****
propagation.required

For example, if a read-only transaction calls a read-write transaction method, the whole transaction will be
read-only

****

1> what if i need to rollback Checked Exception in spring ?
ans: @Transactional(rollbackFor = Exception.class)
So if you throw an Exception or a subclass of it, always use the above with the @Transactional annotation
to tell Spring to roll back transactions if a checked exception occurs.

2> what if i need to can't wanna to rollback unckecked type Exception?
ans: @Transactional(norollbackFor = RuntimeException.class)

3>. what are the various transactions attribute?

ans: propagation.required :- Propagation.REQUIRED will create a new transaction (if none exists for the
current
thread), or will join an existing transaction (if one exists).

propagation.required new:- Create a new transaction, and suspend the current transaction if one exists.

propagation.supported:- if there's a transaction then use current transaction or there's no transaction then
dont create transaction.

propagation.not supported :- if there is a transaction then , dont use current transaction and if
there is no
transaction then dont create transaction.

propagation.mandatory :- use current transaction or if there is no current transaction then throw
exception.

propagation.never:- if there is a transaction then , it throw exception and if no transaction available then
dont create new transaction.

Difference between request and prototype bean scope in spring?

Prototype scope creates a new instance everytime getBean is invoked on the ApplicationContext.
Whereas for Request scope, only one instance is created for an HttpRequest. So in a HttpRequest,
if the getBean method is called twice on Application and there will be only one bean instantiated and reused,
whereas the bean scoped to Prototype in that same single HttpRequest would get 2 different instances.

4.> how @transactional is enable?

When using the @Transactional annotation in Spring, you must add the following line to your Spring
configuration file:
<tx:annotation-driven transaction-manager="transactionManager"/>
The transaction-manager property holds a reference to the transaction manager bean defined in the Spring
configuration file.
This code tells Spring to use the @Transaction annotation when applying the transaction interceptor. Without
it,
the @Transactional annotation is ignored, resulting in no transaction being used in your code.

5.>   what is Spring and its version?

spring is light weight container and its a factory of all kind of beans and version we used 4.3.5.

6>how to configure bean through xml and also with annotation?

with xml we used bean tag(<bean name ="user" class=".class">) bydeafult its scope is singleton

********and if we wanna to change its scope then in bean tag we used scope attribute .

with annotation we used @autowired

** bydeafult its search through (byType)

6> . types of container ?

1. beanfactory container.(interface)

2. applicationcontext container extends beanfactory .(interface)(child of beanfactory interface)

7. which architeture u used >

layered architeture ?

All layers are independent from each others. This is the advantage, If you change one
layer then it will not
impact other layers. In other words each layer is independently replaceable.

#> how many layer or component have in layered architeture?

4 layers

1. presentation layer(spring MVC / jstl/tiles/jsp/servlet)
2. business layer(Spring Aop / service beans)
3. data access layer (spring ORM/Dao)
4. integration layer(Spring mail)

#>/Presentation layer:- Presentation layer contains View and Controller components. View contains
presentation logic and Control contains navigation (or control) logic.

Presentation layer is implemented using Spring MVC, Tiles and JSTL frameworks.

Views are implemented by JSPs. JSP takes data from Model object and render using JSTL tags. Controllers are
implemented by Spring MVC Controller components. Controller communicates with Service classes to perform
business operations. Controller classes are defined in Spring MVC using @Controller annotation.

This layer implements following Design Patterns:
Command
Builder
DTO
Front Control

command design pattern:-

# command design pattern:- ctl for navigation logic

#Bulder design pattern:-Email bulder main aur tiles main

# DTO design patten:- dto class main or pojo class

#front Controler design pattern:- authorozation and authenication of user.

#>/ Business layer:-

Business layer performs business transactions of the Application. It contains business logic and
communicates
with Data Access and Integration layer. It does transaction handling.

