem 1.1** exemplo de configuração do mod\_jk em /etc/httpd/conf.d/mod\_jk.conf:

1. LoadModule jk\_module modules/mod\_jk.so
2. JkWorkersFile /etc/httpd/conf.d/workers.properties
3. JkLogFile /var/log/httpd/mod\_jk.log
4. JkLogLevel error
5. JkMount /hoje-valve no0

JkMount /hoje-valve/\* no0

O exemplo na listagem já inclui diretivas JkMount, que serão explicadas mais adiante, para a execução do aplicativo “hoje-valve” que foi deployado no capítulo anterior. Cada aplicação deployada no Tomcat que se deseje esteja acessível via Apache tem que ser mapeada individualmente.

Para realizar a segunda parte da configuração do mod\_jk, é necessário fornecer o arquivo de configuração do próprio mod\_jk, indicado pela diretiva JkWorkersFile, no caso /etc/httpd/conf.d/workers.properties. Um exemplo aparece na listagem 1.2.

**Listagem 1.2** exemplo de configuração de workers em /etc/httpd/conf.d/workers.properties

worker.list=no0

1. worker.no0.type=ajp13
2. worker.no0.host=127.0.0.1

worker.no0.port=8009

O nome dado à instância do Tomcat, no caso “no0”, poderia ser qualuer um, mas ele tem que ser sempre colocado dentro das diretivas “worker.nome\_da\_instancia.propriedade”. Observe também que este nome é o mesmo que foi colocado na configuração do Apache, como argumento da diretiva JkMount.

O exemplo permite acesso a um servidor Tomcat instalado no mesmo computador que o servidor apache (host=127.0.0.1) e usando a configuração padrão do conector jk que já vem habilitada na configuração default do Tomcat (port=8009).

Em distribuições com SELinux (como é o caso do Fedora) a política de segurança default para o Apache poderá interferir com o mod\_jk. Então desabilite o SELinux com o comando setenforce 0.

Então basta reiniciar o Apache (service httpd restart), e a aplicação “hoje-valve” estará disponível por intermédio do Apache, por uma URL sem porta como http://127.0.0.1/hoje-valve.

Se não funcionar:

* Confirme que o apache esteja realmente rodando, acessando a página http://127.0.0.1/;
* Confirme que a aplicação desejada realmente disponível no Tomcat, acessando a página http://127.0.0.1:8080/hoje-valve;
* Verifique o log de erros do Apache (/var/log/httpd/error\_log);
* Verifique o log de acesso do Apache (/var/log/httpd/access\_log); se ele responder 404 para aplicações no Tomcat, é porque ele não passou estas requisições para o mod\_jk;
* Aumente o nível de log do mod\_jk, alterando a diretiva JkLogLevel para debug no arquivo mod\_jk.conf ;
* Verifique o log do mod\_jk (/var/log/httpd/mod\_jk.log);
* E confirme que o SELinux esteja desabilitado com getenforce.

### Dividindo tarefas entre o Tomcat e o Apache

A configuração apresentada repassa para o Tomcat todas as requisições direcionadas para uma determinada aplicação web. É possível ser mais seletivo, pelo uso das diretivas JkMount e JkUmount.

Por exemplo para deixar que o Apache cuide de arquivos estáticos como páginas HTML e imagens. Já explicamos que provavelmente não haverá ganho de desempenho significativo, e provavelmente haverá até piora, mas este é um exemplo muito requisitado, e pode ser interessante quando o Apache está dentro da DMZ e o Tomcat fora.

Precisamos da aplicação deployada com um pacote aberto, e configurar o Apache para acessar o mesmo diretório do pacote aberto, ou então ter uma cópia deste pacote em algum lugar acessível pelo Apache.

Se a aplicação sofreu auto-deploy como “paginas”, basta acrescentar ao arquivo mod\_jk.conf as linhas:

1. JkMount /paginas/\*.jsp no0

Alias /paginas /var/lib/tomcat6/paginas

E reiniciar o Apache para ter a aplicação disponível pela URL http://127.0.0.1/paginas.

Para comprovar a divisão de tarefas entre o Apache e o Tomcat, deve-se verificar os logs de acesso de cada um, para ver quais páginas foram respondidas apenas pelo Apache e quais foram respondidas por ambos (páginas efetivamente respondidas pelo Tomcat também são relacionadas no log de acesso do Apache).

Usar extensões de arquivos na diretiva JkMount pode não ser muito prático para Servlets, por isso veremos uma outra alternativa, fornecida pela diretiva JkUmount. Com ela, podemos redirecionar todo o contexto para o Tomcat, e indicar as exceções (que serão servidas diretamente pelo Apache). Neste caso, a diretiva JkMount da listagem anterior seria substituída por:

1. JkMount /paginas no0
2. JkMount /paginas/\* no0
3. JkUnMount /paginas/\*.html no0

JkUnMount /paginas/\*.gif no0

JkMount /exemplo5.1 no0

Ou, se sua aplicação segue a convenção de ter subdiretórios separados para imagens, página estáticas e etc, as diretivas poderiam ser:

1. JkMount /paginas no0
2. JkMount /paginas/\* no0
3. JkUnMount /paginas/html/\* no0

JkUnMount /paginas/imagens/\* no0

Note que, neste caso, uma página inicial /index.html seria servida pelo Tomcat! Então, caso sua página inicial não seja uma página JSP, acrescente mais uma diretiva JkUmount para o diretório raiz do contexto.

## Integração de Segurança e Logging

O uso do Apache como front-end para o Tomcat oferece várias oportunidades de otimização de desempenho (ex: uso do mod\_proxy\_cache), mas acaba complicando um pouco a configuração e administração de ambos.

Aplicações com controle de acesso devem preferencialmente ser servidas inteiramente pelo Tomcat, a não ser que as partes estáticas (como imagens) possam ser consideradas de acesso público, pois embora o Apache reconheça autenticação HTTP, ele não reconhece FORM. O Apache também não toma conhecimento dos <security-constrains> definidos no web.xml do pacote WAR.

Não há necessidade de habilitar SSL no Tomcat; basta faze-lo no Apache. O SSL deve ser necessariamente tratado por quem tiver a conexão direta com o navegador, e não compensa o overhead de colocar SSL na comunicação entre o Apache e o Tomcat via AJP.

Por sinal, o AJP não possui nenhum recurso de segurança; apenas o firewall de rede (ou do SO) pode proteger o conector Jk do Tomcat.

Em princípio, basta configurar o log de acessos do Apache, já que ele registra tanto as URLs servidas por ele quanto as servidas pelo Tomcat. Entretanto, o log de acessos do Tomcat pode ser customizado para registrar informações que o Apache não tem como obter, por exemplo tempo de CPU para atender à requisição.

Já erros de aplicação e banco de dados estarão nos logs do Tomcat, não do Apache.

## Exercícios

Laboratório 1.1. Integração Apache com Tomcat

(Prática Dirigida)

**Objetivo:** Instalar e configurar o mod\_jk, de modo que as aplicações deployadas no Tomcat apareçam como parte do site servidor pelo Apache.

Baixe do computador do instrutor arquivo tomcat-connectors-\*.tar.gz e siga as instruções já apresentadas neste capítulo para sua compilação e instalação no Apache.

Em seguida, use os arquivos-modelo de configuração para o Apache e mod\_jk que estão no diretório Cap1/Lab1 e assim configure o Apache pare repassar ao Tomcat as URLs iniciadas por “/hoje-valve”.

Reinicie o Tomcat, desligue o SELinux e teste a aplicação via Apache.

Laboratório 1.2. Servindo arquivos estáticos no Apache

(Laboratório)

**Objetivo:** Configurar uma aplicação de modo que seus arquivos estáticos sejam servidos diretamente pelo Apache, mas páginas JSP sejam executadas pelo Tomcat.

Antes de iniciar este exercício, tenha certeza de que a válvula de log de acessos esteja habilitada na sua instalação do Tomcat. Caso não esteja, consulte as instruções no capitulo 11.

Este exercício traz um exemplo onde temos páginas HTML, imagens e páginas JSP em diretórios separados. Queremos que apenas o diretório de páginas JSP seja redirecionado para o Tomcat, e que o restante seja executado diretamente pelo Apache. Mas não queremos ter que copiar para o Apache os arquivos da aplicação, que estarão deployados no Tomcat.

Inicie deployando a aplicação como um pacote aberto, no contexto “pacotes”

$ cd ~Cap1/Lab2

$ cp -r \* /var/lib/tomcat6/webapps/paginas

E então teste o funcionamento correto da aplicação no Tomcat.

Em seguida, mapeie o contexto na configuração em /etc/httpd/conf.d/mod\_jk.conf, acrescentando a diretiva:

Alias /paginas /var/lib/tomcat6/paginas

Agora reinicie o Apache. Sabemos que a configuração assim está incompleta, mas queremos neste momento ver que as páginas JSP são servidas diretamente pelo Apache e por isso não são executadas corretamente.

Entre na aplicação pela URL http://127.0.0.1/paginas e siga todos os links. Observe como a página JSP falha em exibir a data corrente. Verifique os logs de acesso, e confirme que nenhuma requisição desta aplicação chegou ao Tomcat.

Agora sim vamos configurar o redirecionamento para o Tomcat, acrescentando as linhas a seguir no mod\_jk.conf:

1. JkMount /paginas no0
2. JkMount /paginas/\* no0
3. JkUnMount /paginas/html/\* no0

JkUnMount /paginas/imagens/\* no0

Reinicie o Apache, e acesse novamente a aplicação. Se você fez tudo corretamente, agora a página JSP será exibida corretamente, mas os demais links não irão gerar entradas no log de acesso do Tomcat.

