Spring IoC Container and Spring Bean Example Tutorial

Spring Framework is built on the **Inversion of Control (IOC)** principle. Dependency injection is the technique to implement IoC in applications. This article is aimed to explain core concepts of Spring IoC container and Spring Bean with example programs.

Spring IoC Container

Inversion of Control is the mechanism to achieve loose-coupling between Objects dependencies. To achieve loose coupling and dynamic binding of the objects at runtime, the objects define their dependencies that are being injected by other assembler objects. Spring IoC container is the program that *injects* dependencies into an object and make it ready for our use. We have already looked how we can use Spring Dependency Injection to implement IoC in our applications.

Spring Framework IoC container classes are part of `org.springframework.beans` and `org.springframework.context` packages and provides us different ways to decouple the object dependencies.

`BeanFactory` is the root interface of Spring IoC container. `ApplicationContext` is the child interface of `BeanFactory` interface that provide Spring's AOP features, internationalization etc. Some of the useful child-interfaces of `ApplicationContext` are `ConfigurableApplicationContext` and `WebApplicationContext`. Spring Framework provides a number of useful ApplicationContext implementation classes that we can use to get the context and then the Spring Bean.

Some of the useful ApplicationContext implementations that we use are;

- **AnnotationConfigApplicationContext**: If we are using Spring in standalone java applications and using annotations for Configuration, then we can use this to initialize the container and get the bean objects.

- **ClassPathXmlApplicationContext**: If we have spring bean configuration xml file in standalone application, then we can use this class to load the file and get the container object.
- **FileSystemXmlApplicationContext**: This is similar to ClassPathXmlApplicationContext except that the xml configuration file can be loaded from anywhere in the file system.
- **AnnotationConfigWebApplicationContext** and **XmlWebApplicationContext** for web applications.

Usually if you are working on Spring MVC application and your application is configured to use Spring Framework, Spring IoC container gets initialized when application starts and when a bean is requested, the dependencies are injected automatically.

However for standalone application, you need to initialize the container somewhere in the application and then use it to get the spring beans.

Spring Bean

Spring Bean is nothing special, any object in the Spring framework that we initialize through Spring container is called Spring Bean. Any normal Java POJO class can be a Spring Bean if it's configured to be initialized via container by providing configuration metadata information.

Spring Bean Scopes

There are five scopes defined for Spring Beans.

1. **singleton** – Only one instance of the bean will be created for each container. This is the default scope for the spring beans. While using this scope, make sure bean doesn't have shared instance variables otherwise it might lead to data inconsistency issues.
2. **prototype** – A new instance will be created every time the bean is requested.
3. **request** – This is same as prototype scope, however it's meant to be used for web applications. A new instance of the bean will be created for each HTTP request.
4. **session** – A new bean will be created for each HTTP session by the container.
5. **global-session** – This is used to create global session beans for Portlet applications.

Spring Framework is extendable and we can create our own scopes too, however most of the times we are good with the scopes provided by the framework.

Spring Bean Configuration

Spring Framework provide three ways to configure beans to be used in the application.

1. **Annotation Based Configuration** – By using @Service or @Component annotations. Scope details can be provided with @Scope annotation.

