



# IT SOLUTIONSS KNOWLEDEGE EDUCATION

A **IT Solutions** é uma empresa especializada em capacitar tanto indivíduos quanto empresas por meio de cursos de capacitação técnica em diversas áreas da Tecnologia da Informação (TI). Nossa missão é fornecer o conhecimento e as habilidades necessárias para que os profissionais da TI possam se destacar em um mercado em constante evolução.

Nossos cursos abrangem uma ampla gama de tópicos, desde programação e desenvolvimento de software até administração de redes, segurança da informação, análise de dados e muito mais. Com instrutores experientes e conteúdo atualizado, garantimos que nossos alunos estejam preparados para enfrentar os desafios tecnológicos do mundo atual.

Para as empresas, oferecemos soluções personalizadas de treinamento que atendem às necessidades específicas de suas equipes. Acreditamos que investir na capacitação de funcionários é essencial para impulsionar a inovação e a eficiência nos negócios.

Na IT Solutions, estamos comprometidos em fornecer educação de qualidade que transforma vidas e impulsiona o sucesso empresarial. Se você está buscando aprimorar suas habilidades em TI ou capacitar sua equipe, estamos aqui para ajudar você a alcançar seus objetivos. Junte-se a nós e embarque na jornada de aprendizado tecnológico que pode abrir portas para um futuro brilhante na indústria de TI.

## CONTATOS:

- Website: <https://itsolutionss.com.br/> | E-mail: [comercial@itsolutionss.com.br](mailto:comercial@itsolutionss.com.br)
- WhatsApp: +55 11 98953-8598 | Tel: 11 3171-2227



# Agenda

## CURSO APACHE TOMCAT ADMINISTRATION

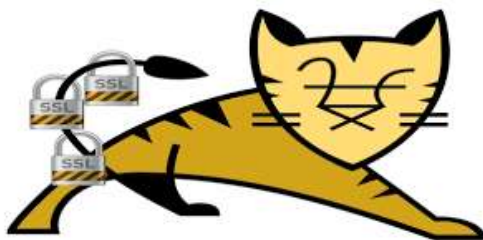
- ☐ Carga horaria: 24h
- ☐ De segunda-feira a quinta-feira das 19h às 23h

## FOMATO DAS AULAS:

- ☐ 100% on-line ao vivo transmitido via Microsoft
- ☐ Na presença de um instrutor/consultor
- ☐ Exercícios práticos
- ☐ Lab para fixar seu conhecimento em sala de aula
- ☐ Se precisar de ajuda compartilhe sua tela com o instrutor

## INTERVALOS DAS AULAS:

- ☐ 15 minutos para uma pausa, um café,
- ☐ O Instrutor vai informar em sala de aula,



# Agenda Conteúdo Programático

## INTRODUCING TOMCAT

1. Apache and Tomcat
2. Application Servers and Web Containers
3. Tomcat Component Tour
4. Versions History and Capabilities

## JAVA ENTERPRISE EDITION ARCHITECTURE

1. Java Enterprise Edition (JEE) Applications
2. Servlets and JSP
3. Tomcat the Web Container

## INSTALLING TOMCAT

1. Installation Options
2. Setting up Java
3. Installing the Tomcat Web Container
4. Validating a Successful Installation

## CONFIGURATION ESSENTIALS

1. The Tomcat Directory Structure
2. Understanding the Configuration Files
3. The Component Architecture
4. JVM Configuration

## LOGGING AND MONITORING

1. Understanding Log Files
2. Troubleshooting
3. Load Testing
4. Interpreting Results
5. Monitoring with Tomcat Manager
6. Threads and Memory

## DEPLOYING APPLICATIONS

1. Deploying WAR Files
2. Hot Deployment
3. Deploying Unpacked Files
4. Deploying with Tomcat Manager

## DEFINING DATASOURCES

1. Configuring a JDBC Datasource
2. Using JNDI Resources
3. Connection Pooling

## WORKING WITH WEB SERVERS

1. Tomcat and the Apache Web Server
2. Advantages and Disadvantages of Web Servers
3. Configuring Apache with Tomcat
4. Virtual Hosting

## CLUSTERING

1. Clustering Benefits
2. Setting up Clustering
3. Load Balancing and Failover

## SECURITY CONSIDERATIONS

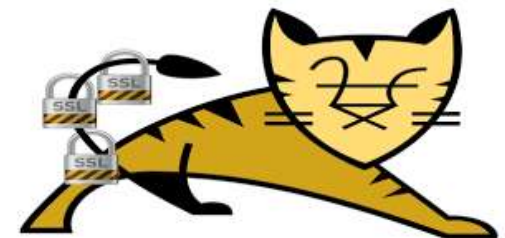
1. File System Security
2. Realms, Roles and Users
3. Java Security Manager
4. SSL Configuration

## HARDENING TOMCAT INSTALATIONS

1. Restricting Interfaces
2. Operating System Privileges
3. Handling Sessions
4. Securing Default Settings

## CUSTOM TOMCAT BUILDS

1. Support Libraries
2. Setting up Apache Ant
3. Generating Builds



# Instrutor: Miguel Vilaça

Especialista consultor ativo no mercado atuando diretamente com:

- ☐ Java,
- ☐ Tomcat,
- ☐ APIs,
- ☐ Microservices,
- ☐ Togaf,
- ☐ Arquitetura,
- ☐ Kafka,
- ☐ Integration,
- ☐ Docker,
- ☐ Kubernetes,
- ☐ IBM
- ☐ Oracle,
- ☐ Salesforce



<https://itsolutionss.com.br/>





# 0.1. Objetivo

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





## 0.2. 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 infraestrutura de produção baseada em servidores de aplicação Java EE.





## 0.3. Pré-requisitos

- ▶ Leitura básica em Inglês Técnico;
- ▶ Conhecimentos básicos de HTML e HTTP (navegadores e servidores Web);
- ▶ Conhecimentos básicos de TCP/IP.
- ▶ Desejáveis, embora não indispensáveis:
  - ▶ Conhecimentos básicos de Linguagem de programação Java
  - ▶ Compilação de programas na linha de comando utilizando o JDK;
  - ▶ Acesso a bancos de dados utilizando JDBC;
  - ▶ Construção de servlets e páginas JSP.





## 0.5. Laboratórios

- ▶ 90% ou mais deste curso poderia ser realizado em Windows, linux ou Mac. O Tomcat e os programas-exemplo deste curso são 100% Java.
- ▶ O curso apresenta recomendações e otimizações específicas para Linux porque este é o ambiente de produção preferencial para o Tomcat.







## 0.6. Perfil do Administrador de Servidores de Aplicação

- ▶ Mantém o ambiente de produção para aplicações
- ▶ A performance e estabilidade deste ambiente depende da qualidade e especificidades destas aplicações
- ▶ Ao contrário de um proxy web, servidor de e-mail ou arquivos, que depende apenas do código do próprio servidor
- ▶ Então o ASA necessita conhecimentos tanto de infraestrutura quanto de desenvolvimento, pois ele está na interseção entre estes dois universos





# **Servidores de Aplicações Java EE usando Tomcat**

## **Capítulo 1 Conceitos do Java EE**



# Objetivo

▶ 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





# 1.1. Java SE x Java EE

- ▶ **Java SE** ou **JSE** – Java Standard Edition  
(antigo J2SE)

- ▶ Fornece a JVM (máquina virtual Java) e as APIs essenciais para coleções, E/S, reflexão, serialização, XML, interfaces gráficas e redes

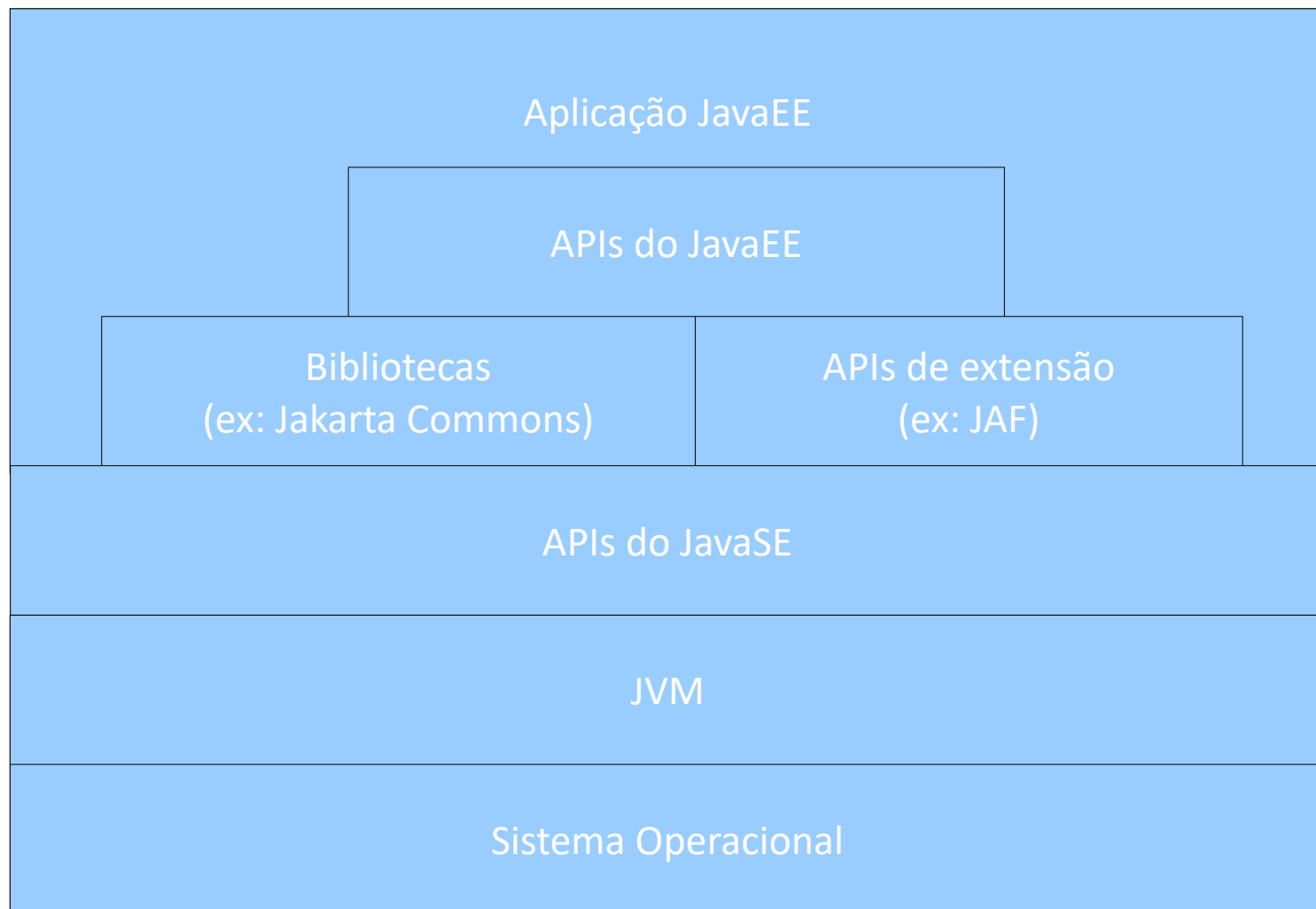
- ▶ **Java EE** ou **JEE** – Java Enterprise Edition  
(antigo J2EE)

- ▶ Tecnologias e APIs da plataforma Java para computação baseada em servidor
- ▶ Conectividade com bancos de dados, serviços de diretórios, correio eletrônico e outros serviços
- ▶ São previstos servidores especializados para aplicações Java EE





# Aplicações Java EE





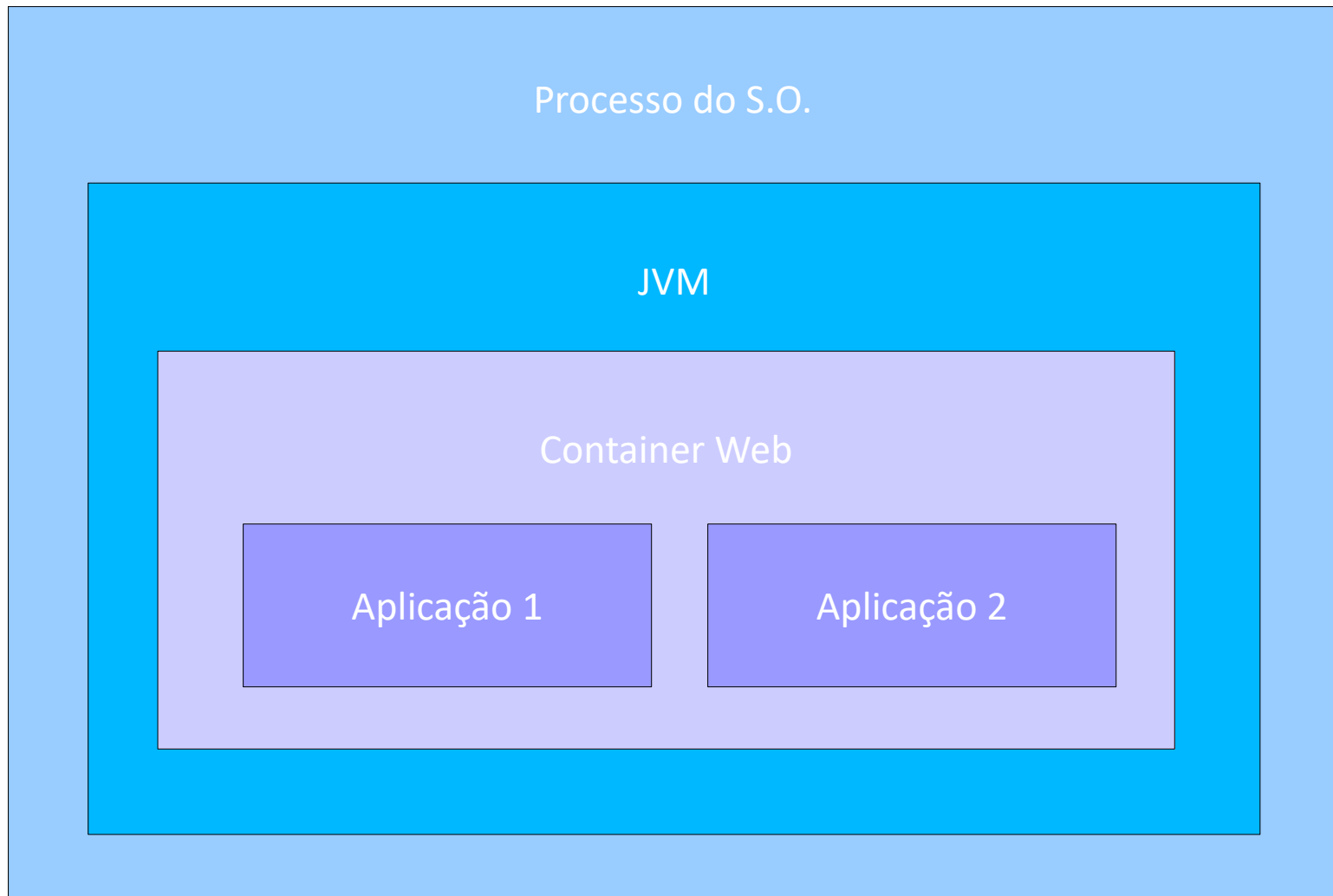
## 1.1.1. Containers Java EE

- ▶ Um servidor de aplicações JavaEE 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





# Aplicação Java EE x Container x JVM x SO





## 1.1.2. Padrões Web do Java EE

- ▶ Aplicações Web do Java EE são formadas por diversos componentes, 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







# Outras APIs do Java EE

- ▶ **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





## 1.2. Frameworks Java EE

- ▶ Uma série de frameworks se tornaram populares no desenvolvimento de aplicações Java EE, mesmo sem ser formalmente parte do padrão, entre eles:
  - ▶ **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 **JavaServer Faces** são outros frameworks populares para o Java EE





## 1.3. Java x Linux

- ▶ Embora o Java da Sun seja um download gratuito, sua licença de uso impede sua inclusão na maioria das distribuições do Linux
- ▶ Por isso que até 2005 era difícil encontrar aplicações Java como parte de distribuições do Linux
- ▶ Mas então projetos 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 Linux e Debian
- ▶ Entre elas: GCJ, Kaffe, CacaoJVM, Jikes





# JDL e OpenJDK

- ▶ A Sun ofereceu uma licença alternativa para o Java, a **JDL** (*Java Distribution License*) que permitiu a inclusão do Java da Sun no Debian e derivados (como Ubuntu)
- ▶ Mas a JDL não foi aceita por outras distribuições como Fedora e Red Hat
- ▶ Em 2007 foi iniciado o **OpenJDK**, a liberação do Java da Sun sob a licença GPL+Classpath Exception
- ▶ Entretanto a Sun não era proprietária de todos os componentes do seu JDK, que depende de alguns componentes binários, não-livres





# OpenJDK e IcedTea

- ▶ A dependência em relação a componentes fechados fez com que alguns desenvolvedores optassem por usar componentes do GNU Classpath para complementar o OpenJDK
- ▶ O resultado é o **IcedTea**, um Java inteiramente livre baseado no OpenJDK





# Certificação Java SE

- ▶ Apesar do progresso de clones do Java baseados no GNU Classpath e do IcedTea, os únicos “Javas” certificados pelos padrões do **JCP** (*Java Community Process*) são os produtos fechados 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.
- ▶ Não há garantia contratual de que aplicações Java do mercado funcionem corretamente com Javas não-certificados





## 1.4. Tomcat x Linux

- ▶ Várias distribuições do Linux hoje incorporam o Tomcat e outros softwares Java populares, como **Ant**, **Struts** e **Eclipse**
- ▶ Entretanto a instalação padrão da maioria delas traz um Java baseado no GNU Classpath, em geral o **GCJ**, e não um Java certificado
- ▶ Este ambiente é suficiente para rodar muitas aplicações, mas empresas em geral preferem utilizar um ambiente certificado
- ▶ Então poderá ser necessário instalar o Java da Sun (ou da IBM ou BEA)
- ▶ Com a adoção do OpenJDK / IcedTea, não haverá mais necessidade de instalar um Java proprietário





## 1.5. 0 Projeto JPackage

- ▶ Os downloads oficiais dos fornecedores de Java certificados não são aderentes aos padrões do Linux, como o **LSB** (*Linux Standards Base*)
- ▶ Já os downloads dos mesmos produtos, inclusos nas distribuições RHEL e SuSE, são aderentes, por terem sido re-empacotados definidos pelo Projeto **JPackage**
- ▶ Usuários de distribuições livres / gratuitas podem seguir as instruções em *jpackage.org* para re-empacotar o Java da Sun e assim ter um ambiente Java certificado e compatível com suas distribuições







# Vantagens do JPackage

- ▶ Arquivos de configuração, log, bibliotecas e binários são instalados nos diretórios padrão do Linux, como */etc*, */var/log*, */usr/lib* e */usr/bin*
- ▶ Não existem múltiplas cópias das mesmas bibliotecas, como Jakarta-Commons, simplificando a instalação de atualizações de segurança
- ▶ Poupa espaço em disco e RAM
- ▶ 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





## 1.5.1. Convenções do JPackage

- ▶ Arquivos JAR em */usr/share/java*
- ▶ Symlinks para nomes de JARs com números de versão
- ▶ Executáveis (java, javac, keytool, etc) gerenciados via alternatives
- ▶ JVMs em */usr/lib/jvm*
- ▶ JavaDocs em */usr/share/javadoc*
- ▶ */usr/share* contém subdiretórios emulando a estrutura original das aplicações, contendo links simbólicos para */etc*, */var/log*, */usr/share/java* e etc





## 1.6. Instalação do java via JPackage

- ▶ Para o OpenJDK em Fedora
  - ▶ `# yum -y install java-X.X.X-openjdk java-X.X.X-openjdk-devel`
- ▶ Para o Sun JDK
  - ▶ Reconstruir os pacotes manualmente
  - ▶ Copiar de alguém que já o fez
  - ▶ Canal Supplemental do RHEL
- ▶ Usar o **alternatives** para assegurar que a versão correta está como default





## 1.6.1. Java via JPackage x Aplicação Zipada

- ▶ Em geral é suficiente definir o JAVA\_HOME apontando para a instalação via Jpackage
- ▶ Para usar o Java padrão, configurado pelo alternatives:
  - ▶ `export JAVA_HOME=/usr/lib/jvm/java`
- ▶ Se precisar usar uma versão ou fornecedor específico:
  - ▶ `export JAVA_HOME=/usr/lib/jvm/java-X.X.0-sun`





## 1.7. Instalação manual do Java

- ▶ Será necessário configurar uma série de variáveis do ambiente de acordo com a sua instalação do Java:
  - ▶ **JAVA\_HOME** para o diretório de instalação:  
`export JAVA_HOME=/usr/java/jdkX.X.XX`
  - ▶ **PATH** para incluir o diretório onde estão os comandos java e javac:  
`export PATH=$JAVA_HOME/bin:$PATH`
  - ▶ **CLASSPATH** para incluir pelo menos o diretório corrente:  
`export CLASSPATH=.:$CLASSPATH`
- ▶ Os instaladores do Java da Sun não configuram estas variáveis!





