Unidade 8: Padrão MVC e DAO

Prof. Daniel Caetano

Objetivo: Apresentar a teoria por trás dos padrões na construção de aplicações Web.

INTRODUÇÃO

Nas aulas anteriores apresentamos a idéia de separar as aplicações em pelo menos dois componentes: o de apresentação e o de processamento. Seguindo esse princípio, foram construídas pequenas Aplicações Web compostas por um Servlet de processamento e um Servlet (JSP) de apresentação.

Essa divisão não é aleatória: ela foi feita segundo algumas regras específicas. Mas que regras são essas? Essas são as únicas separações que faremos em um projeto? Quais são as vantagens? E as desvantagens?

O objetivo desta aula é apresentar, de forma explícita e simplificada, alguns dos mais importantes padrões seguidos em Aplicações Web, independentemente de qual seja a linguagem de programação utilizada.

1. O QUE É UM PADRÃO DE DESENVOLVIMENTO

Sempre que estamos desenvolvendo um sistema, um dos primeiros passos é dividí-lo em partes cada vez menores, até que tenhamos partes pequenas e simples o suficiente para que possamos implementá-las.

Cada solução que um programador dá para cada problema simples - como, por exemplo, escolher o tipo de dado que será usado para armazenar um CPF - tem consequências que podem tornar a manutenção de um sistema desde extremamente simples até algo próximo de impossível.

Assim, é possviel dizer que a forma com que dividimos um sistema, assim como a forma com que implementamos cada uma das pequenas tarefas, tem consequências diretas sobre o nosso sistema, como por exemplo:

a) Sua manutibilidade (flexibilidade e extensibilidade);
b) Seus custos;
c) Reusabilidade de suas partes.

No entanto, grande parte das tarefas que temos de executar em um sistema tradicional já foram exaustivamente implementadas ao longo dos últimos 50 anos, de maneira que, para diversas dessas tarefas, já foram definidas **"melhores práticas"**. Isso significa que, para um grande número de casos, a melhor forma de implementar já é conhecida e documentada.

Essas "melhores práticas" de implementação são conhecidas como **padrões de desenvolvimento**.

2. PADRÃO DE PROJETO

Entre os padrões de desenolvimento, existem aqueles que são denominados padrões de projeto, que são relacionados à melhor maneira de dividir e organizar um sistema, de maneira a solucionar problemas comuns de maneira "elegante", isto é, permitindo grande facilidade de manutenção e estensão, além de facilitar o reuso de cada uma dessas partes.

Os padrões de projeto existem em diferentes níveis de detalhamento. Alguns são extremamente específicos (como implementar a interação entre um sistema novo e um antigo), mas existem também os padrões mais genéricos, que estabelecem apenas alguns critérios a serem seguidos no projeto.

Nesta aula veremos um de cada tipo, lembrando que estes padrões não são soluções definitivas e imutáveis. São apenas diretrizes que, se bem aplicadas, trazem benefícios para o desenvolvimento.

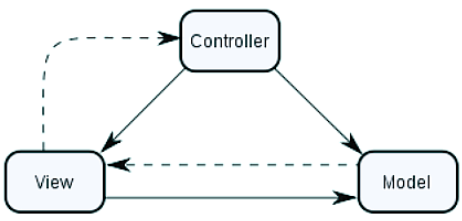
3. PADRÃO MVC

O padrão MVC (*Model-View-Controller* ou, em português, Modelo-Visão-Controle) é um padrão mais macroscópico, digamos assim, do tipo que fornece apenas algumas diretrizes para o projeto. Este tipo de modelo é chamado de **modelo de arquitetura**.

Em essência, o padrão MVC indica que os componentes (servlets, por exemplo) de um sistema devem ser divididos em três categorias distintas:

Modelo: composto pelos componentes de entidade e persistência, esta categoria representa os componentes que representam a modelagem de dados.
Visão: composto pelos componentes de apresentação (janelas, formulários etc.), esta categoria representa os componentes que interagem com o usuário, seja para receber informações, seja para fornecer informações.
Controle: composto pelos componentes de processamento, esta categoria representa os componentes que controlam os processos de negócio, coordenando os outros componentes do sistema para que o resultado final seja aquele esperado pelo usuário.

A relação entre esses componentes é apresentada na figura a seguir:



Nessa representação, as setas "cheias" representam relação direta (direção das requisições) e as setas "tracejadas" indicam relação indireta (direção das respostas).

