More

Making the Second App as the Loosely Coupled Application

=> To change Engine Type of Vechicle class we need to modify the source code kept in @Qualifier annotation... this nothing but tight coupling

as

=> To make this code the loosely coupled code,we need to gather the bean id required for

the Dependent spring bean from the application.properties file .. (The bean required in the @Qualifier(-) annotation)

=> application.properties file is part of spring Boot App's Echo System i.e that file will be recognized

by Spring Boot Application automatically during its Bootstrap process ..So there is no need of configuring that

file seperately using @PropertySource Annotation

In application.propperties (src/main/resources folder)

engine.id=pEngg

 # any key = bean id

(one dependent class spring bean id)

In Vehicle.java (taget spring bean class)

```
@Autowired
//@Qualifier("pEngg") //---> Dependent Bean id is hardcoded, so not recomanded //bad
//@Qualifier("${engine.id}") //---> place holder is allowed only inside the @Value // invalid
//@Qualifier("@Value('${engine.id}')") //---> @Value is used as the indepndent annotation i.e it can not be used inside @Qualifier(-) annotation //invalid
@Value("${engine.id}") // injecting the dependent springbean id to String property
private String id;
//@Qualifier(id) // --> variables can not passed to the @Qualifier annotation private Engine engg;
//invalid
```

Since all above all attempts are failing, we need to use Spring bean cfg file(xml file) linked with @SpringBootApplication class using @Import Resource Annotation

"src/main/java", "src/main/resources","src/test/java" folders

of the MAVEN project are there in the Project build /classpath by default i.e we need not add them to CLASSPATH/BUILDPATH seperately

com.nt.cfgs

|---> applicationContext.xml (spring bean cfg file)

```
@SpringBootApplication
@ImportResource("com/nt/cfgs/applicationContext.xml")
public class BootProj02-SecondApp{
public static void main(String args[]){
```

application.properties

engine.id=pEngg

dEngg

(a)

applicationContext.xml

note: src/main/java, src/main/resources, src/test/java folders are in classpath by default in any maven/gradle Project

<!-- configure the application.proveries file --> <context:propertyplacehodlerconfigur

cation="classpath:application.properties"/>

not required

<!-- provide alias name dependent class bean id by collecting it properties file -->

<alias name="${engine.id}" alias="motor"/>

eg: classpath:application.properties means the maven /gradle project searches for the application.properties file in all folders that are added to the classpath like "src/main/java", "src/main/resources", "src/test/java"

pEngg dEngg

Alias name for the bean id

(b)

Vehicle.java

pEngg

no

As of now there is alternate annotation for <alias> tag.. So we are forced to use spring bean cfg file (xml) here for this concept

Second App With Loose Coupling

=======

@Autowired dEngg

@Qualifier("motor") private Engine engg;

Step1) Keep sperate copy of the Second Application

Step2) Take the sperate spring Bean cfg file in com.nt.cfgs folder having name the applicationContext.xml

Step3)

=> create com.nt.cfgs package in "src/main/java" folder

=> create xml file in the above package

note:: the application.properties/yml file of the spring boot project is part of the Project's eco system i.e it will be configured automatically in the project, So its data can be used

in both java code and xml files

right click on com.nt.cfgs package ---> new ---> other --> search xml -->select xml file-->

name :: applicationContext.xml ---> ... ---> ... --> finish

add the following code in applicationContext.xml file

<?xml version="1.0" encoding="UTF-8"?> <beans

xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

Collect this from sample Spring bean cfg file (applicationContext.xml file)

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<!-- configure the properites file

-->

<context:property-placeholder location="classpath:application.properties"/>

<!-- provide the alias name -->

<alias name="${engine.id}" alias="motor"/> </beans>

Place holder to read value from proeprties file by submitting the given key

step4) Add the dependent spring bean id to application.properties file

In Vechicle.java

@Component("vehicle") public class Vehicle {

@Autowired @Qualifier("motor")

private Engine engg;

step4) Link the applicationContext.xml file with main class

BootProj02-Depedency Injection-LooseCoupling [boot] src/main/java

*

#com.nt

BootProj02 DepedencyInjectionApplication.java

com.nt.cfgs

applicationContext.xml

com.nt.sbeans

@SpringBootApplication

Here we are linking spring bean cfg file (xml file)

