# S**pringboot\_helloword**

## SpringBoot简介

### 1.1 原有Spring优缺点分析

#### 1.1.1 Spring的优点分析

Spring是Java企业版（Java Enterprise Edition，JEE，也称J2EE）的轻量级代替品。无需开发重量级的Enterprise

JavaBean（EJB），Spring为企业级Java开发提供了一种相对简单的方法，通过依赖注入和面向切面编程，用简单

的Java对象（Plain Old Java Object，POJO）实现了EJB的功能。

#### 1.1.2 Spring的缺点分析

虽然Spring的组件代码是轻量级的，但它的配置却是重量级的。一开始，Spring用XML配置，而且是很多XML配置。Spring 2.5引入了基于注解的组件扫描，这消除了大量针对应用程序自身组件的显式XML配置。Spring 3.0引入了基于Java的配置，这是一种类型安全的可重构配置方式，可以代替XML。

所有这些配置都代表了开发时的损耗。因为在思考Spring特性配置和解决业务问题之间需要进行思维切换，所以编写配置挤占了编写应用程序逻辑的时间。和所有框架一样，Spring实用，但与此同时它要求的回报也不少。

除此之外，项目的依赖管理也是一件耗时耗力的事情。在环境搭建时，需要分析要导入哪些库的坐标，而且还需要分析导入与之有依赖关系的其他库的坐标，一旦选错了依赖的版本，随之而来的不兼容问题就会严重阻碍项目的开发进度。

### 1.2 SpringBoot的概述

#### 1.2.1 SpringBoot解决上述Spring的缺点

SpringBoot对上述Spring的缺点进行的改善和优化，基于约定优于配置的思想，可以让开发人员不必在配置与逻辑业务之间进行思维的切换，全身心的投入到逻辑业务的代码编写中，从而大大提高了开发的效率，一定程度上缩短了项目周期。

#### 1.2.2 SpringBoot的特点

* 为基于Spring的开发提供更快的入门体验
* 开箱即用，没有代码生成，也无需XML配置。同时也可以修改默认值来满足特定的需求
* 提供了一些大型项目中常见的非功能性特性，如嵌入式服务器、安全、指标，健康检测、外部配置等
* SpringBoot不是对Spring功能上的增强，而是提供了一种快速使用Spring的方式

