Apache Tomcat

Versão 8.0

Helder da Rocha

(Atualizado em 01/12/2015)

Apache Tomcat

Objetivos do treinamento

- Conhecer a arquitetura e os recursos do Tomcat
- Explorar os recursos do Tomcat na prática
- Configurar o Tomcat para uso eficiente (tuning e monitoração)

Programa

Primeiro dia

1. Introdução

- Demonstração

1. Arquitetura

- Demonstração

1. Instalação e Uso

- Instalar
- Iniciar, encerrar

1. Configuração e server.xml
   *Trocar a porta

- Criar uma instância separada
- Criar um host virtual

1. Aplicações: WebApps e contextos
   *Criar aplicação Servlet / JSP

- Instalar aplicação
- Configurar contextos

1. Manager e ferramentas

- Configurar o Manager

1. JNDI Resources

- Configurar um bean

1. JDBC Datasources

- Configurar um pool e acessar

Segundo dia

1. Realms, autenticação e autorização

- Configurar autenticação
- Configurar autorização

1. Valves

- Configurar uma Valve

1. Apache Connector

- Conectar o Apache de duas formas

1. Load Balancing

- Fazer load balancing de duas formas

1. Logging

- Configurar logging

1. Clustering

- Configurar cluster

1. Tuning, segurança e monitoração

- Demonstrar problemas
- Monitorar memória
- Demonstrar vulnerabilidades
- Ajustar performance

Material

- Documentação online do Tomcat 8: How-tos, Tutoriais e Referências
- Esta apostila
- Slides
- Código-fonte dos exemplos (GitHub)

—

1 Sobre o Tomcat

Apache Tomcat é uma implementação open source das especificações Java Servlet, JavaServerPages Java Expression Language e Java WebSocket. Essas especificações foram desenvolvidas através da Java Community Process.

O Tomcat é distribuído sob a licença Apache 2.0.

A última versão estável lançada até o momento (da publicação desta apresentação) é a versão 8.0 (8.0.32).

Apache Tomcat 8.0 implementa as especificações Servlet 3.1 e JavaServer Pages 2.3.

—

2 Arquitetura

Tomcat é um servidor Web. O termo servidor (server) representa a máquina (host) onde rodam diversos componentes, que oferecem serviços (service). Cada serviço consiste de um ou mais processadores (engine) que o cliente acessa através de um conector (connector). Diversos aspectos desses componentes podem ser customizados.

Principais componentes

Tomcat contém diversos componentes de serviço. Os mais importantes são:

Catalina é o componente principal do Tomcat e implementação padrão de um container Web Java EE que adere à especificação Servlet, que é a base de todas as aplicações Web em Java (inclusive JSP, JSF, etc.) Catalina é, portanto, um Servlet Container e é o primeiro serviço que é iniciado no Tomcat. Além de container para Web apps, a configuração do serviço e os aspectos relativos a segurança das aplicações são responsabilidade do componente Catalina.

Coyote é o componente do Tomcat que suporta o protocolo HTTP 1.1 como um servidor Web. Isto permite que Catalina também acumule a função de servir eficientemente páginas HTML, aplicações CGI, e todas as funções de um servidor Web como o Apache Web Server.

O componente **Coyote JK** oferece a mesma interface idêntica de servidor Web HTTP para o Catalina, mas em vez de processar os dados HTTP internamente, redireciona através do protocolo JK (AJP - Apache JServ Protocol) para um outro servidor, geralmente o Apache Web Server.

Jasper é o processador (parser/compilador) JSP que interpreta e compila código JSP, taglibs, JSTL e gerencia-os em um pool para reuso.

Além desses componentes, Tomcat contém vários outros que cuidam do deployment de aplicações Web (contextos), clustering e alta disponibilidade.

Servidor de aplicações

Apache TomEE é um servidor de aplicações Java EE baseado no Apache Tomcat que combina vários projetos open source oferecendo suporte a EJB, CDI, JPA, JMS, REST, SOAP, JTA e JSF.

Estrutura e variáveis de ambiente

A estrutura de uma instalação do Tomcat depende de como é instalado (standalone, múltiplas instâncias, serviço, cluster, embutido).

Há duas variáveis de ambiente que são usadas para representar o local onde o servidor está instalado:

- `$CATALINA_HOME` - raiz da instalação
- `$CATALINA_BASE` - raiz de cada instância instalada (se só houver apenas uma instância instalada, é igual a `$CATALINA_HOME`)

Outras variáveis de ambiente e diferentes instalações serão detalhadas mais adiante.

Em todas as instalações há uma estrutura padrão de diretórios dentro de $CATALINA_HOME:

- `/bin` - contém os scripts para iniciar, reiniciar, finalizar o Tomcat e outros recursos

- `/conf` - contém arquivos de configuração, dentre eles o server.xml que é o arquivo de configuração principal
- `/logs` - diretório default para armazenar arquivos de log
- `/webapps` - contém as aplicações web (contextos)

Os arquivos de configuração são lidos na inicialização do servidor. Se forem feitas alterações, o servidor precisará ser reinicializado.

Server, Engine, Host e Context

Esta é a hierarquia mais importante dos componentes do Tomcat:

Um servidor (**Server**) pode ter
um ou mais processadores (**Engine**), associados a
um ou mais endereços Web (**Host**), que contém
um ou mais aplicações (**Context**)

O Tomcat permite diversas configurações em cada um desses componentes.

Instalação default

Na instalação default, o Tomcat está pronto para rodar aplicações. A instalação pode ser feita rapidamente através de *hot deployment*: copiando um WAR (componente padrão Java EE) para dentro da pasta `/webapps` enquanto o servidor estiver rodando. Dentro de segundos o WAR será aberto e a aplicação será instalada.

Em uma instalação default do Tomcat, a aplicação, se estiver construída corretamente, poderá pouco depois da cópia ser acessada pelo endereço:

```
http://localhost:8080/contexto
```

—

3 Instalação

Antes de tentar instalar o Tomcat, é importante ter um ambiente Java instalado de forma correta.

O Tomcat 8 requer um ambiente JRE 7 instalado. Deve-se configurar a variável de ambiente `$JAVA_HOME` ou `$JRE_HOME` indicando a localização do ambiente.

Dependendo de como foi instalado, as localizações dos arquivos de configuração, scripts e webapps poderão variar.

A instalação também poderá ser diferente para cada versão. O ideal é verificar o arquivo `RUNNING.txt` correspondente à versão usada, que faz parte da distribuição e documentação do Tomcat.

Instalação como aplicação

Independente de plataforma, o Tomcat pode ser instalado simplesmente baixando o `zip` ou `tar.gz` disponível no site, expandindo o arquivo em um diretório que será o `$CATALINA_HOME`.

`$CATALINA_HOME` deverá ser registrado como variável de ambiente (e `$CATALINA_HOME/bin` no `$PATH` caso seja desejado que os scripts executem de qualquer lugar).

Instalação como serviço

A instalação como *serviço* é dependente de plataforma. A instalação como serviço poderá facilitar o uso do servidor, garantindo o reinício automático, e facilidades para interromper, iniciar e reiniciar o serviço.

Instaladores de serviço garantem a instalação das dependências e configuração das variáveis de ambiente, mas a localização dos arquivos poderá ser diferente.

Serviço Tomcat no Windows

Para instalar um serviço no Windows deve-se baixar o instalador disponível no site tomcat.apache.org.

Tomcat 8 no Ubuntu 15

No Ubuntu 15, o Tomcat 8 pode ser instalado como serviço usando um único comando:

```
$ sudo apto-get install tomcat8
```

A instalação default será em `/etc/tomcat8`.

Tomcat 8 no Ubuntu 14

No Ubuntu 14, o mesmo pode ser feito para instalar o Tomcat 7. para instalar o Tomcat 8, neste caso, é necessário baixar as depedendências, o pacote binário e configurar as permissões.

