Spring F/W

Spring Containes:

Application Context: new

BeneFactory: old

Dependency Injections:

spring container to create objects and providing data to variables.

Injection (I) / Dependency Injection(DI) :

Injection means “Provide data to variable (dependency)

. Setter Injection

2. Constructor Injection

3. LookUp method Injection

4. Interface Injection

1. Injection(I) / Dependency Injection (DI) :

Setter dependency injection (SDI) by using set method of variable container provides data. It uses default constructor and set method.

2. Constructor Injection(CI) or Constructor Dependency(DDI) :

Container provides data while creating object using “Parameter Constructor” It is called as (CI) / (CDI).

IOC and Spring Container:

Spring Container takes care of

 Creating Objects

 Providing data to object

 Link object to another objects

 Destroy Objects

 Spring Container needs two input from programmer.

1. Spring bean (Java Files / class )

2. Spring Config File (XML / JAVA / ANNOTATION)By Taking these inputs , container will create object with data . then programmer has to read objects , use or print then uding test class.

Spring Bean:

It is a class given by programmer which follows rules given by spring

container.

 If we follow those rules and write class , then container will accept our

class and create object to it , else class is rejected no object created.

1. Class must have package statement.

2. Class must be public type.

3. Class can have variable , if exist type must be private.

4. Class must have default constructor with set/get method for every

variable.

Or

Class can have parameterized constructor(Same time both also valid )

5. class can override method from object (java.lang) class given below.

toString() , hashCode() , equals()

6. class can have annotations , which are defined in spring API and also

core annotation ( an annotation in java.lang.package ) are allowed

7. class can extends / implements only spring API ( classes / inerfaces )

and one special interface is allowed i.e. java.io.Serializable(I).

Configuration:

XML:

XML Configuration (Basic Syntax)

<bean class = “\_\_\_\_\_” name = “\_\_\_\_\_\_\_”>

<property name = “\_\_\_\_\_\_\_\_\_”>

<value> \_\_\_\_\_\_\_\_ </value>

<property>

</bean>

. <bean> :: Indicate object , which will be created in Spring container.

2. <property> :: It will call set method of given variable to provide data.

3. <value> :: It indicates data to variable.

4. All above tags are case-sensetive must be written in lower-case only.

XML Example:

===========

Bean:

Package com.app;

public class Employee{

private int empId;

private String empName;

//Default Constructor

//Set , Get

//toString..

}}

2. Spring Configuration file (\_\_\_\_.xml)

<bean class = “com.app.Employee” name = “emp”>

<property name = “empName”>

<value>A</value>

<property>

</bean>

Value can be represented in 3 syntaxes those are:

1. Value as tag:

<bean class = “com.app.Employee” name = “emp”>

<property name = “empId”>

<value> 4 </value>

</property>

<property name = “empName”>

<value> ABC </value>

</property>

</bean>

2. Value as attribute:

<bean class = “com.app.Employee” name = “emp”>

<property name = “empId” value = “4” />

<property name = “empName” value = “ABC”/>

</bean>

3. P-namespace / p-schema

<bean class = “com.app.Employee” name = “emp”

P:empId = “88”

P:empName = “ABC” />

Spring Container test Application:

Here we should write logic to read object (bean) from container and

print it or use it.

 To indicate Spring container , framework has given interface(I)

BeanFactory (old container ) and ApplicationContext (I) (new container).

 We should use any one implementation class of ApplicationContext (I)

few are:

 ClassPathXmlApplication( C ) =====Xml Configu

 FileSystemXmlApplicationContext( C )

 XmlWebApplicationContext( C )

 AnnotationConfigApplicationContext( C )=====Java Config

 AbstractApplicationContext( C ) etc..

For both java and annotation configuration we should use spring

container class (Context) given as AnnotationConfigApplicationContext (

C ) (org.springframework.context.annotation) it takes java config class as

input to create container with objects

JAVA CONFIGURATION:

Spring has provided two basic annotation for java configuration those

are :

@Configuration (org.springframework.context.annotation)

@Bean (org.springframework.context.annotation

1. Write one public class with any name.

2. Apply @Configuration annotation at class level.

3. Define one method per one object which returns same class type of

requested object method name behaves as object name by default.