#### 1.2.3 SpringBoot的核心功能

* 起步依赖

起步依赖本质上是一个Maven项目对象模型（Project Object Model，POM），定义了对其他库的传递依赖，这些东西加在一起即支持某项功能。

简单的说，起步依赖就是将具备某种功能的坐标打包到一起，并提供一些默认的功能。

* 自动配置

Spring Boot的自动配置是一个运行时（更准确地说，是应用程序启动时）的过程，考虑了众多因素，才决定

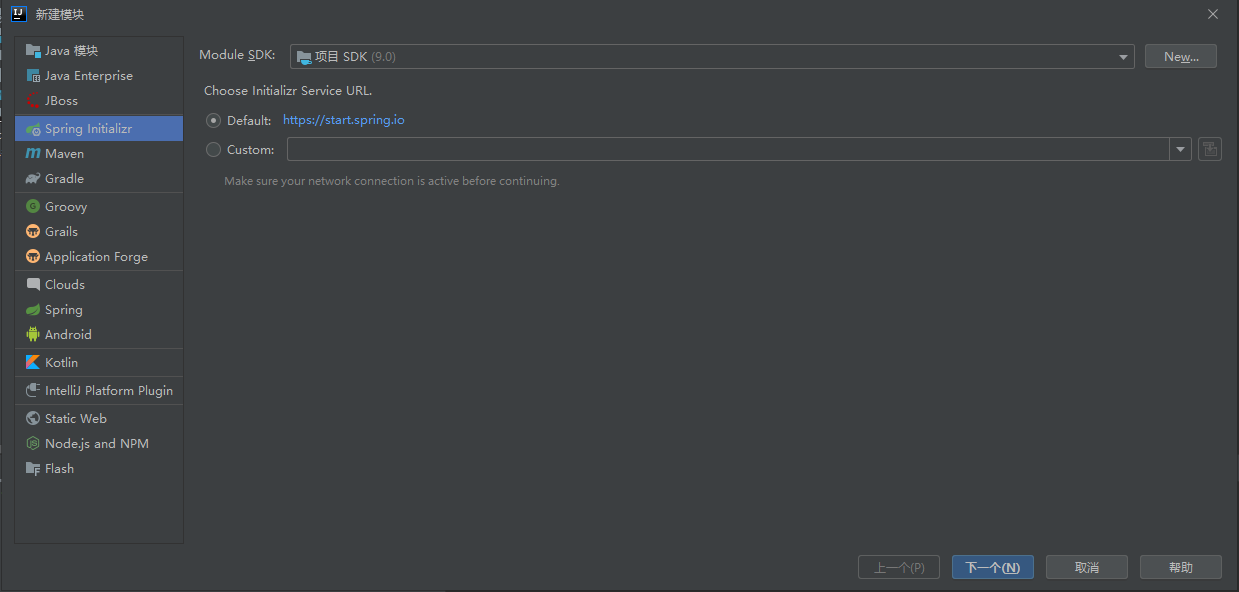
Spring配置应该用哪个，不该用哪个。该过程是Spring自动完成的。

## SpringBoot快速入门

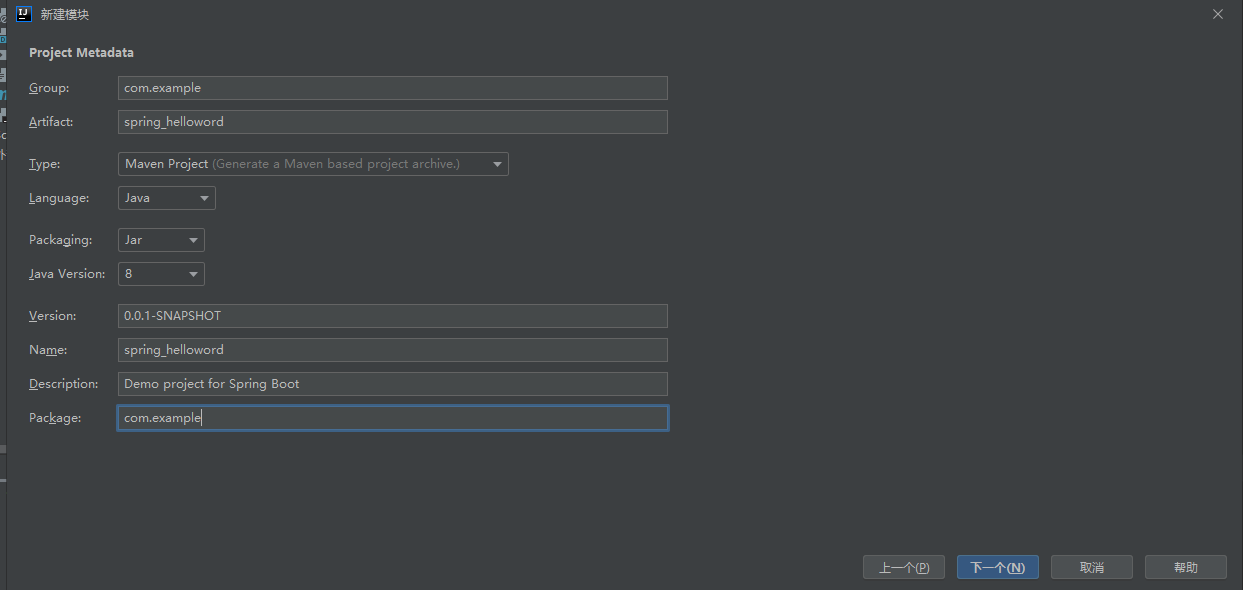
### 2.1 代码实现

#### 2.1.1 创建工程

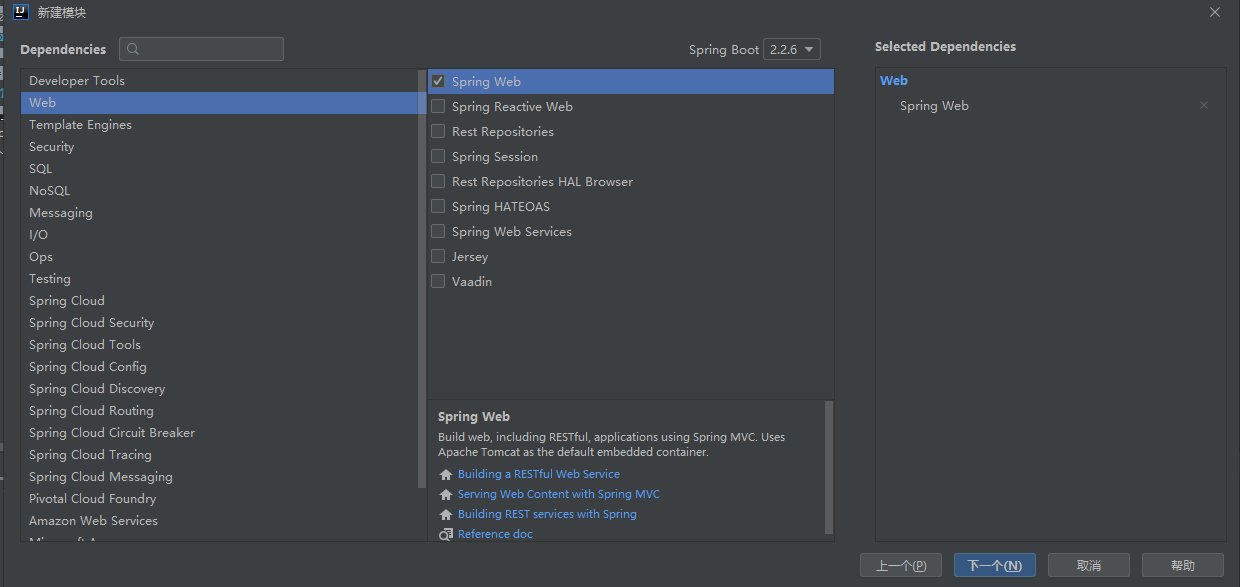
* 使用idea工具创建一个Spring Initializer工程



创建一个springboot\_web框架

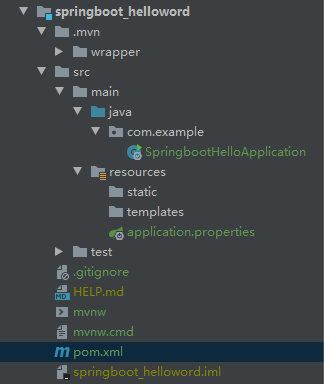


勾选web



点击下一步便创建好一个java工程

* 创建好的工程目录如下



* 在com.example目录下有一个idea为我们创建的SpringbootHelloApplication类。

package com.example;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class SpringbootHellowordApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(SpringbootHellowordApplication.class, args);  
 }  
  
}

该类为SpringBoot引导类

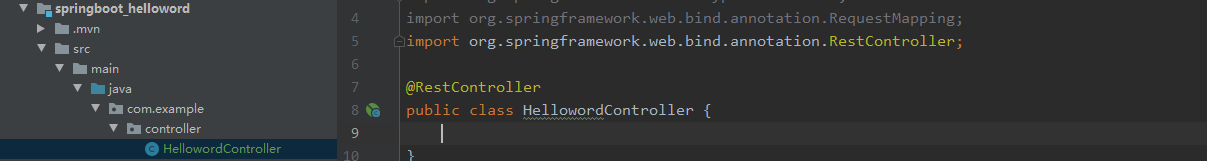
要通过SpringBoot提供的引导类起步SpringBoot才可以进行访问

* 通过idea快速创建的SpringBoot项目的pom.xml中已经导入了我们选择的web的起步依赖的坐标

