# 不使用spring-boot-starter-parent

在开发第一个Spring Boot程序时，在pom.xml文件中添加了spring-boot-starter

-parent，它主要提供了如下的默认配置：

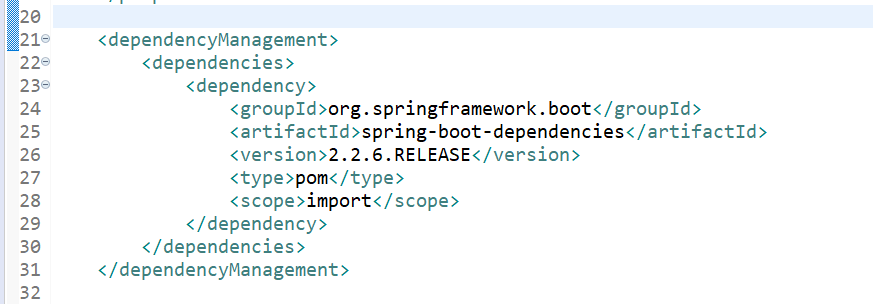
·Java版本默认使用1.8

·编码默认使用UTF-8

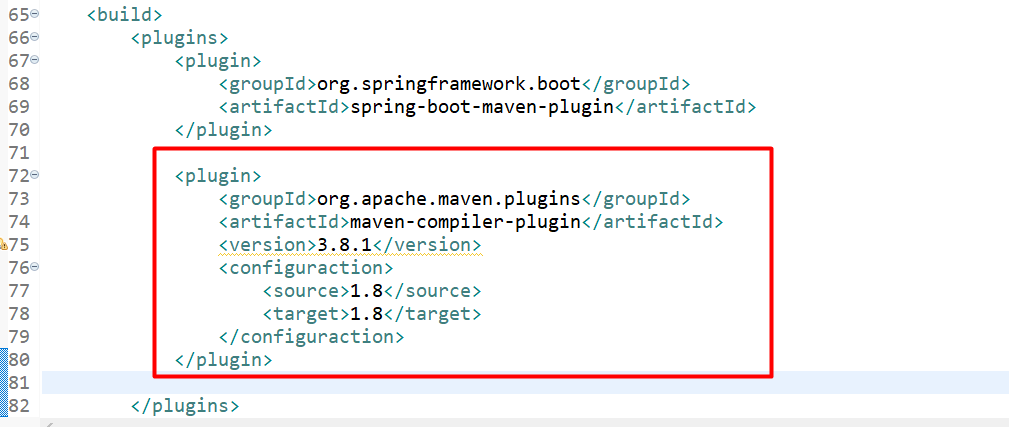
·提供Dependency Management进行项目依赖的版本管理

·默认的资源过滤与插件配置

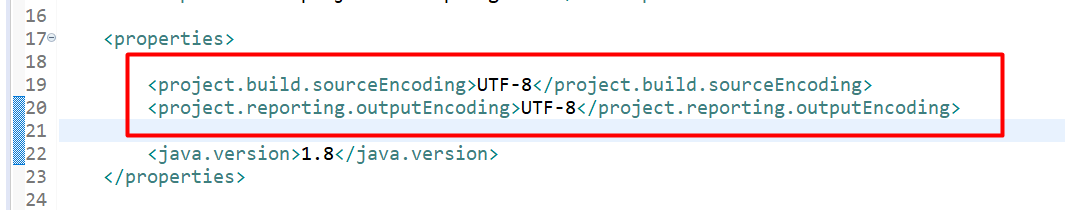
spring-boot-starter-parent虽然很方便，但是公司在开发微服务项目或者多模块项目时一般需要使用公司自己的parent，这个时候如果还想进行项目的依赖版本的统一管理，就需要使用dependencyManagement来实现。代码如下：



此时，就可以不用继承spring-boot-starter-parent了，但是Java的版本，编码格式等都需要开发者手动配置，Java版本的配置很简单，添加一个plugin即可。代码如下：



至于编码格式，代码如下：

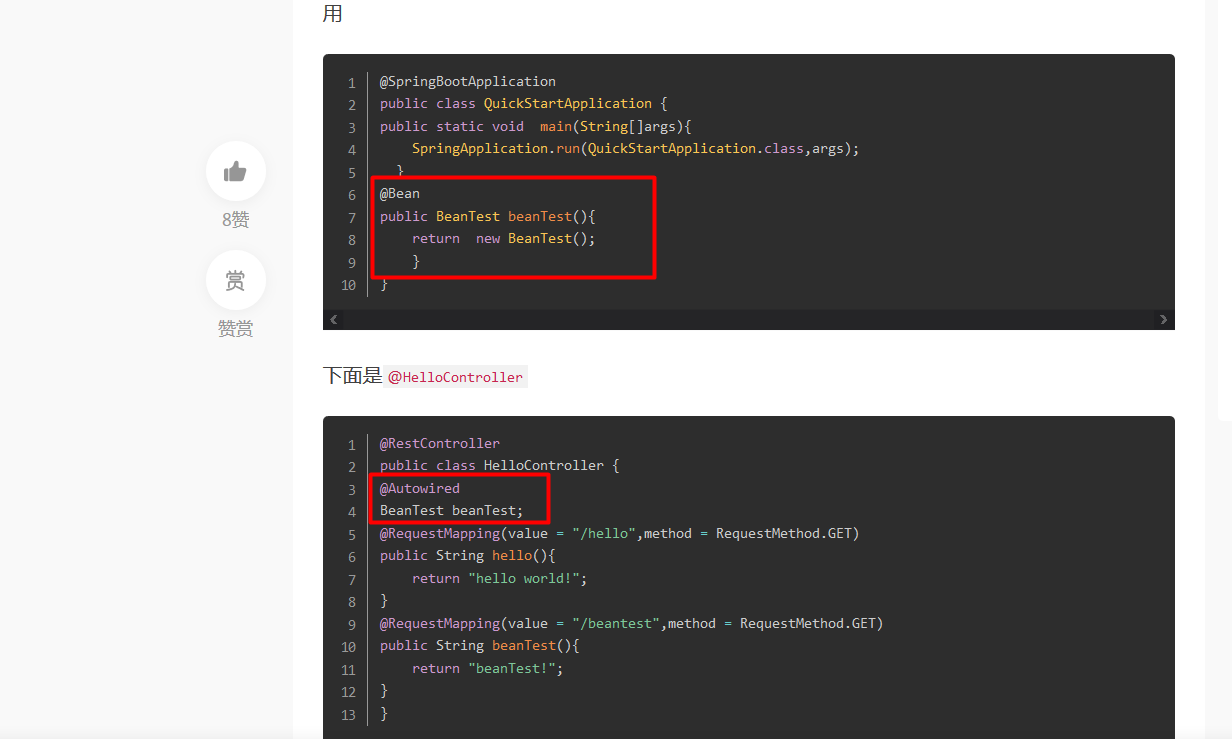


# @SpringBootApplication

通过前面的学习可以知道，@SpringBootApplication注解是标注在项目的启动类上的，它是一个组合注解，由3个注解组成：

1）@SpringBootConfiguration

该注解是@Configuration注解的子注解，所以@SpringBootConfiguration注解的功能就是声明一个配置类，开发者可以将该注解标注在一个类上，并在这个类中配置Bean。



