آموزش Spring

مولف:مهندس افشین رفوآ

spring

Framework

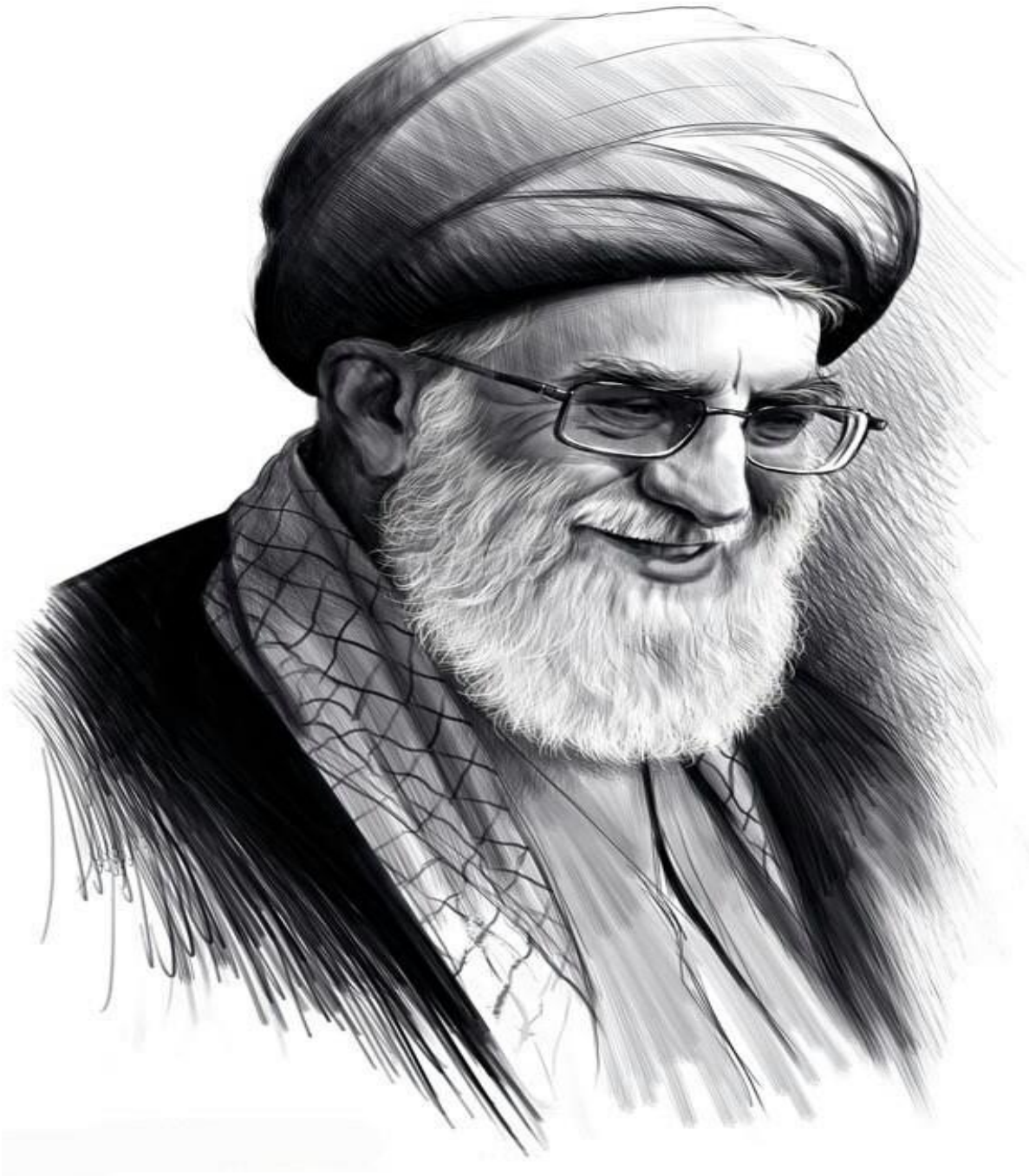
بسم الله الرحمن الرحیم

آموزشگاه تحلیل داده

تخصصی ترین مرکز برنامه نویسی و دیتا بیس در ایران

کتاب آموزش Spring

نویسنده : مهندس افشین رفوآ



تقدیم به نائب امام عصر، حضرت آیت الله خامنه ای که عصا زدنش ضرب آهنگ حیدری دارد

آدرس آموزشگاه : تهران - خیابان شریعتی - بالا تر از خیابان ملک - جنب بانک صادرات - پلاک 651 طبقه دوم - واحد7
88146323 - 88446780 - 88146330

تقدیم به همه ی جویندگان علم که توان و امکان شرکت

در کلاس های ما را ندارند

فهرست

آموزش Spring ـ معرفی spring framework

چهارچوب کاری Spring چیست و چه کاربردی دارد

Springمحبوب ترین چهارچوب کاری جهت توسعه نرم افزارهای سطح بالا به زبان جاوا (Java) می باشد .میلیون ها برنامه نویس در سراسر جهان ، از چهارچوب کاری Spring برای تولید نرم افزارهایی با کارایی بالا ، ساده جهت تست و قابل استفاده مجدد از کدهای آن ، استفاده می کنند.

چهارچوب کاری Spring یک پلتفرم این سورس جاوا است که در ابتدا توسط Rod Johnson و تحت لیسانس سرور آپاچی 2.0 در سال 2003 نوشته شده است.

Springبسیار ساده و کم حجم جهت نصب و انتقال به روی سیستم ها می باشد . نسخه پایه Spring حجمی در حدود 2 مگا بایت دارد.

از قابلیت های اصلی چهارچوب کاری Spring می توان جهت توسعه نرم افزارهای مختلف برروی پلتفرم جاوا استفاده نمود ، ولی افزونه هایی هم وجود دارند که بوسیله آنها می توانید برنامه هایی را برپایه پلتفرم جاوا پیشرفته Enterprice Edition)) یا Java EE تولید کنید.

چهارچوب کاری Spring قصد دارد توسعه نرم افزارها برپایه پلتفرم ((Java 2 Enterprise Edition را ساده تر کرده و روش های کد نویسی را بهبود بخشد . برای این منظور هم از یک مدل قدیمی جاوا به نام Pojo استفاده می کند.

مزایای استفاده از چهارچوب کاریSpring

در لیست زیر به برخی از فواید مهم استفاده از چهارچوب کاری Spring اشاره شده است:

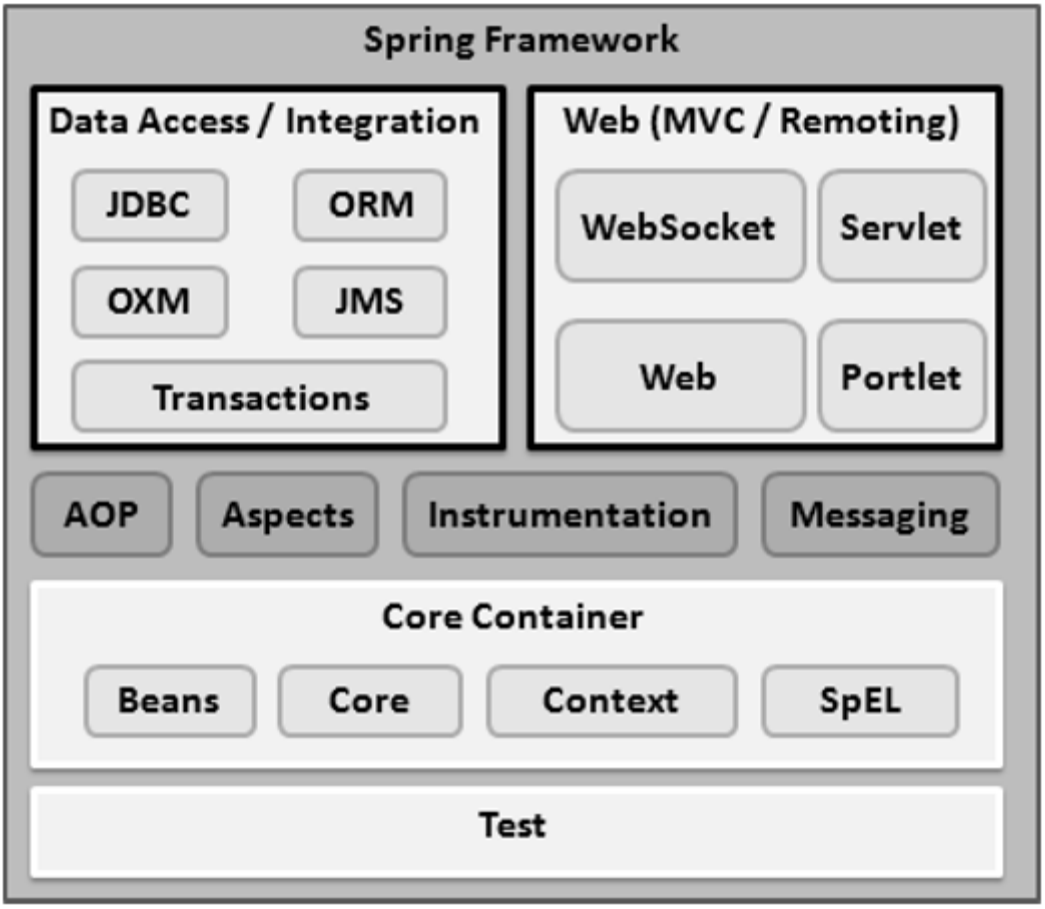
چهارچوب کاری Spring ، برنامه نویسان را قادر می سازد تابا استفاده از تکنیک Pojo ، نرم افزارهایی با کلاس های پیشرفته ایجاد نمایند . فایده استفاده تنها از Pojo این است که شما نیازی ندارید تا از یک سرور واقعی نظیر EJB برای اجرای برنامه ها استفاده کرده و درعوض می توانید از نرم افزارهای ساده تر مثل Tomcat برای این منظور بهره ببرید.

چهارچوب کاری Spring به صورت ماژول گونه (دارای بخش های مجزا) سازماندهی شده است . علی رغم اینکه تعداد کلاس ها و پکیج های موجود در آن بسیار زیاد هستند ، شما فقط بایستی تمرکز خود را برروی کلاس هایی که در آن برنامه می خواهید استفاده کنید ، جمع کرده و به بقیه کاری ندارید.

بررسی معماری spring framework

چهارچوب کاری Spring میتواند یک ابزار همه منظوره برای نرم افزارهای سطح بالا شما باشد . اما به هرحال ، Spring یک چهارچوب ماژول گونه (دارای بخش های مختلف) است و می توانید در هر برنامه فقط ماژول هایی که نیاز دارید را به پروژه اضافه کرده و به بقیه کاری نداشته باشید . در این درس ، به بررسی ماژول های مختلف موجود در چهارچوب Spring خواهیم پرداخت.

چهارچوب کاری Spring مشتمل بر حدود 20 ماژول مختلف بوده که در دیاگرام زیر به بررسی کلی آن پرداخته شده است :



بخش اول :Core Container

بخش Core Container شامل ماژورهای Core ، Context، Beansو Expression Language می باشد که در لیست زیر به تشریح هر کدام پرداخته ایم :

ماژول Core شامل بخش های اصلی چهارچوب Spring از جمله قابلیت های IoC و Dependency Injection می باشد.

ماژول Bean قابلیت Bean Factory را شامل می شود که یک اجرای از طرح اولیه برنامه است.

ماژول Context ، برروی یک پایه مستحکم که توسط Coreو Bean فراهم شده است ، ساخته می شود و یک واسطه برای دسترسی به کلیه اشیای تعریف شده و ساخته شده در برنامه است . رابطه کاربری Application Contextیک هسته مرکزی برای دسترسی به ماژول Context می باشد.

ماژول SPEL ، یک زبان و ابزار قوی را جهت جستجو و دستکاری اشیای برنامه در هنگام اجرا فراهم میکند.

بخش دوم : Data Access / Integration

لایه Data Access / Integration شامل ماژول های JDBC ، ORM ، OXM، JDBCو Transaction می باشد که در لیست زیر به تشریح هرکدام پرداخته شده است :

ماژول JDBC یک لایه مجزای JDBC را فراهم می کند تا نیازی به کدنویسی مرتبط در JDBC وجود نداشته باشد.

ماژول ORM ، لایه های یکپارچه ای را برای انجام عملیات ادغام و تبدیل داده ها (ORM) در API های محبوبی مثل JPA ، JDO، Hibrnateو iBatis فراهم میکند.

ماژول OXM ، یک لایه مجزا را جهت انجام عملیات تبدیل فایل های XML به اشیا و برعکس (object / XML Mapping) توسط JAXB ، XML Beansو یا Xstream فراهم میکند.

ماژول سیستم Transaction ، سیستم مدیریت تراکنش های برنامه نویسی و اطلاع رسانی را در کلاس هایی که رابطه های کاربری خاصی را فراهم میکند.

بخش سوم : Web

بخش لایه Web ، شامل ماژول های Web ، Web-MVC، Web-Socketو Web-Portlet میباشد که در لیست زیر به تشریح آنها می پردازیم :

ماژول Web ، قابلیت های پایه سیستم های وبی مثل آپلود فایل ها را انجام می دهد.

ماژول Web-MVC ، شامل دستورالعمل ها و کدهای لازم جهت معماری MVC در چهارچوب کاری Spring برای وب است.

ماژول Web-Socket ، امکانات لازم برای اتصال های دوطرفه سوکتی مانند بین کلاینت و سرور را فراهم می کند.

ماژول Web-Portlet شرایط اجرای معماری MVC را در محیط های پرتال مانند را فراهم کرده و عملکرد-Web Servletها را منعکس میکند.

آموزش تنظیم و نصب spring

در این درس ، به آموزش تنظیم و نصب محیط های کاری لازم جهت شروع برنامه نویسی با چهارچوب کاری springخواهیم پرداخت . همچنین در این آموزش ، نحوه تنظیم JDK ، نصب ابزار Tomcat ونصب نرم افزار Eclipseرا قبل از تنظیم محیط کاری spring خواهید آموخت .

مرحله اول : تنظیم Java Development Kit یا JDK

شما می توانید آخرین نسخه JDK را از وب سایت جاوای اوراکل به آدرس Java SE Downloads دانلود کنید . دستورالعمل های لازم جهت نصب JDK درون فایل های دانلود شده نرم افزار قرار دارند ، مراحل آن به ترتیب انجام داده تا عمل نصب صورت گیرد. در آخر هم متغیرهای محیطی PATH و JAVA HOMEرا به ترتیب برروی گوشه های حاوی java و javac ست کنید.

این پوشه ها معمولا Java_install_dir/bin و java_install_dir هستند.

اگر سیستم عامل ویندوز برروی کامپیوتر شما نصب شده و JDKدر پوشه C:\jdk\.6.0_15 قرار دارد ، بایستی کدهای زیر را به فایل C:\antoxec.batاضافه کنید :

```
set PATH=C:\jdk1.6.0_15\bin;%PATH%
set JAVA_HOME=C:\jdk1.6.0_15
```

همچنین در ویندوز های NT\2000\XP ، شما میتوانید برروی My computerکلیک راست کرده و ابتدا Propertiesو سپس Advanced و در نهایت Enviroment Variables را انتخاب کنید . سرانجام ، مقدار PATHرا به روز کرده و دکمه OK را بزنید .

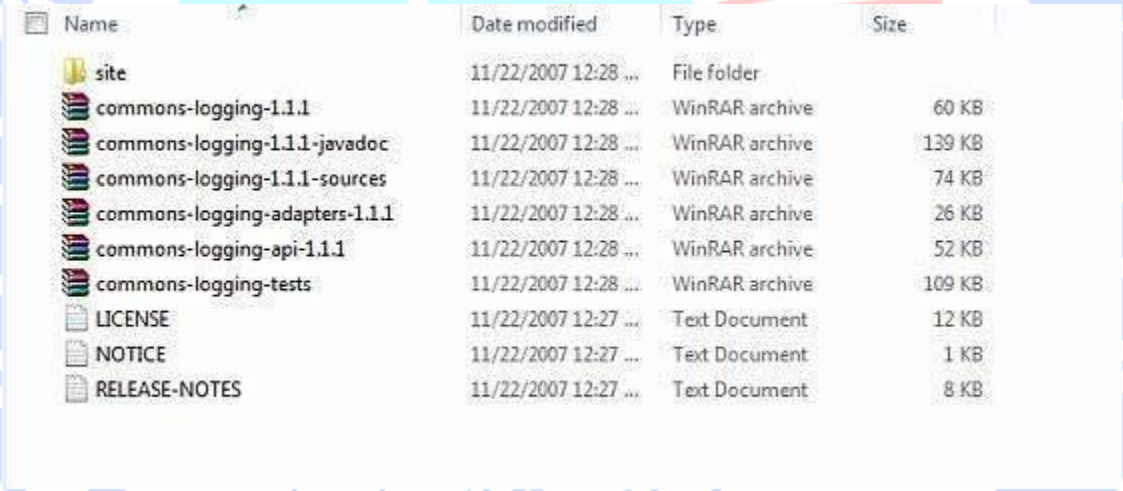
در سیستم عامل های) Unix همانند Linux, slaris و (... ، چنانچه SDK برنامه در پوشه /usr/local/jdk1.6.0_15قرار داشته و میتوانید هسته درایو C را مشاهده کنید ، کدهای زیر را به فایل cshrc.اضافه کنید:

```
setenv PATH /usr/local/jdk1.6.0 15/bin:$PATH
setenv JAVA_HOME /usr/local/jdk1.6.0_15
```

از طرف دیگر ، اگر یک نرم افزار کامل (IDE) مثل Borland ، JBو یا Eclipse برروی کامپیوتر شما نصب است ، با اجرای یک برنامه ساده ، امتحان کنید آیا نرم افزار (IDE) می داند Java در کجای کامپیوتر شما قرار داشته یا خیر ، در غیر اینصورت برحسب دستور العمل های خاصی هربرنامه ، به تنظیم Java بپردازید .

مرحله دوم : نصب API Apache Comman Logging

شما می توانید آخرین نسخه نرم افزار API Apache Comman Logging را از آدرس http://commans.apache.org/logging/دانلود نمایید . پس از اتمام دانلود ، فایل های نصب را در یک مکان مناسب از حالت فشرده خارج کنید . برای مثال در ویندوز پوشه C:/comman-logging-1.1.1 و در لینوکس پوشه /usr/local/commans-logging-1.1.1 میتوانند مناسب باشند . پوشه نصب حاوی یکسری فایل های فشرده و فایل های پشتیبانی و راهنما ، همانند تصویر زیر خواهند بود :



نکته :

توجه داشته باشید که حتما مقدار متغیر CLASSPATH را به صورت درست برروی این دایرکتوری تنظیم کنید ، وگرنه در هنگام اجرا دچار مشکل خواهید شد .

مرحله سوم : نصب نرم افزار IDE Eclipse

تمامی مثال های این بخش آموزش توسط نرم افزار Eclipse IDE نوشته شده اند ، بنابراین لازم است آخرین نسخه این برنامه را برروی کامپیوتر خود داشته باشید.

برای نصب Eclipse IDE ، آخرین نسخه آن را از آدرس http.//www.eclipse.org/downloads/ دانلود نمایید . پس از اتمام دانلود ، فایل های نصب را در یک پوشه مناسب از حالت فشرده خارج سازید . برای مثال در ویندوز پوشه C:\eclipse و در لینوکس پوشه /usr/bcal/eclipse می توانند مناسب باشند . در نهایت هم مقدار متغیر PATH را در برنامه برروی آدرس درست تنظیم کنید.

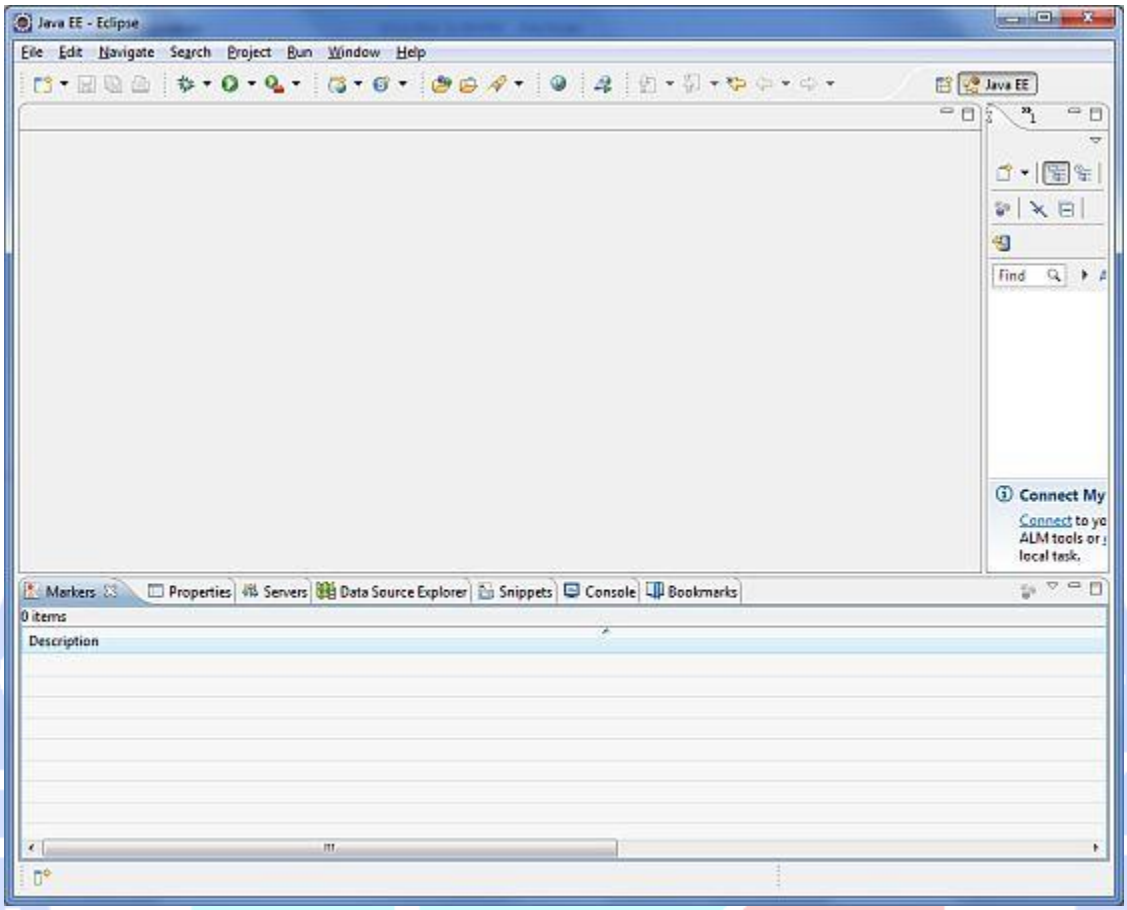
برنامه Eclipse را با اجرای دستور زیر در محیط ویندوز و یا دابل کلیک ساده برروی فایل eclipse.exe اجرا کنید :

```
%C:\eclipse\eclipse.exe
```

در سیستم عامل لینوکس هم میتوانید از خط دستور زیر استفاده کرده و یا برروی فایل eclipse.exe دابل کلیک نمایید :

```
$/usr/local/eclipse/eclipse
```

پس از نصب کامل Eclipse ، صفحه آغازین برنامه بایستی به صورت زیر باشد :



مرحله چهارم :تنظیم کتابخانه های چهارچوب کاری Spring

در مرحله چهارم ، اگر مراحل قبلی به درستی انجام شده بودند ، می توانید چهارچوب کاری Spring را برروی سیستم خود نصب کنید.

