HowToDoInJava

Spring Boot2 @SpringBootApplication Auto Configuration

Last Updated: November 26, 2019 By: Lokesh Gupta Spring Boot, Spring Boot 2

Spring boot is very easy to use and it does a lot of things under the hood, you might not be aware of. In future, a good developer will be who will know exactly what is going on behind **spring boot auto configuration**, how to use it in your favor and how to disable certain sections which you do not want into your project.

To understand most basic things behind spring boot, we will create a minimum boot application with single dependency and single launch class file. We will then analyze the startup logs to get the insights.

Create Spring boot application with launch class

1. Create a new maven project in eclipse with archetype "`maven-archetype-quickstart`".
2. Update `pom.xml` file with `spring-boot-starter-web` dependency and plugin information.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.howtodoinjava</groupId>
  <artifactId>springbootdemo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>springbootdemo</name>
  <url>http://maven.apache.org</url>
```

```
    <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>2.0.0.RELEASE</version>
    </parent>

    <properties>
      <java.version>1.8</java.version>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
      <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
      </dependency>
    </dependencies>

    <build>
      <plugins>
        <plugin>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
      </plugins>
    </build>

    <repositories>
      <repository>
        <id>repository.spring.release</id>
        <name>Spring GA Repository</name>
        <url>http://repo.spring.io/release</url>
      </repository>
    </repositories>
  </project>
```

3. Create launch application.

```
package com.howtodoinjava.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;

@SpringBootApplication
public class App
```

```
{
  public static void main(String[] args)
  {
    ApplicationContext ctx = SpringApplication.run(App.class, args);
  }
}
```

What this launch class does?

Above class is called spring boot application launch class. It used to Bootstrap and launch a Spring application from a Java `main()` method. It typically does following things –

- Create an instance of Spring's ApplicationContext.
- Enable the functionality to accept command-line arguments and expose them as Spring properties.
- Load all the Spring beans as per the configuration. You can do other operations as well as per project need arises.

@SpringBootApplication Annotation

This annotation is a shortcut of applying 3 annotations in one statement –

1. **@SpringBootConfiguration**

   @SpringBootConfiguration is new annotation in Spring boot 2. Previously, we have been using @Configuration annotation. You can use `@Configuration` in place of this. Both are same thing.

   It indicates that a class provides Spring Boot application `@Configuration`. It simply means that annotated class is a configuration class and shall be scanned for further configurations and bean definitions.

2. **@EnableAutoConfiguration**

This annotation is used to enable auto-configuration of the Spring Application Context, attempting to guess and configure beans that you are likely to need. Auto-configuration classes are usually applied based on your classpath and what beans you have defined.

Auto-configuration tries to be as intelligent as possible and will back-away as you define more of your own configuration. You can always manually exclude any configuration that you never want to apply using two methods –

i) Use excludeName()
ii) Using the `spring.autoconfigure.exclude` property in properties file. e.g.

```
@EnableAutoConfiguration(excludeName = {"multipartResolver","mbeanServer"})
```

Auto-configuration is always applied after user-defined beans have been registered.

3. **@ComponentScan**

This annotation provides support parallel with Spring XML's `context:component-scan` element.

Either `basePackageClasses()` or `basePackages()` may be specified to define specific packages to scan. If specific packages are not defined, scanning will occur from the package of the class that declares this annotation.

Run the launch application and check logs

Let's start running it with the simplest option–running as a Java application. In your IDE, right-click on the application class and run it as Java Application. For getting insight of registered beans, I have added modified the launch application as below.

