



DESENVOLVIMENTO JAVA WEB

Ariel da Silva Dias

DESENVOLVIMENTO JAVA WEB

1^a edição

Londrina
Editora e Distribuidora Educacional S.A.
2021

© 2021 por Editora e Distribuidora Educacional S.A.

Todos os direitos reservados. Nenhuma parte desta publicação poderá ser reproduzida ou transmitida de qualquer modo ou por qualquer outro meio, eletrônico ou mecânico, incluindo fotocópia, gravação ou qualquer outro tipo de sistema de armazenamento e transmissão de informação, sem prévia autorização, por escrito, da Editora e Distribuidora Educacional S.A.

Presidente

Rodrigo Galindo

Vice-Presidente de Pós-Graduação e Educação Continuada

Paulo de Tarso Pires de Moraes

Conselho Acadêmico

Carlos Roberto Pagani Junior
Camila Braga de Oliveira Higa
Carolina Yaly
Giani Vendramel de Oliveira
Gislaine Denisale Ferreira
Henrique Salustiano Silva
Mariana Gerardi Mello
Nirse Ruscheinsky Breternitz
Priscila Pereira Silva
Tayra Carolina Nascimento Aleixo

Coordenador

Henrique Salustiano Silva

Revisor

Rogério Colpani

Editorial

Alessandra Cristina Fahl
Beatriz Meloni Montefusco
Gilvânia Honório dos Santos
Mariana de Campos Barroso
Paola Andressa Machado Leal

Dados Internacionais de Catalogação na Publicação (CIP)

Dias, Ariel da Silva
D541d Desenvolvimento Java Web / Ariel da Silva Dias, –
Londrina: Editora e Distribuidora Educacional S.A., 2021.
44 p.

ISBN 978-65-5903-106-1

1. Java. 2. Desenvolvimento. 3. Tecnologia. I. Título.

CDD 005

Evelyn Moraes – CRB 010289/O

2021

Editora e Distribuidora Educacional S.A.
Avenida Paris, 675 – Parque Residencial João Piza
CEP: 86041-100 — Londrina — PR
e-mail: editora.educacional@kroton.com.br
Homepage: <http://www.kroton.com.br/>

DESENVOLVIMENTO JAVA WEB

SUMÁRIO

Introdução à programação Web	05
Servlets e Java Server Pages	21
JSF, arquiteturas MVC e introdução à persistência de dados	34
Hibernate e integração JSF e JPA	49

Introdução à programação Web

Autoria: Ariel da Silva Dias

Leitura crítica: Rogério Colpani



Objetivos

- Compreender os conceitos de back-end e front-end e a relação com à linguagem Java.
- Classificar as arquiteturas de servidores como servidores web e servidores de aplicação.
- Conhecer as principais APIs e tecnologias relacionadas a plataforma Java EE.



1. Programação web

Front-end e back-end são os dois termos usados no desenvolvimento da web, em que o primeiro é responsável por tudo aquilo que o usuário vê e interage; e, por outro lado, o segundo é tudo aquilo que traz a lógica e funcionalidade ao sistema web.

Esses termos são cruciais para o desenvolvimento da web, e são bem diferentes entre si. Apesar disso, cada lado precisa se comunicar e operar efetivamente com o outro, como uma única unidade para contribuir com a funcionalidade do site.

1.1 Front-end

O front-end de um site é o que pode-se ver e interagir em um navegador. Também chamado de client-side (lado do cliente), inclui tudo aquilo que o usuário experimenta diretamente como textos e cores, botões, imagens e menus de navegação.

Digamos que você decida iniciar um negócio. Uma padaria gourmet, por exemplo, e precisa de um site profissional para apresentar sua empresa aos clientes e informar sua localização. Talvez, você inclua algumas fotos e informações sobre seus produtos. Tudo que você precisa são tecnologias de front-end para criar seu site.

O desenvolvedor front-end, frequentemente se relacionará com as seguintes linguagens/tecnologias:

- **HTML (*HyperText Markup Language*):** apesar de comumente ser chamada de “linguagem de programação”, HTML é uma linguagem de marcação, pois utiliza marcas (as TAGs) para criar e organizar o conteúdo da web, de modo que este conteúdo possa ser exibido por um navegador.

- **CSS (Cascading Style Sheets):** é uma folha de estilo e que acompanha o HTML, responsável por definir o estilo do conteúdo de um site, como layout, cores, fontes etc.
- **JavaScript:** esta sim é uma linguagem de programação, a qual é usada para elementos mais interativos, como menus suspensos, janelas modais e formulários de contato. Vale destacar que a interação aqui ocorre do lado do cliente. O JavaScript não interage com o servidor!

Além destas tecnologias de front-end, você encontrará *frameworks*, como Bootstrap e Angular, além de bibliotecas JavaScript, como jQuery e extensões CSS como Sass e LESS. Há uma extensa lista de recursos como esses, que suportam HTML, CSS e JavaScript. O objetivo destes recursos é simplesmente tornar o código (e o processo de escrevê-lo) mais gerenciável e organizado.

Podemos concluir que o front-end diz respeito a um site estático, quando seu conteúdo não muda muito. Para sites estáticos, todas as informações necessárias que determinam o que está na página da web estão no próprio código de front-end. Os sites estáticos são bons para apresentar empresas, restaurantes, portfólios ou perfis profissionais. Mas, se você deseja transformar seu site em algo com o qual os usuários possam interagir, será necessário aprofundar o que está acontecendo nos bastidores do site.

1.2 Back-end

O back-end ou server-side (lado do servidor) é a parte do site que não se vê, responsável por armazenar e organizar dados, e garantir que tudo do lado do cliente realmente funcione. O back-end se comunica com o front-end, enviando e recebendo informações a serem exibidas como uma página da web.

Sempre que preenchemos um formulário de contato, digitamos um endereço da web ou fazemos uma compra (qualquer interação do usuário no lado do cliente), o navegador envia uma solicitação ao lado do servidor, que retorna informações na forma de código de front-end e que o navegador pode interpretar e exibir.

No exemplo da padaria, você (o proprietário) deseja adicionar uma funcionalidade de compra on-line. Deste modo, o cliente poderá comprar os produtos da padaria sem sair de casa. Para isso, seu novo site precisará ter componentes de back-end adicionais para torná-lo um aplicativo da web dinâmico – um site cujo conteúdo pode mudar com base no conteúdo do banco de dados e que pode ser modificado pela entrada do usuário. Isso é diferente de um site estático, que não requer um banco de dados poia seu conteúdo, geralmente, permanece o mesmo.

Dessa forma, o site da padaria precisa de um banco de dados para gerenciar todas as informações de clientes e produtos. Um banco de dados armazena o conteúdo do site em uma estrutura que facilita a recuperação, organização, edição e salvamento de dados. Ele roda em um computador remoto chamado servidor. Existem muitos bancos de dados diferentes amplamente utilizados, como MySQL, SQL Server, PostgreSQL e Oracle.

O site da padaria ainda conterá um código de front-end, mas também precisará ser criado usando uma linguagem que um banco de dados possa reconhecer. Algumas linguagens de back-end comuns são: Java (a linguagem foco de estudo desta disciplina e que será introduzida logo mais), Ruby, PHP, .Net e Python. Essas linguagens de programação geralmente são executadas em estruturas que simplificam o processo de desenvolvimento da web. Rails, por exemplo, é uma estrutura escrita em Ruby. O denominado Ruby on Rails é uma tecnologia popular para a criação de aplicativos dinâmicos da web que tornam o processo muito mais rápido.

Com todas essas partes funcionando juntas, corretamente, os clientes podem visitar o site e pesquisar o tipo específico de lanche que desejam comprar – talvez, desejem criar uma lista de guloseimas feitas especialmente para bebês. Quando os clientes digitam na caixa de pesquisa (no front-end), por exemplo, o aplicativo examina todos os dados do produto armazenados no banco de dados (back-end) e retorna as informações apropriadas na forma de código de front-end que o navegador exibe como a lista solicitada pelo usuário.

1.2.1 Linguagens de programação para o back-end

Apesar de trabalhar no back-end, o desenvolvedor do lado do servidor precisa conhecer o básico de HTML e CSS. Entretanto, a maior parte do seu trabalho será feita usando alguma linguagem de programação como Java, PHP, Node.js, Python e outras que podem ser usadas para codificação no servidor. Apesar das diferentes opções, é importante saber qual escolher.

O Quadro 1 apresenta uma relação das principais linguagens e tecnologias front-end, back-end e também os tipos de bancos de dados.

Quadro 1 – Linguagens, tecnologias back-end e front-end

Tecnologias front-end	Tecnologias back-end	Bancos de dados
HTML / HTML5	Java	MySQL
CSS / CSS3	Python	NoSQL
JavaScript	Node.JS	Oracle
AngularJS	PHP	
Ajax	C#	
Twitter Bootstrap		
VBScript		

Fonte: elaborado pelo autor.

Apesar de a linguagem Java ser o foco dos estudos, cabe aqui uma pequena definição de duas outras linguagens muito populares no desenvolvimento back-end: PHP e NodeJS.

PHP

Conforme descrito no site oficial, o PHP “é uma linguagem de script open source de uso geral, muito utilizada, e especialmente adequada para o desenvolvimento web e que pode ser embutida dentro do HTML” (PHP, 2020).

A diferença entre PHP e JavaScript está no fato de ser executado do lado do servidor. Apesar desta característica, o PHP pode ser utilizado em três áreas: scripts do lado do servidor, scripts de linha de comando e aplicações desktop.

Node.JS

Node.JS foi projetado para criar aplicativos escaláveis no back-end. De modo geral, o Node.JS permite executar o JavaScript no lado do servidor.

Aqui cabe uma consideração importante: Node.JS não é uma linguagem de programação, e também não é um *framework*, mas é definido como um interpretador assíncrono de JavaScript orientado a eventos.

A terceira linguagem é o foco de nossos estudos, trata-se do Java, e veremos sobre essa linguagem no subtópico a seguir.

Apresentadas estas linguagens e tecnologias, partiremos agora para o estudo da linguagem Java, mais precisamente o Java Enterprise Edition ou Java EE.

1.2.2 Java Enterprise Edition – História

Até este momento, estudamos os conceitos de front-end e back-end. Agora, iremos conhecer o Java Enterprise Edition, uma coleção de APIs Java de propriedade da Oracle (logo mais, será apresentada a atual detenção de propriedade) que os desenvolvedores de software podem usar para escrever aplicativos do lado do servidor.

Outras nomenclaturas para o Java Enterprise Edition que você já deve ter visto são: Java EE, Java 2EE, J2EE e agora Jakarta EE. Todos estes são nomes diferentes para a mesma ferramenta.

Originalmente, a Sun Microsystems projetou o Java EE como parte do JDK (*Java Development Kit*) principal da primeira versão da linguagem Java, em que o objetivo era simplificar o desenvolvimento de aplicativos web, diminuindo a necessidade de programação por meio do uso de componentes modulares e reutilizáveis.

Devido à popularidade, no ano de 1999, o JDK do Java foi dividido, e as extensões foram separadas, com isso, surgiu o J2EE ou Java 2 Platform Enterprise Edition. Em 2006, ocorreu o advento do Java 5 e, com isso, o J2EE foi renomeado para Java EE ou Java Platform Enterprise Edition.

O nome Java EE foi utilizado até 2017, quando a Oracle (detentora da linguagem Java) decidiu ceder os direitos do Java EE para a Eclipse Foundation, porém, a linguagem Java ainda é de propriedade da Oracle. Então, legalmente, a Eclipse Foundation teve que renomear o Java EE e, deste modo, 2019 a empresa rebatizou de Java EE para Jakarta EE.