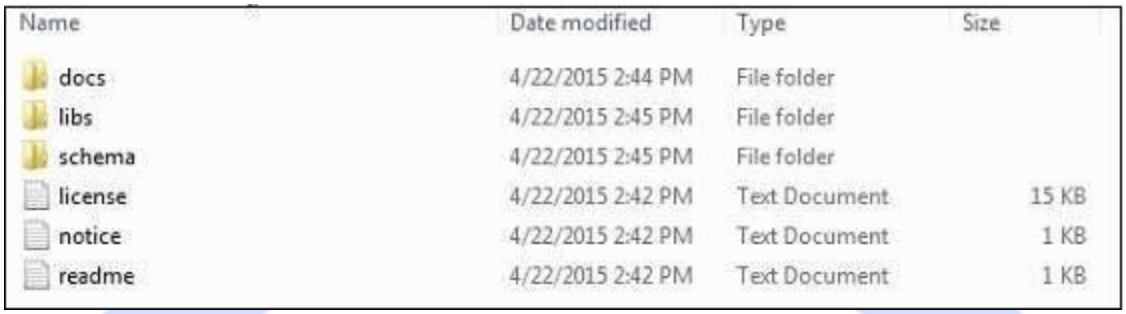
در لیست زیر مراحل نصب و راه اندازی چهارچوب کاری Spring را در سیستم های مختلف آموزش داده ایم :

ابتدا مشخص نمایید که چهارچوب کاری Spring را می خواهید برروی ویندوز و یا لینوکس نصب کنید . زیرا برای ویندوز بایستی فایل download.zip و برای لینوکس فایل download.tz را دانلود نمایید.

میتوانید آخرین نسخه کاری چهارچوب Spring را از

http://repo.spring.io/release/org/springframework/springدانلود کنید.

در زمان نوشتن این آموزش ، ما فایل spring-framework-4.1.6.RELEASE-dist.zipرا دانلود کردیم که در هنگام از فشرده خارج شدن ، پوشه های زیر در محل نصب) برای مثال(E:\spring ایجاد می شوند :



Name	Date modified	Type	Size
docs	4/22/2015 2:44 PM	File folder	
libs	4/22/2015 2:45 PM	File folder	
schema	4/22/2015 2:45 PM	File folder	
license	4/22/2015 2:42 PM	Text Document	15 KB
notice	4/22/2015 2:42 PM	Text Document	1 KB
readme	4/22/2015 2:42 PM	Text Document	1 KB

تمامی کتابخانه های آماده Spring را در پوشه E:\Spring\libsخواهید یافت . همچنین مطمئن شوید مقدار متغیر CLASSPATH را در برنامه برروی آدرس درست خود تنظیم کرده اید تا در هنگام اجرا دچار مشکل نشوید.

از طرف دیگر اگر از نرم افزار Eclipse استفاده می کنید ، نیازی به انجام تنظیم فوق نیست ، زیرا خود برنامه همه تنظیم را انجام می دهد.

در پایان این مراحل ، شما آماده هستید تا نخستین کدهای خود را بر پایه Spring بنویسید . در بخش بعد به آموزش نوشتن کدهای Spring خواهیم پرداخت .

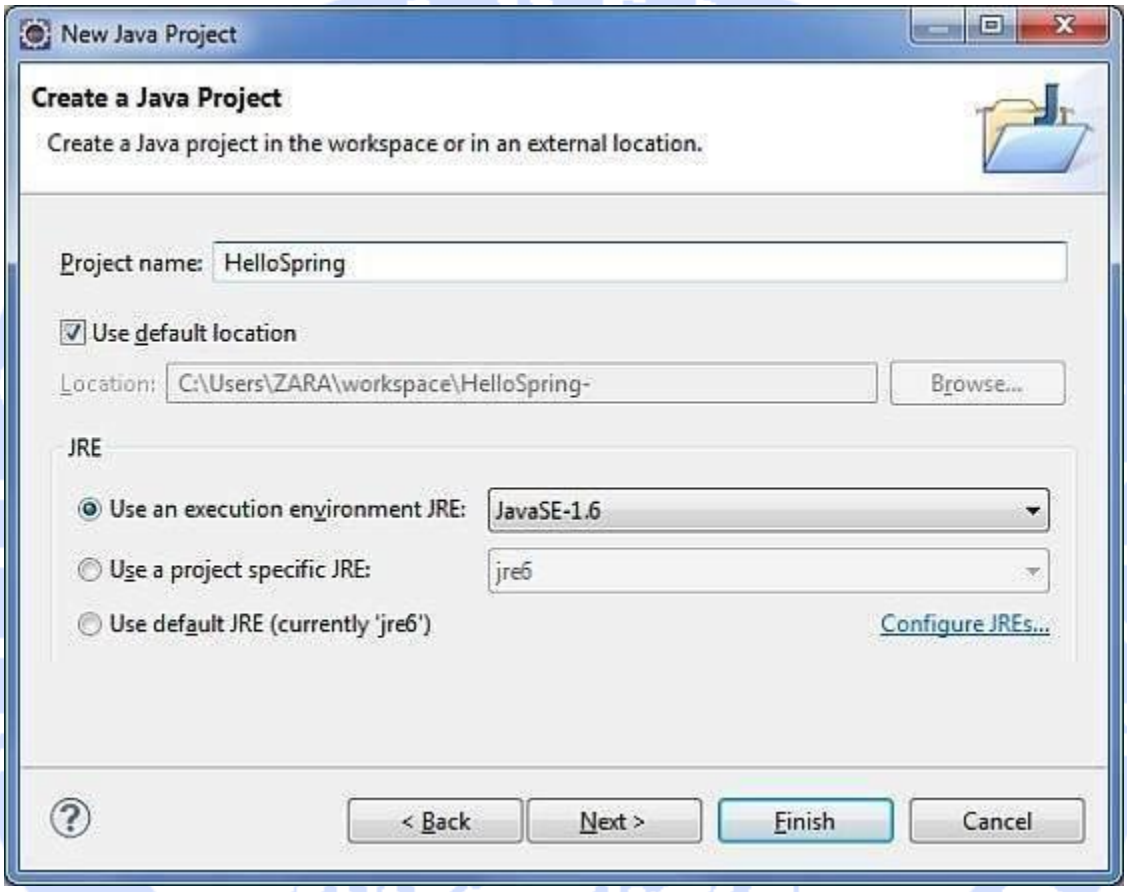
آموزش نوشتن برنامه توسط spring framework

در این درس ، قصد داریم تا کدنویسی با چهارچوب کاری Spring را شروع کنیم . قبل از آغاز کدنویسی ، حتما مطمئن شوید که محیط کاری Spring ، به درستی برروی سیستم شما نصب شده است و برای این منظور بایستی مراحل درس قبل ، آموزش نصب و راه اندازی چهارچوب کاری Spring را به دقت مطالعه کنید . ما همچنین درنظر میگیریم شما کمی با نحوه کار Eclipse آشنایی دارید .

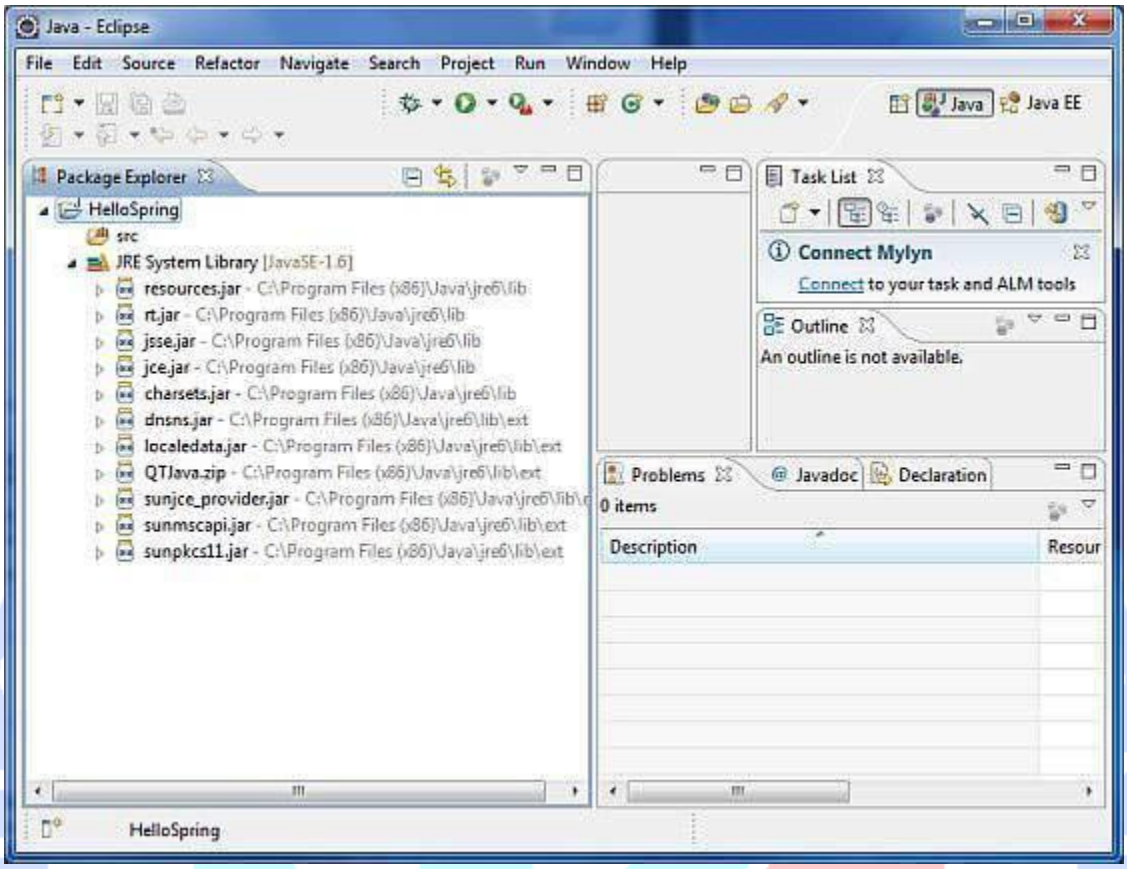
در اولین برنامه ای که با چهارچوب کاری Spring خواهیم نوشت ، کدهای ما مقدار "Hello World" را در خروجی نمایش خواهند داد .

مرحله اول : ایجاد پروژه جاوا

قدم اول ، ایجاد یک پروژه ساده جاوا به وسیله ابزار Eclipseاست . برای این منظور مراحل زیر را در منو طی کرده (File ->New ->Project) و در نهایت ویزارد Java Project را از لیست ویزاردها انتخاب کنید. سپس همانند تصویر زیر ، نام پروژه را Hello Spring تعیین کنید :



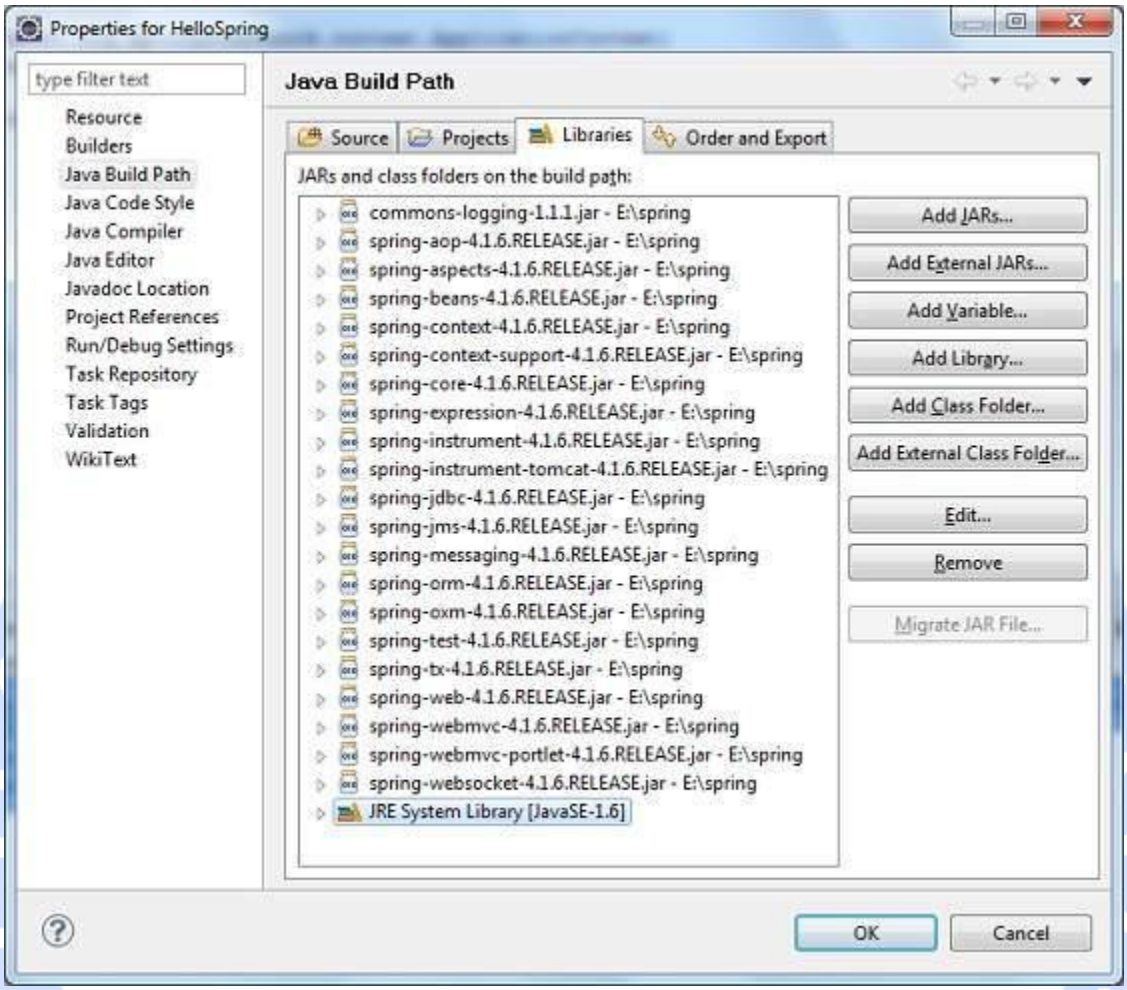
به محض اینکه پروژه جدید شما ایجاد شود ، محتویات تصویر زیر را در منوی Project Explorer خواهید دید :



مرحله دوم : اضافه نمودن کتابخانه های لازم

در مرحله دوم بایستی چهارچوب کاری Spring و کتابخانه های مورد نظر خود را از Common logging API به پروژه اضافه کنیم . برای این منظور برروی نام پروژه خود کلیک نموده و مسیر زیر را طی نمایید (Build Path) :

(Configure Build Path->، تا پنجره Java Build Path همانند عکس زیر باز شود :



سپس دکمه Add External JARS که در زیرمنوی Libraries قرار دارد را کلیک نموده تا چهارچوب کاری Spring و سایر کتابخانه های مورد نظر را از بخش Common Logging ، طبق فهرست زیر به پروژه اضافه کنیم :

commons-logging-1.1.1

spring-aop-4.1.6.RELEASE

spring-aspects-4.1.6.RELEASE

spring-beans-4.1.6.RELEASE

spring-context-4.1.6.RELEASE

spring-context-support-4.1.6.RELEASE

spring-core-4.1.6.RELEASE

spring-expression-4.1.6.RELEASE

spring-instrument-4.1.6.RELEASE

spring-instrument-tomcat-4.1.6.RELEASE

spring-jdbc-4.1.6.RELEASE

spring-jms-4.1.6.RELEASE

spring-messaging-4.1.6.RELEASE

spring-orm-4.1.6.RELEASE

spring-oxm-4.1.6.RELEASE

spring-test-4.1.6.RELEASE

spring-tx-4.1.6.RELEASE

spring-web-4.1.6.RELEASE

spring-webmvc-4.1.6.RELEASE

spring-webmvc-portlet-4.1.6.RELEASE

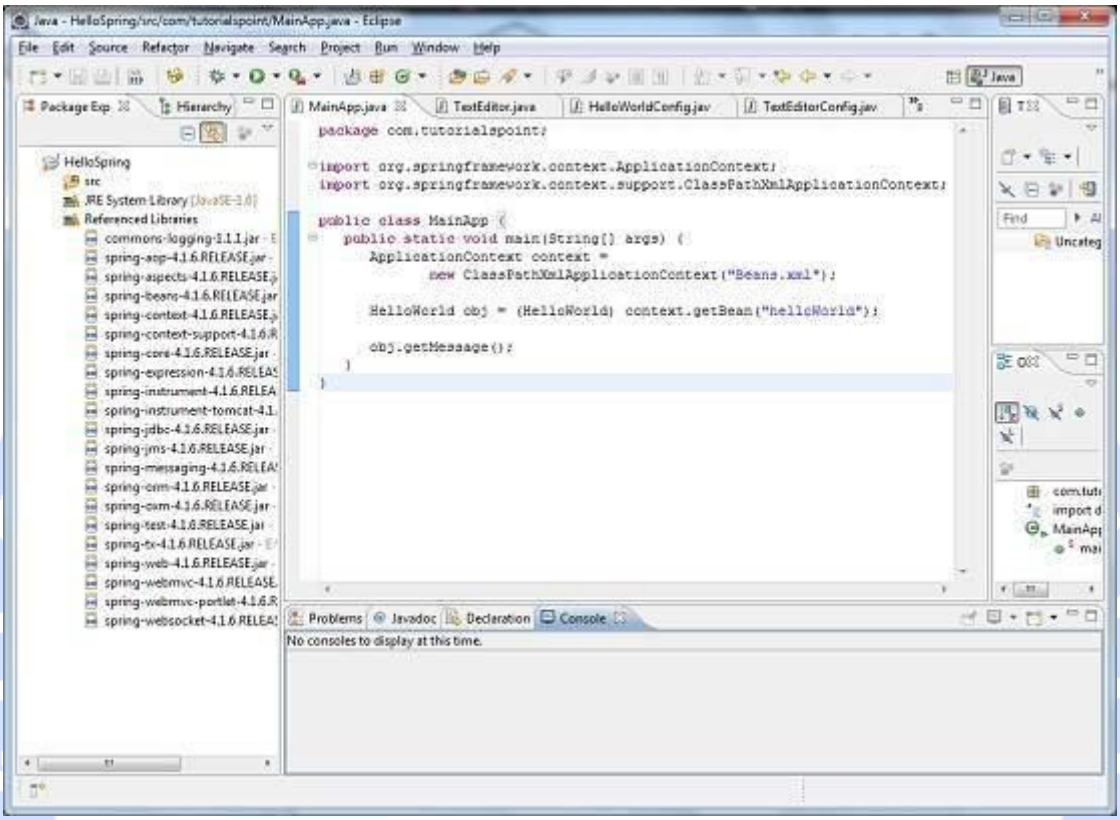
spring-websocket-4.1.6.RELEASE

مرحله سوم : ایجاد فایل های منبع (Source Files)

در مرحله بعد قصد داریم تا فایل های منبع (Source Files) را به پروژه خود اضافه کنیم . برای این منظور ابتدا بایستی یک پکیج با نام دلخواه مثل com.tahlildadeh را ایجاد کنیم . جهت این کار ، برروی گزینه srcاز بخش package explorer کلیک نموده و سپس مسیر را طی کنید :

سپس فایل های HelloWorld.java و MainAPP.java را در زیر پکیج (New ->package)

com.tahlildadehایجاد می کنیم .



