3、配置文件

2020 年 4 月 15 日
21:07

- 全局配置文件(properties、yml)
  - Application.properties
  - Application.yml
- YMAL 语法
  - 1、基本语法
    - K:(空格)v:表示一对键值对（空格必须有）
    - 空格控制层级关系；只要是左对齐的一列数据，都是同一个层级

      ```
      server:
        port: 8080
      ```

      属性和值对大小写敏感
  - 2、值的写法
    - 字面量：普通的值（数字，字符串，布尔）
      - 字符串不用加上引号

        "": 双引号；不会转义字符串里面的特殊字符；特殊字符会作为本身想表示的意思

        name: "zhangsan \n lisi"：输出；zhangsan 换行 lisi

        '': 单引号；会转义特殊字符，特殊字符最终只是一个普通的字符串数据

        name: 'zhangsan \n lisi'：输出；zhangsan \n lisi
    - 对象、Map（属性和值）（键值对）：
      - k: v：在下一行来写对象的属性和值的关系；注意缩进

        对象还是 k: v 的方式

        ```
        friends:
                lastName: zhangsan
                age: 20
        ```

      - 行内写法：

        ```
        friends: {lastName: zhangsan,age: 18}
        ```
    - 数组（List、Set）
      - 用- 值表示数组中的一个元素

```yaml
pets:
 - cat
 - dog
 - pig
```

        - 行内写法

          pets: [cat,dog,pig]

    - 3、配置文件值注入
        - 配置文件

```yaml
person:
    lastName: zhangsan
    age: 18
    boss:  false
    birth: 2020/4/15
    maps: {k1: v1,k2: v2}
    lists:
      - lisi
      - zhaoliu
      - wangwu
    dog:
      name: 小狗
      age: 2
```

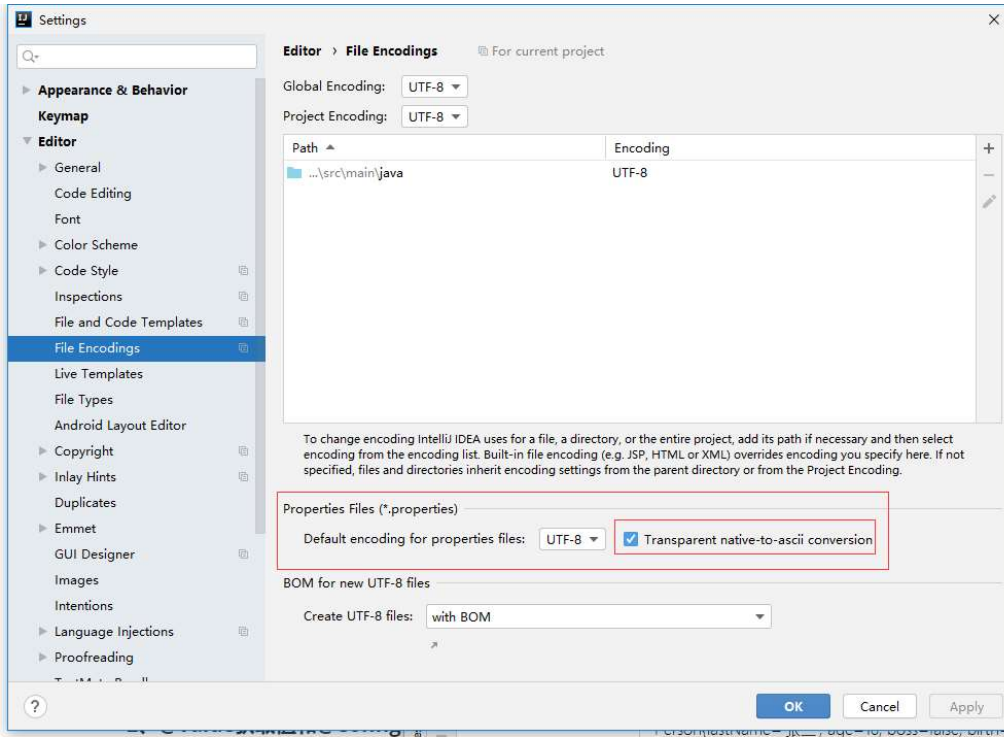
          JavaBean

```java
import org.springframework.stereotype.Component;

import java.util.Date;
import java.util.List;
import java.util.Map;
/*
 * 将配置文件中配置的每一个属性的值，映射到这个组件中
 * ConfigurationProperties:告诉springboot将本类中的所有属性和配置文件中相关配置进行绑定
 * prefix = "person"：配置文件中哪个下面的所有属性进行一一映射
 *
 * 只有这个组件是容器中的组件，才能使用容器提供ConfigurationProperties功能
 */
@Component
@ConfigurationProperties(prefix = "person")
public class Person {
    private String lastName;
    private Integer age;
    private Boolean boss;
    private Date birth;
    private Map<String,Object> maps;
    private List<Object> lists;
    private Dog dog;
```