Business layer is implemented by Spring Service classes. Service classes are defined in Spring using
@Service
annotations and declarative transactions are managed by @Transactional annotation. Spring AOP is used to
implement declarative transactions.

This layer implements following Design Patterns:
Session Facade
Bridge
DTO

#> Session facade:- service pe single entry pointcut:- serciveint

# bridge :- one interface multiple implementation

#DTO

#>/Data Access Layer
Data Access Layer communicates with database using JDBC or ORM frameworks. Hibernate ORM framework is
used in
this application to communicate with database.-o

Data Access Objects (DAOs) are implemented to perform database CRUD (Create, Read, Update and Delete)
operations.

This layer is implemented using Spring ORM and Spring DAO. Spring ORM is used to integrate Hibernate ORM
and Spring DAO is used to implement consistence data access exception hierarchy.

DAOs are defined in Spring using @Repository annotation.

This layer implements following Design Patterns:

DAO

A DAO design pattern helps an application to perform various CRUD operations on the database.

Bridge
DTO

#>/Integration Layer
The Integration layer provides application a way to integrate with other applications, servers and components.
Server could be EMAIL, SMS and Other Web Services.

Spring Mail are used at this layer to integrate Email Server.

#>/Architecture Components
1. JSP
It is implemented by JSP and JSTL tags. It contains presentation logic and render the HTML view

2. Controller class
It is implemented Spring MVC controller. It contains navigation logic. It depend on Service class that is
autowired by IoC container.

3. Service class
It is implemented by Spring Service classes with the help of @Service annotation. It contains business logic.
It handles transactions. When an checked exception is propagated in a method then transaction is committed by
 Service class. It depend on DAO class that is autowired by IoC container.

3. DAO class
It is implemented by Spring DAO classes with the help of @Repository annotation.It contains data access logic.

It depend on Datasource that is autowired by IoC container.

8.way to inject dependecy?
Constructor
Setter

9.how to tell spring ioc container to search for annotation?
It will search for annotated classes

```
<context:component-scan base-package="in.co.sunrays.proj0.*">
```

10.scope of beans?

1.Session,
2.request,
3.prototype,
4.singleton,
5.globalSession

11. what if i need to do by name autowiring ?
i have to used @qulifier annotation

#> @autowired(required=false)?????????????

12. btw beanfactory vs applicationcontext?
BeanFactory: Does not support the Annotation based dependency Injection.

ApplicationContext: Support Annotation based dependency Injection. -
@Autowired, @PreDestroy

BeanFactory: Does not Support
ApplicationContext: Application contexts can publish events to beans that are registered as listeners

BeanFactory: Does not support way to access Message Bundle(internationalization (I18N)

ApplicationContext: Support internationalization (I18N) messages.

BeanFactory: Doesn’t support.
ApplicationContext: Support many enterprise services such JNDI access, EJB integration, remoting.

BeanFactory: By default its support Lazy loading
ApplicationContext: It's By default support Aggresive loading.

13. injection provided by container at runtime or compile time?
ans runtime pe

#> scopes of beans ?

There are 5 types of bean supported in spring

singleton - Only one instance of the bean will be created for each container. This is the default scope for
a bean.

prototype – Return a new bean instance each time when requested

request – Return a single bean instance per HTTP request.

session – Return a single bean instance per HTTP session.

globalSession – Return a single bean instance per global HTTP session.

#> default scope of bean in Spring?
singleton is default scope of a bean in Spring. You have to explicitly change scope of a bean if you want
different scope.

14. life cycle of beans?

15. modes of autowirg

Modes of Auto wiring

Auto wiring is done as per mode values passed to it. There are multiple modes of auto-wiring:

1. autowire="no" : auto-wiring is disabled. Bean references must be defined in the XML file via the
<ref/> element (or "ref"
attribute). This value is set by default.

2. autowire="byName" : auto-wiring is done by property name, setter method is used for dependency injection.

