2.Web相关配置

2.1 Spring Boot提供的自动配置

通过查看WebMvcAutoConfiguration及WebMvcProperties的源码,可以发现Spring Boot为我们提供了如下的自动配置.

1.自动配置ViewResolver

(1) ContentNegoTiatingViewResolver

这是Spring MVC提供的一个特殊的ViewResolver,ContentNegotiatingViewResolver不是自己处理View,而是代理给不同的ViewResolver来处理不同的View,所以它有最高的优先级.

(2)BeanNameViewResolver

在控制器(@Controller)中的一个方法返回值的字符串会根据BeanNameViewResolver去查找Bean的名称为返回字符串的View来渲染视图.

2.自动配置静态资源

(1) 类路径文件

把类路径下的/static,/public,/resources,和/META-INF/resources文件下的静态文件直接映射为/**,可以通过http://localhost:8080/**来访问.

(2) webjar

webjar,就是将我们常用的脚本框架封装在jar包中的jar,把webjar的/META-INF/resources/webjars/下的静态文件映射为/webjar/**,可以通过http://localhost:8080/webjar/**访问

(3)自动配置Formatter和Converter

关于自动配置Formatter和Converter,我们可以看一下WebMvcAutoConfiguration类中的定义:

```
public void addFormatters(FormatterRegistry registry) {
   Iterator var2 = this.getBeansOfType(Converter.class).iterator();

   while(var2.hasNext()) {
      Converter<?, ?> converter = (Converter)var2.next();
      registry.addConverter(converter);
   }

   var2 = this.getBeansOfType(GenericConverter.class).iterator();

   while(var2.hasNext()) {
      GenericConverter converter = (GenericConverter)var2.next();
      registry.addConverter(converter);
   }
```

```
    var2 = this.getBeansOfType(Formatter.class).iterator();

    while(var2.hasNext()) {
        Formatter<?> formatter = (Formatter)var2.next();
        registry.addFormatter(formatter);
    }

}
```

从代码中可以看出,只要我们定义了Converter,GenericConverter和Formatter接口的实现类的Bean,这些Bean就会自动注册到Spring MVC中.

(4) 自动配置的HttpMessageConverters

在WebMvcAutoConfiguration中,我们注册了messageConverters,代码如下:

```
public void configureMessageConverters(List<HttpMessageConverter<?>>
converters) {
    this.messageConvertersProvider.ifAvailable((customConverters) -> {
        converters.addAll(customConverters.getConverters());
    });
}
```

我们自动注册的HttpMessageConverter,包括Spring MVC默认的HttpMessageConverter,在我们的HttpMessageConvertersAutoConfiguration的自动配置文件里还引入了JacksonHttpMessageConvertersConfiguration和GsonHttpMessageConverterConfiguration,使我们获得了额外的HttpMessageConverter:

若jackson的jar包在类路径上,则Spring Boot通过JacksonHttpMessageConvertersConfiguration增加MappingJackson2HttpMessageConverter和MappingJackson2XmlHttpMessageConverter

若gson的jar包在类路径上,则Spring Boot通过GsonHttpMessageConverterConfiguration增加GsonHttpMessageConverter.

在Spring Boot中如果需要新增加自定义的HttpMessageConverter,则只需要定义一个自己的HttpMessageConverters的Bean,然后在此Bean中注册自定义HttpMessageConverter即可:

```
@Bean
public HttpMessageConverters customConverters(){
    HttpMessageConverter<?> customConverter1 = new CustomConverter1();
    HttpMessageConverter<?> customConverter2 = new CustomConverter2();
    return new HttpMessageConverters(customConverter1,customConverter2);
}
```

(5) 静态首页的支持

把静态index.html文件放置在如下目录:

classpath:/META-INF/resources/index.html

classpath:/resources/index.html

classpath:/static/index.html

classpath:/public/index.html

当我们访问应用根路径http://localhost:8080/时,会直接映射

2.2 接管Spring Boot的Web配置

如果Spring Boot提供的Spring MVC默认配置不符合你的需求,则可以通过一个配置类(@Configuration)加上@EnableWebMvc注解来实现自己控制的MVC配置.

当你既需要保留Spring Boot提供的便利,又需要增加自己的额外的配置的时候,可以定义一个配置类并继承WebMvcConfigurerAdapter(Spring5中不建议使用),新的实现时接口WebMvcConfigurer 和类WebMvcConfigurationSupport.无需使用@EnableWebMvc注解,然后按第4章讲解的Spring MVC的配置方法来添加Spring Boot为我们所做的其他配置

2.3 注册Servlet,Filter,Listener

当使用嵌入式的Servlet容器(Tomcat,Jetty等)时,我们通过将Servlet,Filter和Listener声明为Spring Bean而达到注册的效果;或者注册ServletRegistrationBean,FilterRegistrationBean和ServletListenerRegistrationBean的Bean.

(1) 直接注册Bean:

```
@Bean
public xxServlet xxSesrvlet(){
   return new xxServlet();
}
@Bean
public xxFilter xxFilter(){
   return new xxFilter();
}
@Bean
public xxListener xxListener(){
   return new xxListener();
}
```

(2)通过RegistrationBean示例:

```
@Bean
public ServletRegistrationBean servletRegistrationBean(){
   return new ServletRegistrationBean(new xxServlet(),"/xx/*")
}
@Bean
public FilterRegistrationBean servletRegistrationBean(){
   FilterRegistrationBean registrationBean = new FilterRegistrationBean();
   registrationBean.setFilter(new xxFilter);
   registrationBean.setOrder(2);
   return registrationBean;
```

```
}
@Bean
public ServletListenerRegistrationBean servletRegistrationBean(){
    return new ServletListenerRegistrationBean<xxListener>(
        new xxFilter())
}
```