4. Apply @Bean on method so that spring container create object

\*\*\* If method is written in java configuration but @Bean is not applied

then container will not create object

Sytax:

@Configuration

Public class AppConfig{

//no of method = no of objects

@Bean

Public Type objName(){

//create object & set Data..

Return obj;

}

}

REF-TYPE CHILD AS INTERFAC:

HAS-A relation (ref type) container create child object first and then parent.

 But if child type is interface then we should choose any one of its implementation classs and create object to it first. Then link with parent object.

To load Properties using

@value(${}

@PropertySource({"MyProp.properties"})

In spring injection (providing data to variable ) is optional , to make it required use annotation @Required over set method of that variable.

\*\* Making one dependency injection is required is called dependency check.

\*\*Activate annotation using

<context:annotation-config/>

Stand Alone Collection (SAC)

Creating one collection Object in Spring Container without using <bean> tag or outside to all <bean> tags is known as SAC.

1. Every SAC is independent.

2. SACs are re-usable (Create one time link with multiple <bean>s).

3. SACs can be created using our own specific Implementation class (for

interface List, Set, Map only

4. \*\*\* SACs can be created using util-schema

Ex:

<util:listlist-class="java.util.LinkedList"id="lstObj">

<value>10</value>

<value>vicky</value>

<value>5.5</value>

</util:list>

<util:mapmap-class="java.util.LinkedHashMap"id="mapObj">

<entry>

<key>

<value>10</value>

</key>

<value>AA</value>

</entry>

<entrykey="20">

<value>BB</value>

</entry>

<entryvalue="CC">

<key>

<value>30</value>

</key>

</entry>

<entrykey="40"value="DD"/>

</util:map>

Bean Scope In Spring:

how long bean (Object) should be in Spring Container

Possible Bean Scope Are Given as:

1. Singleton (default scope):

It is only default scope given to every <bean> in Spring Container. It indicates one object is created by container when it is started.Maintains same object end container is destroyed.

\*\* One object per one configuration.

2. Prototype

It create new object in Spring container on every access by Application / Programmer.

3. Request

Container creates new object for every request , same maintained until response is commited. Works only in web application (using servlet) in spring.

4. Session

Container creates new object for every new session it is maintained until session invalidated. Work only in web application (using servlet) in spring.

5. Global context/session

It works in portlets based web application. It creates one object for all portlet access.

Types of Annotations for configuration:

1.>StereoType Annotations (Bean creation):

 An annotation which detects the class and creates the object is known as StereoType Annotation.

a) Activate Annotation with base-package.

b) Provide @Component on top of the class.

\*\*\* code to activate Annotations:

<context:component-sacn base-package=”com.app”/>

<context :component-scan base-package=”----“/>

2>Data Annotations (Injection)

1. StereoType Annotation:

There are 5 types , list given as:

@Component

@Controller

@Service

@Repository

@RestController

2. Data Annotation

@Value

@Autowired (for scope concept : @Scope)

2. @Component:

It must be applied on class (not applicable for interface and abstract class). It will inform Container to create object of current class

If no object name is provided then class name (first letter small) is taken as object name (camel-case-conversion).

@Scope:

@Component

@Scope(“prototype”)

Public class Employee { }

@Value (Basic data Annotation):

This annotation is used to inject data to bean (object) in case of Annotation configuration.

It is applicable for Primitive Type, Collection and Reference Type.

Constructor Dependency Injection (CDI)/Constructor Injection (CI) :

Spring container uses parameterized constructor to create object and to provide data.

To Specify one parameter tag is “<constructor-arg></constructor-arg>”.

 This one never going to depend on default constructor and setters and getters.

 Writing bean tag without <constructor-arg> indicates using default constructor

Ex: no <constructor-arg> tag

<bean class=”com.app.bean.Employee”name=”emp”></bean>

Means: Employee emp = new Empoyee();

 If no constructor is written in class then Java Compiler will add default constructor.

