28/08/22

Spring

Spring:- We can develop a java Application using API Technologies like JdBC, servlets, and JSP, where we need to collect all ~~thus~~ dependencies of these related API's

→ If we can develop an application using technologies where we need to write all the codes and it takes lots of time. In order to reduce the time and to develop an application and also to avoid the boilerplate code (unnessecary code) we can go for fremaworks like spring, spring mvc, (&) spring boot

→ Framework is a collection of utilities (API's)

→ spring is the best framework in order to develop is very fast and easily

~~→~~ spring provides dependency injection mechanism with the help of IOC container

IOC Container:-

IOC container is providing the required objects but o when we confire the object class details with the help Configuration file.

→ We use xml file as a configuration where the fil should be ended with .xml extention

→ In order to provide the object class details to the I container we use confiration file i.e., springexample.

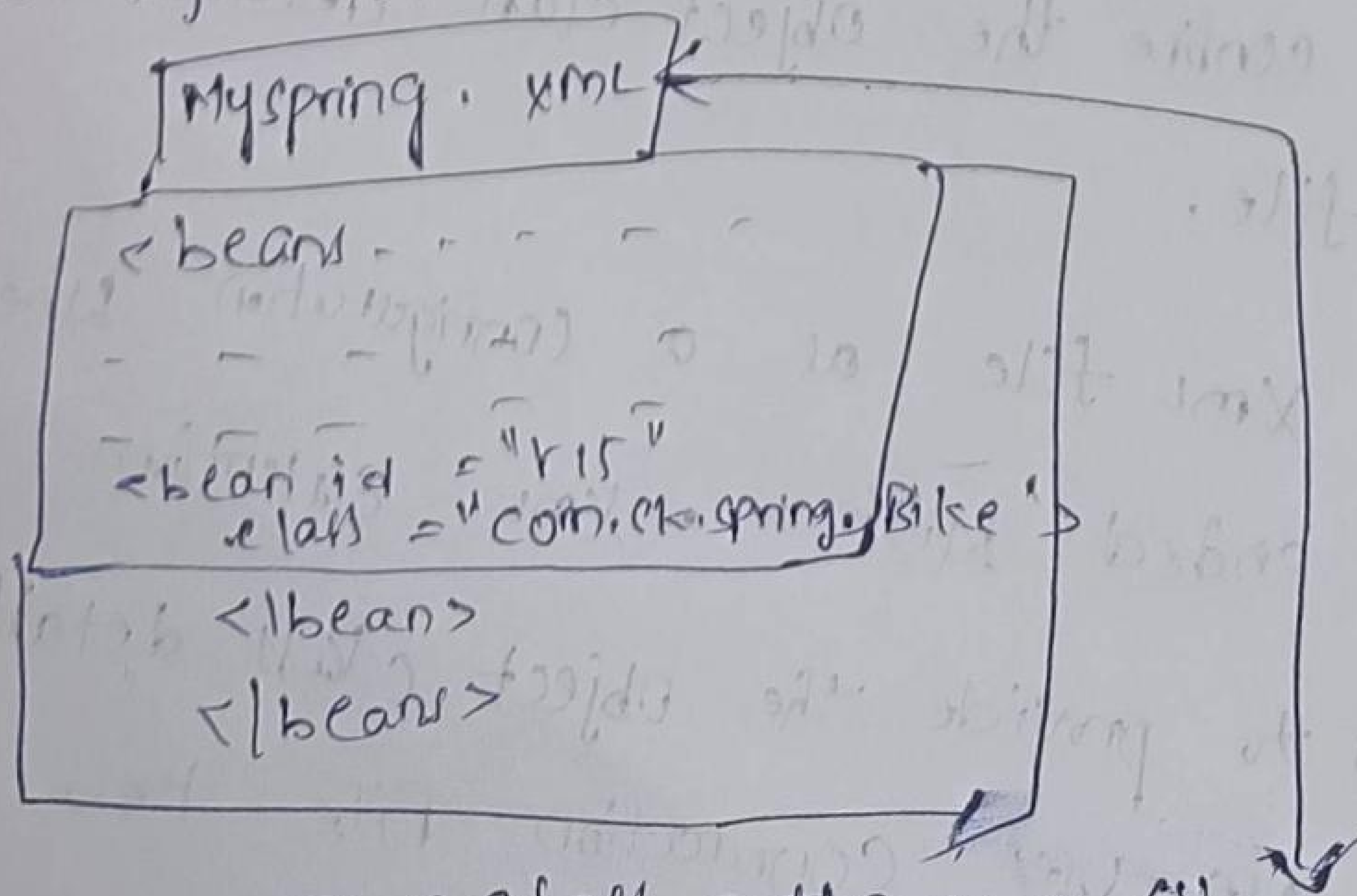
→ We have two types of Ioc's containers
They are
① Beanfactory

