Web Handlers

Gibeon Aquino e Paulo Borba*

Centro de Estudos e Sistemas Avançados do Recife (CESAR)
Centro de Informática, Universidade Federal de Pernambuco

Abstract

With the widespread use of the Internet, more and more web-based information systems are being developed. Nowadays, many of these are developed in Java because of the several quality and productvity factors offered by the language and, mainly, for the amount of developed solutions using it. Java web technologies are very powerful, but are very new too. For this reason, there isn't a good number of design patterns or idioms for Java web-based systems [2]. In order to contribute to solve this problem, here we describe the Web Handlers pattern, which provides a way to structure and organize web-based systems to prevent replication of code.

* Parcialmente financiado pelo CNPq, processo 301021/95-3.
E-mails: gibeon@cesar.org.br e phmb@cin.ufpe.br

Introdução

Hoje, muitos sistemas estão sendo desenvolvidos em Java [8]. O motivo é que ela oferece diversas vantagens sobre as outras linguagens e vem sendo uma tecnologia fortemente aceita no mercado. A quantidade de produtos e soluções desenvolvidas em Java tem aumentado muito, isto está fazendo com que ela se torne um padrão de linguagem de programação.

A API de Java que dá suporte ao desenvolvimento de aplicações *Web* é baseada em *Servlets* [6]. Estes nada mais são do que programas escritos na linguagem Java que ficam residentes em algum servidor *Web*. Eles são definidos como classes e possuem alguns métodos padrões que devem ser implementados para que se comportem como esperado. Os *Servlets* podem ser vistos como uma extensão do servidor *Web*, já que recebem requisições e geram dados em qualquer formato suportado pelo protocolo HTTP. A grande diferença está no fato deles funcionarem de forma semelhante aos programas CGI, ou seja, são capazes de gerar informações dinâmicas para o cliente.

A tecnologia de desenvolvimento de programas para *Web* em Java é muito poderosa, no entanto é muito nova. Por este motivo ainda não existe uma quantidade razoável de Padrões de Projeto que possam ajudar os programadores e projetistas a solucionarem problemas corriqueiros e intrínsecos de sistemas *Web* [2].

Intenção

Provê um mecanismo para estruturar e organizar serviços *Web* baseado em *Servlets* de forma a evitar repetição de código, sem aumentar a complexidade do mesmo.

Contexto

Em sistemas *Web* baseados no modelo de pedido-resposta é comum termos diferentes requisições (pedidos) gerando dinamicamente a mesma página HTML como resposta. Esta situação pode ser percebida em um exemplo bem simplificado de sistema bancário que possui apenas as operações de crédito e débito em conta corrente, além de uma operação de login no sistema. O funcionamento deste é especificado através do mapa navegacional (**Figura 1**), que usa a notação de Diagrama de Estados de UML [5], onde as operações são desenhadas como eventos e as páginas HTML (dinâmicas ou estáticas) como estados.

De fato, analisando a figura percebe-se que as operações Debito, Credito e Login, apesar de serem operações distintas geram como resposta de sua execução a mesma página (Menu de Movimentações). Este fato, apesar de exemplificada com um menu principal, ocorre em diversas outras situações.

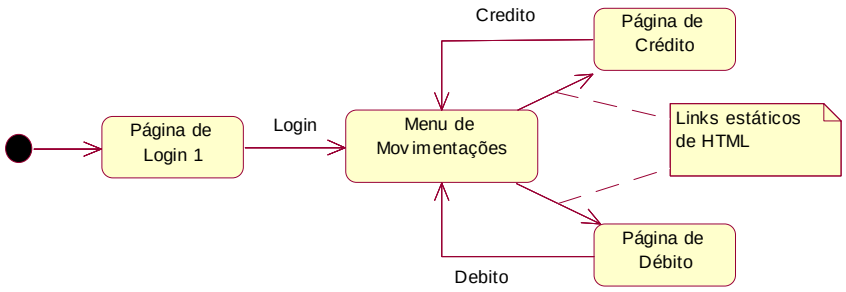


Figura 1

Outro caso comum em sistemas desta natureza é termos um processamento gerando diferentes respostas, dependendo da origem da requisição. Para exemplificar esta situação, considere que o cliente, além de realizar movimentações em conta (débito e crédito), pode fazer atualizações em seu cadastro. O novo comportamento do sistema é especificado na figura abaixo.

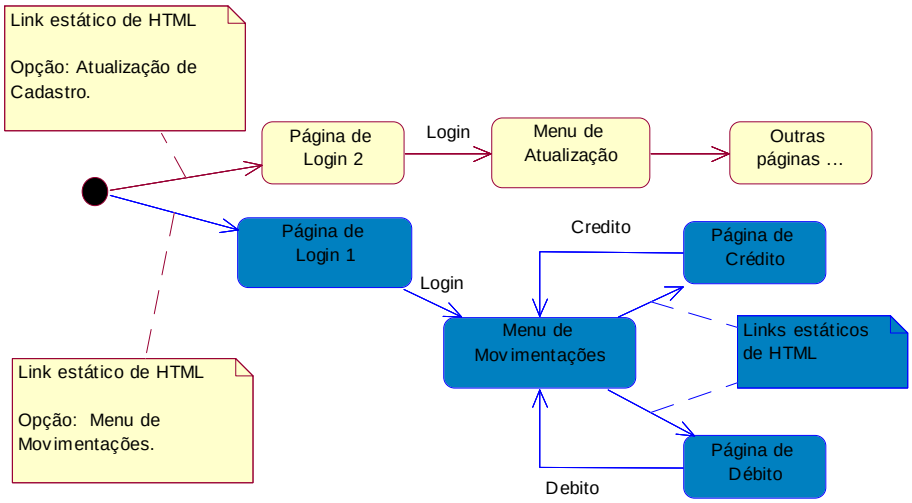


Figura 2

A **Figura 2** exemplifica bem a situação em que a mesma operação, Login, é executada a partir de contextos diferentes (Página de Login 1 e Página de Login 2) e deve gerar uma saída específica (Menu de Atualização e Menu de Movimentações) dependendo da origem da requisição. Estas duas páginas de login possuem o conteúdo bem diferente, pois elas estão em contextos distintos e não possuem relação direta.

