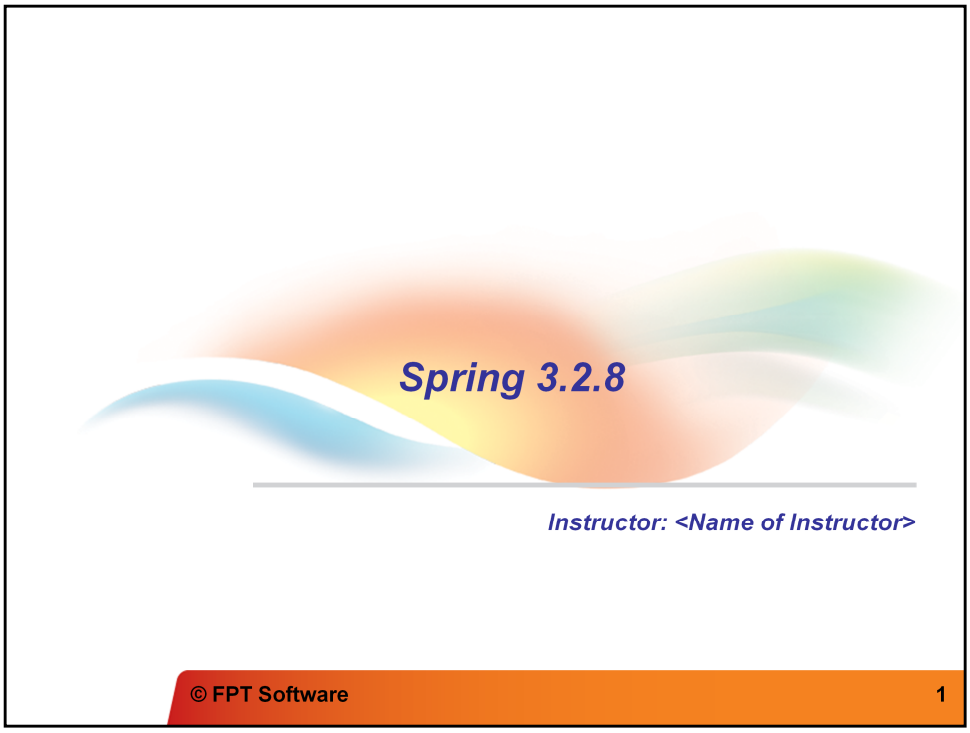


Latest updated by: HanhTT1

Objectives

- Understand some modules of Spring
- Understand definition of Spring Bean, DI & IOC, AOP Transaction
- Understand some features to integrate with other framework: Hibernate, SpringMVC.

Agenda

- Spring Overview
- Spring IOC and DI
- Spring Bean
- Spring AOP Transaction
- Handful of Services

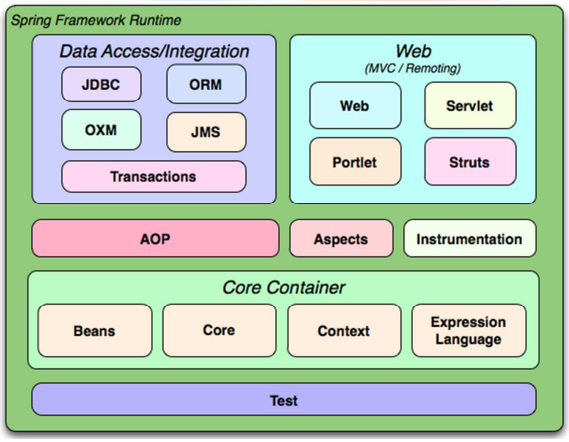
Spring Overview

What is Spring?

- Is Java Flat-form
- Is Framework which Have Comprehensive infrastructure
- Is modular, allowing you to use only those parts that you need, without having to bring in the rest

Spring Overview

The Spring Framework consists of features organized into about 20 modules as shown in the following diagram.