→① Beanfactory:-
Beanfactory is utilize only to develop smallscale Application

→② Applicationcontext :-
It is also Ioc's container where it is build on
the top of Beanfactory
→ so we can conclude that application context is the
Extended ~~[illegible]~~ version of Beanfactory.
→ In our words Applicationcontext is the Child of Beanfactory

Note:-
IOC → Inversion of control

Developing a QuickStart Application using Beanfactory
Ioc container :-
→ In order to provide the object class details to the
IOC container we are help of xml file
→ Beanfactory requires resource object to get the details
of the object class



```
Myspring.xml

<beans - - - -
 - - - - - -
 - - -
<bean id ="r15"
   class ="com.ck.spring.Bike">
<lbean>
<lbeans>
```

Resource r = new class pathResource("myspring.xml)

Resource:-

Resource is an Interface and its child class classpath Resource. is used to map the configuration file to the bean factory

→ the Resource object is standing b/w the configuration and the IOC's container i.e., Beanfactory

```
Resource r = new ClasspathResource("myspring.xml");
BeanFactory b = new XmlBeanfactory(r);   (reference of Resource)
Bike bike = (Bike) b.getBean("r15");
```

We are getting the object with the help of getbean ~~method~~ ("r15") which is a non-static

→ Note
We can perform constructorinjection and setterinjection with the help of xml file.

[mySpring.xml]

```
<Bean - - - -
- - - - - - >
<bean id = "car" class = " . . . . . . ">
<constructor-arg type = "int" value = "1"> </constructor-arg>
<constructor-arg value = "shashank" > </constructor-arg>
<bean>
</beans>
```

If the argument is string datatype no need of specifying datatype (type = "String")

Ex class car {
    int id;
    string name;

```
Car (int id , stringname){
 this .id = id;
 this. name = name;
 } }
```

→ In the pojo class i.e car class

POJO → plane Old Java object

In terms of <u>Hiberanate</u> <u>Framework</u> pojo called as <u>entity</u>

→ In terms of springframeworks poja class is called "<u>Bean</u>"

<u>setterInjection</u> :-

Spring. xml

```
<beans> - - - -
 - - - - - - )

<bean id ="car" class = "- - - - - ">
<property  type = "int" value = "1">
<property  value = "Shashank">
 </bean>
 </beans>
```

If the argument is spring datatype no need of specifying datatype (type = "string")

```
ex  class car
  {
   int id ;
  string name;
```

సాంఘిక మరియు గిరిజన సంక్షేమ శాఖల వారి ఉచిత సరఫరా

Scanned by TapScanner

```
public void setId (int id){
    this.id = id;
}
public int getId (){
    return id ;
}
public int setName(String name)
{  this.name = name;
}
public String getName(){
    return name;
}}
```

<u>Note</u> :-
In case of both constructor and set injections it the datatype is otherthan string such as primitives. We have to use type="primitive datatype".

<u>Note</u> :-
We <u>cannot</u> perform <u>field injection</u> directly with the help of spring xml configuration file.