2）@EnableAutoConfiguration

该注解表示开启自动化配置，是Spring Boot实现自动化配置的核心注解。Spring Boot中的自动化配置是非侵入式的，在任意时刻，开发者都可以使用自定义配置替代自动化配置中的某一项配置。

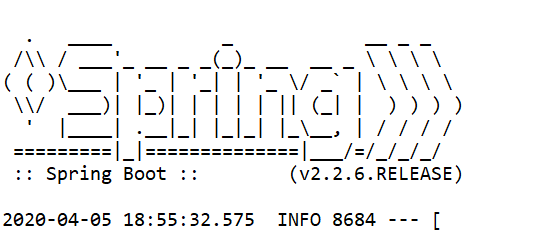
3）@ComponentScan

该注解用于完成包扫描，作用与<context:component-scan>标签相同。由于扫描包也会对其子包进行扫描，因此建议在项目开发中把项目启动类放在根包中。如下：



# 定制banner

Spring Boot项目启动时会打印一个banner，如下：



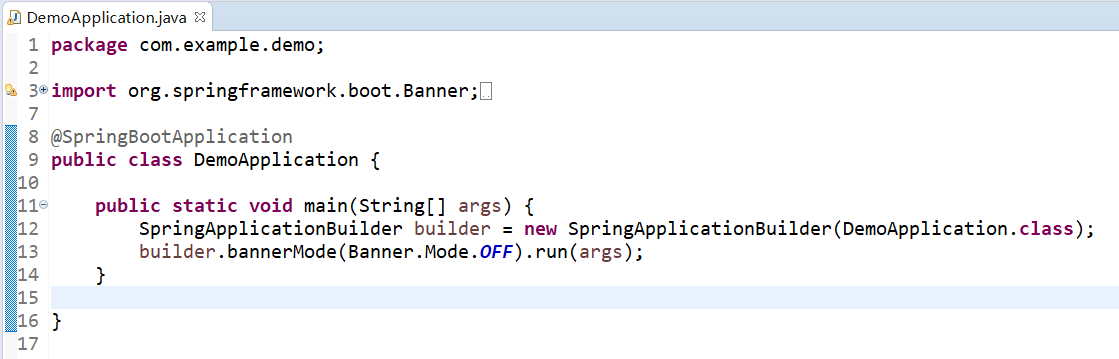
1. 这个banner是可以定制的，在resources目录下创建一个banner.txt文件，在文件中写入的文本将在项目启动时打印出来。如果想让打印的文本变成艺术字体，则可以从以下网站进行转换

<http://patorjk.com/software/taag>

<http://www.network-science.de/ascii/>

然后将生成的艺术字粘贴到banner.txt文件中。

1. 想关闭banner也是可以的，修改项目启动类的main方法，代码如下：



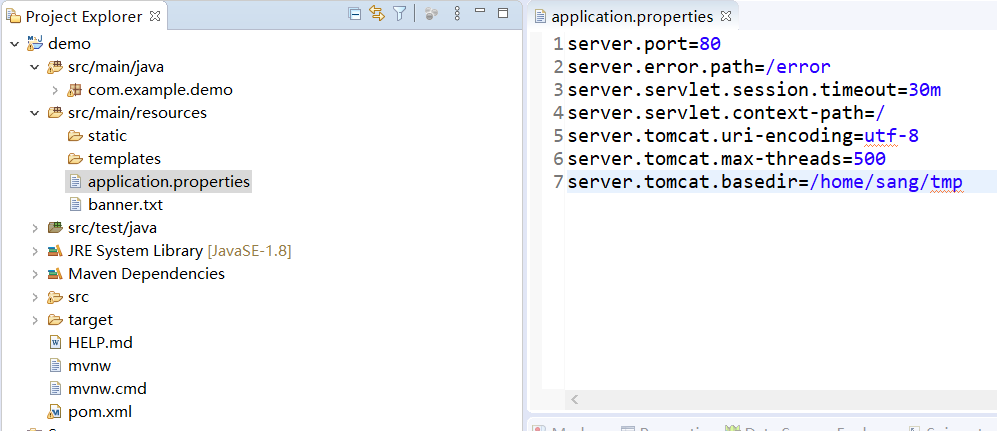
# Web容器配置

在Spring Boot项目中，可以内置Tomcat、Jetty、Undertow、Netty等容器。默认内置的是Tomcat容器。

## Tomcat配置

### 常规配置

当开发者添加spring-boot-starter-web依赖后，默认会使用Tomcat作为Spring Boot的Web容器。如果需要对Tomcat做进一步的配置，可以在application.properties中进行配置，代码如下：



·port配置了容器的端口号，默认为8080；

·error.path配置了当前项目出错时跳转去的页面；

·session.timeout配置了session失效时间，30m表示30分钟，默认为30分钟；

如果不指定m，则以秒为单位；

·context-path表示项目名称，不配置时默认为/，如果配置了，就要在访问路径中

加上配置的路径；

·uri-encoding表示配置Tomcat请求编码；

·max-threads表示Tomcat最大线程数；

·basedir是一个存放Tomcat运行日志和临时文件的目录，若不配置，则默认使用

系统的临时目录；

当然，Web容器的配置远不止这些，以上只是常见的配置，完整的配置可以参考官方文档Appendix A . Common application properties一节。

