Desenvolvimento para Servidores-II

Sessão e Filtro

Neste tópico abordaremos o uso de sessão e filtro em uma aplicação JSF.

Prof. Ciro Cirne Trindade

A necessidade do controle de sessão

- Muitas aplicações requerem que uma série de requisições de um cliente sejam associadas umas com as outras
- Por exemplo, uma aplicação web pode manter o estado do carrinho de compras de um cliente durante várias requisições
- As aplicações web são responsáveis por manter tal estado, chamado uma **sessão**, pois o HTTP é *stateless*

Sessão em JSF

- Para gerenciar sessão em uma aplicação JSF é necessário criar um *managed bean* com escopo **`Session`**
- Isto pode ser feito através da anotação `@SessionScoped`
- *Managed beans* armazenados na sessão devem implementar a interface `Serializable`
- Uma sessão é representada por um objeto da classe `HttpSession`

Exemplo

```
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import java.io.Serializable;

@ManagedBean
@SessionScoped
public class Pagamento implements Serializable {
   ...
}
```

Recuperando a sessão em um *managed bean*

- Para recuperar a sessão é necessário usar o método `getSession()` da classe `ExternalContext`
- Exemplo:

```
FacesContext fc =
        FacesContext.getCurrentInstance();
HttpSession session = (HttpSession)
        fc.getExternalContext().getSession(false);
```

O método `getSession()` espera um argumento booleano que indica se a sessão deve ser criada (true) ou não (false) se ela não existir

Recuperando uma sessão através de um objeto da classe `HttpServletRequest`

- A classe `HttpServletRequest` também possui um método `getSession()` que permite recuperar a sessão corrente
- Exemplo:
  - ```
    HttpSession session =
                        request.getSession();
    ```

Objeto da classe `HttpServletRequest`
    ```

Recuperando as informações associadas a uma sessão (1/2)

- Para recuperar uma informação da sessão, use o método `session.getAttribute("key")`
- Esse método devolve um `Object`, portanto você deve fazer um *type casting* para o tipo apropriado
- O método devolve `null` se não houver o atributo

Recuperando as informações associadas a uma sessão (2/2)

```
FacesContext fc =
      FacesContext.getCurrentInstance();
HttpSession session = (HttpSession)
      fc.getExternalContext().getSession(true);
SomeClass value = (SomeClass)
          session.getAttribute("someIdentifier");
if (value == null) {
    printMessageAttributeNotFound();
}
else {
    doSomethingWith(value);
}
```

Armazenando informações na sessão (1/2)

- Para armazenar informações na sessão, use o método `setAttribute()`
- O método `setAttribute()` substitui qualquer valor anterior do atributo
- Sintaxe:
  - `session.setAttribute("key", value);`

Armazenando informações na sessão (2/2)

```
FacesContext fc =
      FacesContext.getCurrentInstance();
HttpSession session = (HttpSession)
      fc.getExternalContext().getSession(true);
SomeClass value =
   (SomeClass)session.getAttribute("someIdentifier");
if (value == null) {
   value = new SomeClass(...);
   session.setAttribute("someIdentifier", value);
}
doSomethingWith(value);
```

Destruindo a sessão

- Para destruir uma sessão, utilizamos o método `invalidate()` da classe `HttpSession`
- Exemplo:
  - `session.invalidate();`

Exemplo: carrinho de compras

- Para ilustrar o uso de sessão vamos implementar uma aplicação que simule um carrinho de compras de um site de comércio eletrônico
- Nesta aplicação o usuário poderá incluir e excluir itens do seu carrinho de compras

Classe **Item**: representa um item disponível para compra (1/2)

```
package model;

import java.util.Objects;

public class Item {
    private Integer id;
    private String descricao;
    private Integer quantidade;
    private Double valor;

    public Item() {
    }
    public Item(Integer id, String descricao, Integer quantidade,
                Double valor) {
        this.id = id;
        this.descricao = descricao;
        this.quantidade = quantidade;
        this.valor = valor;
    }

    public Integer getId() { return id; }

    public void setId(Integer id) {
        this.id = id;
    }
```

Classe **Item**: representa um item disponível para compra (2/2)

```
    public String getDescricao() { return descricao; }
    public void setDescricao(String descricao) {
        this.descricao = descricao;
    }

    public Integer getQuantidade() { return quantidade; }
    public void setQuantidade(Integer quantidade) {
        this.quantidade = quantidade;
    }

    public Double getValor() { return valor; }
    public void setValor(Double valor) {
        this.valor = valor;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == null || getClass() != obj.getClass())
            return false;
        final Item other = (Item) obj;
        return (Objects.equals(this.id, other.id));
    }

