
# Spring Framework

# Agenda

- Overview
- Spring Believes and Features
- Basic Wiring
- Inversion of Control,Dependency Injection
- Code snippet
- AOP
- Spring MVC

# Spring Framework

UnitedHealth Group

**Framework:-**

- *is a basic conceptual structure used to solve a complex issue*
- *configure, compose complex application from simpler components*

**Spring Framework:-**

Light-weight framework for the development of enterprise-ready appln.

- used to configure
  - declarative transaction management,
  - remote access to your logic using RMI or web services, mailing facilities
  - various options in persisting data to a database
- used in modular fashion
  - allows to use in parts and leave the other components which is not required by the application

# Spring Framework

- A lightweight framework that addresses each tier in a Web application.
  - Presentation layer – An MVC framework that is most similar to Struts but is more powerful and easy to use.
  - Business layer – Lightweight IoC container and AOP support (including built in aspects)
  - Persistence layer – DAO template support for popular ORMs and JDBC
- Reduces code and speeds up development
- Removes common code issues like leaking connections and more.
- Spring config contains implementation classes while your code should program to interfaces.

# Spring Believes

- OO design
- J2EE - easier to use.
- JavaBeans - a Great way of configuring applications.
- Interfaces - reduces the complexity cost.

- Checked exceptions - are overused in Java. Spring shouldn't force you to catch exceptions you're unlikely to be able to recover from.
- Testability - easier to test.
- Application code - "not" depend on Spring APIs

- Not compete with - Good existing solutions, but should foster integration

# Spring Features

- **Easier to maintain** Simplicity
  - *as Good Design is M. Imp than underlying technologies*
- **Decoupled**
  - *Java beans loosely coupled through the interface is a good model*
- **Testability**
  - *easier to test code as Spring application is developed with POJO, no need of J2EE Container to be started as we are testing POJO*
- **Framework**
  - *Inversion Of Control and Dependency Injection*
  - *Aspect Oriented Framework*
- **Lightweight**
  - *Size and overhead single jar file around 1 MB (less than 2 MB)*
  - *Processing overhead required by spring is negligible*

# Spring Features

- **Non Intrusive Framework**
  - *Objects in spring enabled application have no dependencies on spring specific classes. i.e. Application code has minimal or no dependency on Spring APIs*
- **Inversion Of Control**
  - *Objects are passively given their dependent instead of creating or looking for dependent objects for themselves.*
  - *Container gives the dependencies to the object at instantiation  without waiting to be asked*
- **Dependency Injection**
  - *Injecting via setter and/or via constructor*
- **Aspect Oriented**
  - *dividing application based on functionality (aspect)*

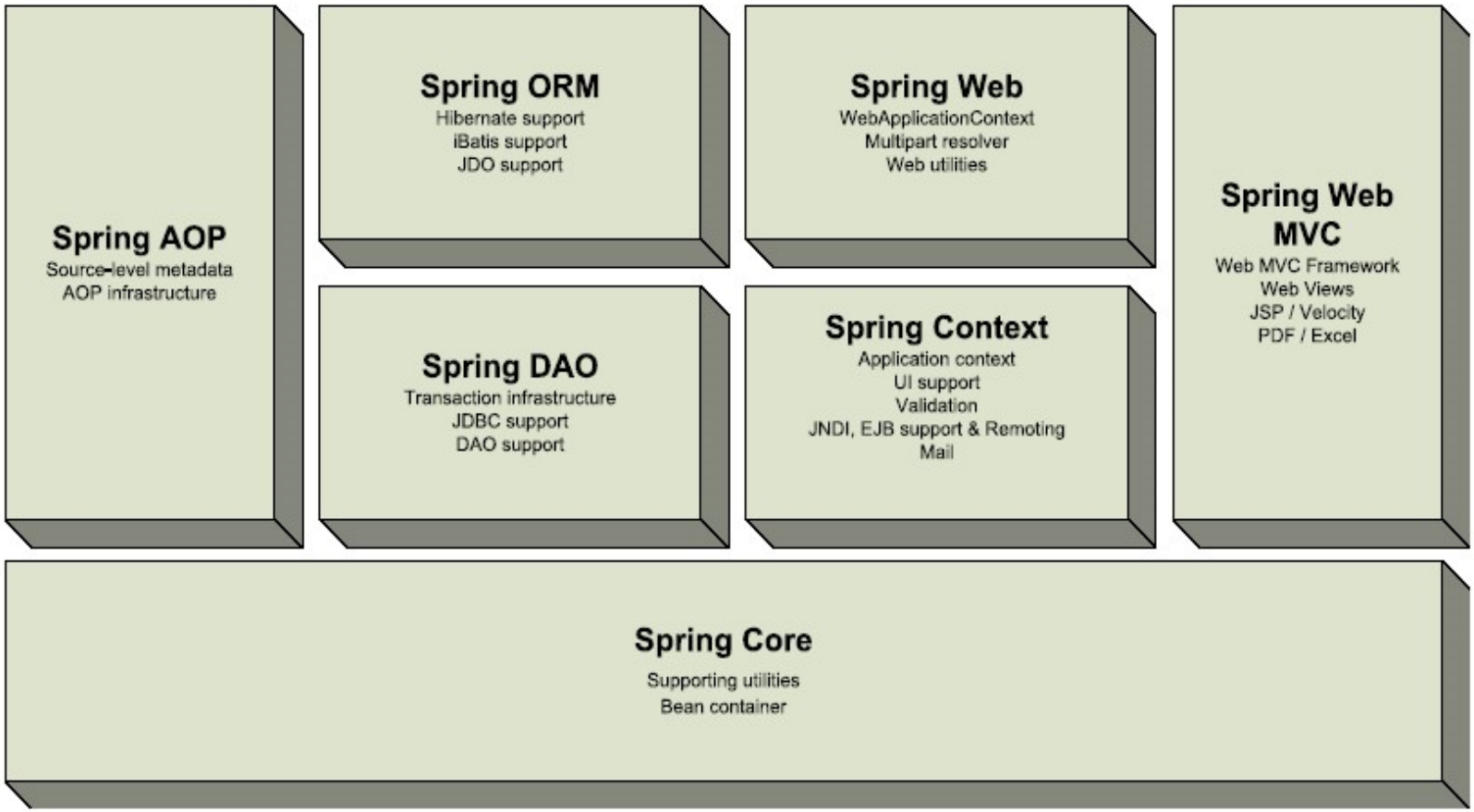
# Spring Features

UnitedHealth Group

**Services**

- **Transaction Management**
  - *provides a generic abstraction layer for transaction management*
  - *allowing the developer to add the pluggable transaction managers*
  - *making it easy to demarcate transactions without dealing with low-level issues*
  - *is not tied to J2EE environments and it can be also used in container less environments*
- **Integration with Hibernate, JDO, and iBATIS:**
  - *provides best Integration services with Hibernate, JDO and iBATIS*
- **MVC Framework**
  - *comes with MVC web application framework, built on core Spring functionality.*
  - *is highly configurable via strategy interfaces, and accommodates multiple view technologies like JSP, Velocity, Tiles, iText, and POI.*
  - *other frameworks can be easily used instead of Spring MVC Framework i.e struts*