Spring Overview

Data Access/Integration

- JDBC module provides a JDBC-abstraction layer
- *ORM (*object-relational mapping APIs): *integrate with JPA*, JDO, Hibernate, and iBatis.
- OXM (Object/XML mapping) implemente for JAXB, Castor, XMLBeans, JiBX and XStream.
- JMS (Java messaging service): producing and consuming messages.
- Transaction: supports programmatic and declarative transaction management.

Spring Overview

Web

- Web: Support some features in web application such as : file upload, file download
- Web-Servlet: contains Spring's model-view-controller (*MVC) implementation for web* applications
- Web-Struts: contains the support classes for integrating a classic Struts web tier (struts 1 or struts 2) within a Spring application
- *Web-Portlet module provides the MVC implementation to be used in a portlet environment and* mirrors the functionality of Web-Servlet module.

Spring Overview

AOP and Instrument

- Spring's *AOP module provides an AOP Alliance-compliant aspect-oriented programming* implementation allowing you to define
- *Aspects module provides integration with AspectJ.*
- *Instrumentation module provides class instrumentation support and classloader implementations* to be used in certain application servers.

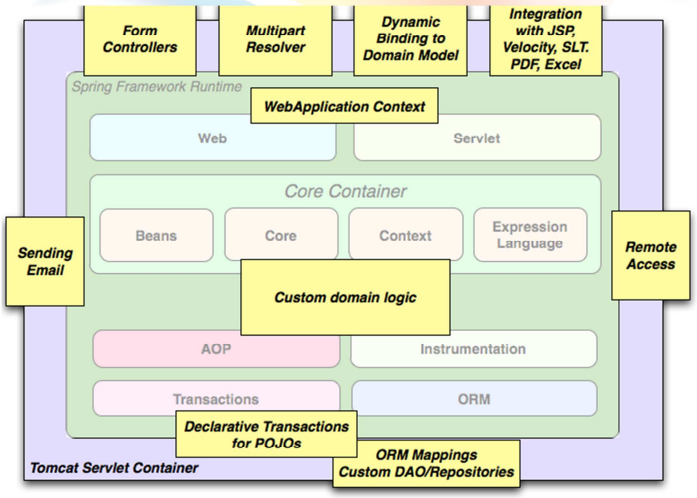
Spring Overview

Test

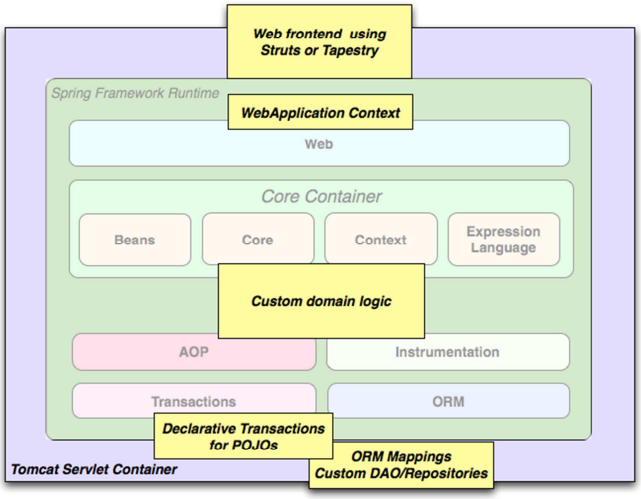
- The *Test module supports the testing of Spring components with JUnit or TestNG*

Spring Overview

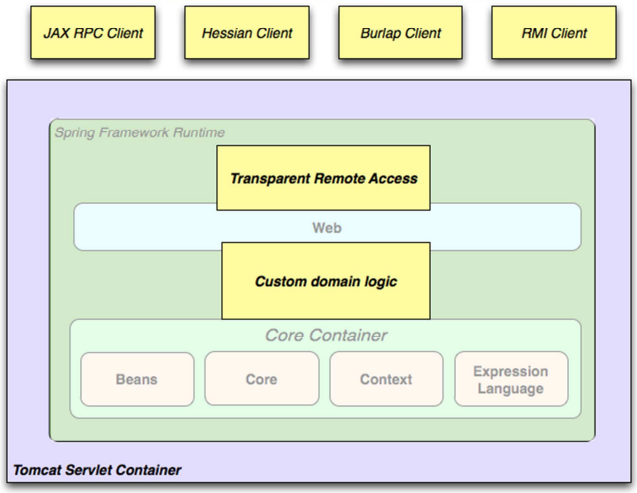
Usage scenarios of Spring Fully Web Application



Usage scenarios of Spring Spring middle-tier



Usage scenarios of Spring Web Services



What's IOC and DI?

