# 三、SpringBoot原理分析

## 3.1 起步依赖原理分析

### 分析spring-boot-starter-parent

按住Ctrl点击pom.xml中的spring-boot-starter-parent，跳转到了spring-boot-starter-parent的pom.xml，xml配 置如下（只摘抄了部分重点配置）：

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-dependencies</artifactId>

<version>2.0.1.RELEASE</version>

<relativePath>../../spring-boot-dependencies</relativePath>

</parent>

按住Ctrl点击pom.xml中的spring-boot-starter-dependencies，跳转到了spring-boot-starter-dependencies的

pom.xml，xml配置如下（只摘抄了部分重点配置）：

<properties>

<activemq.version>5.15.3</activemq.version>

<antlr2.version>2.7.7</antlr2.version>

<appengine-sdk.version>1.9.63</appengine-sdk.version>

<artemis.version>2.4.0</artemis.version>

<aspectj.version>1.8.13</aspectj.version>

<assertj.version>3.9.1</assertj.version>

<atomikos.version>4.0.6</atomikos.version>

<bitronix.version>2.1.4</bitronix.version>

<build-helper-maven-plugin.version>3.0.0</build-helper-maven-plugin.version>

<byte-buddy.version>1.7.11</byte-buddy.version>

</properties>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot</artifactId>

<version>2.0.1.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-test</artifactId>

<version>2.0.1.RELEASE</version>

</dependency>

26 ... ... ...

</dependencies>

</dependencyManagement>

<build>

<pluginManagement>

<plugins>

<plugin>

<groupId>org.jetbrains.kotlin</groupId>

<artifactId>kotlin-maven-plugin</artifactId>

<version>${kotlin.version}</version>

</plugin>

<plugin>

<groupId>org.jooq</groupId>

<artifactId>jooq-codegen-maven</artifactId>

<version>${jooq.version}</version>

</plugin>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<version>2.0.1.RELEASE</version>

</plugin>

47 ... ... ...

</plugins>

</pluginManagement>

</build>

从上面的spring-boot-starter-dependencies的pom.xml中我们可以发现，一部分坐标的版本、依赖管理、插件管 理已经定义好，所以我们的SpringBoot工程继承spring-boot-starter-parent后已经具备版本锁定等配置了。所以 起步依赖的作用就是进行依赖的传递。

### 分析spring-boot-starter-web

按住Ctrl点击pom.xml中的spring-boot-starter-web，跳转到了spring-boot-starter-web的pom.xml，xml配置如 下（只摘抄了部分重点配置）：

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

<project xsi:schemaLocation="<http://maven.apache.org/POM/4.0.0>

<http://maven.apache.org/xsd/maven-4.0.0.xsd>" xmlns="<http://maven.apache.org/POM/4.0.0>"

xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starters</artifactId>

<version>2.0.1.RELEASE</version>

</parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

<version>2.0.1.RELEASE</version>

<name>Spring Boot Web Starter</name> 14

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

<version>2.0.1.RELEASE</version>

<scope>compile</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-json</artifactId>

<version>2.0.1.RELEASE</version>

<scope>compile</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-tomcat</artifactId>

<version>2.0.1.RELEASE</version>

<scope>compile</scope>

</dependency>

<dependency>

<groupId>org.hibernate.validator</groupId>

<artifactId>hibernate-validator</artifactId>

<version>6.0.9.Final</version>

<scope>compile</scope>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>5.0.5.RELEASE</version>

<scope>compile</scope>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.0.5.RELEASE</version>

<scope>compile</scope>

</dependency>

</dependencies>

</project>

从上面的spring-boot-starter-web的pom.xml中我们可以发现，spring-boot-starter-web就是将web开发要使用的 spring-web、spring-webmvc等坐标进行了“打包”，这样我们的工程只要引入spring-boot-starter-web起步依赖的 坐标就可以进行web开发了，同样体现了依赖传递的作用。

## 3.2 自动配置原理解析

按住Ctrl点击查看启动类MySpringBootApplication上的注解@SpringBootApplication

@SpringBootApplication

public class MySpringBootApplication {

public static void main(String[] args) {

SpringApplication.run(MySpringBootApplication.class); }

}

注解@SpringBootApplication的源码

1. @Target(ElementType.TYPE)
2. @Retention(RetentionPolicy.RUNTIME)
3. @Documented
4. @Inherited
5. @SpringBootConfiguration
6. @EnableAutoConfiguration
7. @ComponentScan(excludeFilters = {
8. @Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class),
9. @Filter(type = FilterType.CUSTOM, classes = AutoConfigurationExcludeFilter.class) })

public @interface SpringBootApplication{

/\*\*

* Exclude specific auto-configuration classes such that they will never be applied.
* @return the classes to exclude

\*/

@AliasFor(annotation = EnableAutoConfiguration.class) Class<?>[] exclude() default {};

... ... ...