3. autowire="byType" : auto wiring is done by property data type. If data type of property is matched
with a bean then bean is
auto wired. Bean is auto wired if there is exactly one bean of the property type in the container. If
more than one beans are
found in container then a fatal error is raised.

4. autowire="constructor" : Just like "byType" but applied on constructor arguments. If there is not
exactly one bean of the
constructor argument type in the bean factory, a fatal error is raised.

Configuration of dependency using <property> and <constructor-arg> tags will override auto-wiring.

#>/ Excluding a bean from auto-wiring

autowire-candidate="false"

#>/ If we have applied both @Autowired and xml configuration file autowiring then which one will be applied?

xml configuration overide annotattion besd dependencies.

16. @autowired is applied on setter , construtor, attribute, bydeafult it reslove by bytype

17. dao are defined with ?

18 spring throw which excpetions

19. hibernate intergate kese kiya?9/duispacther sefvlet se full name dekh lena kauns use kiya hai)

20. how to create session in hibertnate ?(getcurrentsession)

21. Service class are definec with?

22. @transaction handling Kese kri ?

23. did u used aop and where?

24. what is aspect , joitnpoint , advice, pointcut?

25. email sending k loye kya kiya

26. email bena in deatils ?

javamailsender (interface)

javamailsenderImpl(class)

mimemessage (class)
mimemessagehelper

27. mailsender.send()

28.
==

Presentation Layer
Implemented by:
Spring MVC
JSP/Servlet
Spring MVC Tags
Search this site

Implements View and Navigation Logics
Uses @Controller annotation

Business Layer
Implemented by Service classes
Service classes are defined by @Service
Service classes handles transactions
Transactions are handled by Spring AOP

Data Access Layer
Uses Spring DAO for generic exception conversion DataAccessException
@Repository annotation is used to implement DAO objects
Uses Spring ORM to integrate Hibernate
Hibernate to implement Data Access Objects

Integration Layer
Beans are used to integrate Java Mail

The @Component annotation has three specialized annotations @
Controller, @Service and @Repository.

#$>? @Autowired annotation

It is used to inject bean dependencies.

It can be applied on attribute, setter method and constructor.

It identifies and resolves bean dependencies using bean type or bean name. @Qualifier annotation is used
along with it to
identify a bean by name.

By default it identifies bean by type and injects dependencies

When applied in constructor then it always resolves dependencies by type

@Autowired with (required=false) is used to optionally inject a dependency.

#>/ Configure dispatcher server in web.xml
DispatcherServlet is mapped in web.xml to handle Spring URLs.

#>/
```
<!--Scan @Repository, @Service, @Component and @Controller spring
beans -->
<context:component-scan base-package="in.co.sunrays.spring" />

<!--Enables MVC annotation like @Valid, @RequestMapping etc. -->
<mvc:annotation-driven />


<!-- enable the configuration of transactional behavior based on
annotations -->
<tx:annotation-driven transaction-manager="hibernateTransactionManager"
/>
```

#>/ The <context:component-scan...>
tag will be used to activate Spring MVC annotation scanning capability which allows to make use of annotations like @Controller and
@RequestMapping etc. Sometimes it is also called
auto-discovery.

#<>? @Repository annotation
is a specialization of the @Component annotation with similar use and functionality. In addition to register

the DAOs
into the IOC container, it converts ORM and JDBC specific exceptions into generic Spring DataAccessException
exception hierarchy.

@#<>/ @Service
@Service annotation is also a specialization of the component annotation. It doesn’t currently provide any
additional
behavior over the @Component annotation, but it’s a good idea to use @Service over @Component in service-layer
classes because it specifies intent better.

[5] @RequestMapping
Maps URL path to a Controller class or its methods.

[6] @RequestParam:
Binds request parameters to method parameters.

[7] @ModelAttribute()
Binds object to a Model.

[8] @Valid
Applies validation to a Form

29 spring mvc ?