کد فایل Hello World.java به صورت زیر می باشد :

```
package com.tutorialspoint;

public class HelloWorld {
   private String message;

   public void setMessage(String message){
      this.message  = message;
   }

   public void getMessage(){
      System.out.println("Your Message : " + message);
   }
}
```

کد زیر نیز ، مربوط به فایل MainAPP.java است :

```
package com.tutorialspoint;
```

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context =
             new ClassPathXmlApplicationContext("Beans.xml");

      HelloWorld obj = (HelloWorld) context.getBean("helloWorld");

      obj.getMessage();
   }
                                                                    }
```

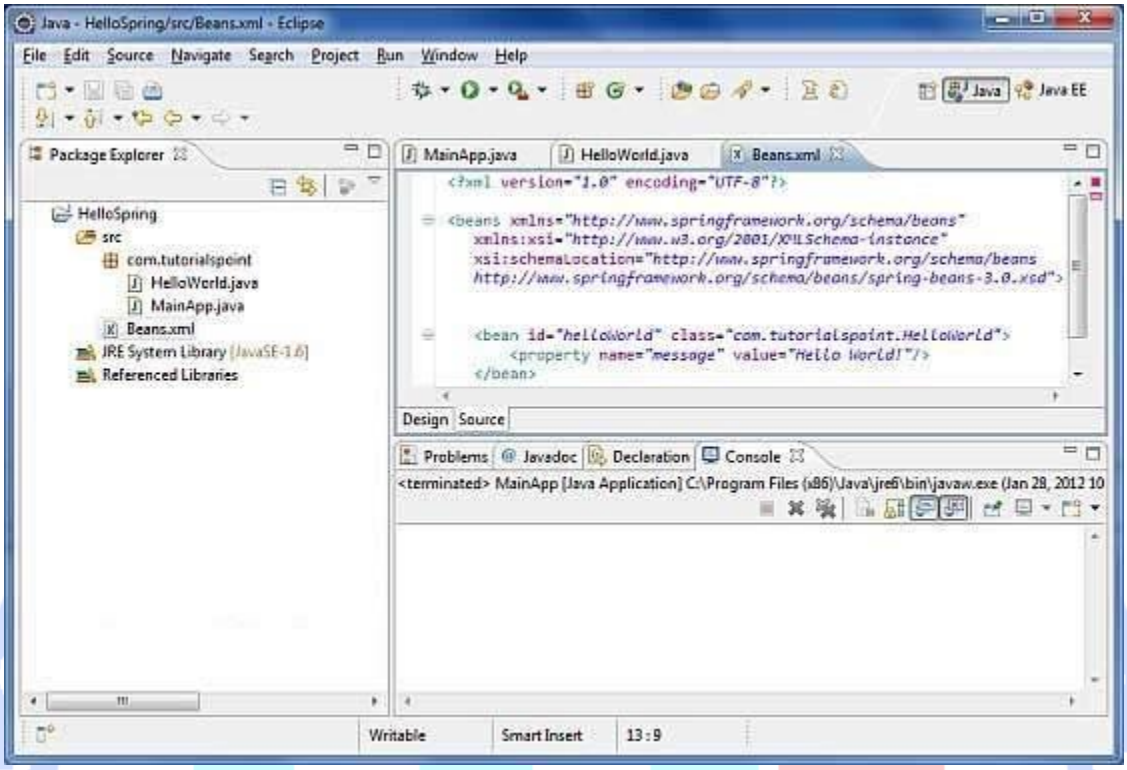
به دو نکته مهم راجع به برنامه main اشاره می کنیم :

اولین نکته مهم این است که بایستی محتوای برنامه را در محلی که کتابخانه API برنامه یعنی ClassPathXml ApplicationContenxtقرار دارد ، ایجاد کنیم API . اشاره شده فایل های bean configurationرا خوانده و بر حسب آن ، بر عملیات ایجاد و مقدار دهی کل اشیای برنامه از جمله اشیای فایل bean configurationنظارت و کنترل میکند.

نکته دوم در مورد کد این است که بایستی به وسیله متد getBean() در محتوای ایجاد شده ، beanلازم برنامه را دریافت کنیم . این متد از Bean ID استفاده کرده تا یک شی عمومی را برگرداند و این شی در نهایت می تواند قالب دهنده اشیای واقعی برنامه باشند .زمانی که شما یک شی را در اختیار داشته باشید ، از آن می توانید برای فراخوانی هر متدی در یک کلاس استفاده کنید.

مرحله چهارم : ایجاد فایل پیکر بندی اطلاعات Bean Configuration

در مرحله چهارم ، شما می بایست یک فایل bean configuration ایجاد کرده که یک فایل XML بوده و همانند یک سیمان عمل کرده و کلید کلاس های bean را به هم می چسباند . این فایل بایستی در زیر پوشه srcهمانطور که در تصویر زیر نشان داده شده است ، قرار گیرد :



معمولا برنامه نویسان جاوا این فایل را با نام Bean.xml ذخیره می کنند ، ولی شما مجاز هستید هرنام دلخواه دیگری را نیز به کار ببرید . فقط توجه داشته باشید که نام این فایل در متغیر CLASSPATH وجود داشته و از طرف دیگر از همین نام در هنگام تولید محتوای فایل MainAPP.java آن طور که در قسمت قبل نشان دادیم ، استفاده کنید .

فایل Bean.xml جهت اختصاص دادن ID های منحصر به فرد برای beanهای مختلف استفاده شده و همچنین ایجاد اشیای مختلف با مقدارهای متفاوت را کنترل می کند ، بدون اینکه تاثیری برروی فایل های اصلی Spring داشته باشد .

برای مثال ، با استفاده از فایل زیر می توانید هرمقدار دلخواهی را به متغیر mwssage پاس داده و بدون تاثیر گذاشتن برروی فایل های MainAPP.javaو HelloWorld.java ، مقادیر مختلف آن را در خروجی چاپ کنید .بیایید به نحوه کار آن نگاهی بیاندازیم :

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

<beans xmlns="www.springframework.org/schema/beans"
xmlns:xsi="www.w3.org/2001/XMLSchema-instance"
xsi:schemalocation="www.springframework.org/schema/beans
    www.springframework.org/schema/beans/spring-beans-3.0.xsd">
```

```
   <bean id="helloWorld" class="com.tutorialspoint.HelloWorld">
       <property name="message" value="Hello World!">
   </property></bean>

</beans>
```

هنگامی که برنامه Spring در حافظه لود می شود ، چهارچوب از فایل فوق جهت ایجاد کلیه bean های تعیین شده در آن استفاده کنید . سپس IDهای منحصر به فرد به هرکدام از اشیایی که در تگ برای پاس دادن مقادیر مختلف متغیرها در هنگام ایجاد هرشی ، استفاده کنید .

مرحله پنجم –اجرای برنامه :

پس از اینکه فایل های اصلی برنامه (Source files) و فایل پیکربندی bean canfiguration را ایجاد نمودید ، می توانید برنامه را کامپایل و اجرا کنید . برای این منظور ، فایل MainAPP.java را انتخاب کرده و سپس از گزینه Run در برنامه Eclipse استفاده نموده و یا دکمه های Ctrl+F11 را باهم فشار دهید . اگر همه چیزدر برنامه شما درست باشد ، زیر در خروجی برنامه Eclipse نمایش داده می شود :

```
Your Message : Hello World!
```

تبریک میگوییم ، شما اولین برنامه Spring خود را با موفقیت ایجاد کردید . شما میتوانید انعطاف پذیری برنامه ذا با تغییر مقدار متغیر message و عدم تغییر در فایل های اصلی برنامه مشاهده کنید . در بخش های بعدی ، کارهای جالبتری را با Spring انجام خواهیم داد.

آموزش Spring Containers

Containersچیست و چه کاربردی دارد ؟

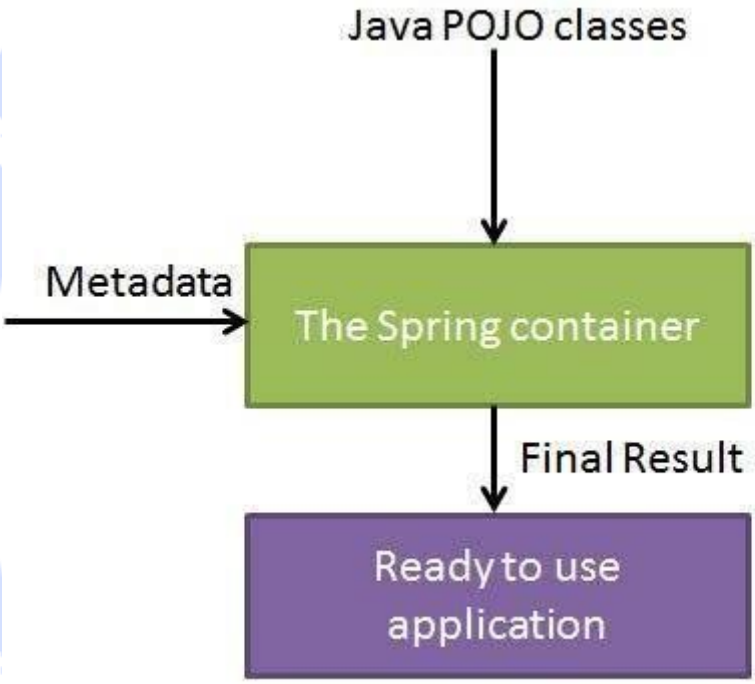
Spring Containersهسته اصلی چهارچوب کاری Spring است Spring Containers . اشیا یا object های برنامه را ایجاد کرده ، آنها را به هم مرتبط می سازد (Wiring) ، تنظیم آنها را انجام داده و چرخه حیات (Life cycle)آنها را از زمان ایجاد تا ازبین رفتن مدیریت می کند.

Spring Containerاز قابلیت injection dependency (DI) برای مدیریت اجزایی که یک برنامه را می سازند ، استفاده می کند . به این اشیا در اصلاح Spring Beansمی گویند که در درس بعدی به بررسی آنها خواهیم پرداخت.

Spring Containerدستورالعمل های لازم برای اینکه چه اشیایی را ایجاد کند ، آنها را چطور تنظیم نموده و

اجرا نماید ، از طریق داده هایی که توسط بخش Configration به آن داده می شود ، دریافت می کند . این اطلاعات پیکر بندی (Configration metadata) را میتوان از طریق یک فایل XML ، یا Java Axnotayion و یا کدهای جاوا ارسال نمود.

دیافراگم زیر یک نمای کلی از اینکه Spring Container چطور کار میکند را نشان می دهد Spring IOC . Containerاز کلاس های POJO و اطلاعات پیکربندی برای ایجاد یک نرم افزار کامل و قابل اجرا استفاده میکند.



چهارچوب کاری Spring دو نوع Contain مجزای زیر را داراست:

Spring BeanFactory Container :

ساده ترین نوع Container چهارچوب کاری Spring بوده و پشتیبانی اولیه لازم جهت DI را فراهم می کند . این Container از طریق آدرس +آدرس قابل دسترسی است.

Bean Factoryو رابط های کاربری وابسته آن همانند Bean Factory Awave ، Initializing Beanو Disposable Beanبرای سازگاری با نسخه های قبل ، همچنان در چهارچوب کاری Spring وجود دارند.

Spring Application Container :

این Container ، کاربردهای سطح بالاتر و ویژه ای را نسبت به Bean Factory در اختیار ما قرار می دهد . از آن جمله می توان به قابلیت دریافت و پردازش پیام های متنی صادره از فایل های properties و قابلیت رساندن رویدادهای برنامه به توابع و اهداف مورد نظر اشاره کرد . این Container از طریق +طریق قابل دسترس است Application Contex . تمامی قابلیت های Bean Factory را شامل می شود . بنابراین توصیه ما بر این است از این Container به جای Bean Factory استفاده کنید . اما Bean Factory را می توانید همچنان برای تولید نرم افزارهای سبک حجم مثل برنامه های موبایل یا Applet که حجم و سرعت اجرای برنامه بسیار مهم است ، استفاده کنید.

آموزش تعریف Beans در Spring

Beanچیست و چه کاربردی دارد ؟

اشیایی که ستون فقرات برنامه شما را تشکیل داده و توسط Spring IOC Container مدیریت می شوند ، Beansمی نامیم . یک Bean شی ای است که توسط یک Spring IOC Containerمعرفی ، ایجاد و اجرا شده است . این Bean ها ، توسط اطلاعات پیکربندی (configuration metadata) که برای یک Container فراهم نموده اید (برای مثال توسط یک فایل XML با تگ ( ) ایجاد می شوند . به آموزش این روش در درس قبلی پرداختیم.

تعاریف لازم جهت یک Bean یا Bean Definition ، اطلاعاتی هستند که به آنها اطلاعات پیکربندی ((configuration metadataگفته و به Container موارد زیر را توضیح می دهند:

چگونگی ایجاد یکBean

جزئیات مربوط به چرخه حیات یک Bean (از ایجاد تا انهدام)

جزئیات مرتبط با Bean یا. dependencies Bean`s

کلیه اطلاعات فوق در قالب مجموعه ای از خواص خلاصه شده که هر Bean را تعریف و ایجاد می کند . در جدول زیر به بررسی آنها پرداخته شده است:

Class : این خاصیت ضرروری بوده و کلاسی که بر مبنای آن Bean بایستی ساخته شود را مشخص می کند.

name : این خاصیت یک نام منحصر به فرد (شناسه) جهت شناسایی Beanدر سطح برنامه تعریف می کند . در اطلاعات پیکربندی مبتنی بر XML ، شما میتوانید از خاصیت name یا id جهت تعیین شناسه منحصر به فرد برای Bean استفاده کنید.

Scape : این خاصیت محدوده عملیات و سطح دسترسی را جهت یک Bean تعیین میکند و در درس های بعدی به صورت ویژه بررسی خواهد شد.

Constructor_arg :از این خاصیت برای تعیین وابستگی های یک Bean استفاده شده و در درس بعدی به آن خواهیم پرداخت.

Properties : این خاصیت نیز برای تعیین وابستگی های یک Bean به کار رفته و در درس های بعد به آن خواهیم پرداخت.

antowriting mode : از این خاصیت نیز برای تزریق وابستگی ها به یک Bean استفاده شده و در درس بعدی به آن خواهیم پرداخت.

Lazy_initialization mode : روش Lazy_initialized به IOC Container میگوید تا نسخه ای از هر beanرا در هنگامی که برای اولین بار درخواست می شود بسازد و نه درهنگام شروع و اجرای برنامه.

Initialization method : این خاصیت یک متد Call back است که بلافاصله پس از اینکه Container کلیه خواص مربوط به یک Bean را تنظیم کرد ، فراخوانی و اجرا می شود . به عبارت دیگر رویدادی است که پس از ساخته شدن یک Bean رخ می دهد . در درس بررسی سیکل زمانی یک Bean به تشریح کامل آن خواهیم پرداخت.

Destruction method : این خاصیت نیز یک متد Call back است که پس از نابود شدن Container ی که Beanموردنظر را تولید کرده ، فراخوانی و اجرا می شود . به عبارت دیگر رویدادی است که پس از نابود شدن یک Bean رخ می دهد . در درس بررسی سیکل زمانی یک Bean به تشریح بیشتر آن خواهیم پرداخت.

بررسی فایل Metadata Spring Configuration

Spring IOC Containerبه طور کامل فایلی جدا از فایل پیکربندی اطلاعات مرتبط با آن یا Configuration Metadataاست . به طور کلی ، سه راه برای فراهم نمودن اطلاعات پیکربندی جهت یک Spring Container وجود دارد که عبارتند از:

- فایل های پیکربندی مبتنی بر. (XML based configuration) XML
- فایل های پیکربندی مبتنی بر. Annotation
- فایل های پیکربندی مبتنی بر. Java

در درس های قبلی باهم به بررسی نحوه ارسال اطلاعات پیکربندی ازطریق یک فایل XML به Container پرداختیم . اکنون می خواهیم مثال دیگری را درمورد فایل های XML بررسی کنیم که در آن از متدهای Lazy initialization، متد initialization method و متد destruction methodاستفاده شده است . کد XML فایل مورد نظر به صورت زیر است:

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

   id="..." class="...">
   id="..." class="..." lazy-init="true">

   id="..." class="..." init-method="...">
```

```
id="..." class="..." destroy-method="...">
```

برای آموزش کامل چگونگی ایجاد ، مقدار دهی و تنظیم یک Bean به درس آموزش نوشتن اولین برنامه Springبروید.

درباره فایل های پیکربندی اطلاعات مبتنی بر Annotation در درس دیگری بحث خواهیم کرد . ماعمدا این درس را در یک مبحث جدا گذاشتیم ، تا قبل از شروع برنامه نویسی با Dependecy Injection و Annotationsبه تشریح چند مبحث مهم دیگر در Spring بپردازیم.

آموزش تعیین میدان Beans Scope در Spring

میدان (scope) یک Beans چیست و چه کاربردی دارد ؟

هنگام تعریف یک شی <bean> در چهارچوب کاری Spring ، شما می توانید یک میدان (Scope) برای آن تعیین کنید . اما میدان (Scape) چیست و چه کارایی دارد . برای مثال فرض کنید شما میخواهید تا برنامه Spring را مجبور کنید ، هر زمان که به یک شی Bean نیاز دارید ، یک نسخه جدید از آن را بسازید . برای این منظور بایستی مقدار خاصیت Scope آن را برروی prototype قرار دهید . از طرف دیگر ، برخی مواقع می خواهید برنامه Spring هرزمان که یک شی Bean را لازم دارد ، همان نسخه قبلی ساخته شده را برگرداند و یک نسخه جدید تولید نکند . برای این منظور نیز بایستی مقدار خاصیت Scope را برروی singleton قرار دهید .

به طور کلی ، پنج حالت میدان (Scope) برای یک Bean در چهارچوب کاری Spring وجود داشته که در جدول زیر به معرفی آنها پرداخته ایم .

3 مدل از این Scope ها مخصوص برنامه های تحت وب Spring هستند :

- Singleton : این حالت که مقدار پیش فرض (default) برنامه هم هست ، هر زمان که به یک شی نیاز دارید ، نسخه تولید شده قبلی آن را به IOC Container می دهد .
- Prototype : در این حالت ، در زمان درخواست هرشی Bean ، برنامه نسخه جدیدی از آن را ساخته و بنابراین هرشی میتواند در آن واحد چندین نسخه از خود داشته باشد .

- Request : این خاصیت ایجاد یک نسخه جدید از شی Bean را به درخواست HTTP سرور مرتبط میکند و مختص برنامه های تحت وب Spring است .
- Session : این خاصیت نیز ایجاد یک نسخه جدید از شی Bean را به وضعیت Session وب مرتبط کرده و فقط مختص برنامه های تحت وب Spring است .
- Global_session : این خاصیت نیز ایجاد یک نسخه از شی Bean را به درخواست سراسری Http سرور مرتبط کرده و مختص برنامه های تحت وب Spring است .

در این درس ، به بررسی دو حالت اول پرداخته و 3 مورد بعدی را پس از اینکه راجع به برنامه های تحت وب Spring صحبت کردیم ، بررسی می کنیم .

میدان Singleton Scope

اگر مقدار خاصیت Scope برروی Singleton تنظیم شده باشد ، Spring IOC Container فقط یک نسخه از شی Bean تعیین شده ایجاد خواهد کرد . این نسخه واحد ایجاد شده ، در یک حافظه کش (Coche) مرتبط با شی Bean ذخیره شده و هر بار که درخواست یا برنامه ای ، آن شی را فراخوانی کند ، نسخه موجود در حافظه برای آن ارسال می شود .

مقدار پیش فرض خاصیت Scope همواره Singleton است ، ولی درصورت نیاز و برای اطمینان خاطر بیشتر ، هنگامی که میخواهید برنامه فقط و فقط یک نسخه از Bean تولید کند ، مطابق کد زیر ، مقدار آن را در فایل پیکربندی برنامه برروی Singleton تنظیم کنید :

```
id="..." class="..." scope="singleton">
```

مثال عملی

اجازه بدهید روند ایجاد یک برنامه Spring و تنظیم خاصیت Scope را در محیط Ecipse باهم به صورت عملی تمرین کنیم . برای این منظور مراحل زیر را به ترتیب انجام دهید :

1. یک پروژه جدید با نام Spring Example ایجاد کرده و یک پکیج با نام Com.tahlildadeh را به پوشه src پروژه اضافه کنید .

2. همانطور که در درس آموزش ایجاد اولین برنامه Spring نشان دادیم ، کتابخانه های لازم جهت برنامه خود را به وسیله دکمه Add External JARs به پروژه اضافه کنید .

3. در زیر مجموعه پکیج Com.tahlildadeh ، کلاس های جاوا Hello World و Main App را ایجاد نمایید .

4. در پوشه Src ، فایل پیکربندی اطلاعات Beans Configuration را با نام Beans.Xml ایجاد نمایید .

5. در مرحله آخر نیز ، محتویات مورد نظر جهت فایل های جاوا ، فایل پیکربندی اطلاعات و سایر فایل ها را برای اجرای برنامه ایجاد نمایید .

سورس کد فایل HelloWorld بایستی به صورت زیر باشد :

```
package com.tutorialspoint;


public class HelloWorld {
   private String message;


   public void setMessage(String message){
      this.message  = message;
   }


   public void getMessage(){
      System.out.println("Your Message : " + message);
   }
}
```

همچنین کد زیر ، مربوط به فایل MainApp.java است :

```
package com.tutorialspoint;
```

```
import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;


public class MainApp {

   public static void main(String[] args) {

      ApplicationContext context = new
ClassPathXmlApplicationContext("Beans.xml");


      HelloWorld objA = (HelloWorld) context.getBean("helloWorld");


      objA.setMessage("I'm object A");

      objA.getMessage();


      HelloWorld objB = (HelloWorld) context.getBean("helloWorld");

      objB.getMessage();

   }

}
```

در نهایت نیز کد زیر مربوط به فایل Beans.Xml است که در آن خاصیت Scope با مقدار Singleton تنظیم شده است :

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


 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">


    id="helloWorld" class="com.tutorialspoint.HelloWorld"
```

```
      scope="singleton">
```

پس از اینکه محتویات فایل های برنامه و فایل پیکربندی اطلاعات را طبق کدهای فوق تنظیم نمودید ، به اجرای برنامه می پردازیم . اگر همه چیز درست باشد ، بایستی خروجی زیر تولید گردد :

```
Your Message : I'm object A

Your Message : I'm object A
```

میدان Portotype Scope

اگر مقدار خاصیت Scope برروی prototype تنظیم شود ، Spring IOC Container ، هربار که درخواستی جهت یک شی Bean ارسال می شود ، یک نسخه جدید از آن شی را میسازد .

به عنوان یک قانون ، از prototype Scope برای تعریف اشیای با ثبات Beans و از Singleton scope برای تعریف اشیای بی ثبات که پایداری نامطمئنی دارند ، استفاده کنید .

برای تعیین یک شی با prototype ، بایستی همانند کد زیر در فایل پیکربندی اطلاعات برنامه مقدار خاصیت Scope را برروی prototype تنظیم نمایید :

```
id="..." class="..." scope="prototype">
```


مثال عملی :

اجازه بدهید روند ایجاد یک برنامه Spring و تنظیم خاصیت Scope را در محیط Ecipse باهم به صورت عملی تمرین کنیم . برای این منظور مراحل زیر را به ترتیب انجام دهید :

1. یک پروژه جدید با نام Spring Example ایجاد کرده و یک پکیج با نام com.tahlildadeh را به پوشه src پروژه اضافه کنید .
2. همانطور که در درس آموزش ایجاد اولین برنامه Spring نشان دادیم ، کتابخانه های لازم جهت برنامه خود را به وسیله دکمه Add External JARs به پروژه اضافه کنید .

3. در زیر مجموعه پکیج Com.tahlildadeh ، کلاس های جاوا HelloWorld و MainApp را ایجاد نمایید

4. در پوشه Src ، فایل پیکربندی اطلاعات Beans Configuration را با نام Beans.Xml ایجاد نمایید

5. در مرحله آخر نیز ، محتویات مورد نظر جهت فایل های جاوا ، فایل پیکربندی اطلاعات و سایر فایل ها را برای اجرای برنامه ایجاد نمایید .

