Java Server Faces

Eventos

JSF-EL

Exercicio

Alteracoes e atualizacoes: Gustavo Wagner © *2008*
gugawag@gmail.com

Slides originais gentilmente cedidos por: Frederico Costa Guedes Pereira © 2007
fredguedespereira@gmail.com

Eventos e tratadores

- A interação do usuários com os componentes visuais JSF produzem ***eventos***
  - JSF abstrai o ciclo de requisição e resposta, típico em aplicações web, com ***componentes*** e ***eventos*** (e seus tratadores).

"Em aplicações JSF, integrar a lógica da aplicação é uma questão de associar tratadores de eventos apropriados aos componentes que os geram."

Eventos e tratadores

- Tipos de eventos:
  - ***Value-changed***: gerados por componentes quando o usuário muda o valor do componente.
  - ***Action***: gerados quando o usuário ativa um componente de comando (botão, âncoras de hiperlinks).
  - ***Data model***: gerados quando um componente de dados seleciona uma linha para processamento.
  - ***Phase***: gerados pelo JSF ao longo do atendimento da requisição
  - ***Eventos definidos pelo usuário***: o usuário pode estender o mecanismo e criar seus próprios eventos.

Eventos e tratadores

- Eventos ***Value-changed***
  - São manipulados/tratados por **valueChangedListeners**

```
<h:inputText  valueChangeListener="#{bb.valueChanged}"/>
<h:panelGrid binding="#{bb.changePanel}" rendered="false">

</h:panelGrid>
```

```
public void valueChanged  (ValueChangeEvent event) {
      HtmlInputText sender =
          (HtmlInputText) event.getComponent();
      sender.setReadonly(true);
      changePanel.setRendered(true);
}
```

método do backing bean

Eventos e tratadores

- Eventos ***Action***
  - Há dois tipos de manipuladores: os que afetam e os que não afetam a navegação
- Listeners que **afetam** a navegação
  - Fazem processamento e devolvem um "resultado" que é usado para descobrir a próxima página
- Listeners que **não afetam** a navegação
  - Manipulam componentes, processam backing beans, objetos de negócio, etc. mas não alteram a página corrente (que é reapresentada)

Eventos e tratadores

- Exemplos:

Resultado estático

```
<h:commandButton value="Login"
        action="success" immediate="true"/>
```

  - Ao pressionar o botão, um evento ***action*** é lançado e tratado pelo *listener default*. A navegação é decidida com base no resultado "success" gerado.

```
<h:commandButton value="Login"
        action="#{loginForm.login}"/>
```

  - Idem, só que o resultado será gerado pelo metodo login() do *backing bean* loginForm

Resultado dinâmico

Eventos e tratadores

- A classe do *backing bean*

```
public class LoginForm {
        ...
        public String login() {
            if (...) {          // login is successful
                return "success";
            } else {
                return "failure";
            }
        }
...
}
```

Eventos e tratadores

- Outros exemplos:

```
<h:commandButton id="criarAluno"
      type="submit" value="Criar"
      actionListener="#{myForm.doIt}" />
```

  - O método **doIt** do *backing bean* myForm é executado quando o botão é pressionado e a página é atualizada

```
public void doIt (ActionEvent event) {
      // ... salva o que tem que salvar
      HtmlCommandButton button =
      (HtmlCommandButton) event.getComponent();
      button.setValue("Salvar");
}
```

Eventos e tratadores

- Eventos ***Data model***
  - Gerados por um objeto DataModel (adaptador de fontes de dados para um componente HtmlDataTable)
  - São registrados via programação e devem implementar a interface DataModelListener
- Eventos ***Phase***
  - Gerados pelo JSF durante uma das 6 fases do ciclo de processamento da requisição
  - São registrados via programação e devem implementar a interface PhaseListener

Mensagens

- Usadas para apresentar mensagens de erros cometidos pelos usuários durante preenchimento de campos ou informações quaisquer.
- Possuem as seguintes propriedades:
  - Texto resumido da mensagem
  - Texto detalhado da mensagem
  - Nível de severidade
- Podem ser adicionadas por quaisquer elementos da aplicação (componentes, validadores, conversores, etc)

Mensagens

- Exemplo:

```
<h:message id="errors" for="helloInput"
      style="color: red"/>
```