↓ with @Configuration class

>

DCNGEngine.java

@ImportResource("com/nt/cfgs/applicationContext.xml") public class BootProj02DepedencyInjectionApplication {

>

Diesel Engine.java

> EletricEngine.java

>

Engine.java

}

>

PetrolEngine.java

>

step5) Run the Client App, change the bean id in application.properties file time to time

Q) What is Ambuiguity Problem in Dependency Managmenent operations and How can we solve that problem having 100% Loose coupling?

Vehicle.java

src/main/resources

application.properties

>

src/test/java

=> if the multiple dependent spring beans of same type are found to inject to single HAS-A property target Spring bean class then this Ambiguity Problem raises.. To solve this problem take the support @Qualifier(-) as the best solution. We can achieve 100% Loose Coupling solution in two ways

> JRE System Library [JavaSE-17]

>

Maven Dependencies

>

src

>

target

HELP.md

(a) Using properties file+ spring bean cfg fille + <alias> tag + @Qualifier

mynw

(b) Using profiles in spring/spring boot (Best Solution)

Another Example on Solving the Ambiguity Problem having 100% Loose Coupling

==:

===

BootIOCProj02-

Strategy DP [boot]

src/main/java

#com.nt

BootlocProj02StrategyDpApplication.java

✓ BootlocProj02StrategyDpApplication

✓com.nt.cfgs

main(String[]):void

applicationContext.xml

> # com.nt.sbeans

✓ src/main/resources

>

application.properties

src/test/java

> JRE System Library [JavaSE-17]

> Maven Dependencies

> src

>

target

HELP.md

mvnw

mvnw.cmd Mpom.xml

```
//ICricketBat.java (Common interface) package com.nt.sbeans;
public interface ICricketBat { public int scoreRuns();
}
//MRFBat.java package com.nt.sbeans;
import java.util.Random;
import org.springframework.stereotype.Component;
@Component("mrfBat")
```

```
public final class MRFBat implements ICricketBat {
public MRFBat() {
System.out.println("MRFBat:: O-param constructor");
}
@Override
public int scoreRuns() {
System.out.println("MRFBat.scoreRuns()");
int score=new Random().nextInt(200);
return score;
//SGBat.java
package com.nt.sbeans;
import java.util.Random;
import org.springframework.stereotype.Component;
@Component("sgBat")
public final class SGBat implements ICricketBat {
public SGBat() {
}
System.out.println("SGBat:: 0-param constructor");
@Override
public int scoreRuns() {
System.out.println("SGBat.scoreRuns()");
int score=new Random().nextInt(200);
//SSBat.java
 }
}
package com.nt.sbeans;
import java.util.Random;
import org.springframework.stereotype.Component;
@Component("ssBat")
public final class SSBat implements ICricketBat {
public SSBat() {
System.out.println("SSBat:: 0-param constructor");
}
@Override
//Cricketer.java (Target class)
package com.nt.sbeans;
import org.springframework.beans.factory.annotation.Autowired; import
```

```
org.springframework.beans.factory.annotation.Qualifier; import org.springframework.stereotype.Component;
@Component("cktr")
public final class Cricketer {
@Autowired @Qualifier("bat")
private ICricketBat bat;
public Cricketer() {
System.out.println("Cricketer:: 0-param constructor");
}
public String batting() {
System.out.println("Cricketer.batting()");
return score;
}
}
// Main class
package com.nt;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.Spring BootApplication; import
org.springframework.context.ApplicationContext; import
org.springframework.context.ConfigurableApplicationContext; import
org.springframework.context.annotation.ImportResource;
import com.nt.sbeans.Cricketer;
@SpringBootApplication
@ImportResource("com/nt/cfgs/applicationContext.xml")
public class BootIocProj02StrategyDpApplication {
public static void main(String[] args) {
public int scoreRuns() {
System.out.println("SSBat.scoreRuns()");
int score=new Random().nextInt(200);
return score;
}
}
application.properties
spring.application.name=BootIOCProj02-StrategyDP
# Cricketing Bat
bat.id=mrfBat
applicationContext.xml
//use the dependent
int runs-bat.scoreRuns();
```

return "Cricketer has scored "+runs+" runs ";