        - 导入配置文件处理器，编写配置就会有提示

```
<!--导入配置文件处理器，配置文件进行绑定就会有提示-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-configuration-processor</artifactId>
    <optional>true</optional>
</dependency>
```

- 1、properties 配置文件在 idea 中默认 utf-8 可能会乱码
  - 

  - 在 properties 文件前加上
    - spring.http.encoding.enabled=true
- 2、@Value 获取值和@ConfigurationProperties 获取值比较

	@ConfigurationProperties	@Value
功能	批量注入配置文件中的属性	一个个指定
松散绑定（松散语法）	支持	不支持
SpEL	不支持	支持
JSR303 数据校验	支持	不支持
复杂类型封装	支持	不支持

配置文件 yml 还是 properties 他们都能获取到值；
如果说，我们只是在某个业务逻辑中需要获取一下配置文件中的某项值，使用@Value；
如果说，我们专门编写了一个 javaBean 来和配置文件进行映射，我们就直接使用 @ConfigurationProperties；

- 3、配置文件值注入值数据校验
  -

```java
@Component
@ConfigurationProperties(prefix = "person")
@Validated
public class Person {
//  @Value("${person.last-name}")
    private String lastName;
//  @Value("#{11*12}")
    private Integer age;
//  @Value("true")
    private Boolean boss;
    private Date birth;
    private Map<String,Object> maps;
    private List<Object> lists;
    private Dog dog;
```

- 4、@PropertySource&@ImportResource&@Bean
  - @PropertySource：加载指定的配置文件

```java
 * 只有这个组件是容器中的组件，才能使用容器提供ConfigurationPro
 **/
@PropertySource(value = {"classpath:person.properties"})
@Component
@ConfigurationProperties(prefix = "person")
//@Validated
public class Person {
//  @Value("${person.last-name}")
    private String lastName;
//  @Value("#{11*12}")
    private Integer age;
//  @Value("true")
    private Boolean boss;
    private Date birth;
    private Map<String,Object> maps;
    private List<Object> lists;
    private Dog dog;
```

- @ImportResource：
  导入 Spring 的配置文件，让配置文件里面的内容生效；
  Spring Boot 里面没有 Spring 的配置文件，我们自己编写的配置文件，也不能自动识别；
  想让 Spring 的配置文件生效，加载进来；
  @**ImportResource** 标注在一个配置类上

```
6
7   @ImportResource(value = {"classpath:beans.xml"})
8   //导入Spring的配置文件让其生效
```

- 不来编写 Spring 的配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean id="helloService" class="com.lhq.springboot.service.HelloService"></bean>
</beans>
```

- Spring Boot 推荐给容器中添加组件的方式；推荐使用全注解的方式
  - 1、配置类@Configuration——>Spring 配置文件
  - 2、使用@Bean 给容器中添加组件