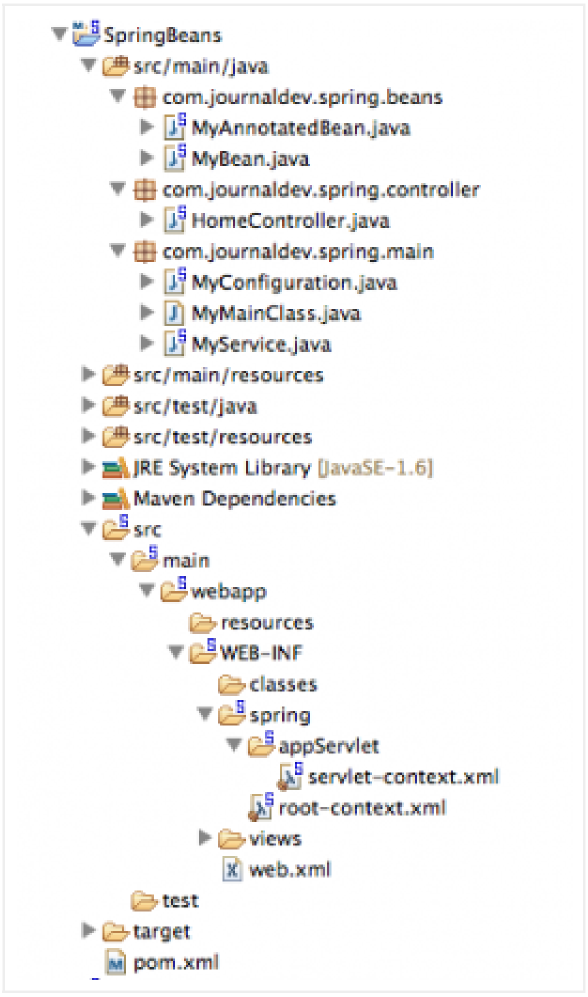
2. **XML Based Configuration** – By creating Spring Configuration XML file to configure the beans. If you are using Spring MVC framework, the xml based configuration can be loaded automatically by writing some boiler plate code in web.xml file.
3. **Java Based Configuration** – Starting from Spring 3.0, we can configure Spring beans using java programs. Some important annotations used for java based configuration are @Configuration, @ComponentScan and @Bean.

Spring IoC and Bean Example Project

Let's look at the different aspects of Spring IoC container and Spring Bean configurations with a simple Spring project.

For my example, I am creating Spring MVC project in Spring Tool Suite. If you are new to Spring Tool Suite and Spring MVC, please read Spring MVC Tutorial with Spring Tool Suite.

The final project structure looks like below image.



```
SpringBeans
  src/main/java
    com.journaldev.spring.beans
      MyAnnotatedBean.java
      MyBean.java
    com.journaldev.spring.controller
      HomeController.java
    com.journaldev.spring.main
      MyConfiguration.java
      MyMainClass.java
      MyService.java
  src/main/resources
  src/test/java
  src/test/resources
  JRE System Library [JavaSE-1.6]
  Maven Dependencies
  src
    main
      webapp
        resources
        WEB-INF
          classes
          spring
            appServlet
              servlet-context.xml
            root-context.xml
          views
          web.xml
    test
  target
  pom.xml
```

Let's look at different components one by one.

XML Based Bean Configuration

MyBean is a simple Java POJO class.

MyBean.java

```java
package com.journaldev.spring.beans;

public class MyBean {

    private String name;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

}
```

Spring Configuration XML File

servlet-context.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/mvc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:beans="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc http://www.s
        http://www.springframework.org/schema/beans http://www.springframework
        http://www.springframework.org/schema/context http://www.springframewo

    <!-- DispatcherServlet Context: defines this servlet's request-processing

    <!-- Enables the Spring MVC @Controller programming model -->
    <annotation-driven />

    <!-- Handles HTTP GET requests for /resources/** by efficiently serving up
    <resources mapping="/resources/**" location="/resources/" />

    <!-- Resolves views selected for rendering by @Controllers to .jsp resourc
    <beans:bean class="org.springframework.web.servlet.view.InternalResourceVi
        <beans:property name="prefix" value="/WEB-INF/views/" />
        <beans:property name="suffix" value=".jsp" />
    </beans:bean>

    <context:component-scan base-package="com.journaldev.spring" />

    <beans:bean name="myBean" class="com.journaldev.spring.beans.MyBean" scope

</beans:beans>
```

Notice that MyBean is configured using *bean* element with scope as singleton.

Annotation Based Bean Configuration

MyAnnotatedBean.java

```java
package com.journaldev.spring.beans;

import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Service;
import org.springframework.web.context.WebApplicationContext;
```

```java

@Service
@Scope(WebApplicationContext.SCOPE_REQUEST)
public class MyAnnotatedBean {

    private int empId;

    public int getEmpId() {
        return empId;
    }

    public void setEmpId(int empId) {
        this.empId = empId;
    }

}
```

MyAnnotatedBean is configured using @Service and scope is set to Request.

Controller Class

HomeController class will handle the HTTP requests for the home page of the application. We will inject our Spring beans to this controller class through WebApplicationContext container.

HomeController.java

```java
package com.journaldev.spring.controller;

import java.text.DateFormat;
import java.util.Date;
import java.util.Locale;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import com.journaldev.spring.beans.MyAnnotatedBean;
import com.journaldev.spring.beans.MyBean;

