Spring

NILESH BHANDARE

Agenda

- **Session I**
  - **What is Spring ?**
  - **Spring Framework**
  - **Spring MVC**
- **Session II**
  - **Implementation Spring MVC**
  - **Data Transfer View to Controller and Controller to View**
- **Session III**
  - **Spring ORM Part A**
- **Session IV**
  - **Spring ORM Part B**

What is Spring ?

- a ***lightweight framework***.
- a ***framework of frameworks*** because it provides support to various frameworks such as **Struts, Hibernate, Tapestry, EJB, JSF,** etc.
- Spring framework comprises several modules such as **IOC, Dependency Injection, AOP, DAO, Context, ORM, WEB MVC** etc
- developed by Rod Johnson in 2003.
- makes the easy development of JavaEE application.

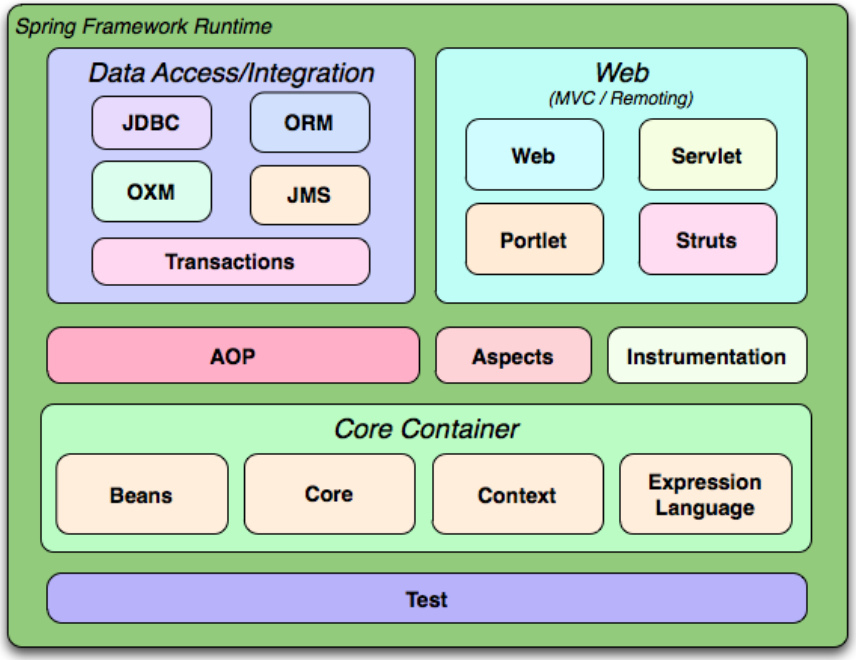
Dependency Injection & Inversion of Control

- **Inversion of Control** is a principle in software engineering which transfers the control of objects or portions of a program to a container or framework

- **Dependency Injection** is **a fundamental aspect of the Spring framework**, through which the Spring container "injects" objects into other objects or "dependencies".
- Dependency injection is a pattern we can use to implement IoC. **where the control being inverted is setting an object's dependencies by IOC controller.**

