**Using MethodInvokingFactoryBean to call setters with multiple arguments**

This can be used for following purposes.

* To execute static method of a class and assign return value to a bean id in the config.
* To execute instance method of a class and assign return value to a bean id in the config.
* To execute a static or instance method which does not return a value. (void return type)

A static target method may be specified by setting the targetMethod property to a String representing the static method name, with targetClass specifying the Class that the static method is defined on.

Alternatively, a target instance method may be specified, by setting the targetObjectproperty as the target object, and the targetMethod property as the name of the method to call on that target object.

Arguments for the method invocation may be specified by setting the arguments property.

An example of calling a static method and then an instance method to get at a Java system

<bean id="sysProps" class="org.springframework.beans.factory.config.MethodInvokingFactoryBean">

<property name="targetClass" value="java.lang.System"/>

<property name="targetMethod" value="getProperties"/>

</bean>

<bean id="javaVersion" class="org.springframework.beans.factory.config.MethodInvokingFactoryBean">

<property name="targetObject" ref="sysProps"/>

<property name="targetMethod" value="getProperty"/>

<property name="arguments" value="java.version"/>

</bean>

**package**test;  
**public class**SomeObject **implements**Interface\_A {  
  **private**String str1;  
  **private**String str2;  
  **public void**setPairOfStrings(String str1, String str2) {  
    **this**.str1 = str1;  
    **this**.str2 = str2;  
  }

**public void**businessMethod() {  
    System.out.println("Inside - SomeObject - Values are: Str1 = " + str1 + " Str2 = "  
        + str2);  
  }  
  
}

In order to populate attributes str1 and str2 we can use the following XML:

<!-- This is the object we need to instantiate -->

<bean id="someobject" class="test.SomeObject"/>

<!-- This is the config that calls the double arg setter method: setPairOfStrings -->

<bean class="org.springframework.beans.factory.config.MethodInvokingFactoryBean">

<property name="targetObject">

<ref local="someobject"/>

</property>

<property name="targetMethod">

<value>setPairOfStrings</value>

</property>

<property name="arguments">

<list>

<value>String1</value>

<value>String2</value>

</list>

</property>

</bean>

How To use ApplicationContext in simple servlet application

1. applicationContext.xml:

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

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-2.5.xsd"

default-lazy-init="true">

<context:annotation-config />

<bean id="springWebApp" class="com.javarticles.web.SpringWebAppContextLoaderExample" scope="singleton"/>

</beans>

Web-app.xml

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

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

version="2.4"

xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee

http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd">

**<context-param>**

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

**<param-value>**

**/WEB-INF/applicationContext.xml /WEB-INF/database1Context.xml /WEB-INF/database2Context.xml**

**</param-value>**

**</context-param>**

<listener>

<listener-class>

org.springframework.web.util.Log4jConfigListener

</listener-class>

</listener>

**<listener>**

**<listener-class>**

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

**</listener-class>**

**</listener>**

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

</web-app>

In servlet class

**ApplicationContext context = WebApplicationContextUtils.getRequiredWebApplicationContext(this.getServletContext());**

[**http://www.knowledgewalls.com/j2ee/books/spring-30-examples/how-to-use-xmlwebapplicationcontext-in-spring-framework-with-example**](http://www.knowledgewalls.com/j2ee/books/spring-30-examples/how-to-use-xmlwebapplicationcontext-in-spring-framework-with-example)

[**http://www.javarticles.com/2015/04/spring-web-application-context-example.html**](http://www.javarticles.com/2015/04/spring-web-application-context-example.html)

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.springframework.web.context.support.XmlWebApplicationContext;

@WebServlet("/MyBeanReader")

public class MyBeanReader extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

        XmlWebApplicationContext context = new XmlWebApplicationContext();

            context.setConfigLocation("/WEB-INF/beans.xml");

            context.setServletContext(request.getServletContext());

            context.refresh();

        SayHello hello = (SayHello) context.getBean("hello");

            hello.sayGoodMorning();

            hello.sayGoodEvening();

            hello.sayGoodNight();

    }

}

<?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-3.0.xsd>">

    <bean id="hello" class="SayHello" />

</beans>

**PropertyPlaceholderConfigurer**

<bean id="appProperties"

class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">

<property name="locations">

<list>

<value>classpath:mail.properties</value>

<value>classpath:database.properties</value>

</list>

</property>

</bean>

By default, if a placeholder could not be resolved by the specified properties files, Spring will try to resolve it with a system property. For example, if Spring could not resolve the placeholder ${user.dir}, it will try with the corresponding system property user.dir. This is call system properties mode fallback. We can change this behavior by specifying a property called systemPropertiesModeName of the PropertyPlaceholderConfigurerbean:

<property name="systemPropertiesModeName" value="SYSTEM\_PROPERTIES\_MODE\_FALLBACK" />

**Spring provides three modes as follows:**

**SYSTEM\_PROPERTIES\_MODE\_FALLBACK**: This is the default mode as mentioned above.

**SYSTEM\_PROPERTIES\_MODE\_OVERRIDE**: In this mode, Spring will resolve the placeholders to system properties first. If a system property does exist, its value will override the value in the properties file.

**SYSTEM\_PROPERTIES\_MODE\_NEVER**: Spring will not take system properties into consideration when resolving the placeholders.

By default, Spring will throw an exception if it could not find a properties file or could not resolve a placeholder. That cause the application fails to start.

To ignore the exception which will be thrown in case a properties file could not be found, specify the following property of the PropertyPlaceholderConfigurerbean:

**<property name="ignoreResourceNotFound" value="true" />**

And to ignore the exception which will be thrown in case a placeholder could not be resolved, specify the following property of the PropertyPlaceholderConfigurerbean:

**<property name="ignoreUnresolvablePlaceholders" value="true" />**

**BeanPostProcessor**

Call back methods are executed before and after initialization of bean implementing interface BeanPostProcessor

public Object postProcessAfterInitialization(Object bean,String beanName) throws BeansException {return bean;}

public Object postProcessBeforeInitialization(Object bean,String beanName) throws BeansException{return bean;}

Classes implementing BeanPostProcessor has to be registered as bean in applicationConext.xml

**BeanFactoryPostProcessor**

Call back method is executed after initialization of ApplicationContextFactory then beans are getting created

public void postProcessBeanFactory(ConfigurableListableBeanFactory beanFactory) throws BeansException {}

Classes implementing BeanFactoryPostProcessor has to be registered as bean in applicationConext.xml

PropertyPlaceHolderConfigurer is an example of BeanFactoryPostProcessor

**ThreadPoolTaskExecutor**

<bean id="taskExecutor"

class="org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor">

<property name="corePoolSize"

value="${thread.core.pool.size}" />

<property name="maxPoolSize"

value="${thread.max.pool.size}" />

<property name="WaitForTasksToCompleteOnShutdown" value="true" />

</bean>

execute(class implementing runnable interface)