**Spring MVC: Requisitos e Fluxo do Processamento da Informação**

O Spring MVC possui uma divisão bem estruturada de suas camadas, tornando o código de uma forma bem organizada.

O framework é compatível com os principais servidores web Java, como o Apache Tomcat, Jboss, BEA Weblogic ou IBM Websphere. Além disso, possui a integração com frameworks para mapeamento do banco de dados como o Hibernate.

Um dos jeitos mais fáceis de se desenvolver um projeto com o spring mvc é utilizando o Spring Tool Suite que pode ser encontrado no site oficial do framework: <https://spring.io/tools>. Essa ferramenta é um eclipse com plugins de desenvolvimento do spring já instalados e configurados, ou seja, nela podemos criar projetos do tipo spring project e no assistente de criação, podemos especificar que queremos que o projeto seja do tipo Spring MVC. Com isso não precisamos nos preocupar em baixar as bibliotecas manualmente ou saber quais são as dependências do maven que precisam ser adicionadas.

Mas se vocês quiserem utilizar em um projeto maven do eclipse, ele pode ser baixado adicionando-se a seguinte dependência dentro do pom.xml:

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>4.1.4.RELEASE</version>

</dependency>

O fluxo do processo das informações do Spring MVC, segue uma sequência de eventos quando uma requisição é enviada ao framework.

1 - Primeiramente o *DispatcherServlet,* recebe a requisição

O *DispatcherServlet* é um dos principais componentes da estruturação do Spring MVC, pois além de ser um mapeador de requisições, representando um único canal de entrada para todas requisições direcionadas, facilitando o gerenciamento da informação, é responsável por encaminhar para qual *Controller* vai receber e processar a requisição, além de apontar o arquivo de *template* específico no qual será renderizado na camada *View*.

2 - O DispatcherServlet verifica o HandlerMapping e carrega o Controller associado a requisição.

HandlerMapping é uma interface que faz a análise e define um mapeamento da requisição.

3 - O Controller processa a requisição através da chamada aos métodos apropriados do serviço e retorna um objeto ModeAndView para DispatcherServlet.

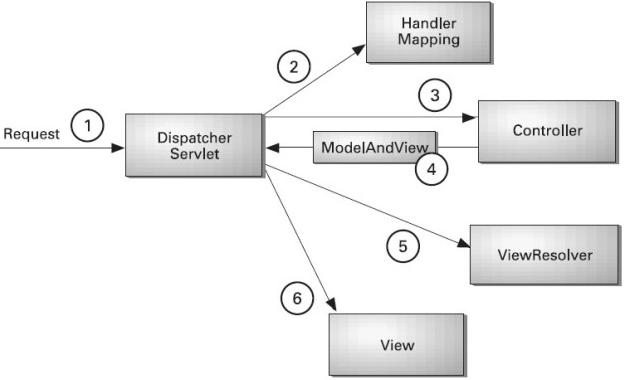
O objeto ModeAndView contém os dados do modelo e o nome da visão.

4 - O DispatcherServlet envia o nome de visão para um ViewResolver para que ele encontre o View que deve ser carregado.

O View Resolver é um gerenciador de visualização, ele procura a página JSP no qual corresponde ao nome da view encaminhada pelo Controller.

5 - Agora o DispatcherServlet passará o objeto modelo para o View para que o resultado seja renderizado.

6 - A View com os dados vindo do modelo, vai renderizar o resultado para o usuário.

****

**Spring MVC: Controllers**

O Controller é uma classe Java que possui os métodos responsáveis por tratar as requisições, e como o próprio nome já diz, faz parte da camada controladora do modelo MVC. Responsável por intermediar as informações junto a View, receber parâmetros e disponibilizar resultados.