# Conclusão

- ▶ Lab 1.1. Instalação e teste do OpenJDK X.X.X
- ▶ Lab 1.2. Instalação e teste do Sun JDK X.X.X
- ▶ Questões de Revisão





# Servidores de Aplicações Java EE usando Tomcat

## Capítulo 2 Instalação do Tomcat





# Objetivo

Nesta aula 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

WAR ~~TEXT~~







## 2.1. Sobre o Tomcat

- ▶ É um servidor de aplicações JavaEE que fornece apenas o Container Web para execução de aplicações Web Java EE
- ▶ Fornece ainda serviços JNDI, JAAS e JMX, de modo que aplicações Web (sem uso de EJBs ou JMS) criadas originalmente para servidores de aplicações mais “parrudos” como o JBoss devem rodar sem modificações no Tomcat
- ▶ Apresenta recursos avançados, como suporte nativo a clustering (desde a versão 5.0)





## 2.1.1. 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
- ▶ Versões mais novas do Tomcat suportam versões mais antigas das especificações
- ▶ As séries 7.x, 8.x, 9.x, 10x ainda são suportadas em termos de correções de bugs
- ▶ O desenvolvimento hoje ocorre na série 11.x





# Tomcat x Java EE

- ▶ Versões suportadas:

▶ Tomcat	Servlets/JSP	Java EE	Java SE
▶ 3.3 1.2		2.2/1.1	1.2
▶ 4.1 1.3		2.3/1.2	1.3
▶ 5.5 5		2.4/2.0	1.4
▶ 6.0		2.5/2/1	5
▶ 7.0 (Servlet 3.0, Java EE 6) em desenvolvimento			





## 2.2. Instalação do Tomcat

- ▶ 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, de modo que basta um JRE
- ▶ Tomcat 6.0.x
  - ▶ JRE 1.5.0 ou superior





## 2.2.1. Instalação via JPackage

- ▶ Como root:
- ▶ `# yum -y install tomcat tomcat-admin-webapps tomcat-docs-webapp tomcat-javadoc tomcat-jsp-2.1-api tomcat-webapps`
- ▶ (Use os repositórios locais da sala de aula; não baixe da Internet, para não atrapalhar outras salas de aula!)





## 2.2.2. Iniciando como Serviço

- ▶ No Fedora:
  - ▶ `service tomcat start`
  - ▶ `service tomcat stop`
- ▶ Debian
  - ▶ `/etc/init.d/tomcat start`
  - ▶ `/etc/init.d/tomcat stop`





## 2.2.3. Instalação Manual do Tomcat

- ▶ Visite <http://tomcat.apache.org> e siga o link para download da versão desejada
- ▶ Baixe a distribuição **Core**, em formato **zip**
- ▶ A versão em formato “Windows executable” cria atalhos no menu iniciar e configura um o Tomcat para execução como serviço do Windows
- ▶ A criação dos atalhos e serviço também pode ser feito manualmente pela versão zip, que inclui ainda os scripts para execução em Linux e Unix





# Instalação em Linux

- ▶ Descompacte o zip do Tomcat no seu diretório home, preservando os subdiretórios contidos no zip:
  - ▶ \$ unzip apache-tomcat-X.X.XX.zip
- ▶ Entre na pasta *bin* e dê permissão de execução para todos os shell scripts
  - ▶ \$ cd apache-tomcat-\*/bin
  - ▶ \$ chmod a+x \*.sh







# Configurando o JAVA\_HOME 1/2

- ▶ O Tomcat espera que esteja definida a variável de ambiente **JAVA\_HOME**, indicando o diretório de instalação do Java
- ▶ Espera-se que um ambiente Java configurado para uso pela linha de comando tenha definido esta variável
- ▶ Mas o JPackage não a define, pois espera que os scripts de inicialização das aplicações utilizem os scripts do JPackage para determinar a localização do Java Default ou de uma versão específica
- ▶ Então é necessário configurar o JAVA\_HOME nos scripts de início do Tomcat





# Configurando o JAVA\_HOME 2/2

- ▶ Edite o arquivo *conf/catalina.sh*
- ▶ No início do arquivo, antes da linha que “limpa” o CLASSPATH, acrescente:
  - ▶ `export JAVA_HOME=/usr/lib/jvm/java-X.X.XX`
- ▶ Esta linha faz com que seja utilizado qualquer que seja o Java padrão.





## 2.4. Início e Término Manual

- ▶ Para iniciar:

- ▶ Entre na pasta *bin* do Tomcat  
\$ cd ~/apache-tomcat-\*/bin
- ▶ Execute o script startup  
\$ ./statup.sh

- ▶ Para terminar:

- ▶ Entre na pasta *bin* do Tomcat  
\$ cd ~/apache-tomcat-\*/bin
- ▶ Execute o script shutdown  
\$ ./shutdown.sh

- ▶ Após cada operação (início e término) confirme a presença do processo Java e verifique que as três portas abertas pelo Tomcat





# Startup e Shutdown

- ▶ Dentro da pasta *bin*, os scripts startup e shutdown (em versões *.sh* para Linux / Unix e *.bat* para Windows) são usados, respectivamente, para iniciar e encerrar o servidor
- ▶ Há ainda executáveis (*.exe*) para atalhos e serviços Windows
- ▶ Scripts de início e término no padrão System V (*/etc/init.d*) não são fornecidos, devem ser criados pelo administrador
- ▶ A configuração padrão do Tomcat escuta as portas 8080 (web), 8009 (AJP) e 8085 (shutdown)





# Processos e Threads

- ▶ Se preferir, confirme com o comando **ps** que o Tomcat está realmente no ar:
  - ▶ **# ps ax | grep tomcat**  
5093 ? Rl 0:01 /usr/lib/jvm/java/bin/java -  
Djava.endorsed.dirs=/usr/share/tomcat5/common/endorsed -classpath ... restante  
dos argumentos omitidos
- ▶ Para ver os threads, acrescente a opção **-L** ao comando **ps**





# Portas TCP

- ▶ Também é possível verificar com o comando `netstat` as portas em uso:

- ▶ `# netstat -anp`

Conexões Internet Ativas (servidores e estabelecidas)

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State	PID/Program name
-------	--------	--------	---------------	-----------------	-------	------------------

...

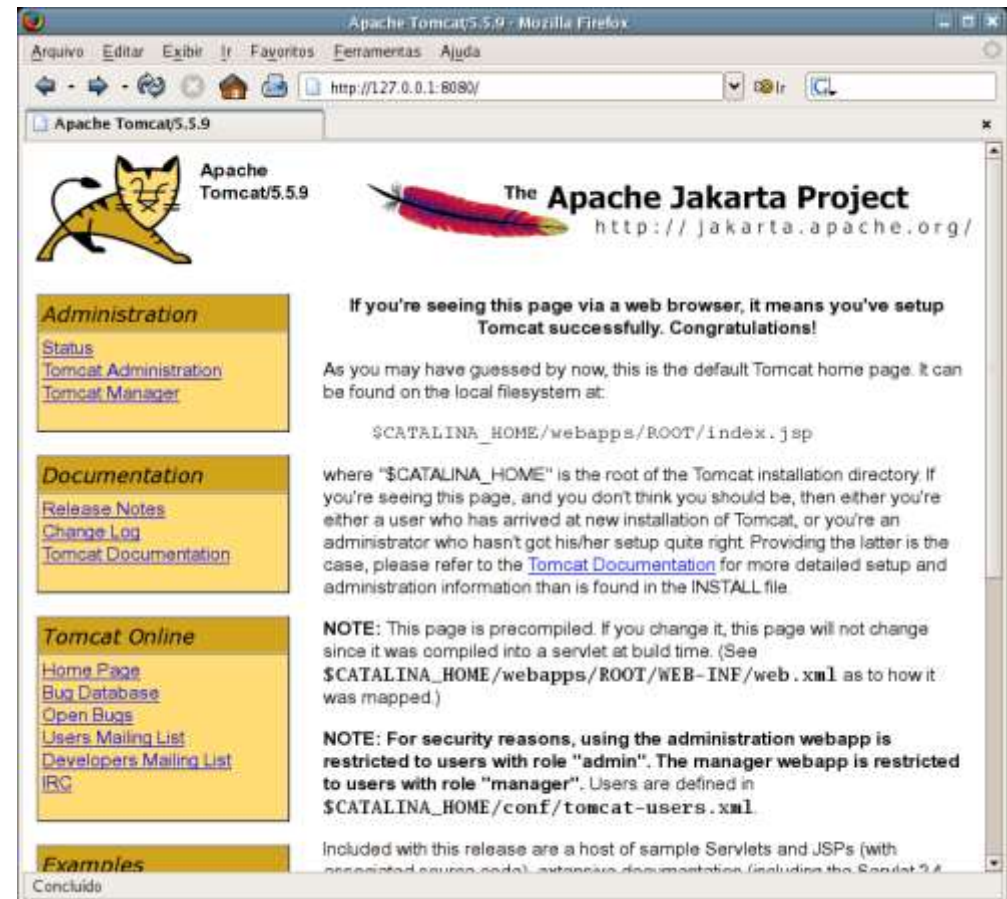
tcp	0	0	0.0.0.0:8080	0.0.0.0:*	OUÇA	5849/java
-----	---	---	--------------	-----------	------	-----------

- ▶ É possível, mas não recomendado, encerrar o Tomcat com o comando **kill**
- ▶ Softwares Java como o Tomcat não reconhecem sinais padrões de serviços de rede do Unix como **SIGHUP**



## 2.3. Testando o Tomcat

- ▶ Visite a URL  
<http://127.0.0.1:8080>
- ▶ O resultado será a página inicial do Tomcat, com links para programas exemplos e documentação





## 2.4. Se Algo Deu Errado...

- ▶ Verifique os logs do Tomcat, em especial *logs/catalina.out*
- ▶ Verifique se o comando **java** pode ser executado diretamente pela linha de comando
- ▶ Verifique se as portas 8080, 8009 e 8085 estavam livres antes do início do Tomcat
- ▶ Verifique se a estrutura de diretórios do Tomcat foi preservada depois da descompactação do arquivo zip
- ▶ Se tudo o mais falhar, encerre todos os processos “java” ativos e reinstale o Tomcat do zero







## 2.5. Documentação do Tomcat

- ▶ Manual de referência + HOW TO's em HTML
- ▶ JavaDoc das classes internas
- ▶ Parte do download “core” ou do pacote tomcat-docs-webapp





# Conclusão

- ▶ Lab 2.1. Instalação do Tomcat via Jpackage
- ▶ Lab 2.2. Instalação manual do Tomcat
- ▶ Lab 2.3. Acesso off-line aos manuais
- ▶ Lab 2.4. Provocando erros na inicialização
- ▶ Questões de Revisão





# **Servidores de Aplicações Java EE usando Tomcat**

## **Capítulo 3** **Arquitetura do Tomcat**





# Objetivo

Agora vamos olhar para o Tomcat “por dentro”, como um administrador deve fazer

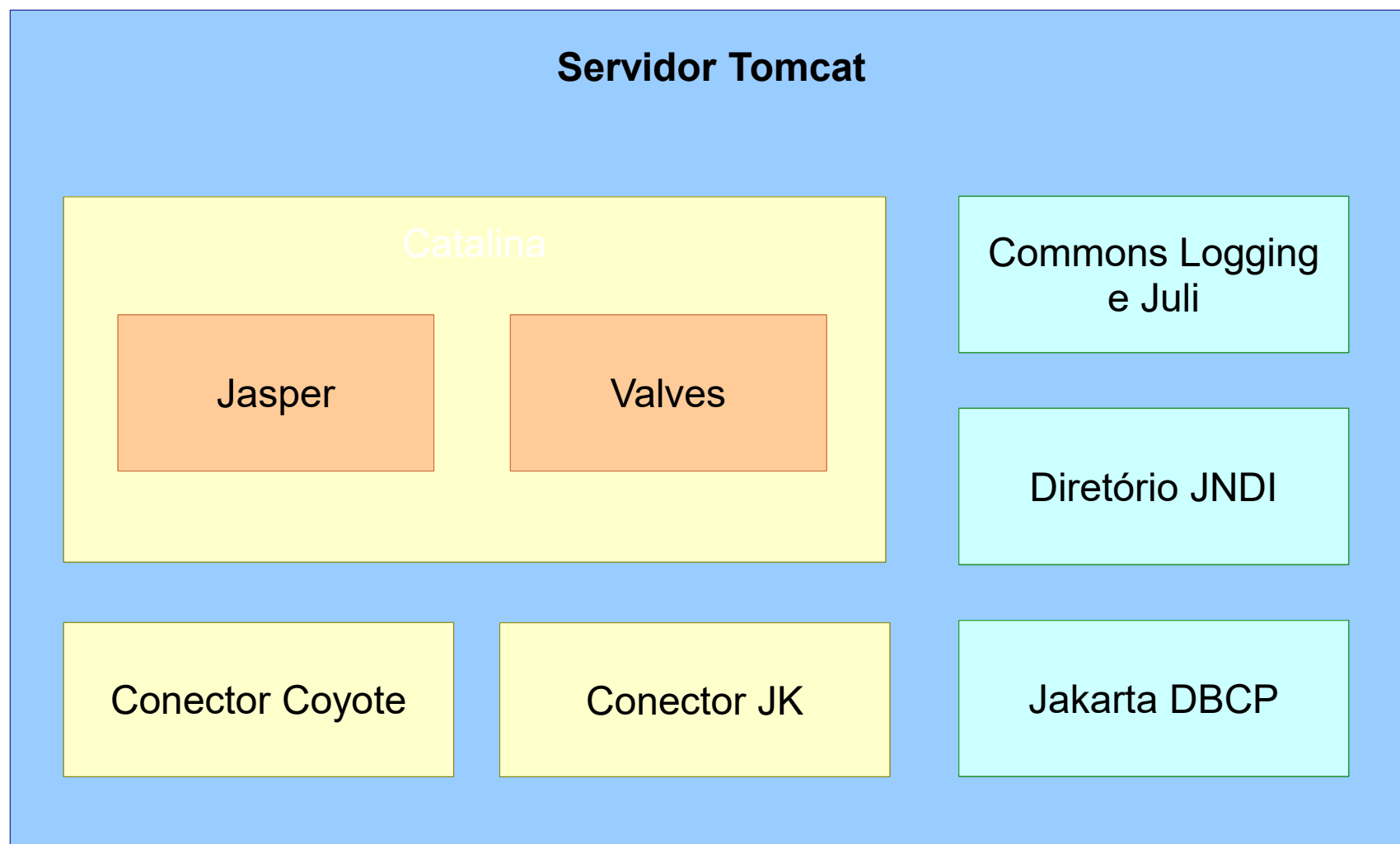
► **Tópicos:**

- Arquitetura do Tomcat
- Aplicações administrativas





## 3.1. Componentes do Tomcat





## 3.2. Arquivos do Tomcat

► diretório */usr/share/tomcat* contém links para os diretórios onde estão os arquivos do Tomcat, simulando a organização de arquivos dos downloads oficiais na Apache Software Foundation

► # `ls -l /usr/share/tomcat/`  
... bin  
... conf -> /etc/tomcat  
... lib -> /usr/share/java/tomcat  
... logs -> /var/log/tomcat  
... temp -> /var/cache/tomcat/temp  
... webapps -> /var/lib/tomcat/webapps  
... work -> /var/cache/tomcat/work





# Layout JPackage

- ▶ O administrador do Tomcat irá se preocupar com três diretórios em particular:
  - ▶ */var/lib/tomcat* contém os diretórios onde são instaladas bibliotecas Java (arquivos *\*.jar*) e o subdiretório *webapps* onde são instaladas aplicações web como arquivos *\*.war* ou como subdiretórios
  - ▶ */etc/tomcat* contém os arquivos de configuração do Tomcat, em especial *server.xml* e *tomcat-users.xml*
  - ▶ */var/log/tomcat* contém os logs do servidor Tomcat e de aplicações que falam uso da API de logging do Java





## 3.2.1. Arquivos de Configuração do Tomcat

- ▶ catalina.policy – security manager da JVM
- ▶ catalina.properties – configurações de classloader
- ▶ context.xml – configurações default de contexto para aplicações web
- ▶ logging.properties – arquivos de log
- ▶ server.xml – arquivo principal de configuração
- ▶ tomcat.conf (apenas JPackage) – opções de inicialização da JVM
- ▶ tomcat-users.xml – definições de usuários e roles
- ▶ web.xml – configurações default de Servlets







## 3.3. Estrutura do server.xml

### ▶ <Server>

O próprio Tomcat

#### ▶ <GlobalNamingResources>

Objetos JNDI globais

#### ▶ <Service>

Serviço oferecido para a rede  
(no momento apenas o container web)

##### ▶ <Conector>

Protocolo para acesso por clientes

##### ▶ <Engine>

Ccontainer web em si

##### ▶ <Host>

Host virtual, baseado em nome ou IP

##### ▶ <Context>

Uma aplicação web (WAR)





# Outros Elementos do server.xml

- ▶ A maioria destes elementos podem ser inseridos em qualquer nível da estrutura do Tomcat:
  - ▶ **<Realm>** fornece configurações de autenticação de login e senha
  - ▶ **<Resource>** define conexões a bancos de dados, servidores de e-mail, EJBs, etc
  - ▶ **<Valve>** modifica o processamento de requisições, por exemplo para gerar logs de acesso ou depuração da requisição HTTP





## 3.4. Tomcat Manager

- ▶ É utilizada para ativar, desativar e recarregar aplicações web (pacotes WAR) hospedados pelo Tomcat
- ▶ Também permite a obtenção de relatórios sobre o status atual do servidor
- ▶ Foi criada para ser acessada por scripts (utilizando **wget**, por exemplo), e não por humanos, por isso sua interface simples
- ▶ Entre nela pela URL `http://127.0.0.1:8080/manager/html`





## 3.4.1. Ativação do Manager

- ▶ Para evitar riscos de segurança, a instalação padrão do Tomcat não define nenhum usuário com acesso às aplicações administrativas
- ▶ Para criar este usuário, deve ser editado o arquivo *conf/tomcat-users.xml*
- ▶ Deve ser acrescentado um usuário contendo o role **manager**.
- ▶ Como acrescentar este usuário será auto-explicativo pelos exemplos fornecidos no próprio arquivo.
- ▶ O Tomcat deve ser reiniciado para que o novo usuário seja reconhecido