<!-- Value As Tag

<bean class = "com.app.core.Employee" name =

"empObj">

<constructor-arg>

<value>10</value>

</constructor-arg>

</bean> -->

<!-- Value as attribute

<bean class = "com.app.core.Employee" name =

"empObj">

<constructor-arg value = "10"/>

<constructor-arg value = "vickyraj"/>

</bean> -->

<!-- Value As C-Schema -->

<beanclass="com.app.core.Employee"name="empObj"

c:empId="10"

c:empName="Vicky Raj"/>

</beans>

<!-- SAC Data List and Map -->

<util:listlist-class="java.util.LinkedList"id="lstObj">

<value>10</value>

<value>20</value>

</util:list>

<util:mapmap-class="java.util.LinkedHashMap"id="mapObj">

<entry>

<key>

<value>10</value>

</key>

<value>AAA</value>

</entry>

<entrykey="20"value="BBB"/>

</util:map>

<!-- Ref As Attribute -->

<bean class = "com.app.core.Employee" name = "empObj">

<constructor-arg value = "10"/>

<constructor-arg value = "VickyRaj"/>

<constructor-arg ref = "addrObj"/>

</bean>

<!--Ref As Using C-Schema / C:Name-Space -->

<!—-

<bean class = "com.app.core.Employee" name = "empObj"

c:empId = "10"

c:empName = "VickyRaj"

c:addr-ref = "addrObj"/>

-->

Wiring:

It is a process of linking parent object with child object. By writing<ref> tag code written by programmer.

Autowiring:

Parent-child object are linked by Spring Container only. Programmer not required to write <property with <ref (SI code) or <constructor-arg with <ref (CI code).

 Autowiring is applicable for Reference Type Dependency Only.

 Autowiring can be done in two way

1. XML 2. Annotation

byType:

This one compares <bean class=”” in XML with DataType of HAS-A variable in parent class.

 If matched then spring container will inject child into parent object.

Sytax:

at parent <bean> tag level

<bean class=”” name=”” autowired=”byType”

Matching Result Table

No.of Matching Result

=0 Null

=1 Injected

=2 NoUniqueBeanDefEx

 byType uses setter Injection.

 byType means DataType

 byType case container generates code like: <property… with <ref..

Example xml:

CASE#1: (Zero Child Bean Matching Found)

<beanclass="com.app.core.Employee"name="empObj"

p:empId="10"

autowire="byType"/>

OUTPUT: Employee [empId=10, addr=null]

CASE#2: (One Child Bean Matching Found)

<beanclass="com.app.core.Address"name="addr"

p:loc="patna"/>

<beanclass="com.app.core.Employee"name="empObj"

p:empId="10"

autowire="byType"/>

OUTPUT: Employee [empId=10, addr=Address

[loc=patna]]

CASE#3 (Multiple Bean Matching Bean Found)

<beanclass="com.app.core.Address"name="addr"

p:loc="patna"/>

<beanclass="com.app.core.Address"name="addr1"

p:loc="patna"/>

<beanclass="com.app.core.Employee"name="empObj"

p:empId="10"

autowire="byType"/>

)

OUTPUT:

org.springframework.beans.factory.NoUniqueBeanDefinitionException:

byName Autowiring:

At parent <bean> tag level provide autowire=”byname” then Spring Container compares HAS-A “variable name” with <bean name=” “ in XML.

 If both are matched then spring container injects child with parent.

CASE#1: (For Zero Bean Matching Found )

<beanclass="com.app.core.Employee"name="empObj"

p:empId="10"

autowire="byName"/>

OUTPUT:Employee [empId=10, addr=null]

CASE#2: (For One Bean Matching Found)

<beanclass="com.app.core.Address"name="addr"

p:loc="hyd"/>

<beanclass="com.app.core.Employee"name="empObj"

p:empId="10"

autowire="byName"/>

OUTPUT: Employee [empId=10, addr=Address [loc=hyd]]

CASE#3: (For Multiple Matching Found)

<beanclass="com.app.core.Address"name="addr"

p:loc="patna"/>

<beanclass="com.app.core.Address"name="addr"

p:loc="patna"/>

<beanclass="com.app.core.Employee"name="empObj"

p:empId="10"

autowire="byName"/>

OUTPUT:

org.springframework.beans.factory.parsing.BeanDefinitionParsingException:

Constructor Autowiring:

If we provide autowire=”constructor” at parent bean tag level then Spring container links child and parent objects using const. Dependency Injection

(CDI) (It will call param. const.)