Uma forma bem intuitiva de implementar sistemas deste tipo com o uso da tecnologia Java para *Web* é criar um *Servlet* para cada par de operação e página de resposta. Cada um destes é responsável por responder a um único tipo de requisição, onde a operação a ser executada e a página a ser gerada são sempre as mesmas. Esta técnica pode ser chamada de Estruturação de *Servlets* Orientados à Página-Operação, pois existirá um *Servlet* para cada combinação de operação e página possível no sistema. A aplicação desta técnica no sistema

bancário descrito anteriormente produz como resultado os *Servlets* que podem ser vistos na **Figura 3**.

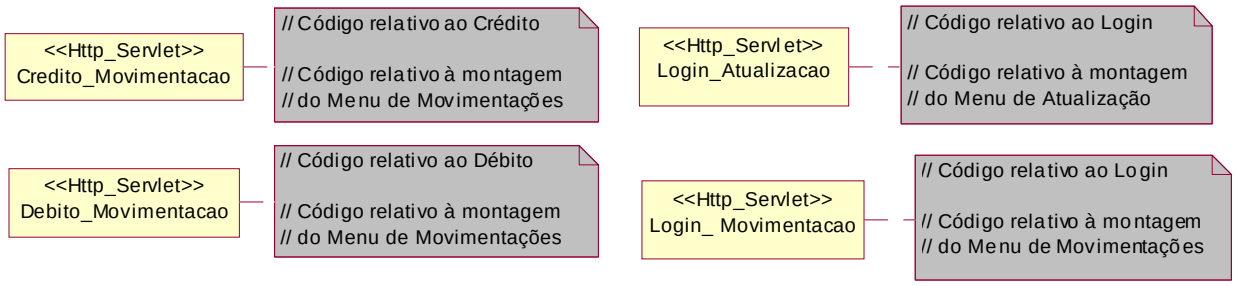


Figura 3

Outra alternativa é criar um único *Servlet* que contém o trecho de código de todas as operações e faz um "*case*" para decidir quais destas deve executar baseado em parâmetros da requisição, montando a página de resposta, que é comum a todos, logo em seguida. Esta técnica pode ser chamada de Estruturação de *Servlet*s Orientados à Páginas, pois existirá um único *Servlet* para cada página do sistema. Utilizando esta técnica o servlet Menu_Movimentacao da **Figura 4** será criado para implementar o comportamento do Credito_Movimentacao, Debito_Movimentacao e Login_Movimentacao mostrados anteriormente na **Figura 3**.

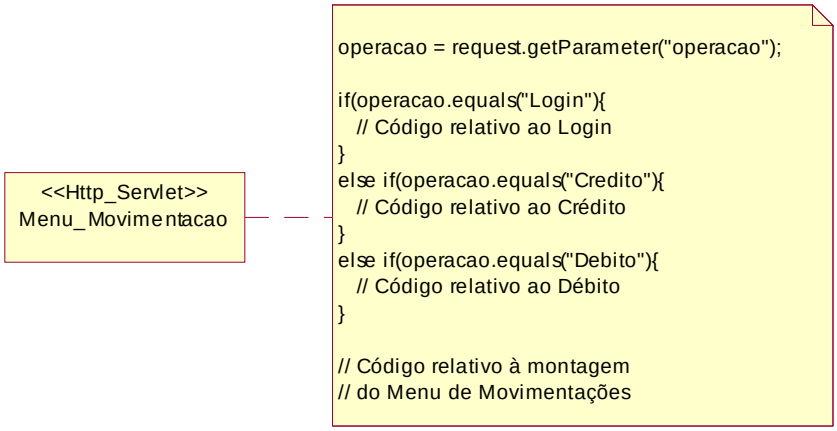


Figura 4

Existe uma outra alternativa, similar a esta última, também muito usada, que é criar um único *Servlet*, que executa a operação e depois faz um "case" baseado em parâmetros da requisição para decidir que página de resposta deve montar. Esta técnica é chamada de Estruturação de *Servlet*s Orientados à Operações, pois existirá um *Servlet* para cada operação do sistema. A **Figura 5** dá uma idéia de seu funcionamento, nela pode-se constatar que o servlet Login utiliza esta técnica agrupando a funcionalidade do Login_Atualização e Login_Movimentação, mostrados na **Figura 3**.

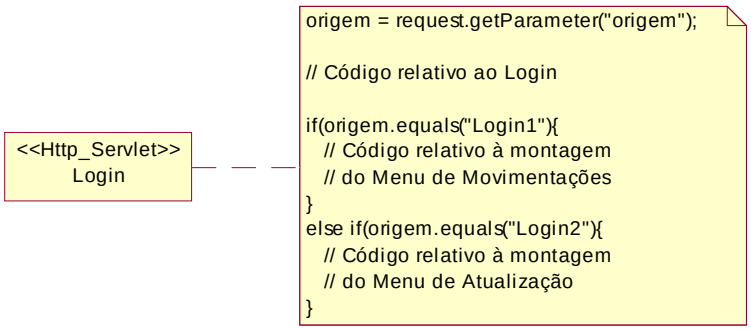


Figura 5

Forças

Na estruturação de *Servlets* orientado à página-operação, a lógica de montagem da apresentação é repetida em alguns S*ervlets* , enquanto que o código relativo à execução da operação fica repetido em outros. A **Figura 3** demonstra bem este fato, onde o código de montagem do Menu de Movimentações é repetido nos *Servlets* `Credito_Movimentacao`, `Debito_Movimentacao` e `Login_Movimentacao`, enquanto que o trecho relativo à operação de login é duplicada no `Login_Atualizacao` e `Login_Movimentacao`.