## Conclusão

Agora que temos a integração entre Apache e Tomcat funcionando, estamos com meio caminho andado para configurar nosso primeiro cluster Tomcat, que será o tema do próximo capítulo.

Questões de Revisão

* É possível instalar o container web do Tomcat (Catalina) dentro do Apache para execução de Servlets e páginas JSP, tal qual é feito, por exemplo, com o PHP?

* O mod\_jk utiliza JNI para fazer a comunicação com o conector Jk?

# Clusters para Escalabilidade

Neste capítulo vemos como configurar clusters Tomcat para balanceamento de carga, isto é, clusters que trazem escalabilidade, mas não alta disponibilidade (que será o tema do próximo capítulo)

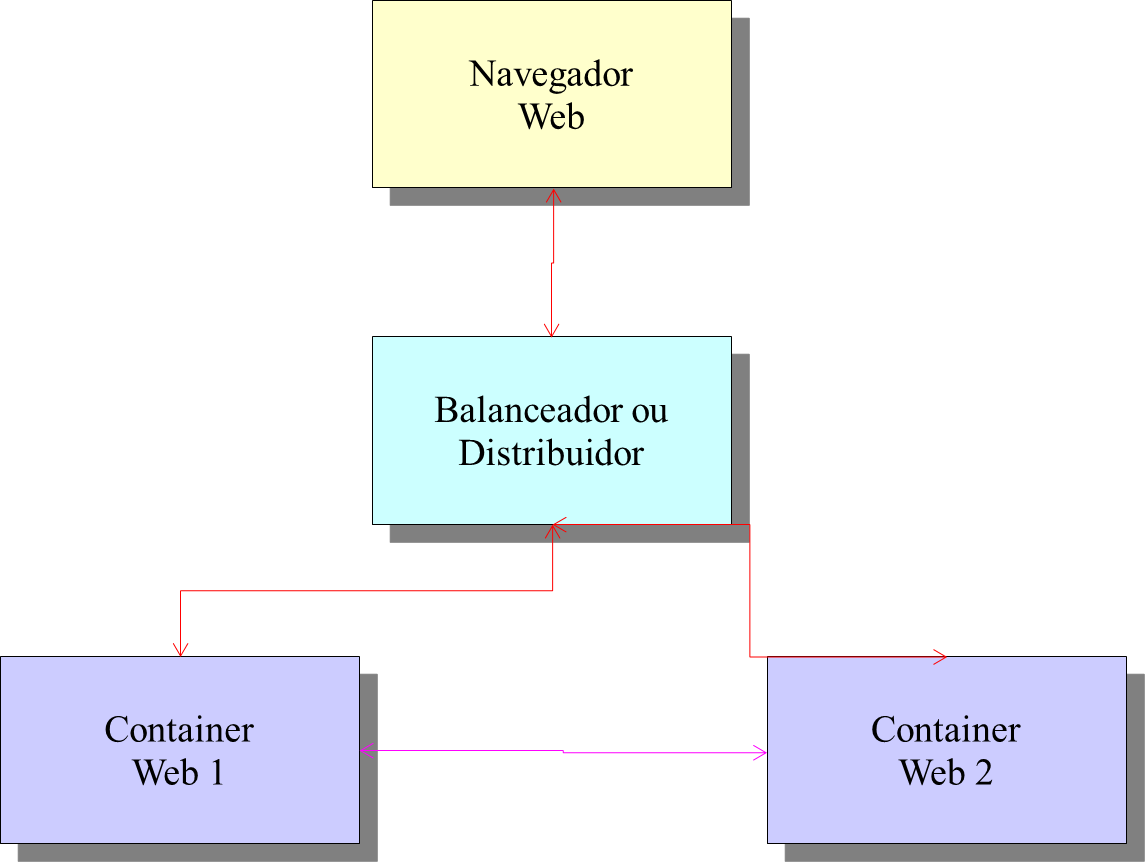
Tópicos:

* Conceitos de clusters Java EE
* Rodando múltiplas instâncias do Tomcat
* Configurando o mod\_jk como balanceador
* Página de status do cluster

## Conceitos de clusters Java EE

A plataforma Java EE prevê desde a sua concepção a execução de aplicações em ambiente de cluster. Este recurso vem quase “de graça” para o desenvolvedor, desde que ele obedeça às regras previstos pela especificação de Servlets e siga as melhores práticas da plataforma; enquanto que outros ambientes de desenvolvimento web ele exige programação cuidadosa e especializada.

No caso de aplicações web, a natureza do protocolo HTTP ao mesmo tempo facilita e impõe restrições. A figura 1.1 apresenta diagrama simplificado do que forma umcluster de servidor de aplicação Java EE, sob o ponto de vista do container web.



**Figura 2.0** – arquitetura de um cluster web Java EE

Observe na figura que existe um componente entre o navegador web e os containers web que formam o cluster. Este componente é externo ao servidor de aplicações, e tem que existir por causa de limitações do protocolo HTTP.

Note também que a seta entre os containers web é de cor diferente das outras setas. Isto indica que há uma comunicação especializada para sincronizar os containers que estão em cluster, de modo que um possa assumir tarefas do outro em caso de falha.

Neste capítulo cuidamos apenas da configuração envolvendo o balanceador, de modo que nosso cluster não terá capacidade de tolerância à falhas. No próximo capítulo será configurada a sincronização entre os servidores Tomcat no cluster, adicionando assim ao nosso cluster recursos de alta disponibilidade.

### Estado em uma aplicação web Java EE

O HTTP é um protocolo sem estado, então em princípio qualquer requisição poderia ser entregue indistintamente a qualquer nó do cluster. Na verdade, um mesmo usuário poderia ser atendido por um servidor diferente a cada página.

Entretanto, aplicações web reais utilizam sessões HTTP para manter o estado de um usuário particular (por exemplo, uma cesta de compras em uma loja on-line). Isto exige que ou o usuário permaneça sempre no mesmo servidor, ou então que seu estado seja de alguma forma disponibilizado para os outros nós do cluster.

A especificação Java EE não impõe uma maneira específica de se disponibilizar este estado, então diferentes servidores de aplicações são livres para implementar estratégias baseadas em replicação ou em armazenamento compartilhado. Mas a especificação diz claramente como isto tem que funcionar sob o ponto de vista da aplicação, de modo que ela possa funcionar corretamente em clusters de qualquer servidor de aplicações Java EE do mercado.

### O Balanceador em um cluster Java EE

O balanceador ou distribuidor de carga é o componente que recebe as requisições do navegador e as encaminha para um dos nós do cluster, idealmente procurando dividir a carga de trabalho entre os servidores e detectando falhas nesses servidores, evitando que os navegadores continuem tentando acessar servidores que estão fora do ar.

Ele cria para o usuário no navegador a ilusão de que existe apenas um único servidor web em vez de vários servidores de aplicação com seus containers web independentes entre si.

Há várias estratégias possíveis para a implementação de um balanceador para aplicações web:

* Switches e roteadores especializados - “switches de layer 7”
* Múltiplos endereços IP para um mesmo nome DNS (DNS round-robin)
* Proxy reverso ou servidor web de font-end

Não assuma que seu balanceador de rede, utilizado para clusters de servidores de e-mail, seja automaticamente a melhor opção para o cluster Tomcat.

### Arquitetura de cluster web do Java EE

Sessões HTTP são implementadas por meio de um cookie “identificador de sessão” conforme manda a especificação de Servlets. E, como já vimos, para que um usuário possa recuperar as informações armazenadas na sua sessão ele deve ser atendido sempre pelo mesmo servidor.

Então o balanceador tem que reconhecer o cookie de sessão e usa-lo para “amarrar” as sessões ao mesmo nó do cluster. Um usuário com uma sessão ativa só irá mudar de nó caso o seu nó original seja derrubado.

## Configurando o mod\_jk como balanceador

O mod\_jk do Apache pode ser configurado para atuar como um balanceador de carga entre vários servidores Tomcat.

Na verdade, ele não fica devendo nada a outros balanceadores do mercado, pois é capaz de empregar várias estratégias de balanceamento de carga, inclusive baseado no uso de processador em cada servidor do cluster!

O mod\_jk também é capaz de lidar com servidores de capacidades diferentes e distribuir a carga (usuários / sessões HTTP) proporcionalmente. O melhor é que os servidores do cluster podem usar sistemas operacionais diferentes, então seu cluster pode ter um nó Linux, outro Solaris e um terceiro com Windows!

O primeiro passo é configurar individualmente cada servidor do cluster como um “worker” no worker.properties do mod\_jk. Entretanto, nenhum dos servidores deve ser relacionado na diretiva workers no início do arquivo.

Em seguida, deve ser definido um novo worker, do tipo “lb” (de load balancer). É este worker quem irá relacionar os demais workers (servidores Tomcat) que são parte do mesmo cluster.

Os mapeamentos de URLs do Apache para o Tomcat deve referenciar o worker do tipo balanceador, e nunca diretamente os workers que representam nós individuais do cluster.

A listagem 1.1 ilustra um arquivo worker.properties para uma configuração de cluster, e a listagem 1.2 apresenta as diretivas JkMount correspondentes no mod\_jk.conf.