سورس کد فایل HelloWorld بایستی به صورت زیر باشد :

```
package com.tutorialspoint;


public class HelloWorld {
   private String message;


   public void setMessage(String message){
      this.message  = message;
   }


   public void getMessage(){
      System.out.println("Your Message : " + message);
   }
}
```

همچنین کد زیر ، مربوط به فایل MainApp.java است :

```
package com.tutorialspoint;


import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;
```

```
public class MainApp {

   public static void main(String[] args) {

      ApplicationContext context = new
ClassPathXmlApplicationContext("Beans.xml");


      HelloWorld objA = (HelloWorld) context.getBean("helloWorld");


      objA.setMessage("I'm object A");

      objA.getMessage();


      HelloWorld objB = (HelloWorld) context.getBean("helloWorld");

      objB.getMessage();

   }

}
```

در نهایت نیز کد زیر مربوط به فایل Beans.Xml است که در آن خاصیت Scope با مقدار Singleton تنظیم شده است :

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


 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">


    id="helloWorld" class="com.tutorialspoint.HelloWorld"

      scope="prototype">
```

پس از اینکه محتویات فایل های برنامه و فایل پیکربندی اطلاعات را طبق کدهای فوق تنظیم نمودید ، به اجرای برنامه می پردازیم . اگر همه چیز درست باشد ، بایستی خروجی زیر تولید گردد:

```
Your Message : I'm object A

Your Message : null
```

بررسی چرخه حیات یک Bean در Spring

آشنایی با چرخه حیات یک Bean در Spring :

چرخه حیات یک Spring Bean بسیار ساده و قابل فهم است . هنگامی که یک Bean ، ایجاد می شود چند مرحله مقدار دهی اولیه برای آن لازم است تا به یک وضعیت با ثبات برسد . از طرف دیگر ، هنگامی که به یک Bean دیگر نیازی نداریم و بایستی از Container حذف شود ،چند مرحله عملیات پاکسازی بایستی انجام شود .

بنابراین ، از زمان ایجاد تا نابود شدن یک Bean ، مجموعه ای از فعالیت ها در پشت صفحه برنامه رخ میدهد . اما در این درس ، ما فقط دو رویداد (Callback method) مهمی که در هنگام ایجاد و ازبین رفتن یک Bean لازم و ضروری هستند را بررسی و تشریح میکنیم .

برای تعیین یک اجرا و پایان جهت هر Bean ، درهنگام تعریف <bean> ، آن را با پارامتری مقدار دهی شده destroy-method و init-method در کد خود ، مینویسیم . خاصیت init-method تعیین کننده رویدادی است که به محض ایجاد یک Bean ، فراخوانی و اجرا میشود . همچنین خاصیت destroy-method نیز شامل رویدادی است که به محض حذف یک Bean از Container ، فراخوانی و اجرا میشود .

رویدادهای مقداردهی اولیه (Initialization Callback) :

رابط کاربری org.springframework.beans.factory.InitializingBean یک متد تنها به صورت زیر را تعیین کرده است :

```
void afterPropertiesSet() throws Exception;
```

بنابراین شما می توانید رابط کاربری فوق را به برنامه خود اضافه کرده و درون آن ، تمامی کدهایی که میخواهید در مرحله مقدار دهی اولیه و ایجاد یک Bean رخ دهد را در تابع afterpropertiesSet() همانند مثال زیر بنویسید :

```
public class ExampleBean implements InitializingBean {

   public void afterPropertiesSet() {

      // do some initialization work

   }

}
```

در فایل های پیکربندی اطلاعات مبتنی بر XML ، شما میتوانید از خاصیت init-method برای تعیین نام یک تابع برای اجرا در مقداردهی اولیه استفاده کنید . این تابع نه خروجی داشته (Void) و نه آرگومان دریافت می کند . مثل کد زیر :

```
id="exampleBean"

   class="examples.ExampleBean" init-method="init"/>
```

همچنین نحوه تعریف کلاس آن نیز به صورت زیر است :

```
public class ExampleBean {

   public void init() {

      // do some initialization work

   }

}
```

رویدادهای تخریب شی (Pestruction Callback)

رابطه کاربری +آدرس یک متد تنها را به صورت زیر تعیین کرده است :

```
void destroy() throws Exception;
```

بنابراین شما می توانید رابطه کاربری فوق را به برنامه خود اضافه کرده و درون آن ، کارهایی که میخواهید در زمان ازبین رفتن یک Bean رخ دهد را در تابع ()destroy به صورت زیر قراردهید :

```
public class ExampleBean implements DisposableBean {

   public void destroy() {

      // do some destruction work

   }

}
```

در فایل های پیکربندی اطلاعات مبتنی بر XML ، شما میتوانید از خاصیت destroy-method برای اجرا در هنگام تخریب کد شی Bean استفاده کنید . این تابع نه خروجی داشته (Void است) و نه آرگومانی دریافت می کند . همانند کد زیر :

```
id="exampleBean"

        class="examples.ExampleBean" destroy-method="destroy"/>
```

همچنین نحوه تعریف کلاس آن نیز به صورت زیر است :

```
public class ExampleBean {

   public void destroy() {

      // do some destruction work

   }

}
```

اگر شما درحال استفاده از Spring IOC Container در یک محیط غیر وب ، مثل یک نرم افزار سطح بالای تحت کلاینت هستید ، میتوانید به وسیله JVM یک تله برای خاموش شدن برنامه خود تعیین کنید .

انجام کار فوق باعث اجرای یک Shutdown بسیار خوب در سیستم شده و تمامی رویدادهای مرتبط با عملیات تخریب اشیای Bean یا (destroy) را به درستی اجرا می کند . بنابراین کلید منابع سیستم به طور صحیح آزاد می شوند .

توصیه ما به شما این است که حدامکان از توابع DisposableBean و Initialization در برنامه های خود استفاده نکنید . به جای آنها از XML بهره بگیرید ، زیرا انعطاف پذیری بهتری در نام دهی به متدهای برنامه شما دارد .

مثال عملی :

با ارایه یک مثال عملی در محیط کاری برنامه Eclipse ، مطالب آموزشی فوق رادر عمل بررسی خواهیم کرد . برای این منظور ، مراحل زیر را انجام دهید :

1. یک پروژه جدید با نام Spring Example را ایجاد نموده و همچنین در زیر پوشه src پروژه ، پکیج com.tahlildadeh را اضافه کنید .
2. همانطور که در درس آموزش ایجاد اولین برنامه Spring نشان دادیم ، به وسیله دکمه Add External JARs کتابخانه های مورد نیاز Spring را به پروژه خود اضافه کنید .
3. در پکیج Com.tahlildadeh کلاس های جاوا Main App و Hello World را اضافه نمایید .
4. در پوشه Src پروژه ، فایل پیکربندی اطلاعات Beans.Xml را اضافه کنید .
5. مرحله آخر تولید محتوای کلیه فایل های جاوا پروژه و تنظیم فایل پیکربندی اطلاعات به صورت زیر است :

```
package com.tahlildadeh;

public class HelloWorld {
private String message;
public void setMessage(String message){
this.message  = message;
}
public void getMessage(){
System.out.println("Your Message : " + message);
}
```

```
public void init(){
System.out.println("Bean is going through init.");
}
public void destroy(){
System.out.println("Bean will destroy now.");
}
}
```

کد زیر نیز مربوط به فایل MainApp. java می باشد . در این کد نیاز دارید تا یک تله Shutdown را به نام registershutdownHook() به کلاس AbstractApplicationContent اضافه کنید . این تله که درواقع یک متد destroy () است ، در هنگام ازبین رفتن هرشی رخ داده و کلیه عملیاتهای تعیین شده برای تخریب شی را انجام داده منابع سیستم (حافظه ، کش و...) را آزاد می کند :

```
package com.tahlildadeh;

import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {

      AbstractApplicationContext context = new
ClassPathXmlApplicationContext("Beans.xml");

      HelloWorld obj = (HelloWorld) context.getBean("helloWorld");
      obj.getMessage();
      context.registerShutdownHook();
   }
}
```

کد زیر نیز مربوط به فایل پیکربندی اطلاعات Bean.Xml است که در آن متدهای مربوط به destroy تعیین شده اند :

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


 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">


    id="helloWorld"
       class="com.tahlildadeh.HelloWorld"
       init-method="init" destroy-method="destroy">
        name="message" value="Hello World!"/>
```

پس از ایجاد فایل های اصلی برنامه و فایل پیکربندی اطلاعات ، میتوانید پروژه را اجرا نمایید . اگر همه چیز درست باشد ، برنامه خروجی زیر را تولید خواهد کرد :

```
Bean is going through init.
Your Message : Hello World!
Bean will destroy now.
```

متدهای مقداردهی اولیه و تخریب پیش فرض برنامه :

اگر در برنامه خود تعداد زیادی Bean با متدهای initialization و destroy با نام مشابه دارید ، لازم نیست تا برای هر Bean متدهای init-method و destroy-method را بنویسید .

به جای این کار میتوانید از قابلیت ویژه چهارچوب Spring برای تعیین متدهای پیش فرض تولید اولیه و تخریب Bean ها استفاده کنید . این متدها درصورتی که یک Bean ، متدهای از پیش تعیین شده نداشته باشد ،

برآن اعمال میشوند . برای این منظور بایستی از متدهای default-init-method و default-destroy-method در المنت <bean> برنامه به صورت زیر استفاده کنید :

```
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"
    default-init-method="init"
    default-destroy-method="destroy">

    id="..." class="...">
```

BeanPostProcessorچیست و چه کاربردی دارد

رابط کاربری BeanPostProcessor ، متدهای وابسته ای را تعیین می کند که به وسلیه آنها میتوانید دستورات برنامه نویسی مورد نظر خود را در برنامه اجرا کنید . شما همچنین می توانید یکسری منطق های کدنویسی را پس از اینکه Spring IOC Container یک شی Bean را مقدار دهی ، تنظیم و ایجاد نمود ، تعیین و اجرا نمایید . این کار بوسیله متصل کردن یک یا چند BeanPostProcessor به برنامه انجام می شود .

شما می توانید چندین رابط کاربردی BeanPostProcessor را در برنامه تعیین نموده و ترتیب اجرای آنها را بوسیله خاصیت crder که توسط رابط ordered هر BeanPostProcessor فراهم شده است ، کنترل کنید .

هر BeanPostProcessor برروی هرنمونه ایجاد شده ازیک Bean یا object عملیات انجام میدهد . به این معنا که ابتدا Spring IOC Container ، نسخه ای از هر Bean را تولید کرده و سپس BeanPostProcessor ، عملیات و کدهای خود را برروی آن انجام می دهد .

یک Application Context ، هر شی Bean ای که در هنگام پیاده سازی رابط Bean Post Processsor تعیین میشود را شناسایی میکند .

پس این Bean هارا به عنوان Post-Processors ثبت نموده ، تا Container برنامه در هنگام ایجاد هر Bean ، به موقع فراخوانی شود .

مثال عملی :

در مثال های عملی زیر به آموزش نوشتن ، ثبت و استفاده از BeanPostProcessor درون یک Applicationtext پرداخته ایم .

برای این منظور یک پروژه جدید Spring را در محیط Eclipse ایجاد نموده و مراحل زیر را انجام دهید :

1. یک پروژه جدید Spring با نام Spring Example را ایجاد نموده و سپس در زیر پوشه src یک پکیج به نام com.tahlildadeh ایجاد نمایید .
2. همانطور که در درس آموزش ایجاد اولین برنامه Spring نشان دادیم ، کتابخانه های لازم جهت برنامه خود را به وسیله دکمه Add External JARs به پروژه اضافه کنید .
3. در زیر مجموعه پکیج com.tahlildadeh ، کلاس های جاوا Hello World و Main App را ایجاد نمایید
4. در پوشه Src ، فایل پیکربندی اطلاعات Beans Configuration را با نام Beans.Xml ایجاد نمایید
5. در مرحله آخر نیز ، محتویات مورد نظر جهت فایل های جاوا ، فایل پیکربندی اطلاعات و سایر فایل ها را برای اجرای برنامه ایجاد نمایید .

کد زیر محتویات فایل HelloWorld.java را نشان میدهد :

```
package com.tutorialspoint;


public class HelloWorld {
```

```
   private String message;

   public void setMessage(String message){
      this.message  = message;
   }

   public void getMessage(){
      System.out.println("Your Message : " + message);
   }

   public void init(){
      System.out.println("Bean is going through init.");
   }

   public void destroy(){
      System.out.println("Bean will destroy now.");
   }
}
```

مثال فوق ، یک نمونه ساده از پیاده سازی BeanPostProcessor است که در آن نام یک Bean ، قبل و بعد از ایجاد هر نسخه از شی Bean ، چا میشود . شما میتوانید کدهای سطح بالاتری را نیز به قبل و بعد از ایجاد یک Bean اضافه نمایید ، زیرا از طریق هر دو متد موجود در Post Processor به آن Bean دسترسی کامل دارید .

کد زیر نیز محتویات فایل InitHelloWorld.java رانشان میدهد :

```
package com.tutorialspoint;

import org.springframework.beans.factory.config.BeanPostProcessor;
```

```
import org.springframework.beans.BeansException;


public class InitHelloWorld implements BeanPostProcessor {


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

      System.out.println("BeforeInitialization : " + beanName);

      return bean;  // you can return any other object as well

   }


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

      System.out.println("AfterInitialization : " + beanName);

      return bean;  // you can return any other object as well

   }


}
```

همچنین کد زیر محتویات فایل اصلی پروژه یا MainApp.java را نشان میدهد . در این فایل نیاز دارید تا یک تله shutdown را که توسط registerShutdownHook() تعیین شده است را در کلاس Abstroct Application Context قرار دهید . قرار دادن این تله shutdown باعث می شود تا برنامه یک خروج مطمئن داشته و پس از اتمام کار ، با اجرای متدهای destroy تعیین شده ، منابع سیستم به طور کامل آزاد شوند .

```
package com.tutorialspoint;


import org.springframework.context.support.AbstractApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;


public class MainApp {

   public static void main(String[] args) {
```

```
      AbstractApplicationContext context = new
ClassPathXmlApplicationContext("Beans.xml");

      HelloWorld obj = (HelloWorld) context.getBean("helloWorld");

      obj.getMessage();

      context.registerShutdownHook();

   }

}
```

کد زیر نیز محتویات مربوط به فایل پیکربندی اطلاعات Beans.xml رانشان داده که در آن متدهای init و destroy اشیای برنامه تعیین شده اند :

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

 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">

    id="helloWorld" class="com.tutorialspoint.HelloWorld"

       init-method="init" destroy-method="destroy">

        name="message" value="Hello World!"/>

    class="com.tutorialspoint.InitHelloWorld" />
```

پس از اینکه فایل های اصلی برنامه و فایل پیکربندی اطلاعات را درست کردیم ، میتوانیم برنامه را اجرا نماییم . اگر همه چیز درست باشد ، برنامه بایستی خروجی زیر را تولید نماید :

```
BeforeInitialization : helloWorld

Bean is going through init.

AfterInitialization : helloWorld

Your Message : Hello World!

Bean will destroy now.
```

آموزش مفهوم ارث بری در Bean های Spring

آشنایی با مفهوم ارث بری در Bean ها و کاربرد آن ؟

کد تعیین یک Bean می تواند شامل اطلاعات زیادی ازجمله اطلاعات پیکربندی ، آرگومان های تابع سازنده ، مقادیر خواص های آن و ...باشد . این اطلاعات همچنین می تواند اطلاعات خاصی رادرباره Container آن مثل متد تعریف اولیه (initialization method) ، متدهای پیش فرض و ... نیز شامل شود .

کد تعیین یک Bean فرزند (child) اطلاعات پیکربندی را از کد تعیین مادر خود به ارث میبرد . البته کدهای تعیین عنصر فرزند می تواند برخی مقادیر ارث برده شده را حذف کرده و یا تغییر دهد . همچنین اطلاعات خاصی را به آن اضافه کند .

ارث بری در تعریف Spring Bean ها ، ربطی به ارث بری کلاس ها در java ندارد ، اما چهارچوب کلی کار یکی است . شما میتوانید تعریف یک Bean مادر را به عنوان الگو قرار داده و هرچند فرزند که میخواهید خصوصیت آن را به ارث ببرد ، از رویش بسازید .

در فایل های پیکربندی اطلاعت مبتنی بر XML ، مادر یک child توسط خاصیت prent آن تعیین شده و بایستی برابر با نام مادر خود باشد .

مثال عملی :

برای درک بهتر مفهوم ارث بری در Spring ، یک پروژه عملی را با انجام مراحل زیر در نرم افزار Eclipse اجرا می کنیم :

1. یک پروژه جدید Spring با نام Spring Example را ایجاد نموده و سپس در زیر پوشه src یک پکیج به نام com.tahlildadeh ایجاد نمایید .
2. همانطور که در درس آموزش ایجاد اولین برنامه Spring نشان دادیم ، کتابخانه های لازم جهت برنامه خود را به وسیله دکمه Add External JARs به پروژه اضافه کنید .
3. در زیر مجموعه پکیج com.tahlildadeh ، کلاس های جاوا Hello World و Main App را ایجاد نمایید
4. در پوشه Src ، فایل پیکربندی اطلاعات Beans Configuration را با نام Beans.Xml ایجاد نمایید
5. در مرحله آخر نیز ، محتویات مورد نظر جهت فایل های جاوا ، فایل پیکربندی اطلاعات و سایر فایل ها را برای اجرای برنامه ایجاد نمایید .

کد زیر ، مربوط به فایل پیکربندی اطلاعات پروژه با نام Beans,xml است . در این فایل ما یک شی Bean به نام "helloWorld" را با دو خاصیت messag1 و messag2 تعریف کرده ایم . همچنین شی Bean به نام "helloIndia" را به عنوان فرزند شی "helloWorld" تعیین کرده ایم . برای این منظور مقدار خاصیت parent عنصر فرزند را برابر با نام عنصر مادر قرار داده ایم .

همانطور که در کد مثال مشاهده می کنید ، عنصر فرزند مقدار خاصیت message2 را از مادر خود به ارث برده ، ولی خاصیت message1 را منتقل نکرده و حذف شده است . همچنین این شی یک خاصیت جدید به نام message3 برای خود تعریف کرده است .

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


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">
```

```
   id="helloWorld" class="com.tahlildadeh.HelloWorld">
      name="message1" value="Hello World!"/>
      name="message2" value="Hello Second World!"/>


   id="helloIndia" class="com.tahlildadeh.HelloIndia" parent="helloWorld">
      name="message1" value="Hello India!"/>
      name="message3" value="Namaste India!"/>
```

کد زیر نیز محتویات فایل HelloWorld.java را نشان میدهد :

```
package com.tahlildadeh;

public class HelloWorld {
   private String message1;
   private String message2;

   public void setMessage1(String message){
      this.message1  = message;
   }

   public void setMessage2(String message){
      this.message2  = message;
   }

   public void getMessage1(){
      System.out.println("World Message1 : " + message1);
   }
```

```
   public void getMessage2(){
      System.out.println("World Message2 : " + message2);
   }
}
```

کد زیر نیز محتویات فایل HelloIndia,java را تعیین کرده است :

```
package com.tahlildadeh;

public class HelloIndia {
   private String message1;
   private String message2;
   private String message3;

   public void setMessage1(String message){
      this.message1  = message;
   }

   public void setMessage2(String message){
      this.message2  = message;
   }

   public void setMessage3(String message){
      this.message3  = message;
   }

   public void getMessage1(){
      System.out.println("India Message1 : " + message1);
   }
```

```
   public void getMessage2(){
      System.out.println("India Message2 : " + message2);
   }

   public void getMessage3(){
      System.out.println("India Message3 : " + message3);
   }
}
```

در نهایت نیز کد زیر مربوط به فایل MainApp.java است .

```
package com.tahlildadeh;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context = new
ClassPathXmlApplicationContext("Beans.xml");

      HelloWorld objA = (HelloWorld) context.getBean("helloWorld");

      objA.getMessage1();
      objA.getMessage2();

      HelloIndia objB = (HelloIndia) context.getBean("helloIndia");
      objB.getMessage1();
```

```
        objB.getMessage2();
        objB.getMessage3();
    }
}
```

پس از تعیین فایل های اصلی برنامه و فایل پیکربندی اطلاعات آن ، برنامه را اجرا کرده و خروجی زیر را تولد خواهد شد :

```
World Message1 : Hello World!
World Message2 : Hello Second World!
India Message1 : Hello India!
India Message2 : Hello Second World!
India Message3 : Namaste India!
```

اگر به خروجی مثال دقت کنید متوجه می شوید که ما خاصیت message2 را به شی "helloIndia" پاس نداده ایم ، اما به علت ارث بری از مادر خود ، دارای مقدار این خاصیت می باشد .

