Problem in solving the ambiguity profem using @Qualifer(-) annotation

=> if u try to solve the ambiguity problem with 100% loosecoupling by the keeping the dependent bean id in the properites file and getting the bean id to @Qualifer(-) annotation we get the the following issuse-

Info.properties

 # Dependent bean id courier.id=dtdc

@Component("fpkt")

In target Flipkart.java

@PropertySource("com/nt/commons/Info.properties")

public final class Flipkart {

//HAS-A property

@Autowired //field Injection

//@Qualifier("${courier.id}") //invalid --- becoz we can not place the place holder ${<key>} in @Qualifier annotation-- allowed in

@Value annotation

//@Qualifier(@Value("${courier.id}")) -- //invalid ---beocz we can not place @Value annotation inside the @Qualifier annotation /*@Value("${courier.id}")

private String beanid;

@Qualifier("beanid")*/ // invalid becoz we can not pass variable name to @Qualifier as the bean id

private Courier courier;

Solving the ambiguity Problem using @Qulifier annotation + Spring bean cfg file + spring bean id alias name + properties file

(xml file)

=>we can provide alias name for the lengthy bean id.. but there is no annotation to do that

..but we have <alias> of the spring bean cfg file to do the same

eg1:

(xml file)

Spring bean cfg file (applicationContext.xml)

<alias name="dtdc" alias="shipment"/>

bean id

(alias name)

we can get bean id from properties file to spring bean cfg file inorder provide alias name for it

eg1:

Spring bean cfg file (applicationContext.xml)

<context:propertyPlaceholderConfigurer location=com/nt/coomoms/Info.properties"/>

(c1)

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

In target class

bean id collected properties file

fixed alias name

(a)

AppConfig.java

(d)

Info.properties

Dependent bean id courier.id=dtdc

@Component("fpkt")

public final class Flipkart { //HAS-A property

(gives the bean id collects

from the properties file through alias and

we can change that bean id form the properties

@Autowired //fieldjection file with out touching the java source code)

@Configuration

@ComponentScan (base Packages = "com.nt.comps")
@ImportResource("com/nt/cfgs/applicationContext.xml") public class AppConfig {

@Qualifier("shipment"

private Courier courier;

}

note: The properties file configured in java classes using @PropertySource can be used only in java classes... where as the the properties file configured in xml file (Spring bean cfg file) can be used in both Xml configurations and java classes (spring beans)

Example Application

step1) Keep any Stragety DP - Spring Application ready

step2) add properties file having ur choice dependent class spring bean id

info.properties (com/nt/commons pkg)

courier.id=dtdc

note:: To link spring bean cfg file with @Configuration class use @ImportResource_annotation

note:: To Link one @Configuration class with another @Configuration class take the support of @Import annotation

eg1::

@Configuration

@ImportResource("com/nt/cfgs/applicationContext.xml")

public class AppConfig{

...

step3) add one spring bean cfg file (xml file -- applicationContext.xml) to the com/nt/cfgs package of the application

applicationContext.xml

<?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"

eg2::

@Configuration

@Import(AppConfig1.class}

public class AppConfig{

...

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">

<!-- properties file configuration -->

<context:property-placeholder location="com/nt/commons/Info.properties"/>

<!-- provide alias name for the dependent spring bean id that is collected from properties file -->

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

(Importing the name spaces)

(Collect from the Internet)

https://docs.spring.io/spring-

framework/docs/4.2.x/spring-framework- reference/html/xsd-configuration.html

(search for spring-context)

bean id collected

(alias name)

</beans>

from the properties file

=> Xml name space is library that contains set of xml tags, To use tags of the namespaces we must import the namespaces These are like importing java packages in the Java app inorder to use the classes of that package in our application..

..