### HTTPS配置

由于HTTPS具有良好的安全性，在开发中得到了越来越广泛的应用，像微信公众号、小程序等的开发都要使用HTTPS来完成。

对于个人开发者而言，一个HTTPS证书的价格还是有点贵，国内一些云服务器厂商提供免费的HTTPS证书，一个账号可以申请数个。

不过在jdk中提供了一个Java数字证书管理工具keytool，在\jdk\bin目录下，通过这个工具可以自己生成一个数字证书，生成命令如下：

**keytool -genkey -alias tomcathttps -keyalg PSA -keysize 2048 -keystore sang.p12 -validity 365**

·-genkey表示要创建一个新的密匙；

·-alias表示keystore的别名；

·-keyalg表示使用的是加密算法RSA，一种非对称加密算法；

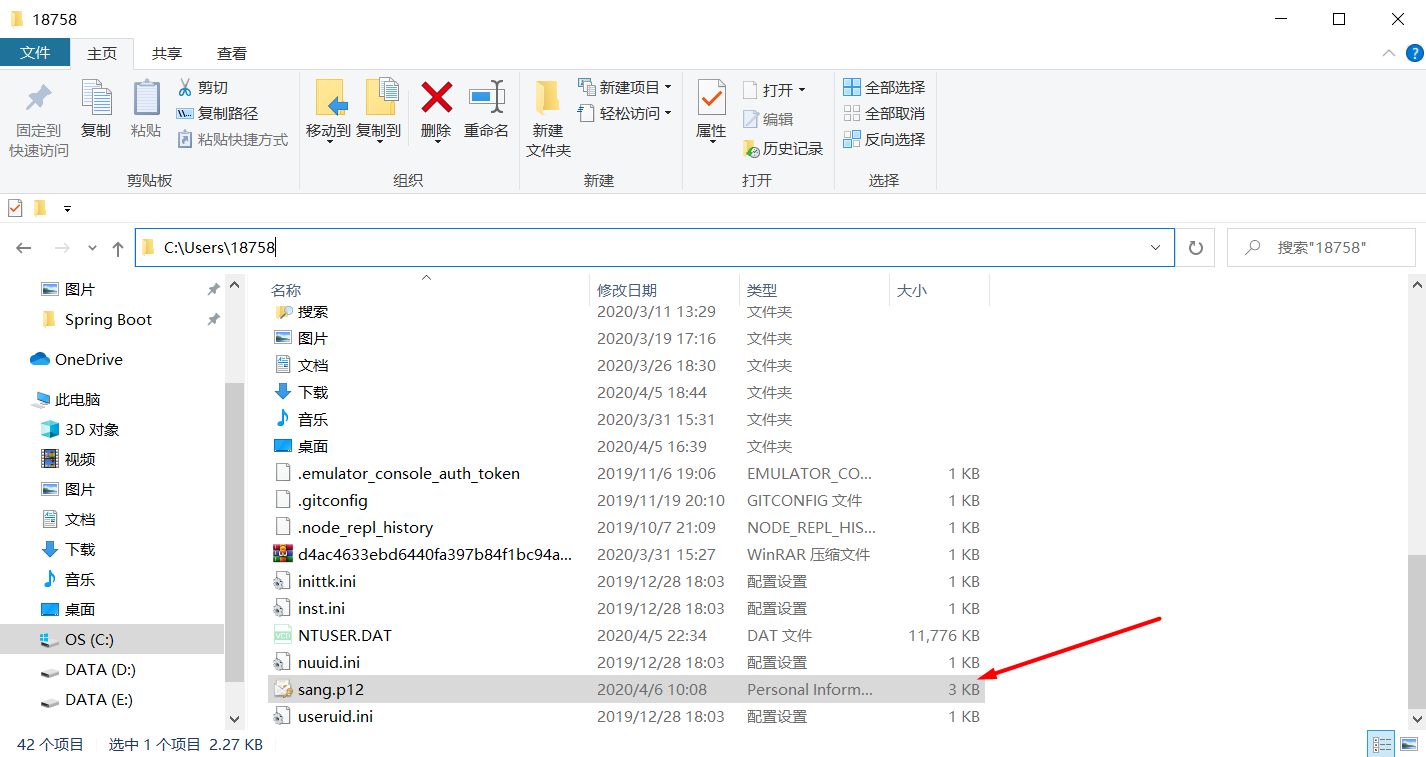
·-keystore表示生成的密匙存放位置和密匙文件名；

·-validity表示密匙的有效时间，单位为天；

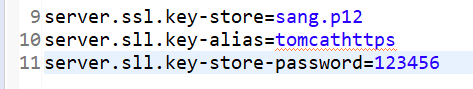
打开cmd创建执行keytool命令，如下：



命令执行后，需要输入密匙口令等信息，根据提示输入即可，输入完成后，会在当前用户目录（可能是执行命令的目录）下生成一个sang.12的文件，如下：



将这个文件复制到项目的根目录下，然后在application.properties中做如下配置：

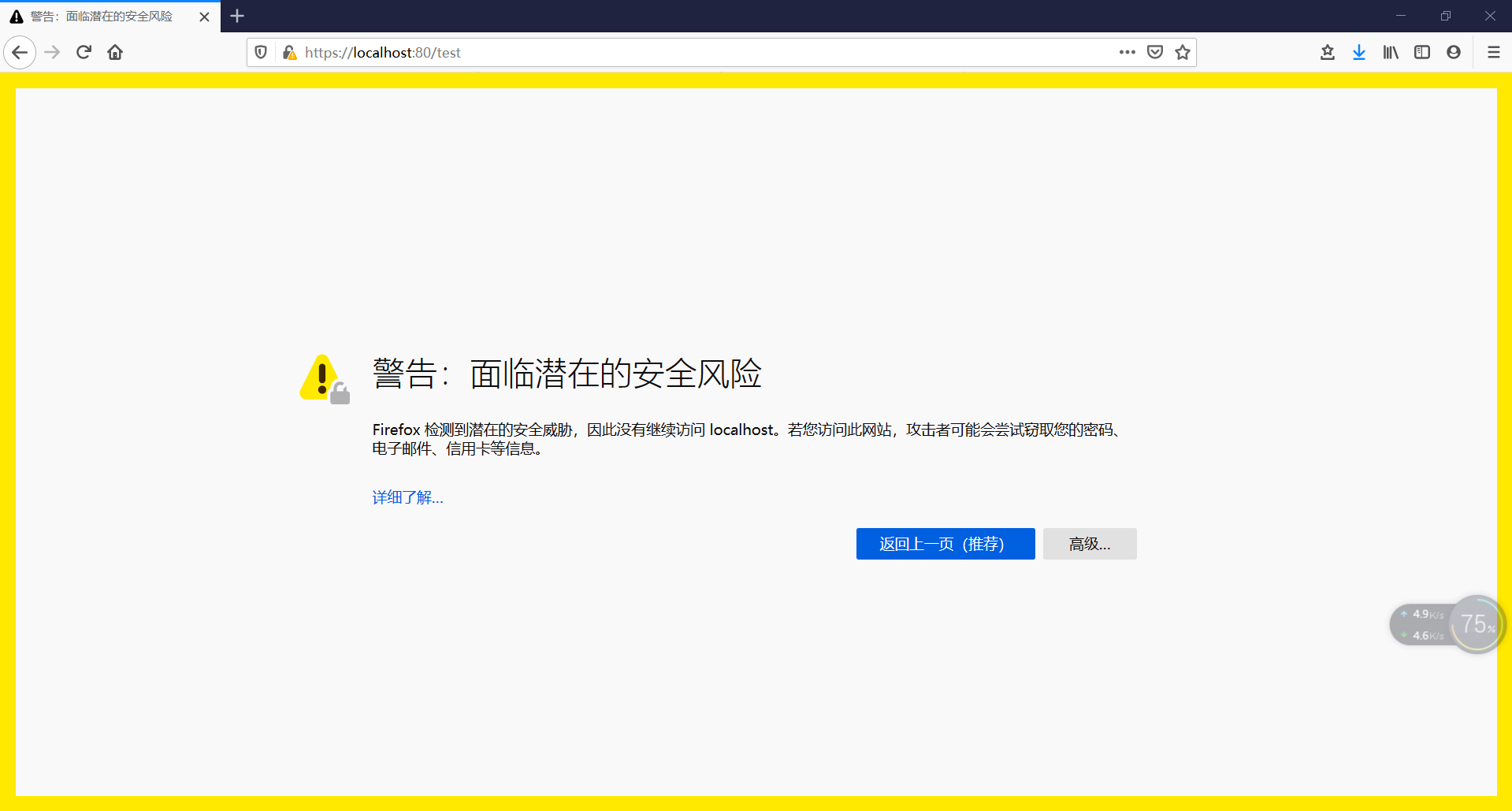


·key-store表示密匙文件名；

·key-alias表示密匙别名；

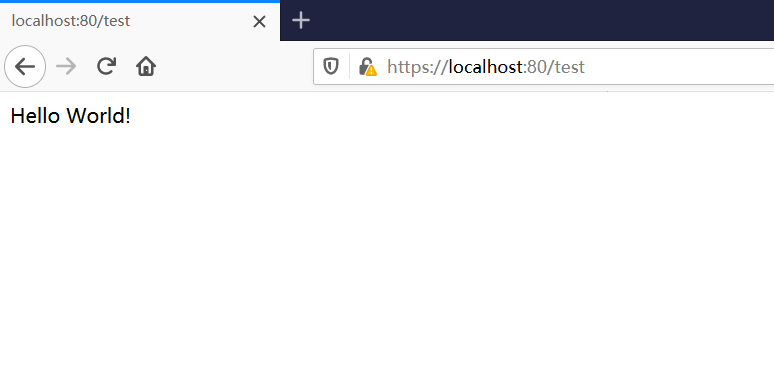
·key-store-password为执行keytool命令生成证书时输入的密码；

