# Spring Boot核心-运行原理

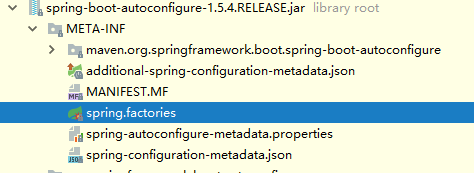
Spring Boot 关于自动配置的源码在spring-boot-autoconfigure-xxx.jar中。可以通过下面三种方式查看当前项目中已启用和未启用的自动配置的报告。

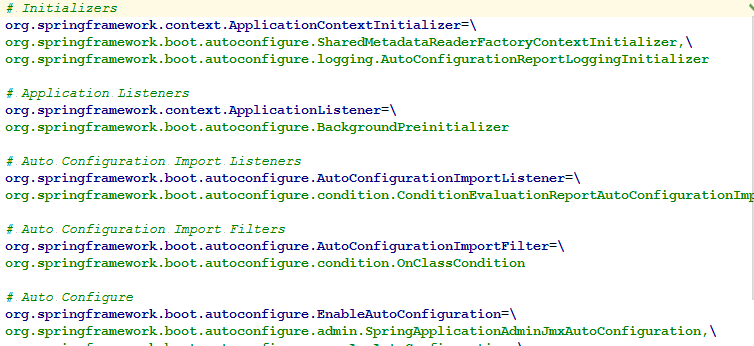
1. 运行jar 时增加--debug 参数
2. 在applic剖ion.properties 中设置属性。
3. 运行时设置JVM启动参数-Ddebug

Spring Boot 的运作原理，它的核心功能是由＠EnableAutoConfiguration 注解提供的。

|  |
| --- |
| @SuppressWarnings(**"deprecation"**) @Target(ElementType.***TYPE***) @Retention(RetentionPolicy.***RUNTIME***) @Documented @Inherited @AutoConfigurationPackage @Import(EnableAutoConfigurationImportSelector.**class**) **public** @**interface** EnableAutoConfiguration {  String ***ENABLED\_OVERRIDE\_PROPERTY*** = **"spring.boot.enableautoconfiguration"**;Class<?>[] exclude() **default** {};String[] excludeName() **default** {}; } |
| @Target(ElementType.***TYPE***) @Retention(RetentionPolicy.***RUNTIME***) @Documented @Inherited @Import(AutoConfigurationPackages.Registrar.**class**) **public** @**interface** AutoConfigurationPackage {} |

@Import 注解导入的配置功能， EnableAutoConfigurationlmportSelector 使用SpringFactoriesLoader.loadFactoryNames方法来扫描具有META-INF/spring.factories文件的jar包，而我们的spring-boot-autoconfigure- xxx.jar 果就有一spring. factories 文件，此文件中声明了有哪些自动配置。





打开上面任意一个AutoConfiguration 文件，一般都有下面的条件注解，在spring-boot-autoconfigure-xxx.jar 的org.springframwork.boot.autocon figure.condition 包下，条件注解如下。

@ConditionalOnBean ：当容器里有指定的Bean 的条件下。

@ConditionalOnClass ：当类路径下有指定的类的条件下。

@ConditionalOnCloudPlatform：当指定了云平台的时候

@ConditionalOnExpression ：基于SpEL 表达式作为判断条件

@ConditionalOnJava：基于JVM 版本作为判断条件。

@ConditionalOnJndi ：在JNDI 存在的条件下查找指定的位置。

@ConditionalOnMissingBean ：当容器里没有指定Bean 的情况下。

@ConditionalOnMissingClass ：当类路径下没有指定的类的条件下。

@ConditionalOnNotWebApplication ：当前项目不是Web 项目的条件下。

@ConditionalOnProperty ：指定的属性是否有指定的值。

@ConditionalOnResource ：类路径是否有指定的值。

@ConditionalOnSingleCandidate：当指定Bean 在容器中只有一个，或者虽然有多个但是指定首选的Bean

@Conditional On WebApplication：当前项目是Web 项目的条件下。

这些注解都是组合了＠Conditional 元注解，只是使用了不同的条件（ Condition ），下面我分析一下@ConditionalOn WebApplication 注解。