=>The popular xml namespaces in spring programming are

a) beans b) context c) util d) aop e) jee f) mvc and etc...

note:: For every namespace there will be one namespace uri and the xsd file (xml schema definitation file)

namespace

xsd file

XSD:: Xml Schema Definitation

url:: Universal Resource Locator uri:: Universal Resource Indicator

beans context

jee

namespace uri

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

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

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

step4) link the spring bean cfg file with Configuration class

In AppConfig.java

```
@Configuration
@ComponentScan(base Packages = "com.nt.comps")
@ImportResource("com/nt/cfgs/applicationContext.xml") public class AppConfig {
}
```

Linking the spring bean cfg file

with the Configuration class

spring-beans.xsd

spring-context.xsd spring-jee.xsd

step5) use @Qualifier (-) annotation in target class on the top of HAS-A property having the fixed alias name "shipment" whose dependent bean id will change based on the change done in the properties file

```
@Component("fpkt")
public final class Flipkart {
//HAS-A property
@Autowired //field Injection @Qualifier("shipment")
```

IOCProj11-StrategyDP-Spring-Solving the ambiguityProblem

JRE System Library [JavaSE-17]

src

✓com.nt.cfgs

>

applicationContext.xml

com.nt.client

> StrategyDPTest.java

com.nt.commons

com.nt.comps

› BlueDart.java

}

>

Courier.java

>

DTDC.java

step6) Run the Client App by changing the dependent class spring bean id from the properties file .. observe the change in the application's output

> > Flipkart.java

#com.nt.config > AppConfig.java

Referenced Libraries

ApplicationContext container

====

=====

ApplicationContext container = BeanFactory Container++

ApplicationContext

Container

BeanFactory

Container

Additional features of ApplicationContext container over BeanFactory container

==========

==========

a) pre-instantiation of singleton scope spring beans (already discussed)

b) Direct support to work with properties files and place holders ${<key>} (discussed)

c) Support for Internationalization (118n) (yet to discuss)

d) Support for Annotation driven programming (verified)

e) Support for 100% code driven(Java Config) progamming (verified)

is

f) Support for Event Handling (we can keep track of when the Container created, stopped or closed) (yet discuss)

g) Ability to close or stop, refresh the IOC container (verified)

h) Allows us to take multiple spring bean cfg files at a time (out dated)

and etc..

(xml files as the spring bean cfg files)

note:: In the BeanFactory Container, we need to give the extra inputs to IOC container to recognize and use the placer holders

where as in ApplicationContext container the place holder ${<key>} will be recognized and used automatically

I18n (Internationalization)

(I <18 letters> n)

=> The process of making our app working for different Locales is called 118n

=> Locale means language + country

eg: en-US (English as it speaks in USA)

fr-FR (french as it speaks in France)

Localization can be written as the L10n

de-DE (German as it speaks in Germany) fr-CA (French as it speaks in CANADA)

hi-IN (hindi as it speaks in India)

and etc..

of

to

=> 118n is no way related to B.logic the app.. it is purely related presentation logics

=> The 118n deals with

-> presentation labels

-> Number formats

=>By

-> Currency Symobls

-> Date formats

-> Time formats

and etc..

By enabling 118n on the application,

we can make more Locale customers coming and doing business with our Applications.. this leads increase the

revenue.

=> Java outputs can be rendered as the Unicode output

as part of

i.e we can use 65,535 (Uni Code Chars) the output generation

enabling 118n on the App or website we can make the that app or website used by different Locale people i.e we can increase the business volume.

eg::google.com, amazon website, banking apps, fb app, instagram app, google maps app and etc..

=> java.util.Local class obj represents one Locale

Locale l=new Locale("fr","FR"); Locale objec representing french Locale

=> Spring gives built-in support for 118n ... we have ctx.getMessage(-,-) to get Presentation label based on the specified Locale object data.

=>To enable 118n in spring app

step1) Take multiple properties files for multiple Locales on 1 per Locale Basis

=>In that one properties file must be taken as the base file having english labels

App.properties (base file) (English labels) App_fr_FR.properties (for french Locale) App_de_DE.properties (for German Locale) App_hi_IN.properties (for hindi Locale) App_te_IN.properties (for telugu Locale)

=> ctx.getBean(-,-) is given to get the spring bean class obj ref based on given bean id

=> ctx.getMessage(-,-) is given to get message from the properties based on the given key and given Locale object

information.

=>The base file name must be continued in the Locale specific properties file s

eg: <basename>_<locale code>.properties

=> In All properties files same keys must be placed but the values should be collected from goolge translator as per the locale.

