Spring Framework

… EN DESARROLLO …

Overview

Spring Framework es un poderoso y ampliamente utilizado marco de desarrollo de software para aplicaciones empresariales en Java. Diseñado para simplificar y acelerar el desarrollo de aplicaciones, Spring ofrece un enfoque integral que abarca desde la configuración hasta la implementación, abordando varios aspectos del desarrollo de software como la inversión de control, la inyección de dependencias, la gestión de transacciones, la seguridad y mucho más.

Una de las características distintivas de Spring es su **enfoque modular y extensible**, permitiendo a los desarrolladores elegir los módulos específicos que necesitan para sus proyectos. Además, fomenta las mejores prácticas de programación y sigue el principio de diseño de "Programación Orientada a Aspectos" (AOP), que facilita la separación de preocupaciones y mejora la modularidad del código.

Spring Framework se utiliza comúnmente para construir aplicaciones empresariales robustas y escalables, facilitando la creación de servicios web, aplicaciones basadas en la arquitectura Modelo-Vista-Controlador (MVC), integración con bases de datos, gestión de transacciones y mucho más. Con una comunidad activa y un ecosistema de proyectos relacionados, Spring ha evolucionado para adaptarse a las cambiantes demandas del desarrollo de software, convirtiéndose en una opción popular entre los desarrolladores Java.

Spring Boot es una extensión del popular Spring Framework que se centra en simplificar drásticamente el proceso de desarrollo de aplicaciones Java, especialmente aplicaciones basadas en Spring. Su objetivo principal es facilitar la creación de aplicaciones autónomas, autocontenidas y listas para la producción con la menor cantidad de configuración posible.

La relación entre Spring Boot y Spring Framework es fundamental, ya que Spring Boot se construye sobre la base sólida proporcionada por Spring. Spring Boot utiliza las características clave de Spring, como la inversión de control (IoC) y la inyección de dependencias, pero agrega una capa de convenciones y configuraciones por defecto para acelerar el desarrollo.

Lo más notable de Spring Boot es su enfoque de "opinión sobre la configuración", lo que significa que proporciona configuraciones predeterminadas sensatas para la mayoría de los casos de uso, permitiendo a los desarrolladores empezar rápidamente con sus proyectos sin tener que configurar extensamente. No obstante, sigue siendo altamente personalizable, permitiendo a los desarrolladores anular las configuraciones por defecto según sea necesario.

Con Spring Boot, el proceso de desarrollo se simplifica mediante la inclusión de un servidor embebido, como Tomcat o Jetty, lo que elimina la necesidad de desplegar la aplicación en un servidor externo. También facilita la gestión de dependencias mediante el uso de la herramienta Spring Initializr para generar proyectos con las dependencias necesarias.

En resumen, Spring Boot es una extensión de Spring Framework diseñada para hacer que el desarrollo de aplicaciones Java sea más rápido, sencillo y eficiente al proporcionar configuraciones por defecto y convenciones inteligentes sin sacrificar la flexibilidad y la potencia que ofrece Spring Framework.

> Introducción generada por ChatGPT

Spring Core Annotations

Estas anotaciones forman parte del paquete org.springframework.beans.factory.annotation y org.springframework.context.annotation.

@Autowired

La anotación `@Autowired` se utiliza para marcar una dependencia que el motor DI de Spring resolverá e inyectará. Esta anotación se puede usar en un **constructor**, en un **método *'setter'*** o en un **campo**:

```
// Constructor injection
class Car {
  Engine engine;

  @Autowired
  Car(Engine engine) {
    this.engine = engine;
  }
}
```

A partir de la versión 4.3, no es necesario anotar constructores con `@Autowired` de forma explícita a menos que se haya declarado al menos dos constructores.

```
// Setter injection
class Car {
  Engine engine;

  @Autowired
  void setEngine(Engine engine) {
    this.engine = engine;
  }
}
``
```

java` // Field injection class Car { @Autowired Engine engine; }

```
`@Autowired` tiene un argumento booleano llamado `required` con un valor predeterminado de `true`. Este argumento ajusta

Si se utiliza la inyección del constructor, **todos los argumentos del constructor son obligatorios**.

- [Javadoc](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/beans/factory/annotati

- [Guide to Spring @Autowired](https://www.baeldung.com/spring-autowire)

- [Constructor Dependency Injection in Spring](https://www.baeldung.com/constructor-injection-in-spring)

### @Bean

La anotación `@Bean` marca un _'factory method'_ que crea una instancia de un _bean_ de Spring:
```