Let's first understand the issue, consider the following class

```
public class ClsCustomer
{
      private ClsAddress address;
      public ClsCustormer () {
            address = new ClsAddress();
      }
}
```

Problem1: References

Problem2: Aware of concrete classes

What' s IOC and DI?

The problems are:

- Customer class controls the creation of address object.
- Address class is directly referenced in the customer class which leads to tight coupling between address and customer objects.
- Customer class is aware of the address class type.

So if for any reason the address object is not able to create the whole customer class will fail in the constructor initialization itself.

What' s IOC and DI?

The solution:

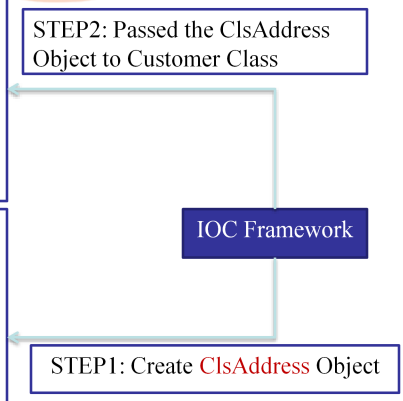
The main problem roots from the customer class creating the address object. If we are able to shift this task / control of object creation from the customer class to some other entity we have solved our problem.

In other sentence if we are able to invert this control to a third party we have found our solution. So the solution name is IOC (Inversion of control).

What' s IOC and DI?

```
public class ClsCustomer
{
        private ClsAddress address;
        public void setAddress(ClsAddress obj) {
                address = obj;
        }
}
```

```
public class ClsAddress
{
        public String  strAddress;
        …
}
```



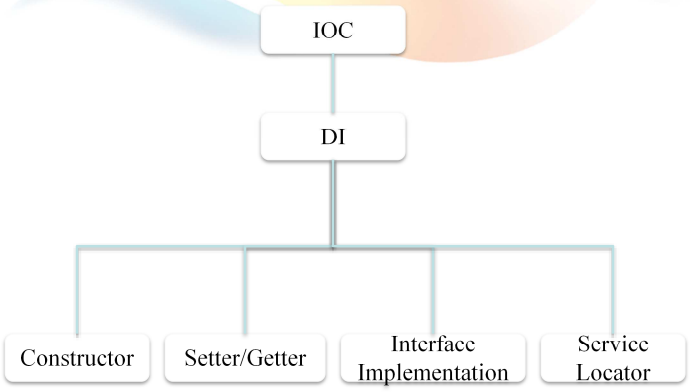
What' s IOC and DI?

IOC framework can be a class, client or some kind of IOC container. So it will be two step procedure IOC framework creates the address object and passes this reference to the customer class.

The way to implement IOC is called DI (Dependency injection)

