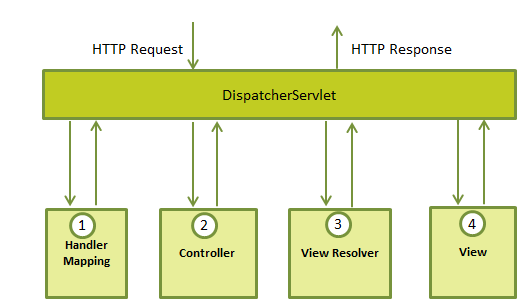
**Spring Framework**

****

* The Spring framework comprises several modules such as IOC, AOP, DAO, Context, ORM, WEB MVC etc.

First writing web.xml and then in web.xml we are pointing spring-servlet.xml using below

<context-param>

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

<param-value>/WEB-INF/HelloWeb-servlet.xml</param-value>

</context-param>

<listener>

<listener-class>

**org.springframework.web.context.ContextLoaderListener**

</listener-class>

</listener>

**@Controller**

public class HelloController{

**@RequestMapping(value = "/hello", method = RequestMethod.GET)**

public String printHello(ModelMap model) {

model.addAttribute("message", "Hello Spring MVC Framework!");

return "hello";

}

}

**Dependency Injection**

public interface WildAnimal {

public String sound() ;

}

public class Lion implements WildAnimal {

public String sound() {

return "Roar";

}

}

@Service

public class Zoo {

private WildAnimal wild;

@Autowired

public void setWild(WildAnimal wild) {

this.wild = wild;

}

public void testSound() {

System.out.println(wild.sound());

}

}

**Dependency Injection example**

<bean id="wild" class="com.javapapers.spring.ioc.Wolf" />

<bean id="zoo" class="com.javapapers.spring.ioc.Zoo">

<property name="wild" ref="wild" />

</bean>

**Getter/Setter dependency injection**

public class Bar {

private Foo foo;

public void setFoo(Foo foo){

this.foo = foo;

}

public String toString(){

return "Bar! Foo : \n" + foo;

} }

<bean id="barBean" class="com.javacodegeeks.snippets.enterprise.services.Bar">

<property name="foo">

<ref bean="fooBean" />

</property>

</bean>

--------- ----------------------------------------------------------------------- -----------------------------------------------------

**Constructor dependency Injection**

1. **Driver.java**

package com.spring.construct;

public class Driver {

private int code;

private String name;

public int getCode() {

return code;

}

public void setCode(int code) {

this.code = code;

}

public void setName(String name) {

this.name = name;

}

public String getName() {

return name;

}

public Driver() {

}

public Driver(int code, String name) {

this.code = code;

this.name = name;

}

@Override

public String toString() {

return "Driver Code: " + this.code + "\nDriver Name: " + this.name;

}

}

1. **Car.java**

package com.spring.construct;

public class Car {

private String name;

private Driver driver;

public Driver getDriver() {

return driver;

}

public void setDriver(Driver driver) {

this.driver = driver;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public Car() {

}

public Car(String name, Driver driver) {

this.name = name;

this.driver = driver;

}

@Override

public String toString() {

return "Car Name: " + this.name + "\n" + this.driver;

}

}

1. **Cardriver.xml**

<bean id="driver" class="com.spring.construct.Driver">

<constructor-arg name="code" value="3263" />

<constructor-arg name="name" value="Michael Schumacher" />

</bean>

<bean id="car" class="com.spring.construct.Car">

<constructor-arg name="name" value="Ferrari" />

<constructor-arg name="driver" ref="driver" />

</bean>

**Main class**

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class CarDriverImpl {

    public static void main(String[] args) {

        ApplicationContext context = new ClassPathXmlApplicationContext(

                "com/spring/construct/carDriver.xml");

        Car car = (Car) context.getBean("car");

        System.out.println(car);

    }

}

**Output**

Car Name: Ferrari

Driver Code: 3263

Driver Name: Michael Schumacher

**Spring – Autowiring**

Spring, “**Autowiring by Name**” means, if the name of a bean is same as the name of other bean property, auto wire it.

