05

Especificação das Servlets

Transcrição

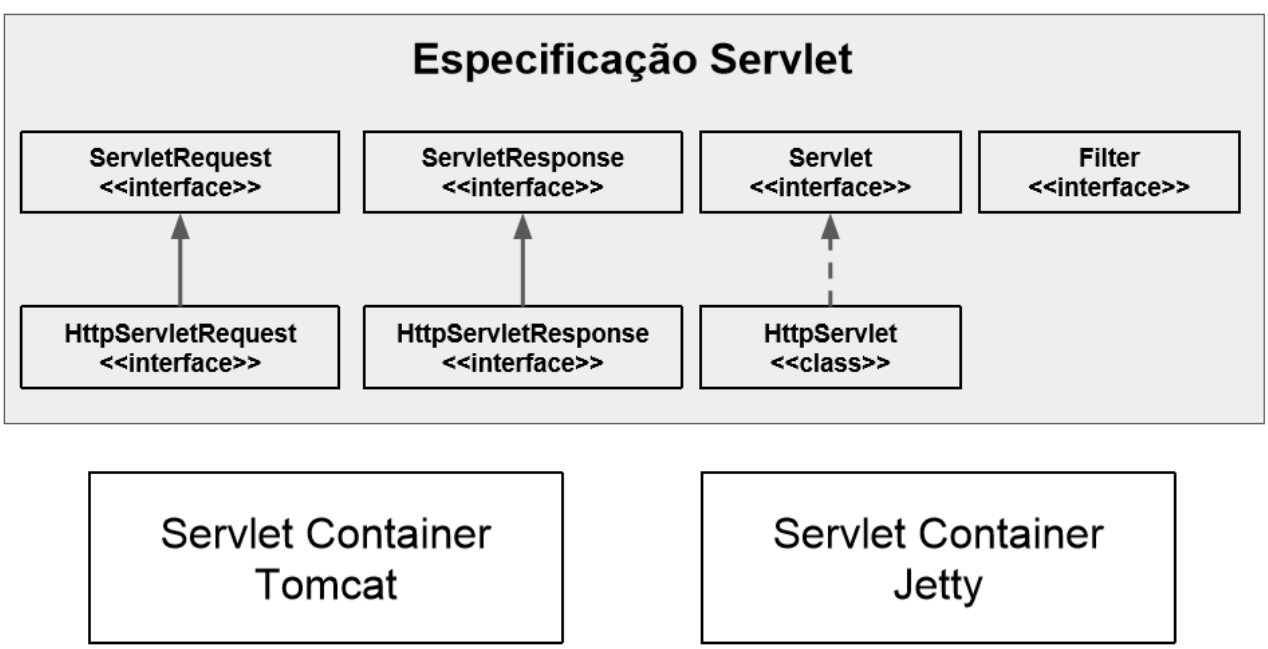
Olá! Anteriormente, nós conseguimos rodar nossa aplicação no Jetty, pois em nenhum momento amarramos nossa aplicação ao servidor Tomcat (ou mesmo ao próprio Jetty) - ou seja, nós programamos de maneira independente do servidor.

Isso quer dizer que nossa aplicação rodaria em Application Servers como Wildfly, Weblogic ou Websphere, já que todos eles entendem Servlets. A diferença é que eles têm diversos recursos que não precisamos para nossa aplicação.

O Tomcat e o Jetty são Servlet Containers, porque eles entendem Servlets.

Na nossa aplicação, nós usamos várias interfaces, como `Filter`, e a classe `HttpServlet`.

Nós não utilizamos diretamente a interface `Servlet`, pois estendemos a classe `HttpServlet` para implementá-la. Porém, usamos `ServletResponse` e `ServletRequest`, que faziam parte dos filtros, e as interfaces mais específicas `HttpServletRequest` e `HttpServletResponse`.



As interfaces não executam códigos, apenas definem quais métodos uma classe precisa ter. Portanto, alguém precisa implementar essas interfaces. Essa responsabilidade é do Servlet Container, que, através dos JARs, cria objetos que possuem os parâmetros da requisição e os dados da resposta.

Nossa aplicação não depende da implementação específica do servidor, apenas das interfaces e da classe genérica que criamos. Nós podemos provar isso no próprio projeto: as pastas "lib" do Tomcat e do Jetty possuem diversos JARs que nem mesmo sabemos para que servem, exceto um, chamado `servlet-api.jar`, no caso do Tomcat, ou `servlet-api-3.1.jar` no caso do Jetty.

É essa API que define as interfaces e a classe genérica `HttpServlet` - portanto, nossa aplicação só utiliza essa API. Dessa forma, conseguimos programá-la de forma independente, e poderíamos fazer o deploy em qualquer servidor.

Essa é a ideia do Java EE (hoje em dia também chamado de Jakarta EE): definir as especificações e deixar que o servidor faça a melhor implementação automaticamente, sem que a nossa aplicaçao dependa disso.

Tanto é que, no Eclipse, se removermos o Tomcat do nosso projeto, tudo que é específico do mundo Servlet (como as interfaces e as classes) não será mais

reconhecido. Porém, para resolvermos isso, basta adiconarmos `servlet-api.jar` - o que podemos fazer de maneira bem rudimentar, apenas para provarmos essa hipótese.

Para isso, clicaremos com o botão direito no projeto `gerenciador` e em seguida em "Build Path > Configure Build Path". Na tela seguinte, selecionaremos "Classpath", clicaremos em "Add External JARs" e então em `servlet-api.jar`. Agora basta clicarmos em "Apply and Close", e nossa aplicação automaticamente voltará a funcionar.

Nós podemos, inclusive, subir o Tomcat e testar nossa aplicação no navegador. Ou seja, no nosso "Build Path", tínhamos vários JARs que nem mesmo eram necessários.

O que queríamos provar é o significado de uma **especificação**: nós criamos algumas interfaces genéricas que alguém implementa - normalmente o servidor ou uma biblioteca que tem a funcionalidade que precisamos.

Agora já sabemos o que é uma especificação Servlet e a nossa aplicação está pronta. Nos vemos no próximo vídeo!