```
/**
 * @Configuration：指明当前类是一个配置类；就是来代替之前的Spring配置文件
 *
 * 在配置文件中使用<bean></bean>标签来添加组件
 *
 */
@Configuration
public class MyAppConfig {
//  将方法的返回值添加到容器中；容器中这个组件默认id就是方法名
    @Bean
    public HelloService helloService(){
        System.out.println("给容器中添加组件了");
        return new HelloService();
    }
}
```

- 4、配置文件占位符
  - 1、随机数
    - ${random.int}、${random.value}、${random.long}
    - ${random.int(10)}、${random.int[1024,65536]}
  - 2、占位符获取之前配置的值，如果没有可以使用指定默认值

```
server.port=8080
```

```
person.last-name=${random.uuid}
person.age=${random.int}
person.birth=2020/4/15
person.boss=false
person.maps.k1=k1
person.maps.k2=k2
person.lists=a,b,c
person.dog.name=${person.hello:hello}dog
person.dog.age=15
```

- 5、Profile
  - 1、多 Profile 文件

    写主配置文件时，文件名可以是 application-{profile}.properties/yml

    eq：application-dev.properties

    application-prod.properties

    application-dev.yml

    application-prod.yml

    默认使用 application.properties 的配置
  - 2、yml 支持多文档块方式

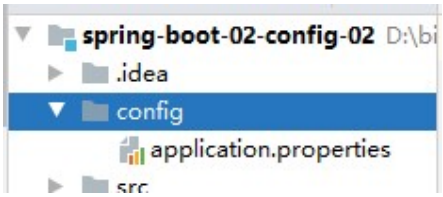
```
server:
port:8081
spring:
profiles:
active:prod
---
server:
port:8083
spring:
profiles:dev
---
server:
port:8084
spring:
profiles:prod
```

  - 3、激活指定 profile
    - 1、在配置文件中指定 spring.profiles.active=dev

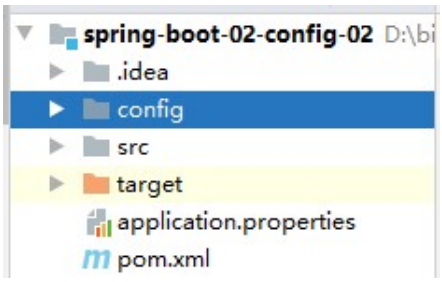
- 2、命令行：--spring.profiles.active=dev
  - Program arguments: --spring.profiles.active=dev
  - C:\Users\李浩琦>java -jar spring-boot-02-config.jar --spring.profiles.activ
  - 可以直接在测试的时候，配置传入命令行参数
- 3、虚拟机参数：-Dspring.profiles.active=dev
  - VM options: -Dspring.profiles.active=dev

- 6、配置文件加载位置
  - springboot 启动会扫描以下位置的 application.properties 或者 application.yml 文件作为 springboot 的配置文件

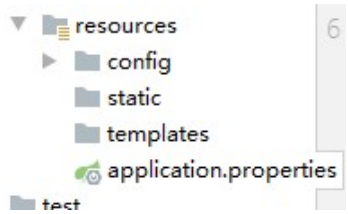
    -file:./config

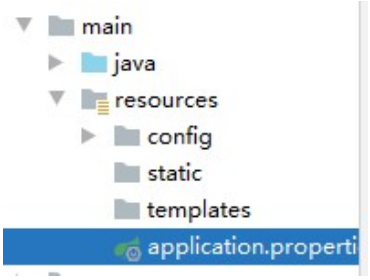
    -file:./

    -classpath:/config/

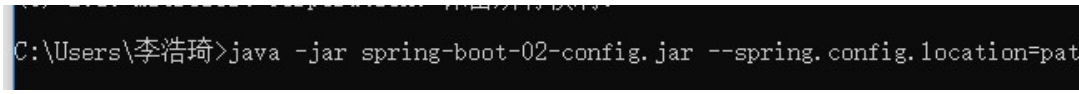
    

    -classpath:/

优先级由高到低，高优先级配置会覆盖低优先级配置；
springboot 会从这四个位置全部加载主配置文件；互补配置；

还可以通过 spring.config.location 来改变默认的配置文件位置
项目打包好以后，可以使用命令行参数的形式，启动项目的时候来指定配置文件的新位置；指定配置文件和默认加载的这些配置文件共同起作用形成互补配置。



- 7、外部配置加载顺序
  SpringBoot 也可以从以下位置加载配置； 优先级从高到低；高优先级的配置覆盖低优先级的配置，所有的配置会形成互补配置
  - 1、命令行参数
    所有的配置都可以在命令行上进行指定
    java -jar spring-boot-02-config-02-0.0.1-SNAPSHOT.jar --server.port=8087 --server.context-path=/abc
    多个配置用空格分开； --配置项=值
  - 2、来自 java:comp/env 的 JNDI 属性
  - 3、Java 系统属性（System.getProperties()）
  - 4、操作系统环境变量
  - 5、RandomValuePropertySource 配置的 random.*属性值

  由 jar 包外向 jar 包内进行寻找；
  优先加载带 profile
  - 6、jar 包外部的 application-{profile}.properties 或 application.yml(带 spring.profile)配置文件
  - 7、jar 包内部的 application-{profile}.properties 或 application.yml(带 spring.profile)配置文件

  再来加载不带 profile
  - 8、jar 包外部的 application.properties 或 application.yml(不带 spring.profile)配置文件
  - 9、jar 包内部的 application.properties 或 application.yml(不带 spring.profile)配置文件
  - 10、@Configuration 注解类上的@PropertySource
  - 11、通过 SpringApplication.setDefaultProperties 指定的默认属性

  所有支持的配置加载来源；
  参考官方文档
- 8、自动配置原理

配置文件能配置的属性参照

- 1、自动配置原理：

  1）、SpringBoot 启动的时候加载主配置类，开启了自动配置功能 ==@EnableAutoConfiguration==

  2）、@EnableAutoConfiguration 作用：

  - 利用 AutoConfigurationImportSelector 给容器中导入一些组件?
  - 可以查看 selectImports()方法的内容；
  - List<String> configurations = getCandidateConfigurations(annotationMetadata, attributes);获取候选的配置
    - SpringFactoriesLoader.loadFactoryNames()

      扫描所有 jar 包类路径下 META-INF/spring.factories

      把扫描到的这些文件的内容包装成 properties 对象

      从 properties 中获取到 EnableAutoConfiguration.class 类（类名）对应的值，然后把他们添加在容器中

  - 将 类路径下 META-INF/spring.factories 里面配置的所有 EnableAutoConfiguration 的值加入到了容器中；