启动项目，在浏览器访问时使用https访问，如下：

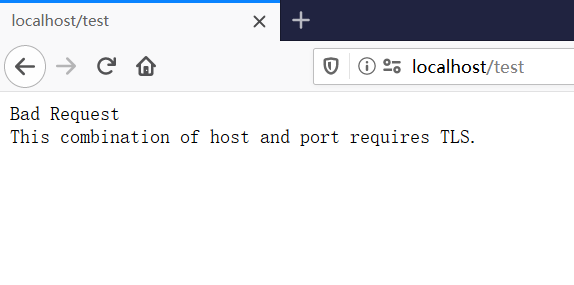


浏览器会提示警告，因为证书是自己生成的，不被浏览器认可，此时添加信任或者继续前进即可，（也看顺便可以查看证书）。如下：

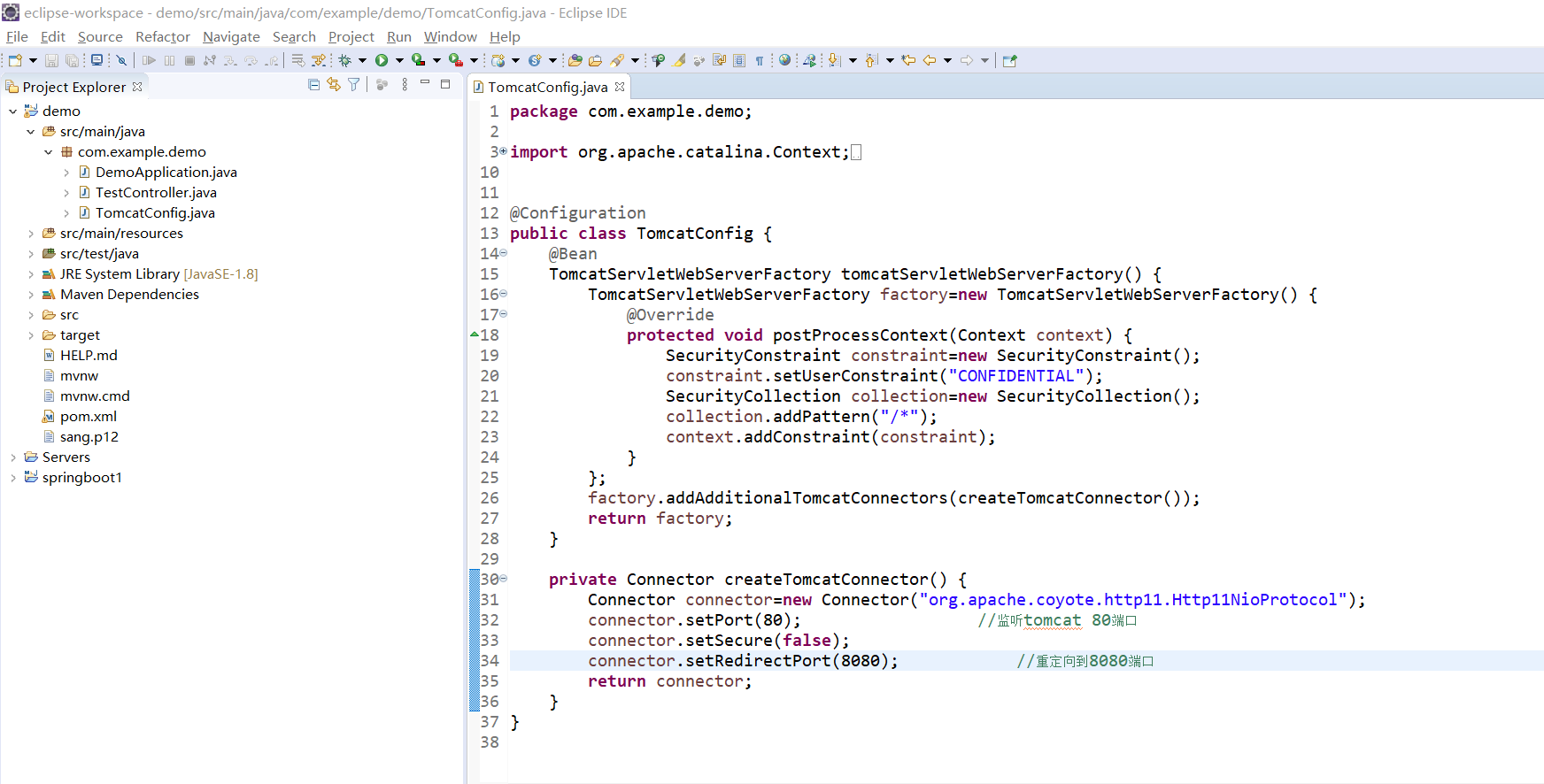




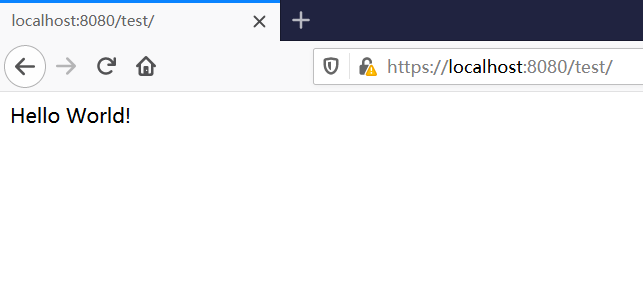
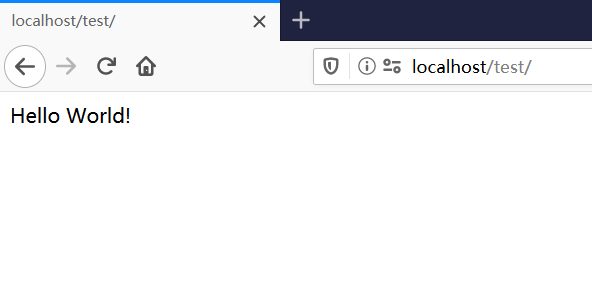
如果此时使用http访问，则会访问失败，



因为Spring Boot不支持同时在配置中启动HTTP和HTTPS。这个时候可以配置请求重定向，将HTTP请求重定向为HTTPS请求，代码如下：

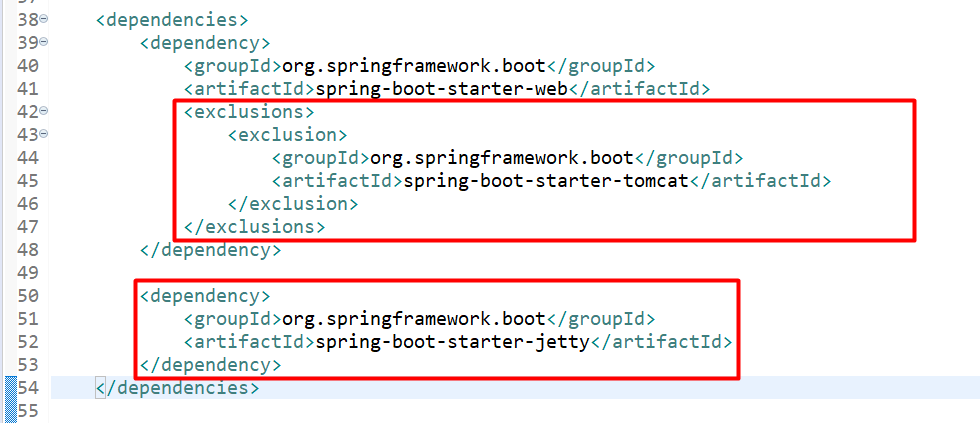


这里配置了一个TomcatServletWebServletFactory，然后添加一个Tomcat的Connector（监听80端口），并将请求转发到8080上去。效果如下：



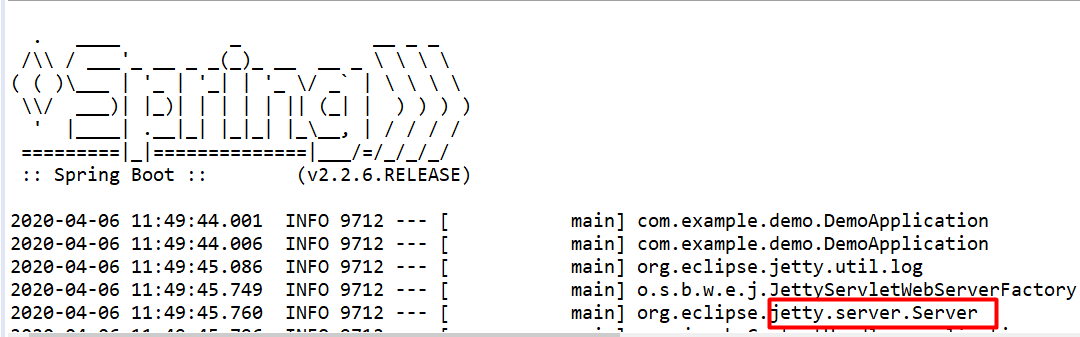