**Listagem 1.1** – arquivo worker.properties definindo um worker clusterizado e seus respectivos membros

1. worker.list=cluster
2. worker.cluster.type=lb
3. worker.cluster.balance\_workers=no0,no1
4. worker.cluster.sticky\_session=true
5. worker.no0.type=ajp13
6. worker.no0.host=127.0.0.1
7. worker.no0.port=8009
8. worker.no0.lbfactor=1
9. worker.no1.type=ajp13
10. worker.no1.host=127.0.0.1
11. worker.no1.port=8109

worker.no1.lbfactor=1

**Listagem 1.2** – mapeamento de uma aplicação em cluster no mod\_jk.conf

1. JkMount /contador cluster

JkMount /contador/\* cluster

A diretiva lbfactor indica a capacidade relativa de cada nó do cluster. Os valores absolutos não importam, o que interessa é a proporção entre eles. Então, um servidor com o dobro da capacidade deve receber o valor 2, enquanto que os demais recebem o valor 1.

A aplicação “contador”, que foi mapeada para o worker clusterizado, deve estar deployada em cada servidor individualmente.

As configurações no mod\_jk não são suficientes para gerar um cluster funcional. Os servidores Tomcat também deve ser configurados para que trabalhem junto com o balanceador.

Mais especificamente, é necessário informar ao tomcat sobre o identificador de nó, que deve ser exatamente o nome do worker correspondente em worker.properties.

Para tal, edite o sever.xml de cada um dos dois nós do cluster, acrescentando no elemento <Engine> o atributo jvmRoute, como exemplificado pela listagem 1.3.

**Listagem 1.3** – configurando o nome do nó (worker) no server.xml

1. <Engine name="Catalina" defaultHost="localhost" **jvmRoute="no0"**>

É importante que cada servidor Tomcat, e suas respecitvas aplicações clusterizadas, devem estar funcionando isoladamente antes que possam funcionar como parte de um cluster.

Uma vez que todas as configurações estejam prontas, é necessário reiniciar tanto o servidor Apache quanto cada servidor Tomcat do cluster.

### A página de status do mod\_jk

O mod\_jk é capaz de gerar por conta própria uma página de status dos nós do cluster, útil para administradores. Para ativar esta página, mapeie uma URL para um worker do tipo “status”, editando o mod\_jk.conf e o worker.properties de acordo. As listagens 1.4 e 1.5 apresentam as edições necessárias.

**Listagem 1.4** – modificações no arquivo worker.properties definindo um worker de status do cluster

1. worker.list=cluster,status

worker.status.type=status

**Listagem 1.5** – modificações no arquivo mod\_jk.conf configurando uma página de acesso para oum worker de status do cluster

1. JkMount /jk status

Como não houveram modificações sobre as configurações dos servidores Tomcat do cluster, basta reiniciar o Apache para habilitar a página de status, que pelo exemplo estará disponível em http://127.0.0.1/jk.

## Configurando novas instâncias do Tomcat no mesmo servidor

Em uma instalação manual do Tomcat, é fácil rodar várias instâncias do Tomcat. Basta fazer uma nova instalação em um diretório diferente, e mudar as portas TCP definidas nos conectores e no elemento <Server>, que é utilizada para o shutdown do servidor.

Na instalação via JPackage, as coisas ficam um pouco mais complicadas devido à necessidade de se preservar o layout de arquivos e permissões padrão LSB e pela organização diferente dos scripts de inicialização.

Aparentemente os scripts de inicialização do JPackage prevêem uma forma “fácil” de se configurar e iniciar uma segunda instância, mas ela simplesmente não funciona.

Por isso o diretório de exemplos do primeiro exercício deste capítulo fornece o script novo\_tomcat.sh para criar novas instâncias, que inicialmente são “clones” da instância padrão em /usr/share/tomcat6, e configuradas em /usr/share/tomcat\_nome-da-instancia.

As novas instâncias compartilham com a instância padrão os mesmos arquivos nas pastas bin e lib, e iniciam com uma cópia idêntica dos arquivos em conf e webapps. O script também gera um script de start e stop em init.d.

Então, se sua instância padrão está customizada em termos de configurações de logs, usuários administrativos, realms, DataSources e etc, estas configurações serão replicadas na nova instância. Mas diretórios adicionais, como o meu-lib que foi criado nos exercícios do capítulo 7, não serão replicados e deverão ser recriados manualmente.

O script não irá mudar portas TCP, o que também terá que ser feito manualmente. E não testa por instâncias com o mesmo nome.

Seja com a instalação manual ou via JPackage (reforçada pelo nosso script de criação de novas instâncias) a listagem 1.6 mostra as alterações necessárias para evitar conflitos de portas TCP:

**Listagem 1.6** – alterações no server.xml do Tomcat para evitar conflitos de portas TCP com outras instâncias no mesmo servidor.

<Server **port="8105"** shutdown="SHUTDOWN">

1. ...
2. <Connector **port="8180"** protocol="HTTP/1.1"
3. ...
4. <Connector **port="8109"** protocol="AJP/1.3"

...

Uma sugestão é usar o dígito da centena no número de porta, de modo a manter os números mnemônicos. Assim a instância padrão usaria as portas 8085, 8080 e 8009; a primeira instância extra usaria as portas 8185, 8180 e 8109; a segunda instância usaria as portas 8285, 8280 e 8209, e assim em diante.

Apenas pelas páginas do Manager não é possível identificar que instância está sendo gerenciada, pois ela não exibe nenhum “nome” da instância, nem o diretório de instalação da instância. A única “dica” de qual instância está sendo gerenciada é o número de porta na URL, por exemplo:

* http://127.0.0.1:**8080**/manager/html (instância padrão)
* http://127.0.0.1:**8180**/manager/html (primeira instância customizada)

Instâncias configuradas desta forma, se não estiverem em cluster, irão interferir uma com a outra, caso estejam rodando as mesmas aplicações. Mas especificamente, irão fazer com que o navegador “confunda” a seções HTTP.

Isto é um problema com a forma como o navegador lida com o cookie de sessão, vinculado ao host e aplicação, que são os mesmos nas duas instâncias. Para evitar este problema, cada instância deve ser configurada com um nome de host (virtual host) diferente.

### Vantagens de se rodar várias instâncias do Tomcat

Neste curso vamos rodar várias instâncias do lado-a-lado no mesmo servidor para poder testar as configurações de cluster. Entretanto este é um cenário mais comum do que se imagina. Entre os motivos para se ter várias instâncias do Tomcat temos:

* Cada uma tem configurações customizadas e permissões de acesso para usuários diferentes, por exemplo uma para desenvolvimento de aplicações, outra para homologação pelo usuário final;
* As instâncias rodam versões diferentes do Tomcat, porque alguma aplicação não funciona corretamente em versões mais novas (isto é um bug da aplicação, mas não raro o administrador tem que “acomodar” estas falhas);
* As várias instâncias estão em cluster, de modo que uma delas possa se derrubada, para atualização, por exemplo, sem interromper o serviço para o usuário.

Antigamente também era comum rodar várias instâncias de um servidor de aplicações para contornar limitações da JVM ou de escalabilidade do SO.

Hoje, JVMs recentes (JavaSE 1.4 ou superior) em versões mais ou menos recentes do Linux (kernel 2.4 ou superior) podem usufruir com eficiência de quanta memória e processadores estiverem disponíveis.

Por outro lado, sistemas de 32-bits ainda impõem limites para a quantidade de memória RAM disponível para a JVM, e sistemas Windows tem problemas de escalabilidade com a quantidade de threads.

## Exercícios

Laboratório 1.1. Duas instâncias do Tomcat lado-a-lado

(Prática Dirigida)

**Objetivo:** Rodar duas instâncias do Tomcat lado-a-lado no mesmo computador.

Este exercício prepara duas instâncias do Tomcat que serão depois clusterizadas. Encerre a instância padrão instalada via JPackage ,e utilize o script fornecido com este exemplo para criar as duas novas instâncias:

# service tomcat6 stop

# sh ~aluno/Cap1/Lab1/novo\_tomcat.sh no1

# sh ~aluno/Cap1/Lab1/novo\_tomcat.sh no2

Em seguida modifique o server.xml de cada instância, respectivamente em /etc/tomcat\_no1 e /etc/tomcat\_no2, e modifique as portas TCP segundo as instruções já fornecidas nesta apostila.

Então inicie cada instância com o comando service:

# service tomcat\_no1 start

# service tomcat\_no2 start

Verifique nos respectivos logs (catalina.out) em /var/log/tomcat\_no1 e /var/log/tomcat\_no2 se ambas tiveram início livre de erros. Em seguida, acesse o Manager em cada uma, verificando pelo link “Server Status” que cada uma está utilizando as portas TCP corretas.

Laboratório 1.2. Um cluster para escalabilidade

(Laboratório)

**Objetivo:** Configurar duas instâncias do Tomcat em cluster, sem alta disponibilidade.

Primeiro, utilize o buildfile fornecido com o exemplo deste laboratório para empacotar a aplicação de teste, que deverá ser instalada manualmente (usando o auto-deploy do Tomcat) nas duas instâncias configuradas no exercício anterior.

$ ant

$ cp dist/contador.war /var/lib/tomcat\_no1/webapps/

$ cp dist/contador.war /var/lib/tomcat\_no2/webapps/

Acesse então a aplicação em cada uma das instâncias:

* http://127.0.0.1:8180/contador
* http://127.0.0.1:8280/contador

Ela é um simples contador, cujo valor aumenta a cada atualização de página. Além disso, a página exibe o identificador da sessão corrente, para ajudar a entender o comportamento com balanceamento e no próximo capítulo também com replicação.