}
```

Managed bean `ItensBackBean`: lista de itens

```
package beans;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
import javax.faces.bean.SessionScoped;
import javax.faces.bean.ManagedBean;
import model.Item;

@ManagedBean
@SessionScoped
public class ItensBackBean implements Serializable {
    private List<Item> itens = new ArrayList<>();

    public ItensBackBean() {
        itens.add(new Item(1, "Cerveja", 10, 2.56));
        itens.add(new Item(2, "Refrigerante", 20, 1.99));
        itens.add(new Item(3, "Suco", 15, 4.19));
        itens.add(new Item(4, "Carvão", 10, 7.89));
        itens.add(new Item(5, "Carne", 5, 29.9));
    }

    public void setItens(List<Item> itens) { this.itens = itens; }
    public List<Item> getItens() { return itens; }
}
```

Cria uma lista de objetos da classe Item

Managed bean
CarrinhoBackBean: carrinho (1/3)

```
package beans;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.context.FacesContext;
import javax.servlet.http.HttpSession;
import model.Item;

@ManagedBean
@SessionScoped
public class CarrinhoBackBean implements Serializable {
    private List<Item> itens = new ArrayList<>();

    public CarrinhoBackBean() { }

    public List<Item> getItens() { return itens; }
    public void setItens(List<Item> itens) {
        this.itens = itens;
    }
```

Managed bean
CarrinhoBackBean: carrinho (2/3)

```
 public Double getTotal() {
    Double tot = 0.0;
    for (Item i : itens) {
        tot += i.getQuantidade() * i.getValor();
    }
    return tot;
}

public String adicionar(Item item) {
    boolean novo = true;
    item.setQuantidade(item.getQuantidade() - 1);
    for (Item i : itens) {
        if (i.equals(item)) {
            i.setQuantidade(i.getQuantidade() + 1);
            novo = false;
            break;
        }
    }
    if (novo)
        itens.add(new Item(item.getId(), item.getDescricao(), 1,
                           item.getValor()));
    return null;
}
```

Managed bean **CarrinhoBackBean**: carrinho (3/3)

```
    public String excluir(Item item) {
        FacesContext fc = FacesContext.getCurrentInstance();
        HttpSession session = (HttpSession)
                    fc.getExternalContext().getSession(false);
        ItensBackBean itensBackBean = (ItensBackBean)
                    session.getAttribute("itensBackBean");
        if (itensBackBean != null) {
            for (Item i : itensBackBean.getItens()) {
                if (i.equals(item)) {
                    i.setQuantidade(i.getQuantidade() + 1);
                    break;
                }
            }
        }
        if (item.getQuantidade() > 1) {
            item.setQuantidade(item.getQuantidade() - 1);
        }
        else {
            itens.remove(item);
        }
        return null;
    }

}
```

index.xhtml (1/4)

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
   <h:head>
      <title>Carrinho de compras</title>
      <h:outputStylesheet library="css" name="cssLayout.css"/>
      <h:outputStylesheet library="css" name="default.css"/>
   </h:head>
   <h:body>
      <h:form>
         <h:dataTable value="#{itensBackBean.itens}" var="i"
                  styleClass="itens" captionClass="itensCaption"
                  headerClass="itensHeader"
                  rowClasses="linhaPar, linhaImpar">
            <f:facet name="caption">Produtos Disponíveis</f:facet>
               <h:column>
                  <f:facet name="header">Id</f:facet>
                  #{i.id}
               </h:column>
```

index.xhtml (2/4)

```
<h:column>
   <f:facet name="header">Descrição</f:facet>
   #{i.descricao}
</h:column>
<h:column>
   <f:facet name="header">Quantidade</f:facet>
   #{i.quantidade}
</h:column>
<h:column>
   <f:facet name="header">Valor</f:facet>
   <h:outputText value="#{i.valor}">
      <f:convertNumber type="currency"/>
    </h:outputText>
</h:column>
<h:column>
   <f:facet name="header">
      Adicionar ao carrinho</f:facet>
   <h:commandButton value="Adicionar"
       action="#{carrinhoBackBean.adicionar(i)}"
       disabled="#{i.quantidade == 0}"/>
</h:column>
</h:dataTable>
<br/>
```

