

引言

不论在工作中，亦或是求职面试，Spring Boot已经成为我们必知必会的技能项。除了某些老旧的政府项目或金融项目持有观望态度外，如今的各行各业拥抱这个已经不是很新的Spring启动框架。

当然，作为Spring Boot的精髓，自动配置原理的工作过程往往只有在“面试”的时候才能用得上，但是如果在工作中你能够深入的理解Spring Boot的自动将无往不利。

Spring Boot的出现，得益于“习惯优于配置”的理念，没有繁琐的配置、难以集成的内容（大多数流行第三方技术都被集成），这是基于Spring 4.x提供的Bean的能力。

Spring Boot的配置文件

初识Spring Boot时我们就知道，Spring Boot有一个全局配置文件：application.properties或application.yml。

我们的各种属性都可以在这个文件中进行配置，最常配置的比如：server.port、logging.level.*等等，然而我们实际用到的往往只是很少的一部分，那么否有据可依呢？答案当然是肯定的，这些属性都可以在官方文档中查找到：

<https://docs.spring.io/spring-boot/docs/2.1.0.RELEASE/reference/htmlsingle/#common-application-properties>

```
# -----
# CORE PROPERTIES
# -----
debug=false # Enable debug Logs.
trace=false # Enable trace Logs.

# LOGGING
logging.config= # Location of the logging configuration file. For instance, `classpath:logback.xml` for Logback.
logging.exception-conversion-word=%wEx # Conversion word used when logging exceptions.
logging.file= # Log file name (for instance, `myapp.log`). Names can be an exact location or relative to the current directory
logging.file.max-history=0 # Maximum of archive log files to keep. Only supported with the default logback setup.
logging.file.max-size=10MB # Maximum log file size. Only supported with the default logback setup.
logging.group.*= # Log groups to quickly change multiple loggers at the same time. For instance, `logging.level.db=org.hibernate
logging.level.*= # Log levels severity mapping. For instance, `logging.level.org.springframework=DEBUG`.
logging.path= # Location of the log file. For instance, `/var/log`.
logging.pattern.console= # Appender pattern for output to the console. Supported only with the default Logback setup.
logging.pattern.dateformat=yyyy-MM-dd HH:mm:ss.SSS # Appender pattern for log date format. Supported only with the default Log
logging.pattern.file= # Appender pattern for output to a file. Supported only with the default Logback setup.
logging.pattern.level=%5p # Appender pattern for log level. Supported only with the default Logback setup.
logging.register-shutdown-hook=false # Register a shutdown hook for the logging system when it is initialized.

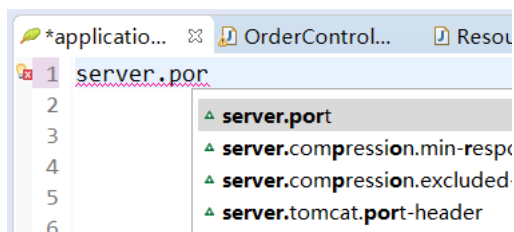
# AOP
spring.aop.auto=true # Add @EnableAspectJAutoProxy.
spring.aop.proxy-target-class=true # Whether subclass-based (CGLIB) proxies are to be created (true), as opposed to standard J
```

<https://blog.csdn.net/u014745069>

（所以，话又说回来，找资料还得是官方文档，百度出来一大堆，还是稍显业余了一些）

除了官方文档为我们提供了大量的属性解释，我们也可以使用IDE的相关提示功能，比如IDEA的自动提示，和Eclipse的YEdit插件，都可以很好的对你需

性进行提示，下图是使用Eclipse的YEdit插件的效果，Eclipse的版本是：STS 4。



以上，是Spring Boot的配置文件的大致使用方法，其实都是些题外话。

那么问题来了：这些配置是如何在Spring Boot项目中生效的呢？那么接下来，就需要聚焦本篇博客的主题：自动配置工作原理或者叫实现方式。

工作原理剖析

Spring Boot关于自动配置的源码在spring-boot-autoconfigure-x.x.x.jar中：

```

spring-boot-autoconfigure-2.0.6.RELEASE.jar
├── org.springframework.boot.autoconfigure
│   ├── admin
│   ├── amqp
│   ├── aop
│   ├── batch
│   ├── cache
│   ├── cassandra
│   ├── cloud
│   ├── condition
│   ├── context
│   ├── couchbase
│   ├── dao
│   ├── data
│   └── diagnostics.analyzer

```

当然，自动配置原理的相关描述，官方文档貌似是没有提及。不过我们不难猜出，Spring Boot的启动类上有一个@SpringBootApplication注解，这个注解项目必不可少的。那么自动配置原理一定和这个注解有着千丝万缕的联系！