}

//get Access to IOC container

ApplicationContext ctx-SpringApplication.run(BootlocProj02StrategyDpApplication.class, args);

// get Target spring bean class obj ref

Cricketer cktr=ctx.getBean("cktr", Cricketer.class);

//invoke the b.method

String msg=cktr.batting();

System.out.println(msg);

//close the container

((ConfigurableApplicationContext) ctx).close();

}//main

}//class

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context" xsi:schemaLocation="

http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-

context.xsd">

<alias name="${bat.id}" alias="bat"/>

mvnw.cmd pom.xml

</beans>

Spring boot Layered Application Development

==========

(non Layered App problems) Limitations of keeping multiple logics in single java class (problem)

(Mini Project here)

=========

Important logics

b.logic/service logic:: main

(a) Clean seperation b/w logics will not there ...So all logics will be mixedup and

the code looks very clumzy

(b) The modifications done in one logics effects other logics.

(c) Enhacement and maintainance of the Project becomes very complex

(d) Parellel development is not possible.. So the productivity is poor

(e) It is not industry standard

(f) Debugging (Identifying the problem) becomes very complex

and etc..

in

logic of the application that deals with calculations, analyzations, Sorting, filtering and etc..

persistence logic :: the logic that performs persistence operations

on the DB s/w (CURD operations)

presentation logic :: the UI Logic that takes inputs from

=> keeping multiple logics in single java class is like staying bachelor room (single room for multiple activities) Solution (Develop the App as the layered Application)

==========

=> A Layer in the app is one or more classes/files collection representing certain category logics

eg:: presentation layer (UI logics), business/service layer(b.logics), persistence layer (DB opeations), controller layer (monitoring logics) and etc..

Typical standalone Layered App

====

Client App (presenation logic)

(UI logics)

[can be taken

(monitoring logics)

controller class

↓

[1 per Project]

|--->CURD operations

note:: A Layer can be a physical or logical Partiation of the Project having single class/file or collection classes or files representing same category of logics

---> service class ----------> DAO_class -

(b.logic)

end user and displays formatted results

to enduser

controller logics :: the logics that monitor and controls all the activities

Note:: In Layered App, we keep different logics in different Layers and make them talking with

each each other

=>A Layer can be a class/file or collection of classes or files representing same category logics

->DB s/w

(persistence logic)

(Data)

↓

↓

↓

[1 per module]

[as needed]

=> All layers are part of the same project, but we keep the classes/files of different Layers in different packages

[1 per db table]

as many]

ARE

=> Since we pack all these layers code either in single jar file or war file we can say Layered apps monolith /Monolithic architecture apps BY DEFAult

Bank Project

==========

=>Daily Transactions

=> Loans

=> **Payroll**

=> Recovery

=> Insurance

=> Deposits

and etc..

=> if we develop the above Project using Spring/Spring Boot

then we need to inject

=>A software Project is collection

-> DAO class obj to Service class obj

modules..

->Service class obj to Controller obj

-> Client App uses ctx.getBean(-) method

to get Controller class obj having

Dependency Injection Dependency lookup

@Autowired

of Modules

=> A Module is Collection of

Applications

Service, DAO objs as internal objs

Service ob

=> We generally develop the layered apps as

MVC Layered apps (M--Model, V--View

C--Controller)

=====JAva Bean vedios ====== https://www.youtube.com/watch?v=io2TciOcF2Y
https://www.youtube.com/watch?v=GOvak6awDrI&t=161s

=>JAva Bean is a java class that contains getter and setter methods

=> setter methods are useful to set/modify the data of bean properties (member variables)

=> getter methods are useful to read the data of bean properties

of

=> An Application is collection of programs => A program is collection of instructions

Caroller note:: Module is called Sub Section of the object Project

=> While sending more than 3 values between the classes of App/Project .. we try to send them in the form single object

called Java Bean class object.

