부록 D. 스프링 3에서 4로 마이그레이션 하기

Spring Legacy Project 메뉴로 만든 Spring MVC 프로젝트의 스프링 프레임워크의 버전은 3.1.1 입니다. 이것을 4.3.4으로 변경하는 방법을 실습합니다.

AS-IS

- springframework-version 3.1.1
- java-version 1.6
- maven-compiler
  - source 1.6
  - target 1.6
- servlet-api 2.5
- jsp-api 2.1
- junit 3.9

TO-BE

- springframework-version 4.3.4
- java-version 1.8
- maven-compiler
  - source 1.8
  - target 1.8
- servlet-api 3.1
- jsp-api 2.2
- junit 4.12

pom.xml 수정하기

java-version을 1.6에서 1.8로 org.springframework-version 버전을 3.1.1에서 4.3.4로 변경합니다.

```
<properties>
    <java-version>1.8</java-version>
    <org.springframework-version>4.3.4.RELEASE</org.springframework-version>
    <org.aspectj-version>1.6.10</org.aspectj-version>
    <org.slf4j-version>1.6.6</org.slf4j-version>
</properties>
```

servlet-api 버전을 2.5에서 3.1.0로 변경합니다. artifactId 이름이 변경되었기 때문에 servlet-api에서 javax.servlet-api로 바꿔주어야 합니다.

```
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>3.1.0</version>
  <scope>provided</scope>
```

```
  </dependency>
```

서블릿 버전이 변경되면 web.xml 파일의 DTD도 버전에 맞게 수정되어야 합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="
  http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_3_1.xsd"
    version="3.1">

  <!-- 생략 -->

</web-app>
```

jsp-api 버전을 2.1에서 2.2로 변경합니다.

```
<dependency>
  <groupId>javax.servlet.jsp</groupId>
  <artifactId>jsp-api</artifactId>
  <version>2.2</version>
  <scope>provided</scope>
</dependency>
```

junit 버전을 3.9에서 4.12로 변경합니다.

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
  <scope>test</scope>
</dependency>
```

maven-compiler-plugin 설정에서 source와 target을 1.8로 변경합니다.

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>2.5.1</version>
  <configuration>
    <source>1.8</source>
    <target>1.8</target>
    <compilerArgument>-Xlint:all</compilerArgument>
    <showWarnings>true</showWarnings>
    <showDeprecation>true</showDeprecation>
  </configuration>
</plugin>
```

프로젝트 속성 수정하기

프로젝트의 `Properties`를 선택합니다. `Java Build Path`를 선택합니다. `Libraries`에서 `JavaSE-1.6`을 `JavaSE-1.8`로 변경합니다.

`Java Compiler` 설정에서 `Compiler compliance level`을 1.8로 변경합니다.

`Project Facets`에서 `Dynamic Web Module`을 3.1로, `Java`는 1.8로 변경합니다. 설정이 변경이 안되는 경우가 있습니다. 이 때는 이클립스를 종료한 후 프로젝트 폴더아래 `.settings` 폴더에 있는 `org.eclipse.wst.common.project.facet.core.xml` 파일을 메모장으로 열어서 직접 수정합니다. 이클립스를 실행하고 변경작업을 수행합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<faceted-project>
  <fixed facet="wst.jsdt.web"/>
  <installed facet="java" version="1.8"/>
  <installed facet="jst.web" version="3.1"/>
  <installed facet="wst.jsdt.web" version="1.0"/>
</faceted-project>
```

프로젝트를 선택한 다음 마우스 오른쪽 키를 누르면 뜨는 팝업 메뉴에서 `Maven >> Update Project`를 선택해서 메이븐이 변경된 설정을 처리하도록 조치합니다.

TEST

모든 설정이 적용되었습니다. 프로젝트를 재 기동하여 테스트 합니다. 프로젝트명에 빨간색이 없고, Problems 탭 뷰에 에러가 없다면 스프링 3에서 4로 업그레이드하는 마이그레이션 작업이 잘 적용된 것 입니다. 프로젝트 기동 테스트 시 로그에 에러가 없어야 합니다.

XML 설정을 자바컨피그 설정으로 변경

스프링 부트는 환경설정 작업에서 XML을 사용하지 않습니다. 스프링 부트처럼 앞에서 작업한 프로젝트의 XML 환경설정 부분을 모두 없애고 대신 애노테이션 및 자바컨피그로 설정을 변경해 봅니다.