O Jakarta EE possui o mesmo conjunto de especificações do Java EE, sem alterações em seus recursos. Assim, durante nossos estudos, sempre iremos nos referir sobre Java Enterprise Edition, afinal, por esse nome é que esta plataforma ficou conhecida e amplamente utilizada em todo mundo.

1.2.3 Java Enterprise Edition – Tecnologias básicas

No subtópico 1.2.2, foi dito que o Java EE fornece APIs ou serviços que simplificam os desafios mais comuns enfrentados pelos desenvolvedores. São 40 APIs do Java incluída no JEE 8. Essas APIs se enquadram em cinco principais categorias:

- **Tecnologias de aplicação web:** com destaque para Java Servlet e Java Server Pages (JSP), as quais fornecem métodos para atividades como capturar o que o usuário digitou em um campo de texto em um formulário ou armazenar um cookie no navegador do usuário.
- **Tecnologias de aplicativos empresariais:** com destaque para Enterprise JavaBeans (EJB), que simplifica a criação de serviços web ou componentes lógicos altamente escalonáveis, Java Message Service (JMS) e o Java Transaction (JTA) para interagir com sistemas externos e com o back-end.
- **Tecnologias de serviços da web:** com destaque para Java API for RESTful Web Services (JAX-RS) e também o XML-Based Web Services (JAX-WS), ambas para facilitar o desenvolvimento e implantação de REST e serviços web baseados em JSON (JavaScript Object Notation);
- **Tecnologias de gestão e segurança:** com destaque para Java Authentication Service Provider Interface for Containers (JASPI), que possibilita implementar a segurança Java EE customizada e gerenciar contêineres Java EE.
- **Especificações relacionadas ao Java EE em Java SE:** com destaque para Java Management Extensions (JMX), Java Database Connectivity, SOAP with Attachments API for Java (SAAJ).

As APIs listadas são apenas uma pequena amostra dos diversos componentes disponíveis para os desenvolvedores no Java EE. Como são

muitas APIs com diferentes métodos e aplicabilidades, é recomendado sempre consultar a documentação oficial no site dos desenvolvedores.

1.2.4 Servidor web

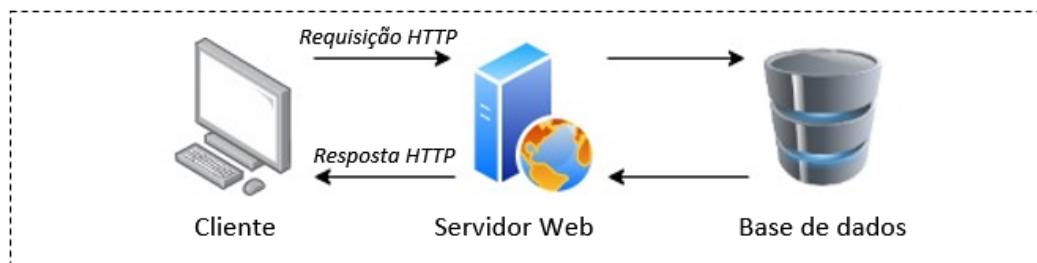
Denominamos servidores web ou servidores da web os softwares ou hardwares que usam o protocolo HTTP e outros para responder as requisições de clientes feitas pela *www (World Wide Web)*.

O software de servidor web controla como um usuário acessa os arquivos hospedados. Este software é acessado a partir dos nomes de domínio dos sites e garante a entrega do conteúdo do site ao usuário solicitante. Como hardware, um servidor web é um computador que mantém e gerencia o software de servidor web e outros arquivos relacionados a um site, como documentos HTML, CSS, imagens e arquivos JavaScript.

O hardware do servidor da web está conectado à internet, e permite a troca de dados com outros dispositivos conectados à mesma rede ou na internet.

Dessa forma, podemos dizer que o processo envolvido no servidor web é um exemplo do modelo de cliente/servidor. Todos os computadores que hospedam sites devem ter um software de servidor da web. Os principais servidores da web incluem Apache e Internet Information Server (IIS) da Microsoft. A Figura 1 apresenta a arquitetura padrão de um servidor web.

Figura 1 – Arquitetura de um servidor web



Fonte: elaborada pelo autor.

Observe na figura que o cliente, o qual pode ser um computador, smartphone ou uma aplicação web, realiza uma requisição a um programa no servidor web. Este programa, por sua vez, realiza uma consulta no banco de dados. Um exemplo prático, suponha realizar uma reserva de passagem aérea. Após agendar o dia, o horário e o destino, e pressionar o botão pesquisar, o sistema web cliente enviará uma requisição ao servidor web para que este verifique a disponibilidade de voos de acordo com as informações que você inseriu. Após a consulta, o programa no servidor web retorna uma página HTML (resposta em HTTP) com as informações desejadas.

Dentre os servidores web, temos:

- **Apache:** a Apache Server é um servidor web gratuito e de código aberto o qual fornece conteúdo da web pela internet. É comumente referido como Apache e, após o desenvolvimento, tornou-se rapidamente o cliente HTTP mais popular da web. Alguns dos módulos mais populares que podem ser adicionados são SSL (*Secure Sockets Layer*), *Server Side Programming Support* (PHP) e configurações de平衡amento de carga para lidar com grandes quantidades de tráfego. O Apache também pode ser implantado no Linux, MacOS e Windows. A única diferença são os caminhos de diretório e os processos de instalação.
- **Apache Tomcat:** também é um servidor web, entretanto, ele executa aplicações em Java ao invés de apenas sites estáticos em HTML.

- **Jetty:** trata-se de um servidor web desenvolvido pela Eclipse Foundation, que pode ser facilmente integrado em dispositivos, frameworks e servidores de aplicativos.
- **Internet Information Services (IIS):** o IIS é um software de servidor web desenvolvido pela Microsoft para o Windows Server. É usado para hospedar aplicativos web e muitos outros conteúdos voltados para a web. O IIS oferece muitos recursos, sendo uma ferramenta popular para todos os tipos de aplicações comerciais e não comerciais. O Microsoft IIS fornece interface gráfica com o usuário, de modo a facilitar a manutenção e gerenciamento da ferramenta.

Outros servidores da Web, incluem o servidor NetWare da Novell, o Google Web Server (GWS), o NGINX e a família de servidores Domino da IBM.

1.2.5 Servidor de aplicação

O servidor de aplicação expõe a lógica de negócios ao cliente, gerando um conteúdo dinâmico. Este tipo de servidor, aprimora as partes interativas de um site.

Para aplicativos web, um servidor de aplicativos será executado no mesmo ambiente que seu servidor web, e os servidores de aplicativos servem de apoio para a construção de páginas dinâmicas e implementar serviços como *clustering*, *failover* e balanceamento de carga. Portanto, os desenvolvedores podem se concentrar na implementação da lógica de negócios.

Uma diferença entre os servidores web e os servidores de aplicação é que, enquanto os servidores de aplicação têm suporte total para a especificação Java EE, os servidores web oferecem suporte a um pequeno subconjunto de funcionalidades. Observe esta relação no Quadro 2 com alguma das principais tecnologias de API do Java EE.

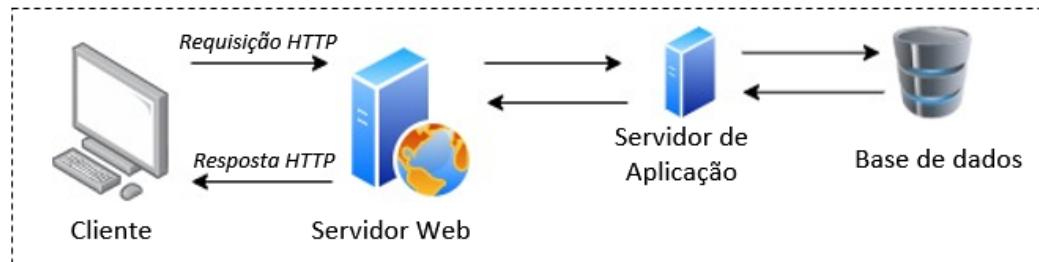
Quadro 2 – Relação comparativa das arquiteturas de servidores e o suporte às tecnologias de API

Servidor web	Servidor de aplicação
Java Servlets	Java Servlets
JSP	JSP
Expression Language	Expression Language
WebSocket	WebSocket
JTA	JTA
	Bean Validation
	EJB
	JPA
	JMS
	JavaMail

Fonte: elaborado pelo autor.

Outra diferença entre as arquiteturas está presente na Figura 2. Observe que além do servidor web, agora temos um servidor de aplicações, o qual é responsável por realizar a consulta no banco de dados.

Figura 2 – Arquitetura de um servidor web



Fonte: elaborada pelo autor.

Ao compararmos as figuras 1 e 2, podemos afirmar que um servidor web lida, principalmente, com o envio de HTML para exibição em um navegador web, sem ter acesso à lógica de negócios. Seu papel é receber requisições e atendê-las, enviando uma resposta usando o protocolo HTTP. Por outro lado, o servidor de aplicação fornece acesso à lógica de negócios para o uso (consumo) por outros programas.

Para melhor compreensão, observe um exemplo prático que caracteriza a diferença e as características de cada um dos tipos/arquiteturas de servidores. Vamos retomar o exemplo da aplicação web da padaria. Considere que a padaria fornece informações de preço e disponibilidade de produtos em tempo real em sua página web principal. Lembre-se que o site permite que o usuário acesse um formulário e adquira um produto para efetuar a compra on-line. Deste modo, quando o usuário/cliente no site envia uma consulta, o site executa uma pesquisa no servidor e retorna os resultados em uma página HTML. Esta implementação pode ser realizada de várias maneiras. Vejamos dois cenários: o cenário 1 teremos um servidor web sem um servidor de aplicativos, e o cenário 2 será com um servidor web com um servidor de aplicativos.

Cenário 1 – Servidor web sem servidor de aplicativos (Figura 1)

Neste cenário, temos um servidor web que trabalha sozinho fornecendo as funcionalidades do site da padaria. Logo, o servidor web recebe uma solicitação vinda do cliente e, em seguida, um programa no servidor trata esta solicitação que pode ser, por exemplo, atualizar a lista de preço e quantidade de produtos. O programa, então, procura os dados no banco de dados e, depois de recuperado, o programa do lado do servidor cria uma página HTML. Em seguida, o servidor web envia de volta ao navegador.

Note, então, que o servidor web recebe uma solicitação HTTP, a processa e, em seguida, responde com página HTML. Considere aqui que a padaria possui em seu estabelecimento físico uma caixa registradora (sistema de ponto de venda ou, simplesmente, sistema PDV) que necessita, constantemente, atualizar as informações de preço e quantidade de produtos. Este controle é realizado pelo Sistema PDV da padaria que está conectado ao servidor web.

Observe que este sistema precisa acessar diretamente o banco de dados no servidor web para obter as informações desejadas. Deste modo, a lógica desenvolvida no site da empresa (obter o valor e a quantidade dos produtos, além de efetuar vendas) e a lógica no sistema PDV que está no estabelecimento (obter o valor e a quantidade dos produtos, além de efetuar vendas) é a mesma. Então, temos aqui dois sistemas que executam a mesma ação.

Cenário 2 – Servidor web com servidor de aplicação (Figura 2)

No cenário 2, temos muitas características do cenário 1, uma vez que o servidor web ainda delega a geração de resposta em relação à solicitação de um script. Porém, agora é possível consultar os preços e a quantidade de produtos em um servidor de aplicação. Deste modo, ao invés de o programa do lado do servidor pesquisar os dados no banco e retornar uma resposta, o programa pode simplesmente chamar o serviço no servidor de aplicativos (um serviço chamado *ObterDadosValorQuantidade*, por exemplo), obter os dados e, depois, realizar o retorno como no cenário 1.