```
@SpringBootApplication
public class App
```

```
{
  public static void main(String[] args)
  {
    ApplicationContext ctx = SpringApplication.run(App.class, args);

    String[] beanNames = ctx.getBeanDefinitionNames();

    Arrays.sort(beanNames);

    for (String beanName : beanNames) {
      System.out.println(beanName);
    }
  }
}
```

Now see the logs –

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.0.0.RELEASE)

2018-04-02 13:09:41.100  INFO 11452 --- [           main] com.howtodoinjava.demo.A
2018-04-02 13:09:41.108  INFO 11452 --- [           main] com.howtodoinjava.demo.A
2018-04-02 13:09:41.222  INFO 11452 --- [           main] ConfigServletWebServerAp
2018-04-02 13:09:43.474  INFO 11452 --- [           main] o.s.b.w.embedded.tomcat.
2018-04-02 13:09:43.526  INFO 11452 --- [           main] o.apache.catalina.core.S
2018-04-02 13:09:43.526  INFO 11452 --- [           main] org.apache.catalina.core
2018-04-02 13:09:43.748  INFO 11452 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[loca
2018-04-02 13:09:43.748  INFO 11452 --- [ost-startStop-1] o.s.web.context.ContextL
2018-04-02 13:09:43.964  INFO 11452 --- [ost-startStop-1] o.s.b.w.servlet.ServletR
2018-04-02 13:09:43.969  INFO 11452 --- [ost-startStop-1] o.s.b.w.servlet.FilterRe
2018-04-02 13:09:43.970  INFO 11452 --- [ost-startStop-1] o.s.b.w.servlet.FilterRe
2018-04-02 13:09:43.970  INFO 11452 --- [ost-startStop-1] o.s.b.w.servlet.FilterRe
2018-04-02 13:09:43.970  INFO 11452 --- [ost-startStop-1] o.s.b.w.servlet.FilterRe
2018-04-02 13:09:44.480  INFO 11452 --- [           main] s.w.s.m.m.a.RequestMappi
2018-04-02 13:09:44.627  INFO 11452 --- [           main] s.w.s.m.m.a.RequestMappi
2018-04-02 13:09:44.630  INFO 11452 --- [           main] s.w.s.m.m.a.RequestMappi
2018-04-02 13:09:44.681  INFO 11452 --- [           main] o.s.w.s.handler.SimpleUr
2018-04-02 13:09:44.682  INFO 11452 --- [           main] o.s.w.s.handler.SimpleUr
2018-04-02 13:09:44.747  INFO 11452 --- [           main] o.s.w.s.handler.SimpleUr
2018-04-02 13:09:45.002  INFO 11452 --- [           main] o.s.j.e.a.AnnotationMBea
2018-04-02 13:09:45.070  INFO 11452 --- [           main] o.s.b.w.embedded.tomcat.
2018-04-02 13:09:45.076  INFO 11452 --- [           main] com.howtodoinjava.demo.A
app
basicErrorController
```

```
beanNameHandlerMapping
beanNameViewResolver
characterEncodingFilter
conventionErrorViewResolver
defaultServletHandlerMapping
defaultValidator
defaultViewResolver
dispatcherServlet
dispatcherServletRegistration
error
errorAttributes
errorPageCustomizer
errorPageRegistrarBeanPostProcessor
faviconHandlerMapping
faviconRequestHandler
handlerExceptionResolver
hiddenHttpMethodFilter
httpPutFormContentFilter
httpRequestHandlerAdapter
jacksonCodecCustomizer
jacksonObjectMapper
jacksonObjectMapperBuilder
jsonComponentModule
localeCharsetMappingsCustomizer
mappingJackson2HttpMessageConverter
mbeanExporter
mbeanServer
messageConverters
methodValidationPostProcessor
multipartConfigElement
multipartResolver
mvcContentNegotiationManager
mvcConversionService
mvcHandlerMappingIntrospector
mvcPathMatcher
mvcResourceUrlProvider
mvcUriComponentsContributor
mvcUrlPathHelper
mvcValidator
mvcViewResolver
objectNamingStrategy
org.springframework.boot.autoconfigure.AutoConfigurationPackages
org.springframework.boot.autoconfigure.condition.BeanTypeRegistry
org.springframework.boot.autoconfigure.context.ConfigurationPropertiesAutoConfigur
org.springframework.boot.autoconfigure.context.PropertyPlaceholderAutoConfiguratio
org.springframework.boot.autoconfigure.http.HttpMessageConvertersAutoConfiguration
org.springframework.boot.autoconfigure.http.HttpMessageConvertersAutoConfiguration
org.springframework.boot.autoconfigure.http.JacksonHttpMessageConvertersConfigurat
org.springframework.boot.autoconfigure.http.JacksonHttpMessageConvertersConfigurat
org.springframework.boot.autoconfigure.http.codec.CodecsAutoConfiguration
org.springframework.boot.autoconfigure.http.codec.CodecsAutoConfiguration$JacksonC
org.springframework.boot.autoconfigure.info.ProjectInfoAutoConfiguration