# Spring Components



# Spring Components

- **Spring Web**
  - *The Spring Web module is part of Spring's web application development stack, which includes Spring MVC.*
- **Spring DAO**
  - *The DAO (Data Access Object) support in Spring is primarily for standardizing the data access work using the technologies like JDBC, Hibernate or JDO.*
- **Spring Context**
  - *builds on the beans package to add support for message sources and for the Observer design pattern,*
  - *the ability for application objects to obtain resources using a consistent API.*

# Spring Components

- **Spring Web MVC**
  - *provides the MVC implementations for the web applications.*
- **Spring Core**
  - *is the most import component of the Spring Framework.*
  - *provides the Dependency Injection features.*
  - *BeanFactory is a factory pattern which separates the dependencies like*
  - *initialization*
  - *creation and*
  - *access of the objects from your actual program logic.*

# Spring Container/Context


- Is at core of Spring Framework
- Uses IOC
- Uses Wiring
- 2 distinct types:-
  - **BeanFactory**
    - *org.springframework.beans.factory.BeanFactory interface*
  - **ApplicationContext**
    - *org.springframework.context.ApplicationContext interface*

# BeanFactory Vs. ApplicationContext

| | |
|---|---|
| • simplest container | build on notion of a BeanFactory by |
| • provides basic support of DI | providing application fw services:- |
| | 1. resolves message from property file |
| | 2. publish application events |
| • implementation of Factory DP | resolve Text Message with I18 N |
| • responsibilities | generic way to load file resources |
| – create and dispense beans | publish events to registered listeners |
| • general purpose factory | AC is preferred over BF |
| • where resources are scarce | Good |
| • Lazily Loading | preloads all singleton on startup |

# Beans Life Cycle

- **Bean Life Cycle**
  - BeanFactory Context Vs Application Context
- **Wiring**
  - Piecing together beans with in Spring Container
  - Spring Container supports wiring through XML

| | | *load xml from/by* |
|---|---|---|
| – *XmlBeanFactory* | - | *java.io.InputStream* |
| – *ClassPathXmlAC* | - | *classpath* |
| – *FileSystemXmlAC* | - | *file system* |
| – *XmlWebAC* | - | *web application* |

# Spring Basic wiring

- Act of creating the association between application components
- **Spring Wiring:-**
  - Controlling the Wiring
    - *Configure Bean's Properties*
  - Configuring
    - *Externalize Deployment Configuration in separate files*
  - Life Cycle Management
    - *Manage Life cycle of Beans*

**AutoWiring:-**

- *Can be by 4 ways byName, byType, constructor, autodetect*
- *<bean id="company" class="com.uhg.india" autowire="autowire type" />*

# Beans

| Prototyping | Singleton |
| --- | --- |
| • define a blue print<br>• Beans are created based on this<br>• Avoid when resources are scarce | by default all spring beans<br>always give the exact same instance<br>useful when need of unique instance<br>*of bean each time it is asked for* |
| <bean id="company"<br>*class="com.uhg.India"*<br>*singleton="false" />* | <bean id="company"<br>*class="com.uhg.India"*<br>*singleton="true" />* |