Observe que a página tem um título que serviria para identificar o servidor utilizado. Altere este título para “no2”, gere um novo pacote com o ant e faça o re-deploy no tomcat\_no2. Agora fica fácil reconhecer, apenas olhando para a página, qual servidor gerou o resultado.

Por fim,verifique que mudar de um servidor para o outro reinicializa a contagem. Este problema será resolvido com a configuração do cluster.

Então modifique as configurações do Apache conforme as instruções deste capítulo para configurar o mod\_jk como balanceador para um cluster formado pelas duas instâncias. O diretório do exemplo traz um modelo para o worker.properties para poupar tempo de digitação, mas você deverá acrescentar as diretivas JkMount em mod\_jk.conf.

Lembre-se também de definir o jmvRoute no <Engine> de cada instância do Tomcat. Então reinicie tudo, o Apache e as duas instâncias do Tomcat, e não esqueça de desabilitar o SELinux, caso contrário ele poderá impedir a conexão do Apache para o Tomcat.

# service httpd restart

# service tomcat\_no1 stop

# service tomcat\_no2 stop

# service tomcat\_no1 start

# service tomcat\_no2 start

# setenforce 0

Agora acesse a aplicação passando pelo Apache, vistando a URL http://127.0.0.1/contador. Note que agora o identificador da sessão inclui também o nome do worker, obtido pelo jvmRoute.

Acesse novamente a aplicação, usando a mesma URL, porém usando um programa navegador diferente, por exemplo o Galeon ou Epiphany. O resultado esperado é uma página indicando no identificador de sessão um nó diferente do cluster, demonstrando que o balanceador está realmente dividindo a carga, ou melhor, as sessões entre os dois usuários.

Laboratório 1.3. Status do cluster

(Laboratório)

**Objetivo:** Configurar e ativar a página de status do mod\_jk

A página de status do mod\_jk auxilia bastante na monitoração e troubleshooting do cluster, por isso vamos ativa-la, seguindo as instruções já apresentadas nesta apostila.

Basta definir o worker do tipo status no worker.properties e em seguida mapear alguma URL para ele no mod\_jk.conf. Reinicie o Apache, e visite a página de status em http://127.0.0.1/jk ou qualquer que seja a URL escolhida por você.

Laboratório 1.4. Desligando a afinidade de sessão

(Desafio)

**Objetivo:** Configurar duas instâncias do Tomcat em cluster, sem alta disponibilidade e também sem afinidade de seção, e observar o comportamento errático.

Para entender a importância do identificador do nó no cookie da sessão HTTP e do uso desta informação pelo balanceador, vamos deixar o balanceador alterar entre os nós como se fossem servidores web estáticos, sem aplicações.

Altere o arquivo workers.properties:

1. worker.cluster.sticky\_session=**false**

Em seguida reinicie o Apache e os dois Tomcats

Abra os dois navegadores e faça cada um acessar a aplicação passando pelo Apache (http://127.0.0.1/contador), do mesmo modo que fizemos no laboratório 2 deste capítulo.

Observe como a contagem sempre fica em “1”, com a requisição sendo atendida alternadamente por servidores diferentes no mesmo navegador

Veja também que o id de sessão muda a cada recarga da página, em ambos os navegadores. O navegador não consegue manter o vínculo com a sessão do usuário!

## Conclusão

Construímos um cluster para escalabilidade, que permite distribuir um gande universo de usuários entre vários servidores Tomcat, de modo que as aplicações não ficam limitadas pela capacidade de um servidor individual.

Mas este cluster não suporta ainda tolerância à falhas, pois se um servidor morrer, todas as sessões armazenadas nele são perdidas, e os usuários tem que reiniciar o que quer que estivessem fazendo nas aplicações.

No próximo capítulo, resolvemos esta limitação do nosso cluster.

Questões de Revisão

* Qualquer tipo de cluster garante a alta disponibilidade de uma aplicação?

* Porque é necessário fornecer um balanceador de rede para um cluster Tomcat?

* Um balanceador de rede sem conhecimento de HTTP, ou seja, um balanceador “puro” de camada 3, seria adequado para um cluster Tomcat?

# Cluster para HA – Alta Disponibilidade

Neste capítulo finalizamos a apresentação dos recursos de clustering do Tomcat, apresentando como configurar um cluster ativo-ativo onde a falha de um nó não seja percebida pelo usuário

Tópicos:

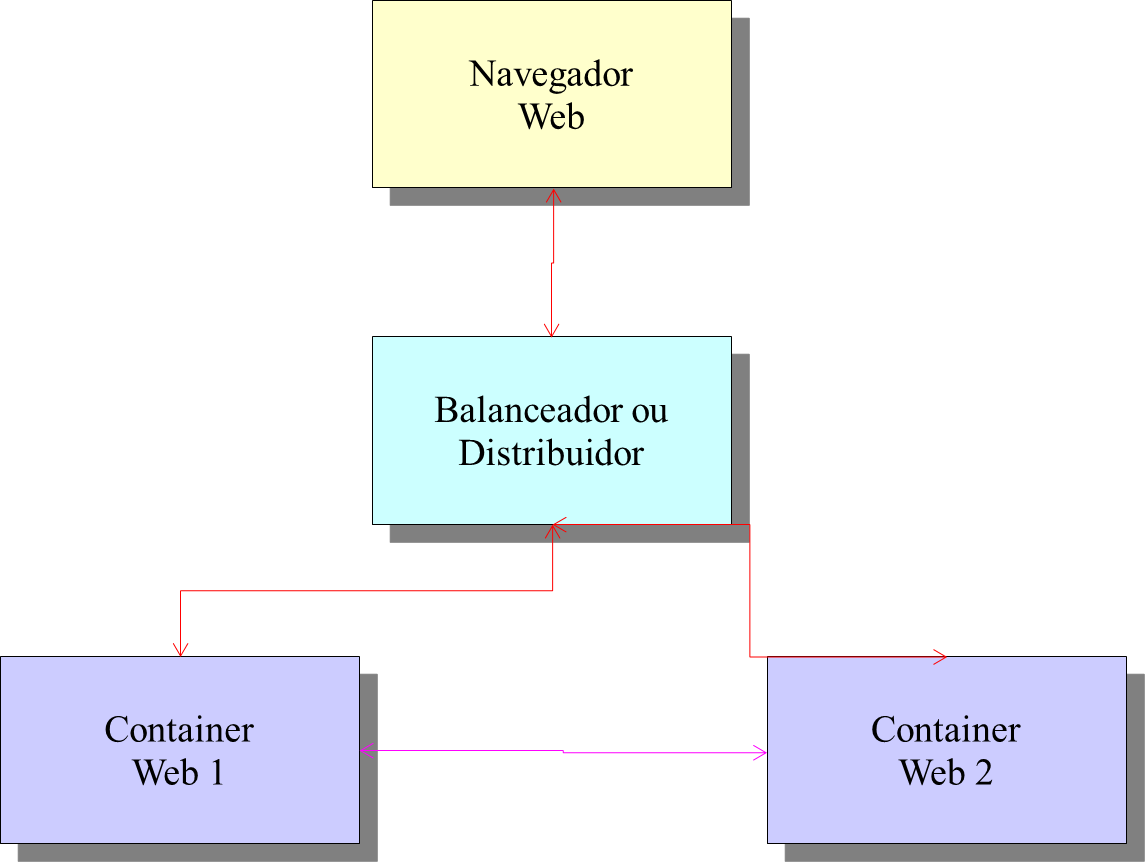
* Aplicações “cluster aware”
* Replicação de sessão HTTP

## Conceitos de clusters Java EE

Recapitulando, o objetivo agora é simular um cluster onde os usuários de um servidor Tomcat são automaticamente e transparentemente migrados para outro servidor, sem perda do estado da aplicação.

Para que isto ocorra, as informações armazenadas na sessão HTTP em cada nó devem ser disponibilizadas para os demais membros do cluster. Desta forma, qualquer nó pode dar continuidade a atividades iniciadas por outro nó

A figura 1.1 apresenta diagrama simplificado do que forma um cluster de servidor de aplicação Java EE, sob o ponto de vista do container web.

****

**Figura 2.1** – arquitetura de um cluster web Java EE

Já vimos esta figura no capítulo anterior, que tratou do balanceador, mas agora estamos tratando da sincronização entre os containers web que estão em cluster.

### Aplicações cluster-aware

Uma aplicação escrita segundo as normas e melhores práticas do Java EE deve funcionar em cluster para HA sem problemas, mas na vida real há várias considerações de modelagem, projeto e implementação que afetam o funcionamento correto e/ou eficiente de uma aplicação em cluster.

Por isso, o Java EE só considera como sendo clusterizadas as aplicações que trouxerem o elemento <distributable/> dentro do seu descritor padrão web.xml. Este elemento sinaliza para o container web que a aplicação foi desenvolvida tendo em mente seu deployment em cluster. Caso este elemento não esteja presente, o servidor de aplicações não irá se preocupar com o fail-over da aplicação, e ela efetivamente estará funcionando apenas com balanceamento de carga, sem alta disponibilidade.

### Erros de aplicação a evitar

O suporte a clustering web com HA (High Availability ou Alta Disponibilidade) do Java EE é transparente, desde que o programador não cometa alguns erros básicos:

* Todos os objetos armazenados na sessão HTTP devem ser serializáveis;
* A aplicação não deve depender de nenhum estado exceto o que está no próprio cliente (como campos escondidos de formulários HTML) e na sessão HTTP;
* Uma requisição HTTP deve corresponder a uma única transação no banco de dados, e uma transação no banco de dados deve ocorrer dentro de uma única requisição HTTP;
* Não utilize singletons, atributos de classe ou atributos de contexto como “caches”; em vez disso, use bibliotecas de cache escritas especialmente para operar em clusters. Só é seguro usar “caches” não clusterizados se eles forem apenas para consulta.