→ We can overcome the xml files by using <u>java file</u>

→ For that We need to create a class <u>AppConfiguration</u> which is appropriate and mention <u>@ configuration</u> on the top of class.

```
@configuration
class AppConfiguration{
~~@configuration~~
}
```

→ the above class represents a java configuration file

→ this java configuration file is utilize to communicate with the IOC container i.e., Application context

→ But we have just specified the class as configuration file By using @ConfigurationAnnotation will not configure any off the java objects. Application container (IOC container) provides the java objects.

only when configure the object class details

```
@ configuration
@ component scan (base package = "com.annot")
class ApplicationConfiguration{

~~class~~ ~~fg~~
```

→ @ component scan is used to scan all the specified java classes which are present in specified package for base package = ("com.annot.spring.dto") this base package is configuration bigly with dto package

→ so will scan the java classes which belongs to dto package only

→ IOC container creates object of the classes which is having @ component annotation

@ component scan(base package = "com.annot")

@ component

→ component scan always we check for annotation like component, Controller, service ... etc.

Note:-

If We Want to scan all type of Annotation in each and Every package We need to follow a pattern Which Will be helpful to develop a project. As mention Below

Group Id : Com.spring

Artifield Id ; ~~Annot~~ Annot_Config

packageName : com.spring.Annot.Config

Always If We name the ~~pkg~~ packagename as combination of group id & Artritype-Id which is good pratice to diagogue.

package com.annot.spring.annot.config

@Configuration
@componentscan (base packages = "com.spring.annot.Config")

the Above Annotation component scan having base package of Which Will be scanning into, Common packagenamed Which Will be scanning into

Dao, service, package

com.spring.annot.config.configuration. Scan all packages
com.spring.annot.Config. Controller
com.spring.annot.config.dao;
com.spring.annot.config.dao.Impl;
com.spring.annot.config.service
com.spring.annot.config.service.Impl;
com.spring.annot.

com. spring. annot . config . dto ;

→ Com. spring . annot . config is the base for all packages ,

Annotation Based configuration :—

With the help Annotation and java configuration file we can perform constructor injection, setter injection and also fieldinjection is possible

Constructor injection with using Annotation :—

```
@ component
@
class car {
eng eng;
car(@ Autowired Eng eng)
{
this. eng = eng
} }
```

```
@component
class Eng {

}
```

constructor injection is by default can be performed without @Autowired

→ By performing constructor injection without @Autowired We get the final output

→ Among all types of Injections Constructor injection is bettern than other type of injections

సాంఘిక మరియు గిరిజన సంక్షేమ శాఖల వారి ఉచిత సరఫరా

Setter Injection :-

```
@ Component
Class Car {
Eng eng;
@ Autowired
public void setEng(Eng e) {
    this. eng = e;
} }
```

```
@ Component
class Eng
{
}
```

In case of setterinjection we don't use @Autowired We get nullpointerException

→ It is mandatry that we need to check whether @Autowired is used (or) not.

Field Injection :-

```
@ component
class Car {
@Autowired
Eng eng;
}
```

```
@component
class Eng {
}
```

In case of field injection we don't use @Autowired We get nullpointerException

→ It is mandatry to Check whether @Autowired is used (or) not.

In order to develop a project we might follow the abstraction layer by creating interface layer

Ex:- com. spring. annot - config . service;
  └> UserService (I)
com . spring . annot - config . service . impl ;
  └ UserserviceImpl (C)

→ usually we can create object directly but IOC container create Objects for us

→ we declare variable using superclass and its child class will be created.

@Autowired
UserService userservice ;

→ IOC container will inject the object to the user service i.e., its child class user service Impl

→ Incase if we user service have more than one child class we get unsatisfied dependency injection exception

```
interface userservice {
}
@component
class userserviceImpl {
  implements userservice
}
```

```
@component
class Userservice Impl2
implements Userservice
{
}
```

We need to use @primary at one of the child class

```
@ Componet
@ primary
class   UserserviceImpl2  implements Userservice {

}
```

→ @ Autowired
   Userservice userservice; // (UserserviceImpl2 is injected)

→ Each and every time we need to switch @ primary annotation b/w different class.

→ Instant we can go for @qualifier

Usage of @qualifier:—

```
interface Userservice
{
}
@ component
class UserserviceImpl implements
                    Userservice
{
}
```

```
@ component
class UserserviceImpl2
    implements Userservice
{
}
```

```
→ @ component
  class Demoservice
  {
  @ Autowired
  @ qualifier ( value = " UserserviceImpl ")
  Userservice userservice;

  }
```