Gestione delle dipendenze tra bean – Parte II

Cosa imparerai

- Come si definiscono e gestiscono le dipendenze tra oggetti in Spring tramite la Dependency Injection

Ricordiamo cos'è la Dependency Injection

La **Dependency Injection** (**DI**) è un design pattern utilizzato per implementare l'IoC.

È un processo in base al quale gli oggetti definiscono le loro dipendenze da altri oggetti.

Il container inietta le dipendenze quando crea il bean.

Il bean non si occupa di nulla, c'è il container che lavora per lui (da cui il nome di *Inversion of Control*).

Il pattern DI può essere implementato in tre modi, ma in Spring ne sono utilizzati due:

- **Constructor Injection**
- **Setter Injection**

Dependency Injection

Setter Injection

Questa modalità è realizzata chiamando il metodo setter dell'oggetto da utilizzare.

```java
public class Biblioteca {
    private BibliotecaService service;

    public Biblioteca() {
        super();
    }

    public void setService(BibliotecaService service) {
        this.service = service;
    }

    public List<Libro> getLibriDisponibili() {
        return service.getLibriDisponibili();
    }
}
```

```java
public class BibliotecaFactory {
    public static Biblioteca getBiblioteca() {
        Biblioteca b = new Biblioteca();
        b.setService(new BibliotecaServiceImpl());

        return b;
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        BibliotecaFactory.getBiblioteca().getLibriDisponibili();
    }
}
```

Setter Injection in Spring

La Setter Injection viene eseguita dal contenitore mediante la chiamata ai metodi setter del bean, dopo aver richiamato un costruttore senza argomenti o un metodo factory statico senza argomenti per istanziare il bean.

Facciamo un esempio…

Il service layer dell'entità *Ordine* accede al service layer dell'entità *Cliente* per recuperare le informazioni di un cliente.

```
public class OrdineService {
    private ClienteService cliente;

    public void setCliente(ClienteService cliente) {
        this.cliente = cliente;
    }

    public String sayHello() {

    public void saluto() {
}
```

*Ecco la DI basata sul metodo **set** della variabile cliente*

Setter Injection in Spring

Come fa Spring ad effettuare la DI tramite il metodo set di una variabile?

Se usiamo XML per i metadati, dobbiamo utilizzare l'elemento **<property … />**.

- Utilizzeremo l'elemento **<ref />** se la variabile è un bean.
- Utilizzeremo l'elemento **<value>** se la variabile è un tipo semplice.

```java
public class OrdineService {
    private ClienteService cliente;
    private String statoPredefinito;

    public void setCliente(ClienteService cliente) {
        this.cliente = cliente;
    }

    public void setStatoPredefinito(String statoPredefinito) {
        this.statoPredefinito = statoPredefinito;
    }

    public String sayHello() {

    public void saluto() {

}
```

```xml
<bean id="ordineService" class="it.test.service.OrdineService">
    <property name="cliente">
        <ref bean="clienteService"/>
    </property>
    <property name="statoPredefinito">
        <value>COMPLETATO</value>
    </property>
</bean>
```

Gestione delle dipendenze tra bean

Posso usare la Constructor e la Setter Injection insieme?

Si, è possibile combinare le due DI.

Meglio usare la Constructor o la Setter Injection?

È una buona prassi usare la Constructor DI per le dipendenze obbligatorie e la Setter DI per le dipendenze opzionali.

Spring sostiene l'utilizzo della Constructor DI, in quanto consente di implementare i componenti dell'applicazione come oggetti immutabili e garantisce che le dipendenze richieste non siano nulle.

Inoltre, i componenti inseriti dal costruttore vengono sempre restituiti inizializzati e pronti all'uso all'oggetto chiamante.

Gestione delle dipendenze tra bean

Esempio di DI – la configurazione di un DataSource

```
<bean id="myDataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
    <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
    <property name="url" value="jdbc:mysql://…"/>
    <property name="username" value="…"/>
    <property name="password" value="..."/>
</bean>
```

Di cosa abbiamo parlato in questa lezione

- Come si definiscono e gestiscono le dipendenze tra oggetti in Spring tramite la Dependency Injection