## Arquitetura de Cluster HA do Tomcat

O Tomcat utiliza multicast IP para identificar os nós do cluster que estão ativos. Na verdade cada nó anuncia sua presença aos demais utilizando os multicasts, então o cluster é “plug-and-play”. Os nós não precisam ser configurados com a identidade dos demais nós, e acrescentar novos membros não exige um reinício total do cluster,como acontece com a maioria dos produtos de cluster.

As informações em si são transferidas por meio de conexões TCP regulares para os demais nós existentes. O Tomcat replica as sessões HTTP entre todos os nós do cluster, e por isso não depende de nenhum armazenamento compartilhado. Na veradade, o cluster HA do Tomcat é totalmente autônomo, não necessitando de nenhum outro middleware, exceto pelo balanceador.

Se um nó fica muito tempo sem anunciar sua presença, os demais nós o consideram como inativo e deixam de lhe enviar informações, e novos nós são aceitos a qualquer momento.

Havendo vários clusters independentes na mesma rede, cada cluster deve ser configurado com um endereço IP de multicast diferente, o que é feito dentro do elemento <Cluser> do web.xml.

Quando ocorrerem modificações nas sessões HTTP de um nó, estas modificações são transferidas por meio de conexões TCP regulares para os demais nós presentes.

Quando um novo nó entra no cluster (se torna ativo) ele localiza os demais nós e pede que cada um lhe envie as informações atuais de todas as sessões HTTP.

## Configuração da Replicação de Sessão

O arquivo de configuração padrão server.xml do Tomcat traz comendado o elemento **<Cluster>** que cuida do multicast IP entre os nós de um mesmo cluster.

Este elemento pode ser configurado vários sub-elementos especializados, mas eles normalmente não são modificados e o elemento simples na configuração padrão já permite um cluster inteiramente funcional. Basta descomenta-lo.

<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/>

A configuração default completa do cluster é apresentada na listagem 1.1 e caso você deseje customizar a configuração o primeiro passo é copiar o modelo completo para dentro do seu server.xml, em substituição à versão simplificada.

**Listagem 1.1** – configuração completa de cluster do Tomcat, com valores padrão

* <Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"

1. channelSendOptions="8">
2. <Manager className="org.apache.catalina.ha.session.DeltaManager"
3. expireSessionsOnShutdown="false"
4. notifyListenersOnReplication="true"/>
5. <Channel className="org.apache.catalina.tribes.group.GroupChannel">
6. <Membership className="org.apache.catalina.tribes.membership.McastService"
7. **bind="0.0.0.0"**
8. **address="228.0.0.4"**
9. **port="45564"**
10. frequency="500"
11. dropTime="3000"/>
12. <Receiver className="org.apache.catalina.tribes.transport.nio.NioReceiver"
13. **address="auto"**
14. **port="4000"**
15. **autoBind="100"**
16. selectorTimeout="5000"
17. maxThreads="6"/>
18. <Sender className="org.apache.catalina.tribes.transport.ReplicationTransmitter">
19. <Transport className="org.apache.catalina.tribes.transport.nio.PooledParallelSender"/>
20. </Sender>
21. <Interceptor className="org.apache.catalina.tribes.group.interceptors.TcpFailureDetector"/>
22. <Interceptor className="org.apache.catalina.tribes.group.interceptors.MessageDispatch15Interceptor"/>
23. </Channel>
24. <Valve className="org.apache.catalina.ha.tcp.ReplicationValve"
25. filter=""/>
26. <Valve className="org.apache.catalina.ha.session.JvmRouteBinderValve"/>
27. <Deployer className="org.apache.catalina.ha.deploy.FarmWarDeployer"
28. tempDir="/tmp/war-temp/"
29. deployDir="/tmp/war-deploy/"
30. watchDir="/tmp/war-listen/"
31. watchEnabled="false"/>
32. <ClusterListener className="org.apache.catalina.ha.session.JvmRouteSessionIDBinderListener"/>
33. <ClusterListener className="org.apache.catalina.ha.session.ClusterSessionListener"/>

</Cluster>

Não iremos nos preocupar em apresentar cada elemento e atributo da configureação de cluster, nos restringindo aqueles onde há maior possibilidade do administrador modificar.

### Vários clusters na mesma rede

Um cenário usual é que sejam formados vários clusters, digamos um para produção e outro para homologação (incluindo testes decarga), ou então vários clusters dedicados à aplicações diferentes.

Mas todos os Tomcats que se enxergarem na mesma rede local irão formar um único cluster, a não ser que os atributos port ou address do elemento <Mebmership> seja modificado.

O endereço é na verdade um endereço de multicast, não um endereço tradicional de classes A, B ou C. Ao contrário dos endereços tradicionais (chamados unicast), várias aplicações ou computadores podem assumir simultaneamente um mesmo endereço de multicast. Pacotes enviados para este endereço são recebidos por todos.

O endereço de multicast tem que ser vinculado a uma interface de rede, aí entra o atributo bind de <Membership>. O valor padrão usa todas as placas de rede do sistema, mas em produção você vai querer segregar o tráfego intra-cluster para uma placa de rede dedicada.

Em geral mudar a porta é suficiente para segregar os clusters, mas em alguns sistemas a pilha IP tem bugs e não diferencia corretamente portas de multicast diferentes no mesmo computador, obrigando a mudança do endereço também.

Já o endereço em <Receiver> é um endereço unicast tradicional, no qual o nó recebe mensagens de replicação. Então múltiplas instâncias do Tomcat poderiam entrar em conflito, se configuradas com o mesmo port. Entretanto, caso o Tomcat encontre a porta em uso, ele irá tentar a próxima porta até o limite definido por autoBind.

### Configuração de Multicast no Linux

Para que o cluster funcione, a rede local tem que estar permitindo tráfego em multicast. Garanta que seus switches não estão bloqueando este tráfego

O sistema operacional deve saber para onde enviar pacotes destinados a um dado endereço de multicast (e ter este suporte ativado) o que é feito como parte das configurações de roteamento estático. No Linux seria, para nossa sala de aula:

# route add -net 224.0.0.0 netmask 240.0.0.0 gw 127.0.0.1

Em um servidor real seria algo como:

# route add -net 224.0.0.0 netmask 240.0.0.0 dev eth1

Na versão 5 do Java da Sun, existe ainda um problema em relação ao IPv6, que é contornado pela propriedade de sistema java.net.preferIPv4Stack=true.

### Cluster x Manager

O Manager não tem conhecimento do cluster, portanto ele tem que ser ativados e acessados de forma independente em cada nó. O deployment de aplicações e ouras configurações, como realms e DataSources, também são feitos em cada nó de forma independente.

Por isso recomenda-se não eliminar o conector HTTP simples da configuração do Tomcat, mesmo que todos os usuários passem pelo balanceador. Deixe o conector HTTP para conexões adminsitrativas e para troubleshooting de aplicações.

## Exercícios

Laboratório 1.1. Cluster com HA

(Prática Dirigida)

**Objetivo:** Configurar o cluster para alta disponibilidade e validar que o “contador” é preservado em caso de falha em um nó

Este exercício usa o cluster que foi montado no capítulo anterior, assim como a aplicação de teste.

Finalize os servidores Tomcat, e edite em cada um o server.xml para inserir a configuração completa de cluster, porém restrita ao endereço IP de loopback, de modo que cada estação da sala de aula forme um cluster isolado com seus dois servidores Tomcat.

As modificações sobre a configuração padrão são:

* <Membership className="org.apache.catalina.tribes.membership.McastService"

1. **bind="127.0.0.1"**
2. address="228.0.0.4"
3. port="45564"
4. frequency="500"
5. dropTime="3000"/>
6. <Receiver className="org.apache.catalina.tribes.transport.nio.NioReceiver"
7. **address="127.0.0.1"**
8. port="4000"
9. autoBind="100"
10. selectorTimeout="5000"

maxThreads="6"/>

Note que a mesma modificação deve ser realizada nos dois nós. Para poupar digitação, o diretório de exemplo deste laboratório fornece a configuração de cluster para ser copiada e modificada no server.xml.

Como queremos usar apenas o loopback, vamos cair no bug do Java da Sun (lembre que o OpenJDK é o Java da Sun com licença GPL) então, em ambas as instâncias do Tomcat, deve ser editado o arquivo tomcat6.conf (na pasta /usr/share/<instancia>/conf ou então em /etc/<instancia>/conf).

Localize a linha:

* #JAVA\_OPTS="-Xminf0.1 -Xmaxf0.3"

E, abaixo dela, acrescente a linha:

* JAVA\_OPTS="-Djava.net.preferIPv4Stack=true"

Então reinicie os dois servidores Tomcat (não é necessário reiniciar o Apache) e faça novamente o teste com dois navegadores acessando via o Apache. O comportamento por enquanto deve ser o mesmo como no cluster de distribuição de carga.

Agora vem a parte interessante: finalize o primeiro Tomcat (no1), seja via o comando service tomcat\_no1 stop ou então identificando o processo java correspondente na lista de processos e então usando kill -9.

Em seguida, recarregue a página no navegador que estava usando este servidor. O identificador de sessão deve permanecer o mesmo, e o contador não deve ser reiniciado.

Você acabou de comprovar que seu cluster tem alta disponibilidade, sobrevivendo à quedas de um membro sem interferir com o usuário!