تعیین یک فایل تعریف Bean به عنوان الگو یا Template :

شما میتوانید تعریف یک Bean را به عنوان الگو تعیین کرده و با حداقل کدنویسی ، سایر عناصر فرزند مورد نظر خود را از روی آن بسازید .

class نبایستی خاصیت (Parent) به عنوان عنصر مادر Beanتوجه داشته باشید که در هنگام تعیین یک قرار دهید . همانطور که در کد زیر true آن را برابر abstractآن را مقدار دهی کرده و از طرف دیگر خاصیت نشان داده شده است :

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


 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">


    id="beanTeamplate" abstract="true">

       name="message1" value="Hello World!"/>

       name="message2" value="Hello Second World!"/>

       name="message3" value="Namaste India!"/>



    id="helloIndia" class="com.tahlildadeh.HelloIndia"
parent="beanTeamplate">

       name="message1" value="Hello India!"/>

       name="message3" value="Namaste India!"/>
```

در مثال فوق ، شی Bean مادر هیچ گاه نمی تواند تولید و مقدار دهی شده و هیچ نسخه ای از آن ساخته نمی شود . به دلیل اینکه تعریف آن ناقص بوده و همچنین به عنوان یک عنصر abstract (مطلق) تعیین شده است .

عناصر Bean ای که همانند کد فوق به صورت abstract تعیین می شوند ، فقط قادرند به عنوان یک الگو و template عمل کرده و عناصر دیگر از روی آنها ساخته شوند .

آموزش مفهوم Dependency Injection در Spring

هر برنامه ای که بر پایه جاوا طراحی شده ، شامل تعدادی شی (objet) است که با همکاری و ارتباط با یکدیگر خروجی نرم افزار را تولید نموده و به کاربر نشان میدهند .

هنگامی که یک برنامه سطح بالا جاوا را طراحی می کنید ، بایستی کلاس های برنامه تا حدامکان از همدیگر جدا بوده و مستقل عمل کنند . این قابلیت باعث می شود تا امکان استفاده مجدد از کلاس ها در سطح برنامه راحت تر شده و همچنین در فرآیند تست (Unit testing) کار ساده تری داشته باشیم . Dependecy Injection

که گاهی اوقات Wiring هم نامیده میشود ، به ما کمک میکند تا کلاس ها را به هم مرتبط کرده و در عین حال کاری میکند تا آنها مستقل از هم عمل کنند .

برای مثال فرض کنید برنامه ای دارید که دارای یک ویرایشگر متن بوده و می خواهید قابلیت بررسی املاء (Spell Checking) را نیز به آن اضافه کنید .

کد استاندارد برای انجام این کار به صورت زیر است :

```
public class TextEditor {

   private SpellChecker spellChecker;



   public TextEditor() {

      spellChecker = new SpellChecker();

   }

}
```

کاری که ما اینجا انجام می دهیم ، این است که یک Dependecy یا اتصال بین TextEditor و SpellChecker ایجاد میکنیم . اما در یک سناریو Inversion of Control یا IOC ، کاری که انجام می شود به صورت زیر است :

```
public class TextEditor {
   private SpellChecker spellChecker;

   public TextEditor(SpellChecker spellChecker) {
      this.spellChecker = spellChecker;
   }
}
```

در اینجا TextEditor نبایستی نگران اجرا شدن SpellChecker باشد . Spellchecker به صورت مستقل پیاده سازی شده و در هنگام اجرای اولیه TextEditor به آن ارجاع می شود . تمامی این فرآیند توسط چهارچوب کاری Spring کنترل می شود .

در مثال فوق ، به طور کامل کنترل را از دست TextEditor خارج کرده و آن را در جایی دیگر (برای مثال فایل پیکربندی Configuration) قرار داده ایم . از طرف دیگر Dependency (برای مثال کلاس SpellChecker) با استفاده از سازنده کلاس S(Class Constructor) به درون کلاس TextEditor تزریق شده است .

بنابراین جریان اجرای کدها توسط Dependency Injection به طور کامل معکوس شده است ، زیرا شما همه وابستگی های برنامه را به یک سیستم خارجی منتقل کرده اید .

روش دوم جهت تزریق Dependency استفاده از متدهای Setter Methods کلاس TextEditor است که در آن ما نسخه ای از شی SpellChecker را می سازیم . این نسخه از شی SpellChecker باعث فراخوانی متدهای Setter شده و خواص TextEditor را مقداردهی خواهد کرد .

بنابراین ، DI به دو روش کلی قابل پیاده سازی است که در جدول زیر به بررسی هردو روش می پردازیم :

تزریق برپایه سازنده (Counstructor-based dependency) :

Constructor-based DI زمانی انجام میشود که Container یک سازنده کلاس (Class Constructor) را با تعدادی آرگومان فراخوانی می کند . هرکدام از این آرگومان ها ، نماینده یک dependcy در کلاس دیگر می باشد .

تزریق برپایه متدهای Setter یا (Setter-based dependcy injection) :

Stter-based DI زمانی انجام می شود که Container متدهای Stter Methods را پس از فراخوانی یک سازنده بدون آرگومان (no-argument constructor) یا یک تابع بدون آرگومان ، جهت مقدار دهی اولیه bean برنامه ، اجرا می کند .

شما در یک برنامه Spring می توانید از هردو روش فوق باهم استفاده کنید . اما بهتر است از آرگومان های Constructor_based برای dependency های اختیاری استفاده کنید .

با استفاده از اصول DI کدهای تمیزتری خواهیم داشت و همچنین جداسازی کدها در زمانی که اشیا با dependency های آنها تعیین شده اند ، راحت تر خواهد بود .

در این روش برنامه نویسی ، اشیاء به وابستگی های خود (dependencyها) نگاه نکرده و نمی دانند کلاس یا وابستگی های خودشان کجا هستند . همه چیز توسط چهارچوب کاری Spring کنترل خواهد شد .

آموزش شی Inner Beans و کاربرد آن

همانطور که میدانید ، کلاسهای درونی در جاوا (Java inner classes) کلاس هایی هستند که درون محدوده کد یک کلاس دیگر تعریف می شوند . به همین صورت inner beans نیز اشیای bean ی هستند که در درون محدوده کد یک bean دیگر تعریف می شوند . بنابراین یک </bean> که درون یک تگ </property> یا </constructor-org> تعریف شود ، یک inner bean نامیده شده و به صورت مثال زیر کدنویسی می شود :

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

 <beans< span=""></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< span=""> id="outerBean" class="..."</bean<>>

      <property< span=""> name="target"</property<>>

         <bean< span=""> id="innerBean" class="..."</bean<>/>
```

مثال عملی کار با inner bean :

بیاید با اجرای یک مثال عملی ، نحوه تعریف inner bean و استفاده از آن را در محیط Eclipse مشاهده کنیم .

برای این منظور مراحل زیر را انجام دهید :

1. یک پروژه جدید با نام SpringExample را ایجاد نموده و همچنین در زیر پوشه src پروژه ، پکیج com.tahlildadeh را اضافه کنید .

2. همانطور که در درس آموزش ایجاد اولین برنامه Spring نشان دادیم ، به وسیله دکمه Add External JARs کتابخانه های مورد نیاز Spring را به پروژه خود اضافه کنید .

3. در پکیج Com.tahlildadeh کلاس های جاوا Main App ، SpellChecker و TextEditor را اضافه نمایید .

4. در پوشه Src پروژه ، فایل پیکربندی اطلاعات Beans.Xml را اضافه کنید .

5. مرحله آخر تولید محتوای کلیه فایل های جاوا پروژه و تنظیم فایل پیکربندی اطلاعات به صورت زیر است

محتویات فایل TextEditor.java بایستی به صورت زیر باشد :

```
public class TextEditor {

   private SpellChecker spellChecker;

    // a setter method to inject the dependency.

   public void setSpellChecker(SpellChecker spellChecker) {

      System.out.println("Inside setSpellChecker." );

      this.spellChecker = spellChecker;

   }

      // a getter method to return spellChecker

   public SpellChecker getSpellChecker() {

      return spellChecker;

   }

    public void spellCheck() {

      spellChecker.checkSpelling();
```

```
   }

}
```

کد زیر نیز مربوط به فایل کلاس وابسته (dependent class) با نام SpellChecker است :

```
package com.tahlildadeh;
 public class SpellChecker {
   public SpellChecker(){
      System.out.println("Inside SpellChecker constructor." );
   }
    public void checkSpelling(){
      System.out.println("Inside checkSpelling." );
   }
   }
```

همچنین کد فایل اصلی برنامه یا MainApp.java به صورت زیر است :

```
package com.tahlildadeh;
 import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
 public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");
       TextEditor te = (TextEditor) context.getBean("textEditor");
       te.spellCheck();
   }
}
```

در نهایت نیز کد زیر مربوط به فایل پیکربندی اطلاعات برنامه یا Beabs,Xml می باشد . در کد زیر از روش Setter-based برای DI استفاده شده و اشیاء به صورت innerbeans تعریف شده اند :

```
xml version="1.0" encoding="UTF-8"?>
 <beans< span=""></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< span=""></bean<> id="textEditor" class="com.tahlildadeh.TextEditor">
      <property< span=""></property<> name="spellChecker">
         <bean< span=""></bean<> id="spellChecker" class="com.tahlildadeh.SpellChecker"/>
```

پس از ایجاد فایل های اصلی و پیکربندی اطلاعات برنامه ، آن را در محیط Eclipse اجرا می کنیم . اگر همه کدها درست باشند ، خروجی زیر بایستی حاصل شود :

```
Inside SpellChecker constructor.
Inside setSpellChecker.
Inside checkSpelling.
```

آموزش تزریق اشیای مجموعه ای Injection Collection

در درس های قبل آموختید که چگونه انواع داده ای اصلی مثل int ، string و ... را با استفاده از خاصیت Value و یا رفرش از یک شی (object Refrence) به وسیله خاصیت ref در تگ <property> موجود در فایل پیکربندی اطلاعات (configuration file) ، تنظیم و ارسال نمایید .

هردوی موارد اشاره شده ، برای ارسال یک مقدار تکی به شی bean استفاده می شوند .

حال اگر بخواهیم مقدارهای چندتایی مثل Java Collection List، Set ، Map و یا properties را به یک شی bean ارسال کنیم ، راه حل چیست ؟. برای حل این مسئله ، Spring چهار نوع داده ای مجموعه را جهت تنظیم و ارسال مقادیر به شرح زیر را در اختیار ما قرار داده است :

- : از این نوع داده ای جهت ارسالی مجموعه ای از مقادیر ، حتی مقادیر تکراری (duplicate) استفاده می شود .
- : از نوع داده ای Set نیز برای ارسال مقادیر مجموعه ای (چندتایی) استفاده می شود ولی داده تکراری در این نوع مجاز نیست .
- : از نوع داده ای map ، جهت ارسال و تزریق مجموعه ای از داده ها ی جفتی -name (value) استفاده می شود که name و value می توانند از هرنوع داده ای باشند.
- : از نوع داده ای props نیز جهت ارسال و تزریق مجموعه ای از داده های جفتی -name (value) استفاده شده که در آن بایستی name و value هر دو از نوع string باشند.

مثال عملی :

جهت بهتر فهمیدن نحوه استفاده از Collection در Spring ، یک مثال عملی همراه با سورس کد را در محیط Eclipse ایجاد و اجرا می کنیم . برای این منظور مراحل زیر را انجام دهید :

1. یک پروژه جدید با نام SpringExample ایجاد کرده و یک پکیج با نام com.tahlildadeh را به پوشه src پروژه اضافه کنید .
2. همانطور که در درس آموزش ایجاد اولین برنامه Spring نشان دادیم ، کتابخانه های لازم جهت برنامه خود را به وسیله دکمه Add External JARs به پروژه اضافه کنید .
3. در زیر مجموعه پکیج com.tahlildadeh ، کلاس های جاوا JavaCollection و Main App را ایجاد نمایید .
4. در پوشه Src ، فایل پیکربندی اطلاعات Beans Configuration را با نام Beans.Xml ایجاد نمایید
5. در مرحله آخر نیز ، محتویات مورد نظر جهت فایل های جاوا ، فایل پیکربندی اطلاعات و سایر فایل ها را برای اجرای برنامه ایجاد نمایید .

کد زیر محتویات فایل JavaCollection.java را نشان می دهند :

```
package com.tahlildadeh;

import java.util.*;


public class JavaCollection {

   List addressList;

   Set  addressSet;

   Map  addressMap;

   Properties addressProp;


   // a setter method to set List

   public void setAddressList(List addressList) {
```

```
      this.addressList = addressList;
   }

   // prints and returns all the elements of the list.
   public List getAddressList() {
      System.out.println("List Elements :"  + addressList);
      return addressList;
   }

   // a setter method to set Set
   public void setAddressSet(Set addressSet) {
      this.addressSet = addressSet;
   }

   // prints and returns all the elements of the Set.
   public Set getAddressSet() {
      System.out.println("Set Elements :"  + addressSet);
      return addressSet;
   }

   // a setter method to set Map
   public void setAddressMap(Map addressMap) {
      this.addressMap = addressMap;
   }

   // prints and returns all the elements of the Map.
   public Map getAddressMap() {
      System.out.println("Map Elements :"  + addressMap);
      return addressMap;
```

```
   }

   // a setter method to set Property
   public void setAddressProp(Properties addressProp) {
      this.addressProp = addressProp;
   }

   // prints and returns all the elements of the Property.
   public Properties getAddressProp() {
      System.out.println("Property Elements :"  + addressProp);
      return addressProp;
   }
}
```

همچنین کد زیر محتویات فایل اصلی برنامه MainApp.java را شامل می شود:

```
package com.tahlildadeh;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context =
             new ClassPathXmlApplicationContext("Beans.xml");

      JavaCollection jc=(JavaCollection)context.getBean("javaCollection");

      jc.getAddressList();
      jc.getAddressSet();
```

```
      jc.getAddressMap();
      jc.getAddressProp();
   }
}
```

در نهایت نیز ، کد فایل پیکربندی اطلاعات برنامه یا Beans.xml نیز بایستی به صورت زیر باشد:

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


 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">



    id="javaCollection" class="com.tahlildadeh.JavaCollection">
       name="addressList">

            INDIA
            Pakistan
            USA
            USA
       name="addressSet">

            INDIA
            Pakistan
            USA
            USA

       name="addressMap">
```

```
            key="1" value="INDIA"/>
            key="2" value="Pakistan"/>
            key="3" value="USA"/>
            key="4" value="USA"/>
      name="addressProp">

            key="one">INDIA
            key="two">Pakistan
            key="three">USA
            key="four">USA
```

پس از اینکه فایل های کد و فایل اصلی برنامه را ایجاد کردیم ، می توانیم خروجی آن را در محیط Eclipse تولید کنیم . در صورت درست بودن فایل های کد ها ، خروجی برنامه بایستی به صورت زیر باشد:

```
List Elements :[INDIA, Pakistan, USA, USA]
Set Elements :[INDIA, Pakistan, USA]
ap Elements :{1=INDIA, 2=Pakistan, 3=USA, 4=USA}
Property Elements :{two=Pakistan, one=INDIA, three=USA, four=USA}
```

تزریق رفرنس های Bean (Injecting Bean Refrence) :

کد تعریف Bean زیر یا (Bean Definition) ، به شما کمک می کند نحوه تزریق رفرنس های یک Bean را به عنوان یک المنت Collection بهتر متوجه شوید . همانطور که در کد زیر می بینید ، حتی شما می توانید رفرنس ها و مقادیر را در یک کد با هم ترکیب نمایید :

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

 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">

    id="..." class="...">

       name="addressList">

             bean="address1"/>
             bean="address2"/>
            Pakistan

       name="addressSet">

             bean="address1"/>
             bean="address2"/>
            Pakistan
       name="addressMap">

             key="one" value="INDIA"/>
             key ="two" value-ref="address1"/>
             key ="three" value-ref="address2"/>
```

برای استفاده از تعریف Bean فوق ، شما بایستی متدهای Setter Methods خود را به گونه ای تعیین کنید تا بتوانند رفرنس های شی Bean را مدیریت نمایند .

آموزش نحوه تزریق مقادیر null یا رشته های خالی :

اگر نیاز داشتید تا یک مقدار متنی خالی (empte string) را به برنامه پاس دهید ، بایستی از کد زیر استفاده کنید :

```
id="..." class="exampleBean">
```

```
    name="email" value=""/>
```

کد مثال فوق ، دقیقا معادل اجرای کد Java زیر است :

همچنین اگر می خواهید یک مقدار کد یا null را به برنامه پاس دهید ، بایستی از کدی همانند زیر استفاده کنید :

```
id="..." class="exampleBean">

    name="email">
```

کد مثال فوق ، دقیقا معادل اجرای کد Java زیر است :

```
exampleBean.setEmail(null)
```

آموزش Beans Auto-Wiring در Spring

آشنایی با Beans Auto-Wiring و کاربرد آن

در درس های قبلی ، با نحوه تعریف اشیای beans به وسیله المنت <bean> و همچنین تزریق <bean> به درون فایل های XML پیکربندی اطلاعات برنامه (Configuration File) به وسیله تگ های <property> و <constructor-org> آشنا شوید .

Spring Container می تواند ارتباطات بین اشیای Beans یک برنامه را با استفاده از عمل اتصال خودکار (antowire) و بدون کمک از المنت های <property> و <constructor-org> انجام دهد .

عمل Autowire باعث کاهش حجم کدنویسی فایل XML پیکربندی اطلاعات ، مخصوصا در برنامه های بزرگ مبتنی بر Spring می شود .

حالت های مختلف Autowiring در Spring

در جدول زیر به معرفی روش های مختلف انجام عمل autowiring پرداخته ایم . این روش ها به Spring Container کمک می کنند بیواند از عمل autowiring در dependency injection استفاده کند . شما می

توانید از خاصیت antowire المنت <bean/> برای تعیین روش autowiring آن در تعریف شی bean استفاده کنید .

- no : این مد ، حالت پیش فرض بوده و به معنای این است که از عمل autowiring استفاده نمی شود . در این حالت بایستی از refrence صیرح شی bean برای عمل اتصال (wiring) استفاده کنید . برای این روش نیاز نیست کار خاصی انجام داده و همانند آنچه است که در بخش های قبلی آموزش DI نشان دادیم .
- by Name : در این حالت عمل autowiring بر مبنای خاصیت name صورت میگیرد . Spring Container به خواص اشیای bean ی که خاصیت autowire آنها برروی مقدار name تنظیم شده اند ، نگاه می کند . سپس تلاش می کند تا اشیای bean با نام مشترک در خاصیت autowire را پیدا کرده و خواص آنها را به هم مرتبط سازد .
- by Type : در این حالت عمل autowiring بر مبنای خاصیت datatype صورت می گیرد . Spring Container به خواص اشیای bean ی که خاصیت autowire آنها برروی byType تنظیم شده اند ، در فایل XML Configuration نگاه می کند .
- سپس تلاش می کند تا یک المنت property را که مقدار خاصیت type آن دقیقا با مقدار خاصیت نام یک bean (خاصیت name) در فایل XML Configuration برابر است را پیدا نموده و اُنها را به هم متصل (wire) کند .
- اگر بیش از یک نمونه از مورد فوق وجود داشته باشد ، در برنامه خطای جدی (fatal) رخ می دهد .
- Constructor : این حالت نیز بسیار شبیه حالت by Type است با این تفاوت که بر مبنای آرگومان های تابع سازنده (Constructor arguments) رخ می دهد .

اگر دقیقا یک نمونه از Constructor argument در فایل XML وجود نداشته باشد ، خطای جدی (fatal) رخ می دهد .

- autodetect : در این حالت به جستجوی خودکار نامیده می شود ، Spring ابتدا تلاش می کند عمل autowiring را برمبنای Constructor انجام دهد . اگر ؟؟؟ قبلی امکان پذیر نباشد ، حالت by Type را امتحان خواهد کرد .

شما می توانید از ودهای byType و Constructor برای اتصال آرایه و سایر داده های مجموعه ای استفاده کنید .

محدودیت های کار با autowiring

اگر در تمام سطح یک پروژه از autowiring استفاده کنید ، این روش به بهترین وجه جواب خواهد داد . اما اگر در کل یک پروژه از autowiring استفاده نشود ، ممکن است باعث سردرگمی برنامه نویسان شده و آنها حفظ تعداد ودی از bean definitions را با این روش انجام دهد .

بنابراین ، autowiring به صورت قابل توجهی نیاز به تعیین properties و Constructor arguments را در برنامه کاهش می دهد . اما بایستی به محدودیت ها و اشکالات عمده autowiring ، قبل از استفاده آنها توجه کنید . به برخی از این محدودیت ها در جدول زیر اشاره کرده ایم :

امکان نادیده گرفته شدن (overriding possibility) :

در هنگام استفاده از autowiring ، شما همچنین می توانید dependency ها را با استفاده از المنت های <property> و <constructor-org> نیز انجام دهید . در صورت استفاده از المنت های ذکر شده ، برنامه autowiring را نادیده خواهد گرفت .

- استفاده جهت انواع داده ای پایه (Primitive data types) : شما نمی توانید از قابلیت autowiring برای نوع های داده ای پایه مثل int ، string ، class و ... استفاده کنید .

- طبیعت پیچ کننده : عملیات autowiring نسبت به wiring صریح کمتر دقیق است . بنابراین در صورت امکان از wiring صریح استفاده کنید .

آموزش پیکربندی فایل ها بر پایه Annotation در Spring

از نسخه 2.5 چهارچوب کاری Spring ، این امکان به وجود آمد تا dependency injection را به وسیله annotations انجام دهیم . بنابراین به جای استفاده از XML برای تشریح یک bean wiring ، می توانید تنظیمات پیکربندی را به خود کلاس شی انتقال دهید . در این حالت annotations را در کلاس مربوطه ، متدها و یا فیلدهای تعریف داده به کار میبریم .

Annotation injection قبل از XML injection اجرا می شود . بنابراین تنظیماتی که توسط XML تعیین شده باشند ، تنظیمات قبلی که توسط annotation بر properties ها انجام شده است را نادیده گرفته و تنظیمات خود را اعمال می کند .

Annotation wiring به صورت پیش فرض در چهارچوب کاری Spring فعال نیست . بنابراین ، قبل از اینکه بتوانیم از annotation-based wiring (اتصال برمبنای Annotation) استفاده کنیم ، بایستی آن را در فایل پیکربندی Spring Configuration فعال کنیم .