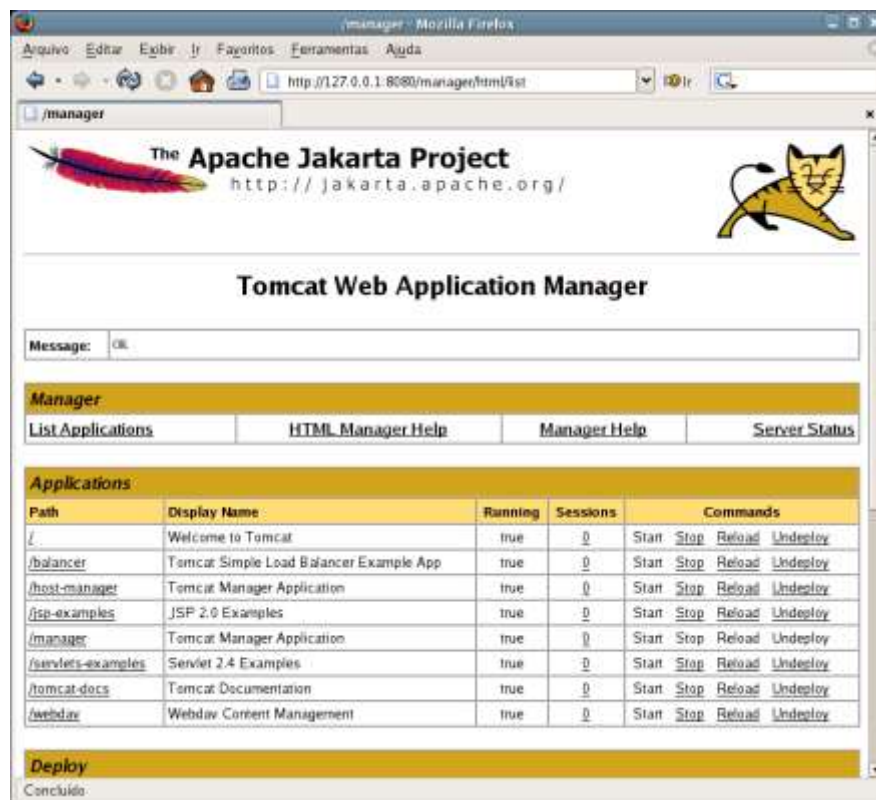


# Editando o Arquivo tomcat-users.xml

```
▶<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <role rolename="manager"/>
  <user username="tomcat" password="tomcat"
    roles="tomcat"/>
  <user username="both" password="tomcat"
    roles="tomcat,role1"/>
  <user username="role1" password="tomcat"
    roles="role1"/>
  <user username="admin" password="senha"
    roles="manager"/>
</tomcat-users>
```



# Exemplos do Manager



The Apache Jakarta Project  
<http://jakarta.apache.org/>

## Tomcat Web Application Manager

Message: OK

**Manager**

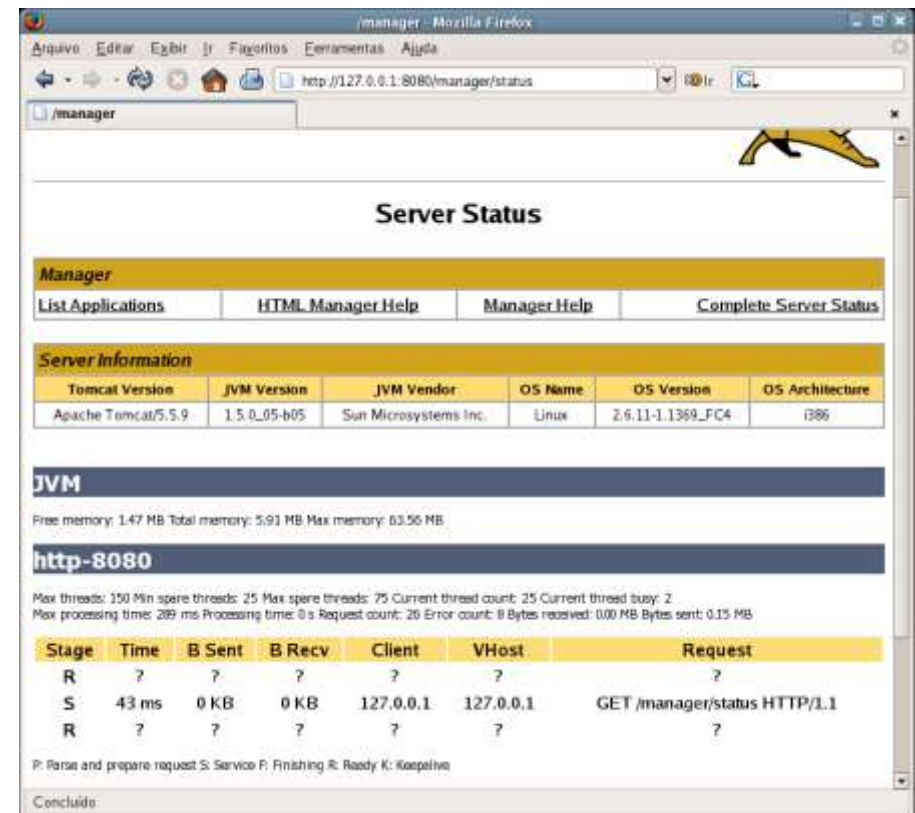
[List Applications](#)
[HTML Manager Help](#)
[Manager Help](#)
[Server Status](#)

**Applications**

Path	Display Name	Running	Sessions	Commands
/	Welcome to Tomcat	true	0	Start Stop Reload Undeploy
/balancer	Tomcat Simple Load Balancer Example App	true	0	Start Stop Reload Undeploy
/host-manager	Tomcat Manager Application	true	0	Start Stop Reload Undeploy
/jsp-examples	JSP 2.0 Examples	true	0	Start Stop Reload Undeploy
/manager	Tomcat Manager Application	true	0	Start Stop Reload Undeploy
/servlets-examples	Servlet 2.4 Examples	true	0	Start Stop Reload Undeploy
/tomcat-docs	Tomcat Documentation	true	0	Start Stop Reload Undeploy
/webdav	Webdav Content Management	true	0	Start Stop Reload Undeploy

**Deploy**

Conclude



## Server Status

**Manager**

[List Applications](#)
[HTML Manager Help](#)
[Manager Help](#)
[Complete Server Status](#)

**Server Information**

Tomcat Version	JVM Version	JVM Vendor	OS Name	OS Version	OS Architecture
Apache Tomcat/5.5.9	1.5.0_05-b05	Sun Microsystems Inc.	Linux	2.6.11-1.1369_FC4	i386

**JVM**

Free memory: 147 MB Total memory: 5.91 MB Max memory: 63.56 MB

**http-8080**

Max threads: 150 Min spare threads: 25 Max spare threads: 75 Current thread count: 25 Current thread busy: 2  
 Max processing time: 289 ms Processing time: 0 s Request count: 26 Error count: 8 Bytes received: 0.00 MB Bytes sent: 0.15 MB

Stage	Time	B Sent	B Recv	Client	VHost	Request
R	?	?	?	?	?	?
S	43 ms	0 KB	0 KB	127.0.0.1	127.0.0.1	GET /manager/status HTTP/1.1
R	?	?	?	?	?	?

P: Parse and prepare request S: Service P: Finishing R: Ready K: Keepalive

Conclude



# Conclusão

- ▶ Lab 3.1 – Conhecendo os arquivos de configuração do Tomcat
- ▶ Lab 3.2 – Ativando o Tomcat Manager
- ▶ Lab 3.3 – Criação de script para monitoração continuada do servidor Tomcat
- ▶ Questões de Revisão





# **Servidores de Aplicações Java EE usando Tomcat**

Capítulo 4

**Instalação de Aplicações no Tomcat**







# Objetivo

- ▶ Nesta aula, aprendemos como instalar e configurar aplicações web para execução sob o servidor de aplicações Tomcat
- ▶ **Tópicos:**
  - ▶ Estrutura de pacotes WAR
  - ▶ Deployment de aplicações automático e manual
  - ▶ Atualização de aplicações
  - ▶ Desligando o auto-deploy





## 4.1. Pacotes WAR e Deployment Descriptors

- ▶ Uma aplicação web em Java deve ser empacotada em um formato chamado WAR
- ▶ O WAR é um JAR (que por sua vez é um ZIP) onde existe uma pasta *WEB-INF* contendo as classes Java da aplicação e o *descriptor de deployment web.xml*
- ▶ Outros arquivos são tratados como páginas HTML estáticas ou páginas JSP dinâmicas
- ▶ Apenas pacotes WAR podem ser entregues para execução pelo Container Web





# Pacotes WAR Abertos e Fechados

- ▶ Embora formalmente o formato WAR seja um arquivo compactado, a maioria dos servidores de aplicação Java EE aceita uma pasta contendo subdiretórios na mesma estrutura
- ▶ É uma conveniência para o desenvolvedor, facilitando o teste de modificações pontuais





## 4.2. Exemplo de WAR

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





## 4.2.1. Compilando o Exemplo

- ▶ Configure o classpath do sistema para incluir a API de Servlets, fornecida pelo Tomcat
- ▶ Entre na pasta que contém as classes do exemplo
- ▶ Compile as classes
- ▶ Verifique que a pasta *WEB-INF/classes/exemplo* da aplicação contém agora dois arquivos *\*.class* que correspondem aos dois arquivos Java presentes na mesma pasta





# Comandos para Compilar o Exemplo

- ▶ (Cada comando deve ser digitado em uma única linha!)
- ▶ `$ export CLASSPATH=$CLASSPATH:/usr/share/java/servlets.jar`
- ▶ `$ cd ~Cap4/Lab1`
- ▶ `$ cd WEB-INF/classes`
- ▶ `$ javac exemplo/*.java`





## 4.2.2. Empacotando o Exemplo

- ▶ Estando na sua pasta home, execute o comando
  - ▶ `$ cd ~/Cap4`
  - ▶ `$ jar cvf hoje.war -C Lab1 .`
- ▶ Observe o uso da opção **-C** e que o último argumento é um ponto, indicando o diretório corrente
- ▶ Isto é necessário porque o pacote WAR não deve contar a pasta de topo da aplicação (no caso, *Lab1*)





# Verificando um Pacote WAR

- ▶ Liste o conteúdo do pacote usando o comando jar:

- ▶ `$ 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/
```

...

- ▶ Também pode ser usado o comando unzip

- ▶ `$ unzip -t hoje.war`







## 4.3. Deployment de Aplicações Web

- ▶ É o processo de instalação de uma aplicação web Java EE dentro de um container web, tornando esta aplicação disponível para seus usuários
- ▶ Envolve garantir que todas as configurações e recursos requeridos pela aplicação estejam disponíveis no servidor onde ela é instalada
- ▶ Há várias formas locais e remotas de realizar o deployment com o Tomcat
  - ▶ Auto-deploy (cópia de arquivos)
  - ▶ Via o Manager





## 4.4. Auto-Deploy

- ▶ A maneira mais fácil de fazer a instalação (*deployment*) de uma aplicação no Tomcat é copiar seu pacote WAR (seja aberto ou fechado) para a pasta *webapps*
- ▶ Feito a cópia, os logs do Tomcat deverão indicar que o novo pacote foi detectado e instalado
- ▶ 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!





## 4.4.1. Permissões de Arquivos

- ▶ A pasta webapps (/var/lib/tomcat/webapps) pode ser escrita por membros do grupo “tomcat”
- ▶ Então se acrescente a este grupo!
- ▶ `# usermod -aG tomcat <<login>>`
- ▶ Faça novo login, e confirme o novo grupo com o comando **id**





## 4.4.2. Re-deployment Automático

- ▶ A instalação padrão do Tomcat monitora pacotes fechados
- ▶ Ou então o descritor *web.xml* de pacotes abertos, recarregando (re-deployment) em caso de mudanças
- ▶ Mudanças em páginas JSP são detectadas e efetivas no próximo acesso a página
- ▶ Mudanças em classes Java exigem a recarga manual da aplicação pelo Manager (link *Reload*) ou um **touch** no *web.xml*





## 4.4.3. Limpando Deployments

- ▶ A cópia de um pacote fechado para o webapps gera uma versão aberta do mesmo pacote
- ▶ Às vezes o undeploy não remove a versão aberta, que interfere em futuros deployments
- ▶ O undeploy via Manager nunca deixa lixo!





## 4.5. Testando o Deployment

- ▶ Verifique os logs
  - ▶ `$ tail /var/logs/tomcat/catalina.out`
  - ...
  - INFO: Deploying web application archive hoje.war
- ▶ Verifique a presença da aplicação no Manager
- ▶ O nome da aplicação será igual ao nome do pacote WAR, ou seja, “hoje”
- ▶ `http://127.0.0.1:8080/hoje`





## 4.6. Deployment Via Manager

- ▶ Além do auto-deploy pela cópia de arquivos, o Tomcat suporta o deployment pelo Manager, que pode inclusive ser feito remotamente
- ▶ Se for um deployment local, o manager pode apontar o Tomcat diretamente para o pacote WAR da aplicação, sem necessidade de copiar para a pasta webapps
- ▶ O nome do contexto (nas URLs) pode inclusive ser diferente do nome do pacote
- ▶ Se for um deployment remoto, deve ser obrigatoriamente fornecido um pacote WAR fechado para *upload*, que será salvo em *webapps*





# Formulários do Manager

- ▶ Primeiro para deployments locais
- ▶ Segundo permite deployments remotos

## Deploy

### Deploy directory or WAR file located on server

Context Path (optional):

XML Configuration file URL:

WAR or Directory URL:

Deploy

### WAR file to deploy

Select WAR file to upload

do/ap-tomcat.src/exemplo1.2/dist/exemplo1.2.war

Arquivo...

Deploy







## 4.6.1. Atualizando e Suspendendo Aplicações

- ▶ Links “Reload” e “Stop” do Manager, em “List Applications”
- ▶ Falhas durante o deployment podem deixar a aplicação suspensa (e não adianta tentar usar o link “Start”)





## 4.7. Desligando o Auto-deploy

- ▶ Configurado no elemento <Host> do server.xml
- ▶ <Host name="localhost" appBase="webapps"  
unpackWARs="true" **autoDeploy="false"**  
xmlValidation="false" xmlNamespaceAware="false">





# Conclusão

- ▶ Lab 4.1. Compilar, empacotar e (auto-)deployar aplicação
- ▶ Lab 4.2. Auto-deploy de pacote aberto
- ▶ Lab 4.3. Recarga via Manager
- ▶ Lab 4.4. Deployment Manual (Manager)
- ▶ Lab 4.5. Desligar auto-deploy – depois religue!
- ▶ Questões de Revisão





# **Servidores de Aplicações Java EE usando Tomcat**

## **Capítulo 5** **Configuração de Contextos**





# Objetivo

- ▶ 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





## 5.1. O Descritor web.xml

- ▶ O arquivo *WEB-INF/web.xml* fornece os parâmetros de configuração padronizados pela API de Servlets
- ▶ A maioria atende ao desenvolvedor de aplicações, mas alguns devem ser modificados pelo administrador
- ▶ Parâmetros que não são definidos pelo padrão são configurados no “descritor proprietário” do servidor de aplicações, no Tomcat é a definição de contextos
- ▶ 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)





## 5.1.1. Página Inicial da Aplicação

- ▶ Definida pelas entradas **<welcome-file>** no *web.xml*.
- ▶ As várias páginas de boas-vindas são escolhidas na ordem em que aparecem no descritor de deployment

```
<welcome-file-list>
```

```
    <welcome-file>index.html</welcome-file>
```

```
    <welcome-file>index.jsp</welcome-file>
```

```
</welcome-file-list>
```





## 5.1.2. Entradas de Ambiente JNDI

- ▶ Parâmetros de configuração para a aplicação
- ▶ Alternativa “enterprise” aos arquivos de propriedades
- ▶ Valores simples vinculados a um nome

<env-entry>

<env-entry-name>saudacao</env-entry-name>

<env-entry-value>rubro-negras</env-entry-value>

<env-entry-type>java.lang.String</env-entry-type>

</env-entry>







## 5.2. 0 Descritor Proprietário context.xml

- ▶ Colocado na pasta *META-INF* do pacote WAR
- ▶ Específico para o Tomcat, ignorado por outros servidores
- ▶ É comum um pacote WAR carregar descritores proprietários para vários servidores diferentes
- ▶ Modificado quase que exclusivamente pelo administrador





## 5.2.1. Sobrepondo Entradas de Ambiente

- ▶ A entrada **<Environment>** em *context.xml* pode sobrepor o valor de **<env-entry>** em *web.xml*

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

```
<Context>
```

```
  <Environment
```

```
    name="saudacao"
```

```
    type="java.lang.String"
```

```
    value="coloradas"
```

```
    override="false"/>
```

```
</Context>
```





## 5.3. Definição de Contextos no Tomcat

- ▶ Implícita
  - ▶ Auto-deploy
  - ▶ Deploy remoto via Manager
- ▶ Explícita
  - ▶ Configuração do contexto via *server.xml*
  - ▶ Configuração do contexto via *conf/host/context.xml*
  - ▶ Deploy local pelo Manager





## 5.3.1. Path e base de Contexto

- ▶ Atributo **path** do contexto no *server.xml* indica o nome do seu diretório nas URLs de acesso pelo navegador
- ▶ Nos demais casos o nome do contexto é dado pelo nome do próprio pacote ou arquivo XML
- ▶ **docBase** aponta para o próprio pacote (em veram explodido)
- ▶ Relativo ao **appBase** do Host pai
- ▶ Vários contextos podem ter o mesmo **docBase** porém **path** diferente





# Exemplo de Contexto

- ▶ Esta definição de contexto poderia estar em *server.xml*
  - ▶ `<Host>`
    - ...
    - `<Context path="app"`
      - `docBase="/usr/share/java/pacote.war">`
      - `</Context>`
    - ...
    - `<Host>`
- ▶ Dentro do elemento **<Context>** poderiam ser aninhadas definições de **<Valve>**, **<Realm>** e **<Resource>**
- ▶ Na prática preferimos usar o META-INF do pacote ou a pasta `conf/Engine/Context`





## 5.3.2. Contextos Explícitos com Auto-Deploy

- ▶ O arquivo de configuração *conf/Engine/nome-do-war.xml* tem que ter o mesmo nome do pacote colocado em *webapps*
- ▶ O arquivo XML devem ser colocado ANTES do pacote em *webapps*





## 5.3.3. Página Inicial do Tomcat

- ▶ É fornecida pelo contexto raiz, que pode ser:
  - ▶ O pacote *ROOT.war*
  - ▶ Ou então por qualquer contexto com **path** vazio
- ▶ É claro, não podem haver dois contextos raízes





## 5.4. Instalação sem deployment

- ▶ Se for definido o arquivo *conf/Engine/Host/nome-do-contexto.xml* não existe um deploy propriamente dito
- ▶ O contexto é ativado na inicialização do Tomcat, e mudanças sobre suas classes ou configurações exigem reinício
- ▶ No tomcat o manager era instalado desta forma:
  - ▶ `conf/Catalina/localhost/manager.xml`
  - ▶ `server/webapps/manager`







# Conclusão

- ▶ Lab 5.1. Entradas de ambiente
- ▶ Lab 5.2. Sobrepondo entradas de ambiente
- ▶ Lab 5.3. Múltiplas instâncias da mesma aplicação, via server.xml
- ▶ Lab 5.4. Múltiplas instâncias da aplicação, via conf/Engine/Host e aplicação padrão
- ▶ Lab 5.5. Página inicial da aplicação





# **Servidores de Aplicações Java EE usando Tomcat**

## **Capítulo 6** **Introdução ao Ant**





# Objetivo

- ▶ 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





## 6.1. O que é o Ant

- ▶ Ferramenta “obrigatória” para automatização de tarefas no desenvolvimento Java
- ▶ Suportada e utilizada pela maioria dos IDEs
- ▶ Espera-se que qualquer ferramenta Java se integre ao ant: testes de unidade, geração de relatórios, etc
- ▶ Útil para o administrador pela sua capacidade de automatizar tarefas como empacotamento e deployment de aplicações





## 6.2. Instalação via JPackage

- ▶ Basta instalar via **yum** o pacote ant.
- ▶ Verifique se já foi instalado usando o comando **rpm**:
  - ▶ rpm -q ant
- ▶ Traz como dependências os pacotes que incluem seus vários plug-ins especializados





## 6.3. Instalação Manual

- ▶ Visite <http://ant.apache.org> e clique no link “Binary Distributions”
- ▶ Baixe o arquivo *apache-ant-<versão>-bin.zip* (ou uma versão mais recente) e descompacte na sua pasta home
- ▶ Defina a variável de ambiente ANT\_HOME apontando para o diretório de instalação do Ant
  - ▶ \$ export ANT\_HOME=\$HOME/apache-ant-\*
- ▶ Acrescente a pasta bin do Ant ao path de comandos do shell
  - ▶ \$ export PATH=\$ANT\_HOME/bin:\$PATH





## 6.3.1. Instalação Manual x JPackage

- ▶ Caso você deseje baixar e instalar manualmente usar um Ant mais recente do que o incluso na distribuição, verifique o arquivo */etc/ant.conf*
- ▶ O conteúdo deste arquivo tem precedência sobre a configuração da variável **ANT\_HOME**, então ele deverá ser editado para indicar a instalação correta do Ant.





## 6.4. Verificando a Instalação do Ant

- ▶ Execute o comando **ant**:
  - ▶ **\$ ant -version**  
Apache Ant version X.X.X compiled on February 23 20XX
- ▶ Confirme que você está usando a instalação correta:
  - ▶ **\$ which ant**  
/usr/bin/ant







## 6.5. Sintaxe dos buildfiles

- ▶ Os scripts do ant chamados de *buildfiles*, e o comando **ant** executa por padrão o script *build.xml*
- ▶ Usam a sintaxe XML, organizado na hierarquia:
  - ▶ Projeto (project)
    - ▶ Alvo (target)
      - ▶ Tarefa (task)
- ▶ Há tarefas pré-definidas para compilação, empacotamento e execução de aplicações Java, além de manipulação de arquivos
- ▶ Alvos podem depender uns dos outros, de modo similar a um Makefile





## 6.5.1. Exemplo de buildfile