Se algo deu errado:

* Verifique com o comando route se existe uma rota para a rede 224.0.0.0/3 (classe D, IP multicast);
* Verifique nos logs dos dois Tomcat e do Apache por mensagens de erro, como sintaxe inválida nos arquivos de configuração;
* Derrube todos os servidores e verifique se as portas quatro TCP utilizadas por cada nó estão livres (deveriam estar, se você realizou os exercícios do capítulo anterior com sucesso);
* Páre todos os servidores, remova o conteúdo das pastas work e temp nos dois Tomcats, e reinicie ambos. Sempre que um servidor Tomcat morrer de forma inesperada, pode ser necessário fazer esta “limpeza”;
* Seja paciente, pois a simulação de um cluster em um mesmo computador às vezes pode parecer errática. O tempo de troca de contexto de um Tomcat para o outro, especialmente em computadores com pouca RAM e vários outros processos ativos, pode ser longo o suficiente para o balanceador ou um dos nós acreditar que o outro nó esteja fora do ar. Por isso, a página de status do mod\_jk pode indicar falha em nós ativos e demorar a reconsiderar o nó como presente no cluster.

Se acontecer dos dois navegadores serem atendidos pelo mesmo servidor, porém ambos os nós estão no ar e configurados corretamente, feche o navegador que está no servidor “errado” (para perder o cookie de sessão HTTP), aguarde alguns segundos e tente novamente. Conexões que migraram para fora de um nó falho não voltam automaticamente quando este nó retorna ao cluster

## Conclusão

Agora sim temos um cluster completo, ativo-ativo, com escalabilidade e alta disponibilidade. Para chegar neste resultado foi necessário juntar um balanceador de carga, que implementamos usando o mod\_jk, com os recursos de cluster para replicação de sessão HTTP do próprio Tomcat.

Questões de Revisão

* Toda aplicação deployada em um Tomcat custerizado será automaticamente ela também custerizada?

* Porque pode ser necessário modificar a configuração de <Membership> de um servidor Tomcat em cluster?

# Monitoração e Tuning

Este capítulo é uma uma introdução ao monitoramento e tunning do servidor Tomcat para obter maior performance e estabilidade

Tópicos:

* Modelo de performance para servidores de aplicação Java EE
* Tunning de threads e conexões
* Tunning da JVM
* Monitoração pelo Manager
* Monitoração JMX
* Dicas de troubleshooting

## Modelo de performance para aplicações Java EE

Não é possível prever a performance ou dimensionar um servidor de aplicações Java EE de forma isolada, ao contrário do que seria possível com um servidor de e-mail ou de arquivos.

A peformance e consumo de recursos de hardware do servidor de aplicações é dependente das aplicações hospedadas por ele. Erros de projeto ou codificação aparentemente inócuos podem ter efeitos graves quando temos centenas de usuários e milhares de requisições de rede.

Os principais elementos a serem observados no tuning do servidor formam conceitualmente um pipeline de processamento formado por:

1. Os threads de entrada, gerados pelos conectores;
2. O uso de processador e memória para a execução destes threads, monitorado a nível do sistema operacional e da JVM (heap);
3. As conexões para o banco de dados, gerenciadas pelos DataSources.

Em uma aplicação bem-escrita, se observará um efeito de”funil”: a quantidade de usuários ativos será bem maior do que a quantidade de threads em uso, porque os usuários gastam muito tempo lendo as páginas, digitando no teclado, e movendo o mouse para o próximo botão ou link.

De modo similar, a quantidade de conexões em uso para o banco de dados deverá ser menor do que a quantidade de threads ativas, pois uma aplicação irá primeiro criticar os dados de entrada, decidir o que fazer com eles, para aí sim acessar o banco de dados. E ela irá gastar um tempo depois de finalizado o acesso para gerar a página de resposta.

Este “efeito de funil” orienta o dimensionamento dos pools de threads dos conectores e pelo pool de conexões dos DataSources. Entretanto é impossível prever o quanto o funil será aberto ou apertado. Para isto é necessário monitorar o servidor e fazer testes de carga com aplicações reais.

O uso de pools de threads e conexões permite eficiência no uso de processador e memória, além de melhorar o tempo de resposta, pois não se perde tempo criando e destruindo threads ou conexões, que são operações caras para o SO. Além disso, os objetos Java correspondentes também são reutilizados, reduzindo a pressão sobre o coletor de lixo. Isto significa mais heap (RAM) e processador disponível para os objetos das aplicações.

## Monitorando o Tomcat via Manager

O Tomcat pode ser monitorado de duas formas:

1. Via a aplicação Manager;
2. Via JMX (Java Management Interface)

A página de status do Manager permite ver o uso de threads por cada conector, as requisições (URLs) sendo atendidas no momento, e o consumo de memória do heap da JVM.

Já a página de lista de aplicações permite visualizar quantos usuários (sessões) estão ativos em cada aplicação e ver os atributos em cada uma, o que fornece uma idéia crua do uso de memória por cada aplicação ou usuário.

### Tuning de conector x balanceador

Cada conector do Tomcat possui atributos de tuning como maxThreads e acceptCount, que determinam respectivamente o tamanho máximo do pool e a quantidade de requisições que podem ficar enfileiradas aguardando por um thread livre, antes que o conector passe a recusar novas conexões.

Caso sua configuração use um servidor de front-end seus parâmetros de pool de processos ou de threads devem ser iguais ou inferiores aos do conector no lado do Tomcat.

Por exemplo, MaxServers no Apache deve ser igual ou inferior a MaxProcessors no Conector JK do Tomcat, se for usado o mod\_jk.

Em um ambiente de cluster, deve-se prever ma folga para eventuais “desbalanceamentos”

Ex: três Tomcats, um Apache. Um único Tomcat falho, e 10 usuários para cada Tomcat:

MaxServers=3\*10, MaxProcessors=2.2\*10

MaxServers=30, MaxProcessors=22

### Dimensionamento de DataSources

Um <Resource> que define um DataSource pode receber atributos maxActive e maxIddle que indicam a quantidade máxima de conexões a serem abertas contra o banco de dados, e a quantidade mínima de conexões ociosas que serão mantidas abertas.

O dimensionamento e monitoração do pool de conexões a bancos de dados (DataSources) é um pouco mais complicado porque o Manager não exibe informações sobre o status atual dos pools (mas elas podem ser obtidas via JMX).

## Memória da JVM

A JVM não aloca memória livre do sistema operacional. Em vez disso, no início da JVM é estabelecido um limite máximo para o heap, onde são armazenadas instâncias de classes Java.

Dependendo da quantidade de usuários, aplicações, páginas e informações na sessão HTTP pode ser necessário aumentar o tamanho do heap, que é configurado com um tamanho inicial (mínimo) e um tamanho máximo.

A página de status do servidor (no Manager) informa a memória em uso e a máxima no heap

no manager. Mas a indicação de memória livre (Free memory) é em relação ao efetivamente alocado para o heap (Total memory), e não em relação ao máximo possível (Max memory).

A configuração é feita por opção de linha de comando da JVM, isto é, do comando java:

* -Xms<size> Tamanho inicial e mínimo do heap;
* -Xmx<size> Tamanho máximo do heap;
* -Xss<size>Tamanho da pilha (variáveis locais e argumentos de chamada de métodos).

O máximo padrão do heap é de 64Mb ou 256Mb, dependendo da versão e fornecedor. Parece pouco, mas lembre que é apenas o heap, outras partes da JVM ocupam centenas de Mb mas estas partes não crescem com a carga da aplicação, ao contrário do heap. Em um sistema operacional de 32-bits, onde um processo individual está limitado a 2Gb, o tamanho máximo do heap será algo entre 1,5 e 1,7Gb.

As informações de uso de memória física total devem ser obtidas pelo sistema operacional. Em Linux, o comando abaixo exibe a memória total (vsz) e em uso na RAM (rss) para os processos que rodam uma JVM.

$ ps axo pid,vsz,rss,cmd | grep java

### Passando opções para a JVM

Os scripts de início do Tomcat utilizam a variável de ambiente JAVA\_OPTS para indicar as opções a serem passadas para a JVM.

Esta variável pode ser passada diretamente para o script de inicialização do Tomcat instalado manualmente, por exemplo:

$ JAVA\_OPTS=-Xmx256M ./startup.sh

Ou então a variável pode ser definida dentro do próprio script.

Mas para a instalação JPackage deve ser editado o arquivo /etc/tomcat6/tomcat6.conf, que é um shell script restrito à definição de variáveis usadas na linha de comando de início do Tomcat.

O valor da variável JAVA\_OPS pode se tornar bastante longo por isso sugerimos o macete de se acrescentar valores, preservando o valor original, em vez de modificar a variável totalmente, por exemplo:

JAVA\_OPTS="$JAVA\_OPTS -Xmx512M"

## Tuning do Cluster

Os recursos de clustering to Tomcat fornecem uma série de parâmetros que podem ser ajustados. A maioria deles exigem conhecimento mais avançado do que o esperado neste curso, mas alguns deles são bem intuitivos.

