Contenedor IoC (Inversión de Control) de Spring

Permíteme desglosar cómo usar el contenedor Spring IoC (Inversión de Control) de una manera clara y práctica. El contenedor Spring IoC es el núcleo del framework Spring, responsable de gestionar el ciclo de vida y la configuración de los objetos de la aplicación (llamados beans). "Invierte"el control manejando la creación de objetos y la inyección de dependencias por ti, en lugar de que tú los crees y los conectes manualmente.

Aquí tienes una guía paso a paso para empezar:

---

1. Entender los conceptos básicos

- **Qué hace**: El contenedor IoC crea, configura y gestiona beans (tus objetos de aplicación) basándose en metadatos que proporcionas (a través de XML, anotaciones Java o código Java).
- **Conceptos clave**:
  - **Beans**: Objetos gestionados por el contenedor.
  - **Inyección de Dependencias (DI)**: El contenedor inyecta dependencias (otros beans o valores) en tus objetos.
  - **Configuración**: Le dices al contenedor qué beans crear y cómo conectarlos.

---

2. Configurar tu proyecto

Para usar Spring IoC, necesitas un proyecto Spring. Si estás empezando de cero: - Usa **Spring Boot** (la manera más sencilla) o Spring plano. - Añade dependencias en tu `pom.xml` (si usas Maven): `xml <dependency> <groupId>org.springframework</groupId> <artifactId>spring-context</artifactId> <version>6.1.3</ver <!-- Usa la última versión --> </dependency>` - Para Spring Boot, usa: `xml <dependency> <groupId>org.spr <artifactId>spring-boot-starter</artifactId> <version>3.2.2</version> <!-- La última disponible --> </dependency>`

---

3. Definir tus beans

Puedes definir beans de tres maneras principales:

a) Usando anotaciones (más común)

- Crea una clase Java simple y anótala con `@Component` (o anotaciones especializadas como `@Service`, `@Repository`, etc.).
- Ejemplo:

```java
import org.springframework.stereotype.Component;

@Component
public class MyService {
    public void doSomething() {
        System.out.println("Haciendo algo!");
    }
}
```

b) Usando configuración Java

- Crea una clase de configuración con `@Configuration` y define beans con `@Bean`.
- Ejemplo:

```java
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class AppConfig {
    @Bean
    public MyService myService() {
        return new MyService();
    }
}
```

c) Usando XML (enfoque legado)

- Define beans en un archivo XML (por ejemplo, `beans.xml`):

```xml
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean id="myService" class="com.example.MyService"/>
</beans>
```

4. Inicializar el contenedor IoC

Necesitas iniciar el contenedor para gestionar tus beans.

a) Con Spring Boot

- Spring Boot lo hace automáticamente. Solo crea una clase principal con `@SpringBootApplication`:

```java
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MyApp {
    public static void main(String[] args) {
        SpringApplication.run(MyApp.class, args);
    }
}
```

- El contenedor escanea las clases con `@Component` y las gestiona.

b) Con Spring plano (basado en anotaciones)

- Usa `AnnotationConfigApplicationContext`:

```java
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class Main {
    public static void main(String[] args) {
        AnnotationConfigApplicationContext context =
            new AnnotationConfigApplicationContext(AppConfig.class);
        MyService service = context.getBean(MyService.class);
        service.doSomething();
        context.close();
    }
}
```

c) Con XML

- Usa `ClassPathXmlApplicationContext`:

```java
import org.springframework.context.support.ClassPathXmlApplicationContext;
```

```
public class Main {
    public static void main(String[] args) {
        ClassPathXmlApplicationContext context =
            new ClassPathXmlApplicationContext("beans.xml");
        MyService service = context.getBean("myService", MyService.class);
        service.doSomething();
        context.close();
    }
}
```

---

5. Inyectar dependencias

El contenedor IoC puede inyectar dependencias en tus beans.

a) Inyección de constructor (recomendada)

```
@Component
public class MyController {
    private final MyService myService;

    public MyController(MyService myService) {
        this.myService = myService;
    }

    public void execute() {
        myService.doSomething();
    }
}
```

b) Inyección de setter

```
@Component
public class MyController {
    private MyService myService;

    @Autowired
    public void setMyService(MyService myService) {
        this.myService = myService;
```

```
    }
}
```

c) Inyección de campo (menos preferida)

```
@Component
public class MyController {
    @Autowired
    private MyService myService;

    public void execute() {
        myService.doSomething();
    }
}
```

- Usa `@Autowired` (específico de Spring) o `@Inject` (estándar JSR-330) para marcar los puntos de inyección.

---

6. Ejecutar y usar los beans

- Una vez que el contenedor esté en funcionamiento, conectará tus beans y sus dependencias. Puedes obtener beans con `context.getBean()` (Spring plano) o dejar que Spring Boot lo maneje automáticamente a través de la inyección.

Ejemplo con Spring Boot:

```
@SpringBootApplication
public class MyApp {
    public static void main(String[] args) {
        ApplicationContext context = SpringApplication.run(MyApp.class, args);
        MyController controller = context.getBean(MyController.class);
        controller.execute();
    }
}
```

---

7. Consejos adicionales

- **Alcances**: Por defecto, los beans son singletons. Usa `@Scope("prototype")` para nuevas instancias por solicitud.

- **Escaneo de componentes**: Añade `@ComponentScan` a tu clase de configuración si los beans están en diferentes paquetes.
- **Perfiles**: Usa `@Profile` para activar beans en entornos específicos (por ejemplo, “dev”, “prod”).
- **Cerrar el contexto**: En aplicaciones no Spring Boot, llama a `context.close()` para liberar recursos.

¡Y eso es todo! Empieza con Spring Boot y anotaciones por simplicidad. Si te atas en alguna parte, como conectar una dependencia complicada o depurar, ¡no dudes en preguntar y profundizaré contigo!