```
<project name="Cap6_Lab1" default="tudo">  
  <target name="tudo"  
    depends="limpa,compila,empacota,instala" />  
  <target name="variaveis">  
    <property name="tomcat" value="/usr/share/tomcat" />  
    ...  
  </target>  
  <target name="instala" depends="variaveis">  
    <copy file="dist/${war}" todir="${tomcat}/webapps" />  
  </target>
```





# Executando o Ant

- ▶ Comando **ant** sem argumentos executa o alvo default indicado em **<project>**
- ▶ A opção **-f** indica o nome do buildfile
- ▶ Outros argumentos indicam alvos
- ▶ Ex: Após modificação apenas de páginas JSP ou descritores de deployment de uma aplicação, não é necessário refazer tudo, mas apenas  
ant empacota instala





# Construção de Pacotes WAR com Apache Ant

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





## 6.5.2. Exemplo de projeto com Ant

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





## 6.6. Tasks customizados do Tomcat

- ▶ Permitem comandar operações do Manager, como deployment, reload, stop e undeployment de aplicações
- ▶ Também podem ser usadas para acionar MBeans do Tomcat





# Declarando os tasks

- Acrescente ao buildfile:

```
► <taskdef name="deploy"  
  classpath="${tomcat}/server/lib/catalina-ant.jar"  
  classname="org.apache.catalina.ant.DeployTask"  
/>  
<taskdef name="undeploy"  
  classpath="${tomcat}/server/lib/catalina-ant.jar"  
  classname="org.apache.catalina.ant.UndeployTask"  
/>
```





# Usando os tasks

- ▶ Novamente dentro do buildfile:

- ▶ `<property name="url"  
value="http://127.0.0.1:8080/manager">`

- ▶ `<target name="instala" depends="variaveis">  
 <deploy url="${url}"  
 username="${username}"  
 password="${password}"  
 path="/${war}" war="dist/${war}.war"/>  
</target>`

- ▶ Lembre de modificar as propriedades **username** e **password** conforme sua configuração em *tomcat-users.xml*







# Conclusão

- ▶ Lab 6.1. Projeto com Ant
- ▶ Lab 6.2. Deployment com Ant





# Servidores de Aplicações Java EE usando Tomcat

## Capítulo 7 Bibliotecas





# Objetivo

- ▶ 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





## 7.1. Bibliotecas Java

- ▶ Bibliotecas Java são em geral fornecidas como pacotes JAR, que são arquivos ZIP contendo classes Java compiladas (arquivos *.class*)
- ▶ Muitas são utilizadas internamente pelo Tomcat
- ▶ Outras são mandatórias em aplicações Java EE
- ▶ Além disso, qualquer aplicação pode necessitar de bibliotecas adicionais





## 7.1.1. Onde Instalar Bibliotecas

- ▶ Na pasta *lib* da sua instalação do Tomcat (visíveis para todas as aplicações)
- ▶ Na pasta *WEB-INF/lib* do próprio pacote WAR (visíveis apenas pela própria aplicação)
- ▶ Ordem de busca por bibliotecas
  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.





## 7.2. Classloaders do Java x Tomcat

- ▶ Um classloader na JVM indica quais classes podem ser vistas por que outras classes
- ▶ O Tomcat roda em um classloader separado das aplicações
- ▶ Cada aplicação também roda em um classloader seaparado
- ▶ O classloader de bibliotecas é “pai” dos classloaders de aplicações
- ▶ Isto evita conflitos de nomes de classes, e previne potenciais bugs de segurança





## 7.2.1. Novo diretório de bibliotecas

- ▶ No tomcat, existe apenas a pasta *lib* e o arquivo *catalina.properties* diz que JARs são carregados por qual classloader
- ▶ É possível acrescentar novos diretórios de bibliotecas para não misturar arquivos do Tomcat com arquivos acrescentados pelo administrador ou desenvolvedor:
  - ▶ `common.loader=${catalina.home}/lib,  
${catalina.home}/lib/*.jar,  
${catalina.home}/meu-lib/*.jar`
  - ▶ (sem quebras de linha)





# Conclusão

- ▶ Lab 7.1. Aplicação com biblioteca externa
- ▶ Lab 7.2. Pasta adicional de biblioteca
- ▶ Lab 7.3. Versões diferentes da mesma classe







# Servidores de Aplicações Java EE usando Tomcat

Capítulo 8

**Acesso a Bancos de Dados**





# Objetivo

- ▶ 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
  - ▶ Resources globais





## 8.1. Introdução ao JDBC

- ▶ É o padrão Java EE para acesso a bancos de dados relacionais
- ▶ Disponível no Java SE, mas sem suporte a XA
- ▶ Boa performance, mas depende da qualidade dos comandos SQL
- ▶ Executa comandos SQL como strings
- ▶ Suporta recursos avançados como stored procedures, prepared statements, catálogo e blobs
- ▶ Tendência a ser “abstraído” pelo uso de frameworks ORM como JPA e Hibernate





## 8.1.1. Tipos de Drivers JDBC

- ▶ Tipo 1: ponte JDBC-ODBC, não use
- ▶ Tipo 2: Biblioteca nativa via JNDI, recomendação de muitos fornecedores mas representa risco contra a estabilidade e portabilidade das aplicações
- ▶ Tipo 3: Gateway de rede, hoje seu uso é limitado a middleware como o Sequoia (antigo C-JDBC)
- ▶ Tipo 4: Implementa o protocolo proprietário do banco em Java, é opção preferencial e suportado pela grande maioria dos fornecedores





## 8.2. DataSources Java EE

- ▶ DataSource Java EE == DataSource JDBC == DataSource JNDI
- ▶ Parte do padrão JCA (*Java Connector Architecture*) suportado apenas parcialmente pelo Tomcat
- ▶ DataSources fornecem a forma recomendada de se obter conexões a BDs em aplicações Java EE, usando lookups JNDI
- ▶ Ou seja, evite o modo Java SE de abrir conexões (usando o DriverManager)
- ▶ O padrão Java EE exige a definição de **<resource-ref>** no *web.xml*, mas o Tomcat não





# Forma correta de abrir conexões ao BD no Java EE

```
InitialContext ctx = new InitialContext();  
ds = (DataSource)ctx.lookup(  
    "java:comp/env/jdbc/Contatos");  
Connection con = ds.getConnection();
```





## 8.2.2. Porque Usar Datasources

- ▶ Conexões a bancos de dados são recursos caros computacionalmente, pode ser inviável manter várias conexões abertas em aplicações com grande quantidades de usuários, sendo que em um dado momento a maioria delas estarão ociosas
- ▶ Abrir e fechar conexões conforme a demanda (a cada página requisitada) é demorado, e iria prejudicar o tempo de resposta do usuário
- ▶ Um DataSource mantém um pool onde conexões ociosas podem ser requeridas e utilizadas por outro processo





## 8.2.3. Pools de Conexão

- ▶ Pode-se estabelecer um mínimo de conexões, de modo que picos repentinos possam ser atendidos prontamente
- ▶ Pode-se estabelecer um máximo, de modo que nem o servidor de aplicações nem o banco de dados fiquem sobrecarregados
- ▶ A aplicação, em vez de abrir e fechar conexões, requisita conexões do pool e as devolve o mais cedo possível
- ▶ (mas é programada como se estivesse abrindo e fechando conexões a todo momento)
- ▶ O servidor de aplicações é o responsável pelo gerenciamento do pool







## 8.3. Configurando de DataSources

- ▶ Podem ser definidas no *web.xml* em **<GlobalNamingResources>** ou então em uma definição de contexto
- ▶ Deve ser inserido o elemento **<Resource>** especificando os parâmetros de conexão JDBC:
- ▶ 

```
<Resource name="jdbc/Contatos" auth="Container"
  type="javax.sql.DataSource"
  maxActive="10" maxIdle="2" maxWait="-1"
  username="tomcat" password="secreta"
  driverClassName="org.hsqldb.jdbcDriver"
  url="jdbc:postgresql://127.0.0.1/contatos"
/>
```
- ▶ Também define os parâmetros do pool





## 8.3.1. Links para recursos globais

- ▶ Devem ser definidos no contexto da aplicação para que ela possa fazer lookups pelo DataSource global usando o `java:comp/env`
- ▶ 

```
<ResourceLink name="jdbc/ContatosLocal"
    global="jdbc/ContatosGlobal"
    type="javax.sql.DataSource"
/>
```
- ▶ No Tomcat, ao contrário e alguns outros servidores Java EE, não é possível fazer buscas diretamente no espaço global





# Instalação do Driver do Banco no Tomcat

- ▶ Como o pool de conexões é mantido pelo próprio Tomcat, ele precisa das classes do driver na pasta *lib* ou alguma outra vinculada ao classloader “common”
- ▶ Em vez de copiar o driver para a pasta do Tomcat, use um link simbólico:
  - ▶ 

```
# ln -s /usr/share/java/postgresql-driver.jar \
```

```
    /usr/share/tomcat/lib
```
- ▶ Não adianta colocar o driver na pasta WEB-INF/lib, mesmo que o DataSource seja configurado no *META-INF/context.xml* da aplicação em vez de no *server.xml* do Tomcat
- ▶ Lembre de reiniciar o Tomcat!





# Conclusão

- ▶ Lab 8.1. DataSource local
- ▶ Lab 8.2. DataSource global
- ▶ Questões de Revisão





# Servidores de Aplicações Java EE usando Tomcat

## Capítulo 9 Autenticação HTTP





# Objetivo

- ▶ 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 do Tomcat





## 9.1. Segurança Declarativa do Java EE

- ▶ A idéia é que o desenvolvedor escreva o mínimo possível de código para autenticar usuários e autorizar o acesso às páginas da aplicação
- ▶ Este controle deve ser delegado para o servidor de aplicações, que poderá fazê-lo de forma mais flexível e segura
- ▶ Plug-ins de autenticação são padronizados pelo **JAAS** (*Java Authorization and Authentication Services*)
- ▶ Configurado em duas partes:
  - ▶ Bases de identidade no servidor de aplicações
  - ▶ Restrições de acesso no descritor padrão da aplicação





## 9.1.1 Resource Collections do Java EE

- ▶ No descritor da aplicação, é definido qual o método de autenticação a ser utilizado
- ▶ Além disso, padrões de URLs são agrupadas e são definidos os usuários e roles (grupos de usuários) com acesso ao conjunto de URLs
- ▶ Podem haver páginas públicas e restritas na mesma aplicação
- ▶ Então um conjunto de **<url-pattern>** é vincuado a um ou mais **<role-name>** dentro de um **<security-constraint>**, criando uma área de páginas protegidas dentro da aplicação







# Exemplo de <security-constraint>

- ▶ Restringe o acesso às páginas abaixo de check-out (finalização da compra em uma loja on-line) aos compradores cadastrados do site (role “usuario”)
- ▶

```
<security-constraint>  
  <web-resource-collection>  
    <web-resource-name>  
      Finalização da compra  
    </web-resource-name>  
    <url-pattern>/checkout/*</url-pattern>  
  </web-resource-collection>  
  <auth-constraint>  
    <role-name>usuario</role-name>  
  </auth-constraint>  
</security-constraint>
```





# Restrição de método HTTP

- ▶ Acrescenta a regra de que as páginas da finalização de uma compra (que são todas formulários HTML) só estejam disponíveis via POST, prevenindo ataques simples de injeção de HTML
- ▶

```
<security-constraint>  
  <web-resource-collection>  
    <web-resource-name>  
      Finalização da compra  
    </web-resource-name>  
    <url-pattern>/checkout/*</url-pattern>  
  </web-resource-collection>  
  <auth-constraint>  
    <role-name>usuario</role-name>  
  </auth-constraint>  
  <http-method>POST</http-method>  
</security-constraint>
```





## 9.1.2. Segurança Programática do Java EE

- ▶ O modelo de segurança declarativa do Java EE, baseado em roles e URLs não atende a todas as necessidades das aplicações
- ▶ Isto leva muitos desenvolvedores a 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 JavaEE prevê APIs para obter informações sobre o usuário autenticado e seus roles, o que é suficiente para a grande maioria dos casos





# APIs de Segurança do Java EE

## ▶ **HttpServletRequest**

- ▶ **getRemoteUser()** retorna o nome de login autenticado pelo protocolo HTTP
- ▶ **getUserPrincipal()** retorna o nome de login autenticado pelo container, seja pelos mecanismos do HTTP ou pelo form-based
- ▶ **isUserInRole()** indica se o usuário corrente pertence ao role especificado

## ▶ **ServletRequest**

- ▶ **isSecure()** indica se a conexão corrente é ou não criptografada





## 9.2. Métodos de Autenticação HTTP

- ▶ O protocolo HTTP prevê três mecanismos de autenticação
  - ▶ **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 reconhecida pelo servidor – é um recurso do SSL
- ▶ A senha fornecida pelo usuário no BASIC ou DIGEST é mantida em memória pelo navegador, que a retransmite a cada página requisitada





## 9.1.2. Limitações do HTTP

- ▶ O BASIC é o único obrigatório, e é seguro apenas se usado em conjunto com SSL / TLS
- ▶ Versões recentes do Firefox e IE suportam DIGEST, que é mais ou menos seguro por si só
- ▶ O CLIENT-CERT é pouco usado na prática pelas dificuldades logísticas de distribuição dos certificados digitais para os usuários
- ▶ Em nenhum destes mecanismos a aplicação tem controle sobre a aparência da janela de login exibida pelo navegador
- ▶ Não existe um logout devido à natureza *stateless* do HTTP





## 9.2.2. Regras de Autenticação no web.xml

- ▶ Não são especificados usuários individuais, e sim as roles as quais eles devem pertencer
- ▶

```
<security-role>  
  <role-name>usuario</role-name>  
</security-role>  
<security-role>  
  <role-name>administrador</role-name>  
</security-role>
```
- ▶ O *web.xml* também indica o mecanismo de autenticação para a aplicação
- ▶

```
<login-config>  
  <auth-method>BASIC</auth-method>  
  <realm-name>  
    Exemplo de Segurança J2EE  
  </realm-name>  
</login-config>
```





## 9.2.3. 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
- ▶ O mecanismo “form-based” utiliza o mecanismo de **sessão HTTP** mantido pelo container para acompanhar um mesmo usuário navegando pela aplicação
- ▶ Este cookie é seguro mesmo sem o uso de SSL / TLS, mas a senha é transmitida em texto claro durante o login – Portanto verifique se o desenvolvedor tomou o cuidado de colocar a página de login em https:







# Configuração da Aplicação

- ▶ Muda apenas o **<login-config>**, que deve especificar as páginas de login e de erro de login (que não devem ser protegidas!)
- ▶ `<login-config>`
  - `<auth-method>FORM</auth-method>`
  - `<form-login-config>`
    - `<form-login-page>/login.jsp`
    - `</form-login-page>`
    - `<form-error-page>/loginInvalido.jsp`
    - `</form-error-page>`
  - `</form-login-config>`
- ▶ Definições de roles e **<security-constraints>** permanecem inalteradas





## 9.2.4. Logout

- ▶ Só é possível fazer explicitamente um logout com a autenticação FORM
- ▶ Neste caso, basta encerrar a sessão HTTP, chamando **HttpSession.invalidate()**
- ▶ Mas nada obriga o usuário a fazer o logout, e o servidor não tem como saber se o usuário deixou o site ou fechou seu navegador
- ▶ As sessões HTTP (e consequentemente os logins) são invalidados por inatividade, conforme configuração no descritor *web.xml*





# Logout por Inatividade

- ▶ Configuração no *web.xml* para invalidação da sessão e logout por inatividade
- ▶ `<session-config>`  
    `<session-timeout>10</session-timeout>`  
    `</session-config>`
- ▶ O tempo de inatividade pode ainda ser configurado programaticamente chamando **`HttpSession.setMaxInactiveInterval()`** passando o tempo em segundos





## 9.3. A Base de Identidade default do Tomcat

- ▶ Já foi vista por alto quando habilitamos o acesso ao Manager
- ▶ Para acrescentar novos usuários:
- ▶ 

```
<tomcat-users>  
  <user username="tomcat" password="tomcat" roles="manager"/>  
  <user username="chefe" password="boss" roles="administrador"/>  
  <user username="fulano" password="testando" roles="usuario"/>  
</tomcat-users>
```
- ▶ Um usuário pode ter vários roles (separados por vírgula)
- ▶ Necessário reinício do Tomcat





## 9.4. Single Sign-On

- ▶ Pelo padrão Java EE, cada pacote WAR forma um “contexto de segurança” independente
- ▶ Então navegar de uma aplicação para outra, no mesmo servidor, provoca novo pedido de login
- ▶ O Tomcat oferece uma válvula que permite violar o padrão Java EE e propagar as credenciais de um usuário autenticado de uma aplicação para outra
- ▶ Basta descomentar no *web.xml*
- ▶ `<Valve className=  
"org.apache.catalina.authenticator.SingleSignOn" />`





# Conclusão

- ▶ Lab 9.1. Aplicação com autenticação BASIC
- ▶ Lab 9.2. Autenticação FORM
- ▶ Lab 9.3. Single Sign-On





# **Servidores de Aplicações Java EE usando Tomcat**

## **Capítulo 10** **Bases de Identidade (Realms)**





# Objetivo

▶ 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







# 10.1. Realms no Tomcat

- ▶ Uma base de identidade do Tomcat é configurada por meio de um elemento **<Realm>** em qualquer ponto da hierarquia Host/Engine/Contexto
- ▶ Dentre os vários tipos de realms fornecidos pelo Tomcat, os mais usuais são:
  - ▶ **UserDatabaseRealm** (arquivo *tomcat-users.xml*)
  - ▶ **DatasourceRealm** (banco de dados relacional)
  - ▶ **JNDIRealm** (diretório LDAP)
  - ▶ **JAASRealm** (usa **LoginModules** padrão JAAS)





# Definição do Realm padrão

- ▶ A definição do Realm default está aninhada ao elemento **<Engine>** do *server.xml*
- ▶ Então ele está disponível a todos os Hosts e consequentemente a todas as aplicações (contextos) definidos no servidor

▶ ...

```
<Engine  
  defaultHost="localhost"  
  name="Catalina">
```

```
  <Realm className="org.apache.catalina.realm.  
  UserDatabaseRealm" resourceName="UserDatabase"/>
```

```
  <Host
```

```
    ...
```





# 0 UserDatabase do realm default

- ▶ O realm padrão do Tomcat é baseado em um “banco de dados em memória” que é inicializado à partir do arquivo *tomcat-users.xml*

▶ ...

```
<GlobalNamingResources>
```

...