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.2.6.RELEASE</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
 <groupId>com.example</groupId>  
 <artifactId>springboot\_helloword</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>springboot\_helloword</name>  
 <description>Demo project for Spring Boot</description>  
  
 <properties>  
 <java.version>1.8</java.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 <exclusions>  
 <exclusion>  
 <groupId>org.junit.vintage</groupId>  
 <artifactId>junit-vintage-engine</artifactId>  
 </exclusion>  
 </exclusions>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

#### 2.1.2 编写代码

* 在com.example中创建一个controller类



对于了解spring框架与三层架构的同学对controller类与@Controller注解并不陌生，但是也有小伙伴会问@RestController是什么呢

@ResponseBody表示方法的返回值直接以指定的格式写入Http response body中，而不是解析为跳转路径。(添加连接)

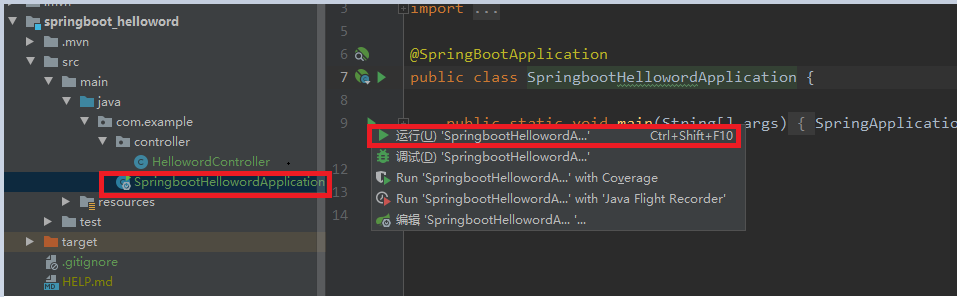
* 在HellowordController中编写方法

@GetMapping("/hello")  
public String sayHello(){  
 return "hello";  
}

@GetMapping这个注释不知道大家是否陌生，我也给大家做了总结，@GetMapping与@RequestMapping的区别(添加连接)