@EnableAutoConfiguration

```

46 @Target(ElementType.TYPE)
47 @Retention(RetentionPolicy.RUNTIME)
48 @Documented
49 @Inherited
50 @SpringBootApplication
51 @EnableAutoConfiguration
52 @ComponentScan(excludeFilters = {
53     @Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class),
54     @Filter(type = FilterType.CUSTOM, classes = AutoConfigurationExcludeFilter.class) })
55 public @interface SpringBootApplication {

```

@SpringBootApplication是一个复合注解或派生注解，在@SpringBootApplication中有一个注解@EnableAutoConfiguration，翻译成成人话就是开启自动配置如下：

```

@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@AutoConfigurationPackage
@Import({AutoConfigurationImportSelector.class})
public @interface EnableAutoConfiguration {

```

而这个注解也是一个派生注解，其中的关键功能由@Import提供，其导入的AutoConfigurationImportSelector的selectImports()方法通过SpringFactoriesLoader.loadFactoryNames()扫描所有具有META-INF/spring.factories的jar包。spring-boot-autoconfigure-x.x.x.jar里就有一个这样的spring.factories文件。

这个spring.factories文件也是一组一组的key=value的形式，其中一个key是EnableAutoConfiguration类的全类名，而它的value是一个xxxxAutoConfiguration的列表，这些类名以逗号分隔，如下图所示：

```

spring-boot-autoconfigure-2.0.6.RELEASE.jar - C:\Users\mht\m2\repository\org.springframework.boot\spring-boot-autoconfigure-2.0.6.RELEASE.jar
├── META-INF
│   ├── additional-spring-configuration-metadata.json
│   ├── MANIFEST.MF
│   ├── spring-autoconfigure-metadata.properties
│   ├── spring-configuration-metadata.json
│   └── spring.factories
├── spring-boot-starter-logging-2.0.6.RELEASE.jar - C:\Users\mht\m2\repository\org.springframework.boot\spring-boot-starter-logging-2.0.6.RELEASE.jar
├── logback-classic-1.2.3.jar - C:\Users\mht\m2\repository\ch.qos.logback\logback-classic-1.2.3.jar
├── logback-core-1.2.3.jar - C:\Users\mht\m2\repository\ch.qos.logback\logback-core-1.2.3.jar
├── log4j-to-slf4j-2.10.0.jar - C:\Users\mht\m2\repository\org.apache.logging.log4j\log4j-to-slf4j-2.10.0.jar
├── log4j-api-2.10.0.jar - C:\Users\mht\m2\repository\org.apache.logging.log4j\log4j-api-2.10.0.jar
├── jul-to-slf4j-1.7.25.jar - C:\Users\mht\m2\repository\org.apache.logging.log4j\jul-to-slf4j-1.7.25.jar
└── javax.annotation-api-1.3.2.jar - C:\Users\mht\m2\repository\javax\annotation-api-1.3.2.jar

14# Auto Configuration Import Filters
15 org.springframework.boot.autoconfigure.AutoConfigurationImportFilter=\
16 org.springframework.boot.autoconfigure.condition.OnClassCondition
17
18# Auto Configure
19 org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
20 org.springframework.boot.autoconfigure.admin.SpringApplicationAdminJmxAutoConfiguration,\
21 org.springframework.boot.autoconfigure.amqp.RabbitAutoConfiguration,\
22 org.springframework.boot.autoconfigure.batch.BatchAutoConfiguration,\
23 org.springframework.boot.autoconfigure.cache.CacheAutoConfiguration,\
24 org.springframework.boot.autoconfigure.cassandra.CassandraAutoConfiguration,\
25 org.springframework.boot.autoconfigure.cloud.CloudAutoConfiguration,\
26 org.springframework.boot.autoconfigure.context.ConfigurationPropertiesAutoConfiguration,\
27 org.springframework.boot.autoconfigure.context.MessageSourceAutoConfiguration,\
28 org.springframework.boot.autoconfigure.context.PropertyPlaceholderAutoConfiguration,\
29 org.springframework.boot.autoconfigure.context.support.SpringPropertyPlaceholderAutoConfiguration,\
30 org.springframework.boot.autoconfigure.data.cassandra.CassandraDataAutoConfiguration,\
31 org.springframework.boot.autoconfigure.data.couchbase.CouchbaseDataAutoConfiguration,\
32 org.springframework.boot.autoconfigure.data.elasticsearch.ElasticsearchDataAutoConfiguration,\
33 org.springframework.boot.autoconfigure.data.gemfire.GemfireDataAutoConfiguration,\
34 org.springframework.boot.autoconfigure.data.hazelcast.HazelcastDataAutoConfiguration,\
35 org.springframework.boot.autoconfigure.data.influxdb.InfluxDbDataAutoConfiguration,\
36 org.springframework.boot.autoconfigure.data.jdbc.JdbcDataAutoConfiguration,\
37 org.springframework.boot.autoconfigure.data.mongo.MongoDataAutoConfiguration,\
38 org.springframework.boot.autoconfigure.data.mssql.MssqlDataAutoConfiguration,\
39 org.springframework.boot.autoconfigure.data.mysql.MySqlDataAutoConfiguration,\
40 org.springframework.boot.autoconfigure.data.nitrite.NitriteDataAutoConfiguration,\
41 org.springframework.boot.autoconfigure.data.redis.RedisDataAutoConfiguration,\
42 org.springframework.boot.autoconfigure.data.schemacrawler.SchemaCrawlerDataAutoConfiguration,\
43 org.springframework.boot.autoconfigure.data.slick.SlickDataAutoConfiguration,\
44 org.springframework.boot.autoconfigure.data.sqlite.SQLiteDataAutoConfiguration,\
45 org.springframework.boot.autoconfigure.data.terracotta.TerracottaDataAutoConfiguration,\
46 org.springframework.boot.autoconfigure.data.vault.VaultDataAutoConfiguration,\
47 org.springframework.boot.autoconfigure.data.xml.XmlDataAutoConfiguration,\
48 org.springframework.boot.autoconfigure.data.xmpp.XmppDataAutoConfiguration,\
49 org.springframework.boot.autoconfigure.data.yamdb.YamdbDataAutoConfiguration,\
50 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
51 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
52 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
53 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
54 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
55 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
56 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
57 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
58 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
59 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
60 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
61 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
62 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
63 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
64 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
65 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
66 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
67 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
68 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
69 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
70 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
71 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
72 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
73 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
74 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
75 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
76 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
77 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
78 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
79 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
80 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
81 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
82 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
83 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
84 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
85 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
86 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
87 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
88 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
89 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
90 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
91 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
92 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
93 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
94 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
95 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
96 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
97 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
98 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
99 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\
100 org.springframework.boot.autoconfigure.data.zookeeper.ZookeeperDataAutoConfiguration,\

