



Desenvolvimento para Servidores-II

Ajax

Neste tópico abordaremos o uso de Ajax integrado a aplicações JSF

Prof. Ciro Cirne Trindade

Por quê usar Ajax? (1/3)

- Desvantagens das aplicações web
 - A interface é pobre
 - HTML é bom para documentos estáticos, mas fraco para aplicações web (HTML 5 é melhor)
 - Comunicação ineficiente
 - HTTP é um protocolo pobre para a forma que usamos aplicações web atualmente

Por quê usar Ajax? (2/3)

- Então por quê tudo mundo quer aplicações web?
 - Acesso universal
 - Todo mundo tem um navegador
 - Qualquer computador conectado a internet tem acesso ao conteúdo
 - Atualizações automáticas
 - O conteúdo vem do servidor, portanto ele nunca está desatualizado

Por quê usar Ajax? (3/3)

- Ajax resolve 3 problemas chave de aplicações web
 - Falta de suporte a atualizações granulares
 - Sincronismo: você congela enquanto espera uma resposta
 - Opções limitadas para elementos de interface

Aplicações web tradicionais x Aplicações Ajax

- Aplicação web tradicional: grandes atualizações infrequentes

- Aplicação Ajax: pequenas atualizações frequentes

Página web 1

Blah blah blah
Yada, yada, blah.
Blah blah blah
Yada, yada, blah.
Blah blah blah
Yada, yada, blah.

request



Servidor

Página web 2

Blah blah blah
Yada, yada, blah.
Blah blah blah
Yada, yada, blah.
Blah blah blah
Yada, yada, blah.

response
uma nova página

Página web

Blah blah blah
Yada, yada, blah.
Blah blah blah
Yada, yada, blah.
Blah blah blah
Yada, yada, blah.

request
via JavaScript



Servidor

response
pequena
porção de
informação
inserida na
página atual

Por quê Ajax em JSF? (1/2)

- Por quê uma biblioteca Ajax específica para JSF?
 - Já existem dezenas de bibliotecas Ajax (jQuery, DWR, GWT, etc.)

Por quê Ajax em JSF? (2/2)

- Vantagens da abordagem de uma biblioteca JSF específica para Ajax
 - Lado do cliente (*front end*)
 - Você pode atualizar elementos JSF (h:inputText, h:outputText, h:selectOneMenu, etc.)
 - Você não precisa codificar em JavaScript
 - Lado do servidor (*back end*)
 - Chamadas Ajax conhecem *managed beans*
 - Incluindo a leitura de campos do formulário e atualização das propriedades do *bean*

■ Forma mais simples

```
<h:commandButton ... action="...">
```

```
    <f:ajax render="id1"/>
```

```
</h:commandButton>
```

```
<h:outputText ... value="#{...}" id="id1"/>
```

- Quando o botão for pressionado, vá ao servidor, execute a ação, compute o valor do elemento JSF cujo id é "id1", envie este valor de volta para o cliente, e então substitua este elemento no DOM pelo novo valor

■ Forma geral

```
<h:commandButton ... action="...">  
    <f:ajax render="id1 id2"  
        execute="id3 id4"  
        event="umEvento"  
    />  
</h:commandButton>
```

■ Atributos

- render: os elementos que devem ser atualizados na página
 - O alvo do render deve estar no mesmo `h:form`
- execute: elementos que devem ser enviados para processamento no servidor
- event: o evento DOM a responder (keyup, blur, etc.)

O atributo render

- Forma geral

- `<f:ajax render="elementId" .../>`

- Ideia

- Id ou lista de ids separados por espaço de elementos JSF cujos valores devem ser devolvidos pelo servidor e atualizados no DOM
 - Esses elementos JSF devem estar no mesmo `h:form` de `f:ajax`

Exemplo: gerador de número sem Ajax

```
<h:form>
  <fieldset>
    <legend>Número aleatório: Sem Ajax
  </legend>
  <h:commandButton value="Mostrar número"
    action="#{numberGenerator.randomize}"/>
  <h2><h:outputText
    value="#{numberGenerator.number}"/>
  </h2>
</fieldset>
</h:form>
```

Exemplo: gerador de números com Ajax

```
<h:form>
  <fieldset>
    <legend>Número aleatório: Com Ajax</legend>
    <h:commandButton value="Mostrar número"
      action="#{numberGenerator.randomize}">
      <f:ajax render="number"/>
    </h:commandButton>
    <h2><h:outputText id="number"
      value="#{numberGenerator.number}"/>
    </h2>
  </fieldset>
</h:form>
```

Managed bean (1/2)

```
@ManagedBean
@RequestScoped
public class NumberGenerator {
    private int number;
    private int range;

    public NumberGenerator() {
        range = 100;
        number = new Random().nextInt(range);
    }

    public int getNumber() { return number; }

    public void setNumber(int number) {
        this.number = number;
    }
}
```