از تنظیمات ارایه شده در کد زیر برای فعال نمودن annotation در پروژه های Spring استفاده کنید . تنظیمات زیر در فایل Configuration file پروژه اعمال می شود :

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


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-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">
```

به محض اینکه ، المنت <context:annotation_config/> تنظیم شود ، شما می توانید از annotation برای ویرایش کدهای خود استفاده کنید . این کار تعیین می کنند کد Spring بایستی به صورت

اتوماتیک مقادیر را به properties ، methods و constructors متصل کند .

در جدول زیر به بررسی برخی از پارامترهای مهم annotation پرداخته شده است :

- @Required annotation: @Required به خواص شی bean در متدهای setter اعمال می شود .
- @Antowired annotation: @Antowired می تواند به خواص شی در متدهای Setter ، متدهای non-setter ، سازنده ها (Constructor) و سایر خواص اعمال شود .
- @Qualifier annotation: @Qualifier به همراه @Antowired برای تعیین اینکه دقیقا کدام bean در برنامه متصل (wire) شود ، به کار می رود .
- JSR-250 Annotations: چهارچوب Spring از annotation های مبتنی بر JSR-250 شامل @Resource ، @PostConstruct و @PreDestory پشتیبانی میکند .

آموزش نوشتن فایل پیکربندی اطلاعات مبتنی بر جاوا

در درس های قبل ، به نحوه تنظیم Spring beans به وسیله فایل های پیکربندی مبتنی بر XML آشنا شدید . اگر با نوشتن کدهای پیکربندی اطلاعات به زبان XML مشکلی ندارید ، به نظر من نیازی نیست تا روش نوشتن این فایل را به زبان جاوا را نیزیاد بگیرید . زیرا در هر دو روش به نتیجه یکسانی رسیده و فایل پیکربندی اطلاعات مبتنی بر XML یا مبتنی بر جاوا خروجی یکسانی دارند .

گزینه نوشتن دستورات فایل پیکربندی اطلاعات برنامه برپایه جاوا ، شما را قادر می سازد بیشتر کدهای تنظیم Spring را بدون نیاز به XML و به وسیله annotation های مبتنی بر جاوا که در بخش های بعدی ، تشریح می کنیم ، بنویسید .

المنت های @Bean Annotation و @Configuration

Annotating یک کلاس به وسیله المنت @Configuration مشخص می کند که آن کلاس می تواند توسط Spring IOC Container به عنوان یک منبع جهت تعیین و تنظیم Bean ها به کار رود .

از طرف دیگر Bean annotation@ به Spring اعلام می کند که یک متد مشخص شده به وسیله Bean@ ، یک object ای را به برنامه باز می گرداند که بایستی در محتوی برنامه Spring ، به عنوان یک Bean در نظر گرفته شود .

ساده ترین حالت یک کلاس Configuration@ ، می تواند به صورت زیر باشد :

```
package com.tahlildadeh ;

import org.springframework.context.annotation.*;


@Configuration
public class HelloWorldConfig {


   @Bean
   public HelloWorld helloWorld(){
      return new HelloWorld();
   }
}
```

کد کلاس فوق ، معادل کد فایل پیکربندی XML زیر خواهد بود :

```
id="helloWorld" class="com.tahlildadeh .HelloWorld" />
```

در اینجا ، نام متدی که به وسیله Bean@ در کد annotation شده ، به عنوان یک bean ID عمل کرده و bean واقعی را ایجاد نموده و به برنامه باز می گرداند . کلاس Configuration شما می تواند بیش از اعلان برای Bean@ داشته باشد .

پس از اینکه کلاس های Configuration شما تعیین شدند ، می توانید آنها را به وسیله AnnotationConfigApplicationContext برای Spring Container فراهم کرده و لود نمایید ، به صورت زیر :

```
public static void main(String[] args) {
   ApplicationContext ctx =
   new AnnotationConfigApplicationContext(HelloWorldConfig.class);

   HelloWorld helloWorld = ctx.getBean(HelloWorld.class);

   helloWorld.setMessage("Hello World!");
   helloWorld.getMessage();
}
```

همچنین می توانید چندین کلاس Configuration را به وسیله زیر برای برنامه لود کنید :

```
public static void main(String[] args) {
   AnnotationConfigApplicationContext ctx =
   new AnnotationConfigApplicationContext();

   ctx.register(AppConfig.class, OtherConfig.class);
   ctx.register(AdditionalConfig.class);
   ctx.refresh();

   MyService myService = ctx.getBean(MyService.class);
   myService.doStuff();
}
```

مثال عملی :

برای درک ، یک مثال عملی جهت مطالب ارایه شده در محیط Eclipse اجرا می کنیم . برای این منظور مراحل زیر را انجام دهید :

1. یک پروژه جدید با نام SpringExample ایجاد کرده و یک پکیج با نام com.tahlildadeh را به پوشه src پروژه اضافه کنید .
2. همانطور که در درس آموزش ایجاد اولین برنامه Spring نشان دادیم ، کتابخانه های لازم جهت برنامه خود را به وسیله دکمه Add External JARs به پروژه اضافه کنید .
3. در زیر مجموعه پکیج com.tahlildadeh ، کلاس های جاوا HelloWorldConfig ، MainApp و HelloWorld را ایجاد نمایید .
4. در پوشه Src ، فایل پیکربندی اطلاعات Beans Configuration را با نام Beans.Xml ایجاد نمایید .
5. در مرحله آخر نیز ، محتویات مورد نظر جهت فایل های جاوا ، فایل پیکربندی اطلاعات و سایر فایل ها را برای اجرای برنامه ایجاد نمایید .

کد زیر ، محتویات فایل HelloWorldConfig.java را نشان می دهد :

```
package com.tutorialspoint;

import org.springframework.context.annotation.*;


@Configuration

public class HelloWorldConfig {


   @Bean

   public HelloWorld helloWorld(){

      return new HelloWorld();

   }

}
```

همچنین کد زیر مربوط به فایل HelloWorld.java می باشد :

```
package com.tutorialspoint;
```

```
public class HelloWorld {
   private String message;

   public void setMessage(String message){
      this.message  = message;
   }

   public void getMessage(){
      System.out.println("Your Message : " + message);
   }
}
```

در نهایت نیز کد فایل MainApp.java بایستی به صورت زیر باشد :

```
package com.tutorialspoint;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.*;

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext ctx =
      new AnnotationConfigApplicationContext(HelloWorldConfig.class);

      HelloWorld helloWorld = ctx.getBean(HelloWorld.class);

      helloWorld.setMessage("Hello World!");
      helloWorld.getMessage();
```

```
    }
}
```

پس از اینکه کلید فایل های اصلی برنامه را ایجاد کرده و کتابخانه های مورد نیاز را نیز به پروژه اضافه کردید ، آن را اجرا خواهیم کرد . در این مثال بایستی توجه داشته باشید که دیگر نیازی به فایل پیکربندی اطلاعات Configuration ، نخواهیم داشت .

اگر همه چیز در برنامه درست بوده باشد ، خروجی زیر را تولید خواهد شد :

```
Your Message : Hello World!
```

تزریق Bean Dependencies :

وقتی که یک @Bean ، وابستگی هایی (dependencies) به یک Bean دیگر دارد ، تشریح این وابستگی بسیار ساده خواهد بود . برای این منظور کدی همانند مثال زیر می نویسیم که در آن یک bean method ، شی bean دیگری را فراخوانی می کند :

```
package com.tutorialspoint;
import org.springframework.context.annotation.*;

@Configuration
public class AppConfig {
   @Bean
   public Foo foo() {
      return new Foo(bar());
   }
   @Bean
   public Bar bar() {
      return new Bar();
   }
}
```

در کد مثال فوق ، شی foo bean یک refrence را از طریق تزریق آرگومان سازنده (Constructor) دریافت می کند . برای مثال درک بهتر مطلب ، یک مثال عملی نیز در ادامه ارایه می دهیم .

مثال عملی :

1. یک پروژه جدید با نام SpringExample ایجاد کرده و یک پکیج با نام com.tahlildadeh را به پوشه src پروژه اضافه کنید .
2. همانطور که در درس آموزش ایجاد اولین برنامه Spring نشان دادیم ، کتابخانه های لازم جهت برنامه خود را به وسیله دکمه Add External JARs به پروژه اضافه کنید .
3. در زیر مجموعه پکیج com.tahlildadeh ، کلاس های جاوا TextEditorConfig ، TextEditor ، SpellChecker و MainApp را ایجاد نمایید .
4. در پوشه Src ، فایل پیکربندی اطلاعات Beans Configuration را با نام Beans.Xml ایجاد نمایید
5. در مرحله آخر نیز ، محتویات مورد نظر جهت فایل های جاوا ، فایل پیکربندی اطلاعات و سایر فایل ها را برای اجرای برنامه ایجاد نمایید .

کد زیر ، محتویات فایل TexEditorConfig.java را نشان می دهد :

```
package com.tutorialspoint;

import org.springframework.context.annotation.*;


@Configuration
public class TextEditorConfig {


   @Bean
   public TextEditor textEditor(){
      return new TextEditor( spellChecker() );
   }


   @Bean
   public SpellChecker spellChecker(){
      return new SpellChecker( );
```

```
    }
}
```

همچنین کد زیر مربوز به فایل TexEditor.java است :

```
package com.tutorialspoint;


public class TextEditor {
   private SpellChecker spellChecker;


   public TextEditor(SpellChecker spellChecker){
      System.out.println("Inside TextEditor constructor." );
      this.spellChecker = spellChecker;
   }
   public void spellCheck(){
      spellChecker.checkSpelling();
   }
}
```

کد زیر نیز ، محتویات یک کلاس وابسته (dependent class) دیگر برنامه به نام SpellChecker.java را نشان می دهد :

```
package com.tutorialspoint;


public class SpellChecker {
   public SpellChecker(){
      System.out.println("Inside SpellChecker constructor." );
   }


   public void checkSpelling(){
```

```
      System.out.println("Inside checkSpelling." );
   }

}
```

در نهایت نیز کد فایل Main App.java به صورت زیر است :

```
package com.tutorialspoint;


import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.*;


public class MainApp {
   public static void main(String[] args) {
      ApplicationContext ctx =
      new AnnotationConfigApplicationContext(TextEditorConfig.class);


      TextEditor te = ctx.getBean(TextEditor.class);


      te.spellCheck();
   }
}
```

پس از اینکه فایل های اصلی برنامه و کتابخانه های لازم آن را اضافه کردید ، می توانید پروژه را اجرا کنید . در این مثال هم توجه داشته باشید که نیازی به فایل پیکربندی اطلاعات مبتنی بر XML نخواهیم داشت . اگر همه چیز برنامه درست باشد ، بایستی خروجی زیر تولید شود :

```
Inside SpellChecker constructor.
Inside TextEditor constructor.
Inside checkSpelling.
```

آموزش @Inport Annotation

@Inport Annotation به ما امکان میدهد تا @Bean Definitions (توضیحات یک Bean) را از یک کلاس Configuration دیگر لود و فراخوانی کنیم . فرض کنید که کد کلاس ConfigA به صورت زیر باشد :

```
@Configuration
public class ConfigA {
   @Bean
   public A a() {
      return new A();
   }
}
```

شما می توانید کد تشریح یک Bean را در تعریف یک Bean دیگر به صورت زیر وارد کنید :

```
@Configuration

@Import(ConfigA.class)

public class ConfigB {

   @Bean

   public B a() {

      return new A();

   }

}
```

اکنون ، وقتی که می خواهیم کد برنامه را بنویسیم ، به جای اینکه نیاز باشد کد هر دو کلاس ConfigA.class و ConfigB.class را تشریح کنیم ، فقط نیاز است کلاس ConfigB را همانند زیر اعلام کنیم :

```
public static void main(String[] args) {
   ApplicationContext ctx =
   new AnnotationConfigApplicationContext(ConfigB.class);
   // now both beans A and B will be available...
   A a = ctx.getBean(A.class);
   B b = ctx.getBean(B.class);
}
```

آمـوزش Lifecycle Callbacks

امکان تعریف متد های callback تعریف اولیه و تخریب شی Bean را همانند آنچه در @Bean annotation توابع init-method و destroy-method کدنویسی Spring XML داشتیم را به ما میدهد . نحوه انجام کار به صورت زیر است :

```
public class Foo {
   public void init() {
      // initialization logic
   }
   public void cleanup() {
      // destruction logic
   }
}

@Configuration
public class AppConfig {
   @Bean(initMethod = "init", destroyMethod = "cleanup" )
   public Foo foo() {
      return new Foo();
   }
}
```

تعیین محدوده شی Bean یا Bean Scope

Scope پیش فرض یک Bean ، مقدار Singleton است . اما می توانید به وسیله المنت @Scope ، آن را به صورت زیر تغییر دهید :

```
@Configuration
public class AppConfig {
   @Bean
   @Scope("prototype")
   public Foo foo() {
      return new Foo();
   }
}
```

آموزش مدیریت رویداد (Event Hhanding) در Spring

آموزش مفهوم Event Handling (مدیریت رویدادها) در Spring

در کلیه مراحل آموزشی قبلی Spring ، متوجه شدید که بخش اصلی یک برنامه Spring محتوی ApplicationContext است که چرخه حیات کامل Bean ها در برنامه کنترل می کند .

ApplicationContext ، رویدادهای مشخصی را در هنگام لود و فراخوانی Bean های برنامه اجرا می کند . برای مثال ، رویداد ContextStartedEvent هنگامی مه Context شروع به لود کرده ، اجرا شده و رویداد ContextStopedEvent در هنگام پایان لود شدن Context رخ می دهد .

مدیریت رویدادها (Event Handling) در ApplicationContext را اجرا کند ، هربار که کلاس ApplicationEvent به ApplicationContext ارسال شود ، شی Bean از این عمل مطلع خواهد شد .

Spring شامل رویدادهای استاندارد زیر می باشد که به تشریح هر یک خواهیم پرداخت :

- ContextResfreshed : این رویداد زمانی مه محتوی ApplicatinContext Event مقدار دهی اولیه (initialized) یا تازه (Refresh) می شود ، رخ می دهد . همچنین این رویداد را می توان با استفاده از متد ()refresh رابط کاربری ApplicationContext اجرا نمود .
- ContextStoppedEvent این رویداد زمانی که اجرا ApplicatinContext به استفاده از متد ()stop رابط کاربری Configurable متوقف می شود ، رخ میدهد .

پس از متد ، می توانید کارهای مربوط به نگهداری و بهینه سازی سیستم را انجام دهید.

- ContextClosedEvent : این رویداد زمانی رخ می دهد که ApplicationContext با استفاده از متد ()Closed رابط کاربری Con بسته می شوید . context ای که بسته شود ، به انتهای چرخه حیات کاری خود خواهد رسید . این Context دیگر نمی تواند لود مجدد (Refreshed) یا شروع مجدد (restart) شود .
- RequestHandledEvent : این رویداد یک event مختص فضای وب بوده و به کلیه Bean های برنامه اعلام می کند که به یک درخواست HTTP سرویس .

نکته مهم :

مدیریت رویدادها در Spring ، یک پروسه تک عملیات (single thread) است . بنابراین زمانی که یک رویدا اجرا می شود ، تا زمانی که کلیه گیرنده های آن رویداد پیام خود را دریافت کرده و به طور کامل اجرا شود ، سایر پروسه های برنامه متوقف شده و روال برنامه stop می شود .

بنایراین اگر مدیریت رویداد (event Handling) در برنامه شما زیاد به کار می روید ، بایستی در طراحی آن بسیار دقت کنید .

گوش دادن به رویدادهای Context :

برای گوش دادن به یک رویداد Context ، هر Bean بایستی رابط کاربری ApplicationListener را که فقط دارای یک متد به نام ()onApplicationEvent را اجرا کند .

برای درک بهتر مطالب ارایه شده ، مثال عملی را مطرح می کنیم تا ببینیم رویدادها (events) چگونه منتشر شده و شما چگونه می توانید کدهای خود را طوری بنویسید تا اجرای آنها بسته به رخ دادن event ها باشند .

مثال عملی را با طی مراحل زیر در محیط Eclipse طراحی می کنیم :

1. یک پروژه جدید به نام SpringExample و یک پکیج جدید را به نام com.tahlildadeh را در پوشه src برنامه ایجاد کنید .
2. همانطور که در بخش نوشتن اولین برنامه با Spring آموزش دادیم ، کتابخانه های لازم برنامه را با استفاده از Extered JARs به پروژه اضافه کنید .
3. کلاس های HelloWorld ، CStartEvantHandler ، CStopEventHandler و MainApp را در پکیج com.tahlildadeh ایجاد کنید .
4. در پوشه src پروژه ، فایل پیکربندی اطلاعات Beans.xml را ایجاد کنید .
5. مرحله آخر ایجاد محتوای کلیه فایل های جاوا پروژه و همچنین فایل پیکربندی اطلاعات Beansها می باشد که در مراحل زیر توضیح داده شده است .

کد زیر را در فایل Hello.World.java قرار دهید :

```
package com.tutorialspoint;
```

```
public class HelloWorld {

   private String message;

   public void setMessage(String message){
      this.message  = message;
   }

   public void getMessage(){
      System.out.println("Your Message : " + message);
   }
}
```

همچنین محتوای فایل CStartEventHandler.java به صورت زیر است :

```
package com.tutorialspoint;

import org.springframework.context.ApplicationListener;
import org.springframework.context.event.ContextStartedEvent;

public class CStartEventHandler
   implements ApplicationListener<ContextStartedEvent>{

   public void onApplicationEvent(ContextStartedEvent event) {
      System.out.println("ContextStartedEvent Received");
   }
}
```

کد زیر نیز مربوط به فایل CStopEventHandler.java است :

```
package com.tutorialspoint;


import org.springframework.context.ApplicationListener;
import org.springframework.context.event.ContextStoppedEvent;


public class CStopEventHandler
   implements ApplicationListener<ContextStoppedEvent>{


   public void onApplicationEvent(ContextStoppedEvent event) {
      System.out.println("ContextStoppedEvent Received");
   }
}
```

محتویات فایل اصلی برنامه MainApp.java به این صورت است :

```
package com.tutorialspoint;


import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;


public class MainApp {
   public static void main(String[] args) {
      ConfigurableApplicationContext context =
      new ClassPathXmlApplicationContext("Beans.xml");


      // Let us raise a start event.
      context.start();


      HelloWorld obj = (HelloWorld) context.getBean("helloWorld");
```

```
      obj.getMessage();

      // Let us raise a stop event.
      context.stop();
   }
}
```

در نهایت نیز ، کد فایل پیکربندی اطلاعات برنامه Bean.xml به صورت زیر تعریف می شود :

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

 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">

    id="helloWorld" class="com.tutorialspoint.HelloWorld">
       name="message" value="Hello World!"/>

    id="cStartEventHandler"
         class="com.tutorialspoint.CStartEventHandler"/>

    id="cStopEventHandler"
         class="com.tutorialspoint.CStopEventHandler"/>
```

پس از اینکه فایل های برنامه را ایجاد کردیم ، می توانیم آن را اجرا کنیم . در صورت درست بودن همه موارد ، خروجی زیر تولید خواهد شد ،

```
ContextStartedEvent Received

Your Message : Hello World!

ContextStoppedEvent Received
```

در آخر هم اگر دوست داشتید ، می توانید رویدادهای دلخواه خود را نوشته و عملکرد آنها را در برنامه تست کنید .

برای نوشتن رویدادهای دلخواه خود ، به درس آموزش نوشتن Custom Events بروید .

```
<a href="http://www.tahlildadeh.com/">Visit W3Schools.com!</a>
```

آموزش نوشتن Custom Events در Spring

برای نوشتن یک رویداد دلخواه در Spring ، بایستی چند مرحله را انجام دهید . در این درس به صورت گام به گام همراه با سورس عملی مثال ، نحوه نوشتن یک Custom Event را آموزش داده ایم .

کارهای زیر را در نرم افزار Eclipse انجام دهید :

1. یک پروژه جدید به نام Spring Example و یک پکیج جدید به نام com.tahlildadeh را در پوشه src برنامه ایجاد کنید . کلیه کلاس های دیگر برنامه ، بایستی در این پوشه تعریف شوند .
2. به وسیله ابزار Add External JARs همانطور که درس نوشتن اولین برنامه Spring آموزش دادیم دادیم ، کتابخانه های لازم را به پروژه اضافه کنید .
3. با گسترش بخش ApplicationEvwnt یک کلاس جدید به نام CustomEvent را اضافه کنید . این کلاس بایستی دارای یک متد سازه پیش فرض default Constructor بوده که از کلاس ApplicationEvent مشتق شده باشد .
4. به محض اینکه event Class Publisher را ایجاد کرده که باعث اجرای ApplicationEventPublisherAwarie می شود .

از طرف دیگر شما نیاز دارید تا این کلاس را در فایل پیکربندی اطلاعات به عنوان یک Bean تعریف کنید . این کار باعث می شود تا Container بتواند Bean مورد نظر را به عنوان یک اجراکننده رویداد Publisher)

(Event شناسایی کرده ، چرا که خود آن Bean باعث اجرای رابط کاربری ApplicationEventpublisherAware می شود .

5. یک event منتشر شده را می توان درون یک کلاس مدیریت نمود . برای این منظور ما کلاس EventClassHandler را انتخاب کرده که خود باعث اجرای رابط کاربری ApplicationListener و متد onApplicationEvent برروی event مورد نظر ما می شود .

6. فایل پیکربندی اطلاعات Bean.xml را در پوشه src ایجاد کرده و کلاس MainApp را نیز که به عنوان هسته اصلی برنامه می باشد ، به پروژه اضافه کنید .

7. مرحله آخر نیز ایجاد محتوی تمامی فایل های جاوا برنامه و مقدار دهی فایل پیکربندی اطلاعات است تا بتوان پروژه را اجرا نمود .