index.xhtml (3/4)

```
<h:dataTable value="#{carrinhoBackBean.itens}" var="c"
     styleClass="itens" captionClass="itensCaption"
     headerClass="itensHeader" footerClass="itensFooter"
     rowClasses="linhaPar, linhaImpar">
   <f:facet name="caption">Produtos do Carrinho</f:facet>
      <h:column>
         <f:facet name="header">Descrição</f:facet>
         #{c.descricao}
      </h:column>
      <h:column>
         <f:facet name="header">Quantidade</f:facet>
         #{c.quantidade}
      </h:column>
      <h:column>
         <f:facet name="header">Valor</f:facet>
         <h:outputText value="#{c.valor}">
            <f:convertNumber type="currency"/>
         </h:outputText>
         <f:facet name="footer">Total:
            <h:outputText
                     value="#{carrinhoBackBean.total}">
               <f:convertNumber type="currency"/>
            </h:outputText>
         </f:facet>
      </h:column>
```

index.xhtml (4/4)

```
                    <h:column>
                        <f:facet name="header">Excluir</f:facet>
                        <h:commandButton value="Excluir"
                              action="#{carrinhoBackBean.excluir(c)}"/>
                    </h:column>
                </h:dataTable>
            </h:form>
        </h:body>
    </html>
```

Estudo de caso: autorização (1/2)

- Um exemplo típico do uso de sessão é o controle de acesso a uma aplicação web (autorização)
- Uma vez autenticado na aplicação (normalmente através de um login e senha), o usuário está autorizado a acessar os recursos da aplicação
- Usuários não autenticados não devem ter acessos a aplicação

Estudo de caso: autorização (2/2)

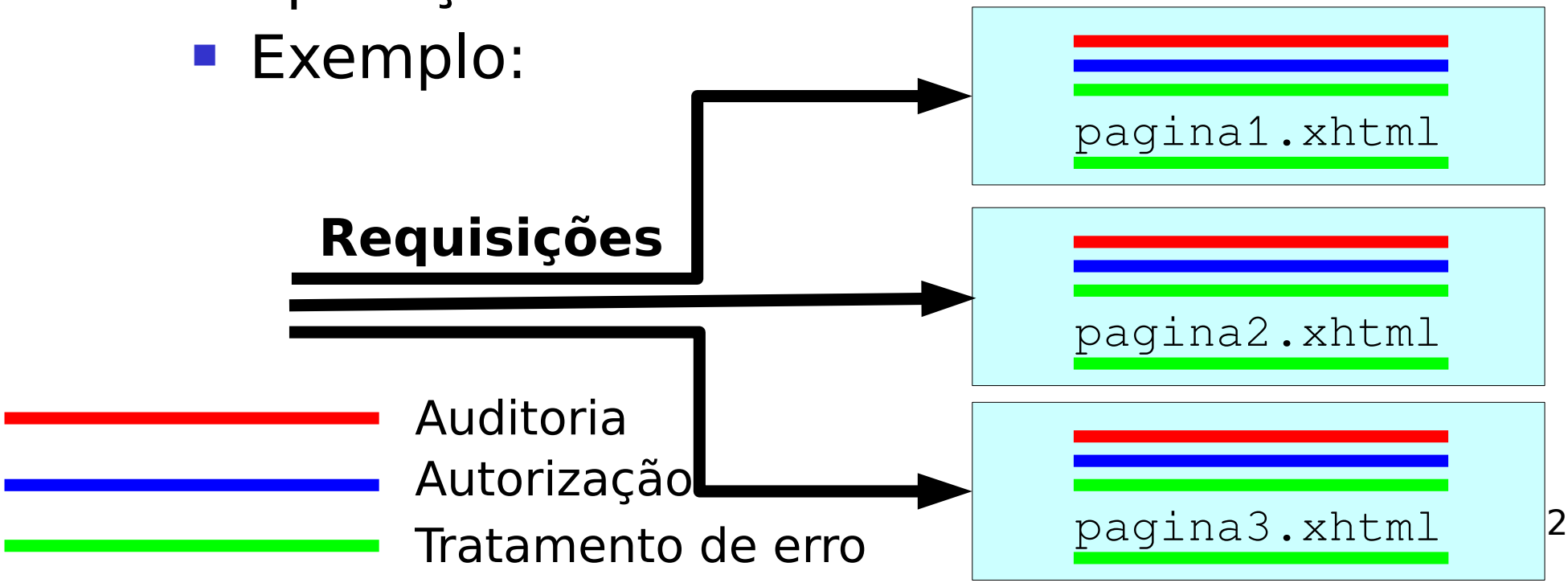
- Para evitar que o usuário tenha que se autenticar a cada nova requisição, seu estado deve ser mantido em uma sessão
- Toda requisição à aplicação deve verificar se o usuário que fez a requisição está autorizado a fazê-la
- Para evitar que este controle tenha que ser realizado em todos os recursos da aplicação, utilizamos um **filtro**

