**Ioc: have 2 containers**

Ioc container is responsible for:

1. will create instances(objects) for pojo classes.
2. Will manage life cycle for pojo classes
3. Into pojoclasses Can do DI

**1)core container: bean factory**

If the instance for any class, will be having a **spring**.**xml** file in that will have <**beans dtd or xsd**(for validating xml file)> </beans> , there will be SAX parser in IOC while loading sml file SAX parser will checks well formed file or not that is open tags should be close and **dtd** and xsd will check weather this tags have definition or not. And **SAX** parser will read the xml file.

So both container while loading will checks the xml documents and read the xml documents. When the user request the instance will be formed so when user req getbean(“”) methos the core container will create one onstance and when another user req it will not create other instance because in xml will declare **scope** as a **singleton**. If u declare scope as a **prototype** it will create each instance for calling each user req. by **default** the instance(object) are **singleton**.

**CORETEST:**

**TEST.JAVA:**

**package** bean;

**public** **class** Test {

//object will create when class access constructore.

**public** Test(){

System.***out***.println("the cons");

}

}

**Spring.xml:**

<! DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN" "http"//www.springframework.org/dtd/spring-beans-2.0.dtd"

<beans>

//it will create 1 instance bydeafult it will singleton if u write prototype it will each instance for each call

<bean id=*"t"* class=*"beans.Test"* scpoe=*"singleton"*></bean>

</bean>

**Client.java:**

**import** javax.annotation.Resource;

**public** **class** Client {

**public** **static** **void** main(String[] args){

//search a rsourse

Resource resource = **new** ClassPathResouce("resource/spring.xml");

//after finding resource load into bean

BeanFactory factory = **new** XmlBeanFactory(resource);

//it will check xml documnt whather the all tags are closed and weel formed

System.***out***.println("document valid");

//create test class object

factory.getbean("t");//but it will create 1 instance beacuse of sibleton.

factory.getbean("t");//add prototype it will create mutliple instance.

factory.getbean("t");

}

}

**2) j2ee container : applicationcontext**

The main difference in application context is while loading only the application context container will create instance. But if u declare prototype it will create instance while the user req not on loading, which is based on user req while called getbean)”’) method. For prototype each user req will crate each instance. **So when scope is prototype the application context is same s beanfactory**.

**Test.java:**

**public** **class** Test {

//object will create when class access constructore.

**public** Test(){

System.***out***.println("the cons");

}

}

Spring.xml:

<! DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN" "http"//www.springframework.org/dtd/spring-beans-2.0.dtd"

<beans>

//it will create 1 instance bydeafult it will singleton if u write prototype it will each instance for each call

<bean id=*"t"* class=*"beans.Test"* scpoe=*"singleton"*></bean>

</bean>

**Client.java:**

**public** **class** Client {

**public** **static** **void** main(String[] args){

ApplicationContext ap = **new** ClassPathApplicationContent("resource/spring.xml");//check xml doc

//byusing class.forname("test").newInstance(); it will create instance if the test class public but if the class is private it will acess because of spring ioc.

//it will create instance to single bean while loading only.

//if u run this it shows loading and will call test class also.

//scope in spring.xml then then just it will validat enot create any instance just like beanfactory

//if u writ eprototype write this below

ap.getbean("t");//now it will create instance as based on user req

ap.getbean("t");

ap.getbean("t");

}

}

**DEPENDENCY INJECTION:**

Is a software design pattern that implements the inversion of control which resolves the dependencies. Dependency is an object which is used and the injection is the passing of a dependency to an dependent object that would be use it. The service is part of the client state.

**SPRING SECURITY**:

Is the framework which is used to authorize and authenticate the spring application. Which gives security to your java web application which is developed using spring framework.

Is the security services applicable to java based web applications build on spring framework. Gives comprehensive services to j2ee based enterprose applications.

**AUTENTICATE:** is that which asks the user to enter valid credentials to login the page.

**AUTHORIZATION**: will validate what role or privilege does the login user has for this application. Based on what user will allows to access the different controls or functionalities over the application.

To enable the spring security you need to add some libraries in xml file and also in web xml write the filter and filter mapping like filtername and filterclass and url pattern same as servlet and servlet mapping.

CREATE XML FILE WRITE ALL LIBRARIES:

<bean>

</bean>

In WEB.XML

<!-- Enable Spring Security -->

<filter>

<filter-name>springSecurityFilterChain</filter-name>

<filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>

</filter>

<filter-mapping>

<filter-name>springSecurityFilterChain&lt;/filter-name>

<url-pattern>/</url-pattern>

</filter-mapping>

The **springSecurityFilterChain**filter i.e the **DelegatingFilterProxy**servlet delegates the request to a set of filters and interceptors defined in your spring security configuration in the application context. Such as authentication, authorization via the defined interceptors.

**SPRING TRANSACTION:**

Is a sequence of action that r treated as a sindle unit. It is the important part of RDMS oriented enterprise application to ensure data integrity. It consists of 4 proprerties:’

1)**ATOMICITY**: a transaction shold be single unit either it will successful or unsuccessful.

