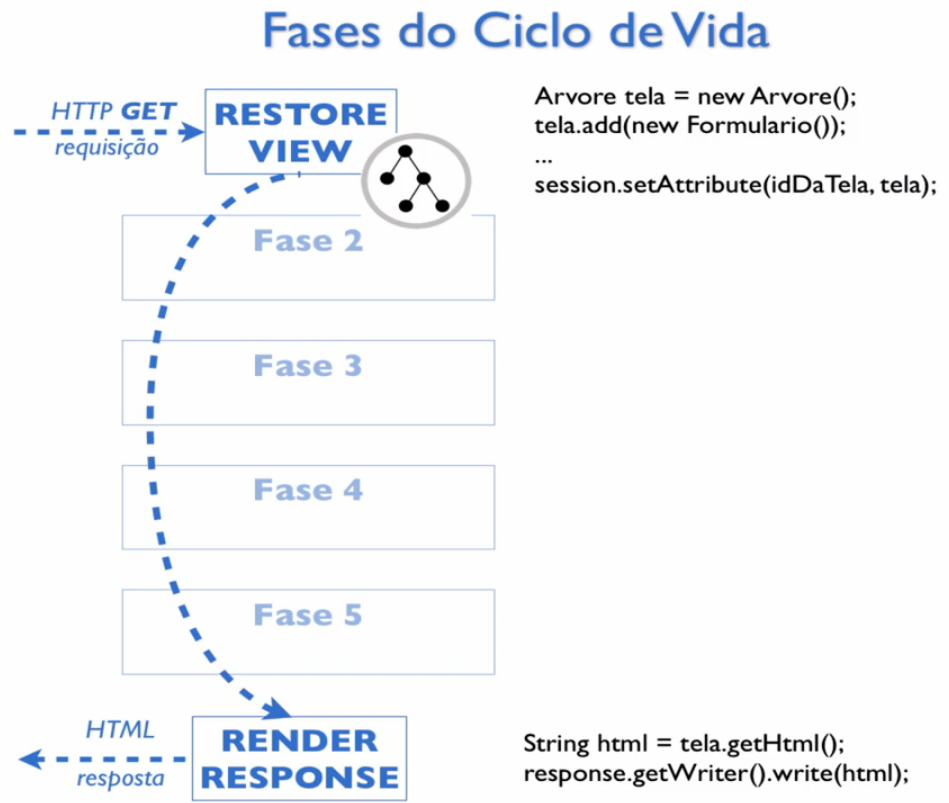
**Criação da view e renderização: RESTORE\_VIEW e RENDER\_RESPONSE**

Ao receber uma requisição HTTP do tipo GET o controlador iniciou o ciclo da vida da tela. Isso significa que ele leu o arquivo xhtml e instanciou todos os componentes. Como isso foi disparado pela requisição inicial está claro que não há nada a fazer além de renderizar a resposta.



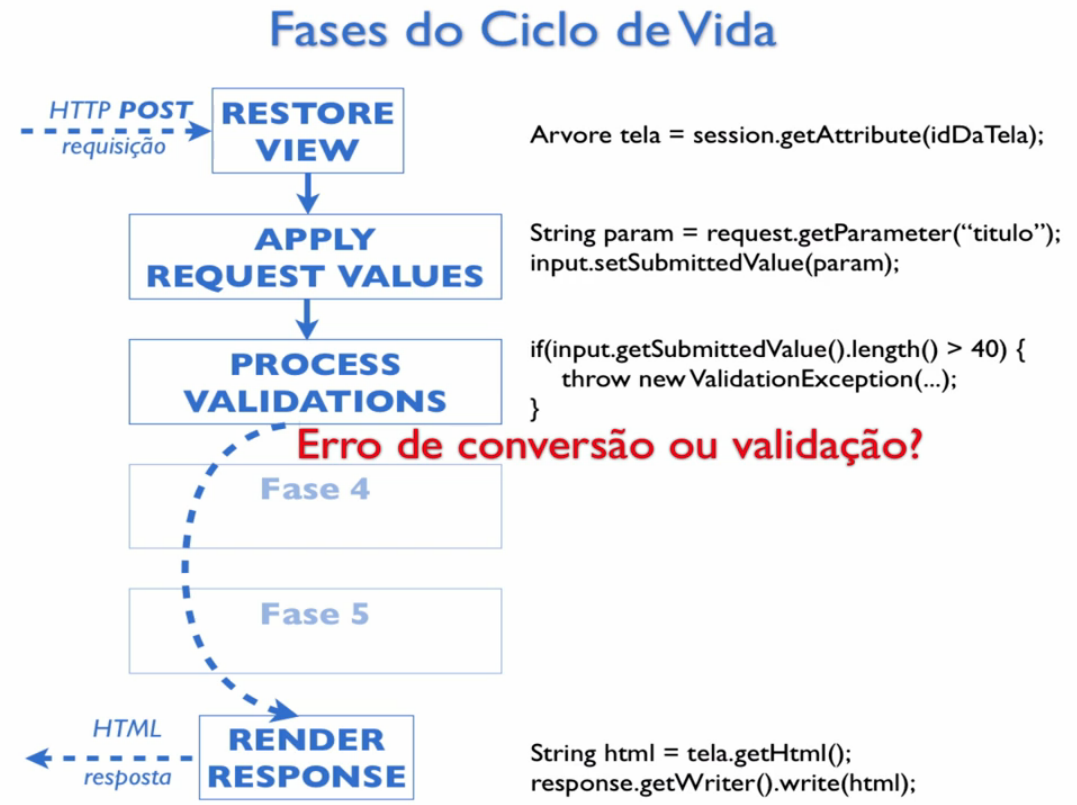
**Analisando as fases: APPLY\_REQUEST\_VALUES e PROCESS\_VALIDATION**

