Thursday, June 30, 2022 9:29 AM

Why we need configuration ?

To Inform container about beans and how to manage them

Three ways to do the configuration -

1. Give a list of beans in the xml
2. Give the list of beans in JavaConfig class
3. Give the package names and ask the container to scan those packages for annotated classes

---

What all instructions or information are we giving to the CONTEXT ?

1. Bean class name -package qualified class name
2. Which properties have to be injected and using which values or ref
3. Bean id
4. Scope of the bean ---default- singleton , prototype

Today

5. Whether the bean should be eagerly initialized or lazily initialized

When the scope is **prototype the initialization is LAZY**

On every **request** of getBean a new instance is created

When the scope is singleton , we can specify LAZY or EAGER

If eager -then the moment CONTEXT reads the xml or Java config immediately instance is required

If lazy - then the instance is created when first time getBean is called

Configuration

beans.xml	Java config	Annotated bean

<bean …. lazy-init="true"	@Bean
	@Lazy
	AddressBean getAddr()
	{
	}
		@Component
		@Lazy
		class BookBean
		{
		}

6. **Dependency Injection** --- make the container set the properties (dependency =properties) of the class
   1. Using setter methods

beans.xml	Java config	Annotated bean
<bean …. > <property name="hh" value="ab" /> <property name="pp" ref="otherbeanid" /> </bean>	@Bean @Lazy AddressBean getAddr() { AddressBean obj = **new AddressBean() ;** Obj.setCity()	@Component @Lazy class BookBean { Public **BookBean()**

	}	{
		setCost(...)
		}
		
		}

BeanFactory instantiates using **default constructor** Calls the setters of the properties specified		

2. Using some parameterized constructor

beans.xml	Java config	Annotated bean

<bean .... >
<constructor-arg name="hh" value="ab" />
<constructor-arg name="pp" ref="otherbeanid" />
</bean>
	@Bean
	@Lazy
	AddressBean getAddr()
	{
	AddressBean obj = **new AddressBean(....., ...) ;**
	
	Obj.setCity()
	
	}
		@Component
		@Lazy
		class BookBean
		{
		
		Public **BookBean()**
		{
		setCost(...)
		}
		
		}

BeanFactory instantiates using **default constructor** Calls the setters of the properties specified		@Autowired above the constructor to be used

---

Autowired annotation

1. This annotation can be applied on constructor / setter / property
2. When it is applied on constructor
   The container tries to inject the beans of the parameter types
3. When it is applied to setter
   the container tries to inject the property after the constructor is called , using that setter method
4. When it is applied on property
   the container tries to inject the property directly after the constructor is called , without using setter

In Autowired annotation it uses Autowire byType = container tries to match the class name of the property /parameter with the bean class name

In beans.xml we can Autowire byName or byType using **autowire** attribute !!!!

byType = tries to inject bean of the same class name , but if two beans of same class name are

present - AMBIGUITY

byName = tries to inject bean having same name as that of property name or parameter name
, but if no bean has the same name as property /parameter then null is given !!!
MATCH the property name and the bean name

DB connectivity using Spring ----------

1. Simple JDBC
2. JdbcTemplate Bean provided by Spring

*** We will use **java.sql. DataSource** to get the connection

*** DataSource is a CONNECTION FACTORY
It will produce **connection pools and maintain a few **pre created** connections for **QUICK** use
*** DataSource must be configured with connection information
1. Driver name
2. Jdbc url
3. Uname
4. password

Exercise --- create a new project

*** We will Java Config class to configure the DataSource !!!
*** We will use Annotated class to configure DAOBean !!!

1. Main Class
   a. Create Context ---------- AnnotationConfigApplicationContext
   b. getBean
   c. Call the methods in the bean

2. DAOBean ----- property of DataSource --- we will inject it using @Autowired on property
   Methods to insert a record in DB
   To get all records from the DB

Exercise --- In the above project
We already have DataSource Bean configured using JavaConfig
We will add one more bean JdbcTemplate in the JavaConfig

*** We write **DAOBeanWithJdbcTemplate**
---property ---- JdbcTemplate ---AutoWired

---methods to insert a row in the table
--- methods to get all rows in the table

MainClass
--- get the **DAOBeanWithJdbcTemplate**
call the methods
