

## Tópicos Abordados

- Ciclo de vida de requisições
- O que são eventos e quais os tipos
  - Action events
  - Value change events
  - Phase Events
  - System Events
- O atributo *immediate*

- ## Tópicos Abordados
- Ciclo de vida de requisições
  - O que são eventos e quais os tipos
    - Action events
    - Value change events
    - Phase Events
    - System Events
  - O atributo *immediate*

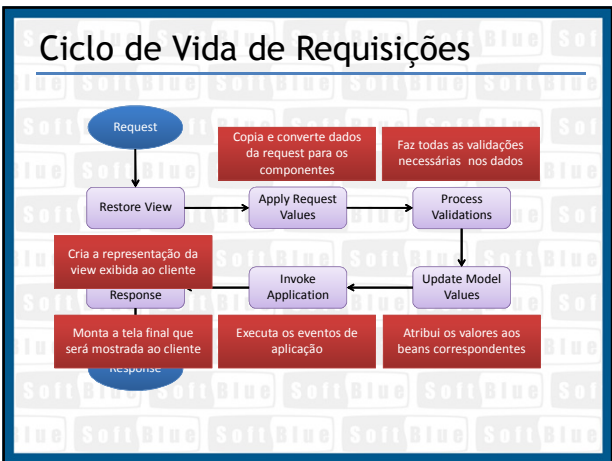
# Ciclo de Vida de Requisições

```
graph TD; Request([Request]) --> RestoreView[Restore View]; RestoreView --> ApplyRequest[Apply Request Values]; ApplyRequest --> ProcessValidations[Process Validations]; ProcessValidations --> UpdateModel[Update Model Values]; UpdateModel --> InvokeApplication[Invoke Application]; InvokeApplication --> Response([Response]); Response --> RestoreView;
```

O diagrama ilustra o ciclo de vida de uma requisição em uma aplicação web, apresentando os seguintes componentes e fluxos:

- Request** (Início da requisição)
- Restore View** (Recupera a representação da view exibida ao cliente)
- Apply Request Values** (Copia e converte dados da request para os componentes)
- Process Validations** (Faz todas as validações necessárias nos dados)
- Update Model Values** (Atualiza os valores dos beans correspondentes)
- Invoke Application** (Executa os eventos de aplicação)
- Response** (Monta a tela final que será mostrada ao cliente)

O fluxo segue a seguinte sequência: Request → Restore View → Apply Request Values → Process Validations → Update Model Values → Invoke Application → Response → Restore View.



## Eventos e Tipos de Eventos

- Eventos em JSF podem ser disparados
  - Em resposta a ações de usuários
  - Pelo próprio JSF
- Eventos são processados no servidor
- Existem 4 tipos de eventos
  - Action Events
  - Value Change Events
  - Phase Events
  - System Events

---

---

---

---

---

---

---

## Action Events

- Eventos disparados quando ocorre um clique em um componente que gera uma ação, como *h:commandLink* ou *h:commandButton*

```
<h:commandButton value="OK"
  actionListener="#{bean.botaoClicado}" />
```

O método `botaoClicado()` será invocado no bean

```
public void botaoClicado(ActionEvent event) {
  //...
}
```

Declaração no bean

---

---

---

---

---

---

---

## Action Events

- Outra opção é criar um listener

```
public class MyListener implements ActionListener {
  public void processAction(ActionEvent event)
    throws AbortProcessingException {
    //...
  }
}
```

- A tag *f:actionListener* é utilizada para referenciar o listener criado

```
<h:commandButton value="OK">
  <f:actionListener type="app.MyListener" />
</h:commandButton>
```

---

---

---

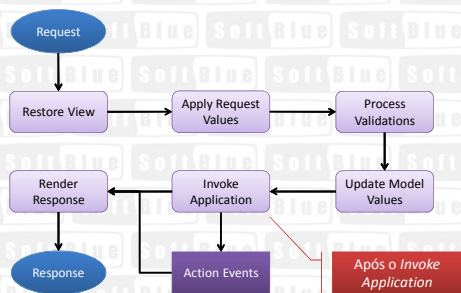
---

---

---

---

## Action Events no Ciclo de Vida



## Value Change Events

- Eventos disparados quando ocorre uma mudança de valor em algum componente, como `h:inputText`, `h:selectOneMenu`, etc.

```
<h:selectOneMenu value="#{bean.valor}" onchange="submit();"
  valueChangeListener="#{bean.mudouValor}">
```

O método `mudouValor()` será invocado no bean

É preciso invocar o `submit()`

```
public void mudouValor(ValueChangeEvent event) {
    //...
}
```