```
  <Resource auth="Container"
```

```
    name="UserDatabase"
```

```
    type="org.apache.catalina.UserDatabase"
```

```
    pathname="conf/tomcat-users.xml"
```

```
    factory="org.apache.catalina.users.
```

```
MemoryUserDatabaseFactory"/>
```

```
</GlobalNamingResources>
```

...





## 10.1.1. Senha Criptografadas

- ▶ Ter senhas de usuários em texto claro em um arquivo como o *tomcat-users.xml*, mesmo que visíveis apenas ao administrador, representa um risco à privacidade destes usuários
- ▶ Em vez das senhas, deve ser armazenado um **hash** (ou **digest**) gerado à partir delas
- ▶ Não é possível calcular a senha original à partir do hash armazenado no arquivo
- ▶ Durante o login, é calculado o hash da senha digitada e ele é comparado com a senha armazenada





# Criptografando senhas em realms

- ▶ Basta acrescentar o algoritmo digest na definição do **<Realm>**
- ▶ Pode ser utilizado qualquer algoritmo de hash suportado pela classe **java.security.MessageDigest** do Java SE
- ▶ `<Realm className="org.apache.catalina.realm.UserDatabaseRealm" resourceName="UserDatabase" digest="MD5" />`
- ▶ Use a classe **org.apache.catalina.realm.RealmBase** para gerar os hashes que devem ser armazenados, ou então chame a classe **MessageDigest** do Java SE





# Exemplo de senha criptografada

- ▶ Eis como ficaria a senha “testando” usando o algoritmo MD5
  - ▶ `<user username="fulano"  
password="caa9c8f8620cbb30679026bb6427e11f"  
roles="usuarios"/>`





## 10.2. Autenticação via Banco de Dados

- ▶ O **DatasourceRealm**, que obtém usuários, senhas e roles de uma DataSource JNDI
- ▶ É possível configurar o nome das tabelas e das colunas consultadas, de modo que o Tomcat pode ser configurado para utilizar praticamente qualquer banco de dados de usuários gerado por outros sistemas





# Definição do DataSource Realm

- ▶ `<Realm className="org.apache.catalina.realm.DataSourceRealm"  
    dataSourceName="jdbc/Contatos"  
    digest="MD5"  
    userTable="usuarios"  
    userNameCol="login" userCredCol="senha"  
    userRoleTable="grupos"  
    roleNameCol="role"/>`
- ▶ Observe o atributo **localDataSource**!







## 10.3. Autenticação via Diretório LDAP

- ▶ O Tomcat também fornece o **JNDIRealm** que permite a autenticação contra serviços de diretório LDAP como o OpenLDAP, Fedora Directory Server, Novell NDS e Microsoft AD
- ▶ A tecnologia LDAP tem sido a preferida para base de identidade corporativa devido ao seu alto grau de padronização e facilidade de escalar
- ▶ Cuidado com a definição do **schema** do seu diretório LDAP!
- ▶ O Tomcat pode tanto ler login, senha e roles do diretório quando realizar diretamente o **bind** do usuário contra o diretório





# Exemplo de JNDI Realm 1/2

- ▶ O próximo slide ilustra um exemplo de Realm que seria adequado para o OpenLDAP em configuração padrão, utilizando o schema **inetOrgPerson**
- ▶ O Tomcat tenta se conectar (binding) 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





# Exemplo de JNDI Realm 2/2

```
▶<Realm
  className="org.apache.catalina.realm.JNDIRealm"

  connectionURL="ldap://localhost:389"

  userBase="ou=people,dc=mycompany,dc=com"
  userSearch="(uid={0})"

  roleBase="ou=groups,dc=mycompany,dc=com"
  roleName="cn"
  roleSearch="(member={0})"
/>
```





# Conclusão

- ▶ Lab 10.1. Armazenando senhas criptografadas
- ▶ Lab 10.2. Autenticando contra um banco de dados
- ▶ Lab 10.3. Autenticando contra um diretório LDAP





# **Servidores de Aplicações Java EE usando Tomcat**

Capítulo 11

**Logs de Depuração e de Acesso**





# Objetivo

- ▶ Nesta aula, seremos apresentados às configurações de logging do Tomcat e como elas devem ser utilizadas pelas aplicações
- ▶ **Tópicos:**
  - ▶ Configurações de logging da JVM
  - ▶ Logs de acesso





# 11.1. Logging no Tomcat

- ▶ O Tomcat utiliza uma versão modificada do Commons Logging do Jakarta
- ▶ Esta versão está amarrada à API de Log do Java SE 1.4
- ▶ Mas pode ser substituída por uma versão do Commons Logging amarrada ao Log4J





# A Importância do Logging

- ▶ A manutenção de um registro de atividades do servidor é fundamental para identificar e isolar problemas de segurança, instabilidade ou performance
- ▶ Por outro lado, logs de servidores web podem crescer rapidamente
- ▶ E misturam informações referentes a vários usuários e aplicações distintos, além das informações referentes ao próprio servidor de aplicações
- ▶ Então eles são maiores e mais difíceis de interpretar
- ▶ O próprio volume de E/S de log (fora a contenção) pode engargalar o servidor







## 11.2. Conceitos de Logging do Java SE

- ▶ **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
- ▶ **Nível** (*level, 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
- ▶ **Saída** (*handler, appender*) indica o local onde o log será registrado, em geral um arquivo texto
- ▶ **Formatador** (*formatter, layout*) diz como uma saída deve formatar as informações recebidas





# Categorias x Saídas x Níveis

- ▶ Em geral categorias são nomeadas conforme o nome da classe Java
- ▶ Várias categorias podem ser vinculada a uma mesma saída, e uma mesma categoria pode ser vinculada a várias saídas diferentes
- ▶ 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
- ▶ Cada saída é associada a um único formatador





# Níveis de Log

- ▶ SEVERE – erros graves, dos quais a aplicação não pode se recuperar
- ▶ WARNING – erros recuperáveis ou ocorrências “suspeitas”
- ▶ INFO – eventos normais da execução da aplicação
- ▶ FINE – acompanhamento da execução normal da aplicação
- ▶ FINEST – informações para depuração da aplicação (valores intermediários)
- ▶ Os níveis são cumulativos “para cima” / menor detalhamento (INFO inclui WARNING e SEVERE)





## 11.2. Por que programar para a API de Logging

- ▶ Os logs gerados pelo servidor de aplicações e frameworks não são suficientes para identificar erros de aplicação
- ▶ Que são os mais comuns (e graves) na prática
- ▶ Não é possível simular concorrência e carga no ambiente de desenvolvedor
- ▶ Nem é possível seguir threads paralelas em um depurador passo-a-passo
- ▶ Toda aplicação deveria incluir comandos de logging desde o início





# Pecados graves (mas infelizmente comuns)

- ▶ Coisas que os programadores fazem que limitam ou mesmo anulam a utilidade dos logs do servidor de aplicações
- ▶ Usar **System.out.println**
- ▶ Chamar **Exception.printStackTrace**
- ▶ **try..catch** vazio





# Bom uso de logging

```
▶ public void doGet(...) ...{  
  
    log.info("Requisitada lista de contatos");  
    DataSource ds = null;  
    try {  
        log.fine("Buscando datasource");  
        InitialContext ctx = new InitialContext();  
        ds = (DataSource)ctx.lookup(  
            "java:comp/env/jdbc/ContatosLocal");  
    }  
    catch (NamingException ex) {  
        log.log(Level.SEVERE,  
            "Erro na busca JNDI", ex);  
        throw new ServletException(ex);  
    }  
    ...  
}
```





## 11.3. O Juli do Tomcat

- ▶ Extensão do logging do Java SE para flexibilizar a geração de saídas
- ▶ O JULI permite um prefixar os nomes de classe dos handlers, gerando assim várias saídas do mesmo tipo
- ▶ A configuração padrão para a Logging API está na pasta *conf*, no arquivo *logging.properties*
- ▶ Recomenda-se que pacotes WAR não tragam suas próprias configurações de logging (nem JARs de bibliotecas de logging)





# Sintaxe do logging.properties

- ▶ Primeiro são relacionadas todas as saídas que serão configuradas e quais delas serão utilizadas para a categoria raiz
- ▶ 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
- ▶ handlers = 1catalina.org.apache.juli.FileHandler,  
java.util.logging.ConsoleHandler







# Sintaxe do logging.properties

- ▶ Em seguida, cada saída é configurada
- ▶ `2localhost.org.apache.juli.FileHandler.level = FINE`
- ▶ `2localhost.org.apache.juli.FileHandler.directory = ${catalina.base}/logs`
- ▶ `2localhost.org.apache.juli.FileHandler.prefix = localhost.`
- ▶ O nome de uma saída tem que iniciar por um dígito!





# Sintaxe do logging.properties

- ▶ Por fim as categorias são configuradas com seus níveis e opcionalmente associadas a saídas
- ▶ `org.apache.catalina.core.ContainerBase.[Catalina].[localhost].level = INFO`
- ▶ `org.apache.catalina.core.ContainerBase.[Catalina].[localhost].handlers = 2localhost.org.apache.juli.FileHandler`
- ▶ Se uma categoria tem sua própria saída, suas mensagens **não são** replicadas na saída da categoria raiz!





## 11.3.1. Arquivos de log na configuração default

- ▶ A configuração padrão gera os arquivos, na pasta *logs*, rodacionados diariamente:
  - ▶ *catalina.<data>.log* contém as mensagens do Tomcat como um todo, incluindo aplicações
  - ▶ *localhost.<data>.log* mensagens específicas para o host “localhost”
  - ▶ *manager.<data>.log* mensagens do Manager
  - ▶ *catalina.out* contém a saída padrão e de erros da JVM, além de uma duplicata das mensagens do *catalina<data>.log* mas **não é rotacionado**
- ▶ Somente os dois primeiros são realmente úteis para o desenvolvedor





# Categorias do Tomcat

- ▶ O Tomcat nomeia suas categorias de log com o prefixo **org.apache.catalina.core.ContainerBase**
- ▶ E depois dele podem ser especificados **[Engine].[Host].[Contexto]**
- ▶ Infelizmente o log de um host não inclui as mensagens dos seus contextos
- ▶ E o log de um contexto também não inclui as mensagens das aplicações dentro dele
- ▶ Estes logs servem apenas para a depuração do próprio Tomcat, algo que o administrador dificilmente vai fazer





## 11.4. Logs da Aplicação

- ▶ É fácil usar o modelo fornecido pelo *logging.properties* para gerar um arquivo de log separado para uma aplicação web
- ▶ O processo consiste em:
  - ▶ Acrescentar um novo handler na relação no início do arquivo
  - ▶ Configurar este handler informando o nível de severidade das mensagens registradas e o prefixo do nome do arquivo
  - ▶ Vincular o handler a uma categoria nomeada conforme o pacote Java (**package**) raiz da aplicação





# Definindo um log separado para uma aplicação

- ▶ Pacote “exemplo” gerando o arquivo de log “contatos” para depuração:
- ▶ `handlers = 1catalina.org.apache.juli.FileHandler,...,  
6contatos.org.apache.juli.FileHandler, java.util.logging.ConsoleHandler`
- ▶ `6contatos.org.apache.juli.FileHandler.level = FINE`
- ▶ `6contatos.org.apache.juli.FileHandler.directory = ${catalina.base}/logs`
- ▶ `6contatos.org.apache.juli.FileHandler.prefix = contatos.`
- ▶ `exemplo.level = FINE`
- ▶ `exemplo.handlers = 6contatos.org.apache.juli.FileHandler`





## 11.4. Configuração para produção

- ▶ O objetivo é minimizar E/S e facilitar o acompanhamento de erros pelo administrador
- ▶ Aumente o nível da categoria raiz e das saídas do Tomcat para SEVERE ou WARNING
- ▶ Elimine as saídas do localhost e manager
- ▶ Gere um log separado para cada aplicações em nível INFO, mas envie **também** para o log do Tomcat
- ▶ Aumente o nível de log de aplicações (ou módulos) específicos conforme a necessidade





# Para não perder as mensagens de (auto-)deploy

- ▶ As mensagens de deployment e undeployment são geradas em nível INFO e se você usa auto-deploy vai querer mantê-las no log:
- ▶ `1catalina.org.apache.juli.FileHandler.level = INFO`
- ▶ `java.util.logging.ConsoleHandler.level = INFO`
- ▶ `.level = WARNING`
- ▶ `org.apache.catalina.startup.HostConfig.level = INFO`







## 11.5. 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
- ▶ Os logs abordados anteriormente informam sobre o funcionamento interno do próprio Tomcat ou das aplicações
- ▶ Logs de acesso registram páginas visitadas pelos usuários, permitindo gerar estatísticas de utilização e tempo de resposta para o site ou aplicações isoladas





# Válvula de Log de acesso

- ▶ Basta descomentar no **server.xml**
- ▶ 

```
<Valve className="org.apache.catalina.valves.  
AccessLogValve"  
directory="logs" prefix="localhost_access_log."  
suffix=".txt" pattern="common"  
resolveHosts="false"/>
```
- ▶ O padrão “common” é igual ao do Apache HTTPd e reconhecido por analisadores como WebAlizer e Awstats
- ▶ É possível gerar padrões customizados com informações úteis que não existiriam em um Apache





# Conclusão

- ▶ Lab 1. Aplicação Usando API de login
- ▶ Lab 2. Configuração de Log para produção
- ▶ Lab 3. Gerando logs de acesso





# **Servidores de Aplicações Java EE usando Tomcat**

## **Capítulo 12** **Configurações de Rede**





# Objetivo

- ▶ 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





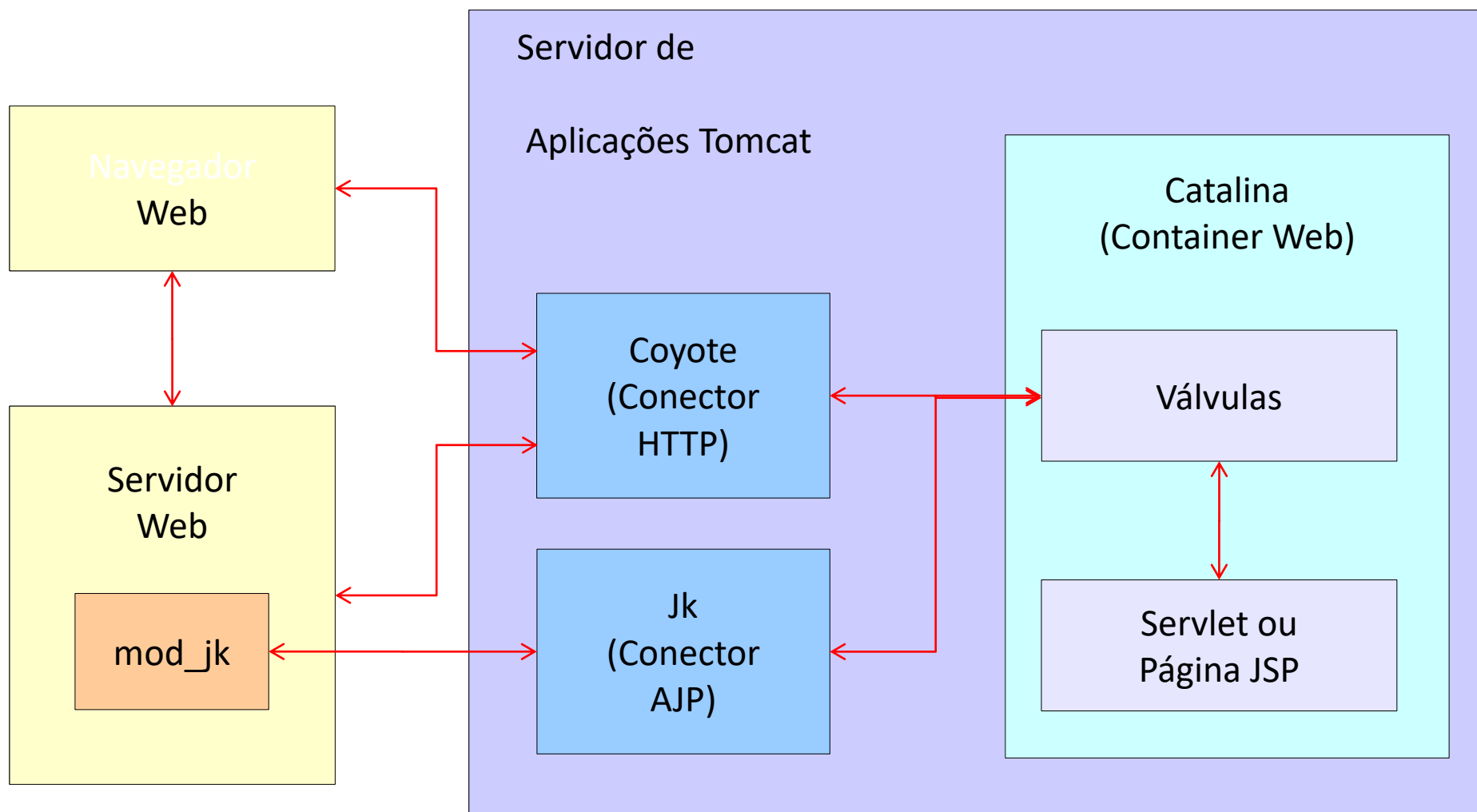
# Conectores e válvulas

- ▶ **Conectores** são responsáveis por receber conexões de redes e repassar os comandos recebidos para o container web (Catalina)
- ▶ O Tomcat pode ser utilizado diretamente como um servidor web porque ele possui um conector que entende o protocolo HTTP
- ▶ Quando existe outro servidor web entre o Tomcat e o navegador do usuário, o Conector Jk permite comunicação otimizada usando o protocolo AJP
- ▶ **Válvulas** recebem as requisições antes das aplicações (Servlets) e podem interferir no atendimento como quiserem





# Fluxo de Requisições HTTP no Tomcat





## 12.1.1. Configuração default do Coyote

- ▶ Apresenta apenas uns poucos atributos, há vários outros que podem ser acrescentados
- ▶ (Veja o manual do Tomcat)
- ▶ `<Service name="Catalina">`
- ▶ `<Connector port="8080" protocol="HTTP/1.1"`  
`connectionTimeout="20000"`  
`redirectPort="8443" />`
- ▶ `<Engine name="Catalina"`  
`defaultHost="localhost">`







## 12.1.2. Rodando o Tomcat na porta 80

- ▶ Em Windows, basta mudar a porta no conector
- ▶ No Unix, portas  $< 1024$  só podem ser abertas pelo root
- ▶ Mas não se recomenda processar requisições remotas como root
- ▶ Servidores nativos mudam de identidade logo após abrir a porta (e antes de aceitar qualquer pacote)
- ▶ A JVM não tem a operação de mudar de identidade
- ▶ Então deve-se contornar usando um firewall (NAT) ou redirecionador de portas





## 12.2. Firewall Interno do Tomcat

- ▶ As válvulas **RemoteAddrValve** e **RemoteHostValve** permitem definir regras de firewall em um engine, host ou contexto
- ▶ Por exemplo, para aceitar apenas hosts do domínio empresa.com.br
  - ▶ `<Valve className="org.apache.catalina.valves.RemoteHostValve" allow="*.empresa\.com\.br"/>`
- ▶ Ou então, para aceitar apenas requisições da rede local da empresa, 192.168.0.0/24
  - ▶ `<Valve className="org.apache.catalina.valves.RemoteAddrValve" allow="192\.168\.0\.\."/>`





# Regras de Host e Endereço IP

- ▶ Note que os valores dos atributos **allow** e **deny** das válvulas de firewall do Tomcat utilizam expressões regulares do pacote Jakarta Regexp <http://jakarta.apache.org/regexp/>
- ▶ Em geral elas seguem mesma sintaxe do grep e Perl
- ▶ Podem ser especificadas várias expressões, separadas por vírgulas
- ▶ Se um dos atributos for omitido, assume-se a expressão regular ".\*" que aceita qualquer coisa





## 12.3. Conexões SSL no Tomcat

- ▶ O Tomcat suporta, por meio do JSSE (Java Secure Sockets Extensions) conexões encriptadas nos padrões SSL e TLS, ou seja, conexões **https**:
- ▶ O JSSE é padrão no JavaSE desde o 1.4
- ▶ Se existir um servidor web nativo atuando como *front-end*, ele tem que processar o SSL e o Tomcat não precisa e nem pode fazer isso





# Ativando o Conector SSL

- ▶ O conector que aceita conexões criptografadas está comentado no *server.xml* padrão do Tomcat, basta descomenta-lo:
- ▶ 