Neste cenário, o servidor de aplicações atende a lógica de negócios para pesquisar as informações de preço e quantidade de produtos. Esta funcionalidade (do servidor de aplicações) não diz nada sobre a exibição ou como o cliente deve usar as informações, ele simplesmente retorna os dados que obteve do banco. Em vez disso, o cliente e o servidor de aplicativos apenas enviam dados de um lado para outro.

Mas qual a vantagem? Aqui temos a reutilização de código. Observe que, ao separar a lógica que retorna os dados do banco e a geração de resposta em HTML, a lógica de obter os preços e quantidade de produtos torna-se reutilizável. Em outras palavras, o segundo cliente, que pode ser o sistema PDV no estabelecimento da padaria, pode realizar a solicitação para o mesmo serviço que o usuário online está requisitando. Observe, então, que dois sistemas diferentes, o sistema

web e o sistema PDV, consomem o mesmo serviço que é obter os dados sobre os produtos.

Resumidamente, podemos afirmar que, ao utilizar o servidor de aplicação, tanto o sistema web quanto o sistema PDV chamam o serviço de obter dados dos produtos como você chamaria um método em um objeto em programação orientada a objetos, ou como você chamaria uma função em programação procedural.

Dentre os servidores de aplicação, temos:

- **Apache TomEE:** este servidor de aplicações é desenvolvido com base no Apache Tomcat. Ele permite que possamos usar alguns dos recursos do Java EE que não são suportados diretamente pelo Tomcat.
- **WebSphere:** assim como a Microsoft desenvolveu um servidor web, a IBM desenvolveu um servidor de aplicações chamado WebSphere. Diferentemente do Apache TomEE, o WebSphere não é um projeto de código aberto.
- **GlassFish:** trata-se de um servidor de aplicações de código aberto que foi, inicialmente, desenvolvido pela Sun Microsystem para plataforma Java EE. Existem duas licenças do GlassFish: a CDDL (Licença de Desenvolvimento e Distribuição Comum) e a GPL (Licença Pública Geral GNU).

Destes servidores, o destaque fica para o GlassFish e para o Apache TomCat. O primeiro, muitas vezes é considerado como a implementação de referência do Java EE, o qual oferece suporte ao EJB, que é uma arquitetura de componentes do lado do servidor gerenciada para construção modular de aplicativos empresariais, ao JMS, ao JavaServer Faces, ao JSP, ao Java Servlet, entre outros.

Portanto, o GlassFish permite a criação de aplicativos corporativos que são portáveis e escaláveis, e que se integram facilmente com tecnologias legadas.

Do mesmo modo, o Apache TomCat implementa várias especificações do Java EE, incluindo Java Servlet, JSP e WebSocket. É considerado um servidor web *Java Puro*, pois foi desenvolvido inicialmente para executar apenas código Java.

No início deste tema, tivemos uma introdução aos conceitos de back-end e front-end. Você viu que o back-end é um conceito que representa tudo aquilo que o usuário não vê, ou seja, é a lógica que está por trás da interface. Logo, é justamente no back-end que vamos trabalhar utilizando o Java EE que, como visto, é uma plataforma com um conjunto de APIs, cujo objetivo é agilizar o trabalho do desenvolvedor. Por fim, conhecemos que, para executarmos uma aplicação no back-end, é necessário um servidor. Temos duas arquiteturas principais que são os servidores de aplicação e os servidores web, cada um possui suas características e funcionalidades.

Referências

DEITEL, Paul J; DEITEL, Harvey M. **Java: Como Programar.** 10. ed. São Paulo: Pearson, 2016.

DEITEL, Paul J; DEITEL, Harvey M. A. **Rich Internet Applications e Desenvolvimento Web para Programadores.** São Paulo: Pearson Prentice Hall, 2009.

PHP. **O que é o PHP?** Disponível em: https://www.php.net/manual/pt_BR/intro-whatis.php. Acesso em: 5 jan. 2020.

SIERRA, Kathy; BATES, Bert. **Use a cabeça!: Java.** Rio de Janeiro: Alta Books, 2009.

Servlets e Java Server Pages

Autoria: Ariel da Silva Dias

Leitura crítica: Rogério Colpani

Objetivos

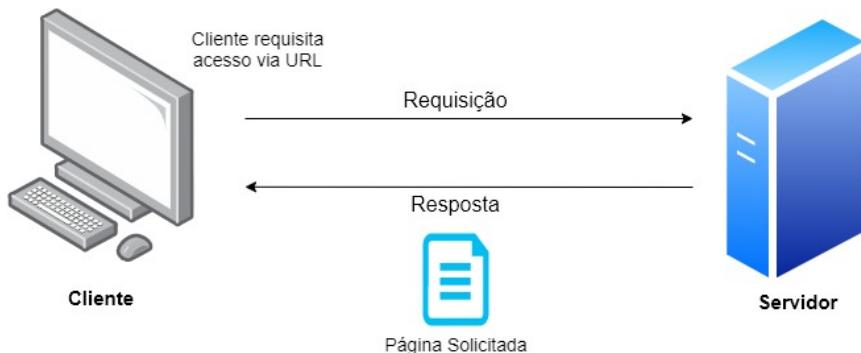
- Compreender o desenvolvimento de aplicações web utilizando a linguagem de programação Java.
- Compreender o conceito de Servlets e como aplicá-lo em uma aplicação web.
- Compreender o conceito de Java Server Pages e como aplicar esta tecnologia em uma aplicação web.

1. Aplicação Java Web

A web consiste em um grande número de clientes e servidores conectados por meio de fios e redes sem fio. Os clientes web fazem solicitações ao servidor web. O servidor, por sua vez, recebe a solicitação, encontra os recursos e retorna a resposta ao cliente. Quando um servidor responde a uma solicitação, ele geralmente envia algum tipo de conteúdo ao cliente.

O cliente usa o navegador da web para enviar a solicitação ao servidor. O servidor, então, envia uma resposta ao navegador com um conjunto de instruções escritas em HTML (*HyperText Markup Language*). Todos os navegadores sabem como exibir a página HTML para o cliente. A Figura 1 apresenta a arquitetura cliente servidor.

Figura 1 – Arquitetura cliente servidor



Fonte: elaborada pelo autor.

O servidor web é um software que pode processar a solicitação do cliente e enviar a resposta de volta (como na Figura 1). Por exemplo, o Apache é um dos servidores web mais usados. O servidor web é executado em alguma máquina física e escuta a solicitação do cliente em uma porta específica.

Um cliente web, por sua vez, é um software que ajuda na comunicação com o servidor. Alguns dos clientes da web mais usados são Firefox, Google Chrome e Safari. Quando solicitamos algo do servidor (por meio

da URL), o cliente web se encarrega de criar uma solicitação e enviá-la para o servidor e, em seguida, analisa a resposta do servidor e apresenta ao usuário.

A resposta é a composição de um site que, em sua maioria, é uma coleção de arquivos estáticos, como arquivos em HTML (página web), imagens, gráficos, entre outros. A web foi criada exatamente com este propósito: apresentar conteúdo estático. Por outro lado, um aplicativo web é um site com funcionalidade dinâmica no servidor. Alguns exemplos de aplicações web são o Google, Facebook, Twitter, YouTube entre outros.

Os servidores web são bons para páginas HTML de conteúdo estático, mas eles não sabem como gerar conteúdo dinâmico ou como salvar dados em bancos de dados. Por isso, precisamos de outra ferramenta que possamos usar para gerar conteúdo dinâmico. Existem várias linguagens de programação para conteúdo dinâmico, como PHP, Python, Ruby on Rails, Java Servlets e JSPs.

Uma aplicação Java Web é usada para criar sites dinâmicos, tendo em vista que o Java fornece suporte para aplicativos da web por meio de Servlets e JavaServer Pages (JSP). Podemos criar um site com páginas HTML estáticas, mas para informações dinâmicas, precisamos de uma aplicação web.

Java Servlet e JSPs são tecnologias do lado do servidor para estender a capacidade dos servidores da web, fornecendo suporte para resposta dinâmica e persistência de dados.

1.1 Servlets

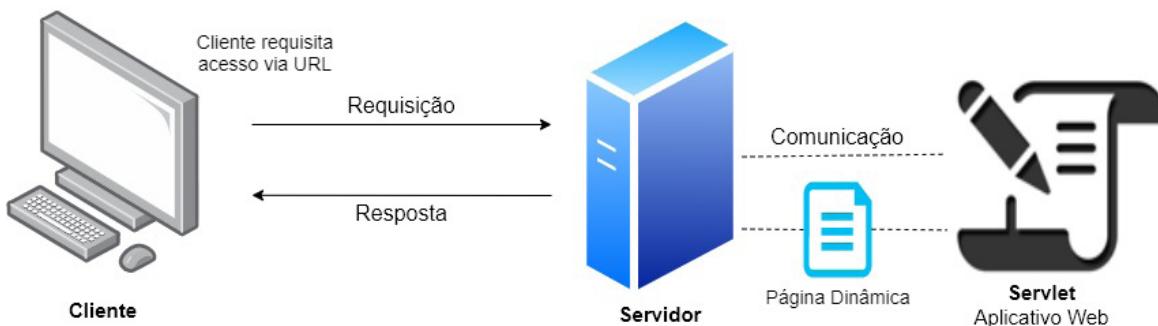
A tecnologia Servlet é baseada em Java EE, e é usada para criar aplicativos da Web Java. Os pacotes javax.servlet e javax.servlet.http

fornecem as interfaces e classes necessárias para que possamos escrever nossos próprios servlets.

Os aplicativos web são aplicativos auxiliares que residem no servidor da web, e são acessados usando o protocolo HTTP (*Hyper Text Transfer Protocol*) e criam páginas da web dinâmicas. Uma página dinâmica pode ser qualquer coisa, como uma página que escolhe aleatoriamente uma imagem para ser exibida.

Como os aplicativos web utilizam o protocolo HTTP, precisamos dos métodos doGet() e doPost() da classe HttpServlet para lidar com serviços específicos HTTP. A Figura 2 apresenta a arquitetura cliente-servidor com servlet.

Figura 2 – Arquitetura cliente servidor com servlet



Fonte: elaborada pelo autor.

O servidor e o cliente web são dois softwares separados, portanto, deve haver alguma linguagem comum para comunicação, neste caso é o HTML. Para que isso aconteça, é preciso um protocolo de comunicação comum, e o HTTP é o protocolo de comunicação entre o servidor e o cliente. Assim, o HTTP é executado em cima do protocolo de comunicação TCP/IP.

Algumas das partes importantes da solicitação HTTP são:

- **Método HTTP:** ação a ser executada, geralmente GET, POST, PUT entre outros.

- **URL:** página para acessar
- **Parâmetros de formulário:** semelhantes aos argumentos em um método Java, por exemplo, usuário, detalhes de senha da página de login.

Algumas das partes importantes da resposta HTTP são:

- **Código de status:** um número inteiro para indicar se a solicitação foi bem-sucedida ou não. Alguns dos códigos de status mais conhecidos são 200 para “sucesso”, 404 para “não encontrado” e 403 para “acesso proibido”.
- **Tipo de conteúdo:** texto, HTML, imagem, PDF, entre outros.
- **Conteúdo:** dados reais que são renderizados pelo cliente e mostrados ao usuário.

Deste modo, podemos definir um Servlet como uma classe que lida com solicitações, as processa e responde com uma resposta ao cliente. Por exemplo, podemos usar um Servlet para coletar a entrada de um usuário por meio de um formulário HTML, consultar registros em um banco de dados e criar páginas da web dinamicamente.

