Spring MVC y MVC

El Model-View-Controller (MVC) es un patrón de diseño de software que separa la representación de la información de la interacción del usuario con ella. Esta separación hace que el diseño sea más modular, lo que permite una mayor reutilización del código. El patrón MVC se usa ampliamente en la programación actual, especialmente en aplicaciones web.

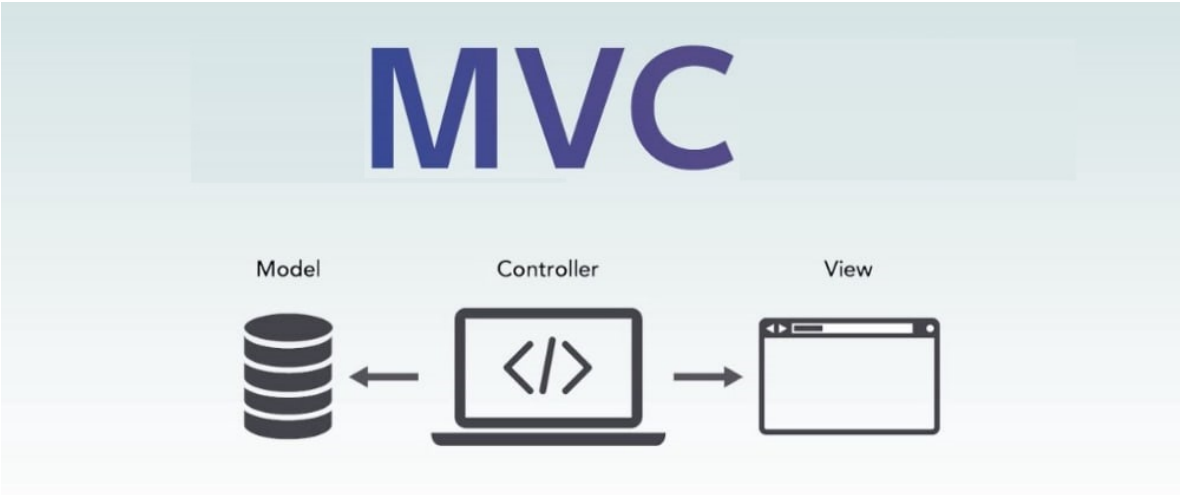
El patrón MVC se compone de tres partes: el modelo, la vista y el controlador.
El modelo contiene los datos, la vista muestra los datos y el controlador maneja la entrada del usuario.

El patrón MVC tiene varios beneficios:

- Permite al desarrollador cambiar más fácilmente los datos sin afectar la vista.
- Permite al desarrollador cambiar más fácilmente la vista sin afectar los datos.
- Permite al desarrollador cambiar más fácilmente el controlador sin afectar los datos o la vista.

El patrón MVC no está exento de inconvenientes:

- Puede conducir a la duplicación de código si el modelo, la vista y el controlador no están bien diseñados.
- Puede ser difícil realizar pruebas unitarias de las partes individuales del patrón MVC.



Spring Boot es un marco que facilita la creación de aplicaciones basadas en Spring independientes y de grado de producción. Toma una visión obstinada de la plataforma Spring y se aparta del camino del desarrollador, lo que facilita la creación de aplicaciones listas para producción.

Spring Boot incluye una serie de funciones integradas, incluida la capacidad de integrar Tomcat, Jetty o Undertow directamente en la aplicación. Esto facilita la creación de un jar ejecutable autónomo que se puede implementar en un servidor con un simple comando java -jar.

El patrón Modelo-Vista-Controlador (MVC) es un patrón de diseño popular para crear aplicaciones web. Spring Boot facilita la creación de aplicaciones basadas en Spring independientes y de grado de producción que puede simplemente ejecutar.

¿Qué diferencias existen entre implementar MVC con Java puro y con Spring?

La implementación del patrón MVC (Modelo-Vista-Controlador) puede variar significativamente dependiendo de si estás utilizando Java puro o el framework Spring Boot.

Implementación con Java Puro:

1. **Modelo:** En Java puro, el modelo puede ser representado por clases Java que contienen la lógica de negocio y los datos. No hay un marco específico para definir modelos, por lo que los desarrolladores tienen la libertad de diseñar sus propias clases de modelo.

2. **Vista:** Las vistas en Java puro generalmente se implementan utilizando tecnologías de servlets y JSP (JavaServer Pages). Los desarrolladores escriben el código HTML y utilizan etiquetas JSP para insertar dinámicamente datos provenientes del modelo.

3. **Controlador:** Los controladores en Java puro son servlets que manejan las solicitudes HTTP y actúan como intermediarios entre las vistas y el modelo. Los desarrolladores deben implementar la lógica de control dentro de los servlets.

Implementación con Spring:

1. **Modelo:** En Spring, el modelo sigue siendo representado por clases Java que contienen la lógica de negocio y los datos, pero Spring proporciona un soporte más robusto para la creación y gestión de modelos a través de anotaciones como ‘@Entity’ para entidades JPA y ‘@Service’ para servicios.

2. **Vista:** Spring utiliza plantillas como Thymeleaf o FreeMarker para las vistas en lugar de JSP. Estas plantillas permiten una integración más fácil con el modelo y proporcionan características adicionales como la manipulación de datos en el lado del cliente.

3. **Controlador:** En Spring, los controladores se implementan como componentes gestionados por Spring, utilizando anotaciones como '@Controller' para manejar las solicitudes HTTP. Spring proporciona un enfoque más declarativo y basado en anotaciones para definir la lógica de control.

Diferencias Clave:

1. **Configuración y Conveniencia:** Spring simplifica la configuración y la integración de diferentes componentes de MVC mediante el uso de convenciones y autoconfiguración, lo que reduce la cantidad de código boilerplate que los desarrolladores tienen que escribir en comparación con Java puro.

2. **Gestión de Dependencias:** Spring maneja automáticamente la gestión de dependencias y la inyección de dependencias, lo que facilita el desarrollo y la mantenibilidad de las aplicaciones.

3. **Escalabilidad y Flexibilidad:** Spring ofrece una arquitectura modular y escalable que facilita la ampliación de funcionalidades y la integración con otras tecnologías.