java @Configuration public class AppConfiguration { @Bean public Engine engine() { return new Engine(); } }

```
**Spring llama a estos métodos** cuando se requiere una nueva instancia del tipo de retorno.

El bean resultante tiene el mismo nombre que el _'factory method'_. Si se requiere que tenga un nombre diferente, se pue
```

java @Configuration public class AppConfiguration { @Bean("engine") public Engine getEngine() { return new Engine(); } }

```
Hay que tener en cuenta que **todos los métodos anotados con `@Bean` deben estar en clases `@Configuration`**.

- [Javadoc](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/annotation/Bean

### @Qualifier
```

```
Se usa la anotación `@Qualifier` junto con `@Autowired` para proporcionar la **identificación del bean** o el **nombre d
```

java class Bike implements Vehicle {}

class Car implements Vehicle {}

```
En caso de ambigüedad, se utiliza `@Qualifier` para indicar a Spring **el bean a inyectar**:
```

java // Using constructor injection @Autowired Biker(@Qualifier("bike") Vehicle vehicle) { this.vehicle = vehicle; }

// Using setter injection @Autowired void setVehicle(@Qualifier("bike") Vehicle vehicle) { this.vehicle = vehicle; }

@Autowired @Qualifier("bike") void setVehicle(Vehicle vehicle) { this.vehicle = vehicle; }

// Using field injection @Autowired @Qualifier("bike") Vehicle vehicle;

```
- [Javadoc](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/beans/factory/annotati

### @Value

Se puede utilizar la anotación `@Value` para inyectar valores de propiedad en beans. Es compatible con constructorres, m
```

java // Constructor injection Engine(@Value("8") int cylinderCount) { this.cylinderCount = cylinderCount; }

```
Por supuesto, inyectar valores estáticos no es útil. Por lo tanto, podemos usar **cadenas _'placeholder'_** en `@Value`

Por ejemplo, un valor en un fichero externo podría ser:
```

text engine.fuelType=petrol

```
Podemos inyectar el valor de esta forma:
```

java @Value("${engine.fuelType}") String fuelType;

```
- [Javadoc](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/beans/factory/annotati
- [A Quick Guide to Spring @Value](https://www.baeldung.com/spring-value-annotation)

### @DependsOn

La anotación `@DependsOn` se puede utilizar para hacer que Spring **inicialice otros beans antes del anotado**. Normalme

Solo necesitamos esta anotación cuando **las dependencias están implícitas**, por ejemplo, carga del controlador JDBC o

Podemos usar `@DependsOn` en la clase dependiente especificando los nombres de los beans de dependencia. El argumento de
```

java @DependsOn("engine") class Car implements Vehicle {}

```
Alternativamente, si se define un bean con la anotación `@Bean`, el _'factory method'_ debería anotarse con `@DependsOn`
```

java @Bean @DependsOn("fuel") Engine engine() { return new Engine(); }

```
- [Javadoc](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/annotation/Depe

### @Lazy

La anotación `@Lazy` se utiliza cuando queremos inicializar un bean de forma diferida. De forma predeterminada, Spring c

Sin embargo, hay casos en los que necesitamos crear un bean cuando se solicita solicitamos, no al iniciar la aplicación.

Esta anotación tiene un argumento con el valor predeterminado de verdadero. Es útil para anular el comportamiento predet

Por ejemplo, marcar beans para que se carguen inmediatamente cuando la configuración global es diferida, o configurar me
```

java @Configuration @Lazy class VehicleFactoryConfig {

```
@Bean
@Lazy(false)
Engine engine() {
    return new Engine();
}
```