```
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
org.springframework.boot.autoconfigure.admin.SpringApplicationAdminJmxAutoConfiguration,\
org.springframework.boot.autoconfigure.aop.AopAutoConfiguration,\
org.springframework.boot.autoconfigure.amqp.RabbitAutoConfiguration,\
org.springframework.boot.autoconfigure.batch.BatchAutoConfiguration,\
org.springframework.boot.autoconfigure.cache.CacheAutoConfiguration,\
org.springframework.boot.autoconfigure.cassandra.CassandraAutoConfiguration,\
org.springframework.boot.autoconfigure.cloud.CloudServiceConnectorsAutoConfiguration,\
org.springframework.boot.autoconfigure.context.ConfigurationPropertiesAutoConfiguration,\
org.springframework.boot.autoconfigure.context.MessageSourceAutoConfiguration,\
```

org.springframework.boot.autoconfigure.context.Property
PlaceholderAutoConfiguration,\
org.springframework.boot.autoconfigure.couchbase.Couc
hbaseAutoConfiguration,\
org.springframework.boot.autoconfigure.dao.Persistence
ExceptionTranslationAutoConfiguration,\
org.springframework.boot.autoconfigure.data.cassandra.C
assandraDataAutoConfiguration,\
org.springframework.boot.autoconfigure.data.cassandra.C
assandraReactiveDataAutoConfiguration,\
org.springframework.boot.autoconfigure.data.cassandra.C
assandraReactiveRepositoriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.cassandra.C
assandraRepositoriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.couchbase.
CouchbaseDataAutoConfiguration,\
org.springframework.boot.autoconfigure.data.couchbase.
CouchbaseReactiveDataAutoConfiguration,\
org.springframework.boot.autoconfigure.data.couchbase.
CouchbaseReactiveRepositoriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.couchbase.
CouchbaseRepositoriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.elasticsearc
h.ElasticsearchAutoConfiguration,\
org.springframework.boot.autoconfigure.data.elasticsearc
h.ElasticsearchDataAutoConfiguration,\
org.springframework.boot.autoconfigure.data.elasticsearc
h.ElasticsearchRepositoriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.elasticsearc
h.ReactiveElasticsearchRepositoriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.elasticsearc
h.ReactiveRestClientAutoConfiguration,\
org.springframework.boot.autoconfigure.data.jdbc.JdbcRe
positoriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.jpa.JpaRepo
sitoriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.ldap.LdapR
epositoriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.mongo.Mon
goDataAutoConfiguration,\
org.springframework.boot.autoconfigure.data.mongo.Mon
goReactiveDataAutoConfiguration,\
org.springframework.boot.autoconfigure.data.mongo.Mon
goReactiveRepositoriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.mongo.Mon
goRepositoriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.neo4j.Neo4j
DataAutoConfiguration,\
org.springframework.boot.autoconfigure.data.neo4j.Neo4j
RepositoriesAutoConfiguration,\