Spring IOC and DI

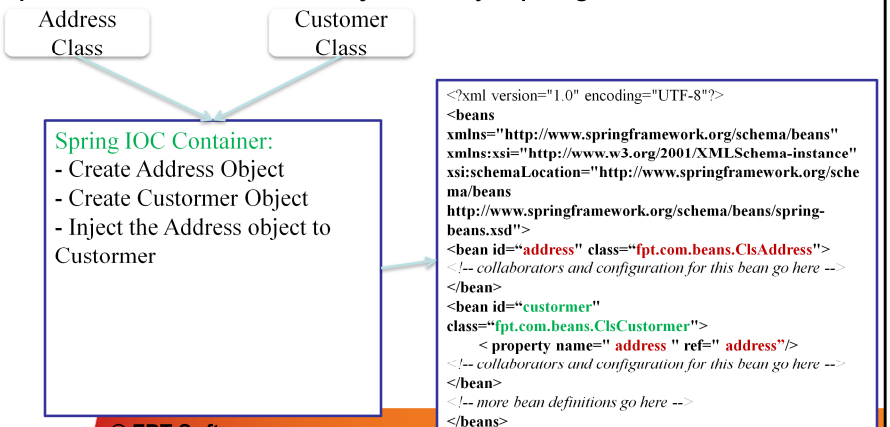
Ways of implement IOC



Spring IOC and DI

Spring IOC and DI

Spring IOC and DI is a process whereby objects define their dependencies which are injected by Spring IOC container.



Spring IOC and DI

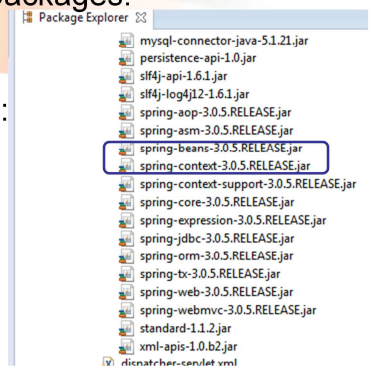
Spring IOC container have 2 basic packages:

org.springframework.beans

org.springframework.context

Spring IOC container have structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
<bean id="…" class="…">
<!-- collaborators and configuration for this bean go here -->
</bean>
<!-- more bean definitions go here -->
…
</beans>
```



Spring IOC contaniner is defined in xml file applicationContext.xml

Spring IOC and DI

Instantiating Spring IOC container:

In Java Class

```
ApplicationContext context =
new ClassPathXmlApplicationContext(new String[]
{"applicationContext.xml"});
```

In web.xml

```
<context-param>
   <param-name>contextConfigLocation</param-name>
   <param-value>classpath:applicationContext.xml</param-value>
</context-param>
```

After that, you can get any object which you defined in this container

```
ApplicationContext context =
new ClassPathXmlApplicationContext(new String[] {"applicationContext.xml"});

ClsAddress address = context.getBeans("address", ClsAddress. Class)
```

Spring Bean

What' s Spring Bean?

In Spring, the objects that form the backbone of your application and that are managed by the Spring IoC *container are called beans*

A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IOC container

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/sche
ma/beans
http://www.springframework.org/schema/beans/spring-
beans.xsd">
<bean id="address" class="com.fpt.dto.ClsAddress">
<!-- collaborators and configuration for this bean go here -->
</bean>
<bean id="custormer" class="com.fpt.dto.ClsCustormer">
      < property name=" address " ref=" address"/>
<!-- collaborators and configuration for this bean go here -->
</bean>
<!-- more bean definitions go here -->
</beans>
```

```
package com.fpt.contacts.dto;
public class ClsCustomer
{
  private ClsAddress address;
  public ClsAddress getAddress() {
       return address;
  }
  public void setAddress(ClsAddress
address) {
       this.address = address;
  }
}
```

Some Bean *Properties*

Property	Explain
class	This attribute is mandatory and specify the bean class to be used to create the bean.
name	This attribute specifies the bean identifier uniquely. In XML-based configuration metadata, you use the id and/or name attributes to specify the bean identifier(s).
scope	This attribute specifies the scope of the objects created from a particular bean definition
constructor-arg	Instance object with argument
properties	Define properties of class.
autowire mode	Set autowire for bean
lazy-initialization mode	A lazy-initialized bean tells the IoC container to create a bean instance when it is first requested, rather than at startup