## Jetty配置

除了Tomcat外，也可以在Spring Boot中嵌入Jetty。Jetty也是一个开源的Web容器。配置方式如下：



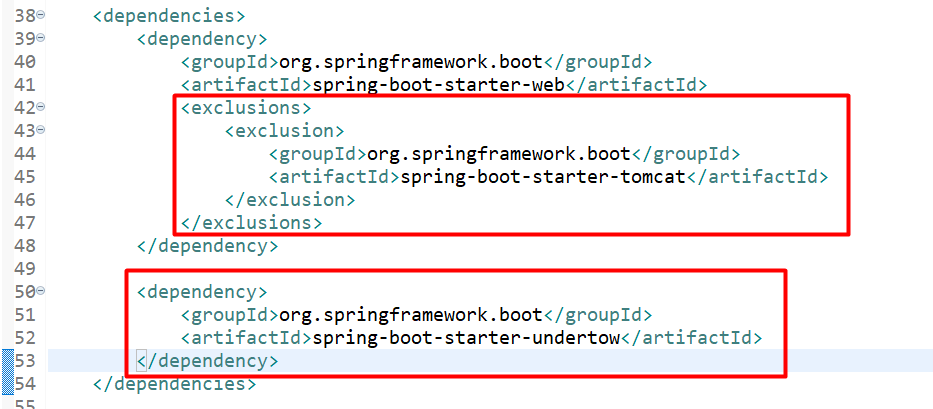
原理是将spring-boot-starter-tomcat排除，使用spring-boot-starter-jetty，表示将默认的Tomcat去除了，使用Jetty容器。

可以从打印信息观察到是否已经更改为Jetty了，如下：



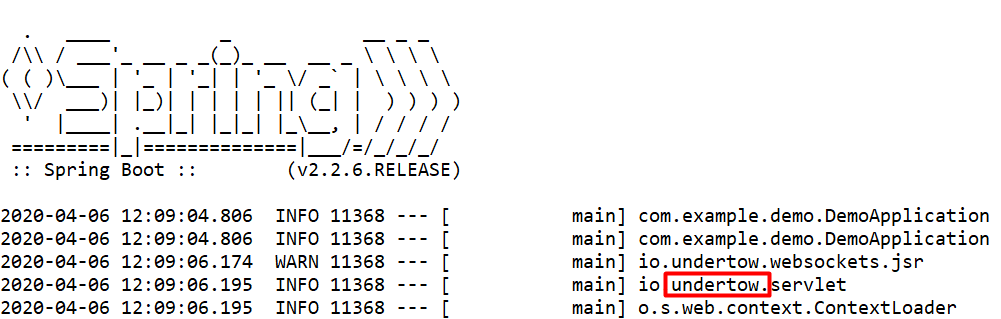
## Undertow配置

Undertow是红帽公司开发的一款Java服务器，具有非常好的性能，在Spring Boot中也得到了很好的支持，配置方式与Jetty类似。如下：



原理是：排除默认的Tomcat容器，添加Undertow容器。

启动项目后，在打印出可观察是否使用的是undertow容器，如下：



# Properties配置

Spring Boot采用了大量的自动化配置，但是对于开发者而言，在实际项目中不免会有一些需要自己手动配置，承载这些自定义配置的文件就是resource目录下的application.properties文件。