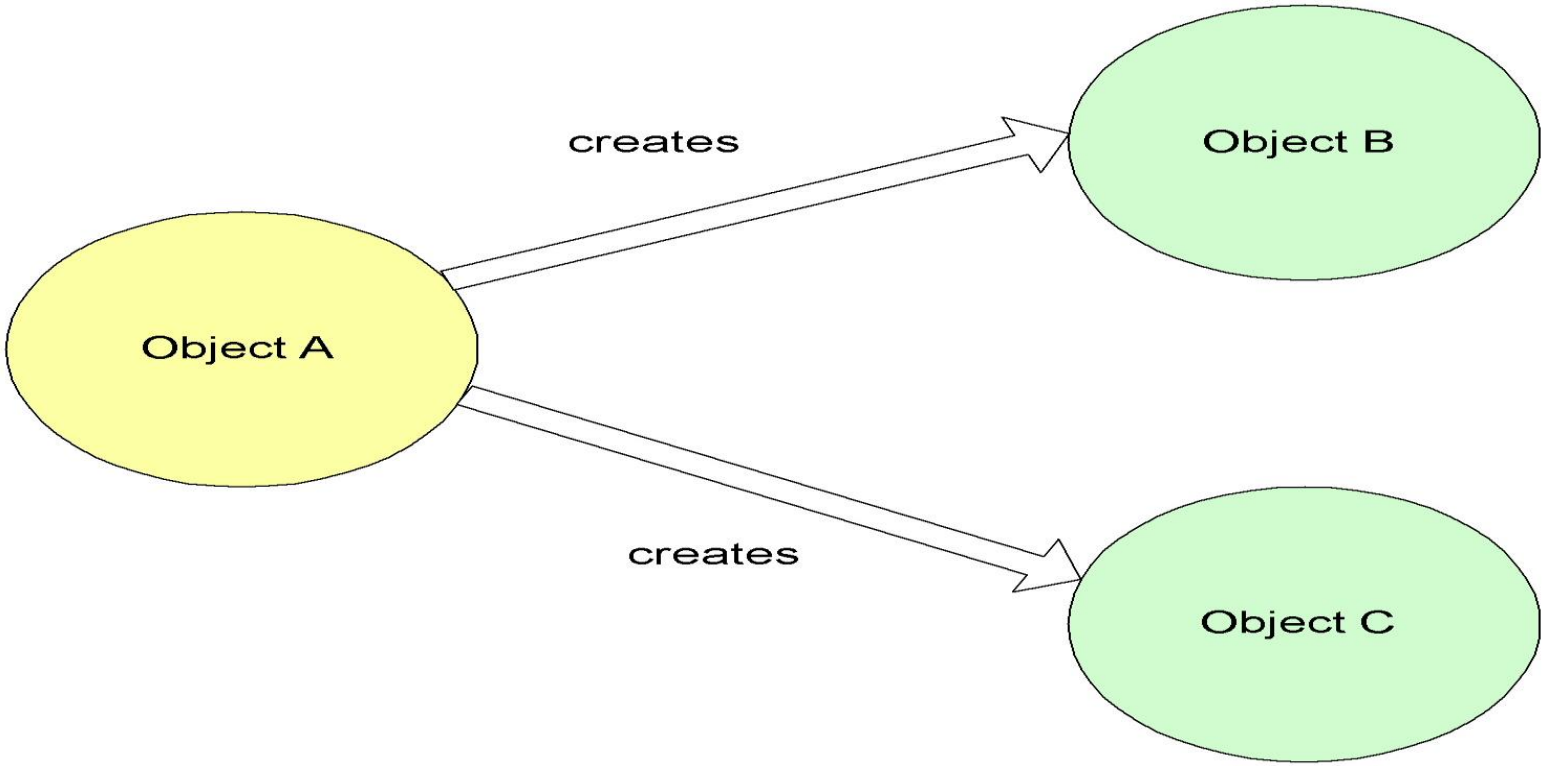
# Inversion of Control

UnitedHealth Group

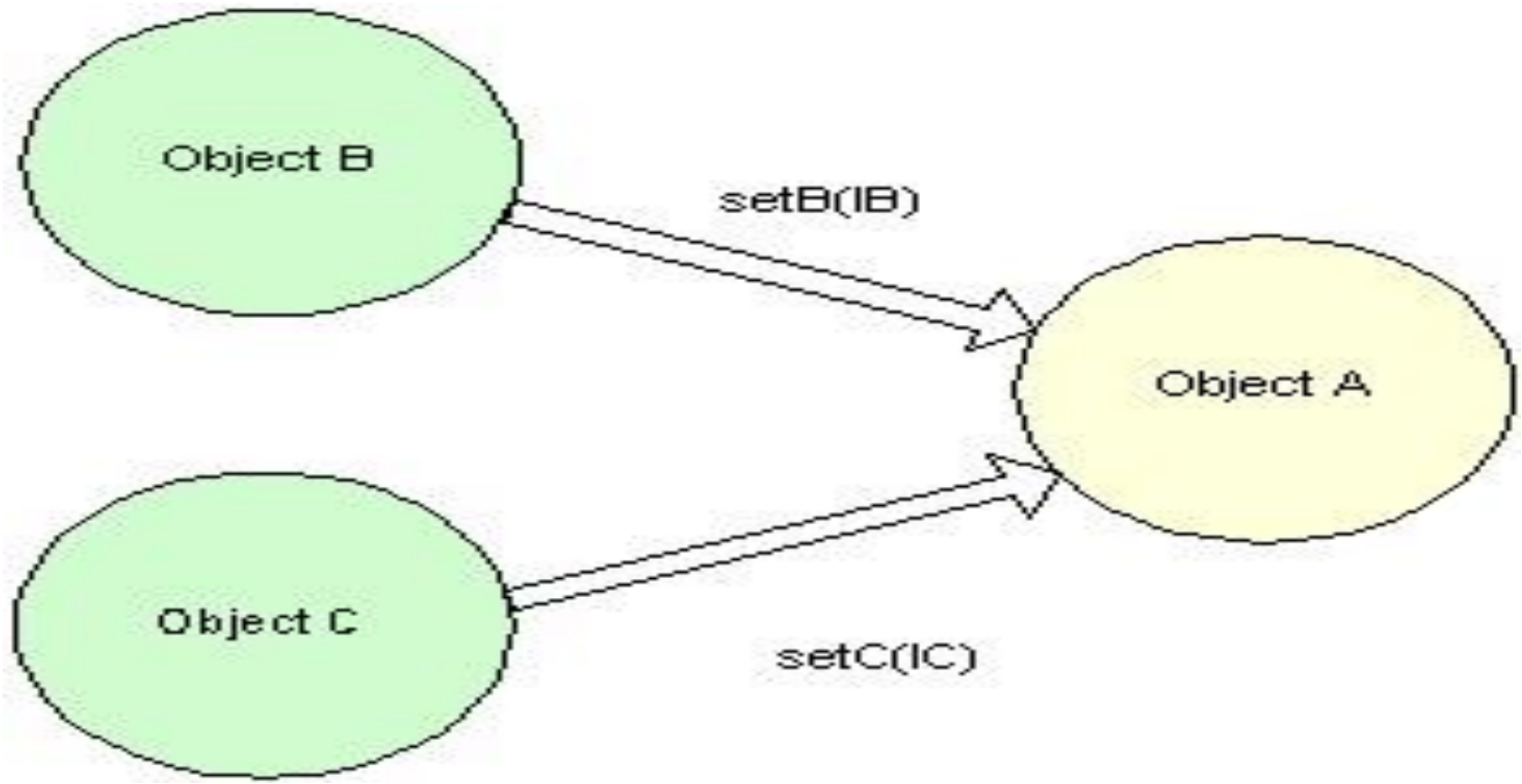
- Dependency injection
- Beans define their dependencies through constructor arguments or properties
- The container provides the injection at runtime
- “Don’t talk to strangers”
- Also known as the Hollywood principle –

  “don’t call me I will call you” – container callbacks
- Decouples object creators and locators from application logic
- Easy to maintain and reuse
- Testing is easier

# Non-IOC/Dependency Injection



# IOC/Dependency Injection



# Dependency Injection


**Injecting Dependencies in Spring:-**

- via setter methods . via constructor
- **1. via setter methods:-**

  – *via referencing other beans/inner beans/wiring collections/properties*

```
<bean id="service" class="com.mycompany.service.ServiceImpl">
        <property name="timeout"><value>30</value>
        </property>
        <property name="accountDao"><ref local="accountDao"/>
        </property>
  </bean>
```

# Dependency Injection via Constructor

UnitedHealth Group

- 2. **via Constructor Injection:-**
  - *Enforces strong dependency*
  - *Bean cannot instantiate without being given all of its dependencies*
  - *Immutability*

```
public class ServiceImpl implements Service {

        private int timeout;

        private AccountDao accountDao;

        public ServiceImpl (int timeout, AccountDao accountDao) {

                this.timeout = timeout;

                this.accountDao = accountDao;

        }

}
```

# Dependency Injection via - Constructor


- 2. **via Constructor Injection:-**

```
<bean id="service" class="com.mycompany.service.ServiceImpl">
    <constructor- arg index=“0”>
        <value>30</value>
    </constructor-arg >
    <constructor- arg index=“1”>
        <ref local="accountDao"/>
    </constructor-arg >
</bean>
```