org.springframework.boot.autoconfigure.internalCachingMetadataReaderFactory
org.springframework.boot.autoconfigure.jackson.JacksonAutoConfiguration
org.springframework.boot.autoconfigure.jackson.JacksonAutoConfiguration$Jackson2Ob
org.springframework.boot.autoconfigure.jackson.JacksonAutoConfiguration$JacksonObj
org.springframework.boot.autoconfigure.jackson.JacksonAutoConfiguration$JacksonObj
org.springframework.boot.autoconfigure.jackson.JacksonAutoConfiguration$ParameterN
org.springframework.boot.autoconfigure.jmx.JmxAutoConfiguration
org.springframework.boot.autoconfigure.security.reactive.ReactiveSecurityAutoConfi
org.springframework.boot.autoconfigure.validation.ValidationAutoConfiguration
org.springframework.boot.autoconfigure.web.client.RestTemplateAutoConfiguration
org.springframework.boot.autoconfigure.web.embedded.EmbeddedWebServerFactoryCustom
org.springframework.boot.autoconfigure.web.embedded.EmbeddedWebServerFactoryCustom
org.springframework.boot.autoconfigure.web.servlet.DispatcherServletAutoConfigurat
org.springframework.boot.autoconfigure.web.servlet.DispatcherServletAutoConfigurat
org.springframework.boot.autoconfigure.web.servlet.DispatcherServletAutoConfigurat
org.springframework.boot.autoconfigure.web.servlet.HttpEncodingAutoConfiguration
org.springframework.boot.autoconfigure.web.servlet.MultipartAutoConfiguration
org.springframework.boot.autoconfigure.web.servlet.ServletWebServerFactoryAutoConf
org.springframework.boot.autoconfigure.web.servlet.ServletWebServerFactoryConfigur
org.springframework.boot.autoconfigure.web.servlet.WebMvcAutoConfiguration
org.springframework.boot.autoconfigure.web.servlet.WebMvcAutoConfiguration$EnableW
org.springframework.boot.autoconfigure.web.servlet.WebMvcAutoConfiguration$WebMvcA
org.springframework.boot.autoconfigure.web.servlet.WebMvcAutoConfiguration$WebMvcA
org.springframework.boot.autoconfigure.web.servlet.error.ErrorMvcAutoConfiguration
org.springframework.boot.autoconfigure.web.servlet.error.ErrorMvcAutoConfiguration
org.springframework.boot.autoconfigure.web.servlet.error.ErrorMvcAutoConfiguration
org.springframework.boot.autoconfigure.websocket.servlet.WebSocketServletAutoConfi
org.springframework.boot.autoconfigure.websocket.servlet.WebSocketServletAutoConfi
org.springframework.boot.context.properties.ConfigurationBeanFactoryMetadata
org.springframework.boot.context.properties.ConfigurationPropertiesBindingPostProc
org.springframework.context.annotation.internalAutowiredAnnotationProcessor
org.springframework.context.annotation.internalCommonAnnotationProcessor
org.springframework.context.annotation.internalConfigurationAnnotationProcessor
org.springframework.context.annotation.internalRequiredAnnotationProcessor
org.springframework.context.event.internalEventListenerFactory
org.springframework.context.event.internalEventListenerProcessor
parameterNamesModule
preserveErrorControllerTargetClassPostProcessor
propertySourcesPlaceholderConfigurer
requestContextFilter
requestMappingHandlerAdapter
requestMappingHandlerMapping
resourceHandlerMapping
restTemplateBuilder
server-org.springframework.boot.autoconfigure.web.ServerProperties
servletWebServerFactoryCustomizer
simpleControllerHandlerAdapter
spring.http.encoding-org.springframework.boot.autoconfigure.http.HttpEncodingPrope
spring.info-org.springframework.boot.autoconfigure.info.ProjectInfoProperties
spring.jackson-org.springframework.boot.autoconfigure.jackson.JacksonProperties
spring.mvc-org.springframework.boot.autoconfigure.web.servlet.WebMvcProperties
spring.resources-org.springframework.boot.autoconfigure.web.ResourceProperties
spring.security-org.springframework.boot.autoconfigure.security.SecurityProperties
spring.servlet.multipart-org.springframework.boot.autoconfigure.web.servlet.Multip
standardJacksonObjectMapperBuilderCustomizer
stringHttpMessageConverter
tomcatServletWebServerFactory
tomcatServletWebServerFactoryCustomizer
tomcatWebServerFactoryCustomizer
viewControllerHandlerMapping
viewResolver
webServerFactoryCustomizerBeanPostProcessor
websocketContainerCustomizer
welcomePageHandlerMapping
```