```
<Connector port="8443"  
  scheme="https" secure="true"  
  keystorePass="secreto"  
  keystoreFile="conf/keystore"  
  clientAuth="false" sslProtocol="TLS"/>
```
- ▶ Ajuste os atributos destacados ao seu arquivo de keystore





## 12.3.1. Gerando Certificados com o **keytool**

- ▶ O JavaSE fornece o **keytool** para manipular keystores e certificados digitais
- ▶ Para gerar um certificado auto-assinado no arquivo *keystore* no diretório corrente:
  - ▶ `$ keytool -keystore keystore -genkey -alias tomcat -keyalg RSA`
  - ▶ Tome nota da senha fornecida
  - ▶ Forneça os dados sobre sua empresa e localização, que serão registrados no certificado
  - ▶ Use a mesma senha para o keystore (antes de perguntar pelos dados pessoais) e depois para a chave do Tomcat (depois de pedir pelos dados pessoais)





# Certificados Auto-Assinados

- ▶ Sites de produção utilizam um certificado adquirido em uma empresa reconhecida por padrão pelos navegadores, como a Verisign
- ▶ Para testes e Intranets, é possível gerar um certificado “auto-assinado”, que traz as mesmas garantias de segurança que um certificado “oficial”
- ▶ O que o certificado “oficial” garante é que o usuário está acessando o site real e não uma imitação
- ▶ Por isso o navegador irá alertar o usuário quanto a certificados auto-assinados





## 12.3.2. Segurança Declarativa x SSL

- ▶ Caso seja definido um **<web-resource-collection>** exigindo conexões encriptadas (CONFIDENTIAL), 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
- ▶ 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







# Restrição de acesso exigindo SSL

- ▶ `<security-constraint>`  
    `<web-resource-collection>`  
        ...  
    `</web-resource-collection>`  
    `<user-data-constraint>`  
        `<transport-guarantee>`  
            **CONFIDENTIAL**  
        `</transport-guarantee>`  
    `</user-data-constraint>`  
    `</security-constraint>`
- ▶ O `<user-data-constraint>` pode ser somado a um `<auth-constraint>` e/ou `<http-method>` no mesmo `<security-constraint>`





# SSL Automático

- ▶ O padrão Java EE diz que se uma página exigir SSL o container web tem que gerar automaticamente um redirect
- ▶ Para isso o conector HTTP necessita saber a porta correta do conector SSL ou do servidor front-end
- ▶ Por isso existe o atributo `redirectPort` nos conectores Coyote e Jk
  - ▶ `<Connector port="8080" protocol="HTTP/1.1"`  
    `connectionTimeout="20000"`  
    `redirectPort="8443" />`





## 12.4. Hosts Virtuais

- ▶ É a configuração onde um mesmo servidor web atende a requisições cujas URLs informem nomes de hosts diferentes
- ▶ Para o usuário, eles são indistinguíveis de hosts “reais”, hospedados por servidores independentes
- ▶ Podem ser configurados de duas formas:
  - ▶ Um mesmo servidor é configurado com vários endereços IP diferentes
  - ▶ Vários nomes DNS diferentes são configurados para o mesmo endereço IP





# Configuração de Hosts no Tomcat

- ▶ As duas formas de se configurar hosts virtuais são indiferentes para o Tomcat
- ▶ Nos dois casos, basta acrescentar um novo elemento **<Host>** ao *server.xml* e decidir que aplicações serão configuradas nele
- ▶ Cada host pode ser sua própria pasta para auto-deploy dada pelo atributo **appBase** e sua própria configuração para auto-deployment





## 12.4.1. Manager x hosts virtuais

- ▶ O Manager gerencia deployment em um único host
- ▶ (Embora a página de status seja para o Tomcat como um todo)
- ▶ Então, ou se acrescenta em cada **<Host>** um novo **<Context privileged="true">** apontando para o WAR do Manager...
- ▶ ... Ou não é possível usar o Manager para o host.





# Conclusão

- ▶ Lab 1. Tomcat como servidor web principal
- ▶ Lab 2. Restringindo acesso por IP
- ▶ Lab 3. Ativando suporte a https:
- ▶ Lab 4. Protegendo páginas com https:
- ▶ Lab 5. Adicionando um host virtual
- ▶ Questões de Revisão





# **Servidores de Aplicações Java EE usando Tomcat**

## **Capítulo 13** **Integração com Servidores Web Nativos**





# Objetivo

- ▶ 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







# Servidores Nativos x Tomcat: Fatos

- ▶ Situações onde um servidor nativo se torna necessário para complementar o Tomcat
  - ▶ Partes do site usando outras tecnologias, por exemplo CGI, PHP, ASP ou Cold Fusion
  - ▶ Uso de plug-ins de servidores web (módulos do Apache ou ISAPI)
  - ▶ Proxies reversos e aceleradores web
  - ▶ Servidor nativo na DMZ e Tomcat protegido na LAN
  - ▶ Distribuidores de carga para cluster





# Servidores Nativos x Tomcat: Mitos

- ▶ Situações onde acreditava-se que servidores web nativos tenham melhor performance que o Tomcat, mas hoje pode-se dizer que não:
  - ▶ 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
- ▶ Neste cenário o servidor nativo pode ter melhor performance:
  - ▶ Uso de aceleradores de hardware para criptografia





## 13.2. 0 mod\_jk

- ▶ É um plug-in de servidor web que acrescenta o suporte ao protocolo AJP
- ▶ Há versões para Apache, ISAPI e NSAPI
- ▶ Deve ser compilado em relação à versão específica e plataforma do servidor web
- ▶ É o upstream do mod\_proxy\_ajp do Apache (que não suporta outros servidores web)
- ▶ O mod\_jk2 foi abandonado, e o mod\_jk 1.2 é **mais recente** do que ele!





## 13.3. Instalando o mod\_jk

- ▶ Há três formas de se obter e instalar o mod\_jk em Linux
  1. Utilizando um pacote fornecido pela sua distribuição
  2. Utilizando binários pré-compilados fornecidos pela ASF
  3. Compilando você mesmo à partir dos fontes
- ▶ Em Linux, provavelmente será necessário fazer (3)





# Compilando o mod\_jk

- ▶ Visite *<http://tomcat.apache.org>* e baixe os fontes do mod\_Jk em “Tomcat Connectors”
- ▶ Por exemplo, *[tomcat-connectors-1.2.28-src.tar.gz](#)*
- ▶ Descompacte em sua pasta home
- ▶ Localize o executável **apxs**, que é parte do Apache; tome nota do caminho
- ▶ Na maioria das distribuições do Linux, será necessário instalar o pacote *httpd-devel* ou similar
- ▶ Daí pra frente, configure, make e su -c make install





# Compilando o mod\_jk