Os servlets estão sob o controle de outro aplicativo Java denominado Container de Servlet. Quando um aplicativo em execução em um servidor da web recebe uma solicitação, o servidor entrega a solicitação ao Servlet Container que, por sua vez, a passa para o Servlet de destino.

1.1.1 Criando um Servlet

Para nossos projetos, utilizaremos o ambiente de desenvolvimento Eclipse. Além disso, como o Servlet é uma tecnologia back-end, ou seja, é executado do lado do servidor, precisamos de um servidor web como o Apache Tomcat instalado.

Antes de tudo é necessário criar um projeto no Eclipse do tipo dinâmico (*New Dynamic Web Project*) no menu “File”. Em seguida, nomeie seu projeto como *PrimeiroServlet* e clique em “Finalizar”.

Neste momento, será criado um projeto com os principais arquivos, porém, será preciso criar o Servlet, o qual ainda não foi criado. Então, retorne ao “File” e, após clicar em “New”, escolha a opção “Servlet”. Com isso, abrirá uma janela para inserir as informações no seu Servlet. No campo “*Class name*” insira o valor *PrimeiroServlet* (nossa classe terá o mesmo nome do projeto). Em seguida, pressione o botão “Finalizar”.

Ao finalizar, será criado um esqueleto do nosso Servlet e, deste modo, não precisaremos digitar todos os métodos e importações. A Figura 3 apresenta um exemplo deste código inicial.

Figura 3 – Código inicial de um servlet

```

1@ import java.io.IOException;
2 import javax.servlet.ServletException;
3 import javax.servlet.annotation.WebServlet;
4 import javax.servlet.http.HttpServlet;
5 import javax.servlet.http.HttpServletRequest;
6 import javax.servlet.http.HttpServletResponse;
7
8 @WebServlet("/PrimeiroServlet")
9 public class PrimeiroServlet extends HttpServlet {
10     private static final long serialVersionUID = 1L;
11
12     public PrimeiroServlet() {
13         super();
14         // TODO Auto-generated constructor stub
15     }
16
17     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
18         // TODO Auto-generated method stub
19         response.getWriter().append("Served at: ").append(request.getContextPath());
20     }
21
22     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
23         // TODO Auto-generated method stub
24         doGet(request, response);
25     }
26 }
```

Fonte: elaborada pelo autor.

Observe a linha 17 o método *doGet*. Este é um método da classe abstrata *HTTPServlet*, usado para suportar solicitações HTTP do tipo GET. Na linha 22 temos o método *doPost*, também pertencente a classe *HTTPServlet*, e é usado para solicitações do tipo HTTP PUT.

Para testarmos nosso primeiro servlet, é preciso adicionar algum HTML com código de dados dinâmico no método doGet. No código da Figura 3, adicione o trecho em destaque na Figura 4.

Figura 4 – Código inicial de um servlet

```

20
21 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
22     // TODO Auto-generated method stub
23     response.getWriter().append("Served at: ").append(request.getContextPath());
24
25     PrintWriter out = response.getWriter();
26     Date data = new Date();
27
28     out.println("<h1>Olá Mundo!</h1> <br>");
29     out.println("<p>Data: " + data.getTime() + "</p>");
}

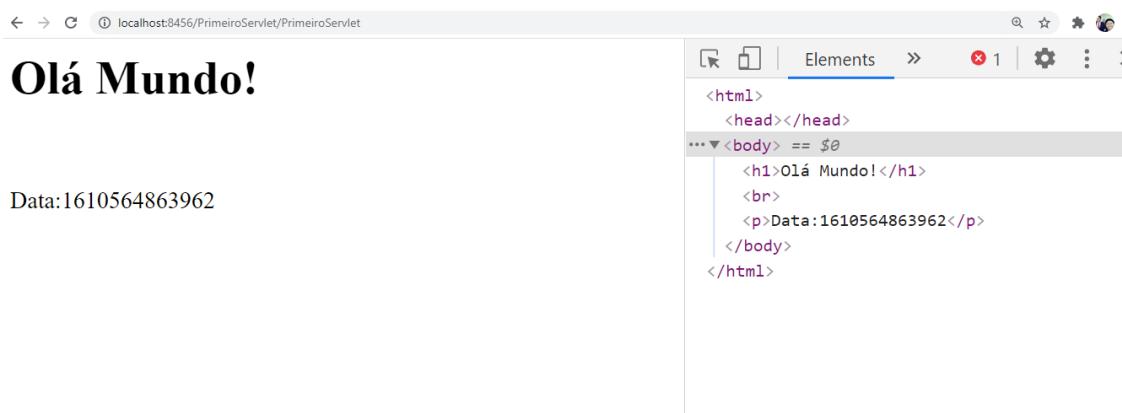
```

Fonte: elaborada pelo autor.

Para testarmos nosso primeiro servlet, vá em Run >> Run As >> Run on Server. Na janela que abrirá, selecione o servidor Tomcat instalado. Após isso, será aberto um navegador no Eclipse e obteremos uma página web que irá mostrar o número de milissegundos desde 1 de janeiro de 1970. Para testar o dinamismo da página, pressione F5 no teclado para atualizar o navegador do Eclipse.

Também é possível abrir a aplicação em outro navegador. Para isso, acesse o navegador que o Eclipse abriu internamente e copie a URL apresentada. Em seguida, cole esta URL em seu navegador preferido, como na Figura 5.

Figura 5 – Navegador web com conteúdo do Servlet



Fonte: elaborado pelo autor.

Observe na Figura 5, lado esquerdo, que o conteúdo que programamos no servlet é apresentado. Observe também que o servlet é usado para gerar HTML e enviá-lo em resposta, se você olhar para a implementação de doGet(), na verdade estamos criando um documento HTML, escrevendo-o no objeto PrintWriter de resposta e adicionando informações dinâmicas onde precisamos. Veja que no depurador da página (F12) não consta o código do servlet, mas sim, o conteúdo HTML apenas.

1.1.2 Segundo exemplo: formulário de entrada

Neste momento, observe outro exemplo. Desta vez, utilizaremos os métodos doGet() e doPost(). No método doGet(), iremos criar um formulário para o usuário informar o nome. Também haverá um botão. Veja o exemplo do método doGet() na Figura 6 – inserido apenas o método doGet e o conteúdo.

Figura 6 – Código fonte do método doGet()

```

1.21  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
1.22      response.setContentType("text/html");
1.23      response.setCharacterEncoding("UTF-8");
1.24
1.25      PrintWriter out = response.getWriter();
1.26      out.append("<!DOCTYPE html>\\r\\n");
1.27      .append("<html>\\r\\n")
1.28      .append("    <head>\\r\\n")
1.29      .append("        <title>Formulário de Entrada</title>\\r\\n")
1.30      .append("    </head>\\r\\n")
1.31      .append("    <body>\\r\\n")
1.32      .append("        <form action=\"PrimeiroServlet\" method=\"POST\">\\r\\n")
1.33      .append("            Insira seu nome aqui: \\r\\n")
1.34      .append("            <input type=\"text\" name=\"usuario\" />\\r\\n")
1.35      .append("            <input type=\"submit\" value=\"Enviar\" />\\r\\n")
1.36      .append("        </form>\\r\\n")
1.37      .append("    </body>\\r\\n")
1.38      .append("</html>\\r\\n");
1.39  }

```

Fonte: elaborada pelo autor.

Veja no código da Figura 6 um formulário entre as linhas 32 e 36. Este formulário foi criado, assim como faríamos em HTML puro. Na linha 32, observe que o atributo action recebe o nome de nosso servlet. Isso ocorre, pois, ao clicar no botão “Enviar”, esta página será invocada novamente e chamará o método doPost da figura 7.

Figura 7 – Código fonte do método doPost()

```

41@ protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
42     String usuario = request.getParameter("user");
43
44     response.setContentType("text/html");
45     response.setCharacterEncoding("UTF-8");
46
47     PrintWriter out = response.getWriter();
48     out.append("<!DOCTYPE html>\r\n")
49         .append("<html>\r\n")
50             .append("      <head>\r\n")
51                 .append("          <title>Mensagem de boas vindas</title>\r\n")
52             .append("      </head>\r\n")
53             .append("      <body>\r\n");
54     if (usuario != null && !usuario.trim().isEmpty()) {
55         out.append(" Olá " + usuario + "!\r\n");
56         out.append(" Você realizou este exemplo com sucesso.\r\n");
57     } else {
58         out.append(" Você não informou um nome!\r\n");
59     }
60     out.append("      </body>\r\n")
61         .append("</html>\r\n");
62 }
```

Fonte: elaborada pelo autor.

Na Figura 7, temos o método doPost() o qual recebe o conteúdo digitado no formulário, neste caso, o nome informado na tela anterior. Observe que, novamente, entre as linhas 53 e 60, temos o corpo de nossa página HTML. Veja que na linha 54 temos uma instrução condicional *if* para validar se o usuário digitou algo no campo ou não. Se sim, a página a ser mostrada é o conteúdo das linhas 55 e 56, se não, o conteúdo apresentado ao usuário será a linha 58. Faça o teste!

1.2 Java Server Pages (JSP)

No subtópico anterior, vimos que o conceito de Java Servlet foi introduzido para gerar conteúdos web dinâmicos que são configurados de acordo com as solicitações dos usuários (por exemplo, em resposta a consultas e solicitações de pesquisa). Entretanto, é difícil usar um Servlet para produzir uma página HTML que seja apresentável e de qualidade por meio das instruções de programação out.println().

Ainda mais difícil é manter ou modificar a página HTML produzida, uma vez que os programadores que escreveram o servlet, podem não ser bons designers gráficos, enquanto um designer gráfico não entende de programação Java.

Java Server Pages (JSP) é uma tecnologia complementar ao Java Servlet que facilita a mistura de conteúdo dinâmico e estático da web. JSP é a resposta do Java às populares *Active Server Pages (ASP)* da Microsoft. JSP, assim como o ASP, fornece uma maneira elegante de misturar conteúdo estático e dinâmico.

Com o JSP, a página principal é escrita em HTML puro, enquanto tags especiais são fornecidas para inserir partes de códigos de programação Java – o mesmo Java puro que você já está familiarizado. A lógica de negócios (programação) e a apresentação são claramente separadas. Isso permite que os programadores se concentrem na lógica de negócios, enquanto o *webdesigner* se concentra na apresentação.

JSP é baseado em Servlet, ou seja, uma página JSP é traduzida internamente em um servlet Java. Então, podemos dizer que Servlet é HTML dentro do Java, enquanto JSP é Java dentro do HTML. Tudo o que você não pode fazer em servlet, não pode fazer em JSP. O JSP torna a criação e manutenção de páginas HTML dinâmicas muito mais fácil do que o servlet. Todavia, JSP tem como objetivo complementar o Servlet, não um substituto.

1.2.1 Criando um JSP

Assim como fizemos com o Servlet, começaremos criando um exemplo simples para que você conheça um pouco da sintaxe. Aqui, temos uma outra propriedade exclusiva do JSP: qualquer arquivo escrito em HTML pode ser convertido em arquivo JSP apenas mudando a extensão do arquivo de “.html” para “.jsp” – e ele funcionaria perfeitamente!

Como já estamos utilizando o Eclipse, continuaremos nele neste novo exemplo para implementar o código da Figura 8. Crie um projeto como já fizemos anteriormente, criando um Dynamic Web Project com o nome de PrimeiroJSP. Em seguida, vá novamente em File >> New >> JSP File e coloque o mesmo nome do projeto. O arquivo ficará no diretório Web Content no seu projeto.

O Eclipse já cria um esqueleto inicial, então, veja como fica o código na Figura 8 a seguir.

O que diferencia o JSP do HTML é a capacidade de usar o código Java dentro do HTML. Em JSP é possível incorporar código Java em HTML usando tags JSP. Por exemplo, execute o código da Figura 8. Observe que toda vez que você executá-lo, ele exibirá a hora atual. É isso que torna esse código dinâmico.