کد زیر مربوط به فایل CustomEvent.java می باشد :

```
package com.tutorialspoint;

import org.springframework.context.ApplicationEvent;

public class CustomEvent extends ApplicationEvent{

   public CustomEvent(Object source) {
      super(source);
   }

   public String toString(){
      return "My Custom Event";
   }
}
```

در ادامه نیز کد مربوط به فایل CustomEventPublisher.java قرار داده شده است :

```
package com.tutorialspoint;


import org.springframework.context.ApplicationEventPublisher;

import org.springframework.context.ApplicationEventPublisherAware;


public class CustomEventPublisher

   implements ApplicationEventPublisherAware {


   private ApplicationEventPublisher publisher;


   public void setApplicationEventPublisher

               (ApplicationEventPublisher publisher){

      this.publisher = publisher;

   }


   public void publish() {

      CustomEvent ce = new CustomEvent(this);

      publisher.publishEvent(ce);

   }

}
```

کد زیر نیز محتوای فایل CustomEventHandler.java را نشان می دهد :

```
package com.tutorialspoint;

import org.springframework.context.ApplicationListener;

public class CustomEventHandler
   implements ApplicationListener<CustomEvent>{

   public void onApplicationEvent(CustomEvent event) {
      System.out.println(event.toString());
```

```
   }
}
```

کد فایل اصلی برنامه یا MainApp.java هم به این شکل می باشد :

```
package com.tutorialspoint;


import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;


public class MainApp {
   public static void main(String[] args) {
      ConfigurableApplicationContext context =
      new ClassPathXmlApplicationContext("Beans.xml");


      CustomEventPublisher cvp =
      (CustomEventPublisher) context.getBean("customEventPublisher");
      cvp.publish();
      cvp.publish();
   }
}
```

در نهایت نیز کد فایل پیکربندی اطلاعات Bean.xml به صورت زیر است :

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


 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">


    id="customEventHandler"

      class="com.tutorialspoint.CustomEventHandler"/>


    id="customEventPublisher"

      class="com.tutorialspoint.CustomEventPublisher"/>
```

در صورتی که محتوی تمامی فایل ها را به درستی ایجاد کرده و شکلی وجود نداشته باشد ، پس از اجرای برنامه ، خروجی زیر حاصل می شود :

```
y Custom Event

y Custom Event
```

آموزش Aop در چهارچوب کاری Spring

یکی از Component های اصلی و کلیدی چهارچوب کاری Spring ، کامپوننت AOP یا Programming Aspect oriented است .

Aspect oriented Programming باعث تفکیک شدن منطق و کدهای برنامه (Logic) به بخش های مجزایی به نام So-Called Concern می شود .

توابعی که بخش های مختلف یک برنامه را به هم وصل می کنند را Cross-Cutting Concerns می گویند . Cross-Cutting Concerns به صورت مفهومی ، از business Logic برنامه جدا هستند . مثال های مختلفی را برای این نوع برنامه های توان ذکر کرد مثل عملیات Logging ، امنیت یا security و ... .

واحد اصلی هم پیمانه کردن کدها در OOP کلاس است ، اما در AOP از مفهوم aspect یه جای کلاس استفاده می شود . Dependency Injection به شما کمک می کند تا اشیای برنامه خود را از هم جدا کنید . از طرف دیگر AOP این امکان را در اختیارتان قرار داده تا Cross-cutting Concern های برنامه را از Object

هایی که آنها را تحت تاثیر قرار می دهند ، تفکیک کنید . AOP همانند trigger ها در زبان های برنامه نویسی مثل Perl یا Net. هستند .

Spring AOP ، رهگیری های کدی در اختیار شما قرار می دهد که به وسیله آنها ، می توانید فرآیند اجرای application را قطع کنید برای مثال ، وقتی که یک متد اجرا می شود شما می توانید کاربردها و کلیدهایی را برای مراحل قبل و بعد از اجرای متد ، به آن اضافه کنید .

تکنولوژی های AOP

قبل از شروع کار با AOP ، اجزه بدهید تا با مفهوم ها و اجرای اصلی AOP آشنا شویم . مواردی که در جدول زیر معرفی شده اند ، مختص Spring نبوده و بیشتربه AOP مربوط می شوند :

- Aspect : Aspect ماژولی است که دارای تعدادی API بوده و زمینه های لازم جهت Cross-Cutting را فراهم می کند . برای مثال ، یک ماژول Logging ، به عنوان AOP Aspect for نامیده خواهد شد .
- Join Point : Join Point نقطه ای در برنامه شماست که از طریق آن می توانید AOP Aspect را به برنامه متصل کنید . همچنین شما می توانید بگویید ، Join Point مکان واقعی است که یک action را از چهارچوب کاری Spring AOP دریافت می کند .
- Advise : این مفهوم ، action ای است که برنامه قبل یا از اجرای متد انجام می دهد . در واقع قطعه کدی است که توسط چهارچوب کاری Spring AOP در خلال اجرای برنامه ، فراخوانی می شود .
- Pointcut : Pointcut مجموعه ای از یک یا چندین Join Point است که advise بایستی در آن اجرا شود . شما می توانید Pointcut ها را به وسیله عبارت های دستوری (expressions) و یا الگوها (pathern) ، همانطور که در مثال های عملی درس خواهید داد ، تعیین نمایید .

- Introduction : یک Introduction به ما امکان می دهد تا متدها و خواص جدیدی را به کلاس های موجود اضافه کنیم .
- Target Object : شی است که توسط یک یا چند aspect مورد توجه قرار گرفته و همیشه یک Proxied خواهد بود . همچنین به آن advised object هم می گویند .
- Wearing : Wearing پروسه اتصال aspect ها به سایر اجزا و اشیای برنامه برای ایجاد advise object می باشد . این پروسه می تواند در زمان کامپایل ، لود و یا اجرا انجام شود .

مدل های مختلف Advice

Spring aspects می توانند با 5 نوع Adviceای که در جدول زیر معرفی شده اند ، کار کنند :

- before : این advice قبل از اجرای متد مورد نظر اجرا می شود .
- after : این advice بعد از اجرای متد مورد نظر و بدون توجه به خروجی آن اجرا می شود
- after-returning :این advice پس از اجرای متد ، به شرطی که کاملا به درستی اجرا شده باشد رخ می دهد .
- after-throwing : این advice نیز پس از اجرای متد ، به شرطی که متد درست اجرا شده و یا اشکالی در آن رخ داده باشد انجام می شود .
- around : این advice نیز قبل و بعد از فراخوانی متد مورد نظر ، اجرا می شود .

شیوه های مختلف اجرای Aspect ها :

Spring از دو رویکرد @AspectJ annotation style و schema-based برای اجرای aspect ها استفاده می کند . این دو رویکرد را در جدول زیر معرفی کرده ایم ، برای دریافت اطلاعات بیشتر برروی نام هریک کلیک نمایید :

XML Schema based : در این مدل Aspect ها با استفاده از کلاس های معمولی برنامه و مبتنی بر فایل پیکربندی اطلاعات XML اجرا می شوند .

AspectedJ based : در این مدل ، Aspect ها با استفاده از یک شیوه به نام AspectJ@ و با کلاس های معمولی جاوا که تحت Java 5 عمل annotation برروی آنها انجام شده است ، اجرا می شوند .

آشنایی با چهارچوب کاری JDBC در Spring

هنگامی که در محیط های برنامه نویسی جاوا ، برای کار با پایگاه های داده از زبان قدیمی JDBC استفاده می کنیم ، کار برنامه نویسی کمی سخت می شود . در این شرایط نوشتن کدهای غیر ضروری برای مدیریت استثنا های برنامه ، باز و بسته کردن ارتباط با پایگاه داده و ... کمی دشوار است .

از طرف دیگر ، چهارچوب کاری JDBC در Spring ، با رویکردی جدید انجام کلیه امور مرتبط با پایگاه داده از کارهای معمولی تا سطح بالا را برعهده گرفته است . برای مثال Spring JDBC ، تمهیدات لازم جهت باز کردن ارتباط با پایگاه داده ، آماده نمودن دستورات SQL جهت اجرا ، پردازش استثناها و مدیریت تراکنش ها و در نهایت بستن ارتباط را فراهم کرده است .

بنابراین تنها کاری که شما لازم است انجام دهید تعیین پارامترهای ضروری برای اتصال به پایگاه داده و نوشتن دستورات SQL برای دریافت اطلاعات مورد نظر از پایگاه داده است .

Spring JDBC چندین روش مختلف جهت اتصال به پایگاه داده و همچنین کلاس های لازم جهت ارتباط با آن را فراهم نموده است . ما در این درس از معمول ترین و رایج ترین راه برای ارتباط با پایگاه داده ، یعنی استفاده از کلاس Jdbc Template در چهارچوب کاری Spring استفاده می کنیم . کلاس JdbcTemplate ، کلاس اصلی و مرکزی Spring جهت ارتباط و کار با پایگاه داده است .

آموزش کلاس JdbcTemplate Class :

کلاس JdbcTemplate ، دستورات SQL ، Qnery های آن را اجرا کرده ، Statment های برنامه را به روز نموده و Stored Procedure ها را فراخوانی و مدیریت می کند . همچنین این کلاس ، محتویات ResultSet ها را خوانده و Parameter های مورد نظر را از آنها استخراج می کند .

از طرف دیگر کلاس Jdbc Template خطاهای JDBC را گرفته و آنها را به generic تبدیل می کند . سپس اطلاعات بیشتری به خطاها اضافه کرده و آنها را در سلسله مراتب خطاهای برنامه در پکیج org.spring framework.doo قرار می دهد .

هر نسخه ای از کلاس JdbcTemplate ایجد شود به صورت threadsafe (انتقال امن داده) تنظیم می شود . بنابراین شما می توانید یک نسخه واحد از کلاس JdbcTemplate ایجاد کرده و آن را به چندین DAO مورد نظر خود تزریق کنید .

یک تمرین رایج در هنگام استفاده از کلاس JdbcTemplate ، تنظیم یک DataSource در فایل پیکربندی اطلاعات برنامه یا Spring Configuration file است . پس بایستی این Data Source مشترک را به وسیله dependency-inject شی Bean به کلاس های DAO مورد نظر خود تزریق نمایید . در نهایت کلاس JdbvTemplate در بخش Setter شی Data Source قرار می گیرد .

تنظیم منبع داده (Data Source)

در بخش مثال عملی ، ابتدا یک جدول به نام Student را در پایگاه داده Test ایجاد می کنیم . در اینجا فرض بر این است که شما در حال کار با پایگاه داده MySQL هستید ، اگر با پایگاه داده دیگری می خواهیم کار کنید بایستی فایل DDL برنامه و همچنین SQL Query های خود را تغییر دهید .

```
CREATE TABLE Student(
   ID   INT NOT NULL AUTO_INCREMENT,
   NAME VARCHAR(20) NOT NULL,
   AGE  INT NOT NULL,
   PRIMARY KEY (ID)
);
```

در مرحله بعد بایستی یک Data Source را برای کلاس Jdbc Template فراهم کنیم تا بتواند شرایط خود را جهت اتصال به پایگاه داده تنظیم نماید . شما می توانید تنظیمات لازم جهت Data Source را همانطور که در فایل XML زیر نشان داده شده است ، تعیین نمایید

```
id="dataSource"

class="org.springframework.jdbc.datasource.DriverManagerDataSource">

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

    name="url" value="jdbc:mysql://localhost:3306/TEST"/>

    name="username" value="root"/>

    name="password" value="password"/>
```

آموزش شی Data Access Object یا DAO

DAO مخفف عبارت Data Access Object بوده که جهت تعامل با پایگاه داده به کار می رود . DAO ها برای این وجود دارند تا بتوانند اطلاعات را از پایگاه داده خوانده و اطلاعات جدید را به آنها وارد نمایند . DAO بایستی قابلیت های خود را در قالب یک رابط کاربری ارائه داده تا سایر اجزای Application بتوانند از طریق آنها به اطلاعات دسترسی داشته باشند .

قابلیت پشتیبانی از DAO در Spring ، امکان این را فراهم کرده تا به راحتی با تکنولوژی مختلف پایگاه داده مثل JDBC ، Hibrnate و ... در صورت پایدار و مطمئن کار نماییم .

اجرای دستورات SQL در Spring

در این بخش به آموزش انجام چهار عمل خواندن ، نوشتن ، ایجاد و حذف اطلاعات که به اختصار (Delete و Update و Read و Creation) CRUD نامیده می شود ، در محیط Spring می پردازیم . CRUD در Spring به وسیله دستورات SQL و شی Jdbc Template انجام می شود .

کد مثال جهت جستجوی یک مقدار عددی در پایگاه داده :

```
String SQL = "select count(*) from Student";
int rowCount = jdbcTemplateObject.queryForInt( SQL );
```

کد مثال جهت جستجوی یک مقدار Long در پایگاه داده :

```
String SQL = "select count(*) from Student";
long rowCount = jdbcTemplateObject.queryForLong( SQL );
```

کد مثال مربوط به یک Query با bind Variable :

```
+String SQL = "select age from Student where id = ?";
int age = jdbcTemplateObject.queryForInt(SQL, new Object[]{10});
```

کد مثال جهت جستجوی یک مقدار String در پایگاه داده :

```
String SQL = "select name from Student where id = ?";
String name = jdbcTemplateObject.queryForObject(SQL, new Object[]{10}, String.class);
```

کد مثال جهت جستجو و بازگرداندن مقادیر یک شی object :

```
String SQL = "select * from Student where id = ?";

Student student = jdbcTemplateObject.queryForObject(SQL,

                  new Object[]{10}, new StudentMapper());


public class StudentMapper implements RowMapper<Student> {

   public Student mapRow(ResultSet rs, int rowNum) throws SQLException {

      Student student = new Student();

      student.setID(rs.getInt("id"));

      student.setName(rs.getString("name"));

      student.setAge(rs.getInt("age"));

      return student;

   }

}
```

کد مثال جهت یافتن و بازگرداندن مقدار چندین شی :

```
String SQL = "select * from Student";

List<Student> students = jdbcTemplateObject.query(SQL,

                         new StudentMapper());


public class StudentMapper implements RowMapper<Student> {

   public Student mapRow(ResultSet rs, int rowNum) throws SQLException {

      Student student = new Student();

      student.setID(rs.getInt("id"));

      student.setName(rs.getString("name"));

      student.setAge(rs.getInt("age"));

      return student;

   }

}
```

کد مثال جهت وارد کردن یک سطر جدید در پایگاه داده :

```
String SQL = "insert into Student (name, age) values (?, ?)";
jdbcTemplateObject.update( SQL, new Object[]{"Zara", 11} );
```

کد مثال جهت به روز رسانی یک سطر در پایگاه داده :

```
String SQL = "update Student set name = ? where id = ?";
jdbcTemplateObject.update( SQL, new Object[]{"Zara", 10} );
```

کد مثال جهت حذف یک سطر row از پایگاه داده :

```
String SQL = "delete Student where id = ?";
jdbcTemplateObject.update( SQL, new Object[]{20} );
```

اجرای دستورات DDL

شما می توانید از متد (...) execute شی Jdbc Template جهت اجرای دستورات SQL یا DDL استفاده کنید . کد مثال زیر ، نحوه ایجاد یک جدول جدید را با استفاده از دستور CREATE نشان میدهد :

```
String SQL = "CREATE TABLE Student( " +
   "ID   INT NOT NULL AUTO_INCREMENT, " +
   "NAME VARCHAR(20) NOT NULL, " +
   "AGE  INT NOT NULL, " +
   "PRIMARY KEY (ID));"


jdbcTemplateObject.execute( SQL );
```

مثال های عملی کار با JDBC در چهارچوب String

در پایان مطالب آموزشی ارایه شده ، 2 مثال مهم جهت درک بهتر کاربرد JDBC در Spring را برایتان تشریح می کنیم :

مثال 1 : Spring JDBC Example :

در این مثال به آموزش نحوه یک بذنامه Spring مبتنی بر چهارچوب JDBC خواهیم پرداخت .

مثال 2 : SQL Stored Procedure in Spring :

در مثال دوم نیز به آموزش نحوه فراخوانی و اجرای توابع آماده SQL یا stored Procedure ها در چهارچوب JDBC در Spring خواهیم پرداخت .

آموزش مدیریت تراکنش ها در Spring

یک تراکنش در پایگاه داده (database transaction) مجموعه ای از عملیات های داده ای متوالی ، است که بایستی به عنوان یک واحد کاری رفتار کند.

این مجموعه عملیات ها یا بایستی به صورت کامل با هم انجام شوند و یا اینکه کلا اثری برسیستم نداشته

باشند . به عبارت دیگر تراکنش یا باید به طور کامل و موفقیت آمیز انجام شده و یا در کل لغو گردد.

مدیریت تراکنش ها (transaction management) بخش مهمی از سیستم مدیریت یکپارچه پایگاه های داده (RDBMS) بوده که درستی و ثبات ثبت و انتقال داده را کنترل می کند . مفهوم اساسی تراکنش های پایگاه داده را در چهار اصل کلیدی زیر می توان خلاصه است که به اختصار ACID نامیده می شوند:

- Atomicity :یک تراکنش بایستی به عنوان یک موجودیت واحد از عملیات های متوالی و مرتبط در نظر گرفته شده و رفتار کند . کلید مراحل یک تراکنش بایستی یا به صورت کامل و موفقیت آمیز انجام شوند یا به کلی لغو گردند.
- Consistency :این مفهوم به معنای حفظ یکپارچگی اطلاعات در پایگاه داده و اطمینان از ارتباط صحیح فیلدها و کلیدها در جدول ها است . از پارامترهای مهم این مفهوم به کلید خارجی (foreign key) یا کلید اصلی (primary key) می توان اشاره کرد.
- Isolstion :در پروسه پردازش تراکنش های پایگاه داده ، ممکن است در زمان واحد تعدادی از تراکنش ها دارای Data Set مشترک باشند . بنابراین بایستی هر تراکنش از سایر تراکنش ها ایزوله شده تا از تداخل داده ها جلوگیری شود.
- Durability :به محض اینکه یک تراکنش در پایگاه داده تکمیل شد بایستی نتیجه آن به صورت دائمی ثبت شده و قابل پاک کردن نباشد . زیرا عدم انجام این کار ، خطر از دست رفتن اطلاعات را به وجود می آورد.

یک سیستم واقعی و اصولی مدیریت یکپارچه پایگاه داده (RDBNS) بایستی اجرای هر چهار عمل فوق را ضمانت کند . ساده ترین رویه برای اجرا و تکمیل یک تراکنش به وسیله SQL در پایگاه داده به صورت زیر است:

- آغاز اجرای تراکنش به وسیله دستور. begin transaction
- انجام عملیات های مختلف حذف ، ویرایش (Update) ، وارد نمودن اطلاعات (insert و ...) به وسیله. SQL queries

- اگر کلید عملیات های مورد نظر با موفقیت انجام شد بایستی دستور Commit (ثبت نهایی) انجام شده یا کل عملیات کسل (rollback) شود.

چهارچوب کاری Spring یک لایه مجزا را برروی API های مرتبط با مدیریت تراکنش ها ، جهت کنترل بهتر برنامه فراهم می کند . سیستم پشتیبانی تراکنش های Spring با هدف فراهم نمودن یک جایگزین برای تراکنش های EJB ایجاد شده که در آن قابلیت های مدیریت تراکنش ها را به POJO ها اضافه کرده است Spring . هر دو رویه برنامه نویسی و یا اعلان را برای مدیریت تراکنش ها ، پشتیبانی می کند EJB . برای انجام تراکنش ها به یک سرور نیازمند است ، اما مدیریت تراکنش های Spring می تواند بدون کمک از یک سرور ، کارهای خود را انجام دهد.

بررسی تراکنش های Global در مقابل Local

تراکنش های محلی (Local Transactions) مختص منابع با معاملات تکی مثل JDBCمی باشد . از طرف دیگر تراکنش های سراسری (Global Transactions) می تواند معاملات چند بعدی در منابع ای مثل سیستم های گسترده و چند کاربره را پوشش دهد.

مدیریت تراکنش محلی یا Local Transaction Managment در سیستم های کامپیوتری متمرکز که در آن کامپوزیت ها و منابع در یک سایت واحد هستند ، مفید است و Local Managment تراکنش های مختص یک پایگاه داده که برروی یک ماشین است را مدیریت می کند . تراکنش های محلی در اجرا بسیار ساده تر هستند. از طرف دیگر ، مدیریت تراکنش های سراسری (Global Transactions Management) در سیستم های کامپیوتری گسترده که در آن کامپوزیت ها و منابع برروی سیستم های مختلف پخش شده اند ، لازم است . در چنین وضعیتی ، مدیریت تراکنش ها هم بایستی برروی کامپیوتر میزبان و هم بر روی سایر کامپیوترهای متصل به شبکه انجام شود.

یک تراکنش سراسری Global یا گسترده به صورت همزمان برروی چندین سیستم اجرا می شود . بنابراین اجرای این نوع تراکنش ها ، نیازمند هماهنگی بین سیستم مدیریت سراسری تراکنش ها و کلید سیستم های کلاینت موجود در سیستم می باشد.

کدنویسی Programmatic در مقابل Declarative

Springدو مدل مدیریت تراکنش ها را پشتیبانی می کند که عبارتند از:

- Programmatic transaction management : در این روش ، مدیریت تراکنش ها به کمک برنامه نویسی انجام می شود . این روش قابلیت انعطاف پذیری زیادی به ما می دهد ، ولی از طرف دیگر برای طراحی کمی مشکل و زمان بر است.
- Declarative transaction management : در این روش مدیریت تراکنش ها از لایه business برنامه جدا شده است . در این روش شما از annotations یا فایل های پیکربندی مبتنی بر XML برای مدیریت تراکنش ها بهره می گیرید.

Declarative transaction managementنسبت به Programmatic transaction management بهتر است ، زیرا انعطاف پذیری کمتری داشته و به شما اجازه می دهد تا تراکنش های خود را توسط کد برنامه کنترل کنید.

اما به عنوان یک نوع از (Crosscutting Concern تقسیم اجزای یک شی ، Declarative transaction managementرا می توان با استفاده از روش AOPبه بخش های مختلف تقسیم کرد.

آموزش Spring Transaction Abstraction

راهنمای لازم برای Spring Transaction Abstraction در رابط کاربری org.springframework.transaction.PlatformTransactionManagerکه کد آن در زیر نمایش داده شده تعیین شده است:

```
public interface PlatformTransactionManager {
   TransactionStatus getTransaction(TransactionDefinition definition);
   throws TransactionException;
   void commit(TransactionStatus status) throws TransactionException;
   void rollback(TransactionStatus status) throws TransactionException;
```

در جدول زیر به بررسی متدهای رابط کاربری فوق می پردازیم:

- متد TransactionStatus getTransaction(TransactionDefinition definition)

این متد تراکنش فعال جاری را بازگزدانده و یا یک تراکنش جدید را برحسب رفتار تعیین شده جهت برنامه ایجاد می کند.

- متد (Transaction Status status) Void Commit این متد ، تراکنش ارسال شده به آن را با توجه به وضعیت تراکنش (statuts) به برنامه می فرستد .

- متد (TransactionStatus status) Void rollback این متد ، عملیات انجام شده برروی تراکنش داده شده به آن را لغو (rollerback) میکند .

TransactionDefinitionرابط کاربری مرکزی قابلیت پشتیبانی transaction در Spring بوده و با کدی همانند زیر تعیین شود