As outras alternativas, *Servlets* orientados à página e orientados à operações, apresentadas anteriormente, possuem características positivas e negativas, como pode ser visto a seguir.

Positivas:

- A estruturação de *Servlets* orientados à página evita a duplicação do código referente à montagem da apresentação. De fato, analisando a **Figura 4**, percebe-se que a lógica que estava contida nos *Servlets* `Credito_Movimentacao`, `Debito_Movimentacao` e `Login_Movimentacao` foram agrupados em um único *Servlet*, `Menu_Movimentacao`, evitando assim a repetição de código de montagem do Menu de Movimentações.
- A estruturação de *Servlets* orientados à operação evita a duplicação do código de processamento das regras de negócio. O *Servlet* `Login`, mostrado na **Figura 5**, foi implementado usando esta estruturação, evitando desta forma que a operação de login seja replicada em outros *Servlets* do sistema. Ele agrupa a funcionalidade dos servlets `Login_Atualização` e `Login_Movimentação`.

Negativas:

- A estruturação de *Servlets* orientados à página não pode ser aplicada para impedir a repetição da lógica das regras de negócio. Além do mais, esta alternativa pode fazer com que o seu *Servlet* fique mais complexo quando a quantidade de operações que geram a mesma página de saída aumenta.
- A estruturação de *Servlets* orientados à operação não é capaz de impedir que a lógica de montagem das páginas se repita. Esta também apresenta o problema de poder ficar complexo quando o número de páginas de saída possíveis para a mesma operação aumenta.

Problema

As alternativas discutidas anteriormente geram problemas que podem ser resumidos em ocorrência de duplicação de código em casos onde aparecem relacionamentos 1:N entre a parte de processamento e a de apresentação inerente a todo *Servlet* e vice-versa. Naturalmente, estes problemas também ocorrem em situações onde existem relacionamentos M:N entre as duas partes. Estes últimos são os mais comuns, já que é difícil encontrar sistemas que apresentem apenas um dos casos.

A Estruturação de *Servlets* Orientados à Páginas pode ser aplicada em casos de relacionamento 1:N de página para operação, evitando repetição de código, mas em alguns casos tornando o *Servlet* mais complexo. Já a Estruturação Orientada à Operações é aplicável em casos de relacionamento 1:N de operação para página, com benefícios e limitações semelhantes. Ambas estruturações resolvem bem o problema de repetição de código em cada um de seus contextos, introduzindo, apenas em alguns casos, problemas de complexidade.

No entanto, estas alternativas não podem ser aplicadas com a mesma qualidade em sistemas com relacionamento M:N, tais como sistemas *Web*. Por isso é necessário encontrar uma solução que possa ser aplicada em todos os casos relatados aqui, preservando os benefícios das alternativas explicitadas anteriormente em cada um dos seus contextos e sem aumentar a complexidade do código.

Solução

A solução é baseada na construção de entidades denominadas *handlers*. Existem dois tipos destes: *Handlers* de Apresentação e *Handlers* de Processamento. Os primeiros contêm apenas código relativo à montagem das páginas dinâmicas e os outros possuem código (ou chamadas) relativo à execução da lógica de negócio. Com esta estruturação será necessário criar um *handler* de processamento para cada operação do seu sistema e um de apresentação para cada página dinâmica.

Cada requisição *Web* irá disparar uma execução do lado do servidor que dinamicamente montará um par de *handlers* (um de apresentação e um de processamento) para responder ao cliente. Esta composição dinâmica é baseada em parâmetros da requisição do cliente. A entidade responsável pela montagem deste par e delegação da requisição é o **Controlador de Handlers.** Este será especificado com mais detalhes nas seções

Estrutura e Implementação.

Os *handlers* de processamento, além de conterem lógica de execução das operações do sistema, possuem uma parte responsável pela validação dos dados vindos do cliente *Web* e outra parte que é a geração de dados para a execução do seu par de apresentação.

Os *handlers* de apresentação também possuem validação de dados, além de código de montagem das páginas dinâmicas. Esta validação é necessária por que estes precisam

validar os dados gerados pelo seu pares, já que eles são entidades independentes e podem ser compostos de diferentes formas.

Por exemplo, para o sistema bancário as seguintes entidades deveriam ser criadas com a utilização do padrão:

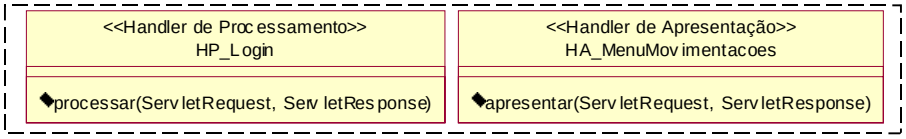
Handlers de Apresentação:

- HA_MenuAtualizacao – Responsável por montar dinamicamente o Menu de Atualização.
- HA_MenuMovimentacoes – Responsável por montar dinamicamente o Menu de Movimentações.