# Configuring Spring

**web.xml:-**

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
  2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app id="WebApp">

   <display-name>IAMC_WEB</display-name>

   <context-param>

         <param-name>contextConfigLocation</param-name>

         <param-value>classpath:springapp-servlet.xml</param-value>

   </context-param>

…….
```

## *spring-beans.dtd*

- Add *spring-beans.dtd* file in WEB-INF directory

## Springapp-servlet.xml

```
…
<beans>
   <!-- Create Proxy Bean -->
   <bean id="loginDAO" class="com.fid.amc.dao.LoginDAOImpl">
            <property name="sessionFactory"><ref bean="sqlMap" /></property>
   </bean>
   <bean id="loginDAOObject"
   class="org.springframework.aop.framework.ProxyFactoryBean">
            <property
   name="proxyInterfaces"><value>com.fid.amc.dao.LoginDAO</value></property>
            <property name="target"><ref bean="loginDAO" /></property>
   </bean>
</beans>
…
```

## Singleton class

```
public class AMCSpringConfiguration {
private static AMCSpringConfiguration instance=null;
private AMCSpringConfiguration(){
  applicationContext = new ClassPathXmlApplicationContext ("springapp-
  servlet.xml");}

public static AMCSpringConfiguration getInstance(){
  if(instance==null){
        synchronized(AMCSpringConfiguration.class){
              if(instance==null){
                    instance=new AMCSpringConfiguration();}
        }
  }
  return instance;
}
```

## Singleton class (ctd.)

```
public ApplicationContext getApplicationContext() {
        return applicationContext;
}
public void setApplicationContext(ApplicationContext applicationContext) {
        this.applicationContext = applicationContext;
}
}
```

## LoginDAO

```
public interface LoginDAO {

  …

  public abstract UserVO loadUserDB(String arnNumber)throws
  AMCException;

  …

  }
```

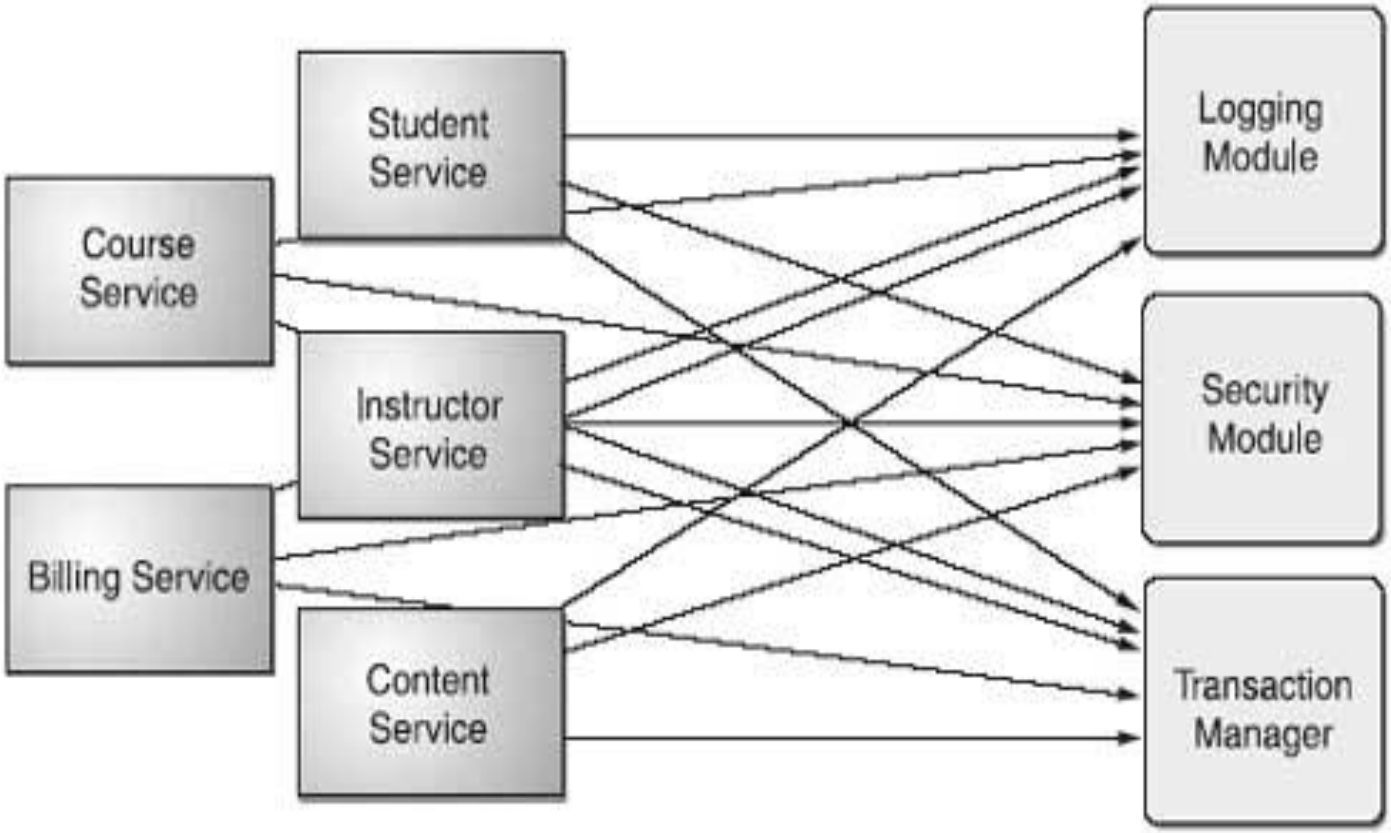
## Java Class

```
ApplicationContext applicationContext=

   AMCSpringConfiguration. getInstance(). getApplicationContext();


LoginDAO dao = (LoginDAO)
   applicationContext.getBean("loginDAOObject");


UserVO userObject=dao.loadUserDB(arnNumber);