30 mvc flow?

Following is the sequence of events corresponding to an incoming HTTP request to DispatcherServlet:
1) After receiving an HTTP request, DispatcherServlet consults the HandlerMapping to call the appropriate
Controller.
2) Controller is called with the help of URL mapped to GET or POST request.
3) Controller calls object of Service class to perform required business operation. After finishing services
call,
4) Controller stores data into model and returns View name to DispatcherServlet.
5) The DispatcherServlet will take help from ViewResolver to pickup the defined view for the request.
6) Once view is finalized, The DispatcherServlet passes the model data to the view which is finally rendered
on
the browser. View takes data from model to display at view page.
All the above mentioned components ie. HandlerMapping, Controller and ViewResolver are parts of
WebApplicationContext which is an extension of the plain ApplicationContext with some extra features

necessary for
web applications.

31. model?

Model is a map object that contains data in form of key and value pair.
Model carries data from Controller
to View.
View gets data from Model using keys and displays them using JSTL and
other Spring tags

#>/ differnce b/t model and modelview?

model and view :- class ka object return karta h.
model:- string type return

32. to enable mvc annotation?

33. auto discover tag?

34. controle ris created with ?

34 frontctl created with ?

adapter ka naam kya hai??

35. how to bind method and class with url ?
The @RequestMapping annotation is used to map a URL to either an entire
class
or a particular handler method.

36.@controller:-

Controllers are defined with help of @Controller annotation. The
@Controller annotation indicates that a
 particular
class serves the role of a controller.

37> There are two ways to apply transactions:
XML Configuration
@Transactional annotation

38> I18N:-
1. Create Message resource files
Create message resource files, one for each language in the root class
path. These files are suffixed by
2 character
language code say "en" for English, "hi" for Hindi, and "sp" for Spanish.

2. configuration:-

2.1:- We need to declare these files in spring configuration file.
We will use class
org.springframework.context.support.ReloadableResourceBundleMessageSource

to define the message resources.

```
<bean
id="messageSource"
class="org.springframework.context.support.ReloadableResourceBundleMessag
eSource
<property name="basename" value="classpath:messages" />
<property name="defaultEncoding" value="UTF-8" />
</bean>
```

2.2 :-
Now we will provide a feature where user will be able to select language for the application. This is
implemented using
org.springframework.web.servlet.i18n.LocaleChangeInterceptor class.
The LocaleChangeInterceptor class will intercept request and make i18n, L10n changes.

```
<bean id="localeChangeInterceptor"
class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
<property name="paramName" value="lang" />
</bean>
```

User can pass request parameter ?lang=hi or ?lang=en to change the Locale and in turn his regional language

2.3:-User's selected Locale is saved in cookies for future subsequent requests.
Class org.springframework.web.servlet.i18n.CookieLocaleResolver is used to store the locale changes in cookies.

39> Get Locale Messages in the Controller

You may also need Locale messages in a Controller when you want to sent some notification messages to View
from
Controller in case of Business Validation fail or Success operation.

You may do it by

********Auto wring MessageSource object and pass Locale parameter in controller method. Here is the
example of LocaleCtl.java that gets a message for resource and set into Model object.

40>/ Interceptors
An Interceptors is the object that intercepts an incoming request and performs operation before sending
request to the
target controller and perform some operations after receiving response from target controller.

You can create an Interceptor by two ways
1. Implement HandlerInterceptor interface

2. Inherit HandlerInterceptorAdapter abstract class.
HandlerInterceptorAdapter
implements HandlerInterceptor interface.

This HandlerInterceptor declares three methods:
1. preHandle(..) is called before the actual target controller is executed
2. postHandle(..) is called after the target controller is executed. and
3. afterCompletion(..) is called after the complete request has executed.

The preHandle(..) method returns a boolean value. You can use this method to break or continue the
processing of
the interceptor execution chain. When this method returns true, the handler execution chain will continue;
when it
returns false, the DispatcherServlet stops further chain execution and transfers control to the desired page.