Após executar, verifique no depurador do navegador (pressionando F12 no teclado) que está disponível apenas o código HTML, sem nenhuma linha de código em Java.

Figura 8 – Exemplo de código fonte JSP

```

1 <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2   pageEncoding="ISO-8859-1"%>
3 <!DOCTYPE html>
4<%> <html>
5<%>   <head>
6     <meta charset="ISO-8859-1">
7     <title>Insert title here</title>
8   </head>
9   <body>
10
11     Olá Mundo! Aqui é um código HTML simples.
12
13<%>
14     int valor = 88;
15     out.println("Olá Mundo! Exemplo de código com JSP");
16
17
18     <h1> A hora atual é: <%= new java.util.Date() %> </h1>
19
20<%>
21     out.println("O valor é: " + valor);
22
23
24   </body>
25 </html>
```

Fonte: elaborada pelo autor.

Observe também na Figura 8 que, na linha 11, há um texto escrito em HTML. Mais abaixo, entre as linhas 13 e 16, existe um código em Java do JSP que são incluídos em tags especiais no formato <% %> (semelhante ao ASP).

Ainda analisando o código, observe que na linha 14, nós declaramos uma variável do tipo inteira recebendo o valor 88. Nós chamamos esta mesma variável na linha 21 e apresentamos o seu valor na tela.

1.3 Comparando JSP e Servlet

O JSP e os servlets são usados para gerar páginas dinâmicas na web. A principal diferença entre eles é que o servlet adiciona código HTML dentro do Java, enquanto JSP adiciona código Java dentro de HTML. Existem alguns outros pontos perceptíveis, apresentados no Quadro 1.

Quadro 1 – Comparativo entre Servlet e JSP

Servlet	JSP
Programa desenvolvido em Java que oferece suporte à tags HTML.	Código HTML que oferece suporte à programação Java.
Usado principalmente na camada de negócios (programação) de um aplicativo corporativo.	Usado na camada de apresentação de um aplicativo corporativo.
Criados e mantidos por desenvolvedores Java.	Criado e mantidos por desenvolvedores web.

Fonte: elaborado pelo autor.

Durante a leitura deste material, você pode conhecer um pouco sobre os servlets e também sobre o JSP. JavaServer Pages é uma tecnologia bem conhecida e versátil para o desenvolvimento de visualizações de aplicativos Java Web. Combinado com servlets, páginas JSP são muito poderosas e fornecem acesso à toda a gama de recursos Java.

Por fim, é importante para seu aprendizado que você replique os códigos apresentados nesta aula, de modo a praticá-los. Somente com prática, você ganhará proficiência em programação Java Web.

Referências

DEITEL, Paul J; DEITEL, Harvey M. **Java: Como Programar.** 10; ed. São Paulo: Pearson, 2016.

DEITEL, Paul J; DEITEL, Harvey M. A. **Rich Internet Applications e Desenvolvimento Web para Programadores.** São Paulo: Pearson Prentice Hall, 2009.

SIERRA, Kathy; BATES, Bert. **Use a cabeça!: Java.** Rio de Janeiro: Alta Books, 2009.

JSF, arquiteturas MVC e introdução à persistência de dados

Autoria: Ariel da Silva Dias

Leitura crítica: Rogério Colpani



Objetivos

- Apresentar o modelo arquitetural de desenvolvimento MVC.
- Compreender os conceitos de JavaServer Faces (JSF).
- Desenvolver exemplos práticos utilizando o JSF.



1. Arquitetura de aplicativo web

A arquitetura de aplicativo web, define como acontecem as interações entre aplicativos, sistemas de middleware e bancos de dados, de modo a garantir que vários aplicativos possam trabalhar juntos. Sendo assim, quando um usuário (cliente) digita uma URL (*Uniform Resource Locator*) e clica em “ir”, o navegador encontra o computador voltado para à internet onde a aplicação está hospedada (servidor), e solicita a página específica (arquivo HTML – *HyperText Markup Language*, por exemplo).

O servidor, então, responde enviando arquivos para o navegador. Após essa ação, o navegador executa esses arquivos para mostrar a página solicitada ao usuário. Agora, o usuário pode interagir com o aplicativo.

Veja então que uma arquitetura de aplicativo web é um padrão de interação entre vários componentes. Toda a descrição acima, da solicitação de uma página até o recebimento dos dados, reflete o conceito de arquitetura cliente-servidor. O tipo de arquitetura de aplicativo da web, depende de como a lógica do aplicativo é distribuída entre os lados do cliente e do servidor.

Neste modelo de arquitetura, temos o cliente que é o front-end, por onde o usuário interage e temos o back-end, que é o servidor do aplicativo e parte inacessível pelo usuário.

Vejamos outros tipos de arquitetura de aplicativos web.

1.1 Arquitetura sem servidor

Permite que os aplicativos sejam executados sem correlação com as tarefas relacionadas à infraestrutura, em que os desenvolvedores não precisam gerenciar os servidores back-end, trabalhando na infraestrutura de terceiros (serviço em nuvem).

O benefício dessa abordagem é que ela permite que os aplicativos executem a lógica do código sem se preocupar com as tarefas relacionadas à infraestrutura.

A arquitetura sem servidor é melhor quando a empresa de desenvolvimento não deseja gerenciar ou dar suporte aos servidores, bem como ao hardware para o qual eles desenvolveram o aplicativo web.

1.2 Arquitetura SPA

Outra tendência é um aplicativo de página única (SPA – *Single Page Application*). É aqui que a interface do usuário web é apresentada por meio de um aplicativo JavaScript avançado. Em vez de carregar páginas completamente novas do servidor a cada nova interação do usuário, os aplicativos web de página única permitem uma interação dinâmica por meio do fornecimento de conteúdo atualizado à página atual.

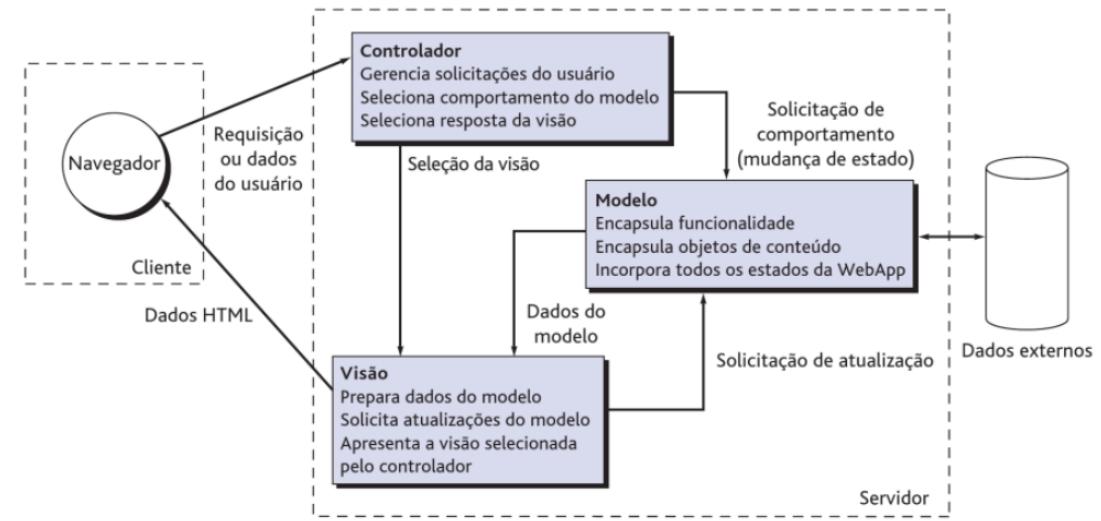
Em termos de requisições, ele usa AJAX ou WebSockets para realizar requisições assíncronas ou síncronas ao servidor web sem ter que carregar a página.

1.3 Padrão arquitetural MVC

O padrão arquitetural MVC (*Model-View-Controller*) é usado para projetar e criar interfaces e a estrutura de um aplicativo. Esse padrão divide o aplicativo em três componentes (modelo, visão e controlador) que são dependentes e conectados entre si. Esta divisão arquitetural é usada para distinguir a apresentação dos dados da maneira como os dados de entrada do usuário e os dados que estão sendo mostrados.

A Figura 1 apresenta cada um destes componentes. Veremos, a seguir, a responsabilidade de cada um deles:

Figura 1 – Arquitetura MVC



Fonte: Pressman (2016, p. 349).

Veja a seguir a descrição de cada um destes componentes da Figura 1.

1.3.1 Modelo (*Model*)

O modelo representa a forma dos dados e da lógica de negócios. Ele mantém os dados do aplicativo. Os objetos de modelo recuperam e armazenam o estado do modelo em um banco de dados. Tal modelo, muitas vezes, é mais conceitual do que física. Por exemplo: no site de uma loja virtual, existe um formulário em que o usuário pode enviar opiniões. O modelo (*model*) deve saber como executar a tarefa de enviar o formulário, ou como executar a tarefa de calcular o frete.

Esta camada **não** interessa em como os dados serão mostrados para o usuário, ou quando a ação de enviar o formulário será executada.

1.3.2 Visão (*View*)

Neste momento, temos a interface de usuário. Esta camada deve saber renderizar os dados em uma interface, seja ela um arquivo de saída (xml, doc, txt, pdf), um JSON ou o próprio navegador do usuário. Para a

camada visão, **não** interessa como obter os dados ou em que momento deve renderizá-los.

1.3.3 Controlador (*Controller*)

O controlador lida com a solicitação do usuário. Normalmente, o usuário interage com a visão, que, por sua vez, gera a solicitação de URL apropriada. Essa solicitação será tratada por um controlador. Logo, o controlador informa quando as coisas devem acontecer. No exemplo citado anteriormente, quando o usuário clicar no botão “enviar” do formulário, este deve ser enviado para o endereço de e-mail do proprietário da loja virtual. Aqui, o papel do controlador foi pegar os dados do formulário e enviar. Não interessa para o controlador como obter os dados ou como exibir estes dados, só interessa quando fazer isso.

1.4 Modelos de aplicativo web

O modelo de um aplicativo web, vai depender do número total de servidores e bancos de dados usados por ele, podendo ser um dos três seguintes:

Com um servidor e um banco de dados. É o modelo de componente de aplicativo web mais simples e menos confiável. Esse modelo usa um único servidor e um único banco de dados. Um aplicativo da web baseado nesse modelo será desativado caso o servidor seja desativado. Portanto, não é muito confiável.

Este modelo de componente de aplicativo, normalmente não é usado para aplicativos reais, sendo usado, principalmente, para executar testes.

Com dois ou mais servidores e um banco de dados. O objetivo com esse tipo de modelo de componente de aplicativo web é que o servidor não armazena nenhum dado. Quando o servidor da web obtém informações de um cliente, ele o processa e as grava no banco de dados, que é gerenciado fora do servidor. Às vezes, também é chamado de arquitetura sem estado.

São necessários pelo menos dois servidores da web para este modelo, e isso é tudo para evitar falhas. Mesmo quando um dos servidores da web fica inoperante, o outro assume o controle.

Todas as solicitações feitas serão redirecionadas automaticamente para o novo servidor, e o aplicativo web continuará a execução. Portanto, a confiabilidade é melhor em comparação ao servidor único com modelo de banco de dados inerente. No entanto, se o banco de dados travar, o aplicativo web também travará.

Com múltiplos servidores e múltiplos banco de dados. É o modelo de componente de aplicativo web mais eficiente, porque nem os servidores da web nem os bancos de dados têm um único ponto de falha. Existem duas opções para esse tipo de modelo: para armazenar dados idênticos em todos os bancos de dados empregados, ou distribuí-los igualmente entre eles.