No exemplo abaixo instala-se o JDK, cria-se um usuário/grupo `tomcat/tomcat`, baixa-se o pacote Tomcat 8, cria-se a pasta /opt/tomcat e expande-se o pacote nessa pasta:

```
$ sudo apt-get install default-jdk
$ sudo groupadd tomcat
$ sudo useradd -s /bin/false -g tomcat -d /opt/tomcat tomcat
$ wget http://mirror.nbtelecom.com.br/apache/tomcat/tomcat-8/v8.0.32/bin/apache-tom
cat-8.0.32.tar.gz
$ sudo mkdir /opt/tomcat
$ sudo tar xvf apache-tomcat-8*tar.gz -C /opt/tomcat --strip-components=1
```

Em seguida deve-se configurar as permissões da pasta recursivamente (para que todo o pacote pertenca ao usuário/grupo `tomcat/tomcat`).

Tomcat no Mac OS

No ambinente deve primeiro estar instalado o pacote *HomeBrew* (`http://brew.sh`). O comando abaixo instala a versão mais recente:

```
$ brew install tomcat
```

Instalação embutida

A instalação também pode ser realizada através de um ambiente de desenvolvimento (IDE) que criará instância separadas para diferentes aplicações.

[Como instalar e configurar Tomcat no Eclipse]

Execução

Para iniciar, finalizar e reiniciar o Tomcat instalado como serviço, é preciso utilizar as ferramentas que diferem em cada plataforma. Em Windows use a aplicação gráfica que é fornecida. Em ambientes Maco OS e Linux, geralmente utiliza-se instruções de linha de comando específicas da plataforma.

Tomcat instalado como aplicação

A maior parte da funcionalidade multiplataforma está no script `catalina` (`catalina.bat` ou `catalina.sh`). Outros scripts são atalhos para funções comuns de `catalina` .

- `catalina` - contém o código que realiza start, stop e restart
- `startup` - roda `catalina start`
- `shutdown` - roda `catalina stop`
- `version` - roda `catalina version`
- `digest` - usado para gerar senhas criptografadas

Há também outros scripts que são exclusivos para a plataforma usada.

O script **catalina** também contém outras opções além de `version` , `start` e `stop` .

- `run` - igual a start, mas não grava logs
  embedded - modo embutido, geralmente usado por IDEs
- `debug` - modo debug
- `-help` lista outros comandos

Os comandos também pode ser definidos como propriedades globais da JVM. Neste caso devem ser declarados na variável `CATALINA_OPTS` .

Os scripts ficam em `$CATALINA_HOME/bin` . Se este caminho estiver no `$PATH` , o Tomcat pode ser controlado de qualquer lugar:

Por exemplo, para iniciar:

- `startup.bat` ou `startup.sh`
- `catalina.bat start` ou `catalina.sh start`

e para finalizar o serviço:

- `shutdown.bat` ou `shutdown.sh`
- `catalina.bat stop` ou `catalina.sh stop`

A instalação básica do Tomcat não oferece serviço de reinício nem de início automático. Para reiniciar é preciso interromper o serviço e depois reiniciar (`catalina stop` , seguido de `catalina start`). O shutdown poderá demorar.

Se o Tomcat for instalado como serviço, é possível que o sistema operacionalo oferece a possibilidade de reiniciar o Tomcat.

Tomcat instalado como serviço

Se Tomcat foi instalado como um serviço, deve-se usar as opções disponíveis na aplicação que gerencia o serviço para iniciar, reiniciar e interromper o processo. Isto depende do sistema operacional usado. O Windows oferece essa opção através de uma interface gráfica. Em linha de comando e sistemas Unix/Linux geralmente usa-se um script `init` ou `service`, por exemplo:

```
/etc/rc.d/init.d/tomcat start
```

ou

```
service tomcat start
```

O nome do serviço poderá ser diferente. Geralmente tem o mesmo nome do serviço instalado (ex: `tomcat7`, `tomcat8`, etc.)

É possível usar ferramentas do sistema operacional para garantir o reinício automático e início ao login do Tomcat mais facilmente se ele estiver instalado como serviço.

Variáveis de ambiente importantes

Na inicialização, o log por default começa imprimindo as variáveis de ambiente. Se o Tomcat for iniciado como aplicação, o log por default é impresso na tela do console. Verifique as variáveis usadas na inicialização.

```
$ catalina.sh start
Using CATALINA_BASE: /home/helderdarocha/apache-tomcat-8.0.32
Using CATALINA_HOME: /home/helderdarocha/apache-tomcat-8.0.32
Using CATALINA_TMPDIR: /home/helderdarocha/apache-tomcat-8.0.32/temp
Using JRE_HOME: /usr/java/jdk1.8.0_01
```

Erros ou falha na inicialização poderão ocorrer com `JAVA_HOME` ou `JRE_HOME` definidas incorretamente.

Teste da instalação

Inicie o serviço rodando os scripts em `$CATALINA_HOME/bin` ou usando ferramenta de administração do serviço Tomcat.

Acesse o site `localhost:8080`.

Erros de instalação comuns

Os principais erros de instalação tem a ver com instâncias rodando previamente e conflito de portas. A porta default do Tomcat é 8080, que é uma porta frequentemente usada por default por vários serviços simples que criam servidores Web. Verifique se sua plataforma já não contém outro serviço rodando nessa porta usando as ferramentas específicas do seu sistema operacional (ex: `grep`, `netstat`). Se você estiver usando algum IDE que roda o Tomcat, ele também pode estar usando a porta, causando conflito.
As portas usadas pelo Tomcat podem ser alteradas com facilidade na configuração.

O **shutdown** do Tomcat não é garantido, e pode demorar. Um servlet pode ter criado um thread com prioridade mais alta, que impede o shutdown, pode ter havido um estouro de memória (que impede o acesso à porta de shutdown).
Se o processo Java do Tomcat não tiver terminado, a porta pode ainda estar sendo bloqueada e será necessário forçar o seu fim usando ferramentas do seu sistema operacional (kill, etc.) ou mesmo, em casos extremos, reiniciar o sistema.

—

4 Configuração

A maior parte da configuração do Tomcat é realizada em arquivos de $CATALINA_HOME/conf. Outras envolvem alterações em scripts e componentes.

Esta seção explorará algumas configurações básicas. Outras configurações serão tratadas em seções a parte.

Os principais arquivos de configuração são:

- `server.xml` - principal arquivo de configuração.
- `web.xml` - arquivo de configuração global para definir defaults para todas as aplicações Web (contém os mesmos elementos do web.xml contido em cada aplicação).
- `tomcat-users.xml` - lista default de roles, usuários e senhas usadas pelo domínio UserDatabaseRealm para autenticação em serviços do Tomcat.
- `catalina.policy` - políticas de segurança Java
- `context.xml` - configuração default para todos os contextos dos hosts da instalação do Tomcat

Após alterar qualquer arquivo de configuração, as alterações só entram em vigor com a reinicialização do servidor.

server.xml

Cada elemento filho do server.xml representa um objeto de configuração dentro da arquitetura do Tomcat. O elemento raiz é `<Server>` e representa todo o servidor. Um `<Server>` deve conter pelo menos um elemento `<Service>`, que contem um ou mais `<Connector>` (cada um em uma porta diferente) associado a um processador `<Engine>`, que contém um elemento `<Host>`. Cada um desses elementos pode ser configurado por sub-elementos e atributos, definindo portas, classes, timeouts, comportamentos, etc.

Além de elementos `<Service>`, um `<Server>` tipicamente contém um ou mais `<Listener>` (que configura componentes que irão processar eventos do ciclo de vida do servidor) e um `<GlobalNamingResources>` que guarda um ou mais resources JNDI.