OBJ

=> Since each Java bean class holds some data to carry .. So it is called model class becoz model means data..

=> we awlays use Java Bean as helper class in the Project /App development .. to pass multiple values from

class,

one class to antoher of different layers in the form of obejcts.

Standalone layered application

Inputs = sname, sadd, m1,m2,m3

service class calucalutes the total and avg

We can develop any app as the layered app i.e we can develope standalone app or web application or distributed app either as arch or micorService arch app having layers

monolith

note: Monolith Arch Project will be multiple layers packed into single jar file but all these layers are part of single project

In MicroServices Architecture, every module is separate Project having layers where each project will packed into jar file/war file. Later all these projects will integrated using third party tools

note:: Across the multiple projects or across the multiple layers of a project if u want to pass more than 3 values then we take support of Java bean calss object. i.e we combine multiple values to single Java bean calss obj and we send that object from one layer to another layer of the same project or different projects

model

class obj

model class obj

model class obj

SQL Query

Client App (presentation logic)

controller class

another model class

(Monitoring Logics)

obj

another model class obj

Service class (B.logics)

DAO class

DB s/w

another (persistence

model class

logics)

model class obj = Java bean class obj

obj output:: student registration success or failure

SQL Query results in form of ResultSet obj

(DB tables having data)

How many types of jdbc con objects are there?

Information

=>To send any java class obj data over the n/w, we must take that object as the Serializable object..

=> To make java object as the Serializable object, the class of the object must implement java.io.Serializable(1)

eg1:: passing student info from Client App to controller to service to DAO class

(with in a project

in the form of Student class obj(Java bean class obj) having multiple details So non Serializable Java Bean enough)

eg2:: PAssing credit card /debit card details from E-Commerce project to

Paypall project in the form of java Bean class obj (Project to Project over

the n/w.. So use Serializable

Java

Bean)

=> Java beans /Model classes are helper classes i.e they do not fall under any category or any layer

Advantages of the LAyered Application separation of

=====

(a) clean

(3/4 BHK Flat) logics is possible, So we can say code is not clumsy

(b) we can get code modularity and reusability

(c) The modifications done in Layer logics does not effect other layer logics

(d) The Project maintainance and enhancement becomes quite easy

(e) Parellel development is possible i.e productivity is good

Direct jdbc con object

=>created by the programmer

manually

eg: Class.forName("-"); //optional

Connection con=

DriverManager.getConnection(-,-,-)

(plain jdbc code)

pooled jdbc con object (Best)

=> It is the jdbc con object that is gathered

from jdbc con pool through DataSource obj

jdbccan pool

represents

DataSource obj

(ds)

Connection con=ds.getConnection();

(POOLED CONNECTION)

(f) It is industry standard architecture to develop the Projects and etc..

=>Both Monithlith architecture and microservices archicture projects can have Layeres

=> Min pool size, max pool size and other param can be configured

by the programmers while creating

the jdbc con pool

=> Jdbc con pool is a factory that contains set of readily avaiable jdbc con objects before

actually being used.

=>The advantages of jdbc con pool are

a)Gives the reusability of jdbc con objs

b)With minimum jdbc con objs, we can make

multiple requsts/client apps talking to Db s/w

c) creating new jdbc con objs, managing them, removing idle jdbc con objs

will be taken care by jdbc con pool itself.. So no burden on the programmer.

What is the jdbc con pool that u have used in ur Projects

Two types of jdbc con pools

?

Standalone

Server Managed jdbc

jdbc con pool

con pool

=>Suitable in standlone Apps

eg:: c3PO

hikariCP (best)

proxool

apache dbcp2

vibur

and etc..

important params of jdbc con pool

=>min size/iniitial size (10)

=> max size (1000)

=> connection timeout (Max time that pool waits to give con obj

(100 secs)

(60secs),

before throwing timeout error)

=> max idle time out (after this amout idle time the con obj will be destroyed)

=> shrink frequency (at regular intervals of shrink frequency the idle and etc.. (2000 secs) connection objs will be destroyed)

=>DataSource obj represents JDBC con pool i.e it acts as the entry and exit point

for the JDBC con pool i.e all the operations on the JDBC con pool will be done using the Datasource object

=> all the jdbc con objs in the jdbc con pool

represents the connectivity with same DB s/w

eg1:: jdbc con pool for oracle

(all jdbc con objs in that pool represents

connectivity with same oracle DB s/w)

=>suitable in web applications, distributed Apps, Enterprise Apps

that are deployable in the servers

=>Every web server /app server gives

built-in jdbc con pool support.. So better to use

that jdbc con pool in deployed apps (web applications, distributed apps)

eg:: Tomcat manged jdbc con pool

weblogic managed jdbc con pool

wildfly managed jdbc con pool

and etc..