}

```
- [Javadoc](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/annotation/Lazy

- [A Quick Guide to the Spring @Lazy Annotation](https://www.baeldung.com/spring-lazy-annotation)

### @Lookup

Un método anotado con `@Lookup` le indica a Spring que devuelva una instancia del tipo de retorno del método cuando lo i

- [Javadoc](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/beans/factory/annotatio

- [@Lookup Annotation in Spring](https://www.baeldung.com/spring-lookup)

### @Primary

A veces necesitamos definir **múltiples beans del mismo tipo**. En estos casos, la inyección no tendrá éxito porque Spri

Ya vimos una opción para manejar este escenario: marcar todos los puntos de conexión con `@Qualifier` y especificar el n

Sin embargo, la mayoría de las veces necesitamos un bean específico y rara vez los otros. Podemos usar `@Primary` para s
```

java @Component @Primary class Car implements Vehicle {}

@Component class Bike implements Vehicle {}

@Component class Driver { @Autowired Vehicle vehicle; }

@Component class Biker { @Autowired @Qualifier("bike") Vehicle vehicle; }

```
En el ejemplo anterior, _'Car'_ es el vehículo principal. Por lo tanto, en la clase _'Driver'_, Spring inyecta un bean c

- [Javadoc](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/annotation/Prim

### @Scope
```

```
Usamos `@Scope` para definir el ámbito de una clase `@Component` o una definición de `@Bean`. Puede ser **_singleton_**
```

java @Component @Scope("prototype") class Engine {}

```
**El ámbito por defecto es 'singleton'**. Esto significa que Spring crea una única instancia del bean y la reutiliza en

- [Javadoc](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/annotation/Scop

### @Profile

Si queremos que Spring use una clase `@Component` o un método `@Bean` solo cuando un perfil específico esté activo, pode
```

java @Component @Profile("sportDay") class Bike implements Vehicle {}

```
- [Javadoc](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/annotation/Proj

- [Spring Profiles](https://www.baeldung.com/spring-profiles)

### @Import

La anotación `@Import` en Spring se utiliza para importar configuraciones adicionales a una configuración principal de I

Si tenemos una clase anotada con `@Configuration` que define beans y configuraciones específicas para la aplicación, se
```

java @Configuration @Import(MyAdditionalConfig.class) public class MainConfig { // Configuración principal de la aplicación }

```
Otro uso de esta antoación es que podemos utilizar **clases específicas anotadas con `@Configuration` sin escaneo de cor
```

java @Configuration @Import({DataSourceConfig.class, SecurityConfig.class}) public class MainConfig { // Configuración principal de la aplicación }

```
Por último, esta anotación sirve para importar clases no relacionadas con `@Configuration`, es decir, además de importar
```

java @Configuration @Import({MyUtilityClass.class, AnotherHelper.class}) public class MainConfig { // Configuración principal de la aplicación }

```
Aquí, _'MyUtilityClass'_ y _'AnotherHelper'_ son clases normales que no necesariamente son configuraciones de Spring.

- [Javadoc](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/annotation/Impo

### @ImportResource

Podemos importar configuraciones XML con esta anotación. Podemos especificar las ubicaciones de los archivos XML utiliza
```

java @Configuration @ImportResource("classpath:/annotations.xml") class VehicleFactoryConfig {}

```
La anotación `@ImportResource` se utiliza exclusivamente para importar configuraciones desde archivos XML dentro del cor

En contraste, `@Import` se utiliza para importar configuraciones y componentes de otras clases de configuración, ya sear

- [Javadoc](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/annotation/Impo
```

```
### @PropertySource

Con esta anotación, podemos definir archivos de propiedades ('.properties' o '.yml') para la configuración de la aplica
```

java import org.springframework.context.annotation.Configuration; import org.springframework.context.annotation.PropertySource;

@Configuration @PropertySource("classpath:config.properties") public class AppConfig { // Configuración adicional de la aplicación }

```
Estos archivos contienen configuraciones como URLs de bases de datos, rutas de archivos, configuraciones de conexión, et

Las propiedades cargadas se integran con el `Environment` de Spring, lo que permite acceder a ellas desde cualquier part
```

java import org.springframework.beans.factory.annotation.Autowired; import org.springframework.core.env.Environment; import org.springframework.stereotype.Component;

@Component public class MyComponent {

```
@Autowired
private Environment env;

public void someMethod() {
    String dbUrl = env.getProperty("database.url");
    // Utilizar la propiedad dbUrl...
}
```