|  |
| --- |
| @Target({ ElementType.***TYPE***, ElementType.***METHOD*** }) @Retention(RetentionPolicy.***RUNTIME***) @Documented @Conditional(OnWebApplicationCondition.**class**) **public** @**interface** ConditionalOnWebApplication {} |
| @Order(Ordered.***HIGHEST\_PRECEDENCE*** + **20**) **class** OnWebApplicationCondition **extends** SpringBootCondition {   **private static final** String ***WEB\_CONTEXT\_CLASS*** = **"org.springframework.web.context."** + **"support.GenericWebApplicationContext"**;   @Override  **public** ConditionOutcome getMatchOutcome(ConditionContext context,  AnnotatedTypeMetadata metadata) {  **boolean** required = metadata  .isAnnotated(ConditionalOnWebApplication.**class**.getName());  ConditionOutcome outcome = isWebApplication(context, metadata, required);  **if** (required && !outcome.isMatch()) {  **return** ConditionOutcome.*noMatch*(outcome.getConditionMessage());  }  **if** (!required && outcome.isMatch()) {  **return** ConditionOutcome.*noMatch*(outcome.getConditionMessage());  }  **return** ConditionOutcome.*match*(outcome.getConditionMessage());  }   **private** ConditionOutcome isWebApplication(ConditionContext context,  AnnotatedTypeMetadata metadata, **boolean** required) {  ConditionMessage.Builder message = ConditionMessage.*forCondition*(  ConditionalOnWebApplication.**class**, required ? **"(required)"** : **""**);  **if** (!ClassUtils.*isPresent*(***WEB\_CONTEXT\_CLASS***, context.getClassLoader())) {  **return** ConditionOutcome  .*noMatch*(message.didNotFind(**"web application classes"**).atAll());  }  **if** (context.getBeanFactory() != **null**) {  String[] scopes = context.getBeanFactory().getRegisteredScopeNames();  **if** (ObjectUtils.*containsElement*(scopes, **"session"**)) {  **return** ConditionOutcome.*match*(message.foundExactly(**"'session' scope"**));  }  }  **if** (context.getEnvironment() **instanceof** StandardServletEnvironment) {  **return** ConditionOutcome  .*match*(message.foundExactly(**"StandardServletEnvironment"**));  }  **if** (context.getResourceLoader() **instanceof** WebApplicationContext) {  **return** ConditionOutcome.*match*(message.foundExactly(**"WebApplicationContext"**));  }  **return** ConditionOutcome.*noMatch*(message.because(**"not a web application"**));  } } |

从isWebApplication 方法可以看出，判断条件是：

1. GenericWebApplicationContext 是否在类路径中：
2. 容器里是否有名为session 的scope;
3. 当前容器的Enviroment 是否为StandardServletEnvironment ·
4. 当前的ResourceLoader 是否为WebApplicationContext ( ResourceLoader 是ApplicationContext 的顶级接口之一）：
5. 我们需要构造ConditionOutcome 类的对象来帮助我们，最终通过ConditionOutcome.isMatch 方法返回布尔值来确定条件。

## 实例分析

在了解了Spring Boot 的运作原理和主要的条件注解后，现在来分析一个简单的Spring Boot 内置的自动配置功能： http 的编码配置。我们在常规的项目中配置http编码时候是在web.xml里配置一个filter

|  |
| --- |
| <filter>  <filter-name>characterEncodingFilter</filter-name>  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>  <init-param>  <param-name>encoding</param-name>  <param-value>UTF-8</param-value>  </init-param>  <init-param>  <param-name>forceEncoding</param-name>  <param-value>true</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>characterEncodingFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |

自动配置要满足两个条件：

1. 能自己置CharacterEncodingFilter 这个Bean;
2. 能自己置encoding 和forceEncoding 这两个参数。

Spring Boot 的自动配置是类型安全的配置，可以在application.properties 中直接设置。

|  |
| --- |
| @ConfigurationProperties(prefix = **"spring.http.encoding"**) **public class** HttpEncodingProperties {  **public static final** Charset ***DEFAULT\_CHARSET*** = Charset.*forName*(**"UTF-8"**);**private** Charset **charset** = ***DEFAULT\_CHARSET***;**private** Boolean **force**;**private** Boolean **forceRequest**;**private** Boolean **forceResponse**;**private** Map<Locale, Charset> **mapping**;  } |

说明：

1. 在application.properties 配置的时候前缀是spring.http.encoding;
2. 默认编码方式为UTF芯，若修改可使用spring.http.encoding.charset＝编码；
3. 设置forceEncoding ，默认为true ，若修改可使用spring. http.encoding. force=false

|  |
| --- |
| @Configuration @EnableConfigurationProperties(HttpEncodingProperties.**class**) // 1 @ConditionalOnWebApplication // 6 @ConditionalOnClass(CharacterEncodingFilter.**class**) // 2 @ConditionalOnProperty(prefix = **"spring.http.encoding"**, value = **"enabled"**, matchIfMissing = **true**) // 3 **public class** HttpEncodingAutoConfiguration {   **private final** HttpEncodingProperties **properties**; // 4   **public** HttpEncodingAutoConfiguration(HttpEncodingProperties properties) {  **this**.**properties** = properties;  }   @Bean // 5  @ConditionalOnMissingBean(CharacterEncodingFilter.**class**)  **public** CharacterEncodingFilter characterEncodingFilter() {  CharacterEncodingFilter filter = **new** OrderedCharacterEncodingFilter();  filter.setEncoding(**this**.**properties**.getCharset().name());  filter.setForceRequestEncoding(**this**.**properties**.shouldForce(Type.***REQUEST***));  filter.setForceResponseEncoding(**this**.**properties**.shouldForce(Type.***RESPONSE***));  **return** filter;  } |

说明：

1. 开启属性注入，通过@EnableConfigurationProperties 声明，使用@Autowired 注入；
2. 当CharacterEncodingFilter 在类路径的条件下；
3. 当设置spring.http.encoding=enabled 的情况下，如果没有设置则默认为true ，即条件符合；
4. 像使用Java 配置的方式配置CharacterEncodingFilter 这个Bean。
5. 当容器中没有这个Bean 的时候新建Beane
6. 在web环境下才创建Bean