Normalmente, não são necessários mais de dois bancos de dados no primeiro caso, enquanto no segundo, alguns dados podem ficar indisponíveis no cenário de uma falha no banco de dados.

Quando a escala é grande, ou seja, mais de cinco servidores da web ou bancos de dados ou ambos, é recomendável instalar平衡adores de carga.

1.5 Java Server Faces (JSF)

Java Server Faces (JSF) é um *framework* MVC Java para a construção de interfaces web orientadas a eventos e baseadas em componentes. Assim como o *Java Server Pages (JSP)*, o JSF permite acesso aos dados e à lógica do lado do servidor. Porém, ao contrário do JSP, que é essencialmente uma página HTML de recursos do lado do servidor, JSF é um documento XML que representa componentes formais em uma árvore lógica. Os componentes JSF são apoiados por objetos Java, que são independentes do HTML e têm toda a gama de recursos Java, incluindo acesso remoto a APIs e bancos de dados.

O conceito principal do JSF é o de encapsular a funcionalidade em componentes reutilizáveis. Isso é semelhante às tags reutilizáveis usadas em JSP, mas os componentes JSF são mais formais.

Embora seja possível usar páginas JSF dentro de JavaServer Pages, é mais comum usar Facelets para construir páginas JSF autônomas. Facelets são páginas XHTML projetadas para definir interfaces JSF. Com o Facelets, é possível usar tags XML para criar uma árvore de componentes que se torna o suporte para uma interface de usuário JSF.

1.5.1 Codificando o Hello World

Vamos então ver na prática como o JSF trabalha. Neste primeiro exemplo, vamos criar o popular Hello World. Antes de começar a codificação, tenha certeza de que o ambiente está pronto com todos os componentes instalados: Apache Tomcat, Eclipse (Java EE) e o Eclipse com o Apache conectados.

Começaremos com um aplicativo simples cujo objetivo é obtermos alguns conceitos básicos e, então, vamos evoluindo este aplicativo. Na Figura 2, temos uma página chamada helloworld.xhtml, a qual receberá a entrada de dados.

Figura 2 – Código fonte da página helloworld

```

1  <!DOCTYPE html>
2  <html lang="pt"
3    xmlns="http://www.w3.org/1999/xhtml"
4    xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
5    xmlns:h="http://java.sun.com/jsf/html"
6  >
7
8  <h:head>
9    <title> Hello World! </title>
10 </h:head>
11
12 <h:body>
13   <h1>Qual seu nome?</h1>
14   <h:form>
15     Primeiro Nome:
16     <h:inputText id="nome" value="#{primeiroNome}" />
17     <br/><br/>
18
19     Segundo Nome:
20     <h:inputText id="sobrenome" value="#{segundoNome}" />
21     <br/><br/>
22
23     <h:commandButton value="submit" action="resposta" />
24
25   </h:form>
26 </h:body>
27
28 </html>

```

Fonte: elaborada pelo autor.

Primeiramente, podemos observar na Figura 2 é que o código é um HTML normal. Na realidade, é algo a mais, é uma página XHTML, o que significa que todas as tags da página precisam ser fechadas corretamente.

Nas linhas de 1 a 6, estamos dizendo ao navegador que o padrão oficial do XHTML está sendo usado. Nas linhas 4 e 5, temos uma declaração para fins do JSF, estamos declarando duas bibliotecas de componentes.

Observe na linha 5 que temos uma letra h recebendo a biblioteca JSF. Logo abaixo, na linha 8, temos uma tag <h:head> e a letra h desta tag é a mesma da linha 5, ou seja, utilizamos esta letra no JSF apenas para referenciar que este elemento <h:elemento> é um componente JSF.

Nas linhas 16 e 20, temos um componente simples e poderoso que é o campo de entrada. Quando o formulário for enviado, o valor será colocado em uma classe Java – veremos isso mais adiante. Por enquanto, vamos apenas capturar o valor de uma página HTML e enviar à ela

resposta.xhtml, a qual é requisitada na linha 23. O conteúdo da página resposta.xhtml está presente na Figura 3.

Figura 3 – Código fonte da página resposta

```

1 <!DOCTYPE html>
2 @<html lang="pt"
3   xmlns="http://www.w3.org/1999/xhtml"
4   xmlns:h="http://java.sun.com/jsf/html"
5   >
6
7 @<h:head>
8   <title> Hello World - Resposta </title>
9 </h:head>
10
11 @<h:body>
12   Olá #{primeiroNome} #{segundoNome}
13 </h:body>
14
15 </html>
```

Fonte: elaborada pelo autor.

Na Figura 3, temos uma estrutura básica semelhante a do código da Figura 2. A diferença é que não temos um formulário, e que aqui estamos recebendo os dados que é o nome e o sobrenome. A saída deste código pode ser vista na Figura 4.

Figura 4 – Aplicativo em execução

Entrada	Saída

Fonte: elaborada pelo autor.

Observe, então, que o usuário inseriu o nome e o sobrenome, e foi enviado para a página resposta, a qual apresentou o conteúdo na tela.

1.5.2 Aprimorando o Hello World

Até aqui, vimos a estrutura inicial e como podemos trabalhar com alguns elementos do JSF. Agora vamos aprimorar o nosso formulário. Neste exemplo, vamos utilizar o conceito de Managed Beans.

Os Managed Beans nada mais são do que uma classe Java regular a qual é usada para conter os dados de um formulário. Deste modo, você enviará um formulário e essa classe vai conter os dados dele. Os Managed Beans são utilizados para manter sua lógica de negócios.

Aqui cabe uma consideração importante. Os *Managed Beans* não podem ser confundidos com os *Enterprise Java Beans* (ou EJB). São componentes totalmente diferentes e voltados para fins diferentes.

Dessa forma, vamos aprimorar o nosso aplicativo do Hello World. Agora, criaremos um Managed Beans, que é uma classe Java chamada “Aluno”. Veja que interessante, nossa página helloworld.xhtml (formulário) captura os dados dos inputs e envia para a classe Aluno.java. Por outro lado, a página resultado.xhtml lê os dados da classe Aluno.java.

Primeiramente, criaremos a classe Java Aluno.java. Esta classe, precisa seguir três regras: 1) criar um construtor público sem argumento; 2) apresentar os métodos gets e sets; 3) por fim, adicionar a anotação @ ManagedBean. O código da Figura 5 apresenta a classe Aluno.java.

Figura 5 – Código fonte da classe “Aluno”

```

3 import javax.faces.bean.ManagedBean;
4
5 @ManagedBean
6 public class Aluno {
7     private String primeiroNome, segundoNome;
8
9     public Aluno() {
10    }
11
12
13     public String getPrimeiroNome() {
14         return primeiroNome;
15     }
16
17     public void setPrimeiroNome(String primeiroNome) {
18         this.primeiroNome = primeiroNome;
19     }
20
21     public String getSegundoNome() {
22         return segundoNome;
23     }
24
25     public void setSegundoNome(String segundoNome) {
26         this.segundoNome = segundoNome;
27     }
28 }
```

Fonte: elaborada pelo autor.

O código está pronto, seguindo as regras necessárias para termos um *Managed Beans*. Para que possamos acessar os atributos da classe Aluno em nossa página, precisamos utilizar o JSF *Expression Language*, cuja sintaxe básica é `#{<nome_do_beans>.<atributo>}`.

Observe o novo código do formulário da nossa página helloworld.xhtml na Figura 6.

Figura 6 – Criação de formulário JSF com *Managed Beans*

```

14<h:form>
15     Primeiro Nome:
16     <h:inputText id="nome" value="#{estudante.primeiroNome}" />
17     <br/><br/>
18
19     Segundo Nome:
20     <h:inputText id="sobrenome" value="#{estudante.segundoNome}" />
21     <br/><br/>
22
23     <h:commandButton value="submit" action="resposta" />
24 </h:form>
```

Fonte: elaborada pelo autor (2020)

Na Figura 6, temos apenas o trecho do formulário da página da Figura 2. Na linha 16, por exemplo, foi destacado a *expression language*, que é `estudante.primeironome`. Este tipo de chamada é o mesmo que fizéssemos `estudante.setPrimeiroNome(...)` em orientação a objetos.

Observe na figura 7 como fica a saída na página resposta.xhtml.

Figura 7 – Criação do código de saída com *Managed Beans*

```

1 <!DOCTYPE html>
2<html lang="pt"
3   xmlns="http://www.w3.org/1999/xhtml"
4   xmlns:h="http://java.sun.com/jsf/html"
5   >
6
7<h:head>
8   <title> Hello World - Resposta </title>
9 </h:head>
10
11<h:body>
12   Olá #{estudante.primeiroNome} #{estudante.segundoNome}
13 </h:body>
14
15 </html>
```

Fonte: elaborado pelo autor.

Na Figura 8, estamos apresentando o nome e o sobrenome do estudante. Observe na linha 12 que a *expression language* estudante.primeiroNome é o mesmo que estudante.getPrimeiroNome() em orientação a objetos.

1.5.3 Conversor de temperatura

Agora, vamos um pouco além em nossa implementação. Criaremos um aplicativo que será responsável por converter uma dada temperatura em Celsius informada pelo usuário em graus Fahrenheit. Nesta aplicação, teremos uma página HTML chamada calculadora.xhtml e um arquivo onde está a lógica de negócio que é a Calculadora.java.

Começaremos o desenvolvimento a partir da classe Java. A Figura 8 apresenta o código da Calculadora.java.

Figura 8 – Código fonte da classe “Calculadora”

```

5  @ManagedBean
6  public class Calculadora {
7      float resultado, tempCelsius;
8
9*     public float getResultado() {□
12
13*     public void setResultado(float resultado) {□
16
17*     public float getTempCelsius() {□
20
21*     public void setTempCelsius(float tempCelsius) {
22         this.tempCelsius = tempCelsius;
23     }
24
25*     public Calculadora() {□
28
29*     public void calcular() {
30         resultado = (tempCelsius * 9/5) + 32;
31     }

```

Fonte: elaborada pelo autor.

Na Figura 8, temos, então, o código com a lógica de negócio, que neste caso, é a conversão de uma dada temperatura. Observe que diferentemente do código anterior, aqui temos, também, um método (**calcular()**) criado por nós que é responsável por realizar a conversão da temperatura.

Vejamos, assim, como é o arquivo HTML na Figura 9.

Figura 9 – Código do arquivo HTML calculadora

```

12*   <h:body>
13     <h1>Entre com uma temperatura em Celsius</h1>
14*   <h:form>
15     Temperatura:
16     <h:inputText id="temperatura" value="#{calculadora.tempCelsius}" />
17     <br/><br/>
18
19     <h:commandButton value="submit" action="#{calculadora.calcular}" />
20
21     <br/> <br/>
22
23     Resultado:
24     <h:outputText id="resultado" value="#{calculadora.resultado}" />
25   </h:form>

```

Fonte: elaborado pelo autor.

Na Figura 9 temos a *view*, ou seja, a visualização. Observe que, para fins didáticos, estamos apresentando apenas o conteúdo do formulário, mas saiba que este formulário está dentro de uma *tag body* como já vimos antes.

Na linha 16, temos uma instrução já conhecida por nós. Estamos atribuindo para o atributo tempCelsius da classe calculadora o valor digitado pelo usuário no componente input (equivalente a calculadora.setTempCelsius(...)).

Na linha 19, temos uma situação diferente, pois em *action*, estamos invocando o método calcular (método declarado na linha 29 da Figura 8). Logo, quando o usuário pressionar o botão, a temperatura será enviada para a classe “Calculadora” e, juntamente, será executado o método calcular().