Managed bean (2/2)

```
public int getRange() {  
    return range;  
}  
  
public void setRange(int range) {  
    this.range = range;  
}  
  
public String randomize() {  
    number = new Random().nextInt(range);  
    return null;  
}  
}
```

O atributo execute

- Forma geral
 - `<f:ajax render="..." execute="..." .../>`
- Ideia
 - Um id ou lista de ids separados por espaço de elementos JSF que devem ser enviados para o servidor para execução
- 4 valores especiais
 - `@this`: o elemento pai de `f:ajax` (*default*)
 - `@form`: o formulário envolvendo `f:ajax`
 - `@none`: nada é enviado
 - `@all`: todos os elementos JSF da página

Exemplo: execute

```
<h:form>
  <fieldset>
    <legend>Número aleatório: Com Ajax (execute)
    </legend>
    Intervalo:
    <h:inputText value="#{numberGenerator.range}"
      id="range" /> <br/>
    <h:commandButton value="Mostrar número"
      action="#{numberGenerator.randomize}">
      <f:ajax render="number" execute="range" />
    </h:commandButton>
    <h2><h:outputText id="number"
      value="#{numberGenerator.number}" /></h2>
  </fieldset>
</h:form>
```

Usando execute="@form"

- Forma geral

```
<h:form>
```

```
...
```

```
<f:ajax render="elementId"
```

```
execute="@form" />
```

```
...
```

```
</h:form>
```

- Ideia

- Enviar todos os elementos do formulário para processamento no servidor

Exemplo: saldo da conta

```
<h:form>
  <fieldset>
    <legend>Saldo Bancário (execute="@form")
    </legend>
    <h:panelGrid columns="2">
      Número da Conta:
      <h:inputText value="#{customerAjax.account}"/>
      Senha:
      <h:inputSecret value="#{customerAjax.password}"/>
    </h:panelGrid>
    <h:commandButton value="Consultar saldo"
      action="#{customerAjax.showBalance}"
      <f:ajax render="mensagem" execute="@form"/>
    </h:commandButton><br/>
    <h2><h:outputText value="#{customerAjax.message}"
      id="mensagem"/></h2>
  </fieldset>
</h:form>
```

Classe Customer (1/3)

```
public class Customer {  
    private int account;  
    private String password;  
    private String firstName;  
    private String lastName;  
    private float balance;  
  
    public Customer() {  
    }  
  
    public Customer(int account, String password,  
                    String firstName, String lastName,  
                    float balance) {  
        this.account = account;  
        this.password = password;  
        this.firstName = firstName;  
        this.lastName = lastName;  
        this.balance = balance;  
    }  
}
```

Classe Customer (2/3)

```
public int getAccount() { return account; }

public void setAccount(int account) {
    this.account = account;
}

public float getBalance() { return balance; }

public void setBalance(float balance) {
    this.balance = balance;
}

public String getPassword() { return password; }

public void setPassword(String password) {
    this.password = password;
}
```

Classe Customer (3/3)

```
public String getFirstName() {  
    return firstName;  
}  
  
public void setFirstName(String firstName) {  
    this.firstName = firstName;  
}  
  
public String getLastName() {  
    return lastName;  
}  
  
public void setLastName(String lastName) {  
    this.lastName = lastName;  
}
```

```
}
```



Managed Bean CustomerAjax (1/2)

```
@ManagedBean
@RequestScoped
public class CustomerAjax extends Customer {
    private String message;

    public CustomerAjax() {
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```



Managed Bean

CustomerAjax (2/2)

```
public String showBalance() {
    CustomerTable service = new CustomerTable();
    Customer c = service.findCustomer(getAccount());
    if (c == null) {
        message = "Conta desconhecida";
    } else {
        if (!getPassword().equals(c.getPassword())) {
            message = "Senha incorreta";
        } else {
            message =
                String.format("Saldo de %s %s é R$ %.2f",
                    c.getFirstName(), c.getLastName(),
                    c.getBalance());
        }
    }
    return null;
}
```


Classe CustomerTable

```
class CustomerTable {  
    private List<Customer> customers;  
  
    public CustomerTable() {  
        customers = new ArrayList<Customer>();  
        customers.add(new Customer(1000, "123", "Ciro",  
                                    "Trindade", 500.0f));  
        customers.add(new Customer(1001, "321", "Fernando",  
                                    "Macedo", 10000.0f));  
    }  
  
    public Customer findCustomer(int account) {  
        for (Customer c : customers)  
            if (c.getAccount() == account)  
                return c;  
        return null;  
    }  
}
```

O atributo event (1/2)

- Forma geral
 - `<f:ajax render="..." event="..." .../>`
- Ideia
 - O nome do evento DOM a responder (não inclua “on”, então é mouseover, keyup, etc.)
- Detalhes
 - Se não for especificado, o evento *default* é usado
 - Eventos de alto nível
 - JSF adiciona 2 eventos extra: `action` e `valueChange`

O atributo event (2/2)

- Eventos *default*
 - action
 - `h:commandButton`, `h:commandLink`
 - valueChange
 - `h:inputText`, `h:inputSecret`,
`h:inputTextarea`, todos os radio button,
checkbox e menus (`h:selectOneMenu`, etc.)

Exemplo: caixas de seleção encadeadas

- Ideia
 - Usar um `h:selectOneMenu` para criar uma lista dos estados brasileiros
 - Quando o usuário selecionar um estado, uma lista de cidades do estado é exibida (novamente, usando `h:selectOneMenu`)
 - Quando uma cidade é selecionada, a população desta cidade é exibida

Página JSF (index.jsf)