<bean class”com.app.Address” name=”a1” p:addrId=”99”/>

<bean class=”com.app.Employee” name=empObj” autowire=”constructor”/>

Qualifier:

CASE#3: (One Matching Found [work like byType] )

<beanclass="com.app.core.Address"name="aob"

p:loc="hyd"/>

<context:annotation-config/>

<beanclass="com.app.core.Employee"name="empObj"

p:empId="10"/>

OUTPUT: Employee [empId=10, addr=Address [loc=hyd]]

CASE#4:

(Multiple Child Bean Found And One <bean name = “” matched)

<beanclass="com.app.core.Address" name="aob"p:loc="hyd"/>

<beanclass="com.app.core.Address"name="addr"

p:loc="hyd"/>

<context:annotation-config/>

<beanclass="com.app.core.Employee"name="empObj"

p:empId="10"/>

OUPUT:Employee [empId=10, addr=Address [loc=hyd]]

(Choosing One Child Bean From Multiple By Programmer [@Qualifier])

Bean CODE

@Qualifier("aob")

private Address addr;

XML:

<beanclass="com.app.core.Address" name="aob"

p:loc="hyd"/>

<beanclass="com.app.core.Address"name="addr"

p:loc="hyd"/>

<context:annotation-config/>

<beanclass="com.app.core.Employee"name="empObj"

p:empId="10"/>

OUTPUT:Employee [empId=10, addr=Address [loc=hyd]

1. What is the Life Cycle of a Bean?

A Spring bean needs to be instantiated when the container starts, based on Java or XML bean definition. The framework may also be required to perform some pre and post-initialization steps to get the bean into a usable state.

After that, when the bean is no longer required, it will be removed from the IoC container. Like the initialization phase, the Spring framework may need to perform pre-and post-destruction steps to free the other system resources.

InitializingBean and DisposableBean Interfaces

Ex:

public class DemoBean implements InitializingBean, DisposableBean

{

//Other bean attributes and methods

@Override

public void afterPropertiesSet() throws Exception

{

//Bean initialization code

}

@Override

public void destroy() throws Exception

{

//Bean destruction code

}

}

Custom init() and destroy() Methods:

Config:

<beans>

<bean id="demoBean" class="com.howtodoinjava.task.DemoBean"

init-method="customInit"

destroy-method="customDestroy"></bean>

</beans>

public class DemoBean

{

public void customInit()

{

System.out.println("Method customInit() invoked...");

}

public void customDestroy()

{

System.out.println("Method customDestroy() invoked...");

}

}

@PostConstruct and @PreDestroy Annotations:

public class DemoBean

{

@PostConstruct

public void customInit()

{

System.out.println("Method customInit() invoked...");

}

@PreDestroy

public void customDestroy()

{

System.out.println("Method customDestroy() invoked...");

}

}

@Qualifier vs @Primary with Examples:

public interface University {

public String display();

}

@Service

public class AlphaUniversity implements University {

@Override

public String display() {

return "This is a message from Alpha University";

}

}

@Service

public class BetaUniversity implements University {

@Override

public String display() {

return "This is a message from Beta University";

}

}

@RestController

public class UniversityController {

@Autowired

private University AlphaUniversity;

@GetMapping ("/university")

public String getUniversity() {

return AlphaUniversity.display();

}

}

Now when we call “/univeristy” API which dependency will be actually called? will it be AlphaUniversity or BetaUniversity?

Error

We will get an error like above mentioned to use either Primary or Qualifier annotation.