App.properties

#presentation labels (english)-- base file

welcome.msg=Good Morning {0} goodbye.msg= See you Soon

application.title=flipkart.com

wish.message=Happy New Year 2024

App_fr_FR.properties

#presentation labels(french) welcome.msg=Bonjour {0} goodbye.msg= À bientôt

application.title=flipkart.com

wish.message=Bonne année 2024

App_de_DE.properties

App_hi_IN.properties

#presentation labels (german)

welcome.msg=Guten Morgen {0}

#presentation labels (hindi)

(15) Submits the key "welcome.msg" in this Locale specific properties file and gets Unicode chars based value (that is शुभ प्रभात

goodbye.msg= Bis bald

application.title=flipkart.com

wish.message=Frohes neues Jahr 2024

\u0917\u0947

welcome.msg=\u0936\u0941\u092D \u092A\u094d\u0930\u092d\u093e\u0924 {0}

goodbye.msg=\u091c\u0932\u094d\u0926 \u0939\u0940 \u0928\u093f\u0930 \u092E\u093f\u0932\u0947\u0902

application.title=flipkart.com

wish.message=\u0928\u092f\u093E \u0938\u093e\u0932 \u092E\u0941\u092c\u093E\u0930\u0915 \u0939\u094B
2024

App_te_IN.properties

#presentation labels (telugu)

welcome.msg=\u0C36\u0C41\u0C2D\u0C4B\u0c26\u0C2F\u0C02 {0}

**goodbye.msg=\u0C24\u0C4D\u0C35\u0C30\u0C32\u0C4B
\u0C15\u0C32\u0C41\u0C26\u0C4D\u0C26\u0C3E\u0C02**

application.title=flipkart.com

**wish.message=\u0C28\u0C42\u0C24\u0C28 \u0C38\u0C02\u0C35\u0C24\u0C4D\u0C38\u0C30
\u0C36\u0C41\u0C2D\u0C3E\u0C15\u0C3E\u0C02\u0C15\u0C4D\u0C37\u0C32\u0C41**

2024

Unicode chars representing telugu labels

note:: All these files must have .properties as the extension ...

note:: All these properties files must have same keys and different values.

step2) Configure ResourceBundleMessageSource class as the Spring Bean in

@Configuration class using @Bean method specifying the name and

location of the Base properties file

note:: ResourceBundlerMessageSource class gives instructions to IOC container to activate specific properties based on the Locale obj data that is given and to collect the messages from that locale specific properties file

note:: Since the IOC container internally calls ctx.getBean(-) having the fixed bean id

"messageSorce" to access and use ResourceBundleMessageSource class obj.. So we must Configure

this class as the spring bean using the same fixed bean id..

```
//AppConfig.java
(messageSource)
(7) Looks for
@Bean methods
in @Configuration
class and finds
1 method
package com.nt.config;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.support.ResourceBundleMessageSource;
@Configuration
(5) Loads this class and makes that class as the spring bean class
public class AppConfig {
by creating object for it
(6) No @ComponentScan annotation
is found
@Bean(name="messageSource") // messageSource is the fixed bean id
public ResourceBundleMessageSource createRBMS() {
ResourceBundleMessageSource source=new ResourceBundleMessageSource();
source.setBasename("com/nt/commons/App");
return source;
(8) IOC container searches for singleton scope spring bean
}
and finds only one spring bean as @Bean method .. So
it creates that spring bean class obj by calling the @Bean method
 }
//Client App
==========
//Client App
```

package com.nt.client; import java.util.Locale;

import java.util.Scanner;

AppConfig class obj(appConfig)

O

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.nt.config.AppConfig;

(1) Run the Application

public class Spring_118nTest {

(2) main method public static void main(String[] args) { //create IOC container (3) IOC container creation AnnotationConfigApplicationContext ctx=

new Annotation ConfigApplicationContext(AppConfig.class); //read language code and country code from the enduser

IOCProj10-118n-Spring src/main/java

com.nt.client

> D Spring_118nTest.java

com.nt.commons

App_de_DE.properties

App_fr_FR.properties

App_hi_IN.properties

App_te_IN.properties

App.properties

> AppConfig.java

>

>

#com.nt.config

src/test/java

JRE System Library [JavaSE-17]

<

Maven Dependencies

>

src

>

target

Mpom.xml

ResourceBundleMessageSource object

Q

(messageSource)

com/nt/commons/App

(9) keeps the singleton scope spring bean class objects

in the internal cache of the IOC container

messageSource

ResourceBundleMessageSource obj ref

appConfig AppConfig class obj ref

keys

values

Scanner sc=new Scanner(System.in);