```

这个@EnableAutoConfiguration注解通过@SpringBootApplication被间接的标记在了Spring Boot的启动类上。在SpringApplication.run(...)的内部就会执行selectImports()方法，找到所有JavaConfig自动配置类的全限定名对应的class，然后将所有自动配置类加载到Spring容器中。

自动配置生效

每一个XxxxAutoConfiguration自动配置类都是在某些条件之下才会生效的，这些条件的限制在Spring Boot中以注解的形式体现，常见的条件注解有如下

@ConditionalOnBean：当容器里有指定的bean的条件下。

@ConditionalOnMissingBean：当容器里不存在指定bean的条件下。

@ConditionalOnClass：当类路径下有指定类的条件下。

@ConditionalOnMissingClass：当类路径下不存在指定类的条件下。

@ConditionalOnProperty：指定的属性是否有指定的值，比如@ConditionalOnProperties(prefix="xxx.xxx", value="enable", matchIfMissing=true)，xxx.xxx为enable时条件的布尔值为true，如果没有设置的情况下也为true。

以ServletWebServerFactoryAutoConfiguration配置类为例，解释一下全局配置文件中的属性如何生效，比如：server.port=8081，是如何生效的（当然默认值，这个默认值来自于org.apache.catalina.startup.Tomcat）。

```
@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 {
```

在ServletWebServerFactoryAutoConfiguration类上，有一个@EnableConfigurationProperties注解：开启配置属性，而它后面的参数是一个ServerProperties就是习惯优于配置的最终落地点。

```
@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;

    @NestedConfigurationProperty
    private final ErrorProperties error = new ErrorProperties();
```

在这个类上，我们看到了一个非常熟悉的注解：@ConfigurationProperties，它的作用就是从配置文件中绑定属性到对应的bean上，而@EnableConfigurationProperties负责导入这个已经绑定了属性的bean到spring容器中（见上面截图）。那么所有其他的和这个类相关的属性都可以在配置文件中定义，也就是说，真正“限制”我们可以在全局配置文件中配置哪些属性的类就是这些XxxxProperties类，它与配置文件中定义的prefix关键字开头的唯一对应的。