```
<h:form>
  <fieldset>
    <legend>Consulta a População</legend>
    <h:panelGrid columns="2">
      Estado:
      <h:selectOneMenu value="#{localizacao.estado}">
        <f:selectItems value="#{localizacao.estados}" />
        <f:ajax render="listaCidades" />
      </h:selectOneMenu>
      Cidade:
      <h:selectOneMenu value="#{localizacao.cidade}"
        disabled="#{localizacao.listaEstadosDesabilitada}"
        id="listaCidades">
        <f:selectItems value="#{localizacao.cidades}" />
        <f:ajax render="populacao" />
      </h:selectOneMenu>
      População:
      <h:outputText value="#{localizacao.cidade}" id="populacao" />
    </h:panelGrid>
  </fieldset>
</h:form>
```

Managed Bean Localizacao

(1/4)

```
@ManagedBean
@SessionScoped
public class Localizacao {
    private String estado;
    private String cidade;
    private boolean listaCidadesDesabilitada;

    public Localizacao() {
        listaCidadesDesabilitada = true;
    }

    public String getEstado() { return estado; }

    public void setEstado(String estado) {
        this.estado = estado;
        listaCidadesDesabilitada = false;
    }
}
```

```
public String getCidade() {  
    return cidade;  
}  
  
public void setCidade(String cidade) {  
    this.cidade = cidade;  
}  
  
public boolean isListaCidadesDesabilitada() {  
    return listaCidadesDesabilitada;  
}  
  
public void setListaCidadesDesabilitada(  
    boolean listaCidadesDesabilitada) {  
    this.listaCidadesDesabilitada =  
        listaCidadesDesabilitada;  
}
```

```
public List<SelectItem> getEstados() {  
    List<SelectItem> estados =  
        new ArrayList<SelectItem>();  
    estados.add(new SelectItem(  
        "Selecione o estado"));  
    for (EstadoInfo s : EstadoInfo.getEstados()) {  
        estados.add(  
            new SelectItem(s.getSigla()));  
    }  
    return estados;  
}
```


Managed Bean Localizacao

(4/4)

```
public SelectItem[] getCidades() {  
    SelectItem[] cidades = {  
        new SelectItem("Selecione a cidade")};  
    if (!listaEstadosDesabilitada &&  
        estado != null) {  
        for (EstadoInfo s :  
            EstadoInfo.getEstados()) {  
            if (s.getSigla().equals(estado)) {  
                cidades = s.getCidades();  
                break;  
            }  
        }  
    }  
    return cidades;  
}
```

Classe auxiliar EstadoInfo

(1/2)

```
public class EstadoInfo {
    private String sigla;
    private SelectItem[] cidades;
    private static EstadoInfo[] estados = {
        new EstadoInfo("SP",
            new SelectItem("Selecione a cidade"),
            new SelectItem("34953", "Adamantina"),
            new SelectItem("224551", "Americana"),
            new SelectItem("222036", "Araraquara"),
            new SelectItem("433153", "Santos"),
            new SelectItem("11821876", "São Paulo"))
        // outros estados
    };

    public EstadoInfo(String sigla,
        SelectItem... cidades) {
        this.sigla = sigla;
        this.cidades = cidades;
    }
}
```

Classe auxiliar EstadoInfo

(2/2)

```
public String getSigla() { return sigla; }

public void setSigla(String sigla) {
    this.sigla = sigla;
}

public SelectItem[] getCidades() { return cidades; }

public void setCidades(SelectItem[] cidades) {
    this.cidades = cidades;
}

public static EstadoInfo[] getEstados() {
    return estados;
}

public static void setEstados(EstadoInfo[] estados) {
    EstadoInfo.estados = estados;
}
}
```

Referências

- GEARY, David; HORSTMANN, Cay. *Core JavaServer Faces*. 3. ed., Prentice-Hall, 2010.
- HALL, Marty, *JSF 2: Integrated Ajax Support*. Disponível em:
<http://courses.coreservlets.com/Course-Materials/pdf/jsf/jsf2/JSF2-Ajax.pdf>, 2013.
- ORACLE Corporation. *The Java EE 7 Tutorial*. Disponível em:
<https://docs.oracle.com/javaee/7/JEETT.pdf>, 2014.