A seguir um documento `server.xml` típico (javacodegeeks.com):

```
<?xml version='1.0' encoding='utf-8'?>
<Server port="8005" shutdown="SHUTDOWN">
  <Listener className="org.apache.catalina.core.JasperListener" />
  <Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="on
" />
  <Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />
  <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener"
 />
  <Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener"
/>

  <GlobalNamingResources>
    <Resource name="UserDatabase" auth="Container"
              type="org.apache.catalina.UserDatabase"
              description="User database that can be updated and saved"
              factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
              pathname="conf/tomcat-users.xml" />
  </GlobalNamingResources>

  <Service name="Catalina">
    <Connector port="8080" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />
    <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
```

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

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

      <Host name="localhost"  appBase="webapps"
            unpackWARs="true" autoDeploy="true">
        <Valve className="org.apache.catalina.valves.AccessLogValve" directory="log
s"
               prefix="localhost_access_log." suffix=".txt"
               pattern="%h %l %u %t "%r" %s %b" />
          <Context path="" docBase="ROOT"/>
             <Context path="/teste" docBase="/home/helderdarocha/teste"
                          reloadable="true" crossContext="true"/>
      </Host>
    </Engine>
  </Service>
</Server>
```

Abaixo uma breve descrição desses principais elementos e atributos.

Elemento raiz:

- `<Server>` - raiz do documento. O atributo `port` informa a porta para onde é enviado o comando de shutdown do servidor. Se o valor for -1, a porta será desabilitada. O atributo `address` informa o IP ou nome da máquina que pode enviar comando de shutdown. Se ausente a máquina será localhost. O atributo `shutdown` informa a string de comando que deve ser recebida na porta informada para dar shutdown no servidor.

Elementos filho:

- `<Listener>` - um ou mais desses elementos informam classes que são notificadas em eventos específicos do ciclo de vida do servidor.
- `<GlobalNamingResources>` - define resources acessíveis via JNDI e podem ser recuperados por nome (atributo `name`).
- `<Service>` - cada um contém uma coleção de conectores (`<Connector>`)associados a um processador (`<Engine>`). Pode haver vários, cada um identificado com um nome (atributo `name`) diferente. Pode também haver um atributo `className` contendo o nome de uma classe que implementa a interface `org.apache.catalina.Service` .

Sub-elementos de `<Service>`

- `<Executor>` - um pool de threads compartilhado que será compartilhado por todos os conectores. Permite definir um número mínimo (`minSpareThreads`) e máximo (`maxThreads`) de threads ativos e outros atributos de threads.
- `<Connector>` - associado com uma porta TCP/IP (atributo `port`) para realizar interface com o cliente; pode ser um servidor web, um conector AJP ou outro. Pode haver mais de um. Pode conter zero ou mais `<Valve>` (filtros de processamento). Atributos estabelecem timeout (`connectionTimeout`), redirecionamento (`redirectPort`), limite de clientes, threads, executor, protocolo usado e muitas outras propriedades.
- `<Engine>` - representa o processador que recebe todas as requisições de um dos conectores do serviço. Cada processador pode ter seu próprio diretório no sistema (`CATALINA_BASE`) Atributos permitem configurar threads, delays, rotas, etc. Além de `<Host>` pode conter `<Realm>` (domínio de segurança), `<Valve>` (filtros), `<Listener>` e `<Cluster>` .

Outros elementos:

- `<Host>` - um endereço virtual, declarado dentro de um `<Engine>` e pode conter realms, contextos, válvulas, clusters, etc.
- `Context` - representa um contexto ou aplicação Web. Contextos também podem ser criados dinamicamente de várias formas.
- `<Realm>` - usado em `<Engine>` ou `<Host>` , determina um domínio de segurança (banco de dados de usuários e grupos)
- `<Cluster>` - configura clustering (replicação de sessão e atributos de contexto e deployment distribuído)
- `<Valve>` - interceptador de requisições HTTP baseado em padrão (atributo `pattern`). O atributo `className` informa o nome da classe que implementa o Valve, o atributo `prefix` informa o nome do arquivo de log e `directory` onde o arquivo de logs será armazenado.

Pode-se ter vários arquivos `server.xml` com configurações distintas, usando um nome diferente (tipicamente `server-nome.xml`). Para executá-los é preciso informar o nome de cada arquivo ao iniciar o Tomcat:

```
$ catalina.sh start -config /conf/server-dois.xml
```

web.xml

Cada aplicação instalada em $CATALINA_HOME/webapps possui um arquivo WEB-INF/web.xml dentro de seu contexto. O arquivo web.xml localizado em $CATALINA_HOME/conf é idêntico, mas define defaults que podem ser sobrepostos em qualquer aplicação. Este arquivo é lido antes do web.xml de cada aplicação, de forma que qualquer configuração que for definido nele valerá para todas as aplicações. Isto inclui definições

de listeners, variáveis, filtros e até servlets.

tomcat-users.xml

Este arquivo contém uma lista de usuários, grupos e senhas usadas para acessar aplicações através do domínio UserDatabaseRealm.

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

catalina.policy

Um arquivo de policy determina permissões para um determinado escopo. O arquivo catalina.policy permite estabelecer restrições de uso e acesso para o Tomcat quando ele for iniciado com a opção `-security`. O escopo consiste de um conjunto de classes em Java e cada permissão estabelece o nível de acesso permitido.

catalina.properties

Este arquivo permite definir propriedades que serão reusadas no arquivo server.xml. Por exemplo, pode-se definir uma propriedade:

```
versao=3.55
```

e acessar em server.xml usando a notação `${...}`:

```
<Valve version="${versao}" ...>
```

Configuração de contexto

Um contexto representa uma aplicação Web padrão, que é um arquivo WAR ou uma pasta contendo WEB-INF/web.xml de acordo com a especificação Java EE para aplicações Web.

O contexto é um caminho virtual para a aplicação. É acessível via cliente Web (ex: http://localhost:8080/contexto)

Há três maneiras de configurar um contexto (aplicação Web) no Tomcat:

1) Declarando em server.xml (dentro de `<Host>`); o caminho, que é o identificador do contexto, é declarado no atributo `path`. O atributo `docBase` informa onde está a aplicação. Pode ser um caminho absoluto ou relativo à pasta appBase do Host:

```
 <Host name="localhost" appBase="webapps">
    <Context path="" docBase="ROOT"/>
    <Context path="/extra" docBase="/opt/docs/extra"/>
</Host>
```

2) Declarado em um arquivo (fragmento) de contexto e configurando a pasta do contexto. Neste caso o `path` é o nome do arquivo menos a extensão `.xml`. O arquivo contém apenas um elemento `<Context>` e sub elementos, e deve estar localizado em $CATALINA_HOME/conf/nome-do-engine/nome-do-host. Por exemplo, o contexto /admin é definido em $CATALINA_HOME/conf/Catalina/localhost/admin.xml, é acessível via http://localhost:8080/admin e contém:

3) Declarado em um componente WAR (que contém a pasta do contexo) depositado uma pasta de hot-deployment (o `<Host>` deve habilitar). Em uma instalação default, esta pasta é `webapps`. Neste caso o `path` é o nome do arquivo menos a extensão `.war` e a pasta é criada automaticamente.

O arquivo `context.xml` de `conf/` permite definir propriedades globais para todos os contextos.

Pastas de configuração de processadores

Pastas dentro de `conf/` que têm o nome de processadores (engines), por exemplo `Catalina` contém configuração para contextos. Dentro da pasta de configuração pode haver uma ou mais pastas com os Hosts do processador, dentro dos quais são configurados os componentes de cada host.

Algumas configurações típicas

Configurações mais complexas (load balancing, clustering, etc.) serão exploradas em seções a parte.

Como criar instâncias diferentes

Crie uma pasta base para instâncias do Tomcat (CATALINA-BASE) onde a nova pasta será armazenada:

```
mkdir /opt/tomcat-instances/
cd tomcat-instances/
```

Depois crie uma pasta para a instância:

```
mkdir site1
cd site1
```

Copie a configuração do Tomcat para esta pasta, e crie versões vazias dos outros diretórios:

```
cp -a $CATALINA_HOME/conf .
mkdir common logs temp server shared webapps work
```

Em server.xml altere a porta de Shutdown e as portas de quaisquer conectores, de forma que tenham valores diferentes da instância principal:

…

Para iniciar essa instância, define CATALINA_BASE com o caminho do diretório criado, e inicie o Tomcat normalmente:

```
$ set CATALINA_BASE="/opt/tomcat-instances/site1"
$ set CATALINA_HOME="/opt/tomcat"
$ export CATALINA_BASE CATALINA_HOME
$ service tomcat start
```

Ou crie um script próprio para essa inicialização.

Como trocar porta de serviço

A porta default de serviço para o Tomcat é 8080. Para trocar por outra porta, localize o Conector principal no arquivo server.xml:

```
<Connector port="8080" protocol="HTTP/1.1"
                   connectionTimeout="20000"
                   redirectPort="8443" />
```

Mude a porta para o valor desejado e reinicie o servidor.

Como alterar configuração da JVM

Configurações da JVM podem ser alteradas através de opções da JVM passada através da variável de ambiente JAVA_OPTS, que pode ser declarada no script de execução ou via linha de comando.

Algumas configurações típicas incluem alteração do tamanho do heap, da área de memória usada para classes (PermGen), propriedades usadas para gerenciamento via JMX (jconsole), etc.

A linha abaixo declara algumas propriedades da JVM para o Tomcat:

```
JAVA_OPTS="-Xdebug -Xms384M -Xmx384M -Dfile.encoding=UTF-8"
```

Como configurar hosts virtuais

Hosts virtuais podem ser configurados em Tomcat adicionando elementos `<Host>` adicionais associando o nome do host a um diretório diferente para webapps. Primeiro, é preciso ter o host virtual já configurado no servidor de nomes (ou estaticamente, para teste ou uso local), e apontando para o endereço onde está rodando o Tomcat. Por exemplo:

```
<Engine name="Catalina" defaultHost="localhost">
      <Host name="localhost" appBase="webapps">
           ...
      </Host>
      <Host name="www.samovar.com.br" appBase="/opt/hosts/samovar/webapps">
           <Alias>samovar.com.br</Alias>
           <Context path="" docBase="ROOT"/>
      </Host>
</Engine>
```

Alias pode ser usado para adicionar aliases para o host.

Mais informações

http://tomcat.apache.org/tomcat-8.0-doc/config/

—

5 Aplicações

O Tomcat é uma aplicação que serve aplicações Web que aderem à especificação Java EE para aplicações Web. Aplicações típicas que são instaladas no Tomcat usam as tecnologias servlets, JSP, JSF, WebSockets, SOAP, REST, etc.

Web app

Uma aplicação Web padrão pode ser representada como um arquivo ZIP comprimido com extensão WAR, ou como um diretório. Ambos devem ter uma estrutura padrão com subdiretórios específicos para páginas estáticas, biblitecas (JARs), classes Java e arquivos de configuração. Essa estrutura é descrita abaixo:

- Raiz do contexto - é a pasta ou WAR que contém tudo. Deve conter os arquivos HTML, JSP e outros que serão acessados diretamente a partir do contexto da aplicação.
- /WEB-INF - é uma pasta privativa que não pode ser acessada pelo cliente. Quaisquer arquivos que estiverem aqui não poderão ser transferidos pelo servidor.
- /WEB-INF/web.xml - o Web Application Deployment Descriptor padrão para a aplicação. Ele contém informações de configuração, servlets, filtros e outros componentes.
- /WEB-INF/classes - É o classpath para classes Java.
- /WEB-INF/lib - O conteúdo de todos os JARs contidos nesta pasta será acrescentado ao classpath da aplicação.

A estrutura do Web App pode ser montada de qualquer maneira, ou podem-se usar ferramentas como Ant e Maven e IDEs para este fim. Pode-se ainda ter uma pasta /META-INF com outros dados, tipicamente arquivos usados pelas aplicações, e `context.xml` usado pelo Tomcat.

Breve introdução a aplicações Web e Java EE

Se você não tem conhecimento de aplicações Java EE, servlets e JSP, veja uma breve apresentação.

Aplicação exemplo

Veja como instalar uma aplicação exemplo.

—

6 Tomcat Manager

Manager

Manager é uma aplicação distribuída pelo Tomcat que oferece uma interface Web para realizar e gerenciar o deployment de aplicações, iniciar, interromper e reiniciar aplicações individuais e obter informações sobre o ambiente e a sua execução. Faz parte de uma instalação default.

Se não estiver configurada, ou se houver a necessidade de configurar um manager para outro host, pode-se baixar, copiar ou vincular a aplicação (WAR), instalar em uma pasta de webapps e configurar o contexto criando um arquivo manager.xml em $CATALINA_BASE/conf/Catalina/localhost:

```
<Context privileged="true" antiResourceLocking="false"
         docBase="${catalina.home}/webapps/manager">
  <Valve className="org.apache.catalina.valves.RemoteAddrValve"
         allow="127\.0\.0\.1" />
</Context>
```

O Valve acima garante que o manager seja acessível apenas de localhost.

Apesar de disponível, a instalação default do Tomcat não é distribuída com o acesso ao Manager habilitado. É preciso escolher um ou mais roles de acesso e associá-los a um usuário novo ou existente em CATALINA_BASE/conf/tomcat-users.xml. Os roles disponíveis são:

- manager-gui — Acesso à interface gráfica completa (/manager/html)
- manager-status — Acesso apenas à página “Server Status” (/manager/status)
- manager-script — Acesso à interface de texto (/manager/text) e à página “Server Status” - permite o deploy e outras funcionalidades via comandos de URL
- manager-jmx — Acesso à interface JMX e à página “Server Status” (/manager/jmxproxy)

Exemplo:

```
<tomcat-users>
    <user name="tomcat" password="tomcat" roles="tomcat" />
    <user name="admin"  password="tomcat" roles="manager-gui" />
    <user name="monitor"  password="tomcat" roles="manager-jmx, manager-script" />
 </tomcat-users>
```

A interface HTML está localizada (em uma instalação default) em http://localhost:8080/manager/html

As interfaces JMX e Script são vulneráveis a ataques externos, e portanto, devem ser usadas apenas em escopos limitados. Permitem amplo acesso a serviços do Tomcat e dados de monitoração. São destinados, principalmente, a aplicações cliente (IDEs), ferramentas de automação e de monitoração. Para maiores informações sobre as interfaces JMX e Script, consulte a documentação em:

http://tomcat.apache.org/tomcat-8.0-doc/manager-howto.html

Outras ferramentas

Ant e Maven

Vários recursos do Manager são acessíveis através de plugins e tarefas do Ant e Maven. Consulte os manuais de Ant e Maven para mais informações.

IDEs

Várias IDEs utilizam as interfaces de administração do Tomcat internamente.

TCD

Outra maneira de fazer o deployment de aplicações é usar uma ferramenta externa como o Ant, o Maven, uma IDE ou o TCD (Tomcat Client Deployer) - que deve ser baixado separadamente.

—

7 JNDI

Tomcat oferece um contexto JNDI InitialContext, que pode ser usado por aplicações Web para ter acesso a recursos disponíveis no registro, usando um lookup padrão JNDI.

Recursos podem ser definidos e declarados de forma estática usando web.xml ou context.xml.

Registro JNDI via web.xml

Os elementos disponíveis em web.xml para registrar informações via JNDI são:

- `<env-entry>` - Permite registrar uma informação associada a um nome usado para referenciá-la (ex: um valor, um caminho, nome de arquivo, etc.)
- `<resource-ref>` - Permite registrar um resource (geralmente uma fábrica de conexões JDBC, sessão de JavaMail, etc.)
- `<resource-env-ref>` - Similar a `resource-ref` e mais simples de usar para resources que não requerem autenticação.

Alguns dos elementos podem ser resolvidos automaticamente pelo Tomcat. Outros podem requerer configuração adicional no Tomcat (feita em `<Context>`)

Registro JNDI via Context

Configuração de recursos específica do Tomcat é realizada em um elemento `<Context>` usando sub-elementos:

- `<Environment>` - Nomes e valores (escalares - strings. números, caminhos, etc.) que serão expostos via contexto JNDI (equivalente a `<env-entry>` no web.xml).
- `<Resource>` - Nome e tipo de dados de um resource disponibilizado à aplicação. (equivalente a `<resource-ref>` do web.xml).
- `<ResourceLink>` - Um vínculo para resource definido no contexto JNDI global. Necessário para dar à aplicação Web acesso a recurso definido em `<GlobalNamingResources>`.
- `<Transaction>` - Adiciona fábrica para instanciar o objeto UserTransaction (acessível via java:comp/UserTransaction).

Registro JNDI via GlobalNamingResources

Um namespace de recursos globais disponível para o servidor inteiro. Os recursos declarados aqui podem ser expostos às aplicações Web usando um `<ResourceLink>`.

Como usar um recurso

Exemplo de acesso a um datasource:

```
Context initCtx = new InitialContext();
Context envCtx = (Context) initCtx.lookup("java:comp/env");

DataSource ds = (DataSource)
  envCtx.lookup("jdbc/EmployeeDB");

Connection conn = ds.getConnection();
// usar a conexão
conn.close();
```

Exemplos

Como registrar um bean, data sources, sessões de JavaMail e outros recursos. Veja exemplos na documentação oficial em
http://tomcat.apache.org/tomcat-8.0-doc/jndi-resources-howto.html

—

8 Configuração de datasources

Há duas formas nativas de configurar datasources em Tomcat. Pode-se também usar pacotes externos (ex: C3PO).

DBCP

A implementação default para pool de conexões JDBC do Tomcat depende de bibliotecas do projeto Apache Commons:

- Commons DBCP
- Commons Pool

A configuração de um Datasource é realizado de forma transparente através de um `<Resource>` criado para um `<Context>`. O exemplo abaixo ilustra a configuração de um DataSource para um banco MySQL:

```
<Context>

    <!-- maxTotal: Maximum number of database connections in pool. Make sure you
         configure your mysqld max_connections large enough to handle
         all of your db connections. Set to -1 for no limit.
         -->

    <!-- maxIdle: Maximum number of idle database connections to retain in pool.
         Set to -1 for no limit.  See also the DBCP documentation on this
         and the minEvictableIdleTimeMillis configuration parameter.
         -->

    <!-- maxWaitMillis: Maximum time to wait for a database connection to become av
ailable
         in ms, in this example 10 seconds. An Exception is thrown if
         this timeout is exceeded.  Set to -1 to wait indefinitely.
         -->

    <!-- username and password: MySQL username and password for database connection
s  -->

    <!-- driverClassName: Class name for the old mm.mysql JDBC driver is
         org.gjt.mm.mysql.Driver - we recommend using Connector/J though.
         Class name for the official MySQL Connector/J driver is com.mysql.jdbc.Dri
ver.
         -->

    <!-- url: The JDBC connection url for connecting to your MySQL database.
         -->

  <Resource name="jdbc/TestDB" auth="Container" type="javax.sql.DataSource"
              maxTotal="100" maxIdle="30" maxWaitMillis="10000"
              username="javauser" password="javadude" driverClassName="com.mysql.j
dbc.Driver"
              url="jdbc:mysql://localhost:3306/javatest" />

</Context>
```

Depois de configurado o Resource no contexto, deve-se configurar um resource-ref no web.xml da aplicação que irá fazer uso do datasource:

```
<web-app ...>
  ...
  <resource-ref>
      <description>DB Connection</description>
      <res-ref-name>jdbc/TestDB</res-ref-name>
      <res-type>javax.sql.DataSource</res-type>
      <res-auth>Container</res-auth>
  </resource-ref>
</web-app>
```

Agora uma aplicação pode obter o datasource via JNDI, através de lookup em `jdbc/TestDB`.

Veja exemplos de aplicações teste para vários bancos de dados em http://tomcat.apache.org/tomcat-8.0-doc/jndi-datasource-examples-howto.html

Tomcat JDBC Connection Pool

Uma alternativa mais nova à solução DBCP. A configuração é idêntica, mas é preciso acrescentar o atributo `factory` indicando essa implementação, que não é default.

```
  <Resource name="jdbc/TestDB" auth="Container" type="javax.sql.DataSource"
              maxTotal="100" maxIdle="30" maxWaitMillis="10000"
              username="javauser" password="javadude" driverClassName="com.mysql.j
dbc.Driver"
              url="jdbc:mysql://localhost:3306/javatest"
                    factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"/>
```

Veja mais em http://tomcat.apache.org/tomcat-8.0-doc/jdbc-pool.html

—

9 Realms

Não existe (ainda) na especificação Servlet nem na especificação Java EE uma API portátil e unificada para segurança. As APIs de segurança do Java EE apenas tratam de atribuição de roles (autorização) e oferecem uma interface para serviços nativos de autenticação.

Realms (base de dados com informações de usuários e roles/grupos) não fazem parte das especificações do Java EE. O Tomcat oferece uma interface Realm que pode ser implementada por plugins para diferentes implementações dessa base de dados. O nome do Realm é uma classe do pacote `org.apache.catalina.*` :

- `JDBCRealm` - dados de autenticação em banco de dados, acessível via driver JDBC
- `DataSourceRealm` - dados de autenticação em banco de dados, acessível via JNDI JDBC DataSource.
- `JNDIRealm` - dados de autenticação em servidor LDAP acessado via JNDI.
- `UserDatabaseRealm` - dados de autenticação em resource JNDI UserDatabase (tipicamente conf/tomcat-users.xml).
- `MemoryRealm` - dados de autenticação na memória, inicializado a partir de arquivo XML (conf/tomcat-users.xml).
- `JAASRealm` - dados de autenticação via JAAS.

Configuração de um Realm

Elemento `<Realm>` pode ser usado dentro de um `<Engine>` , `<Host>` ou `<Context>` . O escopo do Realm será limitado pelo escopo do elemento (se definido no Engine, valerá para todos os Hosts, a menos que seja redefinido localmente no Host, e assim por diante.)

A configuração requer pelo menos o nome da classe, e atributos opcionais, dependendo do tipo de Realm usado:

```
<Realm className="org.apache.catalina.NOME-DO-REALM" .../>
```

Por exemplo, o JDBCRealm requer a existência de uma tabela de usuários e outra de roles, e atributos com driver JDBC, URL JDBC, dados de autenticação, nome da tabela de usuários, nome da tabela de roles, nome das colunas, etc.

Exemplo DataSourceRealm

DB:

```
DROP DATABASE IF EXISTS tomcat_realm;
```

```
CREATE DATABASE tomcat_realm;
USE tomcat_realm;

CREATE TABLE tomcat_users (
    user_name varchar(20) NOT NULL PRIMARY KEY,
    password varchar(250) NOT NULL
);
CREATE TABLE tomcat_roles (
    user_name varchar(20) NOT NULL,
    role_name varchar(20) NOT NULL,
    PRIMARY KEY (user_name, role_name)
);

INSERT INTO tomcat_users (user_name, password) VALUES  ('someuser', '5f4dcc3b5aa765
d61d8327deb882cf99');
INSERT INTO tomcat_roles (user_name, role_name) VALUES ('someuser', 'authenticated'
);
COMMIT;
```

conf/server.xml (dentro de `<Context>` ou elemento mais genérico):

```
<Realm className="org.apache.catalina.realm.DataSourceRealm"
   dataSourceName="jdbc/auth"
   digest="MD5"
   userTable="tomcat_users" userNameCol="user_name"     userCredCol="password"
   userRoleTable="tomcat_roles" roleNameCol="role_name"/>
```

META-INF/context.xml (dentro de `<Context>`):

```
<Resource name="jdbc/auth" auth="Container" type="javax.sql.DataSource"
   maxActive="100" maxIdle="30" maxWait="10000"
   username="realm_access" password="password"    driverClassName="com.mysql.jdbc.D
river"
   url="jdbc:mysql://localhost:3306/tomcat_realm"/>
```

Gerando uma senha

Em vez de guardar uma senha não-criptografada, pode-se usar a ferramenta `digest` e gerar um hash da

senha. O comando:

```
$ digest.sh -a MD5 senha
```

Retorna um hash MD5 da palavra “senha”. Armazene o hash no banco (em vez da palavra não-criptografada) e configure o realm com o atributo `digest="MD5"`, assim quando a senha for digitada ela será convertida em MD5 antes de ser enviada.

Outras opções disponíveis de hash são SHA e MD2.

Usando um Realm

Uma vez configurado o Realm, ele pode ser usado para configurar autenticação de maneira declarativa no web.xml, ou de maneira programática, usando métodos `login()` ou `authenticate()` da API Servlet.

Exemplo de uso de um Realm

O web.xml abaixo está configurado para usar autenticação BASIC e a base de dados de autenticação como realm. O Realm usado será o que foi declarado para o Host/Context:

```
 <web-app ...>
...
    <security-constraint>
        <web-resource-collection>
            <web-resource-name>Digite someuser/password</web-resource-name>
            <url-pattern>/*</url-pattern>
            <http-method>GET</http-method>
            <http-method>POST</http-method>
            <http-method>DELETE</http-method>
            <http-method>PUT</http-method>
        </web-resource-collection>
        <auth-constraint>
            <role-name>authenticated</role-name>
        </auth-constraint>
    </security-constraint>

    <security-role>
        <role-name>authenticated</role-name>
```

```
    </security-role>

    <login-config>
        <auth-method>BASIC</auth-method>
        <realm-name>Fazer login em Tomcat Realm</realm-name>
    </login-config>
</web-app>
```

A tentativa de acessar o contexto irá enviar para o cliente um cabeçalho HTTP Authenticate, que levará o browser a verificar se possui as credenciais de autenticação para o role `authenticated`. Se não tiver, ele mostrará uma janela onde o usuário poderá digitar os dados.

Outros exemplos de autenticação Java EE

Veja exemplos adicionais de autorização e autenticação.

—

10 Valves

Uma válvula (Valve) no Tomcat é um componente de interceptação de requisições. É inserido no pipeline de processamento de um container (pode ser usado no escopo de um Engine, Host ou Context).

Como usar uma válvula

Para usar uma válvula é preciso usar o elemento Valve e pelo menos o atributo className, que deve conter o nome completo de uma implementação da interface Valve:

```
<Valve className="org.apache.catalina.valves.AccessLogValve" ... />
```

Dependendo da Valve usada, outros atributos poderão ser obrigatórios (um AccessLogValve, por exemplo, normalmente requer o uso de atributos para prefixo de arquivo de log, sufixo, diretório e um pattern para formatar a saída). Veja a documentação em http://tomcat.apache.org/tomcat-8.0-doc/config/valve.html para detalhes sobre a configuração de cada tipo de Valve.

Válvulas disponíveis no Tomcat 8

Implementações nativas da interface Valve estão disponíveis no pacote org.apache.catalina.valves. O Tomcat 8 inclui válvulas para:

- Log de acesso: `AccessLogValve`, `ExtendedAccessLogValve`
- Controle de acesso: `RemoteAddressFilter`, `RemoteHostFilter`
- Suporte a proxies: `RemoteIpValve`, `SSLValve`
- Single Sing-On: `SingleSignOnValve`
- Autenticação: `BasicAuthenticatorValve`, `DigestAuthenticatorValve`, `FormAuthenticatorValve`, `SSLAuthenticatorValve`, `SPNEGOValve`
- Outros: `ErrorReportValve`, `CrawlerSessionManagerValve`, `StuckThreadDetectionValve`, `SemaphoreValve`

—

11 Conectores

Um conector (Connector) representa uma porta que o Tomcat usará para monitorar requisições do cliente. Conectores podem criados dentro de serviços (Service) e processadores (Engine) e roteiam requisições do cliente para serviços que possam processá-los. Conectores também podem ser usados para conectar o Tomcat a outros serviços externos, como por exemplo, um servidor Apache, e são essenciais no balanceamento de cargas.

A finalidade de um conector é de apenas monitorar uma porta a espera de uma requisição, e ao receber a requisição, passá-la para um processador (Engine), depois retornar os resultados para a porta. A sintaxe básica é:

```
<Connector port="porta-de-serviço" />
```

Como configurar hierarquias de conectores

O exemplo abaixo (mulesoft.com) ilustra uma configuração simplificada com dois conectores declarados:

```
<Server>
    <Service>
```

```
        <Connector port="8443"/>
        <Connector port="8444"/>

        <Engine>
             <Host name="yourhostname">
                <Context path="/webapp1"/>
                <Context path="/webapp2"/>
             </Host>
        </Engine>

    </Service>
</Server>
```

Os dois conectores acima operam no mesmo serviço, e cada serviço contém um processador (Engine). Ambos irão repassar as requisições para o mesmo processador, que passará para ambas as aplicações (Context). Cada requisição, portanto, irá provavelmente gerar duas respostas (uma de cada aplicação).

Para que cada conector conecte-se a um contexto distinto, é necessário ter serviços/processadores distintos. Para isto podemos alterar a hieraquia da forma abaixo, usando um conector para cada serviço:

```
<Server>
    <Service name="Catalina">
        <Connector port="8443"/>

        <Engine>
             <Host name="yourhostname">
                <Context path="/webapp1"/>
             </Host>
        </Engine>

    </Service>
    <Service name="Catalina-2">
        <Connector port="8444"/>

        <Engine>
             <Host name="yourhostname">
                <Context path="/webapp2"/>
             </Host>
        </Engine>
```

```
    </Service>
</Server>
```

Toda a configuração relacionada a conectores envolve configurações da hierarquia de Service/Engine/Host/Context.

Tipos de conectores

Há dois tipos de conectores em Tomcat:

- **HTTP** - conector que responde ao protocolo HTTP; possui atributos para diversas funções como redirecionamento e proxy. O tipo é identificado pelo atributo `protocol` que normalmente é `HTTP/1.1` (default, se ausente). Outro atributo importante é `SSLEnabled` que, se `true` ativa a camada SSL na requisição/resposta. Pode-se implementar load balancing com este conector (via `mod_proxy`), mas pé menos eficiente que AJP.
- **AJP** - é um conector que responde ao protocolo `AJP` - Apache JServ Protocol, que é uma versão otimizada do HTTP tipicamente usada para permitir que o Tomcat se comunique com um servidor Apache. Conectores AJP são geralmente implementados via plugin `mod_jk`, mais eficiente. Use `protocol="AJP/1.3"`

Os conectores também possuem implementações alternativas usando protocolos nativos (APR), NIO e BIO (Non-blocking e Blocking IO). Use o nome da classe que implementa o serviço no atributo `protocol`.

O componente chamado de Coyote usa um conector HTTP e oferece suporte nativo a HTTP. O componente chamado de Coyote JK, é um conector AJP e usa o protocolo JK para redirecionar o serviço.

Como conectar o Tomcat ao Apache Web Server

São bibliotecas nativas do Apache 2.4 os módulos:

- `mod_proxy_http`
- `mod_proxy_ajp`

A principal vantagem é que fazer parte da distribuição do Apache e não precisam ser instalados separadamente. Suportam tanto HTTP como AJP. Por outro lado, mod_proxy_http é pouco eficiente.

Para muitos casos, o uso de mod_proxy_ajp é uma solução boa, pois utiliza AJP (mais eficiente que HTTP) e não requer configuração adicional. Uma desvantagem é que é mais novo e ainda tem muitos bugs, e algumas limitações (como tamanho máximo de pacotes <8k).

Uma alternativa é `mod_jk`, que tem como desvantagem a necessidade de instalar e configurar

separadamente, mas é a solução mais estável no momento e recomendada pela documentação do Tomcat.

Existem vários outros módulos que podem funcionar mas não são mais suportados (mod_proxy, mod_jk2, mod_jserv, mod_webapp, etc.)

Uma discussão sobre quais módulos usar pode ser encontrada aqui: http://www.tomcatexpert.com/blog/2010/06/16/deciding-between-modjk-modproxyhttp-and-modproxyajp

Para qualquer tipo de conexão, é necessário ter o conector ativo no Tomcat. Verifique se o conector AJP está habilitado no server.xml (geralmente está habilitado por default na porta 8009):

