23.1. ValueChangeEvent no JSF

✅ O que é?

O `ValueChangeEvent` é um evento disparado por **componentes do tipo `UIInput`** sempre que seu valor for alterado. Ele é frequentemente utilizado para reagir a seleções feitas pelo usuário em **combos (`<h:selectOneMenu>`)**, campos de texto, entre outros.

Um uso comum é **atualizar a interface com base em uma escolha feita** pelo usuário, como, por exemplo, ao selecionar um curso de uma lista.

🧩 Exemplo com `valueChangeListener` e Submissão da Página

No JSF 2.0, podemos utilizar o atributo `valueChangeListener` para capturar alterações diretamente no componente.

📄 **Código da Página JSF:**

```
<f:view>
   <h:head>
      <title>ValueChangeListener</title>
   </h:head>

   <h:form>
      <table border="1">
         <tr>
            <td>
               <h:selectOneMenu value="#{aluno.curso}"
                              onchange="submit()"
                              valueChangeListener="#{aluno.valueChangeCurso}">
                  <f:selectItems value="#{aluno.cursos}" />
               </h:selectOneMenu>
            </td>
            <td>
               <h:outputText value="#{aluno.curso}" />
            </td>
         </tr>
      </table>
   </h:form>
</f:view>
```

📌 **Bean `Aluno.java`:**

```
public void valueChangeCurso(ValueChangeEvent evt) {
   aluno.setCurso(evt.getNewValue().toString());
```

}

> **Importante:** esse modelo exige **submissão total da página** (full submit), o que pode não ser ideal em aplicações modernas.

⚡ Exemplo com Ajax (Assíncrono)

Para tornar a interação mais fluida, podemos usar o `<f:ajax>` para que a mudança de valor seja capturada e o resultado exibido **sem recarregar toda a página**.

📄 Código JSF com Ajax (RichFaces ou PrimeFaces):

```
<f:view>
   <h:head>
      <title>ValueChangeListener com Ajax</title>
   </h:head>

   <h:form>
      <a4j:region>
         <table border="1">
            <tr>
               <td>
                  <h:selectOneMenu value="#{aluno.curso}">
                     <f:selectItems value="#{aluno.cursos}" />
                     <f:ajax listener="#{aluno.valueChangeCurso2}" render="resultado" />
                  </h:selectOneMenu>
               </td>
               <td>
                  <h:outputText id="resultado" value="#{aluno.curso}" />
               </td>
            </tr>
         </table>
      </a4j:region>
   </h:form>
</f:view>
```

📌 Bean `Aluno.java` com Ajax:

```
public void valueChangeCurso2(AjaxBehaviorEvent evt) {
   String s = evt.getComponent().getAttributes().get("value").toString();
   aluno.setCurso(s);
}
```

O Ajax permite **atualizações parciais de página**. O atributo `render="resultado"` indica o ID do componente que deve ser atualizado após a execução do evento.

📌 Observações Técnicas

- `valueChangeListener` é chamado **após a fase de Aplicar Valores Requisitados**, mas **antes da Atualização do Modelo**.
- O atributo `onchange="submit()"` usado no exemplo tradicional força o envio do formulário.
- Para Ajax, é recomendado o uso de bibliotecas como **RichFaces** ou **PrimeFaces**, que oferecem suporte mais robusto e componentes ricos.

23.2. Action Event (Evento de Ação)

✅ O que é?

O **ActionEvent** é disparado por **componentes do tipo `UICommand`**, ou seja, componentes que têm a capacidade de disparar uma ação, como:

- `<h:commandButton />`
- `<h:commandLink />`

Esses eventos são usados para **executar métodos no Bean** quando o usuário realiza uma ação (clicar em um botão ou link, por exemplo).

🧩 Exemplo: Usando `h:commandButton`

📄 **Página JSF:**

```
<h:form>
   <h:commandButton value="Enviar" action="#{aluno.salvar}" />
</h:form>
```

📌 **Bean:**

```
public String salvar() {
   // Lógica de persistência ou processamento
   System.out.println("Aluno salvo!");
   return "paginaSucesso"; // Navega para outra página
}
```

> O método associado ao `action` deve **retornar uma `String`** indicando a navegação. Retornar `null` mantém a mesma página.

✅ Dica adicional:

- Use `actionListener` se quiser mais controle sobre o evento antes da navegação:

```
<h:commandButton value="Salvar" actionListener="#{aluno.processar}" />

public void processar(ActionEvent event) {
   // Executa lógica antes da navegação
```

}

23.3. Phase Event (Eventos de Fase)

✅ O que é?

O **PhaseEvent** é disparado **em cada fase do ciclo de vida** do JSF. Isso permite interceptar e executar lógica **antes ou depois de cada fase**, como:

- Restaurar visão
- Aplicar valores
- Validar entrada
- Atualizar modelo
- Invocar lógica de negócio
- Renderizar resposta

🧩 Exemplo: Criando um `PhaseListener`

Você pode usar um `PhaseListener` para depurar ou fazer verificações específicas em fases do ciclo de vida.

📌 Classe `LogPhaseListener.java`

```
import javax.faces.event.PhaseEvent;
import javax.faces.event.PhaseId;
import javax.faces.event.PhaseListener;
import java.util.logging.Logger;

public class LogPhaseListener implements PhaseListener {

    private static final Logger logger = Logger.getLogger(LogPhaseListener.class.getName());

    @Override
    public void afterPhase(PhaseEvent event) {
        logger.info("Depois da fase: " + event.getPhaseId());
    }

    @Override
```

```
    public void beforePhase(PhaseEvent event) {
        logger.info("Antes da fase: " + event.getPhaseId());
    }

    @Override
    public PhaseId getPhaseId() {
        return PhaseId.ANY_PHASE; // Escuta todas as fases
    }
}
```

✅ Registro do PhaseListener no `faces-config.xml`:

```
<lifecycle>
    <phase-listener>com.seuprojeto.LogPhaseListener</phase-listener>
</lifecycle>
```

> Isso fará com que seu `LogPhaseListener` capture **todas as fases do ciclo de vida** e imprima logs, o que é útil para depuração ou execução de lógica personalizada (como controle de acesso, medição de tempo, etc.).

🧠 Conclusão

Evento	Disparado por	Uso principal
ValueChangeEvent	Componentes de entrada (UIInput)	Captura mudanças de valor (ex: combo)
ActionEvent	Componentes de ação (UICommand)	Executa lógica ao clicar em botão ou link
PhaseEvent	JSF Lifecycle	Monitorar e intervir no ciclo de vida