Autowiring if one bean value has same property with the other bean we can use this by auto wiring.

#### ****Advantages of Autowiring****

**1.** Reduces or eliminates the use of <property> tag.  
**2.** Better support for spring autodiscovery.

public class **Address** {

@Override

public String toString() {

return "Address [streetNo=" + streetNo + ", houseNo=" + houseNo

+ ", city=" + city + ", state=" + state + "]";

}

String streetNo;

String houseNo;

String city;

String state;

public String getStreetNo() {

return streetNo;

}

public void setStreetNo(String streetNo) {

this.streetNo = streetNo;

}

public String getHouseNo() {

return houseNo;

}

public void setHouseNo(String houseNo) {

this.houseNo = houseNo;

}

public String getCity() {

return city;

}

public void setCity(String city) {

this.city = city;

}

public String getState() {

return state;

}

public void setState(String state) {

this.state = state;

}

}

import org.springframework.stereotype.Component;

@Component

public class **Company** {

/\*

\* (non-Javadoc)

\*

\* @see java.lang.Object#toString()

\*

\* Method is Overriden to provide proper output while using in sysout.

\*/

@Override

public String toString() {

return "Company [companyName=" + companyName + ", companyCEO="

+ companyCEO + ", address=" + address + "]";

}

String companyName;

String companyCEO;

/\*

\* This property will be Autowired by Name. Spring will search for any bean

\* with name as "address".

\*/

Address address;

/\*

\* Note: Setter and Getter method should be in the form of bean rules. If

\* you write setter & getter methods in other format Autowire will not work

\* as expected.

\*/

public Address getAddress() {

return address;

}

public void setAddress(Address address) {

this.address = address;

}

public String getCompanyName() {

return companyName;

}

public void setCompanyName(String companyName) {

this.companyName = companyName;

}

public String getCompanyCEO() {

return companyCEO;

}

public void setCompanyCEO(String companyCEO) {

this.companyCEO = companyCEO;

}

}

**Springbean.xml**

<bean id="address" class="jbt.bean.Address">

<property name="streetNo" value="2nd Street" />

<property name="houseNo" value="345" />

<property name="city" value="Mumbai" />

<property name="state" value="Maharastra" />

</bean>

<!-- Note: If you specify the property in a Bean you have to provide either

ref or value otherwise Spring will throw an Exception like Below

"Configuration problem: <property> element for property 'address' must specify a ref or value"

-->

<bean id="company" class="jbt.bean.Company" autowire="byName">

<property name="companyCEO" value="VN Gautam" />

<property name="companyName" value="JBT Company" />

</bean>

**Output**

Company [companyName=JBT Company, companyCEO=VN Gautam, address=Address [streetNo=2nd Street, houseNo=345, city=Mumbai, state=Maharastra]]

**@Qualifier** is useful for the situation where you have more than one bean matching the type of dependency and thus resulting in ambiguity.

1. **package com.websystique.spring.domain;**

public interface Car {

    public void getCarName();

}

1. **import org.springframework.stereotype.Component;**

@Component("Ferari")

public class Ferari implements Car{

    public void getCarName() {

        System.out.println("This is Ferari");

    }

}

1. **import org.springframework.stereotype.Component;**

@Component("Mustang")

public class Mustang implements Car{

    public void getCarName() {

        System.out.println("This is Mustang");

    }

}

1. **import org.springframework.beans.factory.annotation.Autowired;**

import org.springframework.beans.factory.annotation.Qualifier;

import org.springframework.stereotype.Component;

@Component

public class Bond {

    @Autowired

 @Qualifier("Mustang")

    private Car car;

    public void showCar(){

        car.getCarName();