Com uma grande quantidade de usuários simultâneos, pode ser necessário aumentar a quantidade de threads que recebem informações dos demais nós, especialmente se o cluster envolver mais de dois nós:

* <Receiver className="org.apache.catalina.cluster.tcp.ReplicationListener"

1. tcpListenAddress="auto"
2. tcpListenPort="4000"
3. tcpSelectorTimeout="100"

**tcpThreadCount="8"**/>

Por padrão, o envio de modificações nas sessões de um nó é assíncrono, o que gera um pequeno risco de perda das últimas modificações das sessões em caso de pane. Mudar para envio síncrono evita este risco às custas de maior consumo de processador, memória e rede:

* <Sender className="org.apache.catalina.cluster.tcp.ReplicationTransmitter"

1. **replicationMode="synchronous"**

ackTimeout="15000"/>

## Monitoração via JMX

Especificação da plataforma Java que permite a qualquer aplicação fornecer “objetos gerenciados”, os MBeans. O JMX é conceitualmente similar ao SNMP para monitoração de dispositivos de rede, e os MBeans são conceitualmente similares aos MIBs do SNMP.

É possível relacionar e buscar MBeans, consultar seus atributos (propriedades) e até de executar ações sobre eles (operações ou métodos). O suporte a JMX É obrigatório em versões mais recentes do Java EE, que também define conjuntos de MBeans padronizados para os servidores de aplicações.

O nome de um MBean segue a forma:

domínio:nome1=valor1,nome2=valor2,..

Onde podem haver vários partes nome=valor. Maiúsculas e minúsculas fazem diferença!

Alguns MBeans são fornecidos pela própria JVM (versão 1.5 em diante), outros pelo Tomcat, que fornece MBeans que representam Aplicações Web, Servlets, DataSources, Realms e Conectores, entre outros.

O mercado oferece vários consoles para gerenciamento e monitoração JMX, similar aos consoles SNMP. Entre eles podemos citar o MC4J e o Hyperic.

### Usando o JConsole com o Tomcat

Para permitir a monitoração por um console JMX, o agente embutido na JVM deve ser ativado na inicialização do Tomcat, pela definição de propriedades de sistema na variável de ambiente JAVA\_OPTS que foi apresentada no tópico sobre memória da JVM:

JAVA\_OPTS="$JAVA\_OPTS -Dcom.sun.management.jmxremote"

O Java da Sun fornece um console JMX gráfico simples, o JConsole. Ele é capaz de se conectar diretamente a uma JVM, dado o seu número de processo:

$ ps ax | grep java

22587 pts/5 Rl :03/usr/lib/jvm/java-1.6.0/bin/java

...

org.apache.catalina.startup.Bootstrap start

$ jconsole 22587

O JConsole apersenta várias abas, contendo gráficos de utilização de memória, relação de threads da JVM e uma relação geral de MBeans. É fácil localizar informações úteis no JConsole, por exemplo:

* O Tomcat nomeia claramente os threads dos conectores, que são as responsáveis por atender requisições
* Todos os componentes do Tomcat são expostos como MBeans, então é possível localizar um DataSource e verificar a quantidade de conexões ativas (em uso)

### Acesso remoto JMX

É possível usar o JConsole ou outro console JMX remotamente, basta definir mais propriedades de sistema para a JVM:

JAVA\_OPTS="-Dcom.sun.management.jmxremote

-Dcom.sun.management.jmxremote.port=5000

-Dcom.sun.management.jmxremote.ssl=false

-Dcom.sun.management.jmxremote.authenticate=false"

Consulte os manuais do JDK da Sun para saber como configurar autenticação no acesso remoto JMX.

O JConsole pode então receber como argumento o endereço IP e porta pra a aconexão à JVM do Tomcat:

$ jconsole 127.0.0.1:5000

### JMX Proxy Servlet do Manager

O Manager do Tomcat oferece também um Servlet que atua como uma interface JMX genérica via HTTP. Entretanto ele não foi feito pensando no uso interativo, e sim via scripts, como plug-ins de monitoração do MRTG e Zabbix.

Quando usaro JMXProxy do manager, lembre sempre de codificar os caracteres especiais dentro do nome lógico de um MBean (como “:” e “=”) dentro das convenções de URLs para HTTP GET.

Para localizar MBeans e listar suas propriedades, utilize o parâmetro qry junto com uma máscara ou o nome completo de um MBean. Por exemplo:

* Um Realm, definido dentro de um Contexto:  
  Catalina:type=Realm,path=/exemplo4.1,host=localhost
* Um Servlet:

Catalina:j2eeType=Servlet,name=contatos,WebModule=//localhost/exemplo3.3,J2EEApplication=none,J2EEServer=none

* O Pool de Threads de um Conector:  
  Catalina:type=ThreadPool,name=http-8080
* Um dos Threads gerenciados por um dos Conectores:  
  Catalina:type=RequestProcessor,worker=http-8080,name=HttpRequest1
* Um DataSource:  
  Catalina:type=DataSource,path=/exemplo4.1,host=localhost,class=javax.sql.DataSource,name="jdbc/Contatos"

E eis algumas URLs para o JMXProxy:

* Listar todos os MBeans disponíveis no Tomcat:  
  http://127.0.0.1:8080/manager/jmxproxy?qry=\*%3A\*

(O “%3A” é o sinal de “:” depois do nome do domínio, que será sempre “Catalina”)

* Relaciona todos os Servlets dentro do Tomcat, e fornece contadores de chamadas e tempo de processamento:  
  http://127.0.0.1:8080/manager/jmxproxy?qry=\*%3Aj2eeType=Servlet%2c\*

(O “%2c” é uma vírgula)

* OK - Number of results: 33

1. Name: Catalina:j2eeType=Servlet,name=contatos,WebModule=//localhost/exemplo3.3,J2EEApplication=none,J2EEServer=nonemodelerType: org.apache.tomcat.util.modeler.BaseModelMBean
2. objectName: Catalina:j2eeType=Servlet,name=contatos,WebModule=//localhost/exemplo3.3,J2EEApplication=none,J2EEServer=none
3. minTime: 9223372036854775807
4. statisticsProvider: false
5. loadTime: 0
6. classLoadTime: 0
7. processingTime: 0
8. requestCount: 0
9. errorCount: 0
10. stateManageable: false
11. eventProvider: false
12. engineName: Catalina

maxTime: 0

* Quantidade de conexões ativas para bancos de dados:  
  http://127.0.0.1:8080/manager/jmxproxy?qry=\*%3Atype=DataSource%2c\*
* Name: Catalina:type=DataSource,path=/exemplo3.3,host=localhost,class=javax.sql.DataSource,name="jdbc/Contatos"

1. ...
2. url: jdbc:hsqldb:file:/home/Miguel/bd/contatos;shutdown=true
3. ...
4. numActive: 0
5. maxActive: 1

numIdle: 0

* Requisições em atendimento:  
  http://127.0.0.1:8080/manager/jmxproxy?qry=\*%3Atype=RequestProcessor%2c\*
* Name: Catalina:type=RequestProcessor,worker=http-8080,name=HttpRequest1

1. ...
2. requestBytesSent: 0
3. bytesReceived: 0
4. requestProcessingTime: 2
5. ...
6. currentQueryString: qry=\*%3Atype=RequestProcessor%2c\*
7. bytesSent: 160829
8. currentUri: /manager/jmxproxy
9. ...
10. stage: 3
11. method: GET

requestBytesReceived: 0

* Threads ocupados e ociosos:  
  http://127.0.0.1:8080/manager/jmxproxy?qry=\*%3Atype=ThreadPool%2c\*
* Name: Catalina:type=ThreadPool,name=http-8080

1. ...
2. maxThreads: 40
3. currentThreadsBusy: 1
4. ...
5. acceptorThreadCount: 1
6. ...

currentThreadCount: 1

## Troubleshooting

Esta seção finaliza a apostila apresentando algumas dicas sobre problemas comuns com o Tomcat (ou melhor, com as aplicações dentro dele);

### OutOfMemory naPermGen

A JVM reserva uma área de memória separada do heap, chamada de PermGen (“Geração Permanente”) onde são guardados os bytecodes e resultado da compilação JIT das classes Java.

Esta região de memória não é reciclada na JVM da Sun, ou seja: versões antigas das classes continuam ocupando memória mesmo depois de serem substituídas por versões mais novas. Com uma quantidade suficiente de deploys, o PermGen fica cheio e a JVM gera uma exceção de OutOfMemory.

O problema pode ser aliviado com a opção -XX:MaxPermSize da JVM. Mas não há como evitar ter que reiniciar o Tomcat para liberar o PermGen.

### datasource leaks

Um dos problemas mais sérios de aplicações são os “vazamentos” (leaks) de conexões ao banco de dados. Eles acontecem por erro de programação, onde conexões são abertas (retiradas do pool) mas nunca fechadas (devolvidas ao pool).

Muitas vezes os leaks surgem do tratamento incorreto de erros pelo programador, que se esquece do bloco finally.

Então, após algum tempo não haverão mais conexões disponíveis para uso por novas requisições. Mas o Tomcat ajuda a identificar o culpado: Dentro do elemento <Resource> que configura um DataSource, podem ser definidos os atributos:

* removeAbandoned="true"

Instrui o Tomcat a forçar a liberação de conexões que parecam ter sido “abandonadas” (portanto, “vazadas”);

* removeAbandonedTimeout="300"

Após quanto tempo sem uso uma conexão é considerada “abandonada”;

* logAbandoned="true"

Registra no log do Tomcat as conexões que foram considerada abandonadas.

### Servidor Congelado

Em casos extremos, como falta de memória ou saturação de processador, o Tomcat pode demorar tanto a responder requisições que parece estar travado. Nestes casos, usar o Manager (incluindo o JMX Proxy) será inviável, pois estas requisições também não serão respondidas.

Utilitários do Sistema Operacional, por exemplo ps, top, vmstat ou sar ajudam a identificar gargalhos, assim como logs de acesso do próprio Tomcat. Mas, exatamente o que o Tomcat estava fazendo quando “congelou”, ou melhor, que aplicaões estavam sendo executadas, e fazendo o que?