@Controller
@Scope("request")
public class HomeController {

    private MyBean myBean;

    private MyAnnotatedBean myAnnotatedBean;

    @Autowired
    public void setMyBean(MyBean myBean) {
        this.myBean = myBean;
    }

    @Autowired
    public void setMyAnnotatedBean(MyAnnotatedBean obj) {
        this.myAnnotatedBean = obj;
    }

    /**
```

```
     * Simply selects the home view to render by returning its name.
     */
    @RequestMapping(value = "/", method = RequestMethod.GET)
    public String home(Locale locale, Model model) {
        System.out.println("MyBean hashcode="+myBean.hashCode());
        System.out.println("MyAnnotatedBean hashcode="+myAnnotatedBean.hashCo

        Date date = new Date();
        DateFormat dateFormat = DateFormat.getDateTimeInstance(DateFormat.LON

        String formattedDate = dateFormat.format(date);

        model.addAttribute("serverTime", formattedDate );

        return "home";
    }

}
```

Deployment Descriptor

We need to configure our application for Spring Framework, so that the configuration metadata will get loaded and context will be initialized.

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com

    <!-- The definition of the Root Spring Container shared by all Servlets ar
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring/root-context.xml</param-value>
    </context-param>

    <!-- Creates the Spring Container shared by all Servlets and Filters -->
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener
    </listener>

    <!-- Processes application requests -->
    <servlet>
        <servlet-name>appServlet</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</serv
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>appServlet</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

</web-app>
```

Almost all the configuration above is boiler-plate code generated by STS tool automatically.

Run the Web Application

Now when you will launch the web application, the home page will get loaded and in the console following logs will be printed when you refresh the page multiple times.

```
MyBean hashcode=118267258
MyAnnotatedBean hashcode=1703899856
MyBean hashcode=118267258
MyAnnotatedBean hashcode=1115599742
MyBean hashcode=118267258
MyAnnotatedBean hashcode=516457106
```

Notice that MyBean is configured to be singleton, so the container is always returning the same instance and hashcode is always same. Similarly for each request, a new instance of MyAnnotatedBean is created with different hashcode.

Java Based Bean Configuration

For standalone applications, we can use annotation based as well as xml based configuration. The only requirement is to initialize the context somewhere in the program before we use it.

MyService.java

```java
package com.journaldev.spring.main;

import java.util.Date;

public class MyService {

    public void log(String msg){
        System.out.println(new Date()+"::"+msg);
    }
}
```

MyService is a simple java class with some methods.

MyConfiguration.java

```java
package com.journaldev.spring.main;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan(value="com.journaldev.spring.main")
public class MyConfiguration {

    @Bean
    public MyService getService(){
        return new MyService();
    }
}
```

The annotation based configuration class that will be used to initialize the Spring container.

MyMainClass.java

```
package com.journaldev.spring.main;

import org.springframework.context.annotation.AnnotationConfigApplicationConte

public class MyMainClass {

    public static void main(String[] args) {

        AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicat
                MyConfiguration.class);
        MyService service = ctx.getBean(MyService.class);

        service.log("Hi");

        MyService newService = ctx.getBean(MyService.class);
        System.out.println("service hashcode="+service.hashCode());
        System.out.println("newService hashcode="+newService.hashCode());
        ctx.close();
    }

}
```

A simple test program where we are initializing the `AnnotationConfigApplicationContext` context and then using *getBean()* method to get the instance of MyService.

Notice that I am calling getBean method two times and printing the hashcode. Since there is no scope defined for MyService, it should be singleton and hence hashcode should be the same for both the instances.

When we run the above application, we get following console output confirming our understanding.

```
Sat Dec 28 22:49:18 PST 2013::Hi
service hashcode=678984726
newService hashcode=678984726
```

If you are looking for XML based configuration, just create the Spring XML config file and then initialize the context with following code snippet.

```
ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext(
                "applicationContext.xml");
        MyService app = context.getBean(MyService.class);
```

That's all for the Spring IoC container and Spring Bean Scopes and Configuration details. We will look into some more features of Spring Beans in future posts. Download the Spring Bean example project from below link and play around with it for better understanding.

Download Spring Beans Project
961 downloads