```
<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
```

Em seguida instale um módulo do Apache.

Conector usando mod_proxy_ajp

Veja mais informações sobre este conector em http://httpd.apache.org/docs/current/mod/mod_proxy_ajp.html

Para usar um módulo nativo do Apache, como o `mod_proxy_ajp` é preciso apenas configurar no httpd.conf do Apache. Verifique que os módulos estejam habilitados (exemplo no MacOS):

```
LoadModule proxy_module libexec/apache2/mod_proxy.so
LoadModule proxy_ajp_module libexec/apache2/mod_proxy_ajp.so
```

No final do arquivo httpd.conf há uma linha para incluir configurações externas (se não houver, crie):

```
Include /private/etc/apache2/other/*.conf
```

Crie na pasta `other` um arquivo `ajp.conf` onde iremos colocar a configuração do AJP. Neste arquivo, configure o módulo:

```
ProxyRequests Off
<Proxy *>
    Order deny,allow
    Deny from all
    Allow from localhost
</Proxy>
ProxyPass                /     ajp://localhost:8009/
```

```
ProxyPassReverse    /    ajp://localhost:8009/
```

O caminho `/` redireciona todas as chamadas para o Tomcat. Pode-se também restringir o redirecionamento a um contexto específico:

```
ProxyPass                /teste    ajp://localhost:8009/teste
ProxyPassReverse    /teste    ajp://localhost:8009/teste
```

Reinicie o servidor Apache (`sudo httpd -k restart` ou via serviço) e teste a conexão.

Conector usando mod_jk

O `mod_jk` não é nativo do Apache (precisa ser baixado, compilado e instalado separadamente), mas tem mais recursos e mais estável em plataformas Linux, além de ser mais fácil de instalar (via `apt-get`). Por outro lado, apresenta bugs em Mac OS 10.11, onde precisa ser baixado, compilado e instalado manualmente.

O mod_jk é configurado através de um arquivo `workers.properties` que define o protocolo usado, porta e host para conexão. O arquivo abaixo possui uma configuração mínima para a conexão:

```
worker.list=servidor
worker.servidor.type=ajp13
worker.servidor.host=localhost
worker.servidor.port=8009
```

Grave o `workers.properties` na pasta `/etc/apache2/other/` e crie um arquivo `jk.conf` para configurar o módulo:

```
# Carregar o módulo
LoadModule jk_module libexec/apache2/mod_jk.so

# Workers e arquivos compartilhados
JkWorkersFile /etc/apache2/other/workers.properties
JkShmFile /var/log/apache2/mod_jk.shm

# Config de log
JkLogFile /var/log/apache2/mod_jk.log
JkLogLevel info
JkLogStampFormat "[%a %b %d %H:%M:%S %Y] "
```

```
JkMount /* servidor
```

Reinicie o servidor Apache (`sudo httpd -k restart` ou via serviço) e teste a conexão.

—

12 Logging

O Tomcat usa internamente o Apache Commons Logging modificado em um pacote chamado JULI. É possível configurar o Tomcat para usar o Log4J em vez do JULI.

Através do JULI qualquer aplicação pode usar a API java.util.logging ou a API da especificação Servlet de forma nativa.

Console

Uma instalação default do Tomcat redireciona a saída do console para o arquivo catalina.out. Tudo o que for gravado em System.err ou System.out será redirecionado para este arquivo.

Log de acesso

Logs de acesso no Tomcat são implementados como uma Válvula. Existem válvulas específicas para interceptar requisições e logar nos arquivos de acesso. Os nomes dos arquivos, prefixos e sufixos são definidos na configuração de cada Valve.

Veja mais informações sobre JULI em http://tomcat.apache.org/tomcat-8.0-doc/logging.html

—

13 Load balancing

Para balancear a carga de duas instâncias rodando em hosts diferentes, é preciso garantir que cada instância possua um conector de redirecionamento ativo e cada Engine possua um identificador que será usado para

balanceamento (atributo `jvmRoute`). O nome pode ser arbitrário. Por exemplo, para um servidor em node1.dominio.com poderia ser:

```
<Engine name="Catalina" defaultHost="localhost" jvmRoute="node1">
  ...
</Engine>
```

e para o servidor em node2.dominio.com:

```
<Engine name="Catalina" defaultHost="localhost" jvmRoute="node2">
  ...
</Engine>
```

Configuração usando Apache e mod_jk

Verifique que existe em cada instância um conector AJP para a porta 8009.

Veja seção sobre Conectores, para instalar e configurar o `mod_jk`. A configuração do plugin é semelhante, mas o arquivo workers.properties, que vai conter as regras de balanceamento, é diferente;

Faça uma cópia do arquivo `other/jk.conf` mostrado na seção sobre conectores, e depois troque a extensão de jk.conf para que ele não seja carregado pelo httpd.conf (ex: jk.conf-basic). Troque o nome do JkWorkersFile para balanced-workers.properties:

```
# Carregar o módulo
LoadModule jk_module libexec/apache2/mod_jk.so

# Workers e arquivos compartilhados
JkWorkersFile /etc/apache2/other/balanced-workers.properties
JkShmFile /var/log/apache2/mod_jk.shm

# Config de log
JkLogFile /var/log/apache2/mod_jk.log
JkLogLevel info
JkLogStampFormat "[%a %b %d %H:%M:%S %Y] "

JkMount /* servidor
```

Agora faça uma cópia de workers.properties com o nome de `balanced-workers.properties`, onde

iremos colocar as regras de balanceamento para dois servidores (`node1` e `node2`):

```
# servidor é o nome do load balancer
worker.list=servidor

#servidor 1
worker.node1.port=8009
worker.node1.type=ajp13
worker.node1.host=node1.dominio.com
worker.node1.lbfactor=1
worker.node1.cachesize=10

#servidor 2
worker.node2.port=8009
worker.node2.type=ajp13
worker.node2.host=node2.dominio.com
worker.node2.lbfactor=1
worker.node2.cachesize=10

# load balancer
worker.servidor.type=lb
worker.servidor.balance_workers=node1,node2
worker.servidor.sticky_session=1
```

A configuração acima realiza balanceamento de carga alternado (round-robin) entre dois servidores. O `lbfactor` é o peso de cada servidor, que aqui está igual. Ele pode ser usado para dar prioridade a um nó em detrimento de outro. Quanto maior o número em relação aos outros, mais requisições ele irá receber. Isto pode ser usado para diferenciar servidores com mais poder de processamento.

O atributo `cachesize` define o tamanho dos thread pools associados ao container (a quantidade de requisições concorrentes que ele irá enviar ao servidor). É importante que esse número não supere o número de threads configurado no Conector em server.xml.

A propriedade `sticky_session` garante que diferentes requisições para a mesma sessão sejam enviadas para o mesmo servidor. Se o valor for 0, isso não será respeitado e a sessão perderá requisições.

Reinicie o Apache para gravar as alterações.

Para testar, execute as instâncias, e envie requisições para o servidor Apache. As requisições devem ser processadas de forma alternada.

Configuração usando Apache e mod_proxy

Verifique que mod_proxy e mod_proxy_balancer estejam habilitados no Apache.