More Details read shared spring notes

Spring Framework

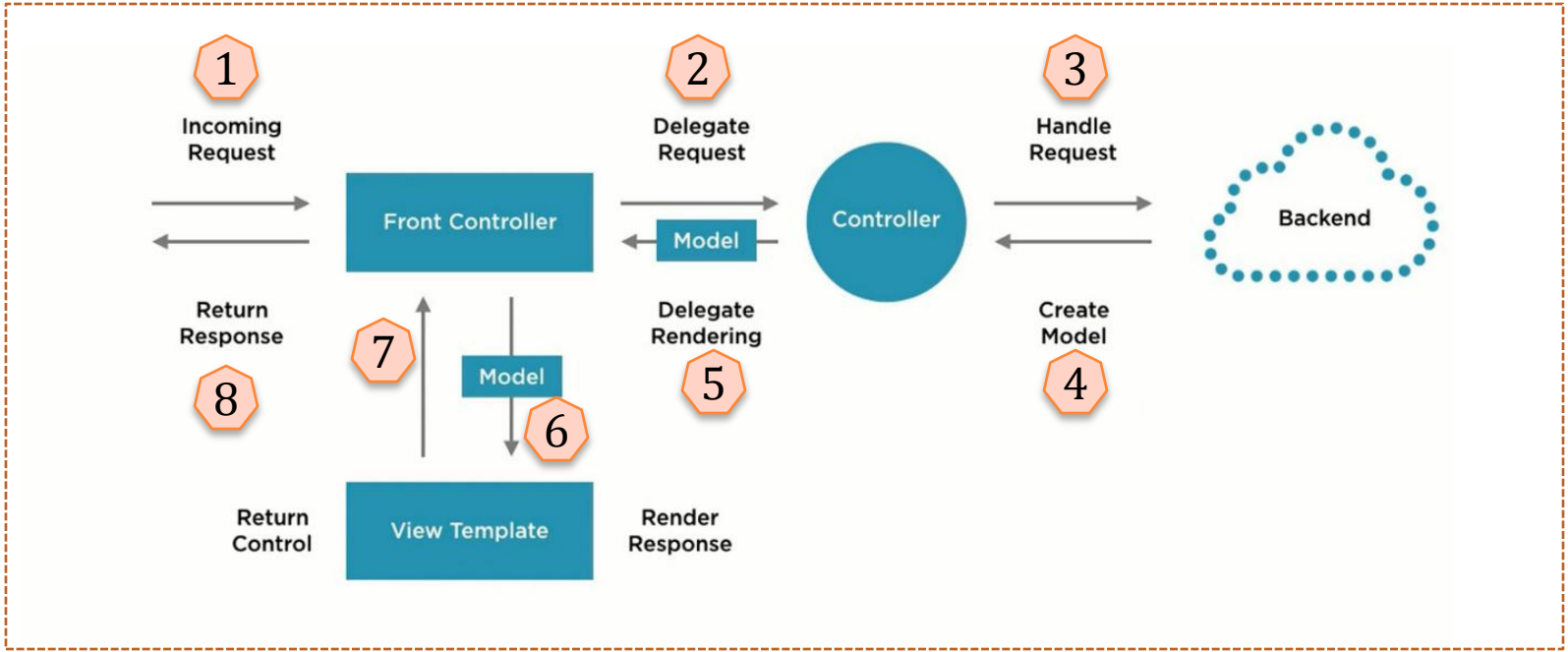


- **Core**: Provides core features like DI (Dependency Injection), Internationalisation, Validation, and AOP (Aspect Oriented Programming)
- **Data Access**: Supports data access through JTA (Java Transaction API), JPA (Java Persistence API), and JDBC (Java Database Connectivity)
- **Web**: Supports both Servlet API (**Spring MVC**) and of recently Reactive API (**Spring WebFlux**), and additionally supports WebSockets, STOMP, and WebClient
- **Integration**: Supports integration to Enterprise Java through JMS (Java Message Service), JMX (Java Management Extension), and RMI (Remote Method Invocation)
- **Testing**: Wide support for unit and integration testing through Mock Objects, Test Fixtures, Context Management, and Caching

IOC Controller

- The IoC container is responsible to instantiate, configure and assemble the objects. The IoC container gets information from the XML file and works accordingly. The main tasks performed by IoC container are:
  - to instantiate the application class
  - to configure the object
  - to assemble the dependencies between the objects

Spring MVC Lifecycle



More details-click here

Spring MVC (Model-View-Controller Design Pattern)

- a **Java framework** which is used to build web applications.
- It implements all the basic features of a core spring framework like IoC, Dependency Injection.
- Spring MVC use **DispatcherServlet**.
  - **DispatcherServlet** is a class that receives the incoming request and maps it to the right resource such as controllers, models, and views.

- **Model** - data of the application (object / collection of objects)
- **Controller** - A controller contains the business logic of an application.
  - @Controller annotation is used to mark the class as the controller.
- **View** - Presentation (JSP pages) Generally, JSP+JSTL is used to create a view page.

Data Transfer- Controller to View

- By creating object of
  - Model
    - `addAttribute(String attributeName, Object attributeValue)`
  - ModelAndView
    - `addObject("message", "Baeldung");`

ModelAndView Object---Controller to View-----Model Object


```
//return type ModelAndView
public ModelAndView home()
{
//create object of ModelAndView
ModelAndView mod=new ModelAndView();

//set data in object
mod.addObject("username","Nilesh");

//set view name
mod.setViewName("index");

//return ModelAndView object
return mod;
}
```

```
public String home(Model model)
{
model.addAttribute("name","nilesh
bhandare");
return "register";
}
```

View to Controller

By using @RequestParam Annotation….

Create simple Spring MCV Application

- StepbyStep Manual Shared….