Springboot is unsure which dependency to use because both AlphaUniversity and BetaUniversity have implemented University.

Now let us use @Primary annotation for resolving this issue,

@Service

@Primary

public class BetaUniversity implements University {

@Override

public String display() {

return "This is a message from Beta University";

}

}

Qualifier:

@Service("Alpha")

public class AlphaUniversity implements University {

@Override

public String display() {

return "This is a message from Alpha University";

}

}

@Service("Beta")

@Primary

public class BetaUniversity implements University {

@Override

public String display() {

return "This is a message from Beta University";

}

}

@RestController

public class UniversityController {

@Autowired

@Qualifier("Alpha")

private University AlphaUniversity;

@GetMapping ("/university")

public String getUniversity() {

return AlphaUniversity.display();

}

}

Output: Alpha

Which is of higher priority @Primary or @Qualifier?

@Qualifier has higher priority than @Primary annotation.

Can we declare both @Primary and @Qualifier together?

Yes, Both annotations can be used together but only one can be executed at a time.

BeanFactory:

implemation:

XMLBeanFactory implements BeanFactory

annotation does not support,

instantiation :Benafactory instantiate beans when getBean() method gets called.

Event Publication: Bean factory does not have the ability to push events to the beans

Loading Mechanism: Lazy loading

Application context:

implemation:FileSystemXmlApplicationContext, ClassPathXmlApplicationContext and AnnotationConfigWebApplicationContex implements ApplicationContext.

annotation does support,

instantiation:Application instainte bean at the time container start.

Event Publication: Application Context have ability to push to push events to the beans.

Loading Mechanism: eagar loading

SPRING ORM:

concept used to perform DB operations in OOPS style.

Mapping concept between java class and DB table.

 Mapping is provided

:: ClassName must be mapped with table name.

:: VariableName must be mapped with column name.

 Mapping can be done using

a. XML coding (old style)

b. Annotation coding (new style)

(Also called as JPA = java persistency API annotation)

 If we follow mapping , then ORM converts objects to row and row to

object without SQL written to programmer

 Table must have one primary key (behaves as unique + not null)

Hibernate has been integrated with JPA for model class mapping with DB

table. All these annotation are provided in package.

Javax.persistency(JPA)

@Entity: this annotation must be applied over class . It maps class with table , variables with columns

@Id: Every table must have one primary key

DataSource:

This interface is used to indicate connection object between java app (JDBC / Hibernate / Spring ) and DB.

We need to configure (<bean> / @Bean) for any one implementation class of this interface

DriverManagerDataSource: to handle single connection object for application which is used for small application

Session factory (I ) [org.hebernate ] :-

 Loading driver class

 Supporting for creating connection

 Creating statements

 Execute operations

 Handle memory for result storing and conversion.

 Spring has provided one impl class using factory bean < session factory > format.

TransactionManagement in spring ORM:

Before performing any DB operation we should start (begin) Transaction (tx)

 If operation is successfully executed then commit (Save changes in DB) else rollback (Cancle changes in database).

 To do this tx management spring has provided API (for ORM)

b. Enable Tx: @EnableTransactionManagement

It will activate and de-activate txmanagement in spring application. It behaves

like switch.

c. Service method annotation.

in service layer method apply annotation like @Transactional for non-select

operation and also for select operation.

@Transactional (readOnly = true)

SPRING WEB MVC:

ViewResolver:

“InternalResourceViewResolver” is a class which can be used to provide prefix(location) and suffix(extension) of UI files.

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"

p:prefix="/WEB-INF/views/"

p:suffix=".jsp"/>

MVC + FC:

 It is a combination design pattern used in dynamic web application development here Front Controller is a servlet. It will received request (HttpServlet) and dispatched to one controller (class) based on URL.

 Controller execute logic and it may communicate to DB and fetch data which will be stored in model memory.

 This model memory will be shared with view (Display Code).

 Data will be placed in views file (Data Rendering) and finally view returned back to FrontController.