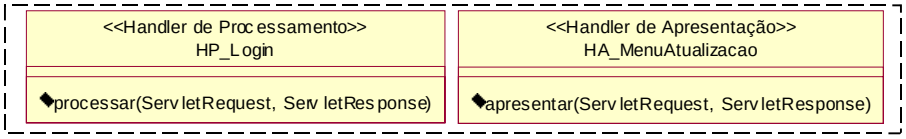
Handlers de Processamento:

- HP_Login – Responsável por executar a operação de login.
- HP_Credito – Responsável por executar a operação de crédito.
- HP_Debito – Responsável por executar a operação de débito.

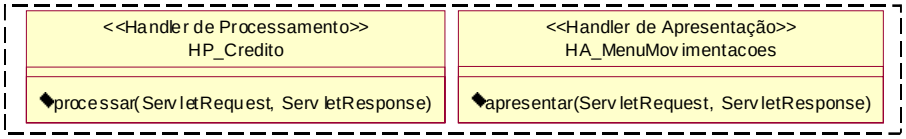
Estas podem ser compostas de diferentes formas para responderem aos diferentes tipos de requisições. A **Figura 6** exemplifica algumas composições possíveis para o sistema bancário com a utilização desse tipo de estruturação.



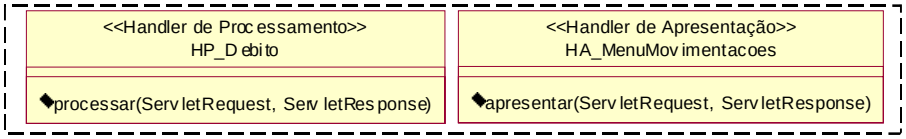
(A) Login a partir da *Página de Login 2*



(B) Login a partir da *Página de Login 1*



(C) Execução da operação de *Crédito*



(D) Execução da operação de *Débito*

Figura 6

De fato, é percebido que o uso de *handlers* é capaz de resolver os problemas de duplicação e complexidade de código, podendo assim ser aplicado na maioria das situações comuns à sistemas *Web*.A **Figura 6** demonstra como os *handlers* podem ser reusados em diferentes requisições, evitando assim a repetição de código.

Aplicabilidade

Use o padrão principalmente em sistemas onde aparecem coisas do tipo:

Figura 7

Uma boa prática é desenvolver o sistema já usando este padrão mesmo quando não ocorrem estas situações, pois desta forma o desenvolvedor já torna o sistema imune à problemas de repetição de código antes mesmo de identificá-los.

Uma forma simples de identificar se o seu sistema necessita do uso deste padrão é desenhando o mapa navegacional do sistema e verificando a existência de ciclos. O aparecimento de ciclos indica que o seu sistema pode apresentar relacionamento 1:N da apresentação para o processamento. Para identificar a ocorrência de relacionamentos 1:N do processamento para a apresentação é só verificar, no mapa navegacional, se a mesma operação é usada em duas transições diferentes.

Estrutura

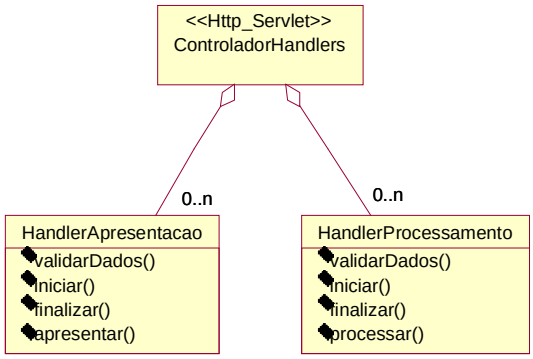


Figura 8

Descrição dos métodos:

- iniciar – Inicia o *handler* no qual ele é chamado, é invocado pelo ambiente.
- finalizar – Finaliza o *handler* no qual ele é chamado.
- apresentar – Contém a lógica referente à montagem das páginas dinâmicas.

- processar – Contém a lógica referente à execução da lógica de negócio.
- validarDados – Contém regras de validação dos dados de entrada.

Participantes

- ControladorHandlers – Responsável pelo controle da execução dos *handlers*.
- HandlerApresentacao – Responsável pela montagem de uma ou mais páginas semelhantes.
- HandlerProcessamento – Responsável pela execução das funcionalidades inerentes ao negócio sendo implementado, e também pela geração de dados para o *handler* de apresentação.

Os *handlers* possuem ciclo de vida semelhante aos dos *Servlets* [6], por isso são oferecidos os métodos iniciar e finalizar, para que o programador possa definir operações que serão executadas na sua inicialização e finalização. Todo *handler* pode implementar o método validarDados, que será executado antes do processar ou do apresentar. Para os *handlers* de processamento ele pode ser usado para implementar regras de validação dos dados da requisição *Web*, enquanto que nos de apresentação ele contém a validação dos dados gerados pelo processamento.