System.out.println("Enter language code::");

(10)

String lang=sc.next(); hi

System.out.println("Enter country code::");

String country=sc.next(); IN

//prepare Locale obj having language code + country code

Locale locale=new Locale(lang,country);

(4) Takes the given class as the configuration class

// read the message from the Locale specific properties file using the ctx.getMessage(-) method

(16) String msg1=ctx.getMessage("welcome.msg", new Object[] {"raja"}, locale);

String msg2=ctx.getMessage("goodbye.msg", new Object[] {}, locale);

String msg3=ctx.getMessage("application.title", new Object[] {}, locale);

String msg4=ctx.getMessage("wish.message", new Object[] {}, locale);

//display the messages System.out.println(msg1+"....."+msg2+"....." ."+msg3+"....."+msg4);

//close the container

ctx.close();

(17)

(18) Container is closed and all the

objects will be vanished

(12) ctx.getMessage(-,-,-) method internally calls ctx.getBean(-,-) method having the fixed bean id "messageSource"

In that process it gets base properties file name that is "com/nt/commons/App" from the object

(13) collects the given key and given Locale object data from the from the ctx.getMessage(-,-,-) method

key ----> welcome. Locale object---> hi IN

msg

(14) Based On Locale object data "hi" "IN" it will pickup the properties

The Object[] of the ctx.getMessage(-) method can pass {0} to {4} arg values

file as <base name>_<lang code>_<country code> that is com/nt/commons/App_hi_IN.properties

Each Message can have max have 5 args {0} to {4}

<terminated> Spring_118nTest [Java Application]
E:\Softwares\Eclipse\eclipse-jee-2023-09-R-win32-x86_64\eclipse\plugin

Enter language code::

}

}

hi

Enter country code::

IN

शुभ प्रभात raja..... जल्द ही फिर मिलेंगे.....flipkart.com.....नया साल मुबारक हो 2024

<terminated> Spring_118nTest [Java Application]
E:\Softwares\Eclipse\eclipse-jee-2023-09-R-win32-x86_64\eclipse\plugins

Enter language code::

X

Enter country code::

y

Good Morning raja.....See you Soon.....flipkart.com.....Happy New Year 2024

What is the difference b/w ctx.getBean(-) method and _ctx.getMessage(-)method?

ans) ctx.getBean(-) method gives the spring bean class obj ref based on the given bean id where as ctx.getMessage(-) method reads the message of the given key from the properties file

that is activated based on the Locale object is that is given

note:: ctx.getMessage(-,-) internally calls ctx.getBean(-) method

method having the fixed bean id (messageSource)

In AbstractApplicationContext.java

```
public static final String MESSAGE_SOURCE_BEAN_NAME = "messageSource";
protected void initMessageSource() {
ConfigurableListableBeanFactory beanFactory = getBeanFactory();
if (beanFactory.containsLocalBean(MESSAGE_SOURCE_BEAN_NAME)) {
this.messageSource = beanFactory.getBean(MESSAGE_SOURCE_BEAN_NAME, MessageSource.class);
// Make MessageSource aware of parent MessageSource.
if (this.parent != null && this.messageSource instanceof HierarchicalMessageSource hms &&
hms.getParentMessageSource() == null) {
// Only set parent context as parent MessageSource if no parent MessageSource
// registered already.
hms.setParentMessageSource(getInternal ParentMessageSource());
```

```
}
if (logger.isTraceEnabled()) {
logger.trace("Using MessageSource ["+this.messageSource + "]");
}
}
else
{
// Use empty MessageSource to be able to accept getMessage calls.
DelegatingMessageSource dms = new DelegatingMessageSource();
dms.setParentMessageSource(getInternal ParentMessageSource());
this.messageSource = dms;
beanFactory.registerSingleton(MESSAGE_SOURCE_BEAN_NAME, this.messageSource);
if (logger.isTraceEnabled()) {
logger.trace("No "" + MESSAGE_SOURCE_BEAN_NAME + "" bean, using ["+this.messageSource + "]");
}
}
@Override
@Nullable
public String getMessage(String code, @Nullable Object[] args, @Nullable String defaultMessage, Locale
locale) {
return getMessageSource().getMessage(code, args, defaultMessage, locale);
}
```