Spring framework 5

Dependency injection

La inyección de dependencias es la técnica donde un objeto provee las dependencias que otro objeto requiere, a continuación algunas características:

-Reduce el **acoplamiento** entre clases

-Las clases no obtienen sus dependencias de otros componentes, éstas son proporcionadas por el **contenedor**

-Es posible **cambiar las dependencias** fácilmente de acuerdo a configuraciones

-Los beans administrados por spring se les conoce como "**beans**"

-No es necesario implementar ninguna interfaz para crear un bean de spring

-Es posible realizar dependency injection a través de XML, anotaciones y código Java

-El contenedor de Spring administrará el ciclo de vida de los beans

Tipos de dependency injection

Los siguientes son los tipos de **dependency injection** disponibles en spring:

-Por **constructor**
-Por métodos **setters**
-Por **atributos** de clase

Inyección por constructor

La anotación **@Autowired** se utiliza para inyectar un bean, es posible hacerlo a través del **constructor** como se muestra:

```
@Service
public class ProfileService {
      private UserRepository userRepository;

      @Autowired
      public ProfileService(UserRepository
userRepository) {
            this.userRepository = userRepository;
      }
 ....
}
```

Inyección por setter

Es posible inyectar componentes a través de los métodos **setter**:

```
@Service
public class ProfileService {
      private UserRepository userRepository;

      @Autowired
      public void setUserRepository(UserRepository
      userRepository) {
            this.userRepository = userRepository;
      }

      public void createProfile() {
            userRepository.createUser();
      }
}
```

Inyección por atributo

Es posible inyectar componentes en la definición del **atributo**:

```
@Service
public class ProfileService {
      @Autowired
      private UserRepository userRepository;

      .....
}
```

Stereotypes

Permiten indicar a spring que debe administrar objetos de la clase anotada, los principales 4 tipos son:

-**@Component**
-**@Service**
-**@Controller**
-**@Repository**

Los ***stereotypes*** permiten dar un rol a las clases que serán administradas por spring, todas ellas tienen el mismo comportamiento y se aplican solo a las implementaciones.

Component

Es la anotación más general y se utiliza para todas las clases que no entran dentro de las categorías **@Controller**, **@Service** o **@Repository**, ejemplo:

```
@Component
public class DateHelper {

}
```

Controller

Define a una clase que será un controller en Spring MVC, es utilizada para la capa de presentación, ejemplo:

```
@Controller
public class BookController {

}
```

Service

La lógica de negocio de una aplicación vive en una capa de servicios, utilizaremos la anotación **@Service** para indicar que la clase pertenece a esta capa:

```
@Service
public class ProfileService {

}
```

Repository

Las clases anotadas con **@Repository** serán clases que realizarán acceso a datos, ejemplo:

```
@Repository
public class ProfileRepository {

}
```

Las clases anotadas con @Repository tendrán la ventaja de que las excepciones que generen serán traducidas de forma automática a subclases de **DataAccessExeption**.

Para habilitar la traducción de excepciones se debe declarar el siguiente bean:

```
@Bean
 public PersistenceExceptionTranslationPostProcessor
exceptionTranslation() {
 return new
PersistenceExceptionTranslationPostProcessor();
 }
```

Qualifier

Dado que spring realiza inyección de dependencias por tipo
La anotación **@Qualifier** permite remover la ambiguedad de referencias al inyectar dependencias.

@Qualifier

En el caso en el que tengamos una interfaz con múltiples implementaciones podemos hacer uso de **@Qualifier** para especificar el bean que deseamos inyectar en nuestro bean, veamos el siguiente ejemplo:

```
interface UserRepository {
}

@Repository
class LdapUserRepository implements
UserRepository{
....
}

@Repository
class DatabaseUserRepository implements
UserRepository {
....
}

@Service
public class ProfileService {

      @Autowired
      @Qualifier("ldapUserRepository")
      private UserRepository userRepository;

----
}
```

@Primary

@Primary se utiliza cuando existe más de un bean del mismo tipo y se desea definir uno por defecto, veamos el siguiente ejemplo:

```
interface UserRepository {
}

@Repository
@Primary
class LdapUserRepository implements
UserRepository{
....
}

@Repository
class DatabaseUserRepository implements
UserRepository {
....
}

@Service
public class ProfileService {

      @Autowired
      private UserRepository userRepository;

----
}
```

En este ejemplo podemos ver que aunque tenemos 2 implementaciones de la interfaz **UserRepository** no es necesario definir un qualifier ya que por defecto tomará **LdapRepository** puesto que está anotado con **@Primary**.

< DEVS4J >
Hecho por Devs para Devs