Declaração no bean

## Value Change Events

- Outra opção é criar um listener

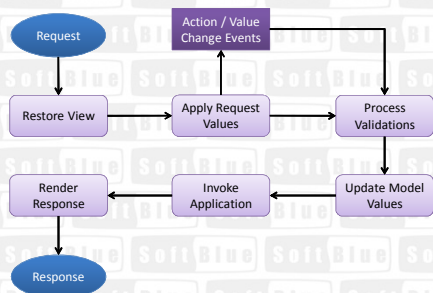
```
public class MyListener implements ValueChangeListener {
    public void processValueChange(ValueChangeEvent event)
        throws AbortProcessingException {
        //...
    }
}
```

- A tag `f:valueChangeListener` é utilizada para referenciar o listener criado

```
<h:selectOneMenu value="#{bean.valor}" onchange="submit();"
  <f:valueChangeListener type="app.MyListener" />
  ...
</h:selectOneMenu>
```



## Immediate e o Ciclo de Vida



## Ignorando a Validação

- Para um value change event ignorar a validação, duas coisas devem acontecer
  - O *immediate* deve ser habilitado no componente
  - O código do evento deve forçar a ida para a fase *Render Response*

```
<h:selectOneMenu onchange="submit();"
    valueChangeListener="#{bean.mudouValor}" immediate="true">
...
</h:selectOneMenu>
```

```
public void mudouValor(ValueChangeEvent event) {
    //...
    FacesContext context = FacesContext.getCurrentInstance();
    context.renderResponse();
}
```

## Ignorando a Validação

- Para um action event, basta habilitar o atributo *immediate* no componente que gera a ação
  - Não é preciso chamar a fase *Render Response*, pois isto é feito automaticamente

## Phase Events

- Gerados pelo JSF quando ocorrem mudanças de uma fase para outra
- São gerados dois eventos por fase
  - O primeiro logo antes de iniciar a fase
  - O segundo logo após o término da fase
- Este tipo de evento está associado a uma view, e não a um componente

---

---

---

---

---

---

---

## Listeners para Phase Events

- Um listener deve implementar **PhaseListener**
- Pode ser configurado de várias formas

Em uma view específica

```
<f:phaseListener type="app.MyPhaseListener" />
```

Globalmente, para todas as views

```
<faces-config>
  <lifecycle>
    <phase-listener type="app.MyListener" />
  </lifecycle>
</faces-config>
```

---

---

---

---

---

---

---

## Criando um PhaseListener

Antes da fase iniciar

```
public class MyPhaseListener implements PhaseListener {
    public void beforePhase(PhaseEvent event) {
        //...
    }
    public void afterPhase(PhaseEvent event) {
        //...
    }
    public PhaseId getPhaseId() {
        return PhaseId.ANY_PHASE;
    }
}
```

Assim que a fase terminar

Fase de interesse

---

---

---

---

---

---

---

## System Events

- Eventos de diversos tipos gerados pelo JSF e que podem ser interceptados pelas aplicações
- Estes eventos podem estar associados normalmente a:
  - *Application*
  - *UIComponent*
  - *UIViewRoot*

## Início e Término da Aplicação

- Dois system events indicam quando a aplicação foi inicializada e quando ela está prestes a ser terminada
  - *PostConstructApplicationEvent*
  - *PreDestroyApplicationEvent*
- Estes eventos estão associados à aplicação como um todo

## Início e Término da Aplicação

```
public class MySystemEventListener implements SystemEventListener {  
    public boolean isListenerForSource(Object source) {  
        return source instanceof Application;  
    }  
    public void processEvent(SystemEvent event)  
        throws AbortProcessingException {  
        ...  
    }  
}
```

O listener implementa *SystemEventListener*

Tipo de origem do evento

Processamento do evento

## Início e Término da Aplicação

- A configuração de um system event associado à aplicação (Application) é feita no arquivo *faces-config.xml*

```
<application>
  <system-event-listener>
    <system-event-listener-class>
      app.MySystemEventListener
    </system-event-listener-class>
    <system-event-class>
      javax.faces.event.PostConstructApplicationEvent
    </system-event-class>
  </system-event-listener>
</application>
```

## Pré-Renderização da View

- O JSF gera um system event momentos antes de uma view ser renderizada
  - *PreRenderViewEvent*
- Este evento está associado a UIViewRoot
- Bastante útil na prática
  - Verificar acesso à página
  - Carregar informações para serem exibidas na view
  - etc.

## Pré-Renderização da View

- A tag **f:event** é utilizada na view

```
<f:event type="preRenderView"
  listener="#{bean.carregarDados}" />
```

- O método é implementado no bean

```
public void carregarDados(ComponentSystemEvent event) {
  //...
}
```



## Colocando em Prática...



Agora que você já aprendeu a teoria, acesse as vídeo-aulas práticas e pratique os assuntos abordados neste módulo!

[Clique aqui para acessar as vídeo-aulas práticas](#)

---

---

---

---

---

---

---

---