2)**consistency**: it represents consistency of the referential integrity of the db, nique primary keys in tables.

3) **ISOLATION**: thre may be many transaction processing on same data at same data but the there will be isolated from data to data.

**Ddurability**:

Once the transaction is complete the result should be made perminant and cannot be erased from db, due to system failure.

**Types of transaction management;**

1)**progrmatic**: which includes program in a transaction

2)**declarative**: seperates transaction from business logic by using annotations.

This will be prefarable,

**Local vs. global transaction**:

Local means a single transaction like jdbc connection.

Global is like multiple transactions on distributed system

Both are important.

**SPRING ORM:**

Spring provides API to easily integrate spring with orm framework such as Hibernate, JPA(Java Persistence API), JDO(Java Data Objects), Oracle Toplink and iBATIS.

Advantage of ORM Frameworks with Spring

There are a lot of advantage of Spring framework in respect to ORM frameworks. There are as follows:

* **Less coding is required**: By the help of Spring framework, you don't need to write extra codes before and after the actual database logic such as getting the connection, starting transaction, commiting transaction, closing connection etc.
* **Easy to test**: Spring's IoC approach makes it easy to test the application.
* **Better exception handling**: Spring framework provides its own API for exception handling with ORM framework.
* **Integrated transaction management**: By the help of Spring framework, we can wrap our mapping code with an explicit template wrapper class or AOP style method interceptor.

**SPRINGBOOT**:

Example :<http://www.onlinetutorialspoint.com/spring-boot/spring-boot-mvc-example-tutorials.html>

**SPRING AOP:**

One of the key components of Spring Framework is the **Aspect oriented programming (AOP)** framework. Aspect-Oriented Programming entails breaking down program logic into distinct parts called so-called concerns. The functions that span multiple points of an application are called **cross-cutting concerns** and these cross-cutting concerns are conceptually separate from the application's business logic. There are various common good examples of aspects like logging, auditing, declarative transactions, security, caching, etc.

The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect. Dependency Injection helps you decouple your application objects from each other and AOP helps you decouple cross-cutting concerns from the objects that they affect. AOP is like triggers in programming languages such as Perl, .NET, Java, and others.

Spring AOP module provides interceptors to intercept an application. For example, when a method is executed, you can add extra functionality before or after the method execution.

**HIBERNATE ANSWERS:**

**CRUD**: are the operations which can be done in data repository, you directly handles the records or data objects. Apart from the records are passive entities. Typically its just database tables and rcords.

**HQL**: is a object oriented query language.same as sql but it performs operations on persistent object and their properties instead of tables and columns. Hql query is translates by hibernate into conventel sql queries. Which in turns perform action on database.

In this the select, from and whre are not case sensitive:

**FROM CLAUSE: SYNTAX**

String hql = “FROM EMPLOYEE”;

Query query = session.createQuery(hql);

List list = query.list();

**AS CLAUSE:**

String hql = “FROM EMLOYEE AS E”;

Query query = session.crateQuery(hql);

List list = query.list();

**SELECT:**

String = “SELECT E.firstname from EMPLOYEE E”;

Query query = session.cretaeQurey(hql);

List list = query.list();

**WHERE:**

String hql = “FROM EMPLOYE WHERE E.ID =1”;

Query query = session.createQuery(hql);

List list = query.list();

**ORDER BY CLAUSE:**

String hql = “FROM EMPLOYEE WHERE E.ID > 10 ORDER BY E.name”;

Query query = session.createQuery(hql);

List list = query.list();

**GROUPBY:**

String = “SELECT SUM(E.SALARY), E.FIRSTNAME FROM EMPLOYEE E” + “GROUP BY E.FIRSTNAME”;

Query query = session.createQuery(hql);

List list = query.list();

**NAMED PARAMETERS**: that accepts input from user

String hql =”FROM EMPLOYEE E WHERE E.ID = :EMPLOYEE\_ID”;

Query query = session.createQuery(hql);

Query.setparametr(“employer\_id”, 10);

List list = query.list();

**UPDATE CLAUSE:**

String =”IPDATE FROM EMPLOYEE SET SALARY=:SALARY”, “WHERE ID = :EMPLOYER\_ID”;

Query query = session.createQuery(hql);

Qurey.setparameter(“salary”, 1000);

Query.setparametr(“employer\_id”, 10);

List list = query.list();

**DELETE:**

String = “DELETE FROM EMPLOYEE” + “WHERE ID = :EMPLOYEE\_ID;

Query query = session.createQuery(hql);

Query.setparametr(“employer\_id”, 10);

Int result = qury = query.executeupdate();

Sysout(“rows affected:” + result);

**INSERT:**

String hql = “insert into employee(firstname, lastname, salary)” + “select fiestname, lastname, salary from old\_employee”;

Query query = session.createquery(hql);

Int result = query.executeupdate();

Sysout(“rows afftecred:” + result);