Config.java

```
package com.example.demo.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.view.JstlView;
import org.springframework.web.servlet.view.UrlBasedViewResolver;

@Configuration
@ComponentScan("com.example.demo") // @Controller
@EnableWebMvc // @RequestMapping
public class Config {
    @Bean
    public UrlBasedViewResolver viewResolver() {
        UrlBasedViewResolver viewResolver = new UrlBasedViewResolver();
        viewResolver.setPrefix("/WEB-INF/views/");
```

```
        viewResolver.setSuffix(".jsp");
        viewResolver.setViewClass(JstlView.class);

        return viewResolver;
    }
}
```

context:component-scan

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

위 설정 대신 아래 애노테이션을 사용한다.

`@ComponentScan("com.example.demo")`

InternalResourceViewResolver

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

위 설정 대신 아래 메소드를 사용한다.

```
@Bean
public UrlBasedViewResolver viewResolver() {
  UrlBasedViewResolver viewResolver = new UrlBasedViewResolver();
  viewResolver.setPrefix("/WEB-INF/views/");
  viewResolver.setSuffix(".jsp");
  viewResolver.setViewClass(JstlView.class);

  return viewResolver;
}
```

mvc:annotation-driven

위 설정 대신 아래 애노테이션을 사용한다.

@EnableWebMvc

root-context.xml, servlet-context.xml

`webapp\WEB-INF\spring\root-context.xml` 파일을 삭제한다. 프로젝트 생성 시 root-context.xml 안에는 아무런 설정이 없다. 만약 추가한 설정이 있다면 해당 설정을 Config 클래스에 추가하자. `webapp\WEB-INF\spring\appServlet\servlet-context.xml` 파일을 삭제한다.

pom.xml

servlet-api 버전이 3.0 이상인지 확인하자.
web.xml이 없어도 에러나지 않도록 아래 플러그인을 추가하자.

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-war-plugin</artifactId>
  <version>2.6</version>
  <configuration>
    <failOnMissingWebXml>false</failOnMissingWebXml>
  </configuration>
</plugin>
```

WebInit.java

```
package com.example.demo.config;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.ServletRegistration.Dynamic;

import org.springframework.web.WebApplicationInitializer;
import org.springframework.web.context.support.AnnotationConfigWebApplicationContext;
import org.springframework.web.servlet.DispatcherServlet;

public class WebInit implements WebApplicationInitializer {

    @Override
    public void onStartup(ServletContext servletContext) throws ServletException {

        AnnotationConfigWebApplicationContext ctx = new
    AnnotationConfigWebApplicationContext();
        ctx.register(Config.class);
        ctx.setServletContext(servletContext);

        Dynamic servlet = servletContext.addServlet(
    "appServlet", new DispatcherServlet(ctx));
        servlet.addMapping("/");
        servlet.setLoadOnStartup(1);
//        servlet.setInitParameter("contextConfigLocation",
//          "/WEB-INF/appServlet-servlet.xml");
    }

}
```

```
<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

위 설정 대신 인터페이스를 구현한다.

```
WebInit implements WebApplicationInitializer
```

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/spring/root-context.xml</param-value>
</context-param>
```

위 설정 대신 다음 코드를 사용한다.

```
AnnotationConfigWebApplicationContext ctx = new
AnnotationConfigWebApplicationContext();
ctx.register(Config.class);
ctx.setServletContext(servletContext);
```

```
<servlet>
  <servlet-name>appServlet</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>appServlet</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
```

위 설정 대신 다음 코드를 사용한다.

```
Dynamic servlet = servletContext.addServlet("appServlet",
new DispatcherServlet(ctx));
servlet.addMapping("/");
servlet.setLoadOnStartup(1);
//servlet.setInitParameter("contextConfigLocation",
//  "/WEB-INF/appServlet-servlet.xml");
```

XML 방식의 설정에서는 root-context.xml에 의한 컨텍스트는 부모 컨테이너가 되고 servlet-context.xml에 의한 컨텍스트는 자식 컨테이너가 되어 상속관계를 같는다. 이와 유사하게 자바컨피그로 구성하려면 AnnotationConfigWebApplicationContext 클래스가 지원하는 setParent() 메소드를 사용하면 된다.

web.xml

`webapp\WEB-INF\web.xml` 파일을 삭제한다.

TEST

프로젝트를 재 기동하여 테스트 해 보자. 스프링 부트는 XML을 기본적으로 사용하지 않지만 개발자가 원한다면 `@ImportResource({"classpath*:context.xml"})` 애노테이션으로 사용할 수 있다.

`classpath vs classpath* 차이점`

- classpath : 현재 프로젝트의 resource만 선택 한다.
- classpath* : 현재 프로젝트에 관련(참조)된 모든 jar를 다 검색하여 리소스를 선택 한다.