 Front Controller sent this as HttpResponse

Handler Mapper:

 Here DispatcherServlet cannot search all controller classes and their method so , FC creates one register which holds details like “For WHAT request , WHICH method ? “ will be executed.

 All methods and their URLs will be listed out and create as a map , known as HandlerMapper

1) Sending data from controller to UI (JSP).

Use Model and View (ModelAndView) (Model Memory) to store data from controller to UI(view) page.

ModelAndView will store data in key = value format key is String type and value is object (java.lang) type [value can store any data type , so super type object].

use ModelAndView object method i.e. addObject(String key , String value);to add data to Model memory

Read data at UI (View) using El (Expression Language) or Scriptlet request (implicit Object) Object method getAttribute(“key”) : Object

2) Sending data from UI To Controller :

a) HTML Form (ModelAttribure):

This concept is used to send data from UI to controller one complete

HTML form in single read an object.

 Here Spring container converts from data to object format on click submit after entering data in below steps.

1. Container create the object to model class [here object name is Class Name , first letter small case.

2. Container read data from HTML from inputs [request.getParameter..].

3. Container parse data if required.

4. Finally set data to ModelAttribute

 For this object creation , programmer has to do below steps

1) Write one model class with any name

2) Write variable in class

No of variable in class = no of form input (in html form)

4) Finally read this object in controller class using below code:

Syntax

@ModelAttribute(“classname”)ClassName localVarName

Example

@ModelAttibute(“employee”) Employee emp

b) Query Parameter (Request Parameters)

Html form is used to send large data (multiple value) to send few values (Just like one or two inputs) use query parameters concept given by servlets API also supported by spring web mvc framework.

 Data will be sent along with URL in key = value format.

 Here both (key , value) are String type by default.

 This data is given as input as input to (request) method in controller.

 To read this syntax is:

@RequstParam(“key”)DataType localVariableName;

http://localhost:8089/mvc/show?sid=20

 Equal servlet meaning is:

String sid = request.getParameter(“sid”)

Int id = Integer.parseInt(sid);

ModelAndView:

 It is a shared memory between controller and UI (view) pages.

 It must be used as method return type in controller.

 It creates two memory parts those are view (controller) and models.

 Same times , controller is not sharing data with view , in this case also

memory will be allocated to model even not used by application

[memory wasted].

 To avoid this performance degrade use new concept like.

1) ModelMap for model.

2) String for view name

Spring Security:

JAAS: ( Java Authentication and Authorization Service )

 It will secure URL’s in 2 level

a. User Identity [un , pwd]

b. Role verification

Authentication :

username , password

Authorization :

It will security work on login and role management of application using JAAS.

 Security is provided to URL using managers.

) Authentication Manager: Store the data in RAM on DB.

InMemoryAuthentication---RAM

JdbcAuthentication: Storing details (un ,pwd , role) in DB using JDBC.

UserDetailsService:Storing details (un ,pwd , role) in DB using ORM

b) AuthorizationManager:

1)permitAll: everyone can access no login & no role required

2. hasAuthority

URL can be accessed by users who must login and should have expected role. If login/role failed cannot access URL.

3. authentication

URL can be accessed by users who must login only. Role check not require

Level 1: Enable Security Filter

Level 2: Configure Authentication / Authorization manager.

Step 1:write one web application using WEB-MVC with multiple URL method.

Step 2:write on spring config file to provide authentication and authorization manager details.

EX:SpringSecurity which should extends class WebSecurityConfigurerAdapter

and Override 2 method.

Step 3:use any password encoder for securing password

a. NoOperationPwdEncoder

b. BCryptPasswordEncoder. (Binary Cryptography) etc.

Step 4:Enable security filter by writing one class. That extends

“AbstractSecurityWebApplicationInitializer”

2. Use security level method like:

.permitAll() :: every one can access.

.hasAuthority(“r1”)::only given role(R1) can view this after login.

.authenticated() ::indicates only login , no role required.

.hasAnyAuthority(“r1” , “r2” , “r3”) ::user should have any role in given

list and can view after.

3. anyRequest() ::it is used to provide all URL’s which are not specified in

configuration.