…..
```

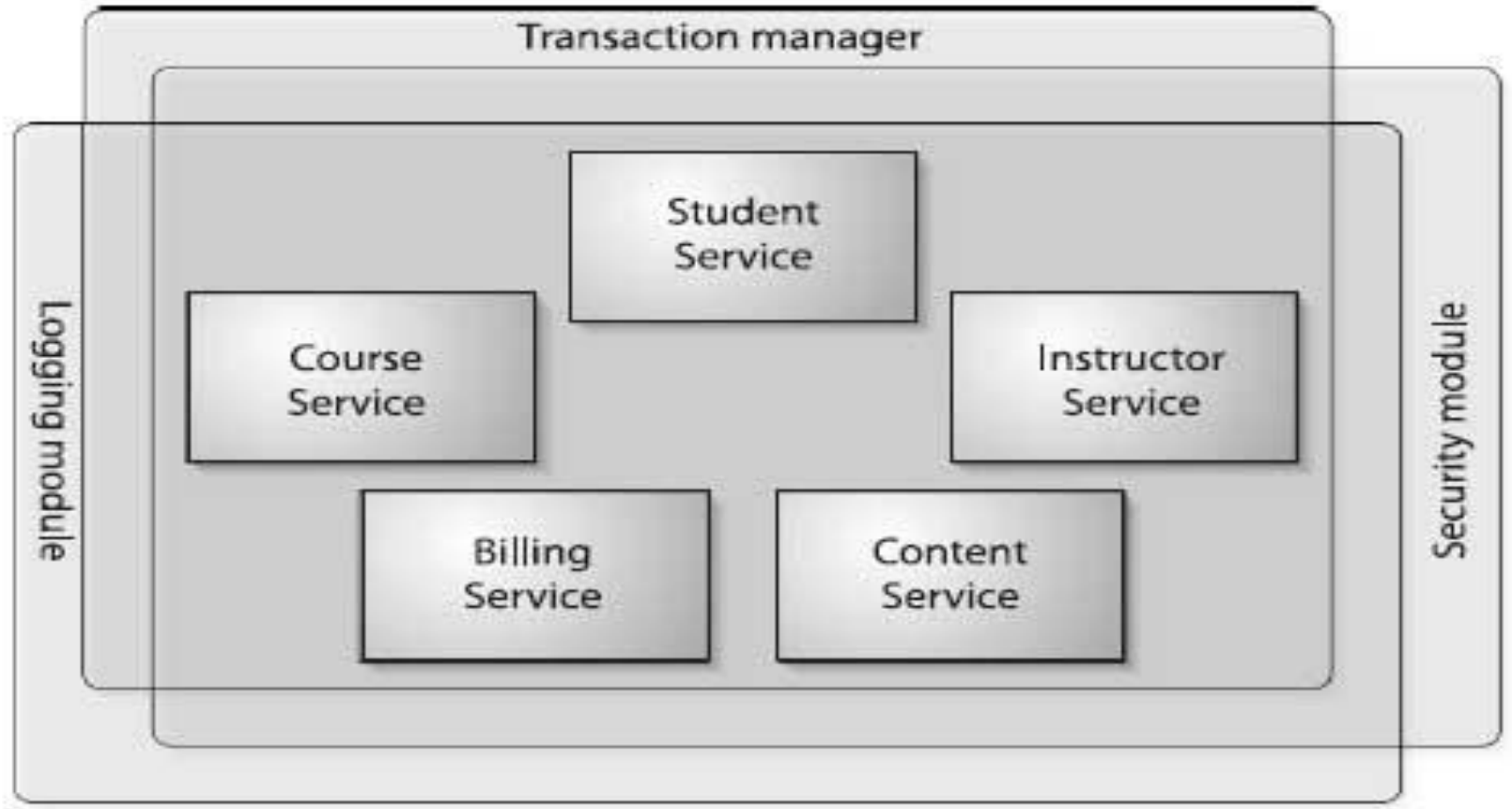
# Aspects AOP


- AOP complements IoC to deliver a non-invasive framework
- Externalizes crosscutting concerns from application code
- Concerns that cut across the structure of an object model
- AOP offers a different way of thinking about program structure to an object hierarchy
- EJB interception is conceptually similar, but not extensible and imposes too many constraints on components
- Spring provides important out-of-the box aspects
- Declarative transaction management for any POJO
- Pooling
- Resource acquisition/release

# Core Business Concerns Vs Crosscutting Enterprise concerns



# Aspects AOP

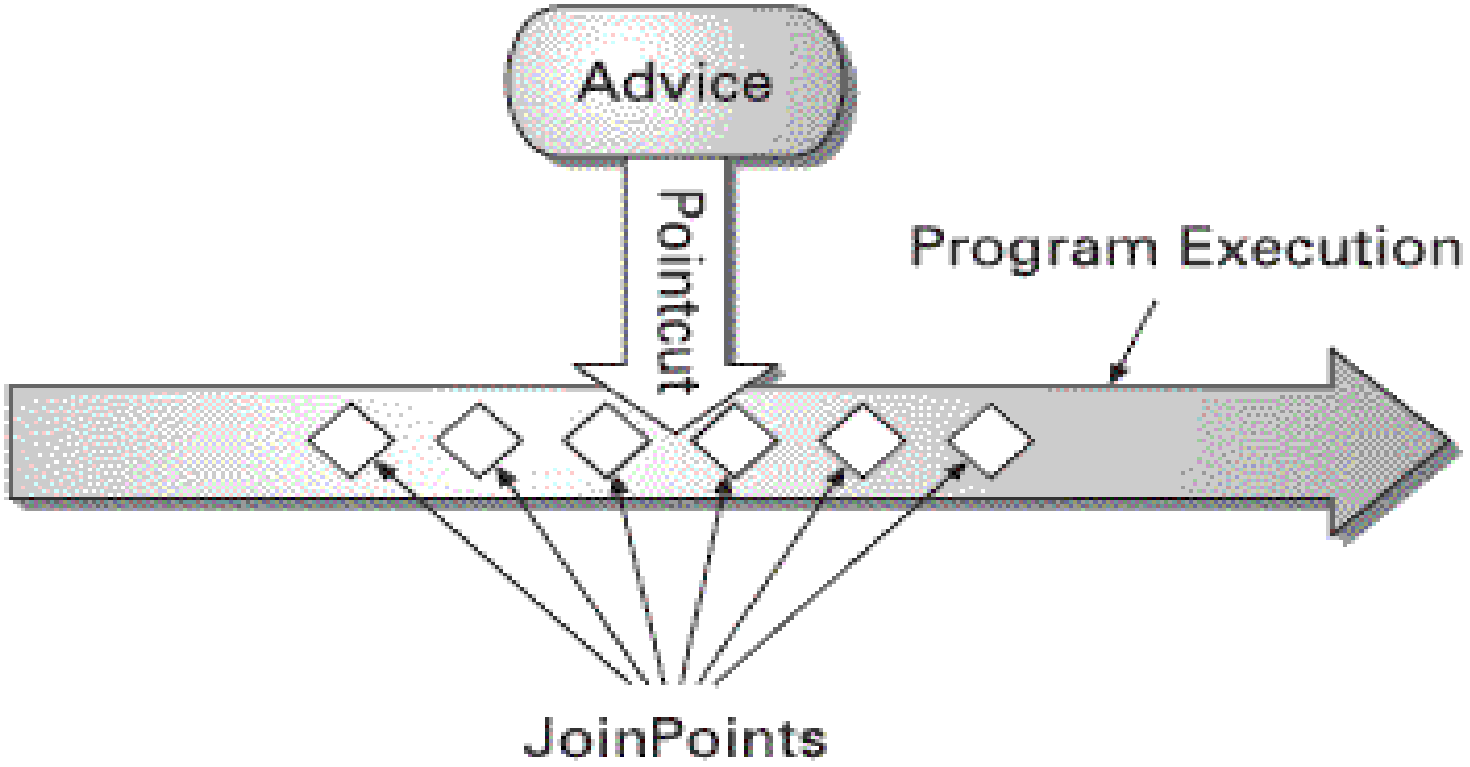


# Components Of AOP

- Aspect
  - *unit of modularity for crosscutting concerns*
- Join point
  - *well-defined points in the program flow*
- Pointcut
  - *join point queries where advice executes*
- Advice
  - *the block of code that runs based on the pointcut definition*
- Weaving
  - *can be done at runtime or compile time. Inserts the advice (crosscutting concerns) into the code (core concerns).*
- Aspects can be used as an alternative to existing technologies such as EJB.
  - *Ex: declarative transaction management, declarative security, profiling, logging, etc.*
- Aspects can be added or removed as needed without changing your code.

# AOP

- **Pointcut - defines which joinpoint gets advice**



# EJB

Spring can be used to support existing EJB as well.

EJB is a standard/specification

- Wide Industry support
- Wide adoption
- Tool ability

Complexities of EJB:-

- Writing an EJB is overly complicated
  - *At least 4 files Home, Remote, Bean Implementation and DD*
- Invasive
  - *in order to use mw services need to use javax.ejb interfaces*
- Entity Beans fall short
  - *Application entity objects are directly coupled with their persistence mechanism*

# Spring Vs EJB

| Feature | EJB | Spring |
|---|---|---|
| **Transaction Management** | JTA | PlatformTransactionManager |
| **Declarative Transaction Support** | 1. Through deployment descriptor<br>2. Cannot declaratively rollback | 1. Through spring configuration file<br>2. Can declaratively rollback. |
| **Persistence** | Support programmatic BMP and declarative CMP | Provides framework for integratibg with persistence technologies like hibernate, JDO etc. |
| **Declarative security** | Thru users and roles | 1. No implementation<br>2. Uses Acegi |
| **Distributed Computing** | Provides container managed remote calls | Provides support for calls via RMI, JAX-RPC and web services |

# Spring MVC Overview

- Spring MVC is a framework that simplifies
  - development of the Web Tier