Class Property

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
<bean id="address" class="com.fpt.dto.ClsAddress">
<!-- collaborators and configuration for this bean go here -->
</bean>
</beans>
```

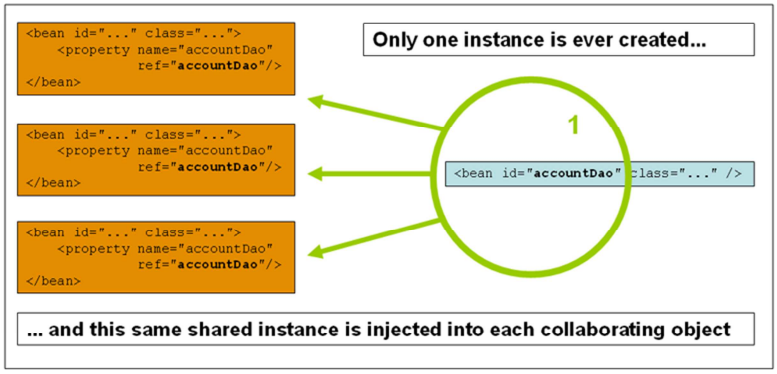
```
package com.fpt.contacts.dto;
/**
 *
 * @author LuanTT
 * ClsAddress
 *
 */
public class ClsAddress {
    private String strAddress;
    private String strPhone;
    private String strNote;

    public String getStrAddress() {
        return strAddress;
    }
    public void setStrAddress(String strAddress) {
        this.strAddress = strAddress;
    }
    public String getStrPhone() {
        return strPhone;
    }
    public void setStrPhone(String strPhone) {
        this.strPhone = strPhone;
    }
    public String getStrNote() {
        return strNote;
    }
    public void setStrNote(String strNote) {
        this.strNote = strNote;
    }

}
```

Scope Property

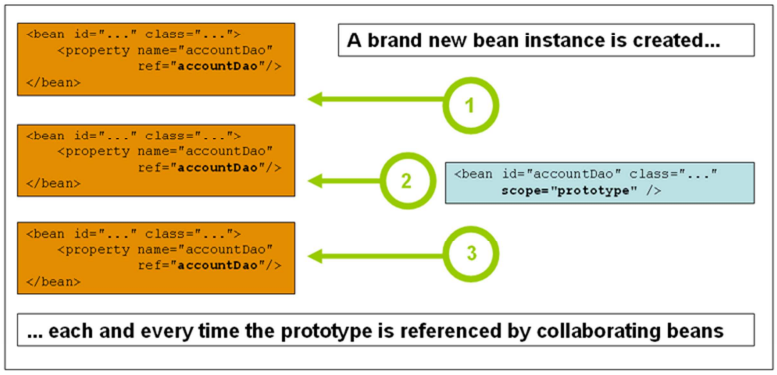
Scope	Description
singleton	Default) Scopes a single bean definition to a single object instance per Spring IoC container.
prototype	Scopes a single bean definition to any number of object instances.
request	Scopes a single bean definition to the lifecycle of a single HTTP request; that is, each HTTP request has its own instance of a bean created off the back of a single bean definition.
session	Scopes a single bean definition to the lifecycle of an HTTP Session. Only valid in the context of a web-aware Spring ApplicationContext.
global session	Scopes a single bean definition to the lifecycle of a global HTTP Session. Typically only valid when used in a portlet context. Only valid in the context of a web-aware Spring ApplicationContext.

Spring Bean

Scope "singleton"



Scope "prototype"



Spring AOP Transaction

- Spring provides AOP support for declarative transactions
- Delegates to a PlatformTransactionManager instance
  - DataSourceTransactionManager
  - HibernateTransactionManager
  - JdoTransactionManager
  - JtaTransactionManager

Spring AOP Transaction

Transaction Configuration:

```
<bean id="sessionFactory"
   class="org.springframework.orm.hibernate.LocalSessionFactoryBean">
        <property name="dataSource"><ref bean="dataSource"/></property>
        <property name="mappingResources">
            <list>
                <value>com/../model/*.hbm.xml</value>
            </list>
        </property>
</bean>

<bean id="transactionManager"
 class="org.springframework.orm.hibernate.HibernateTransactionManager">
  <property name="sessionFactory">
    <ref bean="sessionFactory"/>
  </property>
</bean>
```

Spring AOP Transaction

Declarative Transactions:

- Declarative transactional support can be added to any bean by using TransactionProxyFactoryBean
- Similar to EJB, transaction attributes may be defined on a per-method basis