Ex:IN URL’s only /emp , /admin provided in config , to indicate

remaining 198 URL’s use anyRequest()

1. Every URL can be accessed by everyone.

Ans) .anyRequest().permitAll()

2. /emp can be accessed by ADMIN or EMPLOYEE roles after login.

Ans) .antMatchers(“/emp”).hasAnyAuthority(“ADMIN” , “EMPLOYEE”)

3. /home can be accessed by everyone.

Ans) .antMatchers(“/home).permitAll()

4. All product operations are accessed by Employee only

[/product/view , /product/get , /product/edit , /product/save]

Ans) .antMatchers(“/product\*\*”).hasAuthority(“Employee”)

Or

.antMatchers(“/product/view” , “/product/get” , “/product/edit” ,

“/product/save”

 Spring provide default login form without writing (JSP / HTML) code by

programmer.

.and().formLogin()

 To specify “after login , go to default URL “ code is:

.and().formLogin().defaultSuccessUrl(“/view”)

To specify logout URL Pattern

.and().logout.logoutRequestMatcher

( newAntPathRequestMatcher(“/logout”))

 To specify access denied error page.

.and().exceptionHandling().accessDeniedPage(“/denied”)

1. InMemoryAuthentication:

It will store data in RAM , it is used only for testing process , if DB is not

installed in system. This is best way to test application.

 At runtime we cannot create new user.

 Again stop server , modify code and start.

 Stores data in RAM , on every re-start of server memory deleted and

created again

) JDBCAuthentication:-- It will store data in DB tables(2) table-1 stores user

details and table-2 stores authorities.

=>Communicate to DB table using Spring JDBC uses DataSource (javax.sql)

interface with two special SQL queries.

SQL#1>Load user by username.

SQL#2>Load Authorities by username.

3) Spring Security Using ORM:-

Data inserted into two tables, user basic details are inserted into parent table and roles (Authorities) are inserted into child table.

=>WebSecurityConfigurerAdaptor checks given input data and Spring F/W user data, if matched then goto Success URL (Home Page) else goto Login Page with Error message.

SPRING AOP:

AOP (Aspect Oriented Programming):-

It is used for “Cross-cutting-concern ” . It means separate business logic

and external services.

 External service must behave as plug-in-code, that is without modifying exited application, programmer, should able to add/remove external services Example are :- Log4j,UnitTest,security,JMS,Crytography,Encode

and decode request/response ,filter management, request identity

process, etc……………………….

AOP Terms:

1. Aspect:- It is a class, which indicates external services logic.

2. Advice:- It is a method inside Aspect (class). It is also called as implementation of Aspect.

3. Pointcut: It is an expression which select the business class method to connect with advice. But it will not tell which advice it is

4. Joinpoint:- It is a combination of Advice and Pointcut expression . It means “joinpoint says which business class method need what and how many advice.

5. Target :- It is a pure business class object (before adding/without external services logic).

6. Weaving :- It is process done by weaver (sub component of spring container ).It will add advice logic to target based on join points.

7. Proxy:- It is a final output of weaving which contains business class logic and selected advices logic.

1. @RestController :- [Spring 4.x]

It is a 5th StereoType Annotation ie which detect the class and creates the object in Spring Container.It must be applied on class level.

It internally follows @Controller and @RestController

@Controller-------return model veiw of UI.

@RestController------->JSON/XML

3. @RequestMapping :-

It is used to provide path(URL) at class / method level. class level it is

optional.

# SPRING REST CONSUMER APPLICATION:-

 Use RestTemplate to make HTTP Request calls to provider application from consumer application.

 Template is a Design Pattern used to reduce common lines of code duplicate code/ boiler plate code...).

RestTemplate takes care of :

>Creating client objects

>web resources object

> Default HTTP methods with Header

> Making call to Provider

> Auto conversion of response to ResponseEntity<T>

MediaType:-data representation, language data (object) can be convert to global format and reverse using MediaType Annotations.

ResponseBody: Autoapplied by RestController for every method return type.