Colaborações

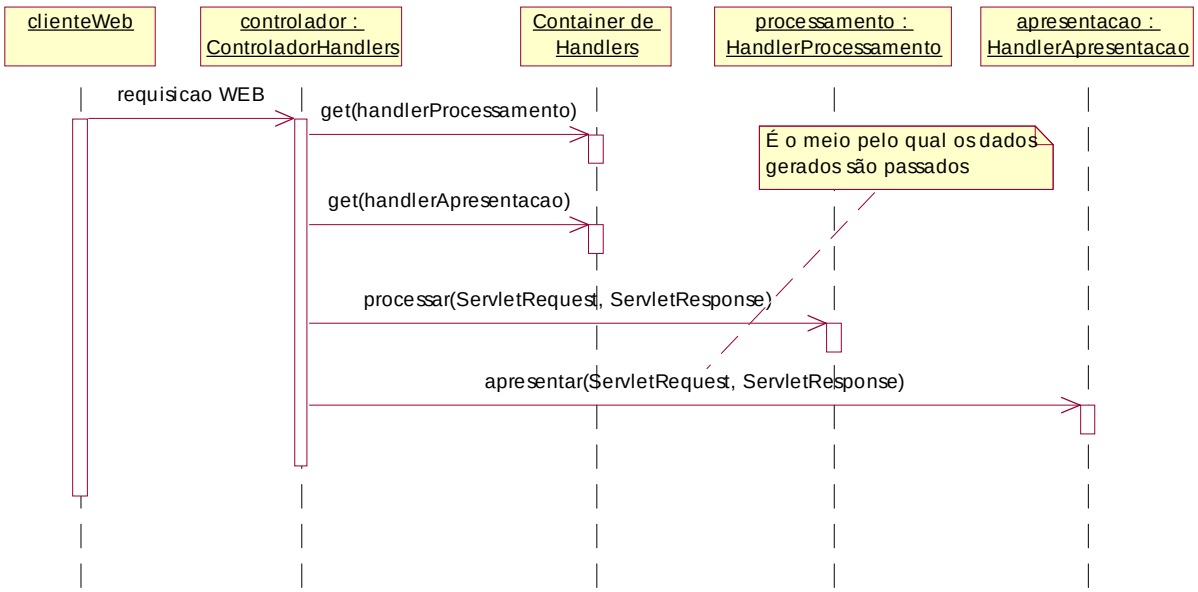


Figura 9

- Toda requisição *Web* é recebida pelo controlador, que interpreta seus parâmetros, recupera o *handler* de processamento e apresentação necessários para executar a requisição e dinamicamente faz a composição dos dois, delegando a requisição para o par.
- O processamento é o primeiro a ser executado através da chamada ao seu método processar. Nele vão estar a lógica de negócio do sistema, ou em alguns casos, chamadas

para classes que implementam esta lógica (EJBs [9], *Facade* [3], etc). Ele também é responsável por produzir os dados de entrada do *handler* de apresentação que será associado a ele.

- O apresentacao é executado após o término com sucesso do processamento através de uma chamada a seu método apresentar. Nele não deve haver lógica de negócio alguma, apenas código referente à montagem da página. Esta entidade vai consultar os dados gerados pelo seu par processamento e construir a página de resposta baseada nestes.

O Container de Handlers é uma entidade que está neste contexto apenas para que se tenha um entendimento melhor do processo de acesso e recuperação dos *handlers*. Ele nada mais é do que o ambiente onde os *handlers* estarão sendo executados, isto é, um servidor *Web* ou mais especificamente um *container Web*.

Conseqüências

- Grande flexibilidade na composição das partes de apresentação e processamento – Os *handlers* de apresentação e processamento são independentes, um dos outros e podem ser integrados de forma diferente para responder a diferentes tipos de requisições, desde que eles sejam compatíveis em relação aos dados produzidos e consumidos.
- Maior reuso de código – O padrão evita a repetição desnecessária de código, pois permite o compartilhamento de entidades para responderem a diferentes requisições.
- Mudanças nos mecanismos de montagem da apresentação (JSP [6], FreeMarker [4] e WebMacro [14]) não causam efeito algum nas entidades de processamento.
- Facilita a implementação de sistemas que requerem diferentes formatos de saída para a mesma operação (Sistema que podem gerar formatos como XML, HTML, WML etc, dependendo do tipo de cliente que acessa-o) – Com o uso do *Web Handlers* o desenvolvedor pode criar *handlers* de processamento que serão compostos com diferentes *handlers* de apresentação, onde estes últimos possuem implementações para cada formato de saída possível.
- Facilita a implementação de componentes (*handlers* de apresentação e processamento) que podem ser reusados em outros sistemas – Com o uso do padrão é mais fácil manter uma biblioteca de *handlers* onde o desenvolvedor pode consultar serviços já implementados e compô-los a fim de obter a implementação de um serviço desejado.
- O padrão ajuda a evitar repetição de código, mas é preciso ter um certo cuidado para não pecar em tornar o sistema modular demais, pois quanto mais modular maior será o número de classes necessárias para implementá-lo.

Implementação

Todos os *handlers* possuem um comportamento bem similar e interfaces bem definidas, por isso é interessante que existam classes e interfaces que dêem apoio ao funcionamento do padrão, provendo comportamento genérico e especificando a interface das operações dos *handlers*. Estas podem ser vistas na figura abaixo:

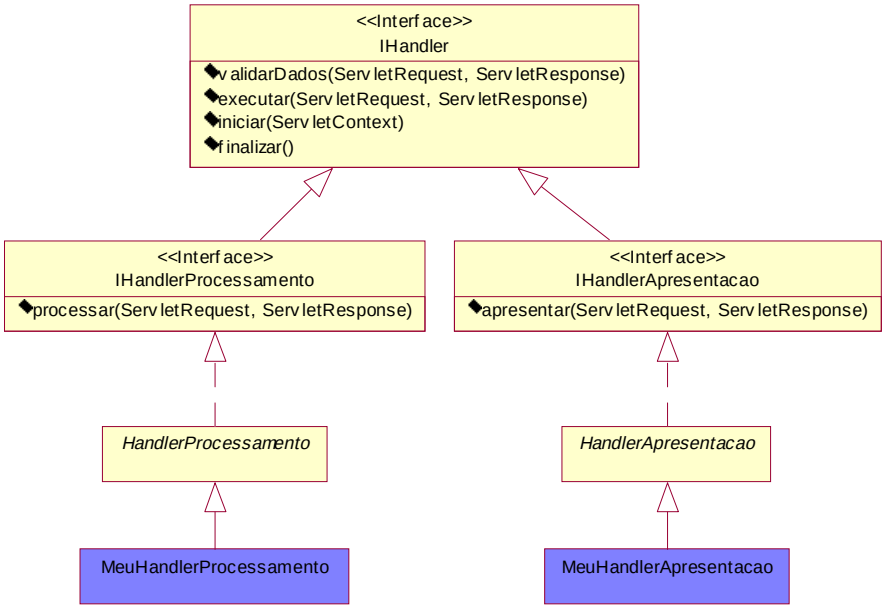


Figura 10

- IHandler - Interface genérica que especifica as funcionalidades que todo *handler* deve oferecer.
- IHandlerProcessamento - Interface que define todas as operações que devem existir em um *handler* de processamento.
- IHandlerApresentacao - Interface que define todas as operações que devem ser oferecidas por um *handler* de apresentação.
- HandlerProcessamento - Classe abstrata que implementa as funcionalidades definidas na interface IHandlerProcessamento. Ela é classe que provê uma implementação padrão para os *handlers* de processamento. Toda entidade de processamento deve estender esta classe para possuir o comportamento de um *handler* de processamento.
- HandlerApresentacao - Classe abstrata que implementa as funcionalidades definidas na interface IHandlerApresentacao. Toda entidade de apresentação deve estender esta classe para possuir o comportamento de um handler de apresentação.

Os tipos ServletRequest, ServletResponse e ServletContext são classes padrões da API de *Servlets* [6]. A primeira é o meio pelo qual os parâmetros da requisição são passados. O segundo é usado para enviar resposta para o cliente *Web*. O último é uma referência para o *Container*, através deste pode-se recuperar parâmetros de configuração do ambiente, comunicar-se com outras entidades que estejam executando neste mesmo contexto etc.

Para tornar o ambiente de *handlers* mais poderoso é interessante que existam algumas funcionalidades providas pelo contexto onde eles estão sendo executados, estas são:

- Instanciação automática dos *handlers* - O ambiente é responsável por localizar e carregar os *handlers* necessários para a execução da requisição.

- *Reload* automático de *handlers* - As modificações feitas em *handlers* já carregados serão enxergadas pelo contexto automaticamente.
- Ciclo de vida bem definido e gerenciado pelo ambiente - Os *handlers*, assim como os *Servlet*s, possuem um ciclo de vida bem definido e que é controlado pelo ambiente no qual eles estão executando.

Todas estas características precisam ser implementadas para estarem presentes no ambiente dos *handlers*, mas como o ambiente de *Servlet*s (*Web Container*) já provê todos estes serviços, é uma boa idéia usá-los para os *handlers*. Uma forma de usar estes recursos de forma efetiva é fazendo com que os *handlers* sejam executados dentro do *Servlet Container*. Para isso eles precisam ter a interface de um *Servlet* e se comportar como tal. Uma pequena modificação na estrutura apresentada anteriormente pode ser feita de forma a atender estes requisitos e preservando a semântica dos *handlers* apresentada até o momento. A **Figura 11** dá uma idéia da alteração necessária no modelo.

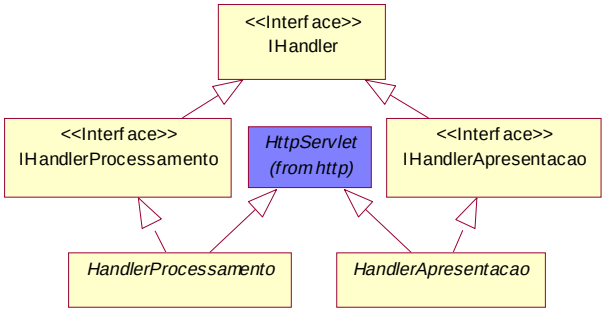


Figura 11

As implementações genérica HandlerApresentacao e HandlerProcessamento continuam implementando as mesmas interfaces, mas agora estendem a classe HttpServlet (pertence a API de *Servlets*) para herdarem o comportamento de *Servlet* e poderem ser gerenciados pelo *Web Container*. Outro detalhe bem interessante é que as implementações genéricas declaram todos os métodos herdados da classe HttpServlet como "final", desta forma evitam que suas subclasses tentem redefinir estes métodos. Enfim, para as classes que o programador precisa implementar, esta mudança não causa nenhum efeito direto.

Na **Figura 12** pode-se ver o diagrama de classes do padrão completo. Foram acrescentadas as classes de exceção ApresentacaoException e ProcessamentoException, ambas herdando de ServletException, e cada uma destas podem ser lançadas pelos métodos dos *handlers* de Apresentação e Processamento, respectivamente.

A última classe que está neste diagrama e ainda não foi explicada é o ControladorHandlers, esta classe é um *Servlet* que faz o papel do Controlador de *handlers* mostrado anteriormente. O motivo do controlador ser um *Servlet* é que ele precisa receber todas as requisições *Web*, e só depois de interpretá-las, repassá-las para os *handlers* responsáveis pelo sua execução.

Uma variação da implementação para resolver o problema do aumento do número de classes do sistema é permitir que o desenvolvedor possa agrupar operações relacionadas ou semelhantes em um único *handler*. Com isso seria necessário implementar um mecanismo de execução de *handlers* mais elaborado, onde além de informações a respeito dos *handlers* a serem executados numa determinada requisição, deveria haver parâmetros indicando que operação executar em cada um deles. Este mecanismo pode ser implementado com a utilização de uma API de Java chamada *Reflection* [7].

