Table of Contents

Bootstrap a Web Application with Spring 4

Return to Content

Contents

- Table of Contents
- 1. Overview
- 2. The Maven pom.xml
- 3. The Java based Web Configuration
- 4. Conclusion

If you're new here, you may want to get my "REST APIs with Spring" eBook [https://my.leadpages.net/leadbox/146382273f72a2%3A13a71ac76b46dc/5735865741475840/]. Thanks for visiting!

Table of Contents

- **1.** Overview
- **2.** The Maven pom.xml
- **Â Â Â 2.1.** Justification of the *cglib* dependency
- **Â Â Â 2.2.** The *cglib* dependency in Spring 3.2 and beyond
- **3.** The Java based web configuration
- **Â Â 3.1.** The web.xml
- **4.** Conclusion

1. Overview

The tutorial illustrates how to **Bootstrap a Web Application with Spring** and also discusses how to make the jump **from XML to Java** without having to completely migrate the entire XML configuration.

2. The Maven pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
```

```
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="
 http://maven.apache.org/POM/4.0.0
 http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <groupId>org</groupId>
 <artifactId>rest</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 <packaging>war</packaging>

 <dependencies>

 <dependency>
 <groupId>org.springframework</groupId>
 <artifactId>spring-webmvc</artifactId>
 <version>${spring.version}</version>
 <exclusions>
 <exclusion>
 <artifactId>commons-logging</artifactId>
 <groupId>commons-logging</groupId>
 </exclusion>
 </exclusions>
 </dependency>

 </dependencies>

 <build>
 <finalName>rest</finalName>

 <plugins>
 <plugin>
 <groupId>org.apache.maven.plugins</groupId>
 <artifactId>maven-compiler-plugin</artifactId>
 <version>3.1</version>
 <configuration>
 <source>1.6</source>
 <target>1.6</target>
 <encoding>UTF-8</encoding>
 </configuration>
 </plugin>
 </plugins>
 </build>

 <properties>
 <spring.version>4.0.5.RELEASE</spring.version>
 </properties>

</project>
```

2.1. The cglib dependency before Spring 3.2

You may wonder why *cglib* is a dependency – it turns out there is a valid reason to include it – the entire configuration cannot function without it. If removed, Spring will throw:

Caused by: java.lang.IllegalStateException: CGLIB is required to process @Configuration classes. Either add CGLIB to the classpath or remove the following @Configuration bean definitions

The reason this happens is explained by the way Spring deals with *@Configuration* classes. These classes are effectively beans, and because of this they need to be aware of the Context, and respect scope and other bean semantics. This is achieved by dynamically creating a cglib proxy with this awareness for each *@Configuration* class, hence the cglib dependency.

Also, because of this, there are a few restrictions for *Configuration* annotated classes:

- Configuration classes **should not be final**
- They should have a constructor with no arguments

2.2. The cglib dependency in Spring 3.2 and beyond

Starting with Spring 3.2, it is **no longer necessary to add cglib as an explicit dependency**. This is because Spring is in now inlining *cglib* – which will ensure that all class based proxying functionality will work out of the box with Spring 3.2.

The new cglib code is placed under the Spring package: *org.springframework.cglib* (replacing the original *net.sf.cglib*). The reason for the package change is to avoid conflicts with any *cglib* versions already existing on the classpath.

Also, the new cglib 3.0 is now used, upgraded from the older 2.2 dependency (see this JIRA issue [https://jira.springsource.org/browse/SPR-9669] for more details).

Finally, now that Spring 4.0 is out in the wild, changes like this one (removing the cglib dependency) are to be expected with Java 8 just around the corner – you can watch this Spring Jira [https://jira.springsource.org/browse/SPR-9639] to keep track of the Spring support, and the Java 8 Resources page [http://www.baeldung.com/java8] to keep tabs on the that.

3. The Java based Web Configuration

```
@Configuration
@ImportResource( { "classpath*:/rest_config.xml" } )
@ComponentScan( basePackages = "org.rest" )
@PropertySource({ "classpath:rest.properties", "classpath:web.properties" })
public class AppConfig{

 @Bean
Â Â  public static PropertySourcesPlaceholderConfigurer properties() {
Â Â  return new PropertySourcesPlaceholderConfigurer();
Â Â  }
}
```

First, the **@Configuration** annotation – this is the main artifact used by the Java based Spring configuration; it is itself meta-annotated with *@Component*, which makes the annotated classes **standard beans** and as such, also candidates for component scanning. The main purpose of *@Configuration* classes is to be sources of bean definitions for the Spring IoC Container. For a more detailed description, see the official docs [http://static.springsource.org/spring/docs/3.1.x/spring-framework-reference/html/beans.html#beans-java].

Then, **@ImportResource** is used to import the existing XML based Spring configuration. This may be configuration which is still being migrated from XML to Java, or simply legacy configuration that you wish to keep. Either way, importing it into the Container is essential for a successful migration, allowing small steps without to much risk. The equivalent XML annotation that is replaced is:

<import resource=”classpath:/rest_config.xml” />*

Moving on to **@ComponentScan** – this configures the component scanning directive, effectively replacing the XML:

```
<context:component-scan base-package="org.rest" />
```

As of Spring 3.1, the *@Configuration* are excluded from classpath scanning by default – see this JIRA issue [https://jira.springsource.org/browse/SPR-8808]. Before Spring 3.1 though, these classes should have been excluded explicitly:

```
excludeFilters = { @ComponentScan.Filter( Configuration.class ) }
```

The *@Configuration* classes should not be autodiscovered because they are already specified and used by the Container – allowing them to be rediscovered and introduced into the Spring context will result in the following error:

Caused by: org.springframework.context.annotation.ConflictingBeanDefinitionException: Annotation-specified bean name ‘webConfig’ for bean class [org.rest.spring.AppConfig] conflicts with existing, non-compatible bean definition of same name and class [org.rest.spring.AppConfig]

And finally, using the **@Bean** annotation to configure the **properties support** – Â *PropertySourcesPlaceholderConfigurer* is initialized in a *@Bean* annotated method, indicating it will produce a Spring bean managed by the Container. This new configuration has replaced the following XML:

```
<context:property-placeholder
location="classpath:persistence.properties, classpath:web.properties"
ignore-unresolvable="true"/>
```

For a more in depth discussion on why it was necessary to manually register the *PropertySourcesPlaceholderConfigurer* bean, see the Properties with Spring Tutorial [http://www.baeldung.com/2012/02/06/properties-with-spring/].

3.1. The web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="
 http://java.sun.com/xml/ns/javaee"
Â Â  Â  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
Â Â  Â  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
Â Â  Â xsi:schemaLocation="
 http://java.sun.com/xml/ns/javaee
 http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
Â Â  Â id="rest" version="3.0">

 <context-param>
 <param-name>contextClass</param-name>
 <param-value>
 org.springframework.web.context.support.AnnotationConfigWebApplicationContext
 </param-value>
 </context-param>
 <context-param>
 <param-name>contextConfigLocation</param-name>
 <param-value>org.rest.spring.root</param-value>
 </context-param>
 <listener>
       <listener-class>org.springframework.web.context.ContextLoaderListener</
listener-class>
 </listener>

 <servlet>
 <servlet-name>rest</servlet-name>
 <servlet-class>
 org.springframework.web.servlet.DispatcherServlet
 </servlet-class>
 <init-param>
 <param-name>contextClass</param-name>
 <param-value>
org.springframework.web.context.support.AnnotationConfigWebApplicationContext
 </param-value>
 </init-param>
 <init-param>
 <param-name>contextConfigLocation</param-name>
 <param-value>org.rest.spring.rest</param-value>
 </init-param>
 <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
 <servlet-name>rest</servlet-name>
 <url-pattern>/api/*</url-pattern>
 </servlet-mapping>

 <welcome-file-list>
 <welcome-file />
 </welcome-file-list>

</web-app>
```

First, the root context is defined and configured to use *AnnotationConfigWebApplicationContext* instead of the default *XmlWebApplicationContext*. The newer *AnnotationConfigWebApplicationContext* accepts *@Configuration* annotated classes as input for the Container configuration and is needed in order to set up the Java based context. Unlike *XmlWebApplicationContext*, it assumes no default configuration class locations, so the *"contextConfigLocation"init-param* for the Servlet must be set. This will point to the java package where the *@Configuration* classes are located; the fully qualified name(s) of the classes are also supported.

Next, the *DispatcherServlet* is configured to use the same kind of context, with the only difference that it's loading configuration classes out of a different package.

Other than this, the *web.xml* doesn't really change from a XML to a Java based configuration.

4. Conclusion

The presented approach allows for a smooth **migration of the Spring configuration** from XML to Java, mixing the old and the new. This is important for older projects, which may have a lot of XML based configuration that cannot be migrated all at once.

This way, in a migration, the XML beans can be ported in small increments.

In the next article on REST with Spring [http://www.baeldung.com/2011/10/25/building-a-restful-web-service-with-spring-3-1-and-java-based-configuration-part-2/], I cover setting up MVC in the project, configuration of the HTTP status codes, payload marshalling and content negotiation.

The implementation of this *Bootstrap a Spring Web App Tutorial* can be downloaded as a working sample project. [https://my.leadpages.net/leadbox/147e9e473f72a2%3A13a71ac76b46dc/5745710343389184/]

This is an Eclipse based project, so it should be easy to import and run as it is.

Â

http://twitter.com/share

java [http://www.baeldung.com/tag/java-2/], Spring [http://www.baeldung.com/tag/spring/]