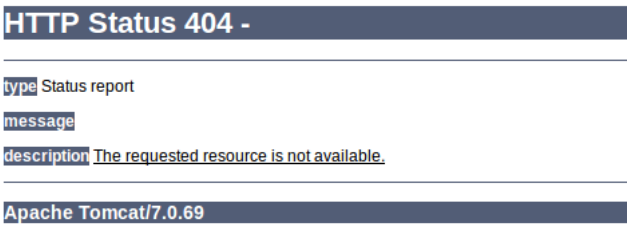
06

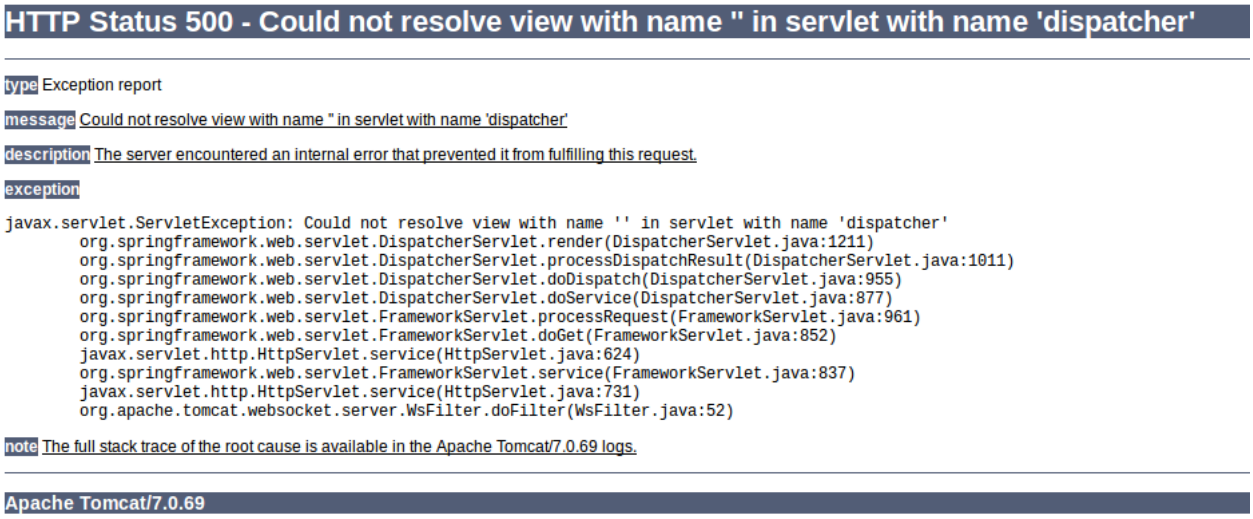
Completando Controller

Transcrição

Se iniciarmos nosso servidor agora, devemos ter alguma mudança visível em nossa aplicação! Acessando-a pelo navegador (em `localhost:8080/casadocodigo`) temos:



Erro 404! Porque? O que aconteceu? Nossas configurações estão todas certas e aparentemente o projeto não funcionou! Acontece que o SpringMVC está com um conflito de recursos, nós mapeamos a rota "/", mas o servidor por padrão, quando recebe uma requisição para o "/" ele retorna o `index` que estiver naquele endereço!Para resolvermos isso, basta deletar o arquivo `index.html`. Atualizando a página teremos:

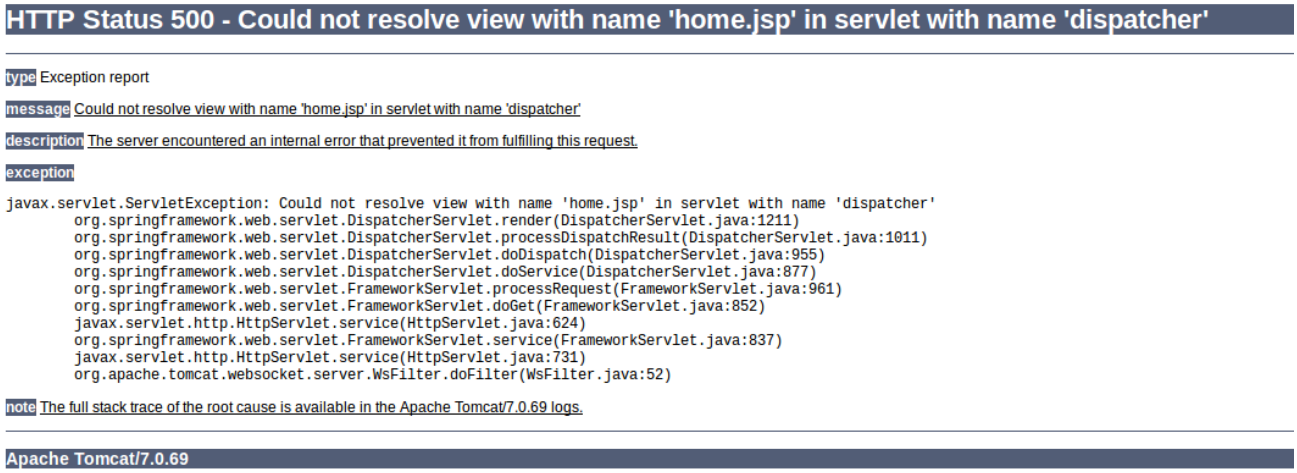


Erro 500! O que aconteceu agora? veja a mensagem de erro: `javax.servlet.ServletException: Could not resolve view with name '' in servlet with name 'dispatcher'`. O SpringMVC está procurando uma `view` de nome vazio? Note que no log do Eclipse que nossa mensagem no `System.out.println` do método `index` foi impressa! Isso indica que o SpringMVC chamou nosso metodo `index`. Mesmo assim ele continua com o erro dizendo que não encontrou uma *view*.

Vamos modificar nosso controler para retornar a `view` que o SpringMVC espera receber. No nosso `HomeController` modificamos o metodo `index` para ficar da seguinte forma:

```
@RequestMapping("/")
public String index(){
    System.out.println("Entrando na home da CDC");
    return "home.jsp";
}
```

Agora estamos retornando uma String que representa o nome da *view*. Se atualizarmos no navegador a página do nosso projeto, podemos visualizar que o erro mudou! Ainda é um **erro 500** mas desta vez o SpringMVC identifica a `view` que foi retornada pelo método: `home.jsp` .



Apesar de termos retornado uma view, não temos esta *view* criada ainda! Crie então o `home.jsp` dentro da pasta `webapp` do seu projeto (Use os atalhos do Eclipse). Seu `home.jsp` deve ficar parecido com este:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loos
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Livros de Java, Android, iPhone, Ruby, PHP e muito mais - Casa do Código</title>
</head>
<body>
    <h1>Casa do Código</h1>
</body>
</html>
```

Apenas este passo não resolve nosso problema, pois o SpringMVC ainda não sabe onde encontrar as *views* do nosso projeto. Antes de fazermos esta configuração, vamos fazer algumas modificações por questões de boas práticas.

1. Crie uma pasta dentro de `webapp` chamada `WEB-INF`.
2. Crie uma pasta dentro de `WEB-INF` chamada `views`.
3. Mova o arquivo `home.jsp` para dentro da pasta `views`.

A pasta `WEB-INF` é uma pasta protegida pelo servidor. Este que não permite que os arquivos dentro dela sejam acessados diretamente. É uma boa prática que deixemos nossas *views* dentro desta pasta para que o usuário não consiga acessar as páginas de forma direta, sem que ele passe pelo controller, pois se a *view* for acessada sem passar pelo *controller*, podemos ter quebra de regras negócio para aquela *view*.

Nosso próximo passo é configurar o projeto para que o SpringMVC consiga encontrar as *views*. Essa configuração é feita na classe de configuração `AppWebConfiguration` . Nesta criaremos um novo metodo que ajudará o SpringMVC a encontrar nossas *views*:

```
@Bean
public InternalResourceViewResolver internalResourceViewResolver(){
    InternalResourceViewResolver resolver = new InternalResourceViewResolver();
    resolver.setPrefix("/WEB-INF/views/");
    resolver.setSuffix(".jsp");
    return resolver;
}
```

Este método na classe `AppWebConfiguration` retorna um objeto do tipo **InternalResourceViewResolver** (Resolvedor Interno de Recursos de View) que ajuda o SpringMVC a encontrar as *views*. O `setPrefix` define o caminho onde estão nossas *views*, já o `setSuffix` adiciona a extensão dos arquivos de *view*.

Note que no final do caminho das *views* há uma barra "/", ela poderia não estar ai, mas caso não estivesse, teriamos que lembrar que sempre que quisessemos retornar uma *view*, teriamos que no *controller*, no retorno da *view* escrever algo como: `return "/pagina.jsp"` , então colocamos esta barra já na configuração para não termos que nos preocupar com isto.

A anotação `@Bean` é para que o retorno da chamada deste metódo possa ser gerenciada pelo SpringMVC, sem ela nossa configuração não funciona. Outra observação válida é que já estamos colocando o sufixo dos arquivos (.jsp). Desta forma, não precisamos colocar a extenção dos arquivos de *view* nos controllers. Sendo assim, modifique o `HomeController` para retornar apenas `"home"` em vez de `"home.jsp"`

Com essas configurações feitas, podemos agora visualizar a página inicial do projeto sem mais problemas. Experimente pôr mais alguns elementos na `home.jsp` , como por exemplo uma tabela que pode listar alguns livros da casa do código! Algo como:

```
[...]
<h1>Casa do Código</h1>
<table>
    <tr>
        <td>TDD na Prática - JAVA</td>
        <td>Google Android</td>
     </tr>
    <tr>
        <td>Certificação OCJP</td>
        <td>Java 8 Prático</td>
    </tr>
</table>
[...]
```