No Eclipse novamente limparemos a console e mais uma vez no navegador usaremos o formulário para executar uma validação. Lembrando que o campo do título automaticamente envia uma requisição AJAX ao perder foco e consequentemente é executada a validação no lado do servidor.

Olhando o console no Eclipse podemos ver que dessa vez foram executadas 4 fases. Temos como segunda fase o APPLY\_REQUEST\_VALUES, e como terceira fase PROCESS\_VALIDATION.

Vamos analisar também esse caso. Como sempre o controlador recebeu a requisição mas agora ela é do tipo POST. Isso significa que o controlador apenas recupera a árvore (por isso se chama RESTORE\_VIEW). Após a recuperação da tela os componentes recebem o valor digitado pelo usuário que vem da requisição (dai vem o nome APPLY\_REQUEST\_VALUE). Nesse caso submetemos apenas o valor do titular, mas em branco. Em outras palavras, o componente do titular recebe um String vazio.

Na terceira fase acontece a conversão, se for preciso, e a validação. O titulo não precisa ser convertido pois o titulo é do tipo String, mas associamos uma validação com o componente. É justamente essa validação que falha. Em geral, se há um problema de conversão ou validação, o controlador pula automaticamente as fases quatro e cinco e pode renderizar o HTML com as mensagens de erro.