O *Controller* suporta os métodos do tipo GET ou POST usado. O método de serviço definirá os dados do modelo com base na lógica de negócios definida e retornará o nome da visualização ao *DispatcherServlet*   
  
Para dizer que a classe será um Controller, basta fazer a anotação **@*Controller*,** assim a classe já passa a servir com a função de um controlador.   
A anotação **@*RequestMapping*** é usada para mapear uma URL para uma classe inteira ou um método de manipulador específico. E para fazer isso, basta passar os atributos como parâmetro de função dentro da anotação.  
O atributo ***value*** indica a URL para a qual o método do manipulador é mapeado e o atributo ***method*** define o método de serviço para manipular a solicitação.

Veja um exemplo simples:

/\*\* Incluir código Spring MVC – C5 \*/

**Spring MVC: Views**

O Spring MVC se destaca por sua separação das tecnologias de visualização. Suporta muitos tipos de visualizações para diferentes tecnologias de apresentação. Estes incluem - JSPs, HTML, PDF, planilhas do Excel, XML, modelos Velocity, XSLT, JSON, Atom e RSS feeds, JasperReports, etc. Mas, mais comumente, são utilizados modelos JSP escritos com JSTL. Veja um exemplo:

//\*\* incluir código Spring MVC – V1 \*/

Aqui **$ {message}** é o atributo que configuramos dentro do Controller. Você pode ter vários atributos para serem exibidos dentro de sua visão.

**Spring MVC: Injeção Dependências**

O Spring MVC foi desenvolvido ao conceito da utilização de Injeção de Dependência, um padrão onde a classe não precisa se preocupar em como conseguir suas dependências, apenas em trabalhar com elas.  
Com isso, ajuda ao desacoplamento do código, tornando mais fácil ao gerenciamento e a realização de testes no sistema.  
E para que o framework identifique os pontos no qual será injetada, é necessário fazer uma anotação na classe com a expressão @Autowired.  
A anotação pode ser utilizada nos em 3 casos:  
- Nas Propriedades;  
- Nos Construtores;   
- Nos Métodos (normalmente, os setters)  
  
Outro requisito para que uma instância possa ser injetada, é transforma-lo em uma Bean Spring.   
No qual é necessário fazer a anotação @Component ou com qualquer uma de suas especializações:  
  
@Component: Esta anotação faz com que o bean registrado no Spring possa ser utilizado em qualquer bean, seja ele um serviço, um DAO, um controller, etc.  
@Service: Anotação no qual diz que a bean faz parte da camada de serviço.   
@Repository: Anotação no qual diz que a bean faz parte da camada de persistência.

//\*\* incluir código Spring MVC – Di1 \*/

**Spring MVC: Interceptadores**

E para entender o interceptador, vamos dar um passo para trás e ver o HandlerMapping . Isso mapeia um método para uma URL, para que o DispatcherServlet possa invocá-lo ao processar uma solicitação.

E o DispatcherServlet usa o HandlerAdapter para invocar o método.

Agora que entendemos o contexto geral, **é aqui que entra o interceptador**. Usaremos o  HandlerInterceptor para executar ações antes do manuseio, após o manuseio ou após a conclusão (quando a exibição for renderizada) de uma solicitação.

O interceptor pode ser usado para interesses transversais e para evitar códigos manipuladores repetitivos, como: registro em log, alteração de parâmetros usados ​​globalmente no modelo Spring etc.

Os interceptores que trabalham com o HandlerMapping na estrutura devem implementar a interface HandlerInterceptor.

Esta interface contém três métodos principais:

* prehandle () - chamado antes do manipulador real ser executado, mas a visualização ainda não foi gerada
* postHandle () - chamado depois que o manipulador é executado
* afterCompletion () - chamado depois que a solicitação completa foi concluída e a visualização foi gerada

Esses três métodos fornecem flexibilidade para todos os tipos de pré e pós-processamento.

E uma nota rápida - a principal diferença entre HandlerInterceptor e HandlerInterceptorAdapter é que, no primeiro, precisamos substituir todos os três métodos: preHandle () , postHandle () e afterCompletion () , enquanto no segundo, podemos implementar apenas os métodos necessários.