```
<bean id="reservationService"
   class="fsoft.ReservationService">
   <property name="transactionManager">
       <ref bean="transactionManager"/>
   </property>
    <property name="target"><ref local="reservationServiceTarget"/></property>
    <property name="transactionAttributes">
      <props>
        <prop key="reserveRoom*">PROPAGATION_REQUIRED</prop>
        <prop key="*">PROPAGATION_REQUIRED,readOnly</prop>
      </props>
    </property>
</bean>
```

Handful of Services

- DAO support: Spring offers templates classes to deal with a Hibernate/JDBC/… connection
- Exception translator: all the proprietary Hibernate/JDBC/… exceptions are catched by Spring, and rethrown as Runtime non-specific consistent exceptions
- Hence the DAO code is not dependant on the underlying datasource!

Handful of Services

- Many ORM tools are supported: Hibernate, JDO, Apache OJB, iBATIS
- Templates using IoC to reduce the amount of code in the DAO objects

```
public List<Association> findAssociationByLogin(String login) {
    return getHibernateTemplate().find("from Association where lower(login) = ?", login.toLowerCase());
}
```

Handful of Services

- Support of RMI
- Very easy to expose and connect to webservices

- Support of JMS
- JMS templates, JMSException translation

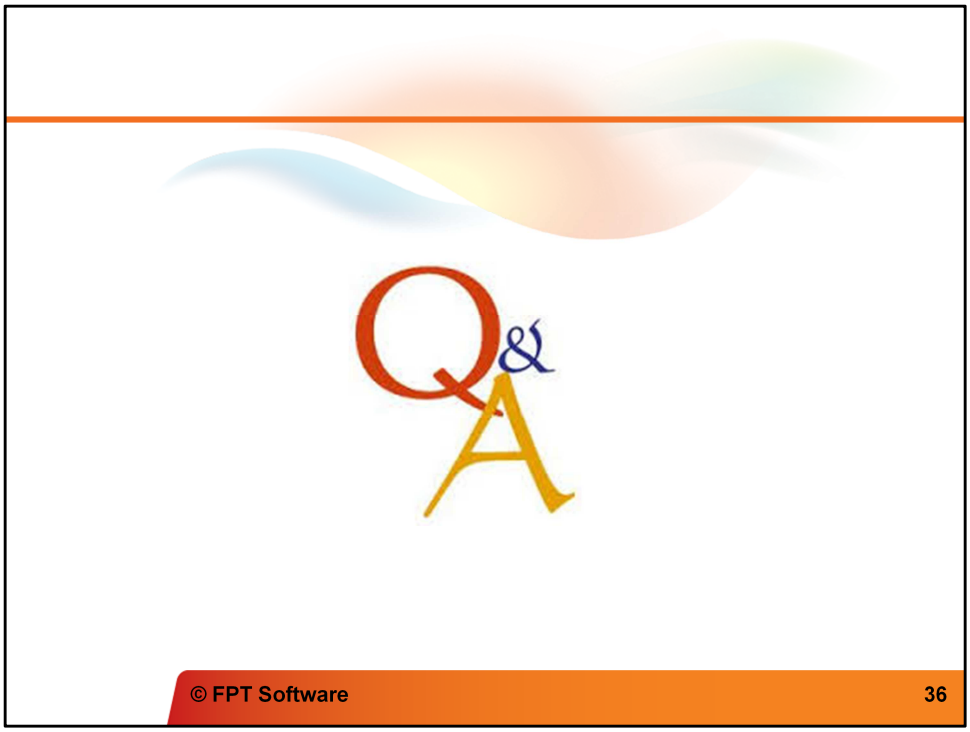
Handful of Services

- A mail abstraction layer
  - Templates

- Jobs scheduling (Quartz, Timer)
  - Cron
  - Business layer unaware

Reference

- http://www.springsource.org/documentation
- http://courses.coreservlets.com/Course-Materials/spring.html


Q&A
© FPT Software
36