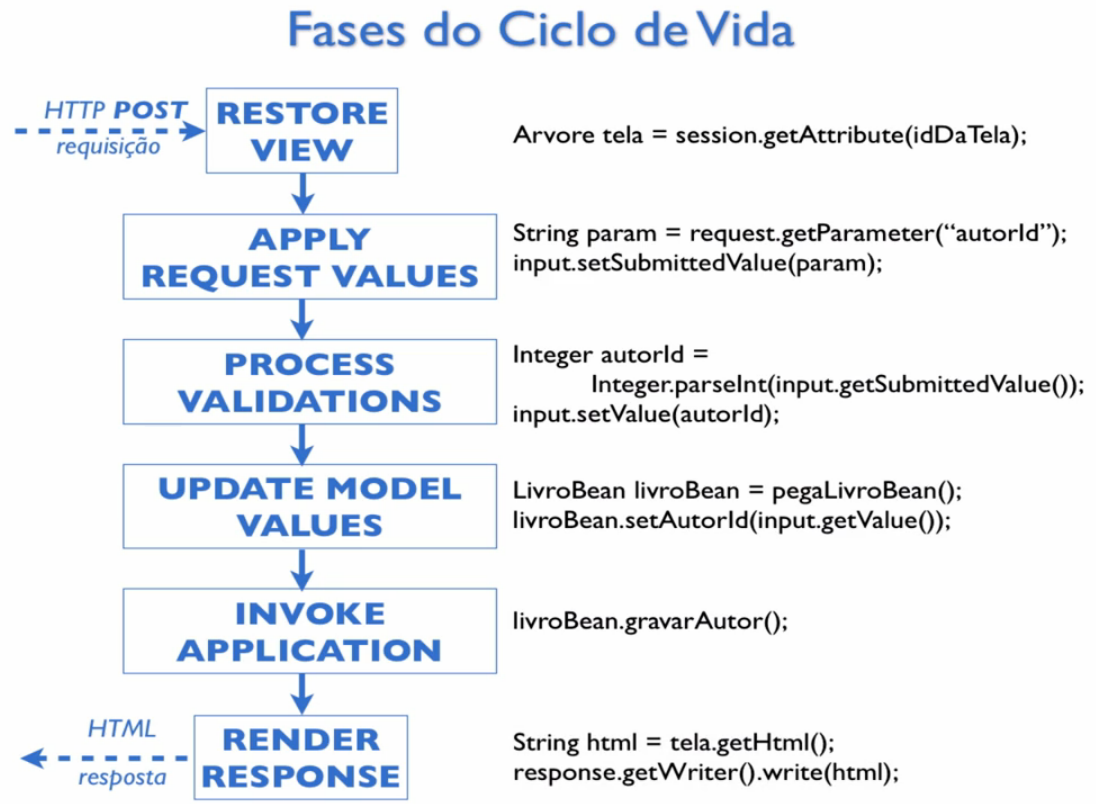
**Todas as fases do ciclo**

Agora testaremos o que acontece quando escolhemos um autor no combobox. Vamos apertar o botão e verificar a console de novo. Repare que agora foram executadas todas as seis fases. Na primeira fase (RESTORE\_VIEW) foi recuperada a árvore de componentes da sessão, na segunda (APPLY\_REQUEST\_VALUES) os componentes receberam os parâmetros da requisição e na terceira (PROCESS\_VALIDATION) todos os dados foram convertidos e validados.

Como nenhum erro ocorreu na terceira fase o controlador continua com a próxima que se chama UPDATE\_VALUES. Aqui o modelo será atualizado com os valores convertidos na fase anterior. Neste caso, LivroBean recebe apenas ID do autor já que só enviamos esse parâmetro pelo AJAX. Em geral, nesta fase, tudo que definimos com a *expression language*nos componentes de inputs é atualizado no modelo.

Agora que o modelo já foi atualizado, nosso LivroBean já pode executar o comando definido via expression language. Este comando é um método no próprio LivroBean e opera sobre os dados convertidos. Esta fase de chamada de métodos no ManagedBean é chamada de INVOKE\_APPLICATION.

Por fim, o JSF devolve uma resposta para o usuário, o que é feito na última fase, RENDER\_RESPONSE.



**Resolvendo o problema de navegação e o atributo immediate**

Abriremos a página e testaremos novamente o link. Ao voltarmos ao Eclipse, vemos no console as fases que foram processadas. Houve um salto da fase PROCESS\_VALIDATION para RENDER\_RESPONSE, sendo assim, nenhum dado do modelo foi atualizado e nenhum método foi invocado, isto porque as fases UPDATE\_MODEL e INVOKE\_APPLICATION foram puladas.

Vamos realizar outro tipo de navegação, mas desta vez através de um método em LivroBean. Vamos alterar o atributo action que agora apontará para o método formAutor(). Fazemos isso através de expression language #{livroBean.formAutor}. Com a classe aberta, adicionaremos o método, mas agora precisamos que seu retorno seja uma String, pois o retorno é a regra de navegação que o JSF seguira. Aqui vamos devolver apenas o nome da página, sem extensão. Adicionaremos também uma saída no console que será impressa apenas se o JSF executar a fase PROCESS\_VALIDATION:

@ManagedBean

@ViewScoped

public class LivroBean implements Serializable {

//codigo omitido

public String formAutor() {

System.out.println("Chamando o formulário do Autor");

return "autor";

}

//codigo omitido

}

Vamos reiniciar o servidor, pois alteramos uma classe. Abrindo mais uma vez e testando o link, vemos que não mudou o fluxo. Olhando no console do Eclipse, percebemos que não foi invocado a fase INVOKE\_APPLICATION, isto e, o JSF não executou o método formAutor().

O problema é que o nosso link com a regra de navegação está disparando a validação do formulário, quando na verdade queremos apenas seguir para outra página. Uma maneira de resolver este problema e adicionar o atributo immediate=true em nosso link. Isso fará com que o método formAutor() seja executado durante a segunda fase APPLY\_REQUEST\_VALUES, antes da validação. Assim a nossa regra de navegação será processada e automaticamente pularemos para última fase.

<h:commandLink value="Cadastrar novo autor" action="#{livroBean.formAutor}" immediate="true"/>

Finalmente a página do autor foi chamada. Podemos verificar também na saída do console. Foram executadas a primeira, segunda e sexta fase.