```
▶ $ cd ~/tomcat-connectors-*-src  
$ cd native  
$ ./configure --with-apxs=/usr/sbin/apxs  
$ make  
$ su -c "make install"
```

- ▶ Note que não é necessário recompilar o Apache
- ▶ Não é necessário nem mesmo ter os fontes do Apache
- ▶ Se a versão da distribuição, versão do apache e arquitetura de processador forem as mesmas, pode copiar o *mod\_jk.so* de um computador para o outro.





## 13.4. Configuração do mod\_jk

- ▶ Assumindo que seu *httpd.conf* inclui todos os arquivos em *conf.d*, como é o caso do Fedora
- ▶ O arquivo */etc/httpd/conf.d/mod\_jk.conf* configura o mod\_jk em si (em relação ao Apache)
  - ▶  
LoadModule jk\_module modules/mod\_jk.so  
JkWorkersFile /etc/httpd/conf.d/workers.properties  
JkLogFile /var/log/httpd/mod\_jk.log  
JkLogLevel error  
JkMount /hoje-valve no0  
JkMount /hoje-valve/\* no0
- ▶ O nome no final do **JkMount** (no0) é o nome de um **worker**, que é uma instância do Tomcat





# Configuração de workers

- ▶ Agora é necessário fornecer o arquivo de configuração de workers indicado pela diretiva **JkWorkersFile**, no caso */etc/httpd/conf.d/workers.properties*:
  - ▶ worker.list=**no0**  
worker.**no0**.type=ajp13  
worker.**no0**.host=127.0.0.1  
worker.**no0**.port=8009
- ▶ Observe o nome do worker (no0)
- ▶ Inicie (ou reinicie) o Apache
  - ▶ # service httpd restart
- ▶ Se o SELinux atrapalhar, desative (mas depois aprenda a configura-lo!)







# Testando o mod\_jk

- ▶ Garanta que a aplicação indicada por **JkMoint** (hoje-valve) esteja instalada no Tomcat, se necessário use o Manager, acessando o Tomcat pela porta 8080
- ▶ Inicie o Tomcat, se ele não estiver rodando
- ▶ Acesse a aplicação pela porta 8080, apenas para ter certeza de que ela esteja disponível
- ▶ Acesse a URL `http://127.0.0.1/hoje-valve`





# Se Algo Der Errado

- ▶ 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 linha no arquivo *mod\_jk.conf* (ou *httpd.conf*)
  - ▶ JkLogLevel **debug**
- ▶ Verifique o log do mod\_jk (*/var/log/httpd/access\_log*)
- ▶ E verifique se o SELinux está desabilitado





## 13.4.1. Dividindo Tarefas Entre o Apache e o Tomcat

- ▶ A configuração que fizemos repassa para o Tomcat todas as requisições direcionadas para uma determinada aplicação web no Tomcat
- ▶ 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
- ▶ Note que, se isto for feito, eventuais configurações de controle de acesso terão que ser duplicadas com os recursos do Apache





# Configurações no Apache

- ▶ Rode a aplicação como um pacote aberto (ou então use o diretório gerado no deploy de um WAR fechado)
- ▶ Use o próprio *mod\_jk.conf* para definir um **Alias** do Apache, que é um diretório de páginas
  - ▶ Alias /paginas /var/lib/tomcat5/webapps/paginas
  - ▶ (sem quebra de linha)
- ▶ Redirecione para o Tomcat apenas as páginas JSP
  - ▶ JkMount /pacotes/\*.jsp no0
- ▶ E reinicie o Apache
  - ▶ # service httpd restart





# Configuração por Exceção

- ▶ Usar extensões de arquivos na diretiva **JkMount** não é muito prático para Servlets por isso veremos uma outra alternativa
- ▶ É possível configurar: “Redirecione tudo EXCETO:”
  - ▶ JkMount /exemplo5.1 no0
  - JkMount /exemplo5.1/\* no0
  - JkUnmount /exemplo5.1/\*.html no0
  - JkUnmount /exemplo5.1/\*.gif no0
- ▶ Outra forma de configurar o “EXCETO”
  - ▶ JkMount /exemplo5.1 no0
  - JkMount /exemplo5.1/\* no0
  - JkUnmount /exemplo5.1/html/\* no0
  - JkUnmount /exemplo5.1/imagens/\* no0





## 13.5. Integração Segurança e Logging

- ▶ O uso do Apache como front-end para o Tomcat oferece várias oportunidades de otimização de desempenho (ex: mod\_proxy), 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
- ▶ Embora o Apache reconheça autenticação HTTP, ele não reconhece FORM
- ▶ Ambos os logs de erros e de acesso (do Apache e do Tomcat) são necessários





# Conclusão

- ▶ Lab 1. Integração Apache com Tomcat
- ▶ Lab 2. Servindo arquivos estáticos no Apache
- ▶ Questões de Revisão





# **Servidores de Aplicações Java EE usando Tomcat**

Capítulo 14

**Escalabilidade com Clusters Tomcat**







# Objetivo

- ▶ Neste capítulo vemos como configurar clusters Tomcat para balanceamento de carga, isto é, clusters que trazem escalabilidade.
- ▶ **Tópicos:**
  - ▶ Conceitos de clusters Java EE
  - ▶ Rodando múltiplas instâncias do Tomcat
  - ▶ Configurando o mod\_jk como balanceador
  - ▶ Página de status do cluster





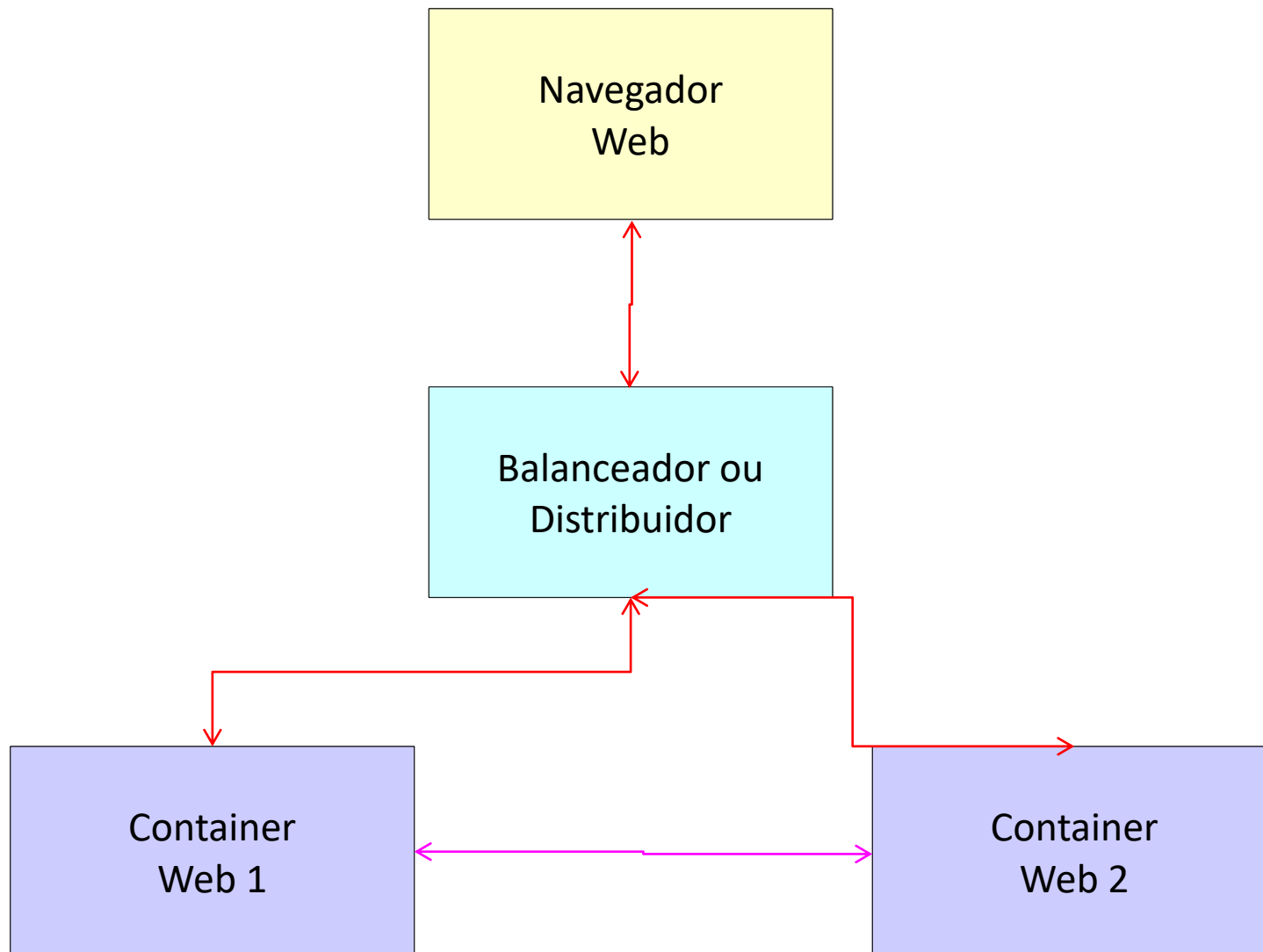
## 14.1. Conceitos de Cluster Java EE

- ▶ 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, enquanto que outras plataformas ele exige programação cuidadosa e especializada
- ▶ No caso de aplicações web, a natureza do protocolo HTTP ao mesmo tempo facilita e impõe restrições





# Cluster Web Java EE





# Distribuição de Carga x Tolerância à Falhas

- ▶ O cluster pode se limitar a distribuir a carga entre os vários nós
- ▶ Por exemplo, um servidor atingiu o limite de sua capacidade, então acrescenta-se um segundo servidor
- ▶ Ou então o cluster pode fornecer também tolerância a falhas
- ▶ Neste caso, os usuários que eram atendidos por um nó falho serão automaticamente redirecionados para um dos nós sobreviventes
- ▶ Não é trivial oferecer a tolerância a falhas sem perder atividades “em progresso”





## 14.1.1. Estado da Aplicação no HTTP

- ▶ O HTTP é um protocolo sem estado, então qualquer requisição poderia ser entregue a qualquer nó do cluster
- ▶ Um mesmo usuário poderia ser a cada *página* atendido por um servidor diferente
- ▶ Entretanto, *aplicações* web reais utilizam artifícios como sessões HTTP para manter o estado de um usuário particular (por exemplo, uma cesta de compras)
- ▶ Este estado tem que ser replicado entre os nós do cluster para que haja tolerância a falhas





## 14.1.2. Balanceador ou Distribuidor

- ▶ É o componente que recebe as requisições do navegador e as encaminha para um dos nós do cluster
- ▶ Cria para o usuário no navegador a ilusão de que existe apenas um único servidor em vez de vários containers web independentes
- ▶ Há várias estratégias para sua implementação:
  - ▶ Switches e roteadores especializados - “layer 7”
  - ▶ Múltiplos endereços IP para um mesmo nome DNS (DNS round-robin)
  - ▶ Proxy reverso ou servidor *font-end*





## 14.1.3. Arquitetura de Cluster Web do Java EE

- ▶ Sessões HTTP são implementadas por meio de um cookie “identificador de sessão” conforme manda a especificação Java EE
- ▶ 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 e “amarrar” as sessões ao mesmo nó
- ▶ Para facilitar o balanceador, o Tomcat insere no cookie um identificador do nó





## 14.2. O mod\_jk como balanceador

- ▶ O mod\_jk do Apache pode ser configurado para atuar como um balanceador de carga entre vários servidores Tomcat
- ▶ Ele inclusive é capaz de lidar com servidores de capacidades diferentes e distribuir a carga (usuários / sessões HTTP) proporcionalmente
- ▶ Os servidores não precisam nem mesmo usar o mesmo SO!







# 0 worker tipo “lb”

- ▶ Edite o arquivo *workers.properties*:
  - ▶ **worker.list=cluster**
  - worker.cluster.type=lb**
  - worker.cluster.balance\_workers=no1,no2**
  - worker.cluster.sticky\_session=true**
  - worker.no1.type=ajp13**
  - worker.no1.host=127.0.0.1**
  - worker.no1.port=8109**
  - worker.no1.lbfactor=1**
  - worker.no2.type=ajp13**
  - worker.no2.host=127.0.0.1**
  - worker.no2.port=8209**
  - worker.no2.lbfactor=1**
- ▶ Foram definidos dois nós, vinculados a um worker “cluster”, que deve ser referenciado pelas diretivas **JkMount** em **mod\_jk.conf**





# Configurando o Tomcat

- ▶ É necessário informar ao Tomcat que há um balanceador na frente
- ▶ Para tal, edite o *server.xml* de cada um dos nós do cluster, acrescentando no elemento **<Engine>** o atributo **jvmRoute**:
  - ▶ `<Engine name="Catalina" defaultHost="localhost" jvmRoute="no1">`
  - ▶ `<Engine name="Catalina" defaultHost="localhost" jvmRoute="no2">`
- ▶ O valor de **jvmRoute** tem que ser o nome do **worker** correspondente!





# Se Algo Der Errado

- ▶ Verifique que os dois servidores Tomcat estão no ar, acessando-os diretamente pelas portas 8180 e 8280
- ▶ Confira os arquivos de configuração **server.xml** em cada nó, atentando para os três números de porta (8105, 8180 e 8109; 8205, 8280 e 8209) e para o **jvmRoute** no Engine
- ▶ Confira também os arquivos de configuração do `mod_jk`, em especial as configurações do balanceador e de cada nó no *workers.properties*
- ▶ Verifique os logs de cada nó Tomcat e do Apache em busca de mensagens de erro





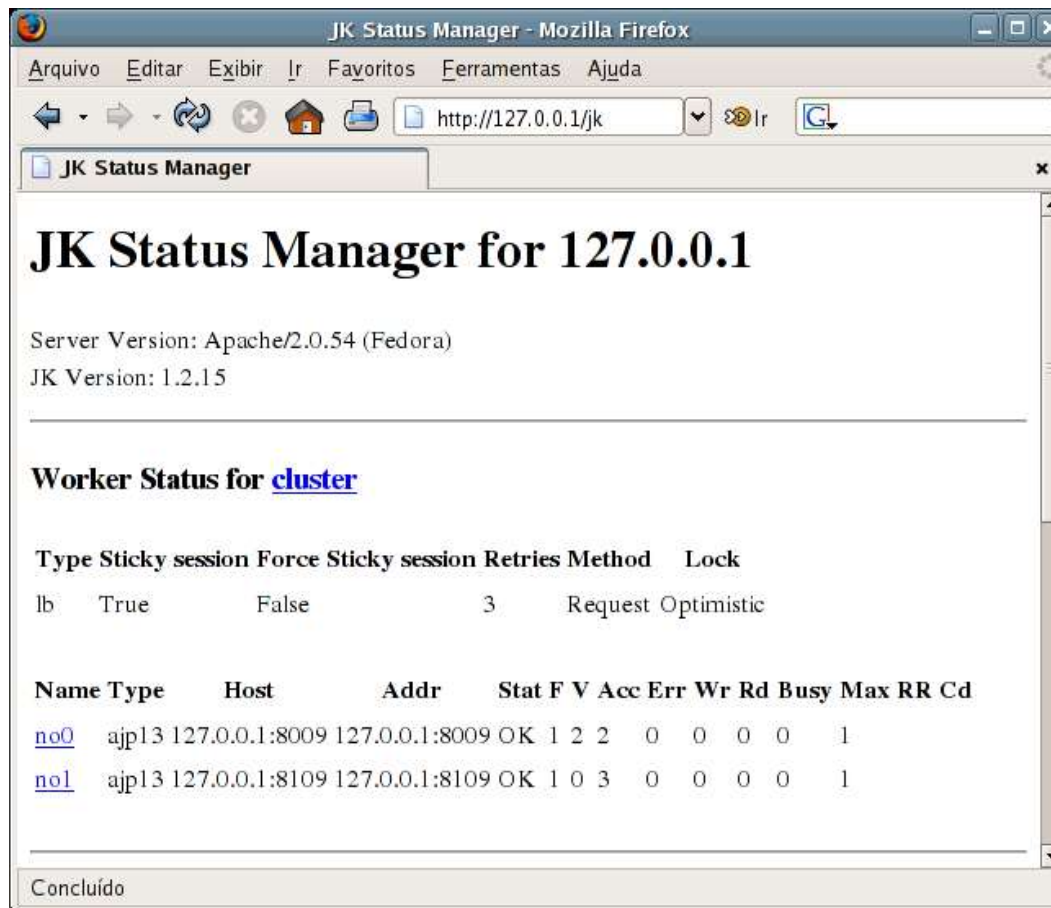
## 14.2.1. Status do Cluster 1/2

- ▶ 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 o “status” do mod\_jk, editando o *mod\_jk.conf*
  - ▶ JkMount /jk status
- ▶ Em seguida, ative a geração da página de status no *worker.properties*
  - ▶ worker.list=cluster,status  
worker.status.type=status  
...
- ▶ Reinicie o Apache (mas não o Tomcat!)



# Página de Status do Jk

- ▶ Veja a página de status acessando a URL <http://127.0.0.1/jk>



The screenshot shows a Mozilla Firefox browser window titled "JK Status Manager - Mozilla Firefox". The address bar displays "http://127.0.0.1/jk". The page content is as follows:

## JK Status Manager for 127.0.0.1

Server Version: Apache/2.0.54 (Fedora)  
JK Version: 1.2.15

---

### Worker Status for [cluster](#)

Type	Sticky session	Force Sticky session	Retries	Method	Lock
lb	True	False	3	Request	Optimistic

Name	Type	Host	Addr	Stat	F	V	Acc	Err	Wr	Rd	Busy	Max	RR	Cd
<a href="#">no0</a>	ajp13	127.0.0.1:8009	127.0.0.1:8009	OK	1	2	2	0	0	0	0	1		
<a href="#">no1</a>	ajp13	127.0.0.1:8109	127.0.0.1:8109	OK	1	0	3	0	0	0	0	1		

Concluído



## 14.3. Configurando instâncias adicionais do Tomcat

- ▶ Na instalação manual, é fácil: basta instalar (unzipar) em outro diretório e mudar as portas
- ▶ Na instalação JPackage, dá mais trabalho manter a aderência ao LSB e permissões
- ▶ (Mas fica mais seguro e organizado)
- ▶ Os pacotes do Fedora em teoria suportam rodar “fácil” várias instâncias, mas os scripts estão com bugs
- ▶ Por isso o laboratório fornece um script alternativo para “clonar” instâncias





## 14.3.1. Vantagens de várias instâncias

- ▶ Segregar permissões de administração ou criticidade
- ▶ Aplicações legadas com problemas de conformidade com o Java EE (e que por isso não funcionam em versões mais novas)
- ▶ Poder derrubar uma instância para atualização, enquanto que a outra mantém o serviço ao usuário
- ▶ NÃO PRECISA para contornar limites de escala do SO ou da JVM (exceto em Windows ou Unix de 32-bits)





# Conclusão

- ▶ Lab 1. Duas instâncias do Tomcat lado-a-lado
- ▶ Lab 2. Cluster para escalabilidade
- ▶ Lab 3. Status do cluster
- ▶ Lab 4. Desligando a afinidade de sessão
- ▶ Questões de Revisão







# **Servidores de Aplicações Java EE usando Tomcat**

Capítulo 15

**Cluster para HA – Alta Disponibilidade**





# Objetivo

- ▶ 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





# 15.1. Conceitos de Cluster 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
- ▶ Para que isto ocorra, as informações armazenadas na sessão HTTP devem ser replicadas entre todos os nós do clusters
- ▶ Desta forma, qualquer nó pode dar continuidade a atividades iniciadas por outro nó





## 15.2. Aplicações “cluster-aware”

- ▶ Pelo Java EE, simplesmente funciona...
- ▶ ... Desde que a aplicação use de forma correta caches, singletons e etc
- ▶ Coisas que funcionam ok em servidores isolados falham horivelmente em cluster (mas não o contrário!)
- ▶ Por isso o *web.xml* deve dizer se a aplicação foi feita (ou homologada) para cluster incluindo o elemento **<distributable/>**





## 15.1.2. Erros de Aplicação a Evitar

- ▶ O suporte a clustering web do JavaEE é transparente, desde que o programador não cometa alguns erros básicos:
  - ▶ Todos os objetos armazenados na sessão HTTP devem se serializáveis
  - ▶ Uma requisição HTTP deve corresponder a uma única transação no banco de dados
  - ▶ Uma transação no banco de dados deve ocorrer dentro de uma única transação 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





## 15.2. Arquitetura de Cluster HA do Tomcat

- ▶ O Tomcat utiliza *multicast IP* para identificar os nós do cluster que ainda estão ativos
- ▶ Na verdade cada nó anuncia sua presença aos demais utilizando os multicasts
- ▶ As informações em si são transferidas por meio de conexões TCP regulares para os demais nós existentes
- ▶ Se um nó fica muito tempo sem anunciar sua presença, os demais nós o consideram como inativo e deixam de lhe enviar informações





# Replicação de sessão HTTP

- ▶ Havendo vários clusters independentes na mesma rede, cada cluster deve ser configurado com um endereço IP de multicast diferente
- ▶ 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





## 15.3. 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
- ▶ A configuração é mínima e não expõe vários sub-elementos e propriedades que o administrador pode precisar modificar
- ▶ Mas um modelo de configuração completo é gigante, com muita coisa que dificilmente será alterada
- ▶ Os parâmetros de rede estão em **<Membership>** e **<Receiver>**







# Configurações de rede da replicação

- ▶ <Host ...>
- ▶ <Cluster ...>
- ▶ <Manager ... />
- ▶ <Channel ... >
- ▶ <Membership ...  
    **bind="0.0.0.0" address="228.0.0.4"**  
    **port="45564"**  
    frequency="500" dropTime="3000"/>
- ▶ <Receiver ...  
    **address="auto" port="4000" autoBind="100"**  
    selectorTimeout="5000" maxThreads="6"/>
- ▶ <Sender ... >
- ▶ <Interceptor ... >





## 15.3.1. Vários clusters na mesma rede

- ▶ Como o cluster é baseado em multicast e por isso “auto-discovery” todos os Tomcats que se enxergarem irão formar um único cluster
- ▶ Para evitar isso, cada cluster deve ter um endereço de multicast diferente, alterando o atributo **address** de **<Membership>**
- ▶ Em produção, configura-se uma placa de rede (e switch) exclusivos para a replicação, por isso o atributo **bind** de **<Membership>**
- ▶ Também pode ser necessário mudar o **address** de **<Receiver>** (para a placa intra-cluster)





## 15.3.2. 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, o que é feito como parte das configurações de roteamento estático
  - ▶ `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 eth0`





## 15.3.3. Cluster x Manager

- ▶ O Manager e não tem conhecimento do cluster, portanto eles tem que ser ativados e acessados de forma independente em cada nó
- ▶ Por isso nossa simulação não eliminou o conector HTTP
- ▶ O deployment das aplicações, além das configurações de hosts virtuais, **<Resource>** e **<Realm>** tem que replicados manualmente em cada os nós





# Conclusão

- ▶ Lab 1.Cluster com HA
- ▶ Questões de Revisão





# Servidores de Aplicações Java EE usando Tomcat

## Capítulo 16 Monitoração e Tuning





# Objetivo

- ▶ Este capítulo é uma introdução ao monitoramento e tuning do servidor Tomcat para obter maior performance e estabilidade
- ▶ **Tópicos:**
  - ▶ Modelo de performance para servidores de aplicação Java EE
  - ▶ Tuning de threads e conexões
  - ▶ Tuning da JVM
  - ▶ Monitoração pelo Manager
  - ▶ Monitoração JMX
  - ▶ Dicas de troubleshooting





# Modelo de Performance Para Aplicações Java EE

- ▶ Não é possível dimensionar um servidor de aplicações de forma independente das aplicações deployadas
- ▶ Erros de codificação e projeto podem se manifestar apenas sob um volume suficiente (e desconhecido) de carga
- ▶ Os principais elementos de tuning do servidor formam um pipeline de processamento:
  - ▶ Os threads de entrada, gerados pelos conectores;
  - ▶ O uso de processador e memória para a execução destes threads);
  - ▶ Conexões para o banco de dados, gerenciadas pelos DataSources.







# Efeito “funil”

- ▶ A utilização de recursos vai diminuindo à medida em que se avança no pipeline:
  - ▶ Os usuários não geram requisições HTTP continuamente, mas gastam tempo lendo e digitando;  
(menos conexões que usuários)
  - ▶ Os threads não usam o banco de dados o tempo todo, mas gastam tempo validando dados de entrada e gerando HTML de saída.  
(menos conexões de BD do que threads)





## 16.2. Monitorando o Tomcat via Manager

- ▶ O Manager apresenta informações sumárias de uso de memória (heap da JVM) e de threads dos conectores na página de status
- ▶ Também permite acompanhar quantidade de usuários (sessões) e memória por usuário (atributos de sessão) na página de aplicações
- ▶ As informações visíveis via Manager, e várias que não o são, podem ser monitoradas via JXM (Java Management Interface)





## 12.6.1 Tuning do Conector x Balanceador

- ▶ Os conectores do Tomcat usam um pool de threads para atender a requisições
- ▶ Os pools maximizam o aproveitamento de CPU e memória: salvam tempo de alocar e liberar recursos, e reduzem o trabalho do coletor de lixo da JVM
- ▶ Além de melhorar o tempo de resposta
- ▶ Como o tempo de processador é muito inferior ao tempo de disco, rede e dos humanos, ter um thread por conexão (ou por usuário / sessão HTTP) seria um desperdício de recursos





# Consultando o Uso de Threads com o Manager

- ▶ O Manager exibe os limites do pool de threads: máximo (Max), reserva (spare), total (current) e ocupados (busy)
- ▶ Também fornece uma lista do que estão fazendo os threads ocupados

## http-8080

Max threads: 150 Min spare threads: 25 Max spare threads: 75 Current thread count: 25 Current thread busy: 2  
Max processing time: 0 ms Processing time: 0.0 s Request count: 0 Error count: 0 Bytes received: 0.00 MB Bytes sent: 0.00 MB

Stage	Time	B Sent	B Recv	Client	VHost	Request
S	194 ms	0 KB	0 KB	127.0.0.1	127.0.0.1	GET /manager/status HTTP/1.1
R	?	?	?	?	?	?

P: Parse and prepare request S: Service F: Finishing R: Ready K: Keepalive





# Conector x Balanceador

- ▶ 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 uma 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





## 16.2.2 Dimensionamento de DataSources

- ▶ O dimensionamento dos pools de conexões a bancos de dados (DataSources) é um pouco mais complicado porque:
  - ▶ O Manager não exibe informações sobre o status atual dos pools
  - ▶ Conexões a bancos de dados tendem a estar em uso por períodos mais longos do que threads, e dependem de recursos de rede
- ▶ Valem os mesmos princípios de dimensionamento (mínimo – **maxIdle**, **máximo** – **maxActive**) vistos para threads
- ▶ Estatísticas de uso podem ser obtidas via JMX





## 16.3. 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
- ▶ A página de status do servidor (no Manager) informa a memória em uso e a máxima no heap





# Consultando Uso de Memória com o Manager

- ▶ 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*)

Server Information			
Tomcat Version	JVM Version	JVM Vendor	OS Name
Apache Tomcat/5.5.16	1.5.0_05-b05	Sun Microsystems Inc.	Linux

## JVM

Free memory: 1.09 MB Total memory: 5.73 MB Max memory: 254.06 MB

## http-8080







# Parâmetros de Memória da JVM

- ▶ O comando **java -X** exibe as várias opções para tuning de memória e coleta de lixo da JVM
- ▶ As opções mais usadas são:
  - ▶ **-Xms<size>**  
Tamanho inicial do heap
  - ▶ **-Xmx<size>**  
Tamanho máximo do heap
  - ▶ **-Xss<size>**  
Tamanho da pilha ()





# Parâmetros de Memória da JVM

- ▶ O máximo padrão do heap é de 64Mb
- ▶ 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
- ▶ O comando abaixo exhibe a memória total (vsz) e em uso na RAM (rss) para os processos da JVM que rodam uma JVM
  - ▶ `$ ps axo pid,vsz,rss,cmd | grep java`
- ▶ Para estabelecer o máximo em 256Mb, use
  - ▶ `$ java -Xmx256M`





## 16.3.1. Passando opções para a JVM (JPackage)

- ▶ Embora seja possível modificar os scripts de início do Tomcat para inserir opções da JVM, é mais limpo usar a variável de ambiente **JAVA\_OPTS**
- ▶ Então, para iniciar o Tomcat com o heap aumentado, insira no *tomcat.conf*
  - ▶ `JAVA_OPTS="$JAVA_OPTS -Xmx256M"`
- ▶ E reinicie o servidor
- ▶ Confira pela página de status do Manager, o aumento do máximo para o heap





## 16.3.1. Passando opções para a JVM (manual)

- ▶ Embora seja possível modificar os scripts de início do Tomcat para inserir opções da JVM, é mais limpo usar a variável de ambiente **JAVA\_OPTS**
- ▶ Então, para iniciar o Tomcat com o heap aumentado:
  - ▶ `$ JAVA_OPTS=-Xmx256M ./startup.sh`
- ▶ Confira pela página de status do Manager, o aumento do máximo para o heap
- ▶ Outra opção é inserir, no início do *startup.sh*
  - ▶ `export JAVA_OPTS=-Xmx256M`





## 16.4. Tuning do Cluster

- ▶ Pode ser necessário aumentar a quantidade de threads de replicação:

```
▶ <Receiver className="org.apache.catalina.cluster.tcp.  
ReplicationListener"  
    tcpListenAddress="auto"  
    tcpListenPort="4000"  
    tcpSelectorTimeout="100"  
    tcpThreadCount="8"/>>
```

- ▶ Ou mudar o modo de replicação para síncrono:

```
▶ <Sender className="org.apache.catalina.cluster.tcp.  
ReplicationTransmitter"  
    replicationMode="synchronous"  
    ackTimeout="15000"/>
```





## 16.5. Monitoração via JMX

- ▶ Especificação da plataforma Java que permite a qualquer aplicação fornecer “objetos gerenciados”, os MBeans
- ▶ Define também uma forma de se localizar e obter informações de MBeans, ou até de executar ações sobre eles
- ▶ É obrigatório em versões mais novas do JavaEE, que também define conjuntos de MBeans padrão para os servidores de aplicações
- ▶ O mercado oferece vários consoles para gerenciamento e monitoração JMX, similar aos consoles SNMP
- ▶ Mbean seriam equivalentes aos MIBs do SNMP





# O JMX Proxy Servlet

- ▶ É um componente do Manager (indisponível para uso interativo) que permite executar operações sobre MBeans
- ▶ Útil especialmente pela facilidade de integração a sistemas de NMS como MRTG, BigBrother, CACIT, OpenView ou Tivoli
- ▶ Também pode ser usado dentro de shell scripts por meio dos utilitários **wget** ou **curl**
- ▶ Lembre sempre de codificar os caracteres especiais dentro do nome lógico de um MBean (como “:” e “=”) dentro das convenções de URLs para HTTP GET





# Monitoração do Tomcat via JMX

- ▶ O Tomcat deve ser iniciado com o agente JMX da JVM ativado
- ▶ Em modo local, basta definir uma propriedade de sistema
  - ▶ `$ JAVA_OPTS="-Dcom.sun.management.jmxremote" ./startup.sh`
- ▶ Em seguida, verifique qual o processo da JVM que roda o Tomcat e conecte a este processo usando o **jconsole** do JavaSE
  - ▶ `$ ps ax | grep java`  
22587 pts/5 Rl :03/usr/java/jdkX.X.X/bin/java ...  
`$ jconsole 22587`







# Nome de um MBean

- ▶ Segue a forma:
- ▶ **domínio:nome1=valor1,nome2=valor2,..**
- ▶ Onde podem haver vários partes **nome=valor**.
- ▶ Maiúsculas e minúsculas fazem diferença!





# Exemplos de MBeans

- ▶ Um Realm, definido dentro de um Contexto:
  - ▶ Catalina:type=Realm,path=/exemplo4.1,host=localhost
- ▶ Um usuário (que já foi autenticado por alguma aplicação):
  - ▶ Users:type=User,username="tomcat",database=UserDatabase
- ▶ Um Servlet:
  - ▶ Catalina:j2eeType=Servlet,name=contatos,WebModule=//localhost/exemplo3.3,J2EEApplication=none,J2EEServer=none





# Exemplos de MBeans

- ▶ Pool de Threads de um Conector
  - ▶ Catalina:type=ThreadPool,name=http-8080
- ▶ Um dos Threads de 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"





## 16.5.1. Usando o JConsole com o Tomcat

- ▶ O **jconsole** é parte do JDK da Sun (e do OpenJDK)
- ▶ O acesso JMX deve ser habilitado por opções da JVM:
- ▶ `JAVA_OPTS="$JAVA_OPTS -Dcom.sun.management.jmxremote"`
- ▶ A conexão é feita ao processo da JVM:
- ▶ 