*** Interceptor will be configured in configuration file with the help of <mvc:interceptor> tag.

Interceptor FrontCtlInt will intercept all incoming requests since it is mapped to "/*" URI
Pattern.

```
<mvc:interceptors>
<mvc:interceptor>
<mvc:mapping path="/*" />
<bean class="in.co.sunrays.ctl.FrontCtlInt" />
</mvc:interceptor>
```

41>/ Request parameters can be bound with method parameters or a Form bean object in a Controller.

Controller uses

1. annotation @RequestParam to bind request parameters with
method arguments.
2. annotation @ModelAttribute to bind request parameters to a
bean object and store bean into Model object.

Method Parameter Binding

Annotation @RequestParam to bind request parameters with method arguments.
Here is example code that binds a "message" request parameters to method parameter "message".
This is equivalent to String message = request.getParameter("message");

```
@RequestMapping(value = "/param", method = RequestMethod.GET)
public String requestParam(@RequestParam String message) {
System.out.println("Request Parameter : " + message);
return "UrlMappingView";
}
```

=============

JavaMailSender interface

Class JavaMailSenderImpl
java.lang.Object

org.springframework.mail.javamail.JavaMailSenderImpl

All Implemented Interfaces:
JavaMailSender, MailSender

What's going on here with Exception?

Let me clear this up for you.
@Transactional only rolls back transactions for unchecked exceptions.
For checked exceptions and their subclasses, it commits data. So although an exception is raised here,
because it's a checked exception, Spring ignores it and commits the data to the database, making the system
inconsistent.

The Spring documentation says:

While the EJB default behavior is for the EJB container to automatically roll back the transaction on a
system exception
(usually a runtime exception), EJB CMT does not roll back the transaction automatically on an application
exception
(that is, a checked exception other than java.rmi.RemoteException).

--> While the Spring default behavior for declarative transaction management follows EJB convention
(roll back is automatic only on unchecked exceptions), it is often useful to customize this.
Pay attention to the last line: "it is often useful to customize this."
So how can we customize it?

It's very simple, just use the following with @Transactional:

@Transactional(rollbackFor = Exception.class)
So if you throw an Exception or a subclass of it, always use the above with the @Transactional
annotation to tell Spring to roll back transactions if a checked exception occurs.

Do not use @Transactional annotation methods with private, protected or default modifiers.

Prelaod :-

Pre loaded data the data that is displayed at View HTML Forms irrespective of any error. Generally drop-down
lists

are considered as pre-loaded data.
Annotation @ModelAttribute can be used on a method of controller. When @ModelAttribute is applied on a
 method of
controller then this method will be invoked before any method mapped with @RequestMapping annotation.

Binding:-

Controller uses
1. annotation @RequestParam to bind request parameters with method arguments.

2. annotation @ModelAttribute to bind request parameters to a
bean object and store bean into Model object.

Optional Parameter
By default request parameter is required. If it is not sent in request then you will get an exception.
 To make it optional
you can set "required" attribute of @RequestParam to false.
@RequestParam(value="id", required=false)).

HTML form biding in View
Spring tag f:form and commandName="loginForm" is used to bind model attribute "loginForm" to HTML form
elements.

how many tag libabry u imported in jsp page

form tag libabry
<%@taglib uri="http://www.springframework.org/tags/form" prefix="form"%>

jointpoint:- @tx
advice kaha hui :- asking for ki kya krna hai rollback or commits
pointcut :- kaha krna hai jese ki beofr in b/w after

Dao factory: ioc container hi ek tarah se dao factory h bcoz ioc container is created object of dao.

frontctl ki bean ka scope:- singlton

prototype design pattern:-

expert classes :- al custom classes are expert class.

eancapsulation

polymorphism :- brifge design pattern.

inheritance:- extends ki h classes.