Pelo diagrama, percebe-se que os componentes do tipo Controle fazem requisições tanto para componetes do tipo Modelo quanto do tipo Visão. Os componentes do tipo Visão, por sua vez, fazem requisições apenas aos componentes do tipo Modelo. Por fim, os componentes do tipo Modelo são passivos, não dando início a requisições para componentes do tipo Controle e Visão.

A razão para essa separação é simples:

a) As mudanças mais frequentes estão na interface com o usuário (Visão) e, normalmente, não exigem mudanças no processo de negócio (componentes de Controle) e na modelagem de dados (componentes de Modelo).
b) Mudanças menores na modelagem de dados (componentes de Modelo) não devem exigir modificações no restante do sistema (componentes de Visão e Controle).
c) Algumas modificações no processo de negócio (componentes de Controle) devem ser possíveis sem exigir alterações nos demais componentes do sistema (Visão e Modelo).

De fato, a parte mais mutante é a interface com o usuário, já que é comum que a empresa queira modificá-la, de tempos em tempos, para ter um "visual mais moderno". A modelagem de dados é modificada com frequência bem menor e, em geral, em razão de modificações na estrutura interna do banco de dados ou para acrescentar informações que nem sempre são visíveis ao usuário, para a geração de estatísticas. Finalmente, o processo de negócio só muda quando também muda a empresa para a qual o sistema foi produzido o sistema. Por exemplo: era uma empresa de vendas e, agora, passa a fazer também locações.

Assim, a separação facilita a manutenção por centralizar algumas dessas mudanças em partes específicas do sistema, não existindo a chance para que alguém "modificando a interface" quebre o funcionamento do processo de negócio.

Mas essas não são as únicas vantagens: a separação em blocos desse tipo permitem uma maior possibilidade de reuso dos componentes desenvolvidos. Ainda que a interface (Visão) de um sistema raramente possa ser aproveitada em outro - ao menos não sem grandes modificações -, os componentes de Controle e, em especial, os de Modelo normalmente podem ser "transplantados" de um sistema para outro praticamente sem alterações significativas, preservando muito o investimento de desenvolvimento.

3.1. O Padrão MVC para a Web

É importante observar, porém, que o este modelo de arquitetura não foi definido para a Web, tendo sido criado na década de 1970, para aplicações desktop, tradicionais, com janelas. Como há diferenças fundamentais entre estes dois tipos de aplicação (desktop x web),

não é possível seguí-lo à risca em aplicações Web, mas é possível preservar as linhas gerais e, com isso, preservar grande parte de seus benefícios.

Quando aplicado à Web, os componentes de visão são associados às páginas HTML e seus formulários; os componentes de Controle são associados aos servlets de processamento e, finalmente, os componentes de Modelo são classes de Entidade, que representam os dados (como o Produto e o Livro que foram desenvolvidos nas primeiras aulas do curso), bem como a comunicação com o banco de dados.

NOTA: para maximizar o reaproveitamento, os servlets de processamento podem ser definidos de maneira a apenas repassar tarefas para classes Java de Controle. Por simplicidade, neste curso optamos por integrar a parte de controle aos servlets de processamento.

Como você deve ter percebido, nas aulas anteriores de Banco de Dados não seguimos muito esta especificação: o mesmo Servlet processa, acessa o banco de daods e imprime os resultados. Vamos ver agora como podemos organizar melhor nossa aplicação.

4. PERSISTÊNCIA

Em um sistema orientado a objetos, incluindo aqueles desenvolvidos segundo o MVC, não existe uma preocupação específica com o armazenamento dos dados em um banco de dados. Isso ocorre porque, originalmente, o modelo orientado a objetos prevê um sistema de **memória persistente**, isto é, um sistema em que a memória não se apaga quando o equipamento é desligado.

Atualmente, este esquema de memória persistente só está disponível para algumas linguagens (como SmallTalk) e em algumas categorias de equipamentos (como grande parte dos celulares e alguns palm-tops). Nestes casos o uso de um banco de dados não é apenas inútil, como também totalmente indesejável.

Por outro lado, quando desenvolvemos sistemas para computadores de memóra principal volátil - como o PC -, o que inclui as aplicações Web, é preciso **"simular" a existência de uma memória persistente**, tarefa essa que fica delegada para os **componentes de persistência**.