```
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_balancer_module modules/mod_proxy_balancer.so
```

Há três algoritmos de balanceamento que podem ser usados: byrequests, bytraffic e bybusiness através da propriedade `lbmethod`. A propriedade `stickysession` deve conter o nome do cookie. O fragmento abaixo ilustra como poder ser implementado um balanceamento simples entre dois hosts usando o `mod_proxy`:

```
<IfModule proxy_module>
    ProxyRequests Off
    ProxyPass / balancer://mycluster
    ProxyPassReverse / balancer://mycluster
    <Proxy balancer://mycluster>
         BalancerMember ajp://node1.domain.com:8009 route=node1 loadfactor=1
         BalancerMember ajp://node2.domain.com:8009 route=node2 loadfactor=1

         ProxySet lbmethod=bybusyness
         ProxySet stickysession=JSESSIONID

         Order Deny,Allow
         Deny from none
         Allow from all
    </Proxy>
 </IfModule>
```

Veja mais em https://httpd.apache.org/docs/2.4/mod/mod_proxy_balancer.html

Onde `contexto` é o nome do arquivo WAR, sem a extensão `.war`.

—

14 Clustering

A configuração de Cluster oferece muitas opções e é bastante complexa, mas a configuração default muitas vezes requer nenhuma ou poucas alterações para a maior parte dos casos.

O elemento `<Cluster>` pode ser colocado no `<Engine>` ou no `<Host>` .

```
<Engine>
...
    <Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/>
</Engine>
```

A configuração default, replica *todas* as sessões em *todas* as instâncias do cluster. Ideal para um cluster pequeno e quando a aplicação está *instalada em todos eles*.

Pode-se configurar as propriedades do cluster, através da configuração de sub-elementos de `<Cluster>` : `<Manager>` , `<Channel>` , `<Valve>` , `<Deployer>` , etc.

Componentes de Cluster

Cluster

`<Cluster>` é o elemento principal, dentro do qual os detalhes podem ser configurados.

Manager de sessões

A replicação da sessão é realizada pelo Manager. Por default, o Cluster define um `<Manager>` caso não tenha um declarado explicitamente usando a implementação `DeltaManager` . A configuração default é

```
<Manager className="org.apache.catalina.ha.session.DeltaManager"
                expireSessionsOnShutdown="false"
                notifyListenersOnReplication="true"/>
```

Esse manager replica a sessão para todos os nós do cluster. Funciona bem para clusters pequenos, mas em clusters grandes pode ser um problema. Replica inclusive para nós que não têm a aplicação que usará a sessão instalado. Para lidar com isso, deve-se usar o BackupManager:

```
<Manager className="org.apache.catalina.ha.session.BackupManager"
                expireSessionsOnShutdown="false"
                notifyListenersOnReplication="true"
```

```
                    mapSendOptions="6"/>
```

O `<Manager>` é o template definido para todas as aplicações Web marcadas como `<distributable/>` no seu web.xml.

Requisitos para replicação de sessões

- Garantir que todos os atributos da sessão sejam serializáveis (precisam implementar java.io.Serializable)
- web.xml deve ter elemento `<distributable />`
- Garantir que as URLs de uma sessão sejam idênticas (se houver qualquer diferença, uma nova sessão será criada, e não replicada)
- Incluir (descomentar) elemento `<Cluster>` em server.xml e configurar se necessário.
- Se usado com servidor Apache via mod_jk, é preciso configurar atributo jvmRoute:
  `<Engine name="Catalina" jvmRoute="tomcat_1" >`
  onde tomcat_1 é o nome do worker em `workers.properties`
- Se usado com balanceamento de cargas, configurar o balanceador no modo de Sticky Session.
- Se nós do cluster estiverem na mesma máquina, deve-se configurar `<Receiver>` para cada `<Channel>` dentro de `<Cluster>` usando um valor diferente para o atributo `port` .

Channel

Channel e seus sub componentes são parte da camada IO do grupo do cluster, o módulo de comunicação Apache Tribes. É um framework de mensageria de baixo acoplamento.

Channel gerencia um conjunto de subcomponentes que juntos criam um framework de mensageria usado internamente pelos componentes que precisam enviar mensagens entre diferentes instâncias.

```
<Channel className="org.apache.catalina.tribes.group.GroupChannel">
...
</Channel>
```

Channel contém quatro componentes importantes:

Membership - componente responsavel por automaticamende descobrir novos nós no cluster e prover notificações para nós que não deram sinal de vida. Implementação usa multicast. Pode-se dividir clusters em grupos que recebem heartbeats diferentes, mudando o endereço de multicast.

```
<Membership className="org.apache.catalina.tribes.membership.McastService"
                    address="228.0.0.4"
                    port="45564"
```

```
                    frequency="500"
                    dropTime="3000"/>
```

Sender - componente que gerencia todas as conexões de saída e mensagens de dados que são enviados pela rede de um nó para outro. Este componente permite que mensagens sejam enviadas em paralelo. Sender contém o componente **Transport**, que é a camada IO de mais baixo nível. A implementação default usa sockets non-blocking.

```
    <Sender className="org.apache.catalina.tribes.transport.ReplicationTransmitter"
>
        <Transport className="org.apache.catalina.tribes.transport.nio.PooledPara
llelSender"/>
    </Sender>
```

Receiver - monitora a chegada de mensagens enviadas por outros nós. Aqui pode-se configurar o thread pool do cluster, já que mensagens entrantes serão processadas mais rapidamente em um pool.

```
<Receiver className="org.apache.catalina.tribes.transport.nio.NioReceiver"
                address="auto"
                port="5000"
                selectorTimeout="100"
                maxThreads="6"/>
```

Interceptor - pode-se incluir interceptadores para controlar a maneira como as mensagens são enviadas e recebidas, dentre outras configurações.

```
 <Interceptor className="org.apache.catalina.tribes.group.interceptors.TcpFailureDe
tector"/>
 <Interceptor className="org.apache.catalina.tribes.group.interceptors.MessageDispa
tch15Interceptor"/>
 <Interceptor className="org.apache.catalina.tribes.group.interceptors.ThroughputIn
terceptor"/>
```

Valve

A implementação de Cluster em Tomcat usa Valves para monitorar quando as requisições entram e saem do container. As Valves usadas em clusters são implementações da interface ClusterValve e são três:

- `ReplicationValve` - notifica o cluster no final da requisição para que ele possa decidir se há

dados a serem replicados
- `JvmBinderValve` - se acontecer um failover de mod_jk, o valve irá substituir o atributo jvmWorker no ID da sessão, para que requisições futuras sejam direcionados a esse nó.
- `ClusterSingleSignOn` - suporta SSO no cluster. A identidade de segurança autenticada por uma aplicação web é reconhecida pelas outras aplicações web no mesmo host virtual e propagada para os outros nós do cluster.
-

Deployer

Este componente permide que aplicações sejam instaladas nos servidores participantes do cluster (Farm Deployer).

```
<Deployer className="org.apache.catalina.ha.deploy.FarmWarDeployer"
                tempDir="/tmp/war-temp/"
                deployDir="/tmp/war-deploy/"
                watchDir="/tmp/war-listen/"
                watchEnabled="false"/>
```

Cenário com cluster

Duas instâncias. Sequência de eventos. Veja http://tomcat.apache.org/tomcat-8.0-doc/cluster-howto.html

1. TomcatA starts up
2. TomcatB starts up (Wait that TomcatA start is complete)
3. TomcatA receives a request, a session S1 is created.
4. TomcatA crashes
5. TomcatB receives a request for session S1
6. TomcatA starts up
7. TomcatA receives a request, invalidate is called on the session (S1)
8. TomcatB receives a request, for a new session (S2)
9. TomcatA The session S2 expires due to inactivity.

Configuração de um cluster com mod_jk

https://www.mulesoft.com/tcat/tomcat-clustering
http://blog.c2b2.co.uk/2014/04/how-to-set-up-cluster-with-tomcat-8.html

—

15 Tuning & Security

http://blog.c2b2.co.uk/2014/05/tomcat-performance-monitoring-and-tuning.html

–

16 Referências

[1] Documentação oficial em tomcat.apache.org
[2] Vokotic and Goodwill. Tomcat 7. Apress, 2011.
[3] Brittain and Darwin. Tomcat: the definitive guide, 2nd edition. O'Reilly and Associates, 2008. (Tomcat 6)
[4] Laurie & Laurie. Apache: The Definitive Guide, 3rd Edition. Chap 18: mod_jserv and Tomcat. O'Reilly and Associates,