/\*\* Inserir código Spring MVC Interceptor – I2 \*/

**Spring MVC: Validadores**

A validação da entrada recebida do usuário para manter a integridade dos dados é uma parte importante da lógica do aplicativo. A validação de dados pode ocorrer em diferentes camadas.

O Spring MVC no qual é baseado na plataforma JAVA EE 6, aproveita um dos recursos disponíveis para fazer a validação. O modelo Bean Validation, no qual é suportado por restrições na forma de anotações colocadas em um campo, método ou classe de um componente JavaBeans, como um bean gerenciado e pode ser utilizado em qualquer camada da aplicação.

Com o Bean Validation declaramos através de anotações as regras de validação dentro do nosso modelo:

/\*\* Inserir código Spring MVC Validator – V1 \*/

Com a anotação do Bean Validation na camada de controller. É necessário avisar o Spring MVC que queremos executar a validação. Isso é feito pela anotação Valid que devemos usar na antes do parâmetro da ação:

/\*\* Inserir código Spring MVC Validator – V2 \*/

O Spring MVC pode armazenar o resultado dos erros de validação em um objeto do tipo BindingResult. Este objeto BindingResult se torna um parâmetro da ação. No qual pode ser utilizado para que em vez de ser lançado uma exceção, seja redirecionado para outra página.

/\*\* Inserir código Spring MVC Validator – V3 \*/

/\*\* Inserir código Spring MVC Validator – V4 \*/

Para a exibição das mensagens de validação na camada da View, é utilizado uma tag especial que o Spring MVC oferece. A tag se chama **form:errors**:

<form:errors path="tarefa.descricao" />

O atributo path indica com que atributo essa mensagem está relacionada.

/\*\* Inserir código Spring MVC Validator – V5 \*/

**Spring MVC: Plugins**

O Spring MVC possui diversos módulos disponíveis para acoplamento no seu projeto, para facilitar ainda mais do desenvolvimento para serviços específicos.

• Spring Security útil para inclusão de funcionalidades de autenticação e autorização.

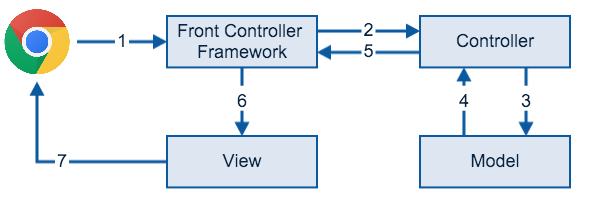
• Spring Data para aplicações que usam novas tecnologias de armazenamento de dados e serviços na nuvem.

• Spring Social para fácil integração com redes sociais.

• Além de outros. Disponível no site oficial do framework <http://spring.io/projects>

**Modelo MVC**

Segue abaixo a imagem de uma representação do fluxo do processo de uma requisição do ponto de vista do padrão MVC

[](https://s3.amazonaws.com/algaworks-blog/wp-content/uploads/Fluxo-do-Spring-MVC.png)

Os detalhes dos passos citado na imagem acima, segue da sequinte forma:  
  
1. Fazemos uma requisição HTTP através de uma URL no navegador, no qual é enviado para um servidor web com Spring MVC.  
2. O framework então através do seu *controller* (controlador), faz a pesquisa para localizar qual a classe é responsável por tratar a requisição.  
3. O *controller* encaminha os dados para o *model (modelo),* seguindoas boas práticas, camada responsável por executar as regras de negócios, por exemplo: validações, cálculos e acesso ao banco de dados.  
4. Após realizar as operações, o *model* retorna o resultado ao *controller.*  
5. O controller, então devolve o endereçamento da *view* (visão), junto aos dados que serão renderizados junto a página.  
6. O framework localiza a view que será processado os dados, renderizando o resultado em uma página web.   
7. Finalizando, a página web é retornando ao navegador do usuário.