  - simplifies testing through dependency injection
  - simplifies binding request data to domain objects
  - simplifies form validation and error handling
  - simplifies implementation of multiple view technologies
    - *JSP, Velocity, Excel, PDF, …*
  - simplifies multiple page workflow

# Spring MVC Overview

UnitedHealth Group

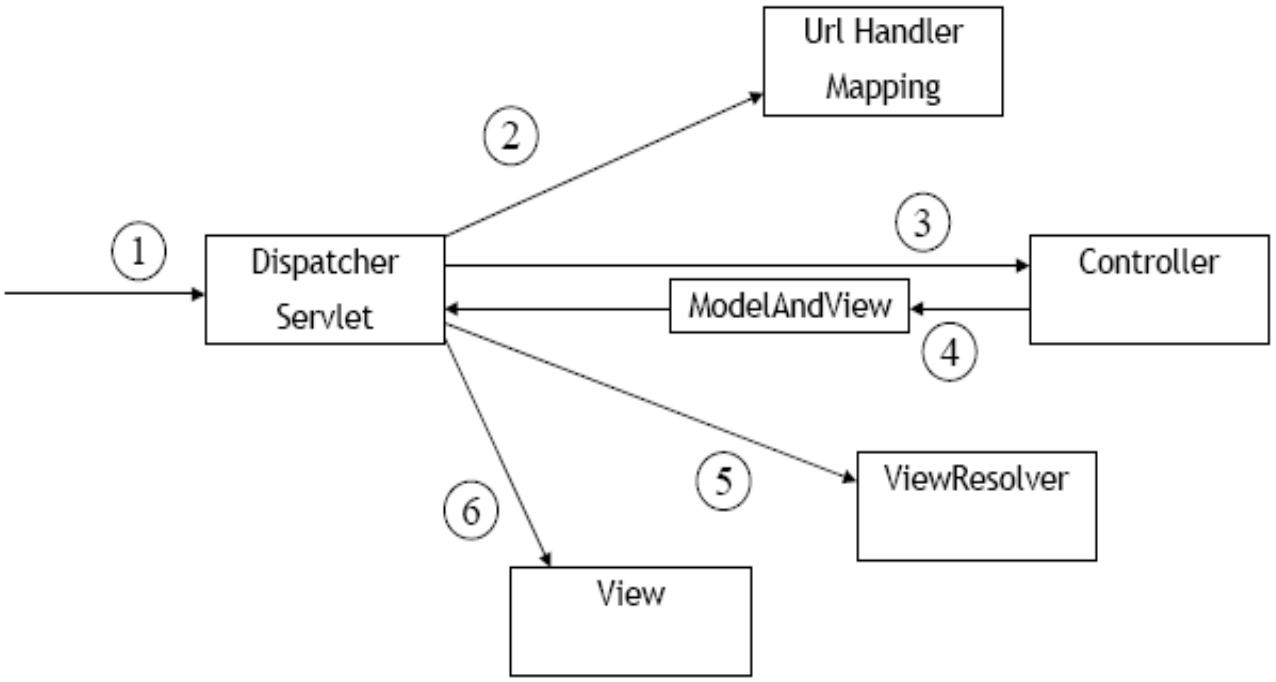
- The Spring MVC Framework offers a simple interface based infrastructure for handing web MVC architectures
- Spring MVC components are treated as first-class Spring beans
- Other Spring beans can easily be injected into Spring MVC components
- Spring MVC components are easy to test

# Spring MVC Overview

UnitedHealth Group

- **1. Controller** (org.springframework.web.servlet.mvc.Controller)
  - must implement **ModelAndView handleRequest(request,response)**
- **2. View** (org.springframework.web.servlet.mvc.View)
  - must implement **void render( model, request, response)**
- **3. Model**
  - To complete the MVC trio,
    - *note that the model is typically handled as a java.util.Map which is returned with the view*

# Request Life Cycle



# Spring MVC

- **DispatcherServlet:-**

  – Spring MVC's front controller

  – Coordinates the entire request lifecycle

  – Configured in web.xml

  – Loads Spring application context from XML file

    > default is **< servlet-name >-servlet.xml**) that usually contains <bean> definitions for the Spring MVC components

# Spring MVC


- Request moves around
  - Dispatcher servlet
  - Handler mappings
  - Controller
  - View Resolvers
- Automatically populate model objects from incoming request parameters
- Dispatcher servlet
  - first Component to receive request
  - front controller
    - which delegates responsibility for a request to other components of an application to perform actual processing

# Spring MVC Request Flow


- 1. Client sends a request
- 2. Dispatcher servlet receive the client request
- 3. Dispatcher servlet
  - *starts by querying one or more Handler mapping*
- 4. Handler mapping map URL Pattern to Controller objects
- 5. Once Dispatcher have Controller object
  - *Dispatches the request to Controller*
- 6. Controller perform little or no business logic itself or delegate to other
- 7. Upon Completion of business logic
  - *Controller returns ModelAndView Object to Dispatcher servlet*
- 8. ModelAndView
  - *contains view object or logical name of a view*
- 9. Dispatcher servlet
  - *queries ViewResolver to look up the View object*
- 10. Dispatcher servlet
  - *dispatches the request to view indicated by ModelAndView object*

# Configuring Dispatcher Servlet in web.xml

- web.xml:-

…