Simplificadamente, o papel do componente de persistência é, de maneira transparente (ou quase) para o sistema, armazenar e recuperar objetos de entidade em um banco de dados, de maneira que nenhum outro componente do sistema precise entrar em contato direto com o banco de dados. Existem diversos padrões que permitem esse resultado.

4.1. MVC Nível 1 e 2

Uma das primeiras idéias que se tem ao pensar em persistência é a seguinte:

"Ora, se a persistência serve para armazenar e recuperar objetos de entidade de um banco de dados, que tal inserir esse código dentro do próprio objeto de entidade?"

A afirmação faz todo o sentido do mundo e, de fato, essa é uma das formas de fazê-lo; na verdade, foi a primeira forma de fazer a coisa, que ficou conhecida como MVC Nível 1.

Na próxima aula iremos implementar uma classe de entidade seguindo o MVC Nível 1 e verificar como esse tipo de implementação funciona. Entretanto, este modelo leva a algumas dificuldades, quando se manipula objetos complexos; em especial, quando se quer garantir que apenas uma cópia do objeto permanece na memória.

Para resolver esses problemas, foi proposto o MVC Nível 2, em que a manipulação do Banco de Dados é feito por uma classe externa, usualmente seguindo o padrão DAO.

4.2. O Padrão DAO

O Padrão DAO estabelece alguns critérios para que sejam criadas classes específicas para o acesso ao banco de dados. Este tipo e implementação permite a solução do problema de duplicidade de objetos na memória, e tem uma vantagem adicional: é fácil remover ou substituir o suporte ao banco de dados.

Por que iríamos desejar remover o suporte ao banco de dados? Bem, cada vez mais, estão disponíveis dispositivos de memória persistente no mercado e, sendo assim, para garantir uma boa capacidade de reuso de código, **é interessante que seja fácil remover o suporte ao banco de dados**, para que nosso sistema seja facilmente adaptado para ambientes naturalmente persistentes.

A forma mais simples de obter esse resultado é isolando a parte de persistência de uma classe de entidade em uma outra classe específica para este fim. Por exemplo: nosso sistema tem a classe **Cliente** e, para as tarefas de persistência, foi criada a classe **ClienteDAO**. A nomenclatura deste jeito não é obrigatória, mas ela facilita a identificação das classes de entidade e sua correspondente classe de persistência.

Nesse caso, a classe ClienteDAO é uma classe de serviço, isto é, ela serve apenas para agregar alguns métodos. Desta forma, não é necessário criar um objeto desta classe para que se possa executar seus serviços. Adicionar um objeto ao banco de dados torna-se simplesmente uma questão de comandar:

```
ClienteDAO.adiciona( nome_do_objeto_do_cliente );
```

Para cada classe de entidade teremos uma classe DAO correspondente. Se houver uma classe de entidade Pedido, teremos uma classe PedidoDAO. Se houver uma classe de entidade Produto, teremos uma classe ProdutoDAO... e assim por diante.

4.3. Outros Padrões de Persistência

O Java EE, em suas versões mais recentes, apresenta uma interface própria de persistência, denominada JPA (*Java Persistence API*). A JPA **não é uma biblioteca**, mas sim uma interface padronizada que pode ser adotada por aqueles que quiserem implementar um framework de persistência. A idéia é que, se diversos frameworks implementarem a JPA, o uso de todos eles será bastante similar. O JPA é baseado na arquitetura do tipo MVC Nível 1 e é usada por diversos frameworks.

Um dos frameworks de persistência mais comuns e conhecidos para Java é **Hibernate**. O Hibernate é integrado ao Java e proporciona uma série de facilidades para a implementação de persistência em um sistema; por outro lado, o Hibernate exige uma configuração específica para seu funcionamento. O Hibernate padrão, portanto, não segue a JPA; entretanto, foi criada uma versão denominada **Hibernate JPA**, que permite a manipulação do Hibernate seguindo o padrão JPA.

Um outro framework de persistência que implementa o JPA é o TopLink, fornecido pela própria Oracle, junto com o NetBeans. Mais para o final do curso veremos como trabalhar com o TopLink e aprenderemos um pouco mais sobre o JPA.

5. BIBLIOGRAFIA

DEITEL, H.M; DEITEL, P.J. **Java: como programar** - Sexta edição. São Paulo: Pearson-Prentice Hall, 2005.