```
org.springframework.boot.autoconfigure.data.solr.SolrRe
positoriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.redis.Redis
AutoConfiguration,\
org.springframework.boot.autoconfigure.data.redis.Redis
ReactiveAutoConfiguration,\
org.springframework.boot.autoconfigure.data.redis.Redis
RepositoriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.rest.Reposit
oryRestMvcAutoConfiguration,\
org.springframework.boot.autoconfigure.data.web.Spring
DataWebAutoConfiguration,\
org.springframework.boot.autoconfigure.elasticsearch.jes
t.JestAutoConfiguration,\
org.springframework.boot.autoconfigure.elasticsearch.res
t.RestClientAutoConfiguration,\
org.springframework.boot.autoconfigure.flyway.FlywayA
utoConfiguration,\
org.springframework.boot.autoconfigure.freemarker.Free
MarkerAutoConfiguration,\
org.springframework.boot.autoconfigure.gson.GsonAutoC
onfiguration,\
org.springframework.boot.autoconfigure.h2.H2ConsoleAu
toConfiguration,\
org.springframework.boot.autoconfigure.hateoas.Hyperm
ediaAutoConfiguration,\
org.springframework.boot.autoconfigure.hazelcast.Hazelc
astAutoConfiguration,\
org.springframework.boot.autoconfigure.hazelcast.Hazelc
astJpaDependencyAutoConfiguration,\
org.springframework.boot.autoconfigure.http.HttpMessag
eConvertersAutoConfiguration,\
org.springframework.boot.autoconfigure.http.codec.Codec
sAutoConfiguration,\
org.springframework.boot.autoconfigure.influx.InfluxDbA
utoConfiguration,\
org.springframework.boot.autoconfigure.info.ProjectInfo
AutoConfiguration,\
org.springframework.boot.autoconfigure.integration.Integ
rationAutoConfiguration,\
org.springframework.boot.autoconfigure.jackson.Jackson
AutoConfiguration,\
org.springframework.boot.autoconfigure.jdbc.DataSource
AutoConfiguration,\
org.springframework.boot.autoconfigure.jdbc.JdbcTempla
teAutoConfiguration,\
org.springframework.boot.autoconfigure.jdbc.JndiDataSo
urceAutoConfiguration,\
org.springframework.boot.autoconfigure.jdbc.XADataSour
ceAutoConfiguration,\
```