Por fim, na linha 24 temos o equivalente ao calculadora.getTempCelsius(), afinal, estamos obtendo o valor da variável resultado.

JSF é um *framework* MVC, implementando o padrão modelo, visualização e controlador. Este último exemplo implementado, nos ajuda a compreender melhor o conceito do MVC. A visualização é responsável por exibir os dados no modelo e o controlador é responsável por configurar o modelo e encaminhar o usuário para a visualização correta.

Deste modo, podemos afirmar que a visualização é a página calculadora.xhtml com seu conjunto de tags. Eles definem o layout da interface do usuário. A outra metade do uso de JSF é o lado do servidor, em que a classe Calculadora.java suporta o componente da interface.

Neste tema, você conheceu a arquitetura MVC e a comparou com a arquitetura em três camadas. Lembrando que, enquanto a arquitetura em três camadas divide o aplicativo inteiro em interface, lógica e dados, o modelo arquitetural MVC divide a interface do usuário em *view*, modelo e controlador. Por fim, você foi apresentado ao JavaServer Faces, que é o padrão Java para criar interfaces de usuário baseadas na web. É muito importante e fundamental praticar os códigos apresentados, pois, assim, você adquirirá fluência em JavaServer Faces. Bons estudos!



Referências

- DEITEL, Paul J; DEITEL, Harvey M. A. **Rich Internet Applications e Desenvolvimento Web para Programadores.** São Paulo: Pearson Prentice Hall, 2009.
- KURNIAWAN, Budi; DECK, Paul. **Servlet, JSP and Spring MVC.** Quebec: Brainy Software Inc, 2015.
- PRESSMAN, R. **Engenharia de software: uma abordagem profissional.** São Paulo: Bookman. 2016.
- SIERRA, Kathy; BATES, Bert. **Use a cabeça!: Java.** Rio de Janeiro: Alta Books, 2009.
- VOHRA, Deepak. **JavaServer Faces 2.0: Essential Guide for Developers.** Toronto: Nelson Education, 2014.

Hibernate e integração JSF e JPA

Autoria: Ariel da Silva Dias

Leitura crítica: Rogério Colpani



Objetivos

- Compreender o conceito de persistência de dados.
- Compreender o conceito de JDBC e sua aplicação.
- Relacionar JDBC, JPA e Hibernate no processo de desenvolvimento de uma camada de persistência de dados.



1. Java e persistência de dados

Compreender o significado de persistência é importante para avaliar diferentes sistemas de armazenamento de dados. Dada a importância do armazenamento de dados na maioria dos aplicativos modernos, fazer uma escolha mal-informada pode significar um tempo de inatividade substancial ou perda de dados. Nesta postagem, discutiremos abordagens de design de armazenamento de dados e persistência, e forneceremos algumas informações básicas sobre elas no contexto do Cassandra.

Podemos definir o conceito de persistência como a continuação de um efeito após a remoção de sua causa. No contexto do armazenamento de dados em um sistema de computador, isso significa que os dados sobrevivem após o término do processo com o qual foram criados. Em outras palavras, para um armazenamento de dados ser considerado persistente, ele deve gravar em um armazenamento não volátil.

Neste capítulo falaremos sobre persistência de dados, mais especificamente, vamos trabalhar com JPA (Java Persistence API).

1.1 JDBC

O *Java Database Connectivity*, ou simplesmente JDBC, é uma API Java cujo objetivo é gerenciar a conexão com um banco de dados, gerando consultas e comandos. Deste modo, ele manipula um conjunto de resultados provindos de um banco de dados. O JDBC foi lançado como parte do JDK 1.1 no ano de 1997, e foi um dos primeiros componentes desenvolvidos para persistência de dados em Java.

A primeira versão do JDBC foi concebida essencialmente como uma API para ser executada do lado do cliente, de modo a permitir que uma aplicação Java interagisse com uma fonte de dados. Porém, a partir

de sua versão 2.0, o JDBC passou a contar com um pacote opcional de conexão do lado do servidor.

1.1.1 JDBC na prática

O JDBC é uma API de fácil manipulação e, por isso, podemos trabalhar com ela utilizando um ambiente de desenvolvimento (IDE) como o Eclipse ou Netbeans, bem como trabalhar diretamente em um editor de texto.

Começaremos realizando as importações essenciais para realizar a conexão com o banco de dados. A figura a seguir apresenta as referidas importações.

Figura 1 – Código com as importações

```
3④ import java.sql.Connection;  
4 import java.sql.DriverManager;  
5 import java.sql.SQLException;  
6 import java.sql.ResultSet;  
7 import java.sql.Statement;
```

Fonte: elaborada pelo autor.

Vamos analisar cada uma das cinco importações:

- Connection: representa a conexão com o banco de dados.
- DriverManager: responsável por obter a conexão com o banco de dados.
- SQLException: realiza os tratamentos de erros de SQL entre o aplicativo Java e o banco de dados.
- ResultSet e Statement: são responsáveis por modelar os conjuntos de resultados e instruções SQL.

Agora, vamos avançar um pouco mais. O próximo passo é realizar uma conexão com o banco de dados SQLite. A Figura 2 apresenta um exemplo do código (vou omitir as importações e apresentar apenas o trecho do referido código).

Figura 2 – Construindo a classe para conexão

```

9 class Principal{
10    public static void main(String[] args) {
11        Connection conn = null;
12        try {
13            String url = "jdbc:sqlite:C:\\SQLITE\\sql\\teste.db";
14            conn = DriverManager.getConnection(url);
15
16            System.out.println("Conexão OK!");
17
18        } catch (SQLException e) {
19            throw new Error("Problema!!!", e);
20        } finally {
21            try {
22                if (conn != null) {
23                    conn.close();
24                }
25            } catch (SQLException ex) {
26                System.out.println(ex.getMessage());
27            }
28        }
29    }
30 }
```

Fonte: elaborada pelo autor

Na linha 13 do código, temos o endereço de onde foi criado o banco de dados SQLite. Observe que o nome do banco é teste.db. Neste mesmo diretório é necessário que esteja também o driver JDBC SQLite, que é um arquivo com extensão *.jar* que implementa a API JDBC para o banco de dados. Cada gerenciador possui seu próprio driver JDBC.

Esta classe da Figura 2 apenas está se conectando ao banco, o passo final agora é realizarmos uma consulta simples ao banco. A Figura 3, por sua vez, apresenta esta implementação. Observe que o código da Figura 2 foi atualizado.

Com o objeto de conexão já desenvolvido, podemos consultar o banco de dados. Na Figura 3, utilizamos os objetos Connection e Statement para consultar o banco de dados SQLite usando o JDBC.

Na linha 17, do código temos a *string* de consulta no banco com o comando SQL relacionado ao Select. Veja que realizamos uma consulta no banco de dados utilizando um objeto Statement (conn.createStatement() na linha 19).

Na linha 20, estamos executando o comando executeQuery o qual retorna um ResultSet, e que estamos iterando na linha 21 para obter os nomes do banco e apresentar na tela.

Figura 3 – Classe para conexão com o banco e consulta

```

9  class Principal{
10  public static void main(String[] args) {
11      Connection conn = null;
12      try {
13          String url = "jdbc:sqlite:C:\\SQLITE\\sql\\teste.db";
14          conn = DriverManager.getConnection(url);
15
16          Statement stmt = null;
17          String query = "select * from tabela";
18          try {
19              stmt = conn.createStatement();
20              ResultSet rs = stmt.executeQuery(query);
21              while (rs.next()) {
22                  String nome = rs.getString("nomePessoa");
23                  System.out.println(nome);
24              }
25          } catch (SQLException e) {
26              throw new Error("Problema!!!", e);
27          } finally {
28              if (stmt != null) { stmt.close(); }
29          }
30
31      } catch (SQLException e) {
32          throw new Error("Problema!!!", e);
33      } finally {
34          try {
35              if (conn != null) {
36                  conn.close();
37              }
38          } catch (SQLException ex) {
39              System.out.println(ex.getMessage());
40          }
41      }
42  }
43 }
```

Fonte: elaborada pelo autor.

Por fim, após realizar a consulta no banco de dados, estamos fechando a conexão com o banco por meio de uma chamada conn.close() na linha 36.

O JDBC é uma das APIs mais antigas do Java, a qual nos fornece uma maneira fácil de implementar a manipulação de dados. Entretanto, embora seja uma API simples, os desenvolvedores têm apresentado muito interesse por JPA para desenvolver a camada e persistência de dados para seus aplicativos Java, vejamos o motivo a seguir.

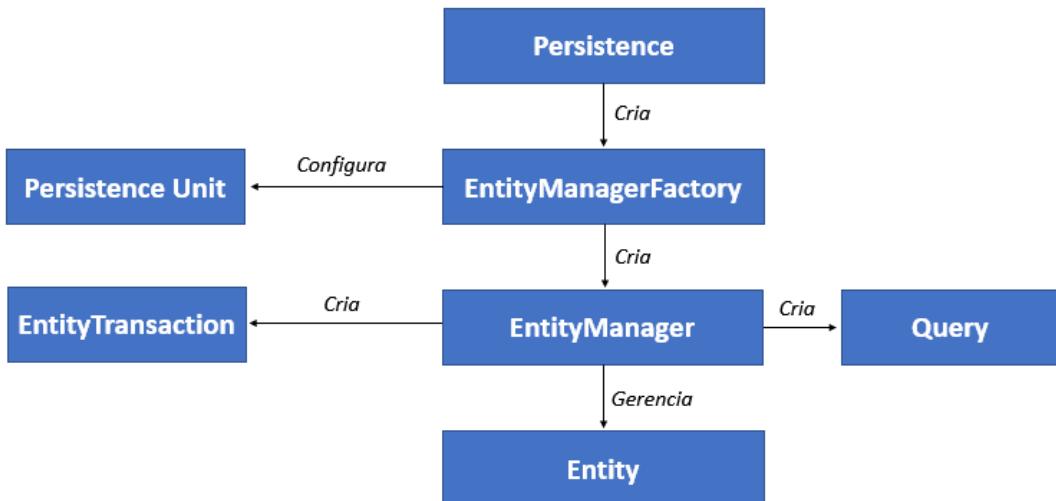
1.2 Java Persistence API (JPA)

O Java Persistence API (JPA) é um conjunto padrão de funcionalidades e conceitos para acessar bancos de dados a partir de aplicativos Java. A principal vantagem do JPA sobre o JDBC é que no JPA, os dados são representados por classes e objetos em vez de tabelas e registros como no JDBC. O uso de objetos Java simples (POJO) para representar dados persistentes pode simplificar significativamente a programação do banco de dados.

Uma implementação JPA (às vezes chamada de provedor JPA) é necessária para interagir com um banco de dados relacional como Oracle, DB2, SQL Server ou MySQL. As mais populares implementações JPA são o Hibernate (código aberto), TopLink (Oracle), EclipseLink (Eclipse Persistence Platform), Open JPA (Apache) e MyBatis (*open source*). Essas implementações são ferramentas *Object Relational Mapping* (ORM). O mapeamento faz a ponte entre a representação dos dados no banco de dados relacional (como tabelas e registros) e a representação no aplicativo Java (como classes e objetos).

A Figura 4 apresenta a arquitetura JPA e as suas principais classes.

Figura 4 – Arquitetura JPA



Fonte: elaborada pelo autor

Vejamos a seguir o que são cada das classes da Figura 4 e como elas se relacionam:

1.2.1 *Entity*

As entidades (*entity*) são os objetos de persistência, armazenados como registros no banco de dados. Cada *entity* representa uma tabela em um banco de dados relacional. Cada instância de uma *entity* corresponde a uma linha desta tabela. JPA utiliza anotações e/ou XML para mapear as *entities*. A figura a seguir apresenta um exemplo de *entity*.