Spring ORM

13

MR. NILESH BHANDARE
ndbhandare@mitaoe.ac.in
9096142430

Spring ORM

- In hibernate framework, we provide all the database information hibernate.cfg.xml file.
- But if we are going to integrate the hibernate application with spring, we don't need to create the hibernate.cfg.xml file. We can provide all the information in the applicationContext.xml file.
- The Spring framework provides **HibernateTemplate** class, so you don't need to follow so many steps like create Configuration, BuildSessionFactory, Session, beginning and committing transaction etc.

```
//creating configuration
Configuration cfg=new Configuration();
cfg.configure("hibernate.cfg.xml");

//creating seession factory object
SessionFactory factory=cfg.buildSessionFactory();

//creating session object
Session session=factory.openSession();

//creating transaction object
Transaction t=session.beginTransaction();

Employee e1=new Employee(111,"arun",40000);
session.persist(e1);//persisting the object

t.commit();//transaction is commited
session.close();
```

Hibernate

Why Spring ORM ?

```
Employee e1=new Employee(111,"arun",40000);
hibernateTemplate.save(e1);
```

Spring ORM (hibernate)

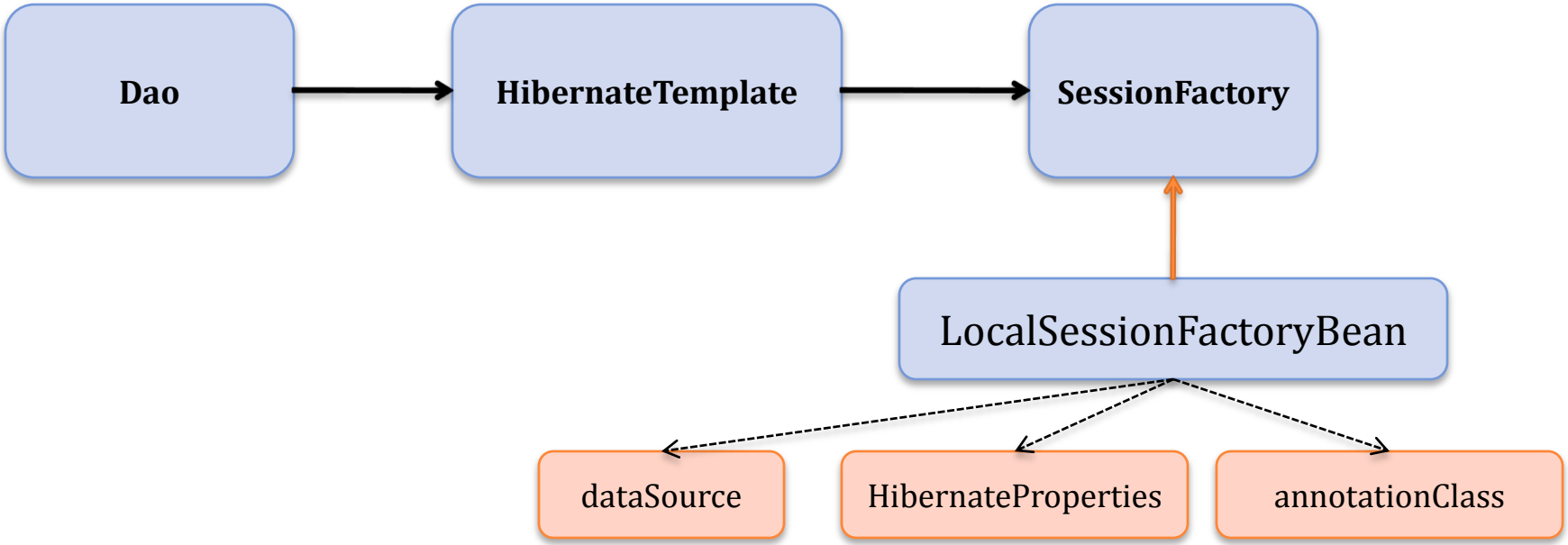
Why Spring ORM?

- **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.

HibernateTemplate class

No.	Method	Description
1)	void persist(Object entity)	persists the given object.
2)	Serializable save(Object entity)	persists the given object and returns id.
3)	void saveOrUpdate(Object entity)	persists or updates the given object. If id is found, it updates the record otherwise saves the record.
4)	void update(Object entity)	updates the given object.
5)	void delete(Object entity)	deletes the given object on the basis of id.
6)	Object get(Class entityClass, Serializable id)	returns the persistent object on the basis of given id.
7)	Object load(Class entityClass, Serializable id)	returns the persistent object on the basis of given id.
8)	List loadAll(Class entityClass)	returns the all the persistent objects.

Working Spring ORM



Step by Step process to create Spring ORM application

- StepbyStep Manual Shared….

THANK YOU

NILESH BHANDARE

NDBHANDARE@MITAOE.AC.IN

9096142430