```
org.springframework.boot.autoconfigure.jdbc.DataSource
TransactionManagerAutoConfiguration,\
org.springframework.boot.autoconfigure.jms.JmsAutoCon
figuration,\
org.springframework.boot.autoconfigure.jmx.JmxAutoCon
figuration,\
org.springframework.boot.autoconfigure.jms.JndiConnecti
onFactoryAutoConfiguration,\
org.springframework.boot.autoconfigure.jms.activemq.Ac
tiveMQAutoConfiguration,\
org.springframework.boot.autoconfigure.jms.artemis.Arte
misAutoConfiguration,\
org.springframework.boot.autoconfigure.groovy.template.
GroovyTemplateAutoConfiguration,\
org.springframework.boot.autoconfigure.jersey.JerseyAut
oConfiguration,\
org.springframework.boot.autoconfigure.jooq.JooqAutoCo
nfiguration,\
org.springframework.boot.autoconfigure.jsonb.JsonbAuto
Configuration,\
org.springframework.boot.autoconfigure.kafka.KafkaAuto
Configuration,\
org.springframework.boot.autoconfigure.ldap.embedded.
EmbeddedLdapAutoConfiguration,\
org.springframework.boot.autoconfigure.ldap.LdapAutoC
onfiguration,\
org.springframework.boot.autoconfigure.liquibase.Liquib
aseAutoConfiguration,\
org.springframework.boot.autoconfigure.mail.MailSender
AutoConfiguration,\
org.springframework.boot.autoconfigure.mail.MailSender
ValidatorAutoConfiguration,\
org.springframework.boot.autoconfigure.mongo.embedde
d.EmbeddedMongoAutoConfiguration,\
org.springframework.boot.autoconfigure.mongo.MongoA
utoConfiguration,\
org.springframework.boot.autoconfigure.mongo.MongoRe
activeAutoConfiguration,\
org.springframework.boot.autoconfigure.mustache.Musta
cheAutoConfiguration,\
org.springframework.boot.autoconfigure.orm.jpa.Hiberna
teJpaAutoConfiguration,\
org.springframework.boot.autoconfigure.quartz.QuartzAu
toConfiguration,\
org.springframework.boot.autoconfigure.rsocket.RSocket
MessagingAutoConfiguration,\
org.springframework.boot.autoconfigure.rsocket.RSocket
RequesterAutoConfiguration,\
org.springframework.boot.autoconfigure.rsocket.RSocket
ServerAutoConfiguration,\
```

```
org.springframework.boot.autoconfigure.rsocket.RSocket
StrategiesAutoConfiguration,\
org.springframework.boot.autoconfigure.security.servlet.
SecurityAutoConfiguration,\
org.springframework.boot.autoconfigure.security.servlet.
UserDetailsServiceAutoConfiguration,\
org.springframework.boot.autoconfigure.security.servlet.
SecurityFilterAutoConfiguration,\
org.springframework.boot.autoconfigure.security.reactive
.ReactiveSecurityAutoConfiguration,\
org.springframework.boot.autoconfigure.security.reactive
.ReactiveUserDetailsServiceAutoConfiguration,\
org.springframework.boot.autoconfigure.security.rsocket.
RSocketSecurityAutoConfiguration,\
org.springframework.boot.autoconfigure.security.saml2.S
aml2RelyingPartyAutoConfiguration,\
org.springframework.boot.autoconfigure.sendgrid.SendGr
idAutoConfiguration,\
org.springframework.boot.autoconfigure.session.SessionA
utoConfiguration,\
org.springframework.boot.autoconfigure.security.oauth2.
client.servlet.OAuth2ClientAutoConfiguration,\
org.springframework.boot.autoconfigure.security.oauth2.
client.reactive.ReactiveOAuth2ClientAutoConfiguration,\
org.springframework.boot.autoconfigure.security.oauth2.
resource.servlet.OAuth2ResourceServerAutoConfiguratio
n,\
org.springframework.boot.autoconfigure.security.oauth2.
resource.reactive.ReactiveOAuth2ResourceServerAutoCo
nfiguration,\
org.springframework.boot.autoconfigure.solr.SolrAutoCo
nfiguration,\
org.springframework.boot.autoconfigure.task.TaskExecuti
onAutoConfiguration,\
org.springframework.boot.autoconfigure.task.TaskSchedu
lingAutoConfiguration,\
org.springframework.boot.autoconfigure.thymeleaf.Thym
eleafAutoConfiguration,\
org.springframework.boot.autoconfigure.transaction.Tran
sactionAutoConfiguration,\
org.springframework.boot.autoconfigure.transaction.jta.Jt
aAutoConfiguration,\
org.springframework.boot.autoconfigure.validation.Valida
tionAutoConfiguration,\
org.springframework.boot.autoconfigure.web.client.RestT
emplateAutoConfiguration,\
org.springframework.boot.autoconfigure.web.embedded.
EmbeddedWebServerFactoryCustomizerAutoConfiguratio
n,\
org.springframework.boot.autoconfigure.web.reactive.Htt
pHandlerAutoConfiguration,\
```