You see how many beans got registered automatically. That's beauty of spring boot. If you want to dig deeper into why any particular bean got registered? You can see that by putting a debug flag at application startup.

Simply pass –`Ddebug=true` as VM argument.

Now when you run the application, you will get lots of debug logs having similar information :

```
CodecsAutoConfiguration.JacksonCodecConfiguration matched:
  - @ConditionalOnClass found required class 'com.fasterxml.jackson.databind.Objec

CodecsAutoConfiguration.JacksonCodecConfiguration#jacksonCodecCustomizer matched:
  - @ConditionalOnBean (types: com.fasterxml.jackson.databind.ObjectMapper; Search

DispatcherServletAutoConfiguration.DispatcherServletConfiguration matched:
  - @ConditionalOnClass found required class 'javax.servlet.ServletRegistration';
  - Default DispatcherServlet did not find dispatcher servlet beans (DispatcherSer

DispatcherServletAutoConfiguration.DispatcherServletRegistrationConfiguration matc
  - @ConditionalOnClass found required class 'javax.servlet.ServletRegistration';
  - DispatcherServlet Registration did not find servlet registration bean (Dispatc

  ...
  ...
  ...
```

Above logs tell why a particular bean was registered into spring context. This information is very useful when you debug the issues with auto configutation.

Similarily, everytime we add a new dependency to a Spring Boot project, Spring Boot auto-configuration automatically tries to configure the beans based on the dependency.

I hope that information discussed above will help you in future while debugging spring boot related issues.

Happy Learning !!

Recommended Reading:

1. Java WatchService Example to Auto Reload Properties
2. Spring Boot – Custom PropertyEditor Configuration Example
3. Spring Boot – Embedded Tomcat Configuration
4. Spring Boot DataSource Configuration
5. Spring boot JPA + Hibernate + HikariCP Configuration
6. Spring-boot-starter Maven Templates
7. Spring-boot-starter-parent Example
8. Spring Boot war Packaging Example
9. Spring Boot Dev Tools Tutorial

0. 13 Spring Best Practices for Writing Configuration Files

Join 7000+ Awesome Developers

Get the latest updates from industry, awesome resources, blog updates and much more.

Email Address

Subscribe

** We do not spam !!*

Leave a Comment

Name *

Email *

Website

☐ Add me to your newsletter and keep me updated whenever you publish new blog posts

Post Comment

Search ...

HowToDoInJava

A blog about Java and related technologies, the best practices, algorithms, and interview questions.

Meta Links

- About Me
- Contact Us
- Privacy policy
- Advertise
- Guest Posts

Blogs

REST API Tutorial