App Server = web server++

=>Pool means set of same items, it gives reusability of same items

=> cache means set of different items, it gives reusability if different items

App

note:: Spring Boot gives hikaricp as the built-in jdbc con pool through Autoconfiguration

itad jdbc, jpa related starters to the build path of project

(spring-boot-starter-JDBC or spring-boot-starter-JPA)

|---> When these two or any one starter is added

then we get HikariCP Datasource object

pointing to the JDBC Con pool automatically

as spring bean through AutoConfiguration process

(So there is no need of taking @Bean methods)

of

In both Standalone Apps

and web applications

In spring,Spring boot Apps we can work with all types standalone jdbc con pools and all kinds of server managed

jdbc con pools.. But in spring boot Apps, we get hikari cp_based JDbc con pool support automatically through AutoConfiguration

process. (Becoz HikariDataSource obj becomes spring bean through AutoConfiguration process)

The above said standalone JDBC con pools and the server managed jdbc con pools can be used in both

spring /spring boot Apps and in non-spring/spring boot Apps

What is mostly used JDBC Connection pool /DataSource in our Spring/spring boot Apps?

Ans) Hikari CP represented HikariDataSource object becoz of its perfromence and

its ability to come as spring bean through AutoConfiguration in spring boot Apps

note:: Becoz of portability (ability to move the code) advantage we prefer using standalone jdbc con pool

like HikariCP in both standalone apps and web applications /Distributed apps

note:: In spring/spring boot apps, we inject Datasource obj (spring bean) to DAO class obj using

dependency Injection concept

What is the difference b/w pool and cache/buffer?

ans) Pool means set of same items i.e gives the reusability of

Po

same items cache/buffer means set of different items i.e gives the reusability

of different items

cache

=>Server Managed JDBC Con pool configurations will change server to server So we do not prefer this con pool even in the web applications/distributed apps that are deployable in the web servers/ App servers

Gives the

reusability of

same items

To implement this

use List Collection

eg:: jdbc con pool

13

Gives the reusability

of different items

To implement this use

map collection

eg:: Intenal ache of IOC container

=> All the con objs in the jdbc con pool represents connectivity with same DB s/w

eg: jdbc con pool for oracle means all the jdbc con objs in that jdbc con pool represents connectivity with same oracle Db s/w

eg: jdbc con pool for mysql means all the jdbc con objs in that jdbc con pool represents connectivity with same mysql Db s/w

eg: jdbc con pool for postgresql means all the jdbc con objs in that jdbc con pool represents connectivity

jdbc con pool for oracle

on

jdbc con pool for mysql

ton

with same postgreSQL DB s/w

jdbc.con pool for postgreSQL

con

DataSource obj

DataSource obj (on)

DataSource

obj

=> DataSource obj represents jdbc con pool i.e each and every operation jdbc con pool

can be done through DataSource object like getting access to exiting jdbc con objs, creating new jdbc con objs, managing the jdbc con objs,closing the jdbc con objs and etc.. =>DataSource obj means it is the object of a class that implements javax.sql.DataSource (1) directly or indirectly

=>In spring /Spring boot Apps we can take DataSource obj as spring bean and can be injected to DAO class obj

by using @Autowired annotation and DAO class uses that injected DataSource object to get jdbc con objs from

jdbc con pool in the methods that performs persistence operations

=> In spring boot Apps, the HikariDataSource obj will be created through AutoConfiguration process to JDBC con pool.. It get jdbc driver details from application.properties file to create those jdbc con objs

pointing

in the jdbc con pool