

Anotaciones Inyección de Dependencia Spring



En lugar de utilizar XML para describir una dependencia de un componente, podemos usar la configuración en la propia clase componente mediante anotaciones, sobre el atributo en cuestión o sobre un método setter o constructor



*Es una alternativa
a la inyección de
dependencia basada
en XML*

Anotaciones introducidas en Spring:

Spring 2.0

- @Required

Spring 2.5

- @Autowired
- Anotaciones en común con la Plataforma Java 1.0 (JSR-250):
@Resource, @PostConstruct,
@PreDestroy

Spring 3.0

- JSR 330 (Inyección de Dependencia de la Plataforma Java): @Inject, @Qualifier, @Named, y @Provider

@Autowired



Anotación @Autowired



Especifica que se inyectará un bean o componente de spring en un atributo de otro bean, es decir para inyectar beans de la aplicación en el componente actual

Por defecto, la inyección falla si no encuentra candidatos disponibles

Se utiliza en el código fuente de Java para la especificación de requisitos DI (en lugar de XML)

Anotación @Autowired

Lugares donde se puede utilizar @Autowired

- Atributos
- Métodos setter
- Constructor
- Otros métodos arbitrarios



@Autowired vía método Setter

```
public class SimpleMovieLister {  
  
    private MovieFinder movieFinder;  
  
    // El bean/objeto MovieFinder será inyectado automáticamente  
    @Autowired  
    public void setMovieFinder(MovieFinder movieFinder) {  
        this.movieFinder = movieFinder;  
    }  
  
    // ...  
}
```

@Autowired en método arbitrario

- También podemos aplicar la anotación *Autowired* a otros métodos de cualquier forma de nombre y número de argumentos:

```
public class RecomendarPelicula {  
  
    private CatalogoPelicula catalogo;  
    private ClienteDao clienteDao;  
  
    // Beans CatalogoPelicula y ClienteDao son  
    // inyectados automáticamente  
    @Autowired  
    public void prepare(CatalogoPelicula catalogo, ClienteDao dao) {  
        this.catalogo = catalogo;  
        this.clienteDao = dao;  
    }  
    // ...  
}
```


@Autowired en constructor y atributo

```
public class RecomendarPelicula {  
  
    // @Autowired en atributo  
    @Autowired  
    private CatalogoPelicula catalogo;  
  
    private ClienteDao clienteDao;  
  
    // @Autowired en constructor  
    @Autowired  
    public MovieRecommender(ClienteDao dao) {  
        this.clienteDao = dao;  
    }  
  
    // ...  
}
```

@Required



@Required

La anotación @Required se aplica a los métodos setter del bean

Lanzará una excepción si no se ha establecido

Permite comprobar si el atributo se ha inyectado o no sin necesidad de utilizarlo en el código de la clase:

@Required

- *Permite comprobar si el atributo se ha inyectado o no sin necesidad de utilizarlo en el código de la clase:*

```
public class SimpleMovieLister {  
  
    private MovieFinder movieFinder;  
  
    // @Required indica que el atributo del bean en cuestión  
    // debe ser inyectado en el momento de la configuración,  
    // ya sea mediante un valor explícito asignado al atributo  
    // en la definición property del bean (en archivo xml)  
    // o mediante anotación autowiring.  
    @Required  
    public void setMovieFinder(MovieFinder movieFinder) {  
        this.movieFinder = movieFinder;  
    }  
  
    // ...  
}
```

*@Qualifier
Spring*



Ajustando @Autowired con Calificadores

Debido a que la inyección @Autowired por tipo puede dar lugar a múltiples candidatos, es necesario para tener más control sobre el proceso de selección utilizando calificadores.