```
<servlet>
   <servlet-name> training</servlet-name>
   <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name> training</servlet-name>
<url-pattern>*.htm</url-pattern>
</servlet-mapping>
```

…

## Breaking up the application context

- 3 layered web application :-
  - web layer — training-servlet.xml
    - *Bean definition pertaining to controller*
  - service layer — training-service.xml
    - *Bean definition pertaining to service layer*
  - persistence layer — training-data.xml
    - *Bean definition pertaining to data layer*

- Depending on how you deploy ,you have 2 context loaders
  - *ContextLoaderListener* — *servlet 2.2 (servlet->listener)*
  - *ContextLoaderServlet* — *servlet 2.3 (listener->servlet)*

    *initialize listeners before servlet*

## Spring Configuration File

- By default spring configuration file location
  - */WEB-INF/application-Context.xml*
- Entry in web.xml:-
  - *Comma separated lists of path relative to web-application root*

```
<context-param>
<param-name>contextConfigLocation<param-name>
<param-value>/WEB-INF/training-service.xml,/WEB-INF/training-data.xml<param-value>
</context-param>
```

## Spring MVC – Build Home Page

- **Steps to build Home Page in Spring MVC:-**
  - write *Controller* class - performs logic behind homepage
  - *configure* Controller in DispatcherServlet's context configuration file (training-servlet.xml)
  - *configure* View Resolver - to tie to Controller to JSP
  - write *JSP* - render the home page to user

## Building the Controller

```
public class HomeController implements Controller{

   public ModelAndView handleRequest ( HttpServleRequest
            req,HttpServletResponse res) throws Exception {

            return ModelAndView (“home”,”message”,welcome);

   }

   private String welcome;

   public void setWelcome(String welcome){

            this.welcome=welcome;

   }

}
```

# Configuring Controller bean

training-servlet.xml:-

<bean name="/**home.htm**"

*class="com.uhg.india.training.mvc.**HomeController**" >*
*<property name="welcome" >*
*<value>Welcome to Spring Training</value>*
*</property>*
</bean>

- name attribute serves :-
  - name of the bean
  - URL Pattern for request that should be handled by HomeController bean
- Default HandlerMapping
  - *used by DispatcherServlet is BeanNameUrlHandlerMapping*
    - *Which uses base name as URL pattern*

# Declaring a View Resolver

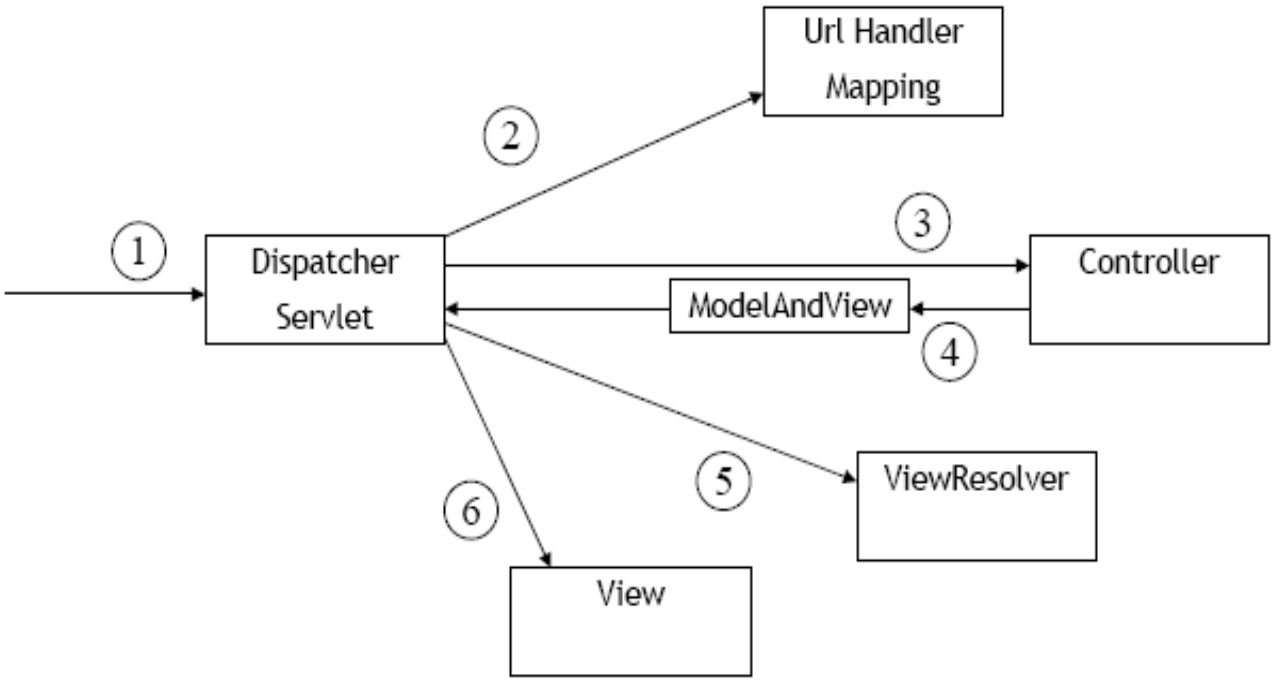
```
<bean id="viewResolver"
  Class="org.springframework.web.servlet.view.InternalResourceViewResolver">
  <property name="prefix"><value>/WEB-INF/jsp/</value></property>
  <property name="suffix"><value>.jsp/</value></property>
</bean>
```

- InternalResourceViewResolver prefixes' the view name
  - returned from ModelAndView with value from prefix and suffix property

## Creating JSP