    }

}

1. **import org.springframework.context.annotation.AnnotationConfigApplicationContext**;

import org.springframework.context.support.AbstractApplicationContext;

import com.websystique.spring.config.AppConfig;

import com.websystique.spring.domain.Bond;

public class AppMain {

    public static void main(String args[]) {

        AbstractApplicationContext context = new AnnotationConfigApplicationContext(

                AppConfig.class);

        Bond bond = (Bond) context.getBean("bond");

        bond.showCar();

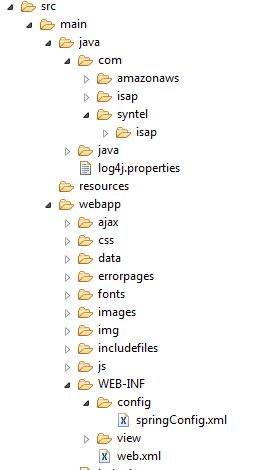
    }

}

**Output**

This is Mustang

**Real time flow for spring**



**Web.xml**

<servlet>

<servlet-name>myBatisServlet</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

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

<param-value>/WEB-INF/config/springConfig.xml</param-value>

</init-param>

</servlet>

**Spring-config.xml**

<bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<property name=*"prefix"* value=*"/WEB-INF/view/"* />

<property name=*"suffix"* value=*".jsp"* />

</bean>

<bean id=*"dataSource"* destroy-method=*"close"*

class=*"org.apache.commons.dbcp.BasicDataSource"*>

<property name=*"driverClassName"* value=*"com.mysql.jdbc.Driver"*/>

<property name=*"url"* value=*"jdbc:mysql://192.168.175.68:3306/isap\_env\_prov"* />

<property name=*"username"* value=*"root"*/>

<property name=*"password"* value=*"Syntel123$"*/>

<property name=*"initialSize"* value=*"10"*/>

</bean>

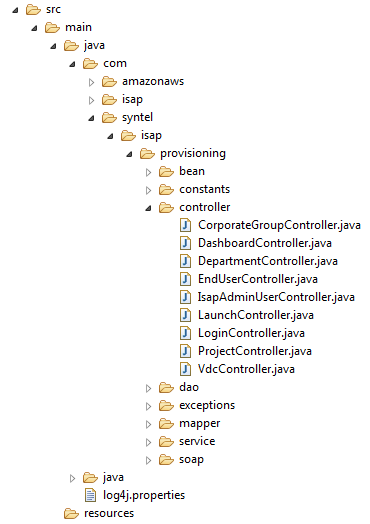
<bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>

<property name=*"dataSource"* ref=*"dataSource"* />

<property name=*"typeAliasesPackage"* value=*"com.syntel.isap.provisioning.bean"*/>

<property name=*"mapperLocations"* value=*"classpath\*:com/syntel/isap/provisioning/mapper/\*.xml"* />

</bean>



* **Controller class**

@RequestMapping(value="/corporategroups", method=RequestMethod.GET)

**public** ModelAndView corporateGroupLists(HttpSession session) {

List<CorporateGroups> corporateGroupList = **new** ArrayList<CorporateGroups>();

String view = "admin/corporateGroups";

LOGGER.info("Inside CorporateGroups GET");

ModelAndView model = **new** ModelAndView();

User user= (User) session.getAttribute("userValue");

corporateGroupList = corporateGroupService.getcorporateGroupList();

LOGGER.info("user Session Values"+user.getCg\_id());

model.addObject("corpgrplist", corporateGroupList);

model.setViewName(view);

**return** model;

}

@RequestMapping(value="/getCorpUsers", method = RequestMethod.GET)

**public** @ResponseBody List<User> getCorpUsersInJSON(HttpSession session) {

List<User> usersList = **new** ArrayList<User>();

User user=**new** User();

usersList = corporateGroupService.getUsersByCgIdAndRoleID(user);

**return** usersList;

}

**Core Java**

* Interator (ArrayList)and enumerator (Vector) both are user to traverse the object, but iterator can be used to add and remove the objects, but enumerator is used only for read only.