Componente ao qual está associada

Navegação

- Mecanismo declarativo que permite a navegação entre as páginas da aplicação web

faces-config.xml

```
<navigation-rule>
    <description>
        Navegação a partir da página hello.jsp
    </description>
    <from-view-id>/hello.jsp</from-view-id>
    <navigation-case>
        <from-outcome>success</from-outcome>
        <to-view-id>/goodbye.jsp</to-view-id>
    </navigation-case>
</navigation-rule>
```

A linguagem JSF EL

- Baseada na EL de JSP 2.0, com algumas diferenças:
  - Uso de #{ } ao invés de ${ }
  - Podem atualizar propriedades dos objetos
  - Podem referenciar métodos dos objetos
  - Não existe o escopo **page** na JSF-EL
  - Não necessitam de JSP para serem avaliadas
  - Não suportam funções-EL como o JSP
  - Definem mais dois objetos implícitos: **facesContext** e **view**

Mais sobre EL: Resumo de JSF-EL (retirado do livro JSF in Action)

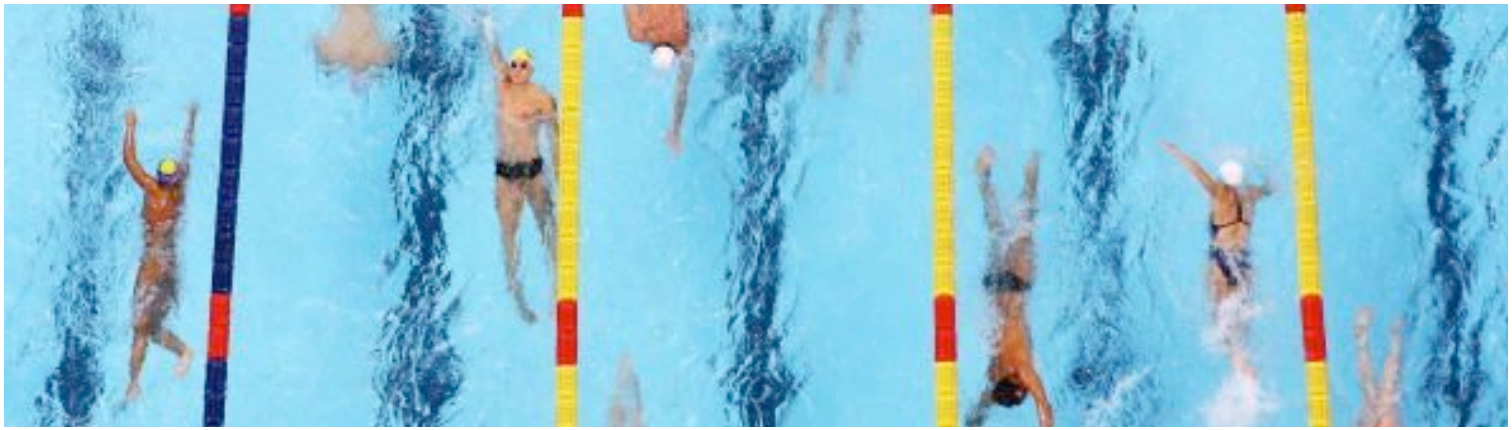
Example	Description
`#{myBean.value}`	Returns the `value` property of the object stored under the key `myBean`, or the element stored under the key `value` if `myBean` is a `Map`.
`#{myBean['value']}`	Same as `"{#myBean.value}"`.
`#{myArrayList[5]}`	Returns the fifth element of a `List` stored under the key `myArrayList`.
`#{myMap['foo']}`	Returns the object stored under the key `foo` from the `Map` stored under the key `myMap`.
`#{myMap[foo.bar]}`	Returns the object stored under the key that equals the value of the expression `foo.bar` from the `Map` stored under the key `myMap`.
`#{myMap['foo'].value}`	Returns the `value` property of the object stored under the key `foo` from the `Map` stored under the key `myMap`.
`#{myMap['foo'].value[5]}`	Returns the fifth element of the `List` or array stored under the key `foo` from the `Map` stored under the key `myMap`.
`#{myString}`	Returns the `String` object stored under the key `myString`.
`#{myInteger}`	Returns the `Integer` object stored under the key `myInteger`.