1）application.properties配置文件一共可以出现在4个位置：

·项目根目录下的config文件夹中

·项目根目录下

·classpath下的config文件夹中

·classpath下

2）如果这4个位置都有application.properties文件，那么加载的优先级如下：

·项目根目录下的config文件夹中

·项目根目录下

·classpath下的config文件夹中

·classpath下

1. 如果开发者不想使用application.properties作为配置文件名，也可自行定义，如创建app.properties文件作为配置文件，放置在src/java/resource目录下，那么只能将项目打成jar包，然后通过**java -jar**命令运行项目时追加spring-.config.name=app参数使用app.properties作为配置文件。

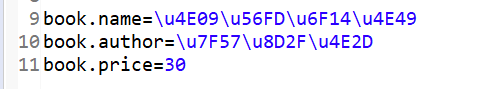


# 类型安全配置属性

配置文件（如application.properties）最终会被加载到Spring Environment中；在Spring提供了@Value注解以及EnvironmentAware接口可以将Spring Environment中的数据注入到属性上，而Spring Boot对此进一步提出了类型安全配置属性，这样即使在数据量非常庞大的情况下，也可以方便地将配置文件中的数据注入Bean中。

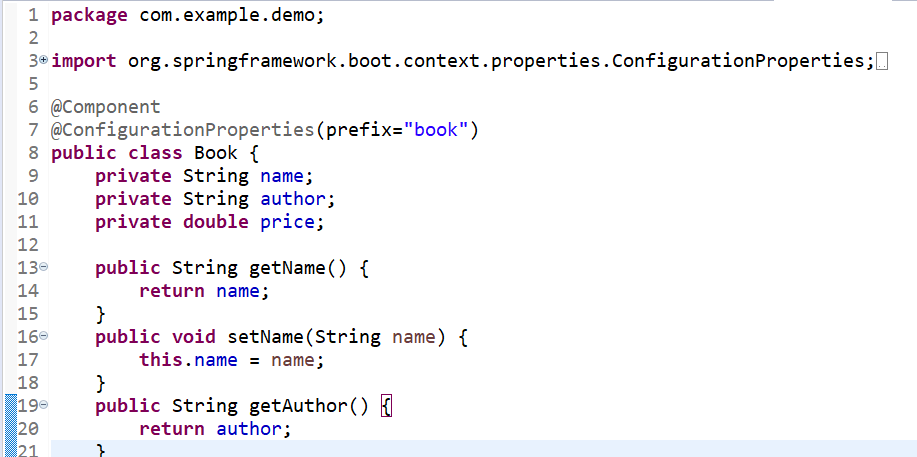
如：

1. 在application.properties文件中添加如下代码：



注：Eclipse中，在properties文件中输入中文会自动进行转码。

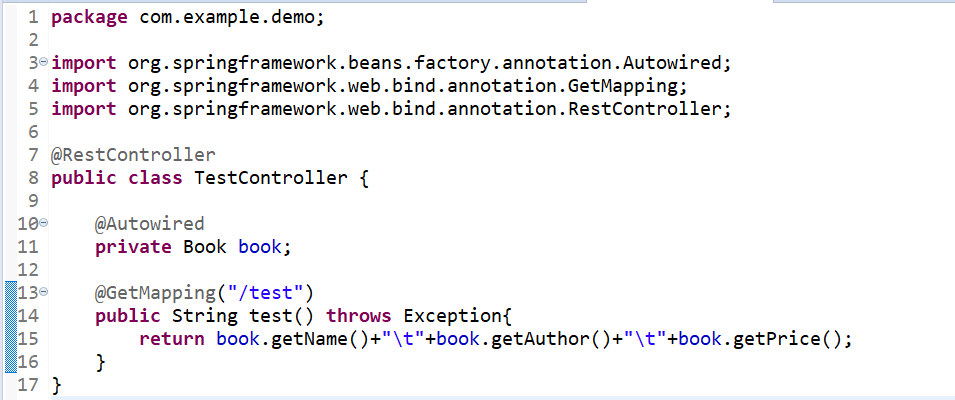
2）创建Bean，代码如下：



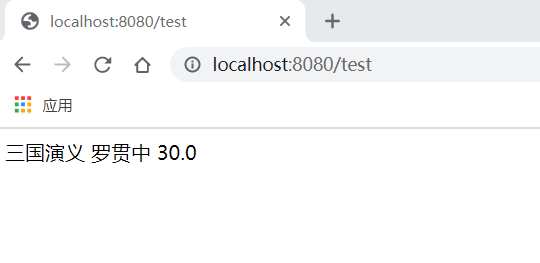
@ConfigurationProperties注解中的prefix属性描述了要加载的配置文件中属性的前缀；

Spring Boot采用了一种宽松的规则来进行属性绑定，如果Bean中的属性名为authorName，那么配置文件中的属性可以是book.author\_name、book.author-name、book.authorName或者book.AUTHORNAME；但需要注意乱码问题。

3）编写控制器，进行测试，代码如下：



1. 启动项目，在浏览器中输入地址进行访问，结果如下：



# YAML配置

## 常规配置

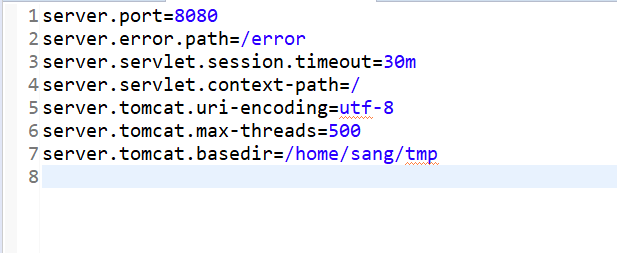
YAML是JSON的超集，简洁而强大，是一种专门用来书写配置文件的语言，可以替代application.properties。

在创建一个Spring Boot项目时，引入了spring-boot-started-web依赖，它会间接的引入snakeyaml依赖，snakeyaml会实现对YAML配置的解析。