Una forma de lograr esto es con la anotación @Qualifier de Spring

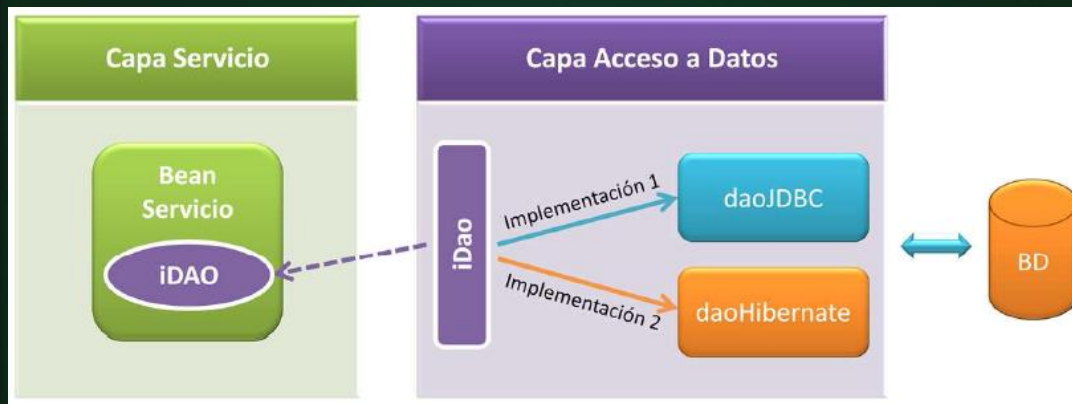


Ajustando @Autowired con Calificadores

```
public class RecomendarPelicula {  
  
    // Entre los múltiples candidatos de tipo CatalogoPelicula,  
    // seleccionamos el con nombre de bean "principal"  
    @Autowired  
    @Qualifier("principal")  
    private CatalogoPelicula catalogo;  
  
    // ...  
}
```



Ajustando @Autowired con Calificadores



Con esto evitamos conflictos y ambigüedad para inyectar una subclases o clases que implementan una interfaces, de este modo podemos indicar que implementación de componente inyectar (asociados al mismo tipo) es necesario tener un mayor control sobre el proceso de selección

Ajustando @Autowired con Calificadores

- La anotación @Qualifier también puede ser usada en argumentos de métodos o constructores:

```
public class RecomendarPelicula {  
  
    private CatalogoPelicula catalogo;  
    private ClienteDao clienteDao;  
  
    @Autowired  
    public void prepare(  
        @Qualifier("principal") CatalogoPelicula catalogo,  
        ClienteDao dao) {  
  
        this.catalogo = catalogo;  
        this.clienteDao = dao;  
    }  
    // ...  
}
```

*@PostConstruct,
@PreDestroy y
@Resource*

*JSR 250 (Anotaciones en comunes
de la plataforma Java)*



@PostConstruct y @PreDestroy

Permite invocar un evento para inicializar y destruir un bean



Puede haber una sola anotación de cada uno por clase

@PostConstruct y @PreDestroy

- *Permite invocar un evento para la inicialización y destrucción de un bean*
- *JSR-250 Anotaciones CDI Plataforma Java*

```
public class CachingMovieLister implements CachingLister {
```

```
    @PostConstruct
```

```
    public void populateCache() {
```

```
        // accede al cache del video durante la inicialización...
```

```
    }
```

```
    @PreDestroy
```

```
    public void clearCache() {
```

```
        // elimina el cache del video en la destrucción...
```

```
    }
```

```
}
```

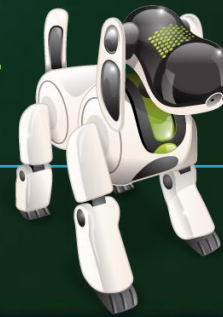
@PostConstruct: indica que el método debe ser llamado cuando una instancia del componente es instanciada por el contenedor

```
public class CachingMovieLister {  
  
    @PostConstruct  
    public void populateMovieCache() {  
        // accede al cache del video durante la inicialización...  
    }  
}
```



@PreDestroy : indica que el método debe ser llamado cuando finaliza el contexto y destruyendo también sus variables y estado

```
public class CachingMovieLister {  
  
    @PostConstruct  
    public void populateMovieCache() {  
        // accede al cache del video durante la inicialización...  
    }  
  
    @PreDestroy  
    public void clearMovieCache() {  
        // elimina el cache del video en la destrucción...  
    }  
}
```



*Anotación @Inject
(JSR 330
Plataforma Java)*



Anotación @Inject

Anotación @Inject (Plataforma Java)