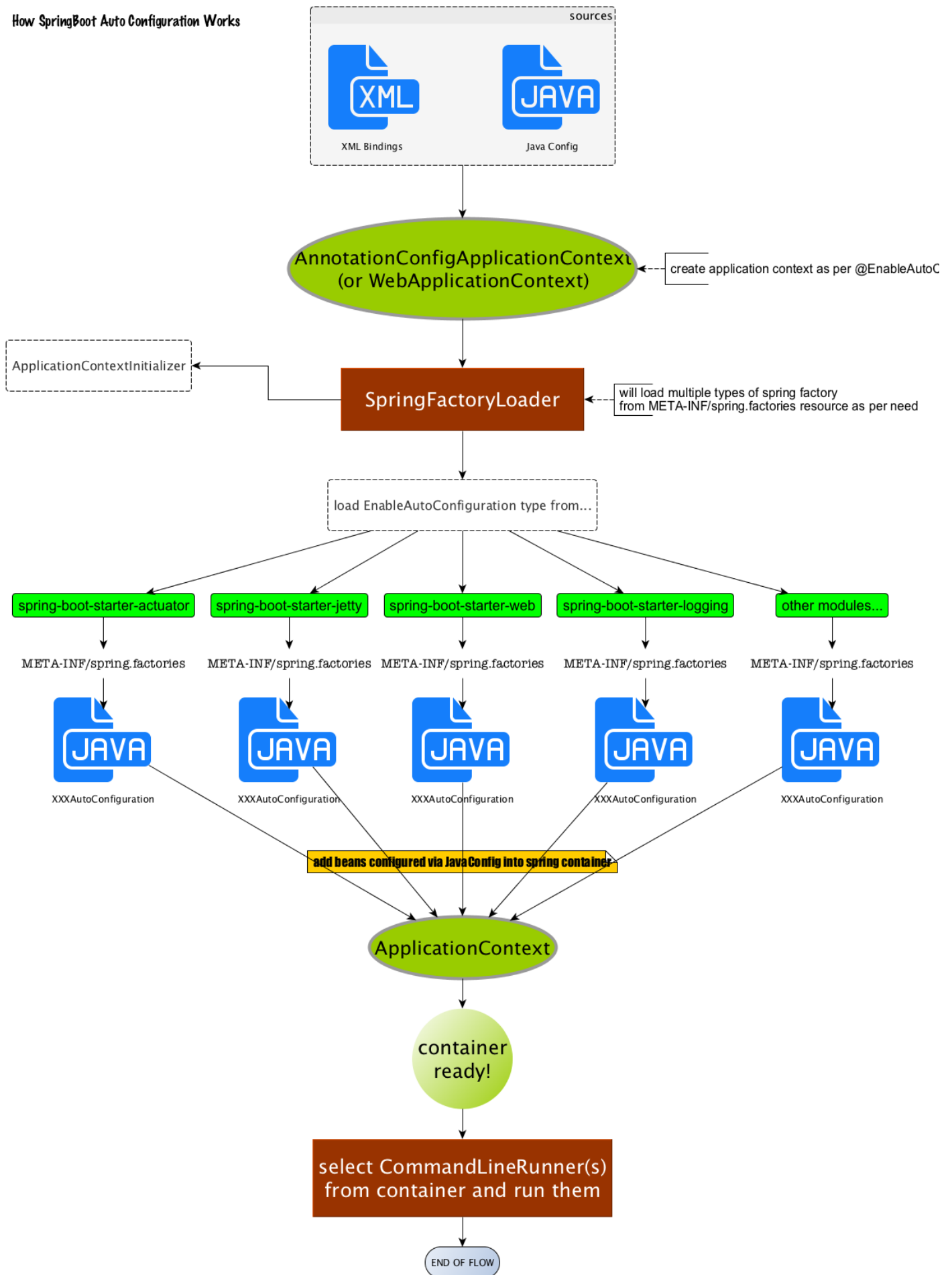
至此，我们大致可以了解。在全局配置属性如：server.port等，通过@ConfigurationProperties注解，绑定到对应的XxxxProperties配置实体类上封装bean，然后再通过@EnableConfigurationProperties注解导入到Spring容器中。

而诸多的XxxxAutoConfiguration自动配置类，就是Spring容器的JavaConfig形式，作用就是为Spring容器导入bean，而所有导入的bean所需要的属性通过XxxxProperties的bean来获得。

可能到目前为止还是有所疑惑，但面试的时候，其实远远不需要回答的这么具体，你只需要这样回答：

Spring Boot启动的时候会通过@EnableAutoConfiguration注解找到META-INF/spring.factories配置文件中的所有自动配置类，并对其加载，而这些配置类都是以AutoConfiguration结尾来命名的，它实际上就是一个JavaConfig形式的Spring容器配置类，它能够通过以Properties结尾命名的类中取得在配置文件中配置的属性如：server.port，而XxxxProperties类是通过@ConfigurationProperties注解与全局配置文件中对应的属性进行绑定的。

通过一张图来理解一下这一繁复的流程：



综上所述是对自动配置原理的讲解。当然，在浏览源码的时候一定要记得不要太过拘泥与代码的实现，而是应该抓住重点脉络。

一定要记得XxxxProperties类的含义是：封装配置文件中相关属性；XxxxAutoConfiguration类的含义是：自动配置类，目的是给容器中添加组件。

而其他的主方法启动，则是为了加载这些五花八门的XxxxAutoConfiguration类。