}

其中，

@SpringBootConﬁguration：等同与@Conﬁguration，既标注该类是Spring的一个配置类

@EnableAutoConﬁguration：SpringBoot自动配置功能开启

按住Ctrl点击查看注解@EnableAutoConﬁguration

@Target(ElementType.TYPE)

@Retention(RetentionPolicy.RUNTIME)

@Documented

@Inherited

@AutoConfigurationPackage

@Import(AutoConfigurationImportSelector.class)

public @interface EnableAutoConfiguration { 8 ... ... ...

}

其中，@Import(AutoConﬁgurationImportSelector.class) 导入了AutoConﬁgurationImportSelector类

按住Ctrl点击查看AutoConﬁgurationImportSelector源码

public String[] selectImports(AnnotationMetadata annotationMetadata) { ... ... ...

List<String> configurations = getCandidateConfigurations(annotationMetadata,

attributes);

configurations = removeDuplicates(configurations);

Set<String> exclusions = getExclusions(annotationMetadata, attributes);

checkExcludedClasses(configurations, exclusions);

configurations.removeAll(exclusions);

configurations = filter(configurations, autoConfigurationMetadata);

fireAutoConfigurationImportEvents(configurations, exclusions);

return StringUtils.toStringArray(configurations); }

protected List<String> getCandidateConfigurations(AnnotationMetadata metadata,

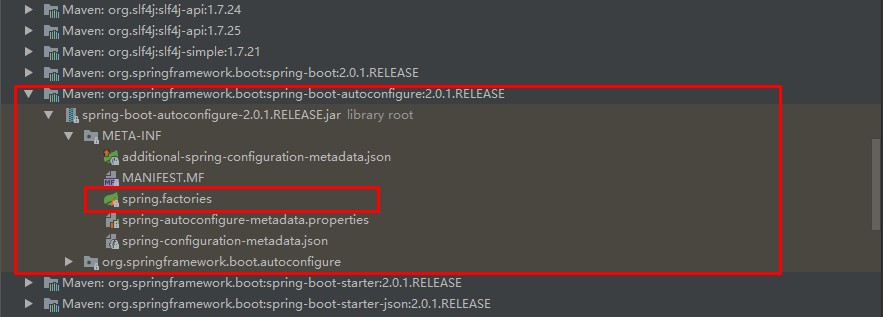
AnnotationAttributes attributes) {

List<String> configurations = SpringFactoriesLoader.loadFactoryNames(

getSpringFactoriesLoaderFactoryClass(), getBeanClassLoader());

return configurations; }

其中，SpringFactoriesLoader.loadFactoryNames 方法的作用就是从META-INF/spring.factories文件中读取指定 类对应的类名称列表



spring.factories 文件中有关自动配置的配置信息如下：

org.springframework.boot.autoconfigure.web.reactive.function.client.WebClientAutoConf

iguration,\ org.springframework.boot.autoconfigure.web.servlet.DispatcherServletAutoConfiguration

,\ org.springframework.boot.autoconfigure.web.servlet.ServletWebServerFactoryAutoConfigu ration,\ org.springframework.boot.autoconfigure.web.servlet.error.ErrorMvcAutoConfiguration,\ org.springframework.boot.autoconfigure.web.servlet.HttpEncodingAutoConfiguration,\

org.springframework.boot.autoconfigure.web.servlet.MultipartAutoConfiguration,\

上面配置文件存在大量的以Conﬁguration为结尾的类名称，这些类就是存有自动配置信息的类，而

SpringApplication在获取这些类名后再加载

我们以ServletWebServerFactoryAutoConﬁguration为例来分析源码：

@Configuration

@AutoConfigureOrder(Ordered.HIGHEST\_PRECEDENCE)

@ConditionalOnClass(ServletRequest.class)

@ConditionalOnWebApplication(type = Type.SERVLET)

@EnableConfigurationProperties(ServerProperties.class)

@Import({ ServletWebServerFactoryAutoConfiguration.BeanPostProcessorsRegistrar.class,

ServletWebServerFactoryConfiguration.EmbeddedTomcat.class,

ServletWebServerFactoryConfiguration.EmbeddedJetty.class,

ServletWebServerFactoryConfiguration.EmbeddedUndertow.class })

public class ServletWebServerFactoryAutoConfiguration { 11 ... ... ...

}

其中，

@EnableConﬁgurationProperties(ServerProperties.class) 代表加载ServerProperties服务器配置属性类

进入ServerProperties.class源码如下：

@ConfigurationProperties(prefix = "server", ignoreUnknownFields = true)

public class ServerProperties {

/\*\*

\* Server HTTP port.

\*/

private Integer port;

/\*\*

\* Network address to which the server should bind.

\*/

private InetAddress address;

}

其中，

preﬁx = "server" 表示SpringBoot配置文件中的前缀，SpringBoot会将配置文件中以server开始的属性映射到该类 的字段中。映射关系如下：