Parte de la especificación JSR 330 Plataforma Java

@Inject se puede utilizar en lugar de la anotación @Autowired de Spring

@Inject, no tiene el atributo required como en la anotación @Autowired, el cual sirve para indicar si el valor a inyectar es opcional o no

*Configuración Spring
Basada en código Java
(NO XML)*



Anotaciones @Configuration y @Bean

Anotar una clase con @Configuration indica que la clase puede ser utilizada por el contenedor Spring IoC como una fuente de definiciones beans (en contraposición del archivo XML)

```
import app.dominio.UsuarioDao;
```

```
@Configuration
```

```
public class AppConfig {
```

```
    // La anotación @Bean juega el mismo papel
```

```
    // que el elemento <bean/> en la configuración XML
```

```
    @Bean
```

```
    public UsuarioDao crearUsuarioDao() {
```

```
        return new UsuarioDao();
```

```
    }
```

```
}
```

Anotaciones @Configuration y @Bean

Lo definido anteriormente en la clase AppConfig, es equivalente al XML:

```
<beans ...>  
  <bean id="usuarioDao" class="app.dominio.UsuarioDao"/>  
</beans>
```

Anotaciones @Configuration y @Bean

Un ejemplo de cuando un bean tiene una dependencia de otro bean:

@Configuration

```
public class AppConfig {
```

@Bean

```
public TransferService transferService() {  
    return new TransferServiceImpl(accountRepository());  
}
```

@Bean

```
public AccountRepository accountRepository() {  
    return new InMemoryAccountRepository();  
}  
}
```

Anotaciones @Configuration y @Bean

Lo definido anteriormente en la clase AppConfig, es equivalente a decir:

```
<beans ...>
```

```
    <bean id="accountRepository"  
class="com.formacionbdi.account.repository.InMemoryAccountRep  
ository"/>
```

```
    <bean id="transferService"  
class="com.formacionbdi.account.service.TransferServiceImpl">  
        <property name="accountRepository" ref="accountRepository"/>  
    </bean>
```

```
</beans>
```

*@Component y
Anotaciones Relacionadas
(@Repository,
@Service,
@Controller)*





@Component, @Repository, @Service, @Controller

- *@Component es un estereotipo genérico para cualquier componente manejado por Spring*
- *@Repository, @Service, y @Controller son especializaciones de @Component para usos más específicos:*
 - ❖ *@Repository para persistencia*
 - ❖ *@Service para servicios y transacciones*
 - ❖ *@Controller para controladores MVC*

*Por qué usar @Repository,
@Service, @Controller
por sobre @Component?*

Podemos anotar nuestras clases de componentes con @Component, pero en vez de eso, si anotamos con @Repository, @Service, o @Controller, nuestras clases se adaptan más y mejor para cada caso en particular, por ejemplo en cómo son procesadas y cómo se relacionan con aspectos específicos de Spring (componentes AOP)

Anotación @Repository

Una clase anotada con "@Repository" tiene un mejor traducción y legible manejo de errores con `org.springframework.dao.DataAccessException`. Ideal para implementar componentes que acceden a los datos (DataAccessObject o DAO)



Anotación @Service

Una clase anotada con "@Service" juega un rol de servicios de lógica de negocio, ejemplo patrón Fachada para DAO Manager (Facade) y manejo de transacciones



Anotación @Controller?

Una clase anotada con "@Controller" juega un rol de controlador en una aplicación Spring Web MVC



Declaración component-scan



Declaración Escaneo de Componentes

El package especificado vía atributo "base-package" por ejemplo "com.formacionbdi.miaplicacion" será escaneado en búsqueda de cualquier tipo de clases anotadas con `@Component` o en su defecto con sus anotaciones estereotipadas (`@Service`, `@Repository`, `@Controller`), las cuales serán registradas como "beans de spring" dentro del contenedor, Spring Boot lo maneja de forma automática usando la anotación `@SpringBootApplication` en la clase principal.



`@SpringBootApplication` internamente incluye las anotaciones: `@Configuration`, `@EnableAutoConfiguration`, y `@ComponentScan`