```
org.springframework.boot.autoconfigure.web.reactive.ReactiveWebServerFactoryAutoConfiguration,\
org.springframework.boot.autoconfigure.web.reactive.WebFluxAutoConfiguration,\
org.springframework.boot.autoconfigure.web.reactive.error.ErrorWebFluxAutoConfiguration,\
org.springframework.boot.autoconfigure.web.reactive.function.client.ClientHttpConnectorAutoConfiguration,\
org.springframework.boot.autoconfigure.web.reactive.function.client.WebClientAutoConfiguration,\
org.springframework.boot.autoconfigure.web.servlet.DispatcherServletAutoConfiguration,\
org.springframework.boot.autoconfigure.web.servlet.ServletWebServerFactoryAutoConfiguration,\
org.springframework.boot.autoconfigure.web.servlet.error.ErrorMvcAutoConfiguration,\
org.springframework.boot.autoconfigure.web.servlet.HttpEncodingAutoConfiguration,\
org.springframework.boot.autoconfigure.web.servlet.MultipartAutoConfiguration,\
org.springframework.boot.autoconfigure.web.servlet.WebMvcAutoConfiguration,\
org.springframework.boot.autoconfigure.websocket.reactive.WebSocketReactiveAutoConfiguration,\
org.springframework.boot.autoconfigure.websocket.servlet.WebSocketServletAutoConfiguration,\
org.springframework.boot.autoconfigure.websocket.servlet.WebSocketMessagingAutoConfiguration,\
org.springframework.boot.autoconfigure.webservices.WebServicesAutoConfiguration,\
org.springframework.boot.autoconfigure.webservices.client.WebServiceTemplateAutoConfiguration
```

- 每一个这样的 xxxAutoConfiguration 类都是容器中的一个组件，都加入到容器中；用他们来做自动配置；

3）、每一个自动配置类进行自动配置功能；

4）、以 **HttpEncodingAutoConfiguration（Http 编码自动配置）**为例解释自动配置原理；

```
//表示这是一个配置类，以前编写的配置文件一样，也可以给容器中添加组件
@Configuration(
proxyBeanMethods=false
)
//启动指定类的 ConfigurationProperties 功能；将配置文件中对应的值和 HttpProperties 绑定起来；并把 HttpProperties 加入到 ioc 容器中
@EnableConfigurationProperties({HttpProperties.class})
```

