educación IT
CODE YOUR FUTURE

Bootcamp Java Developer

Fase 3 - Java Architect

Módulo 29

educación IT
CODE YOUR FUTURE

Spring Framework

Spring MVC

Configuración

Spring tiene sus propios módulos que conforman un framework, para el desarrollo de aplicaciones Java basadas en web, que sigue el patrón de diseño MVC. A este conjunto de módulos se le conoce con el nombre de *Spring MVC*.

Al igual que Struts 2, **Spring MVC es un framework de capa de presentación, basado o dirigido por peticiones**. Esto quiere decir que los programadores estructuran el diseño de una aplicación web en términos de métodos que manejan peticiones HTTP.

El framework define una serie de interfaces que siguen el patrón de diseño Strategy para todas las responsabilidades que debe manejar. El objetivo de cada interfaz es ser simple y clara, para que a los usuarios de Spring MVC les sea fácil crear implementaciones propias. La parte central de Spring MVC es un componente llamado el **`DispatcherServlet`**, que sigue el patrón de diseño *front controller*, este envía las peticiones a los componentes designados para manejarlas.

Las interfaces más importantes definidas por Spring MVC son:

- **`Controller`:** se encuentra entre el Modelo y la Vista; maneja las peticiones que entran y redirige a las respuestas apropiadas.
- **`HandlerAdapter`:** realiza la invocación del manejador de la petición (el denominado `DispatcherServlet` delega esa tarea a este componente), esto incluye inyectar los parámetros adecuados usando reflexión.
- **`HandlerInterceptor`:** Intercepta de entrada las peticiones. Es similar, pero no igual, a los filtros en la API de Servlets.
- **`HandlerMapping`:** Selecciona objetos que manejan las peticiones de entrada basadas en cualquier atributo o condición interna o externa a esos atributos.
- **`LocaleResolver`:** Resuelve y opcionalmente guarda el Locale de un usuario individual.
- **`MultipartResolver`:** Facilita trabajar con la carga de archivos, envolviendo las peticiones de entrada.
- **`View`:** Responsable de regresar una respuesta al cliente. Algunas peticiones pueden ir directo a la vista sin pasar por el modelo.
- **`ViewResolver`:** Selecciona una vista basado en un nombre lógico para la vista.

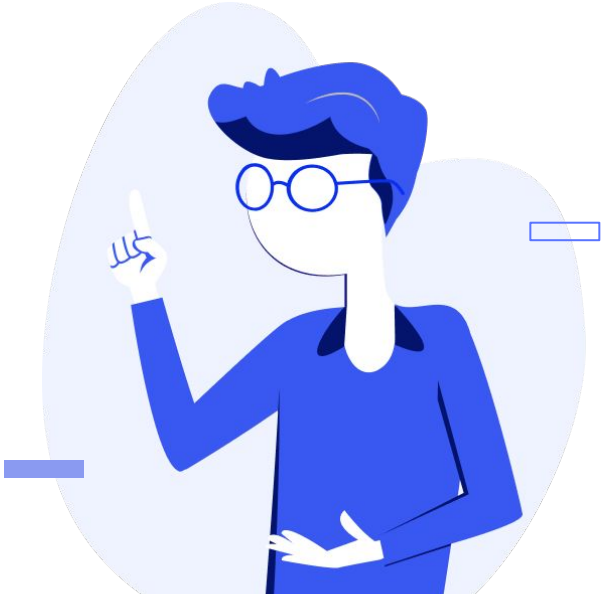
Cada una de las interfaces anteriores tiene una responsabilidad importante dentro del framework y la abstracción que ofrecen permite tener variaciones en sus implementaciones. Spring MVC contiene implementaciones de casi todas estas interfaces que están construidas sobre el API de Servlets.

Principio de diseño "Abierto / Cerrado"

Un principio clave en Spring MVC es el principio de diseño *"Abierto / Cerrado"* o, por su nombre en inglés, ***Open for extension, closed for modification***. Algunos métodos en las clases *core* de Spring MVC están marcados como final para que los desarrolladores no puedan sobreescribirlos y proporcionar un nuevo comportamiento, de esta forma el framework garantiza que **su comportamiento básico no puede ser modificado**.

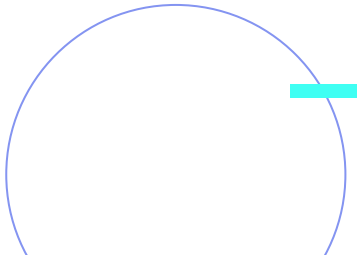
DispatcherServlet

El `DispatcherServlet` es la pieza central de Spring MVC. Spring MVC como muchos otros frameworks orientado a **peticiones**, está diseñado alrededor de un Servlet central que despacha las peticiones a los controladores y ofrece otra funcionalidad que facilita el desarrollo de aplicaciones web. Sin embargo, el `DispatcherServlet` hace más que sólo eso, se encuentra **completamente integrado con el contenedor de IoC de Spring**, lo que permite usar el resto de características de Spring.