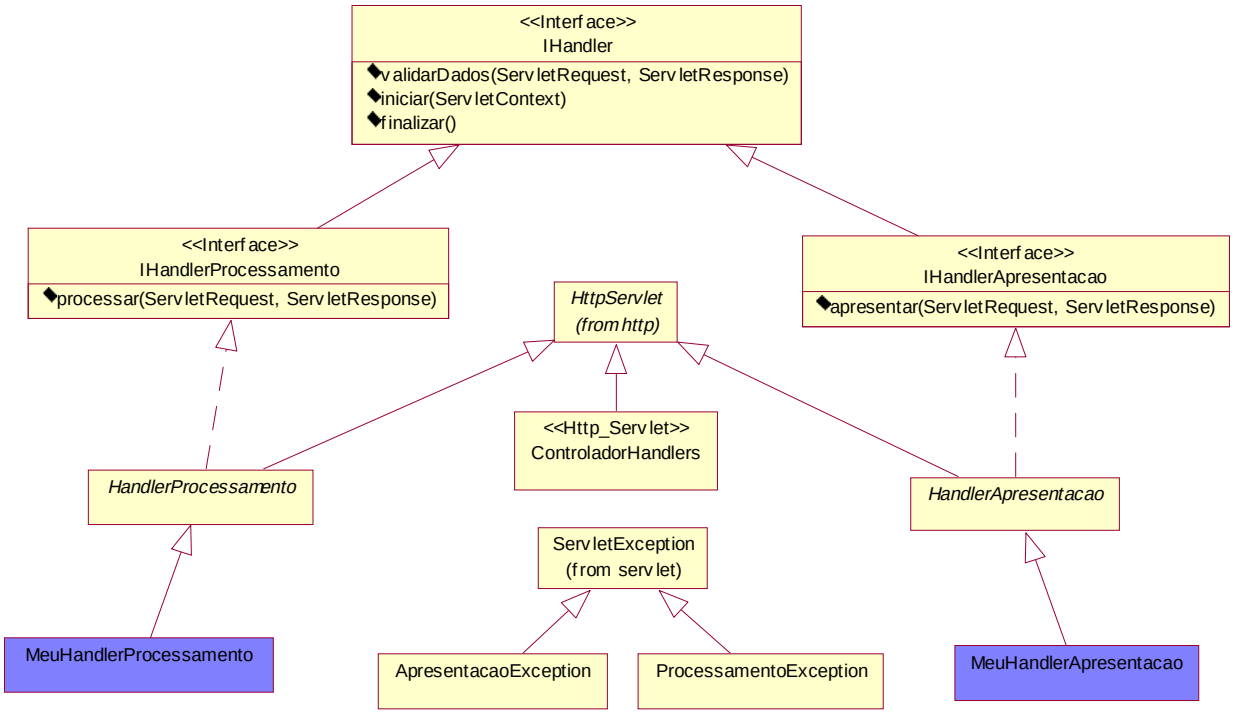


Figura 12

Código de Exemplo

Para exemplificar o uso do padrão serão mostradas as implementações de duas classes do sistema bancário, um *handler* de apresentação e outro de processamento. A **Figura 6**, mostrada anteriormente, dá uma idéia geral da estrutura destas classes e como elas são compostas para atender as requisições do sistema. No trecho abaixo é mostrada a implementação concreta destas classes em Java.

HP_Login

Esta classe é um *handler* de processamento que executa a operação de login no sistema. Como pode ser visto na linha 1 este precisa estender a classe `HandlerProcessamento`, herdando assim o comportamento de *handler* de processamento. Na linha 3 é declarada uma variável do tipo `Sistema`, que agrupa todos os serviços do sistema e é uma implementação do padrão *Facade* e *Singleton* [3]. Na inicialização deste *handler* (chamada

ao método `iniciar`) é recuperada uma instância do sistema para que seja usado durante o processamento (linha 7). O método `validarDados` deste *handler* verifica se os parâmetros da requisição necessários para execução estão presentes (linha 13 e 14), caso não estejam é lançada uma exceção indicando que o processamento deve ser interrompido. As linhas 24 e 25 são usadas para recuperar o valor do login e senha, passados como parâmetro da requisição. Estas vão ser usados para fazer a validação do usuário no sistema através de uma chamada ao método `validarUsuario` do objeto *Facade* (linha 27). Se o login ocorrer com sucesso, as informações do usuário serão recuperadas e armazenadas em um objeto do tipo `Usuario` através de uma chamada ao método `recuperarUsuario` da *Facade* (linha 28). Após este processo, o objeto é armazenado no `request` para que possa ser recuperado pelo *handler* de apresentação par nesta requisição.

```
1:   public class HP_Login extends HandlerProcessamento {
2:
3:      private Sistema sistema;
4:
5:      public void iniciar(ServletConfig config)
6:         throws ProcessamentoException{
7:         sistema = Sistema.getInstancia();
8:      }
9:      public void validarDados(HttpServletRequest request,
10:                              HttpServletResponse response)
11:        throws ProcessamentoException{
12:        // Verifica se os parâmetros necessários para execução estão presentes
13:        if(request.getParameter("login") == null ||
14:           request.getParameter("senha") == null){
15:           throw new ProcessamentoException("Parâmetros imcompatíveis",null);
16:        }
17:     }
18:     public void processar(HttpServletRequest request,
19:                           HttpServletResponse response)
20:        throws ProcessamentoException{
21:        String login,senha;
22:        Usuario usuario;
23:
24:        login = request.getParameter("login");
25:        senha = request.getParameter("senha");
26:
27:        if(sistema.validarUsuario(login,senha)){
28:           usuario = sistema.recuperarUsuario(login);
29:           request.setAttribute("usuario",usuario);
30:        }
31:        else{
32:           throw new ProcessamentoException("Login Inválido",null);
33:        }
34:     }
35:  }
```

HA_MenuMovimentacao

O `HA_MenuMovimentacao` é um *handler* de apresentação, para isso ele estende a classe `HandlerApresentacao`. Seu método `validarDados` verifica a existência de um parâmetro com o nome `usuario` (linha 7). O método `apresentar` recupera o objeto do tipo `usuario` através do `request` (linha 23). A lógica principal de montagem da página está na linha 26, onde a página é montada através de uma chamada ao método `Skin.processaPagina()`.