```
//Spring 底层@Conditional 注解（Spring 注解版），根据不同的条件，如果满足指定的条件，整个配置类里面的配置就会生效；  判断当前应用是否是 web 应用，如果是，当前配置类生效
@ConditionalOnWebApplication(
type=Type.SERVLET
)
//判断当前项目有没有这个类 CharacterEncodingFilter；SpringMVC 中进行乱码解决的过滤器；
@ConditionalOnClass({CharacterEncodingFilter.class})
//判断配置文件中是否存在某个配置 spring.http.encoding.enabled；如果不存在，判断也是成立的
//即使我们配置文件中不配置
@ConditionalOnProperty(
prefix="spring.http.encoding",
value={"enabled"},
matchIfMissing=true
)
//他已经和 SpringBoot 的配置文件映射了
publicclassHttpEncodingAutoConfiguration{
privatefinalEncodingproperties;

publicHttpEncodingAutoConfiguration(HttpPropertiespro
perties){
this.properties=properties.getEncoding();
}

@Bean//给容器中添加一个组件，这个组件的某些值需要从 properties 中获取
@ConditionalOnMissingBean//判断容器没有这个组件
publicCharacterEncodingFiltercharacterEncodingFilter(){
CharacterEncodingFilterfilter=newOrderedCharacterEnco
dingFilter();
filter.setEncoding(this.properties.getCharset().name());
filter.setForceRequestEncoding(this.properties.shouldFor
ce(org.springframework.boot.autoconfigure.http.HttpPro
perties.Encoding.Type.REQUEST));
filter.setForceResponseEncoding(this.properties.shouldFo
rce(org.springframework.boot.autoconfigure.http.HttpPro
perties.Encoding.Type.RESPONSE));
returnfilter;
}
```

根据当前不同的条件判断，决定这个配置类是否生效?

一但这个配置类生效；这个配置类就会给容器中添加各种组件；这些组件的属性是从对应的 properties 类中获取的，这些类里面的每一个属性又是和配置文件绑定的；

5）、所有在配置文件中能配置的属性都是在 xxxxProperties 类中封装着；配置文件能配置什么就可以参照某个功能对应的这个属性类

```
@ConfigurationProperties(
prefix="spring.http")//从配置文件中获取指定的值和 bean 的属性进行绑定
Public Encoding(){
this.charset=DEFAULT_CHARSET;
}
```

精髓：

1）、SpringBoot 启动会加载大量的自动配置类

2）、我们看我们需要的功能有没有 SpringBoot 默认写好的自动配置类；

3）、我们再来看这个自动配置类中到底配置了哪些组件；（只要我们要用的组件有，我们就不需要再来配置了）

4）、给容器中自动配置类添加组件的时候，会从 properties 类中获取某些属性。我们就可以在配置文件中指定这些属性的值；

xxxxAutoConfigurartion：自动配置类；

给容器中添加组件

xxxxProperties:封装配置文件中相关属性；

- 2、细节
  - 1、@Conditional 派生注解（Spring 注解版原生的 @Conditional 作用）

    作用：必须是@Conditional 指定的条件成立，才给容器中添加组件，配置配里面的所有内容才生效；

@Conditional 扩展注解	**作用（判断是否满足当前指定条件）**
@ConditionalOnJava	系统的 java 版本是否符合要求
@ConditionalOnBean	容器中存在指定 Bean；

@ConditionalOnMissingBean	容器中不存在指定 Bean；
@ConditionalOnExpression	满足 SpEL 表达式指定
@ConditionalOnClass	系统中有指定的类
@ConditionalOnMissingClass	系统中没有指定的类
@ConditionalOnSingleCandidate	容器中只有一个指定的 Bean，或者这个 Bean 是首选 Bean
@ConditionalOnProperty	系统中指定的属性是否有指定的值
@ConditionalOnResource	类路径下是否存在指定资源文件
@ConditionalOnWebApplication	当前是 web 环境
@ConditionalOnNotWebApplication	当前不是 web 环境
@ConditionalOnJndi	JNDI 存在指定项

自动配置类必须在一定的条件下才能生效；
我们怎么知道哪些自动配置类生效；
==我们可以通过启用 debug=true 属性；来让控制台打印自动配置报告==，这样我们就可以很方便的知道哪些自动配置类生效；