Reduzindo o acoplamento através de filtros (1/4)

- Em qualquer aplicação surgem requisitos que não são diretamente relacionados com a regra de negócios
- Um exemplo clássico desses requisitos não funcionais é a autorização
- Como implementar estas funcionalidades?
- A primeira ideia seria colocar o código diretamente nas páginas web

Reduzindo o acoplamento através de filtros (2/4)

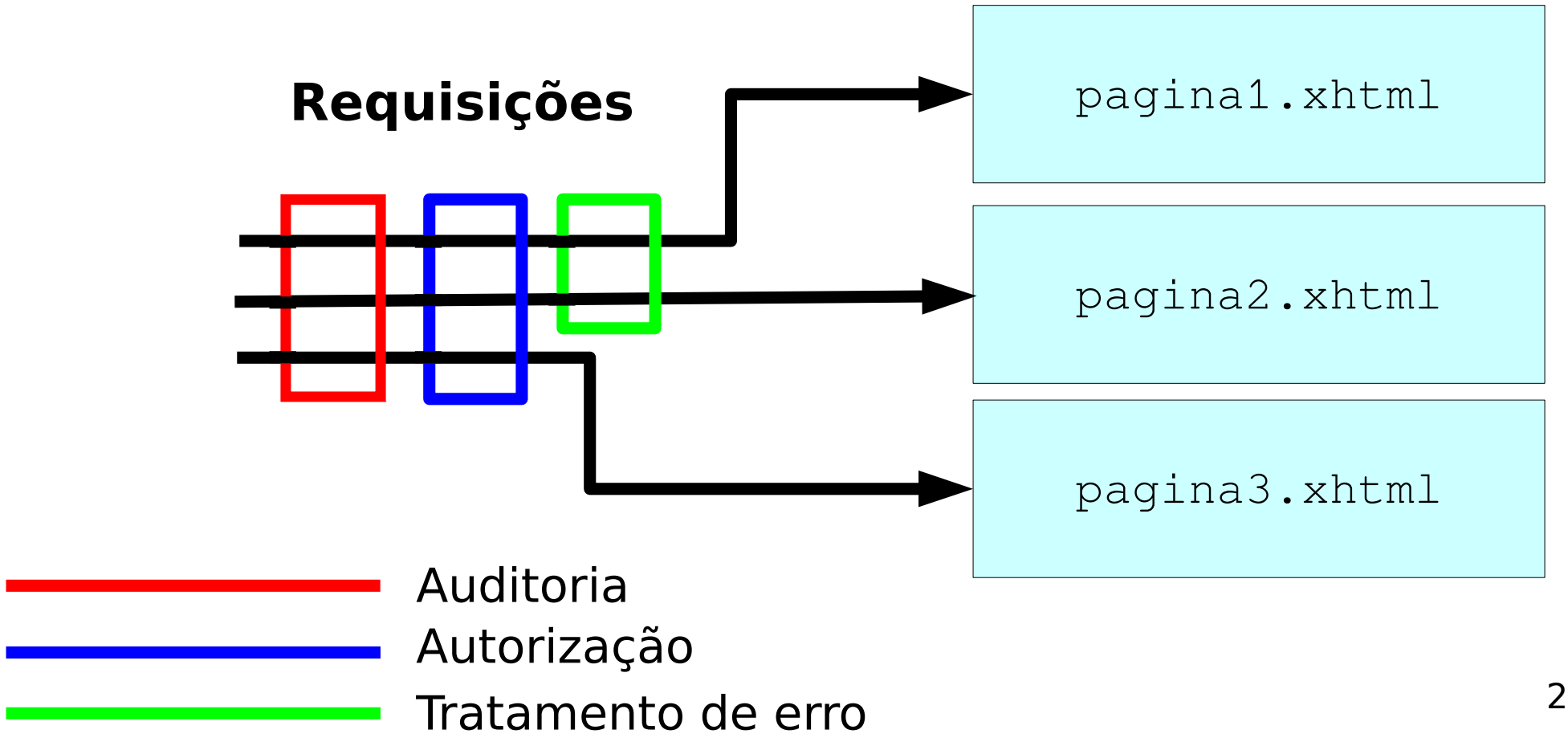
- Agindo assim, além da apresentação é preciso implementar os outros requisitos
- Isso deve ser feito em todas as páginas da aplicação
- Exemplo:



Reduzindo o acoplamento através de filtros (3/4)

- Filtros são classes que permitem que executemos código antes da requisição e também depois que a resposta foi gerada
- Uma boa analogia é pensar que os recursos são quartos em uma casa
- Para acessar um quarto é preciso passar por várias portas
- As portas são os filtros, onde você passa na ida e na volta

Reduzindo o acoplamento através de filtros (4/4)



Criando um filtro (1/6)

- Para criarmos um filtro, basta criar uma classe que implemente a interface `javax.servlet.Filter`
- Ao implementar a interface `Filter`, temos que implementar 3 métodos:
  - `init`
  - `destroy`
  - `doFilter`

Criando um filtro (2/6)

- O método que fará todo o processamento que queremos executar é o `doFilter`, que recebe três parâmetros:
  - `ServletRequest`
  - `ServletResponse`
  - `FilterChain`

Criando um filtro (3/6)

- Forma geral de um filtro:

```
public class ExemploDeFiltro implements Filter {

    public void init(FilterConfig filterConfig) { }

    public void destroy() { }

    public void doFilter(ServletRequest request,
        ServletResponse response, FilterChain chain)
            throws IOException, ServletException {
        // todo o processamento vai aqui
    }
}
```

Criando um filtro (4/6)

- A ideia do filtro é processar requisições, mas ele poderá fazer isso de maneira mais genérica para vários tipos de requisições
- Com um filtro podemos fechar “a porta”
- Esse poder vem do argumento `FilterChain` (a cadeia de filtros)
- Ele permite indicar ao *container* que a requisição deve prosseguir seu processamento
- Isso é feito com uma chamada ao método `doFilter` da classe `FilterChain`

Criando um filtro (5/6)

```
public void doFilter(ServletRequest request,
                     ServletResponse response,
                     FilterChain chain) throws
            IOException, ServletException {
   // passa pela porta
   chain.doFilter(request, response);
}
```

Criando um filtro (6/6)

- Um filtro não serve para processar toda a requisição, a ideia é ele interceptar várias requisições semelhantes, executar algo, mas depois permitir que o processamento normal da requisição prossiga
- Qualquer código colocado antes da chamada ao método `doFilter` será executado na ida, qualquer código depois, na volta
  - Com isso podemos fazer uma verificação de acesso antes da lógica, ou abrir um recurso antes e na volta fechar o mesmo

Registrando o filtro

- A única coisa que precisamos fazer para que o nosso filtro funcione é registrá-lo, para que o *container* saiba que ele precisa ser executado
  - Isso pode ser feito através do web.xml declarando o filtro e quais URLs serão filtradas
  - Ou através de uma anotação na própria classe que implementa o filtro

Registrando o filtro através de anotação

- A anotação `@WebFilter` pode ser usada na classe que implementa o filtro para registrá-lo
- Esta anotação possui 4 elementos principais
  - `filterName`: o nome do filtro
  - `urlPatterns`: URLs aos quais o filtro se aplica
  - `servletNames`: os nomes dos servlets aos quais o filtro se aplica
  - `dispatcherTypes`: tipos de requisição as quais o filtro se aplica

Exemplo de anotação para registrar um filtro

```
@WebFilter(filterName = "AutorizacaoFilter",
    urlPatterns = {"/faces/protected/*"},
    dispatcherTypes = {DispatcherType.REQUEST})
public class AutorizacaoFilter implements Filter {
   public void doFilter(ServletRequest request,
          ServletResponse response, FilterChain chain)
          throws IOException, ServletException {
      Usuario user = (Usuario)((HttpServletRequest)request).
          getSession().getAttribute("usuario");
      if (user != null && user.isAutorizado())
         chain.doFilter(request, response);
      else {
         String contextPath =
             ((HttpServletRequest)request).getContextPath();
         ((HttpServletResponse)response).
             sendRedirect(contextPath +
             "/faces/index.xhtml?msg=Faça o login!");
      }
   }
   ...
```

Referências

- GEARY, David; HORSTMANN, Cay. *Core JavaServer Faces*. 3. ed., Prentice-Hall, 2010.
- ORACLE Corporation. *The Java EE 7 Tutorial*. Disponível em: https://docs.oracle.com/javaee/7/JEETT.pdf, 2014.