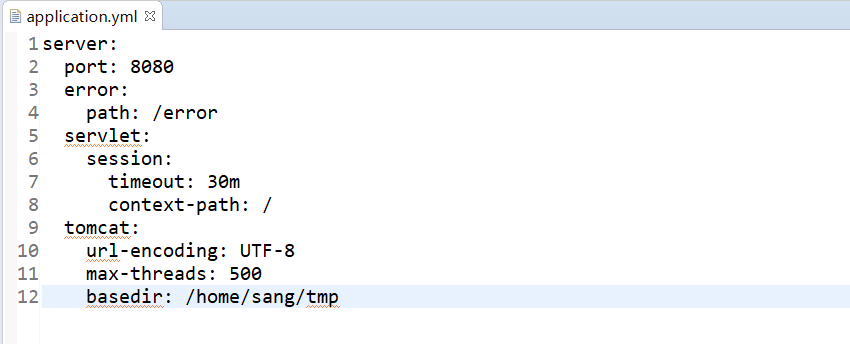
YAML使用简单，利用缩进来表示层级关系，对大小写敏感。

如：

存在application.properties文件，内容如下：

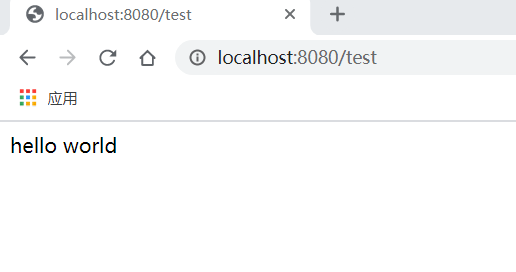


使用YAML文件取代它，如下：



注意：不能使用Tab键缩进，只能使用空格来缩进，两个空格一缩进；还有就是如果“：”后面有值，需要在值前面打一个空格。

将application.properties删除，启动项目，进入浏览器访问，如下：



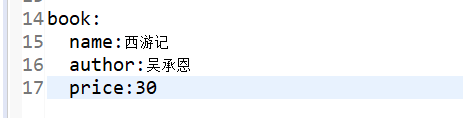
访问成功。

使用application.yml代替application.properties也遵循着application.properties文件的放置规则以及优先级规则。