```
<html>
<head><title>UHG India</title></head>
<body>Welcome to UHG Spring Training</body>
</html>
```

# Request Life Cycle



# Mapping Requests To Controllers

UnitedHealth Group

- Handler Mappings map a specific **controller bean ←----→URL Pattern**
  - Similar to <servlet-mapping> in web.xml
- 3 types of Spring HandlerMapping implementation:-
  - **BeanNameUrlHandlerMapping**
    - *maps controller to URLs based on controller bean name*
  - **SimpleUrlHandlerMapping**
    - *maps controller to URLs using a property collection defined in config file*
  - **CommonsPathMapHandlermapping**
    - *Maps controller to URLs using source metadata placed in controller code*
- DispatcherServlet **by default** use BeanNameUrlHandlerMapping

# BeanNameURLHandlerMapping


- Tell the **DispatcherServlet** which controller to invoke for a request
- Implement the **HandlerMapping** interface
- Spring provides several implementations

- **BeanNameUrlHandlerMapping**

```
<bean name="/index.htm"
  class="uhg.controllers.WelcomeController"> …
<bean name="/manageuhgs.htm"
  class="uhg.controllers.ManageuhgsController"> …
<bean name="/edituhg.htm"
  class="uhg.controllers.uhgFormController">
```

# BeanNameURLHandlerMapping

UnitedHealth Group

- Simple approach for mapping a **Controller ← to a → URL**

```
<bean id="beanNameUrlMapping"
   class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping" />
<bean name="/home.htm"
        class="com.uhg.india.training.mvc.HomeController" >
    <property name="welcome" >
        <value>Welcome to Spring Training</value>
    </property>
</bean>
```

- Default Handler mapping used by DispatcherServlet
  - *No need to declare explicitly in context configuration file*
- Quite simple, but creates
  - ***coupling between Presentation Layers URL and Controller Name***

# SimpleURLHandlerMapping

- map **URL Patterns directly to Controller**
  - without having to name your beans in a special way
  - Is wired with a java.util.Properties using props
- <bean id="urlMapping"

```
class="org.sfw.web.servlet.handler.SimpleUrlHandlerMapping">
<property name="mappings">
<props>
<prop key="/home.htm">HomeController</prop>
<prop key="/manageuhgs.htm">ManageUHGController</prop>
<prop key="/edituhg.htm">UHGFormController</prop>
</props>
</property>
</bean>
```

# CommonsPathHandlerMappings

- Maps controller to URLs using **source metadata** placed in controller code
  - *uses source level meta-data (a PathMap attribute) compiled into the controller using the Jakarta Commons Atttributes compiler*

```
– /**
   * @@
  org.springframework.web.servlet.handler.commonsattributes.PathMap
  (“/displayProvider.html)
  */
public class DisplayProviderController extends
  AbstractCommandController{
...
}
```

# Working With Multiple HandlerMapping

UnitedHealth Group

- can mix and match handler mappings
- All HandlerMapping implements Spring's Ordered interface
  - *Set their order property to indicate which has precedence with relation to others*
  - *Lower the value of order property the higher the priority*

```
<bean id="beanNameUrlMapping" class="">
  <property name="order"><value>1</value></property-name>
</bean>
<bean id="simpleUrlMapping" class="">
  <property name="order"><value>1</value></property-name>
  <property name="mappings">…</property>
</bean>
```

# Controllers

- Spring provides many implementations:
  - **AbstractController**
  - **AbstractCommandController**
  - **ParameterizableViewController**
  - **MultiActionController**
  - **SimpleFormController**
  - **AbstractWizardController**

# Controllers

- **DispatcherServlet**

  –*Receive requests from and coordinate business functionality*

- implement the **Controller** interface

  public ModelAndView handleRequest(HttpServletRequest request,HttpServletResponse response) throws Exception;

- return instance of **ModelAndView** to DispatcherServlet

- ModelAndView

  – **contains** the model (a **Map**) and

    > either a logical view name,

    > or an implementation of the **View** interface

# View Resolvers

- Resolve logical view names returned from controllers into View objects
- Implement the **ViewResolver** interface
- Spring provides several implementations
- UrlBasedViewResolver

```
<bean id="viewResolver"
class="org.springframework.web.servlet.view.UrlBasedViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/"/>
        <property name="suffix" value=".jsp"/>
</bean>
```

# BeanNameViewResolvers

```
<bean id="viewResolver"
        class="org.sfw.web.servlet.view.BeanNameViewResolver"/>
<bean id="manageuhgs"
        class="org.springframework.web.servlet.view.JstlView">
            <property name="url">
                <value>/WEB-INF/jsp/manageuhgs.jsp</value>
            </property>
</bean>
<bean id="manageuhgsAsExcel“
        class="uhg.views.ManageuhgsExcelView"/>
```

# ResourceBundleViewResolver

- **ResourceBundleViewResolver**

```
<bean id="viewResolver"
        class="org.sfw.web.servlet.view.ResourceBundleViewResolver">
        <property name="basename">
                <value>views</value>
        </property>
</bean>
```

**views.properties**

```
Welcome (class)=org.springframework.web.servlet.view.JstlView
welcome.url=/WEB-INF/jsp/welcome.jsp
```

# XmlViewResolvers


```
<bean id="viewResolver"
class="org.sfw.web.servlet.view.XmlViewResolver">
<property name="location"><value>views</value></property>
</bean>
```

**views.xml**

```
<bean name="welcome"
class="org.springframework.web.servlet.view.JstlView">
<property name="url">
<value>/WEB-INF/jsp/welcome.jsp</value>
</property>
</bean>
```

# View

- Renders the output of the request to the client
- Implement the **View** interface
- Built in support for:

  JSP, XSLT, Velocity, Freemarker, Excel, PDF, JasperReports