Cont.: Mais sobre EL: Resumo de JSF-EL

`#{user.role == 'normal'}`	Returns `true` if the `role` property of the object stored under the key `user` equals normal. Returns `false` otherwise.
`#{(user.balance - 200) == 0}`	If the value of the `balance` property of the object stored under the key user minus 200 equals zero, returns `true`. Returns `false` otherwise.
`Hello #{user.name}!`	Returns the string "Hello" followed by the `name` property of the object stored under the key `user`. So if the user's name is Sean, this would return "Hello Sean!"
`You are #{(user.balance > 100) ? 'loaded' : 'not loaded'}`	Returns the string "You are loaded" if the `balance` property of the object stored under the key `user` is greater than 100; returns "You are not loaded" otherwise.
`#{myBean.methodName}`	Returns the method called `method` of the object stored under the key `myBean`.
`#{20 + 3}`	Returns 23.

Operadores em JSF EL (retirado do livro JSF in Action)

Syntax	Alternative	Operation
.		Access a bean property, method, or `Map` entry
[]		Access an array or `List` element, or `Map` entry
()		Creates a subexpressions and controls evaluation order
? :		Conditional expression: `ifCondition ? trueValue : falseValue`
+		Addition
-		Subtraction and negative numbers
*		Multiplication
/	div	Division
%	mod	Modulo (remainder)
==	eq	Equals (for objects, uses the `equals()` method)
!=	ne	Not equal
<	lt	Less than
>	gt	Greater than
<=	le	Less than or equal
>=	ge	Greater than or equal
&&	and	Logical AND
||	or	Logical OR



Exercicio

Jogo Master

- Seu objetivo eh desenvolver os JSFs do Jogo Master. Para isso, use os conhecimentos em EL obtidos ateh agora e mais o componente *HTMLDataTable* descrito abaixo:

```
<h:dataTable value="#{coisaBean.nomes}" var="nome">
    <h:column>
            <h:outputText value="#{nome}"/>
   </h:column>
 </h:dataTable>
```

- No exemplo acima, coisaBean.nomes eh uma lista de strings. O *value* do HTMLDataTable receve uma lista e, para cada elemento dessa lista, ele coloca na variavel definida em *var* (no caso acima, nome)

Jogo Master

- Para esse jogo, jah foram desenvolvidos algumas classes: Carta, que representa uma carta com perguntas e respostas (lista de strings), e classe LeitorCartas, que lhe devolve uma determinada Carta;
- As telas do jogo devem parecer com as telas mostradas a seguir;

Tela Inicial

Gerar cartao...

Créditos: Gustavo Wagner(gugawag@gmail.com) e Frederico Guedes Pereira (fredguedespereira@gmail.com)

Tela apos ter clicado no botao Gerar Cartao da Tela Inicial

Créditos: Gustavo Wagner(gugawag@gmail.com) e Frederico Guedes Pereira (fredguedespereira@gmail.com)

Perguntas do cartao c3:

1)Qual a diferença entre HttpServlet e GenericServlet?
2)Diga a assinatura completa de 3 métodos de HttpServletRequest e para que servem.
3) Qual o ciclo de vida de um servlet?
4) Quais as duas formas de definir o tempo máximo de inatividade de uma sessão?
5) Explique a diretiva taglib.

Possiveis respostas... Gerar novo cartao...

Tela mostrada apos clicar no botao Possiveis Respostas

Créditos: Gustavo Wagner(gugawag@gmail.com) e Frederico Guedes Pereira (fredguedespereira@gmail.com)

Perguntas do cartao c3:

1)Qual a diferença entre HttpServlet e GenericServlet?
2)Diga a assinatura completa de 3 métodos de HttpServletRequest e para que servem.
3) Qual o ciclo de vida de um servlet?
4) Quais as duas formas de definir o tempo máximo de inatividade de uma sessão?
5) Explique a diretiva taglib.

Possiveis Respostas:

Sem resposta.
Sem resposta.
Sem resposta.
Sem resposta.
Sem resposta.

Gerar novo cartao...

Arquivo de Perguntas e Respostas

- O arquivo de perguntas e respostas se encontra em: {diretorio_projeto}/WebContent/resources/ cartoes.properties