Figura 5 – Código de uma *entity*

```

9  @Entity
10 @Table(name="produto")
11 public class Produto {
12
13@   @Id
14     @GeneratedValue(strategy = GenerationType.IDENTITY)
15     private int id;
16     private String nome;
17     private double valor;
18
19@   public int getId() {
20     return id;
21   }
22
23@   public void setId(int id) {..}
24
25@   public String getNome() {..}
26
27@   public void setNome(String nome) {..}
28
29@   public double getValor() {..}
30
31@   public void setValor(double valor) {..}
32
33}
34
35}
36
37}
38
39}
40
41}
42

```

Fonte: elaborada pelo autor.

Observe que uma *entity* é uma classe Java como outra qualquer. A diferença são as marcações: na linha 9, estamos informando que se trata de uma *entity*; na linha 10, estamos informando o nome da tabela; e nas linhas 13 e 14, estamos indicando que a variável privada *id* é um índice no banco de dados.

Esta classe representa a estrutura da tabela *produto* que encontra-se no banco de dados. Certamente, nós temos uma tabela chamada *produto*, com os campos *id*, *nome* e *valor*. Uma instância (objeto) desta classe *produto* é um registro na base de dados.

Podemos utilizar anotações JPA para mapear uma classe Java para a tabela correspondente. Algumas anotações JPA estão abaixo:

- **@Entity:** é usado para transformar uma classe em uma *entity*, portanto, deve conter um construtor sem argumentos.
- **@Table:** é usado para criar uma tabela para a *entity* referenciada.

- **@Id:** é usado para indicar a chave primária na tabela.
- **@Generatedvalue:** é usado para gerar automaticamente o valor do campo @Id.
- **@Column:** é usado para definir os detalhes da coluna para a qual um atributo é mapeado.

As anotações @Entity, @Table e @Id são obrigatórias na definição de uma *entity*.

1.2.2 Persistence Unity

Este não é uma classe Java, mas sim, um arquivo XML com configuração, o qual define um conjunto de todas as classes de *entity* que serão gerenciadas por instâncias da EntityManager em um aplicativo.

Por padrão, as *Persistence Unity* são definidas pelo arquivo de configuração persistence.xml (no Hibernate este arquivo recebe o nome de hibernate.cfg.xml), o qual é utilizado no JPA para criar a conexão e configuração do ambiente de banco de dados. A Figura 6 apresenta um exemplo deste arquivo.

Figura 6 – Arquivo persistence.xml

```

1<persistence-unit name="projeto">
2    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
3    <properties>
4        <property name="javax.persistence.jdbc.user" value="root"/>
5        <property name="javax.persistence.jdbc.password" value="123456"/>
6        <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/projeto_JPA"/>
7        <property name="javax.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver" />
8
9        <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLInnoDBDialect"/>
10       <property name="hibernate.show_sql" value="true"/>
11   </properties>
12 </persistence-unit>
```

Fonte: elaborada pelo autor.

Observe que o arquivo de configuração inclui a URL de conexão JDBC, credenciais de usuário de banco de dados e classe do driver.

1.2.3 Persistence e EntityManagerFactory

Esta classe contém os métodos estáticos para obter a instância da classe EntityManagerFactory, a qual é criada em tempo de execução pela *Persistence Unity*. Para criarmos uma instância da classe EntityManagerFactory, devemos digitar na classe Persistence:

```
EntityManagerFactory emf = Persistence  
createEntityManagerFactory("projeto")
```

O valor passado por parâmetro projeto é a *Persistence Unity* configurada no arquivo persistence.xml na linha 1.

1.2.4 EntityManager e EntityTransaction

Até aqui, não manipulamos o banco de dados. Logo, cabe à interface EntityManager realizar as principais interações no banco, como criar instâncias de persistência, remover instâncias, encontrar *entities* pela chave primária e realizar consultas em *entities*.

Para criarmos uma EntityManager, devemos digitar o seguinte código:

```
EntityManager entityManager = emf.createEntityManager();
```

Onde emf é uma instância da classe EntityManagerFactory.

Como iremos manipular o banco, é preciso realizar uma transação, ou seja, um conjunto de operações SQL que são confirmadas ou revertidas. Qualquer tipo de solicitação iniciada pelo objeto da EntityManager é inserido em uma transação. Um objeto EntityManager ajuda a criar uma EntityTransaction.

A figura a seguir apresenta um exemplo de código para salvar dados de um produto no banco de dados.

Figura 7 – Código fonte para salvar um item no banco de dados

```

8 public class MeuAplicativo {
9
10    public static void main(String[] args) {
11
12        EntityManagerFactory emf = null;
13        EntityManager entityManager = null;
14        EntityTransaction transaction = null;
15
16        try {
17            emf = Persistence.createEntityManagerFactory("projeto");
18            entityManager = emf.createEntityManager();
19            transaction = entityManager.getTransaction();
20            transaction.begin();
21
22            Produto prod = new Produto();
23            prod.setId(1);
24            prod.setNome("Caneca");
25            prod.setValor(5.40);
26            entityManager.persist(prod);
27
28            transaction.commit();
29
30        } catch (Exception e) {
31            transaction.rollback();
32        } finally {
33            entityManager.close();
34            emf.close();
35        }
36    }
37 }
```

Fonte: elaborada pelo autor.

Observe pela figura todas as principais classes do JPA que nós utilizamos. Veja, também, que na linha 22 criamos uma instância da classe produto e inserimos este item no banco de dados.

1.2.5 *Query*

A *query* é uma interface utilizada para controlar o processo de consulta em um banco de dados. Um objeto EntityManager auxilia na criação de um objeto do tipo *query*, cuja implementação depende do provedor de persistência.

O código a seguir é um exemplo de consulta que pode ser realizada na tabela produto.

Query query = entityManager.createQuery("SELECT s FROM PRODUTO s");

```
List<Produto> s = query.getResultList();  
s.forEach(produto -> System.out.println(produto.getNome()));
```

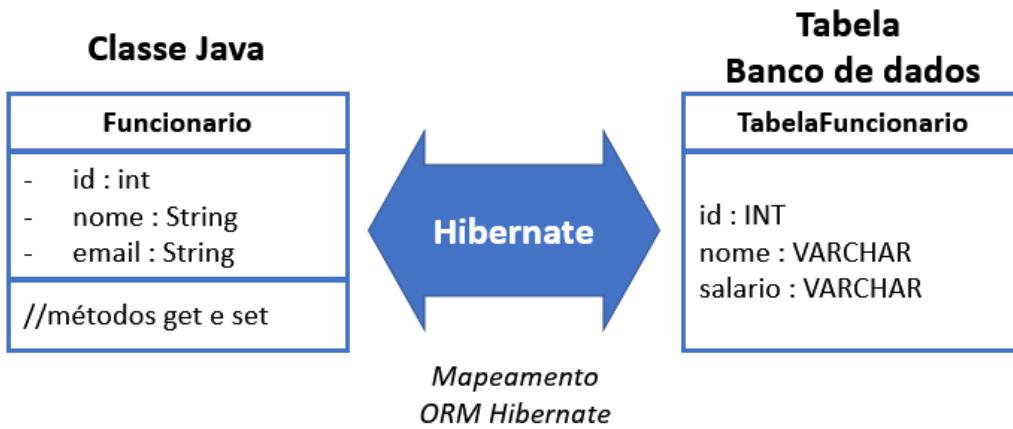
Assim, observe que criamos uma query e passamos a consulta para selecionar todos os itens da tabela produto. Em seguida, criamos uma lista com os resultados da consulta; por fim, criamos uma estrutura de repetição responsável por percorrer a lista e apresentar o nome do produto na tela.

1.2.6 Hibernate

Como brevemente citado anteriormente, o Hibernate é uma implementação ORM baseada em Java que fornece um *framework* para mapear objetos de domínio de aplicativo para as tabelas de banco de dados relacional e vice-versa, como pode ser visto na Figura 8.

O Hibernate atua como uma camada adicional no topo do JDBC, permitindo que você implemente uma camada de persistência independentemente do banco de dados. Ele fornece uma implementação de mapeamento objeto-relacional que mapeia seus registros de banco de dados para objetos Java e gera as instruções SQL necessárias para replicar todas as operações para o banco de dados.

Figura 8 – Mapeamento relacional com Hibernate

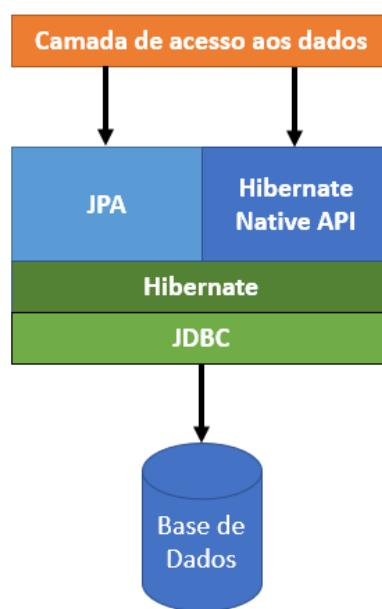


Fonte: elaborada pelo autor.

Na Figura 8, o diagrama mostra um *Object Relational Mapping (ORM)* entre a classe Java chamada Funcionario e a tabela no banco de dados.

Toda comunicação entre o Hibernate e o banco de dados é realizada pelo JDBC. Logo, na Figura 8, considere que o JDBC está à direita da seta azul. Em outras palavras, o Hibernate, como uma solução ORM fica no “meio do caminho” entre a camada de acesso aos dados do aplicativo Java e o banco de dados, como pode ser visto na figura a seguir.

Figura 9 – Arquitetura Hibernate



Fonte: elaborada pelo autor.

Por esta arquitetura da Figura 9, podemos observar que o aplicativo Java faz uso das APIs do Hibernate de modo que possa manipular os dados.

1.2.7 Qual a diferença entre JPA e Hibernate?

Você deve ter notado que JPA e Hibernate são muito parecidos. E é exatamente aqui que surge uma pergunta muito comum: “Qual a diferença entre JPA e Hibernate?”. Entretanto, aqui cabe uma ponderação, afinal, não existe uma relação ou uma proximidade entre os dois conceitos.

O JPA é uma **especificação** desenvolvida de forma colaborativa pela comunidade Java (JCP). Por outro lado, o Hibernate é uma **implementação** da especificação JPA.

Importante que você saiba que existe apenas uma especificação JPA, porém, existem diferentes implementações como TopLink, OpenJPA e o Hibernate. Estes e outros projetos competem com o objetivo de fornecer implementações que sejam mais eficientes, mais rápidas, entre outros critérios. Deste modo, o Hibernate é uma das muitas implementações, por isso, não é possível comparar uma e a outra.

Por outro lado, é possível que haja recursos de mapeamento avançados incorporados à estrutura do Hibernate e que ainda não estão disponíveis nas especificações do JPA.

Neste tema, estudamos o JDBC, uma das APIs mais antigas do Java a qual fornece uma solução fácil para desenvolvimento de aplicativos web. Apesar de sua simplicidade, a especificação JPA é a mais utilizada e, dentre as implementações. Há um destaque, vale ressaltar, para o Hibernate para o desenvolvimento da camada de persistência de dados.



Referências

COELHO, Hébert. **JPA Eficaz: As melhores práticas de persistência de dados em Java.** São Paulo: Casa do Código, 2014.

DEITEL, Paul J; DEITEL, Harvey M. A. **Rich Internet Applications e Desenvolvimento Web para Programadores.** São Paulo: Pearson Prentice Hall, 2009.

VOHRA, Deepak. **JavaServer Faces 2.0: Essential Guide for Developers.** Toronto: Nelson Education, 2014.



BONS ESTUDOS!