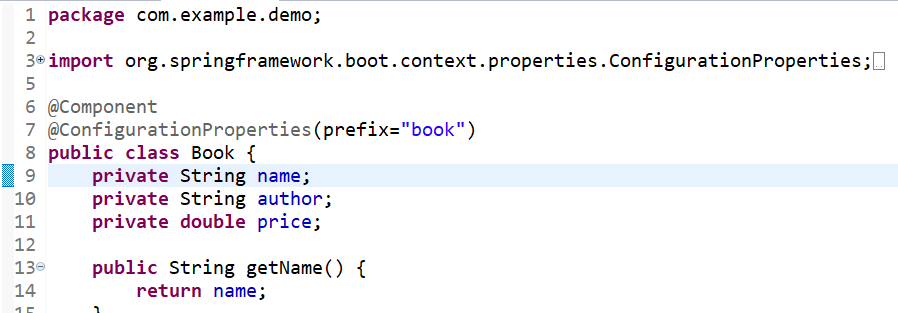
## 复杂配置

YAML不仅支持常规配置，也支持复杂配置，如下：

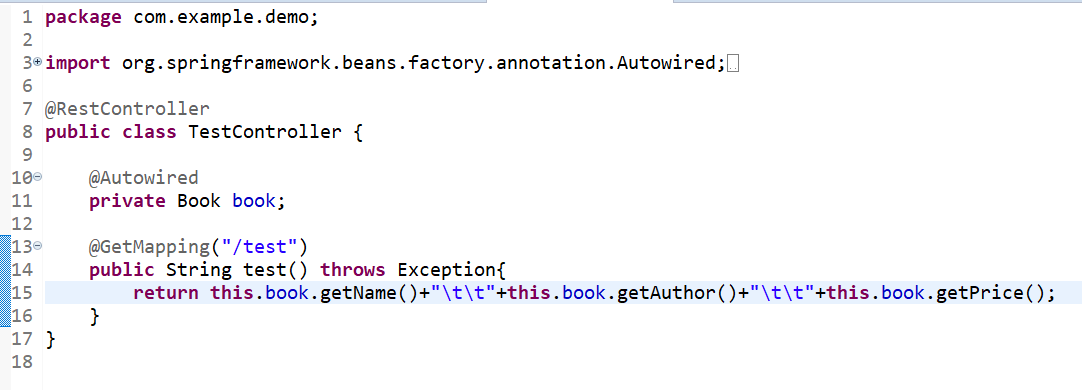
1. 在application.yaml加入以下代码，如下：



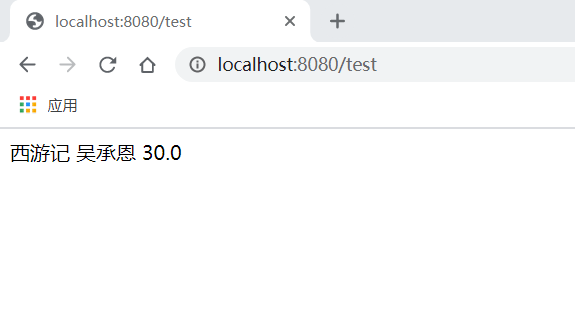
1. 创建Bean，如下：



1. 编写控制器，如下：

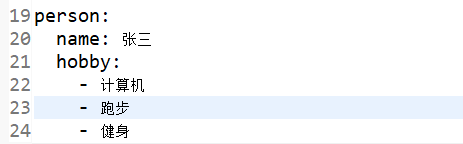


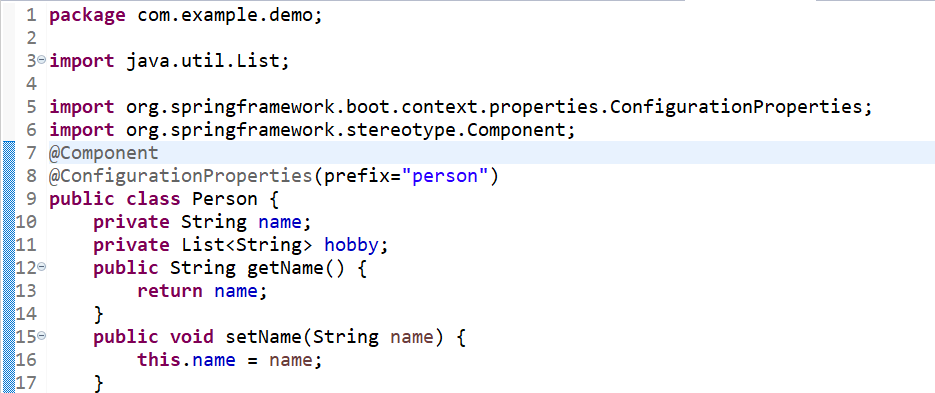
1. 启动项目，进入浏览器访问，如下：



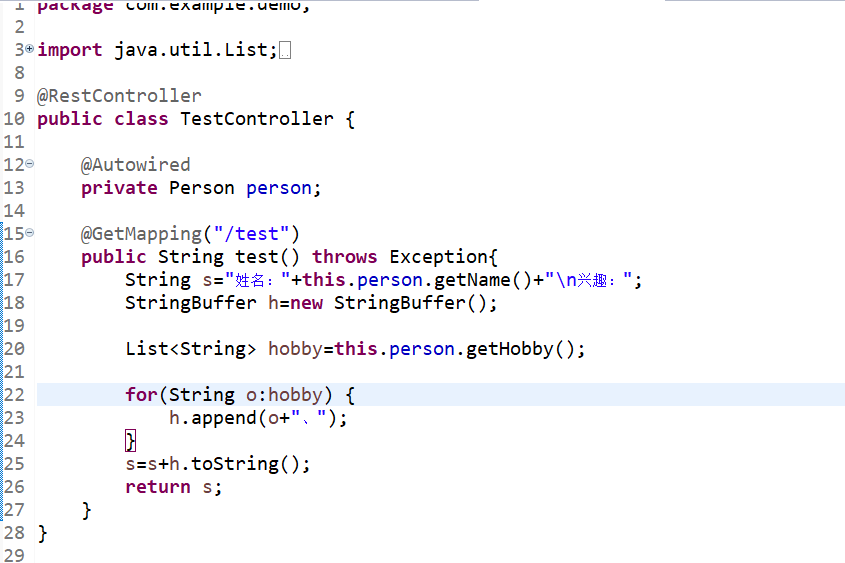
访问成功。

另外，YAML还支持将一个列表的值注入到List类型数据中，如下：

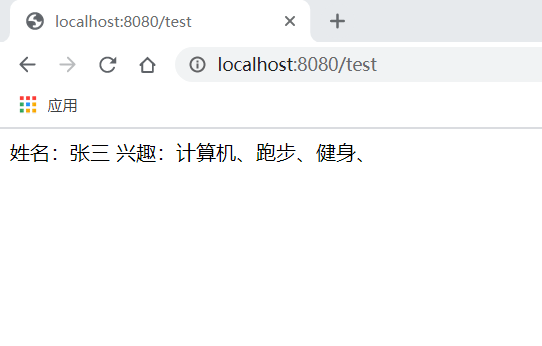
1. 在yaml文件中配置如下：
2. 创建Bean，如下：



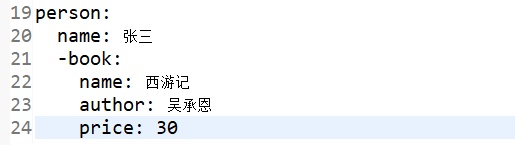
1. 编写控制器，如下：

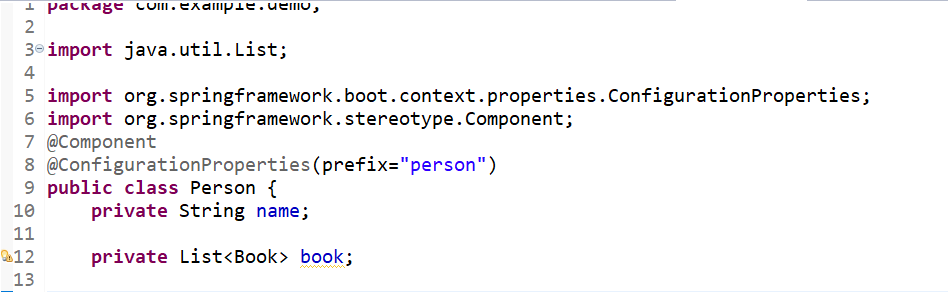


1. 浏览器运行，如下：



1. YAML中的列表中的每一项也可以是一个对象，如下：





在Spring Boot中使用YAML虽然方便，但YAML也有缺点，例如无法使用@PropertySource注解加载YAML文件，如果项目中有这种需求，还是需要使用Properties格式的配置文件。

（@PropertySource注解用于加载资源文件到Spring的Environment中。）

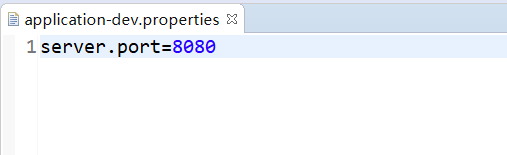
# Profile

开发者在项目发布之前，项目一般需要频繁地在开发环境、测试环境以及生产环境之间进行切换，这个时候大量的配置需要频繁的修改，例如数据库配置、redis配置、jms配置等，频繁修改带来了巨大的工作量，还容易出现副作用，Spring为此提供了解决方案，即@Profile注解。

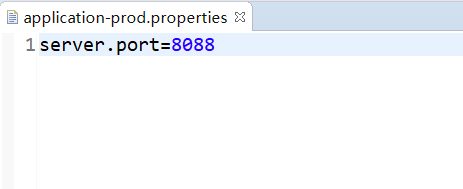
Spring Boot则更进一步提供了更加简洁的解决方案，Spring Boot中约定的不同环境下配置文件名称规则为application-{profile}.properties，{profile}占位符表示当前环境的名称，示例如下：

1. 创建配置文件：在resource目录下创建两个配置文件：application-dev.properties和application-prod.properties，分别表示开发环境中的配置和生产环境中的配置。其中：

application-dev.properties文件内容如下：

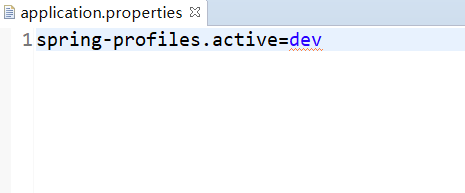


application-prod.properties文件内容如下：



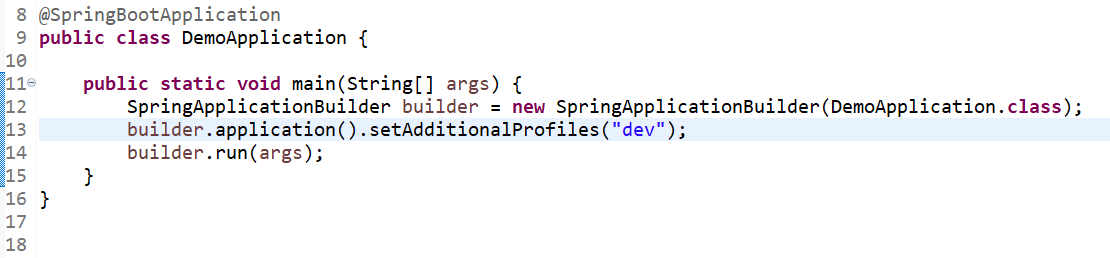
在这里为了演示方便，所以只配置了端口号。

1. 选择环境：在application.properties文件中添加如下代码：



表示使用application-dev.properties文件启动项目，若将dev修改为prod，表示使用application-prod.properties文件启动项目。项目启动后，就可以根据相应的端口进行访问了。

除了在application.properties文件中配置外，我们也可以通过项目启动类配置，代码如下：



如果是通过打包运行，则在使用**java -jar**命令启动项目时，追加参数。如下：

