Material descargable

egg

SPRING FRAMEWORK

¿QUÉ ES SPRING FRAMEWORK?

Definición de framework

Antes de abordar Spring, es importante definir qué es un **framework**. Un framework es un entorno de trabajo diseñado para facilitar el desarrollo de software. Proporciona herramientas, bibliotecas y una estructura predefinida que agiliza el proceso de programación, reduce errores, fomenta el trabajo colaborativo y mejora la calidad del producto final.

Los frameworks ofrecen una arquitectura base y pueden ser independientes del lenguaje de programación. Sin embargo, es común encontrar frameworks especializados para lenguajes específicos.

SPRING FRAMEWORK

Spring es un **framework alternativo al stack de tecnologías estándar de JavaEE**. Su popularidad radica en la introducción de conceptos como la **inyección de dependencias** y el uso de **POJOs** (*Plain Old Java Objects*) como objetos de negocio.

Este framework se ha convertido en la opción preferida para el desarrollo de aplicaciones empresariales en Java, ya que permite escribir código de **alto rendimiento, liviano y reutilizable**. Su propósito es **estandarizar, agilizar y optimizar** el desarrollo de software, resolviendo problemas comunes en la programación.

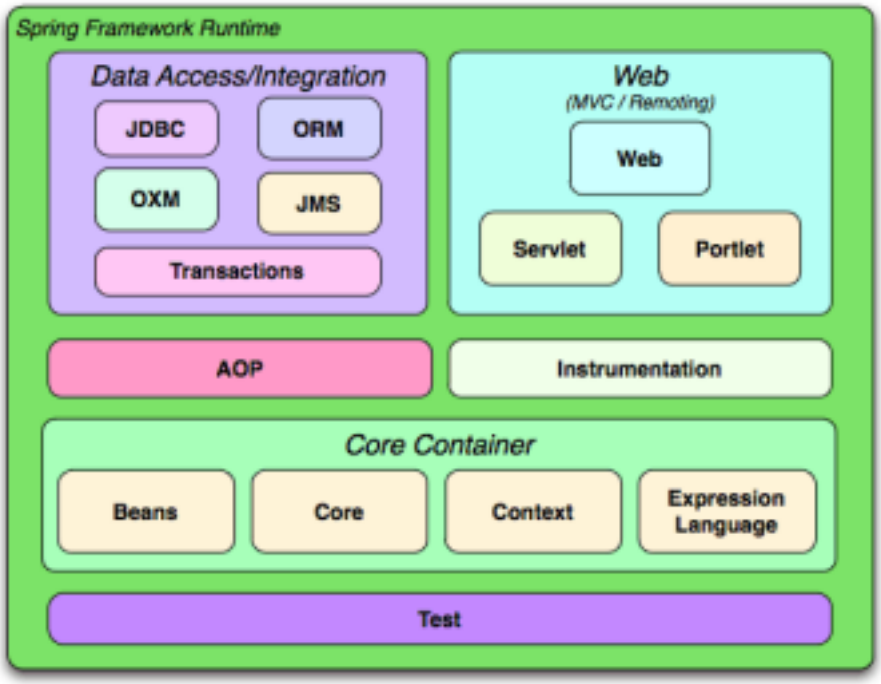
Principales funcionalidades de Spring

Spring ofrece una serie de módulos, siendo el más destacado el **mecanismo de**

inyección de dependencias. Sin embargo, cuenta con otras funcionalidades clave:

- **Core Container**: proporciona soporte para **inyección de dependencias** e **inversión de control**.
- **Web**: permite desarrollar aplicaciones web, tanto con vistas **MVC** como **RESTful APIs**.
- **Acceso a datos**: proporciona abstracciones para **JDBC**, compatibilidad con **ORMs** como Hibernate, sistemas **OXM (Object-XML Mappers)**, **JMS** y manejo de transacciones.
- **Instrumentación**: facilita la instrumentación y el monitoreo de clases.
- **Pruebas de código**: incluye un framework de testing con soporte para **JUnit** y **TestNG**, permitiendo probar los componentes de Spring de manera eficiente.

Estos módulos son opcionales, por lo que puedes elegir los que mejor se adapten a tu proyecto.



SPRING MVC

Antes de profundizar en la inyección de dependencias, es importante conocer **Spring MVC**, una funcionalidad clave para el desarrollo de aplicaciones web.

Spring Web MVC es un subproyecto de Spring diseñado para facilitar la creación de aplicaciones web utilizando el **patrón Modelo-Vista-Controlador (MVC)**. Este

enfoque ayuda a organizar mejor el código, separando la lógica del negocio, la interfaz de usuario y la gestión de solicitudes.

¿Qué es el patrón de diseño MVC?