}

```
`@PropertySource` aprovecha la función de anotaciones repetidas de Java 8, lo que significa que podemos marcar una clase
```

java @Configuration @PropertySource("classpath:/annotations.properties") @PropertySource("classpath:/vehicle-factory.properties") class VehicleFactoryConfig {}

```
- [Javadoc](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/annotation/Prop

### @PropertySources

Podemos usar esta anotación para especificar múltiples configuraciones de `@PropertySource`:
```

java @Configuration @PropertySources({ @PropertySource("classpath:/annotations.properties"), @PropertySource("classpath:/vehicle-factory.properties") }) class VehicleFactoryConfig {}

```
Tenga en cuenta que desde Java 8 se puede lograr el mismo resultado con la función de anotaciones repetidas.

- [Javadoc](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/annotation/Prop

---

## [Spring Web Annotations](https://www.baeldung.com/spring-mvc-annotations)

Estas anotaciones forman parte del paquete [org.springframework.web.bind.annotation](https://docs.spring.io/spring-frame

### @RequestMapping
```

```
En pocas palabras, `@RequestMapping` marca métodos manejadores de peticiones dentro de clases anotadas con `@Controller`

- **path (o sus alias name y value)**: indica a qué URL está mapeado el método.

- **method**: define los métodos HTTP compatibles.

- **params**: filtra las peticiones basándose en la presencia, ausencia o valor de parámetros HTTP.

- **headers**: filtra las peticiones basándose en la presencia, ausencia o valor de cabeceras HTTP.

- **consumes**: especifica los tipos de medios que el método puede consumir en el cuerpo de la petición HTTP.

- **produces**: especifica los tipos de medios que el método puede producir en el cuerpo de la respuesta HTTP.
```

java @Controller class VehicleController {

```
@RequestMapping(value = "/vehicles/home", method = RequestMethod.GET)
String home() {
    return "home";
}
```

}

```
Podemos proporcionar configuraciones predeterminadas para todos los métodos manejadores en una clase `@Controller` si ap

Por ejemplo, la siguiente configuración tiene el mismo efecto que la configuración del ejemplo anterior:
```

java @Controller @RequestMapping(value = "/vehicles", method = RequestMethod.GET) class VehicleController {

```
@RequestMapping("/home")
String home() {
    return "home";
}
```

}

```
Además, `@GetMapping`, `@PostMapping`, `@PutMapping`, `@DeleteMapping` y `@PatchMapping` son variantes de `@RequestMapp

Estas anotaciones están disponibles desde la **versión 4.3 de Spring**.

- [Javadoc](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/annotation/Re

### @RequestBody

La anotación `@RequestBody` mapea el cuerpo de la solicitud HTTP a un objeto:
```

java @PostMapping("/save") void saveVehicle(@RequestBody Vehicle vehicle) { // … }

```
La deserialización es automática y depende del tipo de contenido de la solicitud.

- [Javadoc](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/annotation/Re

### @PathVariable
```

```
Esta anotación indica que un argumento de método está vinculado a una variable de plantilla URI. Si el nombre de la part
```

java import org.springframework.web.bind.annotation.*;

@RestController @RequestMapping("/users") public class UserController {

```
@GetMapping("/{userId}")
public String getUserById(@PathVariable Long userId) {
    return "Obteniendo usuario con ID: " + userId;
}
```

}

```
Sin embargo, se puede vincular un argumento (o varios argumentos) de método a una de las partes de la plantilla con `@Pa
```

java import org.springframework.web.bind.annotation.*;

@RestController @RequestMapping("/products") public class ProductController {

```
@GetMapping("/{category}/{productId}")
public String getProductDetails(
        @PathVariable("category") String category,
        @PathVariable("productId") Long productId) {
    return "Detalles del producto: categoría = " + category + ", ID = " + productId;
}
```

}

```
Además, se puede marcar una variable de ruta como opcional estableciendo el argumento `required = false`:
```

java @RequestMapping("/{id}") Vehicle getVehicle(@PathVariable(required = false) long id) { // … } ```

- Javadoc

@RequestParam

TODO

- Javadoc

@ResponseBody

TODO

- Javadoc

@ExceptionHandler

TODO

- Javadoc

@ResponseStatus

TODO

- Javadoc

@RestController

TODO

- Javadoc

@ModelAttribute

TODO

- Javadoc

@CrossOrigin

TODO

- Javadoc

Enlaces de interés

- https://spring.io/
- https://docs.spring.io/spring-framework/reference/
- https://docs.spring.io/spring-boot/docs/current/reference/html
- https://www.baeldung.com/spring-core-annotations
- https://www.baeldung.com/spring-boot-start

Licencia