Como recurso extremo de depuração, a JVM é capaz de gerar uma relação completa do estado de todos os seus threads, o chamado Thread Dump. Ele contém um stack traces para cada thread da JVM, de modo que se sabe exatamente que método estava sendo executado por cada uma.

O thread dump também indica se o thread está executando, pronto ou aguarando por E/S; ou se está bloqueado em um semáforo

Um thread dump pode ser gerado via JMX (se a JVM ainda consegir responder ao console JMX) ou então na saída padrão pelo sinal SIGQUIT, de modo que o thread dump no Tomcat será salvo em catalina.out:

# ps ax | grep tomcat

13978 pts/5 Sl 0:06 /usr/lib/jvm/java-1.6.0/bin/java -Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager -Djava.util.logging.config.file=/home/Miguel/apache-tomcat-6.0.13/conf/logging.properties

...

# kill -QUIT 13978

# vi ../logs/catalina.out

Seguem alguns exemplos de thread dumps, para auxiliar o aluno na interpretação dos mesmos:

* Este thread está ocioso, aguardando que uma requisição HTTP seja repassada pelo conector
* "http-8080-1" daemon prio=1 tid=0x09d33980 nid=0x36a4 in Object.wait() [0xb19d6000..0xb19d6fb0]

1. at java.lang.Object.wait(Native Method)
2. - waiting on <0x892f03a0> (a org.apache.tomcat.util.net.JIoEndpoint$Worker)
3. at java.lang.Object.wait(Object.java:474)
4. at org.apache.tomcat.util.net.JIoEndpoint$Worker.await(JIoEndpoint.java:416)
5. - locked <0x892f03a0> (a org.apache.tomcat.util.net.JIoEndpoint$Worker)
6. at org.apache.tomcat.util.net.JIoEndpoint$Worker.run(JIoEndpoint.java:442)

at java.lang.Thread.run(Thread.java:595)

* Clicado “Full Server Status” no Manager
* "http-8080-1" daemon prio=1 tid=0x09d33980 nid=0x36a4 runnable [0xb19d6000..0xb19d6fb0]

1. at javax.management.ObjectName.getKeyPropertyList(ObjectName.java:1480)
2. ...
3. at com.sun.jmx.interceptor.DefaultMBeanServerInterceptor.queryMBeans(DefaultMBeanServerInterceptor.java:472)
4. at org.apache.catalina.manager.StatusTransformer.writeContext(StatusTransformer.java:679)
5. at org.apache.catalina.manager.StatusTransformer.writeDetailedState(StatusTransformer.java:594)

at org.apache.catalina.manager.StatusManagerServlet.doGet(StatusManagerServlet.java:299)

## Exercícios

Laboratório 1.1. Memória com o Manager e ps

(Prática Dirigida)

**Objetivo:** Verificar o efeito de configurações sobre o tamanho do heap da JVM

Se sua estação tem menos de 1,5Gb de RAM, sugerimos que encerre os Tomcats em cluster e volte ao Tomcat default da distribuição rodando ele sozinho.

Utilizando as instruções fornecidas neste capítulo, verifique qual o tamanho do heap configurado para o Tomcat, e compare com a memória total ocupada pelo processo.

Em seguida, configure a JVM para alocar mais (ou menos) memória e veja a mudança no Manager.

Laboratório 1.2. Monitoração JMX via JConsole

(Laboratório)

**Objetivo:** Obter informações de acesso ao banco de dados via JMX

Utilizando as instruções fornecidas neste capítulo, ative o acesso JMX local e utilize o JConsole sobre um servidor Tomcat que tenha um DataSource configurado.

Localize o MBean do DataSource e identifique a quantidade de conexões abertas e em uso ou ociosas.

Laboratório 1.3. Monitoração JMX via Manager

(Desafio)

**Objetivo:** Obter informações de acesso ao banco de dados via JMXProxy

Acompanhar o uso do banco de dados via JConsole não é muito prático, porque ele não é capaz de gerar gráficos ou tabelas sobre um MBean qualquer.

Mas, com um pouco de criatividade e programação em shell, é possível usar o Manager para obter uma monitoração continuada das conexões em um DataSource.

## Conclusão

Os concetos de monitoração e tuning encerram nosso curso de Administração do Tomcat. Ele é um software poderoso e sofisticado, com várias possibilidades interessantes que o tempo não permitiu que fossem abordadas neste curso.

Portanto não pare por qui, consulte os manuais do Tomcat em http://tomcat.apache.org, veja os artigos disponíveis na Internet e experimente com o servidor!

Questões de Revisão

* A monitoração via JXM está ativada por padrão no Tomcat?

* O thread que gera a própria página de status é exibido na monitoração do Manager?

1. A terceria edição, o Java ME, é focada em PDAs e Celulares [↑](#footnote-ref-1)
2. No momento em que o IcedTea foi iniciado, o GCJ estava implementando 95% do Java SE 1.4 e 92% do Java SE 5. [↑](#footnote-ref-2)
3. Curiosamente, o que se chama hoje de “plataforma Intel de 64-bits” ou EMT64 é na verdade a plataforma de 64 bits da AMD. A plataforma de 64-bits criada pela Intel, o Itanium, não teve sucesso no mercado, e os atuais Core Due, Xeon e etc da Intel são clones da arquitetura do Athlon da AMD. [↑](#footnote-ref-3)
4. Na verdade o instalador do Sun JDK 1.6.0 faz esta configuração, mas não o faz de acordo com os padrões do LSB. [↑](#footnote-ref-4)
5. Embora muitos pensem que Tomcat é o “Tom Cat”, ou seja, um gato de estimação, na verdade ele é um felino selvagem de porte médio que vive na Indonésia, arquipélago que inclui a ilha de Java. [↑](#footnote-ref-5)
6. Não confundir o termo “container” neste contexto com “container web” do Java EE. [↑](#footnote-ref-6)
7. O JAAS (Java Authentication and Authorization Services) define um padrão para a construção de LoginModules, permitindo a produtos de autenticação e controle de acesso fornecerem “drivers” padronizados para qualquer servidor de aplicação Java EE. [↑](#footnote-ref-7)
8. Às vezes ferramentas Windows criam problemas por considerar que maiusculas e minísculas não fazem diferença, gerando assim arquivos como Web-inf/Web.xml e o pacote WAR é considerado inválido pelo container web. [↑](#footnote-ref-8)
9. O 7-Zip (www.7-zip.org) é uma aplicação gráfica e livre para Windows, em nossa humilde opinião de longe o melhor da sua categoria, mesmo se comparado com compactadores proprietários para esta plataforma. [↑](#footnote-ref-9)
10. Não confunda o uso do termo “pacote” quando falamos de JAR ou de WAR com o uso do mesmo tempo quando se refere à estrutura de nomes de classes Java, isto é, as declarações **package** e **import** nos fontes Java. [↑](#footnote-ref-10)
11. Aparentemente esta limpesa não é mais necessária nas atualizações mais recentes do Tomcat 6.0, mesmo assim o tópico está na apostila porque seu ambiente de produção pode estar utilizando versões anteriores, como a 5.5. [↑](#footnote-ref-11)
12. A forma correta de uma aplicação acessar o conteúdo de arquivos dentro do seu próprio pacote é usar o método **getResourceAsStream()** da classe **ServletContext**. [↑](#footnote-ref-12)
13. Não confunda as entradas de ambiente JNDI com as variáveis de ambiente do sistema operacional! [↑](#footnote-ref-13)
14. Veremos em capítulos futuros que é possível definir pastas *webapps* diferentes para cada host, e assim criar áreas dentro de um mesmo servidor Tomcat onde usuários diferentes tenham ou não permissão de fazer auto-deploy. [↑](#footnote-ref-14)
15. O utilitário **configure**, padrão hoje na maioria dos softwares livres escritos em C, procura contornar estas limitações gerando automaticamente os Makefiles para a compilação dos programas. No mundo Java não há necessidade de nada similar. [↑](#footnote-ref-15)
16. Entretanto, provedores de hospedagem geralmente habilitam o security manager e isto pode complicar a instalação de bibliotecas e aplicações no ambiente compartilhado. [↑](#footnote-ref-16)
17. Na verdade, o uso de hash só dificulta o processo, obrigando um hacker ou um administrador mal-intencionado a usar dicionários ou força-bruta para quebrar as senhas. [↑](#footnote-ref-17)
18. No Java EE, o JNDI é uma API genérica para serviços de diretório. O servidor de aplicações mantém um serviço de diretórios interno, para localizar DataSources e outros recursos, mas também pode usar JNDI para acessar diretórios LDAP, DNS, NIS e outros. [↑](#footnote-ref-18)
19. O schema do LDAP é análogo a de um banco de dados relacional, e define a estrutura dos objetos armazenados no diretório. [↑](#footnote-ref-19)
20. Segurança em profundidade é a idéia de se implementar a mesma política de segurança utilizando vários mecanismos independentes e redundantes entre si, de modo que um hacker não consiga acesso ao encontrar uma forma de contornar apenas um dos mecanismos. [↑](#footnote-ref-20)
21. URA (Unidade de Resposta Audível) é o tipo de aplicação que “fala” menus para atendimento telefônico automatizado. [↑](#footnote-ref-21)
22. Já usuários Windows podem contar em conseguir um binaŕio pronto no site do oficial do Tomcat, pelo menos para 32-bits. [↑](#footnote-ref-22)