El patrón **Modelo-Vista-Controlador (MVC)** organiza una aplicación en tres componentes principales:

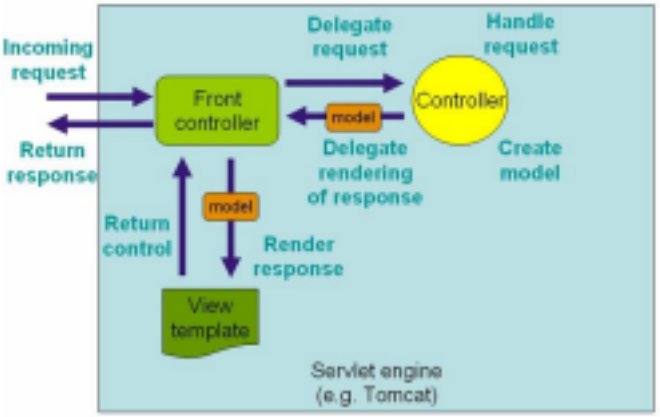
- **Modelo**: Maneja el acceso a los datos, la lógica del negocio y la interacción con la base de datos. No debe contener código relacionado con la presentación de la información.
- **Vista**: Se encarga de la presentación de los datos al usuario. Aunque puede acceder al modelo, no debe contener lógica de negocio.
- **Controlador**: Actúa como intermediario entre el modelo y la vista. Recibe las solicitudes del usuario, las procesa y determina qué respuesta enviar a la vista.

Este patrón permite una mayor escalabilidad y facilita el mantenimiento del código al mantener separadas las responsabilidades de cada componente.

Procesamiento de una petición en Spring MVC

Spring MVC se basa en el **patrón Modelo-Vista-Controlador (MVC)** para gestionar las peticiones HTTP y sus respuestas. Además, sigue el **patrón Front Controller**, lo que significa que todas las solicitudes pasan por un único punto de entrada.

A continuación, se describe el flujo típico de procesamiento de una petición en Spring MVC:



1. **Recepción de la petición**
   - Todas las peticiones HTTP son dirigidas a un **Front Controller**.
   - En el caso de Spring, este controlador es la clase **DispatcherServlet**, que actúa como intermediario.
2. **Determinación del controlador**
   - El **DispatcherServlet** analiza la URL y determina qué controlador (**Controller**) debe procesar la solicitud.

    - Para esta tarea, se utiliza un componente llamado **HandlerMapping**.
3. **Ejecución del controlador**
    - El controlador ejecuta la lógica de negocio y devuelve los datos encapsulados en un objeto **Model**.
    - También devuelve un **nombre lógico de la vista** (generalmente una cadena de texto).
4. **Resolución de la vista**
    - Un componente llamado **ViewResolver** se encarga de traducir el nombre lógico de la vista en una ubicación física dentro del proyecto.
5. **Renderización de la respuesta**
    - Finalmente, el **DispatcherServlet** redirige la petición a la vista correspondiente, que genera la respuesta para el usuario.

Este mecanismo permite estructurar las aplicaciones de manera modular, separando la lógica de negocio, la gestión de peticiones y la presentación de la información.

Inyección de dependencias en Spring

La **inyección de dependencias (Dependency Injection, DI)** es una de las características fundamentales de Spring. Su propósito es evitar que las clases instancien directamente los objetos de los que dependen, delegando esta tarea a un **contenedor de dependencias**.

En lugar de que cada clase cree sus propios objetos, Spring se encarga de instanciarlos y proporcionarlos cuando sean necesarios. Esto mejora la **modularidad, el mantenimiento y la reutilización del código**.

¿Cómo funciona la inyección de dependencias?

La inyección de dependencias puede realizarse de dos maneras:

- **Mediante el constructor**: los objetos requeridos se proporcionan como parámetros al constructor de la clase.
- **Mediante métodos setter**: los objetos son asignados a través de métodos `set()`.

Spring utiliza un **contenedor de inyección de dependencias** que gestiona automáticamente la creación y provisión de los objetos. Esta configuración se puede definir de dos formas:

1. **Mediante un archivo XML** (por ejemplo, `application-context.xml`).
2. **Mediante anotaciones en el código Java**, lo que permite una configuración más concisa.

¿Qué es un Bean en Spring?

En el contexto de Spring, un **Bean** es un objeto que es gestionado por el **contenedor de Spring**. Para que una clase sea considerada un Bean de Spring, debe cumplir con los siguientes requisitos:

- Sus atributos deben ser **privados** (`private`).
- Debe contar con métodos **getter y setter** públicos para los atributos que se deseen exponer.
- Debe tener un **constructor público sin parámetros**.

A diferencia de los JavaBeans tradicionales, los **Spring Beans son instancias administradas por el contenedor de Spring**, lo que permite su uso controlado y optimizado.

El Contenedor de Spring y la Inversión de Control (IoC)

Spring implementa el principio de **Inversión de Control (IoC, Inversion of Control)**. Este concepto se basa en la idea de que **en lugar de que una clase cree sus propias dependencias, es el contenedor de Spring quien se encarga de proporcionarlas**.

Este patrón también se conoce como el **Principio de Hollywood**: *"No nos llames, nosotros te llamaremos"*.

El contenedor IoC de Spring se encarga de:

- **Gestionar la creación de objetos** automáticamente.
- **Resolver las dependencias** entre los objetos de la aplicación.
- **Administrar el ciclo de vida** de los objetos según la configuración establecida.

Beneficios de la inyección de dependencias en Spring

Uno de los principales beneficios de la inyección de dependencias es la **eficiencia en el manejo de recursos**. Por ejemplo, si una clase necesita una conexión a la base de datos:

- **Sin inyección de dependencias**, cada instancia de la clase crearía su propia conexión, lo que podría afectar el rendimiento.
- **Con inyección de dependencias**, la conexión a la base de datos se crea una sola vez y se comparte entre todas las instancias, optimizando los recursos del sistema.

Para más información, puedes consultar la documentación oficial de Spring MVC: 🔗 **Spring MVC Documentation**