Este método recebe com entrada um array de chaves, outro de valores e um nome de arquivo. Este arquivo é um *template* de uma página HTML contendo alguns identificadores especiais nos locais onde serão inseridas informações dinâmicas. A tarefa deste método é varrer o arquivo procurando ocorrências de alguma das palavras do *array* de chaves e substituí-las por palavras do *array* de valores. Na linha 27 a página de resposta é enviada para o cliente *Web*.

```
1:   public class HA_MenuMovimentacao extends HandlerApresentacao {
2:
3:      public void validarDados(HttpServletRequest request,
4:                               HttpServletResponse response)
5:         throws ProcessamentoException{
6:         // Verifica se os parâmetros necessários para execução estão presentes
7:         if(request.getAttribute("usuario") == null){
9:            throw new ProcessamentoException("Parâmetros imcompatíveis",null);
10:        }
11:     }
12:
13:     public void apresentar(HttpServletRequest request,
14:                            HttpServletResponse response)
15:        throws ApresentacaoException{
16:
17:        PrintWriter out;
18:        String pagina;
19:        Usuario u;
20:
21:        out = response.getWriter();
22:        try{
23:          u = (Usuario) request.getAttribute("usuario");
24:          String[] chaves = {"$USUARIO"};
25:          String[] valores = {u.getName()};
26:          pagina = Skin.processaPagina(chaves,valores,"Menu_Movimentacao.html");
27:          out.println(pagina);
28:        }
29:        catch(Exception e){
30:           throw new ApresentacaoException("Erro de Apresentação",e);
31:        }
32:     }
33:  }
```

Usos Conhecidos

- Portal Encontre & Compre [10] – Sistema de consultas dos anunciantes Listel, que também permite que o visitante faça transações de negócios on-line com os anunciantes.
- O Sistema de Fomento Lattes [13]– Sistema de informação para gestão de programas de fomento ao desenvolvimento científico e tecnológico. O módulo responsável pela emissão de parecer consultor Ad hoc usa o padrão aqui descrito em sua implementação.
- Prospectar [11] – Sistema de prospecção tecnológica do Governo Federal.
- O Sistema de Acompanhamento e Avaliação do PADCT (Módulo de Licitações) – Este sistema encontra-se em desenvolvimento. O objetivo principal deste módulo é o cadastramento, o registro da situação, e a emissão de relatórios, das licitações de recursos do exterior ocorridas durante a execução dos projetos do programa PADCT III.

Todos os sistemas descritos anteriormente usam o padrão de *Web handlers* na construção de seus serviços *Web*. A implementação do padrão proposta neste documento foi fruto de um trabalho de correção dos problemas identificados nas implementações anteriores, mas a estrutura do padrão continua a mesma.

Padrões Relacionados

Na construção dos *handlers* de apresentação pode ser usado o padrão *Skin* [12] para implementar a lógica de montagem das páginas dinâmicas. O uso do padrão neste contexto traz uma série de benefícios ao desenvolvimento pois permite a separação entre o código HTML e Java, facilitando o desenvolvimento e a manutenção do sistema.

É interessante usar o padrão de projeto *Facade* [3] para agrupar a lógica de negócio do sistema em um único ponto e fazer com que os *handlers* de processamento chamem estas funcionalidades, ao invés de implementarem-na diretamente em seus corpos.

Agradecimentos

Á Rossana M. Castro Andrade, alocada para ser "shepherd" deste artigo, pela sua participação e empenho no envio de comentários e sugestões valiosas para a melhoria da descrição deste padrão.

Ao CESAR por adotar o uso deste Padrão de Projeto na implementação de seus sistemas *Web* e por auxiliar a pesquisa deste e de outros padrões para este tipo de ambiente.

Referências

[1] CESAR – Centro de Estudos e Sistemas Avançados do Recife. http://www.cesar.org.br.

[2] Deepak Alur, John Crupi, and Dan Malks. Sun Java Center J2EE™ Patterns First Public Release: Version 1.0 Beta. March 2001. http://developer.java.sun.com/developer/technicalArticles/J2EE/patterns/

[3] Erich Gamma et al. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.

[4] FreeMarker – http://freemarker.sourceforge.net/

[5] Grady Booch, James Rumbaugh e Ivar Jacobson. The unified modeling language user guide. Addison Wesley, 4th edição, 1999.

[6] James Duncan Davidson, Danny Coward. Java™ Servlet Specification, v2.2 - Final Release. 1999.

[7] Java Core Reflection API Documentation. http://java.sun.com/j2se/1.3/docs/guide/reflection/spec/java-reflectionTOC.doc.html

[8] Java Technology official Home Page. http://java.sun.com/.

[9] Matena, Vlada e Stearns, Beth. APPLYING ENTERPRISE JAVABEANS:Component-Based Development For The J2EE Platform. Addison-Wesley, 1th edição (29 de Dezembro de 2000).

[10] Portal Encontre & compre – http://www.encontrecompre.com.br/.

[11] Prospectar – Sistema de prospecção tecnológica. http://prospectar.cesar.org.br/admin

[12] Rani Pinchuk and Yonat Sharon. The Skin Pattern. 7th. Pattern Languages of Programs Conference, 2000. http://jerry.cs.uiuc.edu/plop/plop2k/proceedings/Sharon/Sharon.pdf

[13] Sistema de Fomento Lattes. http://www.cnpq.br/servicosrestritos/.

[14] Web Macro servlet framework. http://www.webmacro.org/.