```
$ ps ax | grep java
22587 pts/5 Rl :03/usr/lib/jvm/java-X.X.X/bin/java
...
org.apache.catalina.startup.Bootstrap start
$ jconsole 22587
```





## 16.5.2. Acesso Remoto JMX

- ▶ O **jconsole** também pode monitorar JVMs remotamente:
- ▶ `JAVA_OPTS="-Dcom.sun.management.jmxremote  
-Dcom.sun.management.jmxremote.port=5000  
-Dcom.sun.management.jmxremote.ssl=false  
-Dcom.sun.management.jmxremote.authenticate=false"`
- ▶ A conexão é feita para a porta especificada:
- ▶ `$ jconsole 127.0.0.1:5000`





# Sumário do JMX Console

J2SE 5.0 Monitoring & Management Console: 22587@localhost

Connection

Summary Memory Threads Classes MBeans VM

---

**Summary**

Uptime: 7 minutes      Process CPU time: 14,000 seconds  
Total compile time: 1,423 seconds

---

**Threads**

Live Threads: 48      Peak: 48  
Daemon threads: 45      Total started: 49

---

**Memory**

Current heap size: 7.435 kbytes      Committed memory: 10.340 kbytes  
Maximum heap size: 65.088 kbytes  
Objects pending for finalization: 0  
Garbage collector: Name = 'Copy', Collections = 246, Total time spent = 1,427 seconds  
Garbage collector: Name = 'MarkSweepCompact', Collections = 0, Total time spent = 0,000 seconds

---

**Classes**

Current classes loaded: 2.456      Total classes unloaded: 13  
Total classes loaded: 2.469

---

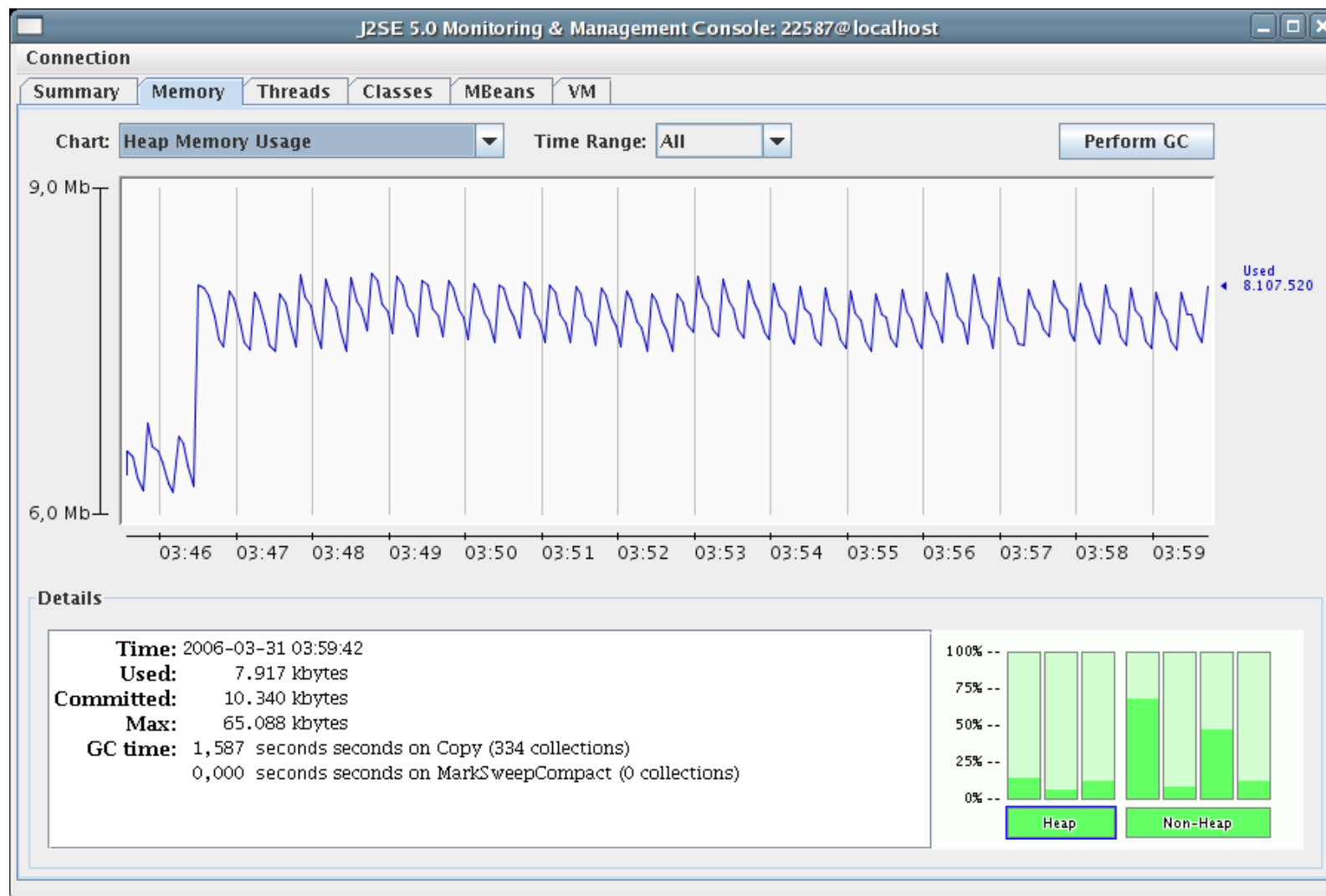
**Operating System**

Total physical memory: 1.035.464 kbytes      Free physical memory: 24.176 kbytes  
Committed virtual memory: 295.720 kbytes





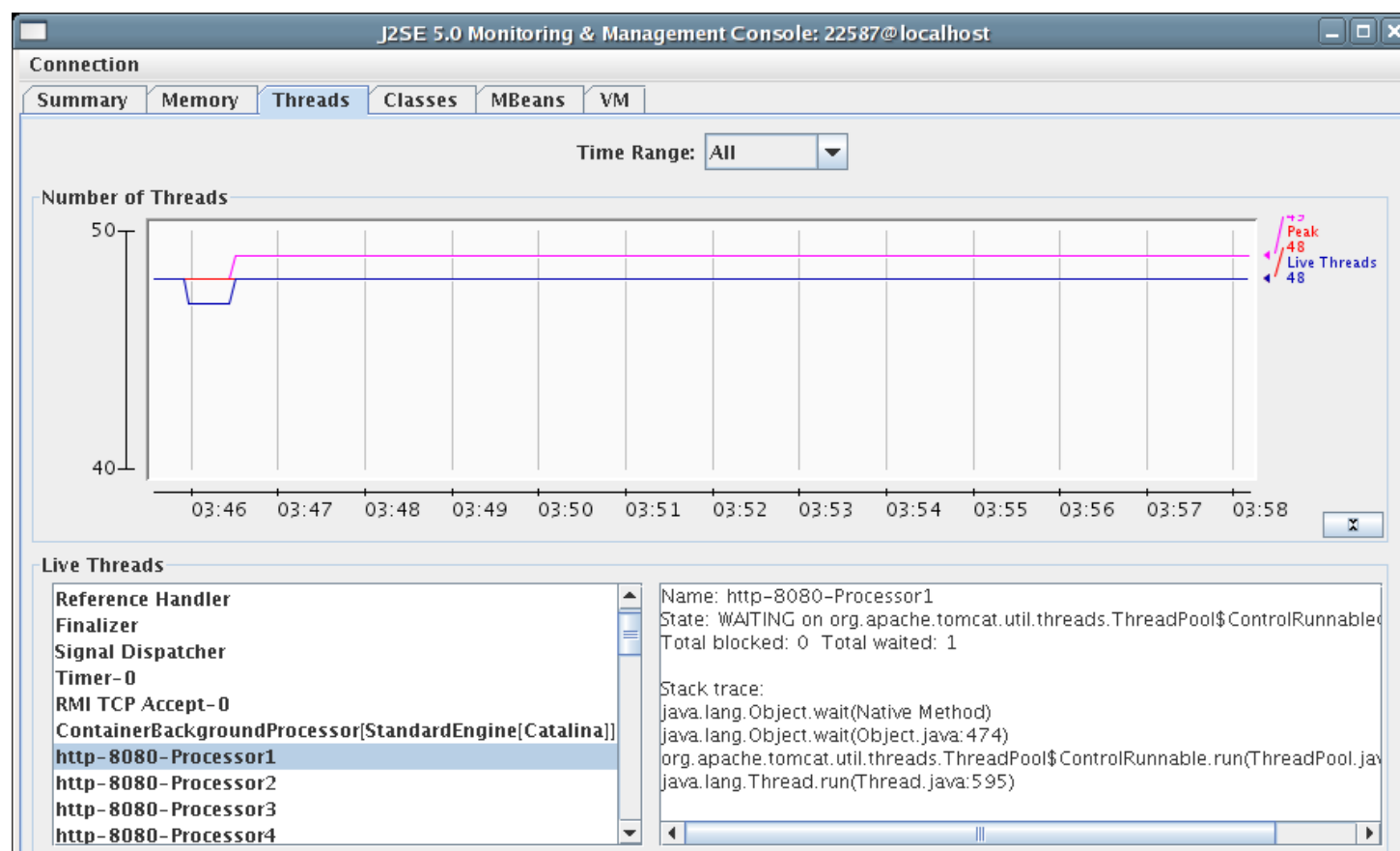
# Memória via JMX





# Threads via JMX

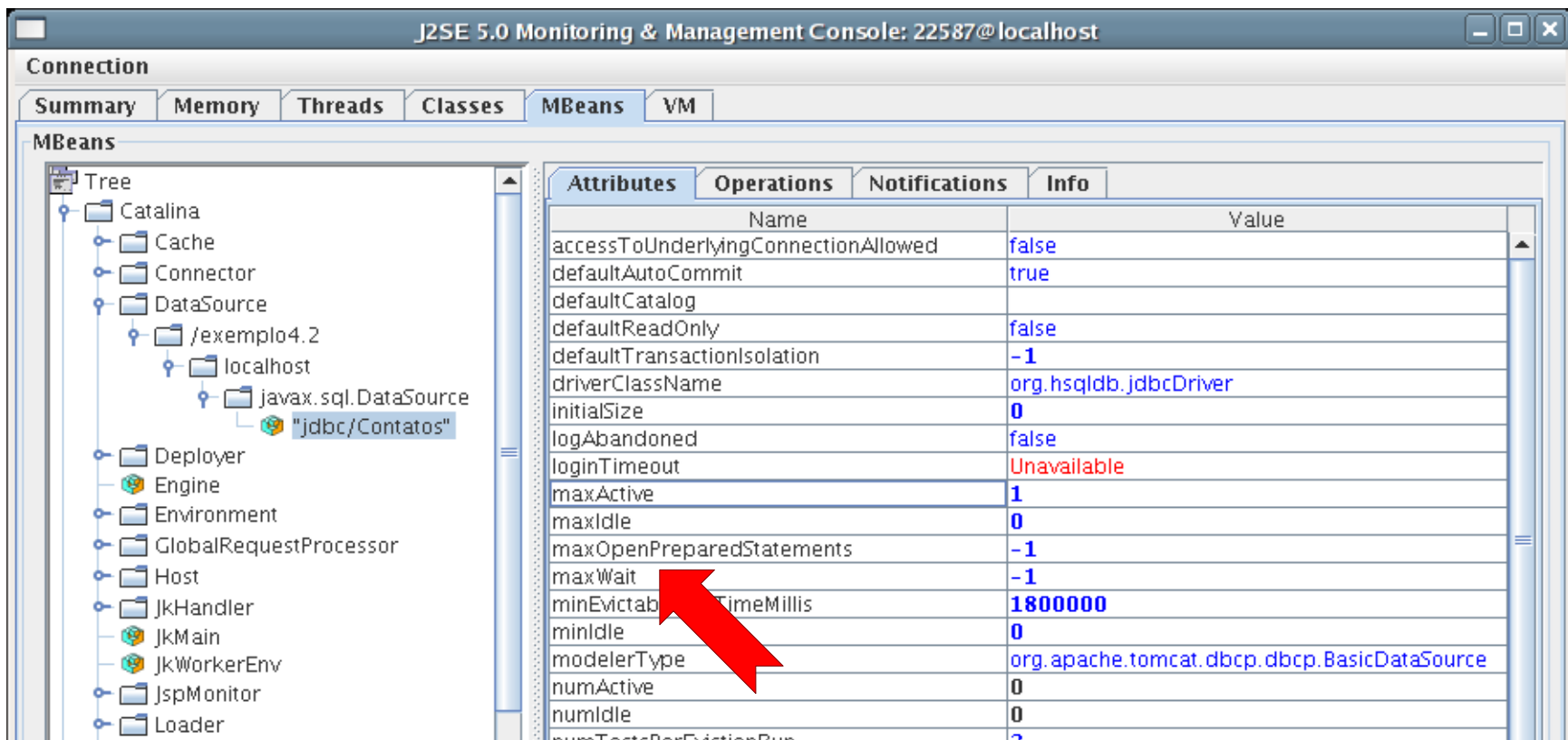
- Observe que o Tomcat nomeia claramente os threads dos conectores, que são as responsáveis por atender requisições





# DataSources via JMX

- ▶ Todos os componentes do Tomcat são expostos como MBeans, então é possível localizar o DataSource dentro do contexto e verificar a quantidade de conexões ativas (em uso)



The screenshot shows the J2SE 5.0 Monitoring & Management Console for 22587@localhost. The 'MBeans' tab is selected, and the 'Tree' view on the left shows the hierarchy: Catalina > Connector > DataSource > /exemplo4.2 > localhost > javax.sql.DataSource > "jdbc/Contatos". The 'Attributes' tab is active, displaying a table of DataSource properties.

Name	Value
accessToUnderlyingConnectionAllowed	false
defaultAutoCommit	true
defaultCatalog	
defaultReadOnly	false
defaultTransactionIsolation	-1
driverClassName	org.hsqldb.jdbcDriver
initialSize	0
logAbandoned	false
loginTimeout	Unavailable
maxActive	1
maxIdle	0
maxOpenPreparedStatements	-1
maxWait	-1
minEvictableIdleTimeMillis	1800000
minIdle	0
modelerType	org.apache.tomcat.dbcp.dbcp.BasicDataSource
numActive	0
numIdle	0
numTestsPerEvictionRun	2

A red arrow points to the 'minEvictableIdleTimeMillis' attribute, which has a value of 1800000.



## 16.5.3. JMX Proxy Servlet

- ▶ O Manager do Tomcat 5.5 em diante fornece um Servlet genérico para operações JMX
- ▶ Ele não aparece nos links, exigindo codificação manual das URLs, mas é ótimo para scripts!





# Consulta a MBeans

- ▶ O parâmetro **qry** lista MBeans dado o nome
- ▶ 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
- ▶ O Tomcat fornece MBeans que representam Aplicações Web, Servlets, DataSources, Realms e Conectores, entre outros





# Exemplos de Qry:

## Listar Servlets 1/2

- ▶ Lista 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
- ▶ (resultado no próximo slide)





# Exemplos de Qry:

## Listar Servlets 2/2

► OK - Number of results: 33

Name:

Catalina:j2eeType=Servlet,name=contatos,WebModule=//localhost/exemplo3.3,J2EEApplication=none,J2EEServer=none

modelerType: org.apache.tomcat.util.modeler.BaseModelMBean

objectName:

Catalina:j2eeType=Servlet,name=contatos,WebModule=//localhost/exemplo3.3,J2EEApplication=none,J2EEServer=none

minTime: 9223372036854775807

statisticsProvider: false

loadTime: 0

classLoadTime: 0

processingTime: 0

requestCount: 0

errorCount: 0

stateManageable: false

eventProvider: false

engineName: Catalina

maxTime: 0





# Exemplos de Qry: DataSources

- ▶ Quantidade de conexões ativas para bancos de dados

- ▶ [http://127.0.0.1:8080/manager/jmxproxy?qry=\\*&type=DataSource%2c\\*](http://127.0.0.1:8080/manager/jmxproxy?qry=*&type=DataSource%2c*)

- ▶ Name:

Catalina:type=DataSource,path=/exemplo3.3,host=localhost,class=javax.sql.DataSource,name="jdbc/Contatos"

...

url: jdbc:hsqldb:file:/home/fernando/bd/contatos;shutdown=true

...

numActive: 0

maxActive: 1

numIdle: 0





# Exemplos de Qry: RequestProcessors

## ► Requisições em atendimento

► [http://127.0.0.1:8080/manager/jmxproxy?qry=%3Atype=RequestProcessor%2c\\*](http://127.0.0.1:8080/manager/jmxproxy?qry=%3Atype=RequestProcessor%2c*)

► Name: Catalina:type=RequestProcessor,worker=http-8080,name=HttpRequest1

...

requestBytesSent: 0

bytesReceived: 0

requestProcessingTime: 2

...

currentQueryString: **qry=%3Atype=RequestProcessor%2c\***

bytesSent: 160829

currentUri: **/manager/jmxproxy**

...

stage: 3

method: GET

requestBytesReceived: 0





# Exemplos de Qry: Thread Pools

- ▶ Threads ocupados e ociosos

- ▶ [http://127.0.0.1:8080/manager/jmxproxy?qry=\\*&type=ThreadPool%2c\\*](http://127.0.0.1:8080/manager/jmxproxy?qry=*&type=ThreadPool%2c*)

- ▶ Name: Catalina:type=ThreadPool,name=http-8080

...

maxThreads: 40

currentThreadsBusy: 1

...

acceptorThreadCount: 1

...

currentThreadCount: 1







## 16.6. Troubleshooting

- ▶ Ao longo deste curso fomos apresentados a várias ferramentas úteis para depuração de problemas com o Tomcat, entre elas:
  - ▶ Válvula de logs de acess
  - ▶ Logging API do Java SE
  - ▶ Tomcat Manager
  - ▶ JMX via JConsole
- ▶ Mas uma ferramentas adicional será apresentada:
  - ▶ Thread Dumps





## 16.6.1. OutOfMemory na PermGen

- ▶ A PermGen (“geração permanente”) é uma área separado do heap que salva bytecodes de classes
- ▶ A JVM da são não reaproveita a área ocupada por versões antigas de classes após um re-deployment
- ▶ Então ela irá se esgotar mais cedo ou mais tarde
- ▶ O problema pode ser aliviado com a opção - **XX:MaxPermSize** da JVM





## 16.6.2. DataSource Leaks

- ▶ Um dos problemas mais sérios de aplicações são os “vazamentos” (leaks) de conexões ao banco de dados
- ▶ Isto acontece por erro de programação, onde conexões são abertas (retiradas do pool) mas nunca fechadas (devolvidas ao pool)
- ▶ Então, após algum tempo não haverão mais conexões disponíveis para uso por novas requisições
- ▶ Em geral a causa é não usar **finally** depois





# Detectando DataSource Leaks

- ▶ Dentro de um elemento **<Resource>** que configura um DataSource, podem ser definidos os atributos:
  - ▶ **removeAbandoned="true"**  
Instrui o Tomcat a forçar a liberação de conexões que pareçam 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





## 16.6.3. 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”?





# Thread Dump

- ▶ Ao se enviar um sinal **SIGQUIT** para o processo que roda uma JVM, a JVM salva um Thread Dump textual na saída padrão
- ▶ No Tomcat, a saída padrão é redirecionada para *log/catalina.out*
- ▶ O Thread Dump 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 aguardando por E/S; ou se está bloqueado em um semáforo





# Gerando um Thread Dump

- ▶ Primeiro, identifique o processo do Tomcat
- ▶ Depois, envie o SIGQUIT
- ▶ Então abra o catalina.out
- ▶ **# ps ax | grep tomcat**  
13978 pts/5 Sl 0:06 /usr/lib/jvm/java-X.X.X/bin/java -  
Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager -  
Djava.util.logging.config.file=/home/fernando/apache-tomcat-  
6.0.13/conf/logging.properties  
...  
**# kill -QUIT 13978**  
**# vi ../logs/catalina.out**





# Exemplo de Thread Ocioso

- ▶ 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]
  - at java.lang.Object.wait(Native Method)
  - waiting on <0x892f03a0> (a org.apache.tomcat.util.net.JIoEndpoint\$Worker)
  - at java.lang.Object.wait(Object.java:474)
  - at org.apache.tomcat.util.net.JIoEndpoint\$Worker.await(JIoEndpoint.java:416)
  - locked <0x892f03a0> (a org.apache.tomcat.util.net.JIoEndpoint\$Worker)
  - at org.apache.tomcat.util.net.JIoEndpoint\$Worker.run(JIoEndpoint.java:442)
  - at java.lang.Thread.run(Thread.java:595)







# Exemplo de Thread Trabalhando

## ► Clicado “Full Server Status” no Manager

► "http-8080-1" daemon prio=1 tid=0x09d33980 nid=0x36a4 runnable [0xb19d6000..0xb19d6fb0]  
at javax.management.ObjectName.getKeyPropertyList(ObjectName.java:1480)

...  
at

com.sun.jmx.interceptor.DefaultMBeanServerInterceptor.queryMBeans(DefaultMBeanServerInterceptor.java:472)

at org.apache.catalina.manager.StatusTransformer.writeContext(StatusTransformer.java:679)

at org.apache.catalina.manager.StatusTransformer.writeDetailedState(StatusTransformer.java:594)

at org.apache.catalina.manager.StatusManagerServlet.doGet(StatusManagerServlet.java:299)





# Conclusão

- ▶ Lab 1. Memória com Manager e ps
- ▶ Lab 2. Monitoração JMX via JConsole
- ▶ Lab 3. Monitoração JMX via Manager