```
public interface TransactionDefinition {
   int getPropagationBehavior();
   int getIsolationLevel();
   String getName();
   int getTimeout();
   boolean isReadOnly();
}
```

در جدول زیر به بررسی متدهای کد فوق و کاربرد آنها خواهیم پرداخت:

- متد int getpropagation Behavior این متد رفتار propagation را برمی گرداند . Springکلید propagation های یک تراکنش را در راستای EJB CMT تعیین می کند .

- متد()int getIsolationLevel این متد درجه ای که در آن بایستی تراکنش مورد نظر از کار سایر تراکنش ها مجزا و ایزوله شود را برمی گرداند .

- متد()string getName این متد نام تراکنش مورد نظر را برمیگرداند .

- متد()intget Timeout این متد تعداد ثانیه هایی که طی آن ، تراکنش بایستی کامل شود را برمی گرداند.

- متد ()Boolean isReadOnlyاین متد تعیین می کند آیا تراکنش جاری Read-only هست یا خیر.

رابط کاربری TransactionStatus برای کد تراکنش برنامه یک راه ساده فراهم کرده بتواند اجرای transactionهای برنامه و وضعیت جستجوی تراکنش ها را کنترل کند . به صورت کد زیر :در لیست زیر نیز به بررسی مقادیر مختلف ممکن جهت وضعیت ایزوله کردن (isolation level) هر تراکنش پرداخته ایم :

- مقدار TransactionDefinition.ISOLATION_DEFAULTاین خاصیت مقدار پیش فرض اول ایزوله شدن تراکنش است .
- مقدار TransactionDefinition.ISOLATION_READ_COMMITTEDاین مقدار تعیین می کند که خواندن تراکنش قبل از اجرای کامل (dirty read) امکانپذیر نیست . ولی خواندهای non-repeatable و Phontom read ممکن هستند.
- مقدار TransactionDefinition.ISOLATION_READ_UNCOMMITTEDاین مقدار تعیین میکند هر 3 حالت خواندن تراکنش در حالت dirty read ، non-reapeatableو Phontom read امکانپذیر است.
- مقدار TransactionDefinition.ISOLATION_REPEATABLE_READاین مقدار نیز تعیین می کند که فقط حالت خواندن Phntom read ممکن بوده و حالت های dirty readو non-reapeatable غیر ممکن است.
- مقدار TransactionDefinition.ISOLATION_SERIALIZABLEدر این حالت کلیه حالت های مختلف خواندن تراکنش ممنوع است.

جدول زیر نیز به بررسی و تشریح مقادیر مختلف ممکن برای Propagation پرداخته است:

- حالت TransactionDefinition.PROPAGATION_MANDATORY در این وضعیت تراکنش جاری حمایت می شود . اگر تراکنش جاری وجود نداشته باشد ، خطا رخ می دهد.
- حالت TransactionDefinition.PROPAGATION_NESTED در این حالت اگر تراکنش جاری وجود داشته باشد به همراه تراکنش درونی آن اجرا می شود.

- حالت TransactionDefinition.PROPAGATION_NEVER در این حالت تراکنش جاری پشتیبانی نمی شود . اگر تراکنش جاری در سیستم وجود داشته باشد ، خطا رخ می دهد.
- حالت TransactionDefinition.PROPAGATION_NOT_SUPPORTED در این حالت تراکنش جاری پشتیبانی نشده و برنامه بیشتر اطلاعات ثابت-non) tranactionally) را پردازش می کند.
- حالت TransactionDefinition.PROPAGATION_REQUIRED در این حالت تراکنش جاری پشتیبانی می شود . اگر تراکنشی وجود نداشته باشد ، سیستم یک تراکنش جدید ایجاد می کند.
- حالت TransactionDefinition.PROPAGATION_REQUIRES_NEW در این حالت برنامه یک تراکنش جدید ایجاد کرده و درصورتی که از قبل تراکنشی وجود داشته باشد ، آن را تعلیق می کند.
- حالت TransactionDefinition.PROPAGATION_SUPPORTS در این حالت تراکنش جاری اجرا می شود ، ولی اگر تراکنشی وجود نداشته باشد ، اطلاعات ثابت برنامه پردازش می شود.
- حالت TransactionDefinition.TIMEOUT_DEFAULT در این حالت سیستم از مدت زمان انقضای پیش فرض (default timeout) استفاده می کند . اگر هم timeoutپشتیبانی نشود ، کاری صورت نمی گیرد.

رابط کاربری TransactionStatus برای کد تراکنش برنامه یک راه ساده فراهم کرده بتواند اجرای transactionهای برنامه و وضعیت جستجوی تراکنش ها را کنترل کند . به صورت کد زیر :

```
public interface TransactionStatus extends SavepointManager {
   boolean  isNewTransaction();
   boolean hasSavepoint();
   void setRollbackOnly();
   boolean isRollbackOnly();
   boolean isCompleted();
```

```
}
```

در جدول زیر به بررسی متدهای مرتبط با TransactionStatus پرداخته ایم:

- متد boolean hasSavepoint() این متد یک مقدار Boolean را برمی گرداند که آیا تراکنش جاری دارای یک savepoint هست یا خیر .
- متد boolean isCompleted این متد نیز یک مقدار Boolean را برمی گرداند که آیا تراکنش جاری کامل انجام شده یا خیر . یا اینکه قبلا به صورت کامل ثبت شده یا rollback شده است .
- متد boolean isNewTransaction این متد یک مقدار Boolean برمی گرداند که آیا تراکنش جاری یک تراکنش جدید است یا خیر .
- متد boolean isRollbackOnly این متد نیز یک مقدار Boolean را برمی گرداند که آیا تراکنش جاری به عنوان یک تراکنش (rollback-only تراکنشی که بایستی rollback شود.) تعیین شده یا خیر .
- متد Void setRollbackOnly این متد تراکنش جاری را به عنوان یک تراکنش-rollback only تعیین می کند .

آموزش فریم ورک MVC در Spring

فریم ورک MVC چیست ؟

فریم ورک Spring MVC ، معماری model-view-Controller و کامپوزیت های آماده آن را برای ایجاد نرم افزارهای تحت وب انعطاف پذیر و دارای بخش های مجزا فراهم میکند.

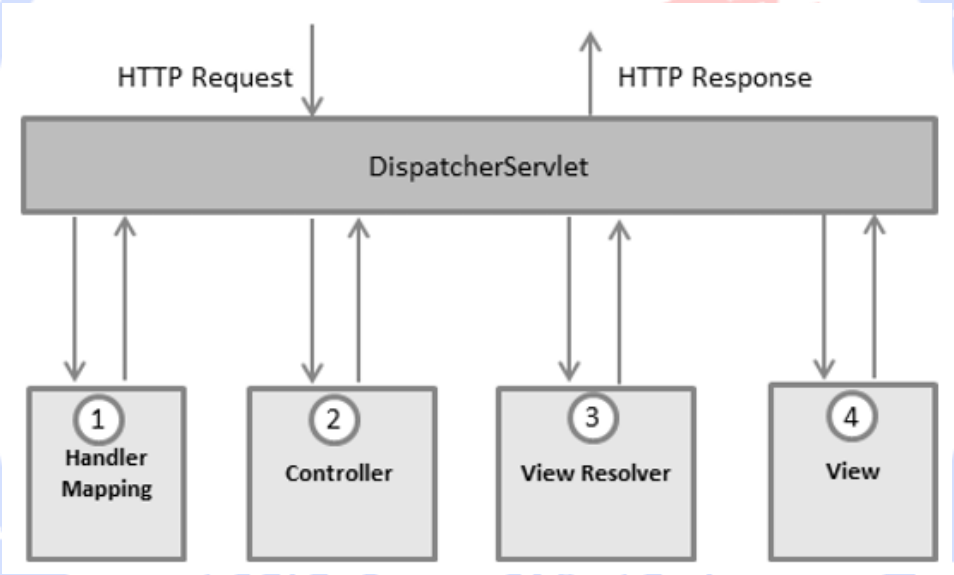
الگوی MVC باعث جدا شدن و مستقل عمل کردن اجزای اصلی یک UI logic (application و business و input logic) از یکدیگر شده ، در حالی که ارتباط بین آنها به وسیله روش های مختلفی فراهم می شود.

- بخش Model اطلاعات برنامه را کپسوله سازی کرده و از تعدادی POJO تشکیل می شود .

- بخش View وظیفه رندکردن مدل داده ای برنامه و تولید خروجی HTML آن را جهت نمایش در مرورگر
- بخش Controller هم وظیفه پردازش درخواست های کاربر را داشته و بایستی model مناسب برنامه را تولید کرده و جهت رندر به View ارسال کند.

آموزش DispatcherSerlet

فریم ورک (model-view-controller) Spring MVC حول یک کامپوزیت به نام DispatcheServlet طراحی شده که کلیه درخواست ها و پاسخ های HTTP را مدیریت می کند . فرآیند پردازش درخواست ها در Spring MVCتوسط DispatcherServtel در دیاگرام زیر نمایش داده شده است:



در لیست زیر ، ترتیب وقوع رویدادهای مرتبت با پردازش یک درخواست HTTP توسط DispatcherServlet بیان شده است:

- پس از دریافت یک درخواست HTTP ، کامپوزیت DispatcherServlet از کامپوزیت HandlerMappingبرای فراخوانی Controller مناسب ، کمک می گیرد.
- Controllerدرخواست را دریافت کرده و سرویس مناسب را بسته به متدهای GETیا POSTجهت اجرای درخواست فراخوانی می کند . سرویس فراخوانی شده model data را

بر مبنای logvc business مناسب ، تنظیم می کند و پس نام View را به DispatcherServletپلاش می دهد.

- پس از اینکه view به طور کامل آماده شد ، DispatcherServletمدل دیتا (data model)را به view بازمی گرداند تا برای اجرا به مرورگر ارسال شود .
- تمامی کامپوزیت های اشاره شده در قسمت بالا برای مثال Handler Mapping ، Controllerو ViewResolver بخشی از WebApplicationContext هستند که خود یک کامپوزیت توسعه یافته از ApplicationContext می باشد .این کامپوزیت دارای یکسری قابلیت های لازم و اضافه برای نرم افزارهای تحت web است.

تنظیمات لازم: (Required Configuration)

شما بایستی آدرس و مشخصات درخواست هایی که می خواهید Dispatcher Servlet مدیریت را به وسیله قابلیت Mapping URL در فایل web.xml مشخص می کنید.

کد زیر نحوه تعریف مشخصات و آدرس دهی برای DispatcherServlet را در یک مثال عملی نشان می دهد :

```
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemalocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

    <display-name>Spring MVC Application</display-name>

   <servlet>
      <servlet-name>HelloWeb</servlet-name>
      <servlet-class>
         org.springframework.web.servlet.DispatcherServlet
      </servlet-class>
      <load-on-startup>1</load-on-startup>
   </servlet>

   <servlet-mapping>
      <servlet-name>HelloWeb</servlet-name>
      <url-pattern>*.jsp</url-pattern>
   </servlet-mapping>

</web-app>
```

فایل web.xml در پوشه WebContent/WEB-INF پروژه وب نگهداری می شود . برای مقدار دهی HelloWeb Dispatcher Servletدر کد فوق ، فریم ورک تلاش می کند تا محتوی برنامه را از فایلی به نام [servlet-name]-servlet.xml که در پوشه WebContent/WEB-INF قرار دارد ، بخواند . در این مثال نام مورد نظر ما در پوشه HelloWeb-servlet.xml است.

در مرحله بعد ، تگ آدرس یا URL هایی که بایستی توسط DispatcherServlet مدیریت شوند را تعیین می کند . در این مثال کلید درخواست های HTTP ای که با jsp، پایان می یابند ، توسط HelloWeb DispatcherServletمدیریت می شوند.

اگر نمی خواهید برنامه شما به صورت پیش فرض به دنبال فایل با نام [servlet-name]-servlet.xml در پوشه WebContent/WEB-INFبروید ، بایستی نام و محل قرارگیری این فایل ها را با اضافه کردن یک ContentLoaderListenerبه فایل web.xml به صورت زیر تعیین کنید :

```
<web-app...>
<!-------- DispatcherServlet definition goes here----->
....
<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>


 </web-app...>
```

در کد زیر هم می توانیم تنظیمات لازم (required configuration) برای فایل HelloWeb-servlet.xml را که در پوشه WebContent/WEB-INFبرنامه قرار دارد را مشاهده کنیم :

```
 <beans xmlns="http://www.springframework.org/schema/beans"
xmlns:context="http://www.springframework.org/schema/context"
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
   http://www.springframework.org/schema/context
   http://www.springframework.org/schema/context/spring-context-3.0.xsd">
```

```
   <context:component-scan base-package="com.tutorialspoint">

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

</context:component-scan></beans>
```

نکات مهم درباره فایل HelloWeb-servlet.xml را در ادامه بررسی کرده ایم:

- فایل [servlet-name]-servlet.xml برای تعیین bean های برنامه به کار میرود . این فایل کلید تنظیمات برای هر bean ای که در محیط عمومی برنامه (Global Scope) تعیین شده است را جایگزین می کند.
- InternalResourceViewResolver دارای قواعد لازم برای ترجمه کردن نام viewهای برنامه است . همنطور که در قواعد مثال فوق تعیین شده ، یک view logical به نام helloبرای اجرا به یک view موجود در پوشه /WEB-INF/jsp/hello.jsp واگذار شده است.

در مراحل بعدی این درس ، به آموزش نحوه نوشتن کامپوزیت های واقعی مورد نظرتان مثل Controller خواهیم پرداخت.

تعیین یک: Controller

همانطور که گفتیم Dispatch Servlet هر request را به controller مرتبط خود واگذار می کند تا کدهای آن اجرا شود Controller@ annotation . تعیین کننده کلاس ای است که قواعد مربوط به یک controller را اعلام می کند.

همچنین RequestMapping@ جهت اتصال URL به یک کلاس یا یک متد خاص به کار می رود . مثل کد زیر :

```
@Controller
@RequestMapping("/hello")
public class HelloController{
```

```
   @RequestMapping(method = RequestMethod.GET)
   public String printHello(ModelMap model) {
      model.addAttribute("message", "Hello Spring MVC Framework!");
      return "hello";
   }

}
```

در کد فوق ، @Controller annotation ، کلاس را به عنوان یک Spring MVC Controller تعیین می کند .

همچنین اولین استفاده از @Request Mapping کلیه متدهای مدیریت برنامه در این کنترل متعلق به

مسیر /hello هستند annotation . بعدی کد یعنی @RequestMapping(method = RequestMethodGET) تعیین کننده متد Print Hello () به عنوان متد سرزیس دهنده پیش فرض

کنترلی برای درخواست های HTTP GET می باشد . شما می توانید متد دیگری را نیز برای مدیریت درخواست Postبرنامه در همین URL تعیین کنید.

همچنین شما می توانید کد کنترلی فوق را به صورت دیگری نیز بنویسید . در این روش جدید ، می توانید خواص جدیدی را به @RequestMapping اضافه کنید:

```
@Controller
public class HelloController{

   @RequestMapping(value = "/hello", method = RequestMethod.GET)
   public String printHello(ModelMap model) {
      model.addAttribute("message", "Hello Spring MVC Framework!");
      return "hello";
   }
     }
```

خاصیت value تعیین کننده URL ای است که مدیریت کننده متد (handler method) به آن متصل است .

همچنین خاصیت method تعیین کننده متد سرویس جهت مدیریت درخواست های HTTP GET می باشد .

نکات مهم زیر را نیز بایستی در مورد کد کنترلی فوق یادآوری کنیم:

- شما business logic مورد نظر خود را در یک Service method تعیین می کنید .
  همچنین می توانید متدهای دیگر را درصورت لزوم ، درون این متد فراخوانی کنید.
- بنابر business logic ای که تعیین کرده اید ، یک model را در این متد ایجاد می کنید .
  شما می توانید خواص مختلف model ایجاد شده را تعیین کنید که توسط View برای ایجاد

خروجی نهایی قابل دسترس خواهند بود و مثال فوق یک model logicalخاصیتی به نام " message "است ایجاد می کند.

- متدی که به عنوان متد سرویس دهنده (service method) تعیین شده ، می تواند یک مقدار String را برگرداند . این مقدار String حاوی نام view ای است که جهت رند کردن modelبه کار می رود . مثال فوق مقدار " hello " را به عنوان نام logical view برمی گرداند.

ایجاد: JSP Views

Spring MVCاز Viewهای مختلف جهت تولید خروجی هایی با تکنولوژی های متفاوت پشتیبانی می کند . این تکنولوژی ها شامل JSP ، HTML، PDF، EXCEL، XML، JSON، Atomو Rss...هستند . اما به صورت معمول ما از تمپلیت JSP که به وسیله JSTL نوشته شده است ، استفاده می کنیم . به وسیله کد زیر ، یک Viewساده به نام hello را در مسیر /WEB-INF/hello/hello.jsp ایجاد می کنیم :

```
< html >
   < head >
   <title>Hello Spring MVC</title>
   < /head >
   < body >
   <h2>${message}</h2>
   < /body >
< /html >
```

در کد فوق ${ message } خاصیتی است که آن را درون Controller تنظیم کرده ایم . شما می توانید هرچند خواص که نیاز دارید را تعیین نموده تا در View نمایش داده شود.

آموزش کار با Log4J Logging در Spring

استفاده از Log4J در پروژه ها و نرم افزارهای Spring بسیار ساده است . همانطور که می دانید Log4J یک سیستم log کردن اطلاعات در Java است.

مثالی که در این درس ارائه خواهیم داد ، مراحل ساده استفاده و ترکیب Log4J در پروژه های Spring را نشان می دهد.

در اینجا فرض براین است که Log4J برروی سیستم شما نصب است . در غیر اینصورت به آدرس

http://logging.apache.orgرفته و فایل های نصب Log4J را دانلود نمایید . سپس آنها را در یک پوشه extractکرده و فایل اصلی log4j-x.y.2.jarرا نصب نمایید.

در مرحله بعدی به سراغ نرم افزار Eclipseرفته و با اجرای مراحل زیر ، یک برنامه تحت وب با یک فرم دینامیک را با استفاده از چهارچوب کاری Spring ایجاد نمایید.

1. یک پروژه جدید به نام Spring Example را ایجاد کرده و پکیج com.tahlildadeh را در پوشه src پروژه اضافه نمایید.
2. با استفاده از ابزار Add External JARs و همانطور که در درس ایجاد یک پروژه ساده با Spring تشریح کردیم ، کتابخانه های لازم جهت اجرای برنامه را به پروژه اضافه کنید.
3. درون پکیج com.tahlildadeh کلاس های HelloWorld و MainAPP را ایجاد نمایید.
4. در پوشه src پروژه ، فایل پیکربندی اطلاعات Beans.xmlرا ایجاد کنید.
5. فایل تنظیمات log4j Configuration را با نام log4j.properties در پوشه src ایجاد کنید.
6. مرحله آخر تولید محتوی لازم برای کلید فایل های java برنامه و فایل پیکربندی اطلاعات Bean.xml است که در ادامه به تشریح کدهای آنها خواهیم پرداخت.

کد زیر محتویات فایل HelloWorld.java برنامه را نشان می دهد :

```
package com.tutorialspoint;

public class HelloWorld {
   private String message;

   public void setMessage(String message){
      this.message  = message;
   }

   public void getMessage(){
      System.out.println("Your Message : " + message);
   }
}
```

کد زیر نیز مربوط به فایل اصلی برنامه MainApp.java است :

```
package com.tutorialspoint;
```

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.apache.log4j.Logger;

public class MainApp {

   static Logger log = Logger.getLogger(MainApp.class.getName());

   public static void main(String[] args) {
      ApplicationContext context =
             new ClassPathXmlApplicationContext("Beans.xml");

      log.info("Going to create HelloWord Obj");

      HelloWorld obj = (HelloWorld) context.getBean("helloWorld");

      obj.getMessage();

      log.info("Exiting the program");
   }
}
```

شما می توانید پیام های debug و error را همانطور که پیام info تولید شد ، ایجاد نمایید . در مرحله بعد کد فایل پیکربندی اطلاعات Beans.xml قرار داده شده است:

```
<!--?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="helloWorld" class="com.tutorialspoint.HelloWorld">
       <property name="message" value="Hello World!">
   </property></bean>

</beans>
```

در آخر هم کد فایل تنظیمات log برنامه با نام log4j.properties نمایش داده شده است . این فایل تنظیمات و قواعد لازم جهت برای تولید پیام log برنامه را مشخص می کند:

```
# Define the root logger with appender file
log4j.rootLogger = DEBUG, FILE

# Define the file appender
log4j.appender.FILE=org.apache.log4j.FileAppender
# Set the name of the file
log4j.appender.FILE.File=C:\\log.out

# Set the immediate flush to true (default)
```

```
log4j.appender.FILE.ImmediateFlush=true

# Set the threshold to debug mode
log4j.appender.FILE.Threshold=debug

# Set the append to false, overwrite
log4j.appender.FILE.Append=false

# Define the layout for file appender
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.conversionPattern=%m%n
```

پس از اینکه محتوی کلیه فایل های برنامه را تولید کردید ، می توانیم آن را اجرا نماییم . اگر همه چیز درست باشد ، برنامه پیام زیر را در خروجی چاپ خواهد کرد:

بلافاصله پس از اجرای برنامه ، می توانید مسیر C:11 را در کامپیوتر خود چک نمایید . خواهید دید فایلی به نام log.out ایجاد شده که درون آن پیام logبرنامه قابل مشاهده است . کدی همانند زیر:

```
Your Message : Hello World!
```

آموزش API (JCL) Jakarta Commons Logging

شما می توانید از API Jakarta Commons Logging برای تولید log های برنامه های Spring خود استفاده کنید JCL . را می توانید از آدرس http://jakarta.apacte.org/commons/loggingدانلود نمایید.

تنها فایلی که در پکیج دانلود شده از آدرس فوق به آن نیاز خواهیم داشت ، commons-logging-x.y.z.jar است . این فایل را بایستی در مسیر کلاس برنامه (classpath) همانند روشی که برای فایل log4j.x.y.z.jarبه کار بردیم ، قرار دهید.

برای استفاده از قابلیت logging در برنامه های Spring بایستی از یک شی org.oracle.commons.logging.logاستفاده کنید . پس با توجه به نیاز برنامه تان ، می توانید یکی از متدهای این شی را که در لیست زیر قرار دارند ف فراخوانی کنید:

- Fatal (object message)
- Error (object message)
- Worn (object message)

- Info (object message)
- Debug (object message)
- Trace (object message)

در کد زیر ، JCL APIرا به جای Log4J در فایل MainAPP.java قرار داده ایم . به تفاوت کدها دقت نمایید :

```
package com.tutorialspoint;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.apache.commons.logging. Log;
import org.apache.commons.logging. LogFactory;

public class MainApp {

   static Log log = LogFactory.getLog(MainApp.class.getName());

   public static void main(String[] args) {
      ApplicationContext context =
             new ClassPathXmlApplicationContext("Beans.xml");

      log.info("Going to create HelloWord Obj");

      HelloWorld obj = (HelloWorld) context.getBean("helloWorld");

      obj.getMessage();

      log.info("Exiting the program");
   }
}
```

شما بایستی مطمئن شوید که فایل commons-logging-x.y.z.jar در پروژه شما قرار داشته و پس به اجرا و کامپایل برنامه بپردازید.

اکنون که JCL APL را در برنامه خود جایگزین کرده ایم ، بقیه کدها و تنظیمات فایل پیکربندی اطلاعات برنامه را بدون تغییر رها کنید . با اجرای برنامه بایستی خروجی همانند روش Log4Jتولید شود.

TAHLILGAR DADEHA INSTITUTE
آموزشگاه تحلیلگر داده ها