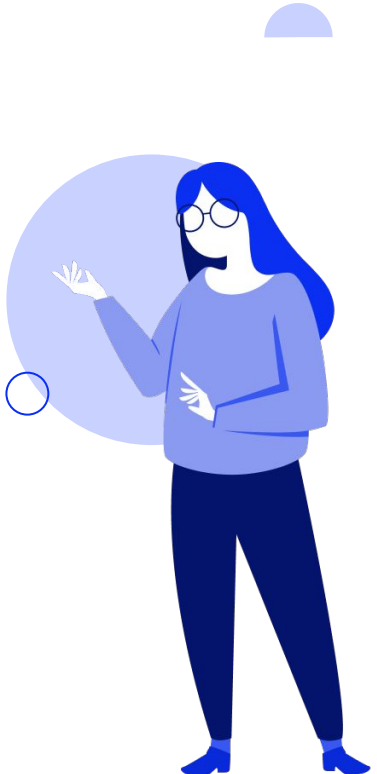


Flujo de navegación del `DispatcherServlet`:

1. El cliente hace una petición a la aplicación web, esta petición llega al denominado `DispatcherServlet`.
2. El `DispatcherServlet` determina qué componente debe atender la petición y la envía a este.
3. El `Controller` implementa la lógica específica para responder la petición, para lo que puede hacer uso de cualquier recurso que esté al alcance de cualquier aplicación Java (incluyendo conexiones con servicios web o con base de datos).
4. Una vez que el `Controller` termina su proceso, regresa la petición al denominado `DispatcherServlet`, estableciendo los datos adecuados del modelo e indicando el nombre lógico de la vista que debe regresarse al cliente y un modelo lleno con los nombres y valores de los atributos que se usarán para generar la vista final.



5. En base al nombre lógico regresado por el `Controller`, el `DispatcherServlet` usa un `ViewResolver` para determinar qué recurso debe utilizar para generar la vista final que se mostrará al usuario, este recurso puede ser una JSP, una página HTML, un template de Velocity, un archivo de Excel, un PDF, y otros.
6. El `DispatcherServlet` obtiene la vista que será regresada al cliente.
7. El `DispatcherServlet` finalmente regresa la vista adecuada al cliente.



Componente ViewResolver

A partir del paso 5 se indica **qué componente y de qué forma se resolverá la vista** (en este caso una página) debe regresarse en respuesta a la petición que haya hecho el usuario. Para esto Spring tiene un componente cuyo nombre es **`ViewResolver`** que se encarga de, en base a un nombre lógico, **identificar el recurso (o página) al que se está haciendo referencia**.

La resolución de vistas en Spring es muy flexible. Un `Controller` es típicamente responsable de preparar el modelo con datos y seleccionar el nombre de la vista que será regresada al usuario, sin embargo, también podría escribir la respuesta directamente en el *stream* de respuesta. **La resolución del nombre de la vista es altamente configurable** y se puede integrar con representación basada en tecnologías de templates como JSP, Velocity y Freemarker, o generar directamente XML, JSON, Atom, y muchos otros tipos de contenido.

Spring MVC tiene varias implementaciones de `ViewResolver`, como las que se muestran en la próxima pantalla.

ViewResolver	Descripción
`AbstractCachingViewResolver`	Es un `ViewResolver` abstracto que coloca vistas en caché. Algunas veces las vistas necesitan cierta preparación antes de que puedan ser usadas; extendiendo esta clase proporciona ese caché.
`XmlViewResolver`	Implementación de `ViewResolver` que acepta un archivo de configuración en XML con los mismos DTDs que los bean factories de Spring. El archivo de configuración por default es `"/WEB-INF/views.xml"`.
`ResourceBundle ViewResolver`	Implementación de `ViewResolver` que usa definiciones de beans en un `ResourceBundle` especificado por el nombre base del bundle. Típicamente definimos el bundle en un archivo de propiedades localizado en el `classpath`. El nombre por default es `"views.properties"`.
`UrlBasedViewResolver`	Implementación simple de `ViewResolver` que realiza la resolución directa de nombres de vistas lógicas a URLs, sin una definición explícita de mapeo. Es apropiado si nuestros nombres lógicos coinciden con los nombres de los recursos de vista de forma directa, sin la necesidad de mapeos adicionales.

ViewResolver	Descripción
`InternalResourceViewResolver`	Subclase de `UrlBasedViewResolver` que soporta vistas basadas en JSPs. Tiene otras subclases útiles como `JstlView` y `TilesView`.
`VelocityViewResolver / FreeMrkerViewResolver`	Subclase de `UrlBasedViewResolver` soporta vistas creadas usando templates creados usando Velocity o Freemarker, respectivamente y subclases propias de estos.
`ContentNegotiatingViewResolver`	Implementación de `ViewResolver` que resuelve una vista basada en la petición para determinar el nombre del archivo de la vista o en la cabecera `"Accept"`.

¡Sigamos trabajando!