#### 2.1.3 开始运行

点击SpringboothHellowordApplication



执行SpringBoot起步类的主方法，控制台打印日志如下：

. \_\_\_\_ \_ \_\_ \_ \_

/\\ / \_\_\_'\_ \_\_ \_ \_(\_)\_ \_\_ \_\_ \_ \ \ \ \

( ( )\\_\_\_ | '\_ | '\_| | '\_ \/ \_` | \ \ \ \

\\/ \_\_\_)| |\_)| | | | | || (\_| | ) ) ) )

' |\_\_\_\_| .\_\_|\_| |\_|\_| |\_\\_\_, | / / / /

=========|\_|==============|\_\_\_/=/\_/\_/\_/

:: Spring Boot :: (v2.0.1.RELEASE)

2018-05-08 14:29:59.714 INFO 5672 --- [ main]

com.itheima.MySpringBootApplication : Starting MySpringBootApplication on

DESKTOP-RRUNFUH with PID 5672

(C:\Users\muzimoo\IdeaProjects\IdeaTest\springboot\_quick\target\classes started by

muzimoo in C:\Users\muzimoo\IdeaProjects\IdeaTest)

... ... ...

o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/\*\*] onto handler of type

[class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]

2018-05-08 14:30:03.126 INFO 5672 --- [ main]

o.s.j.e.a.AnnotationMBeanExporter : Registering beans for JMX exposure on

startup

2018-05-08 14:30:03.196 INFO 5672 --- [ main]

o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http)

with context path ''

2018-05-08 14:30:03.206 INFO 5672 --- [ main]

com.itheima.MySpringBootApplication : Started MySpringBootApplication in 4.252

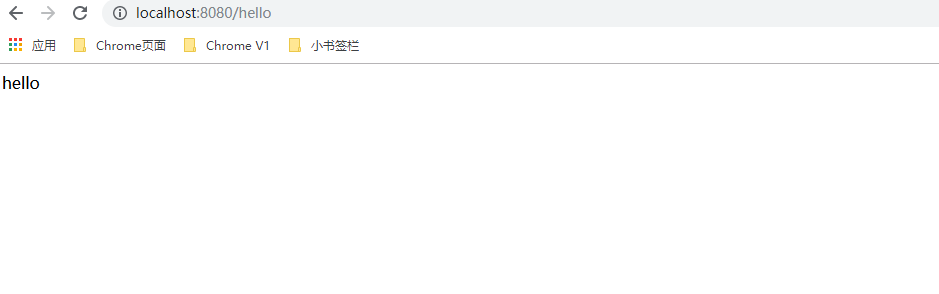
seconds (JVM running for 5.583)

通过日志发现，Tomcat started on port(s): 8080 (http) with context path ''

tomcat已经起步，端口监听8080，web应用的虚拟工程名称为空

### 2.2 运行结果

打开浏览器访问url地址为：http://localhost:8080/hello



# S**pringboot\_properties**

这个项目我们来主要讲解基于springboot框架的配置文件的配置与读取。

废话不多说直接上代码

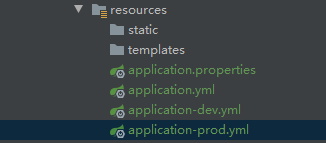
## 代码实现

### 创建工程

像上一篇文章所讲述的项目工程搭建，我们还是使用springboot框架的web工程。不了解的同学可以去看我上一篇的文章，这里就不在演示项目是如何搭建的了。

### 编写代码

* 在resource目录中添加配置文件



对于配置文件得写法和作用我在下一章中再仔细解释，现在的你只要知道我们得目的就是读取配置文件中得信息就好了。

* application.yml

server:  
 port: 8080  
 servlet:  
 context-path: /demo  
spring:  
 profiles:  
 active: prod

* application-prod.yml

application:  
 name: prod环境 @artifactId@  
 version: prod环境 @version@  
developer:  
 name: prod环境 xkcoding  
 website: prod环境 http://xkcoding.com  
 qq: prod环境 237497819  
 phone-number: prod环境 17326075631

* 在com.example目录下创建property包并创建ApplicationProperty和DeveloperProperty类,用于去取出application-prod.yml中得配置。
* ApplicationProperty

@Data  
@Component  
public class ApplicationProperty {  
 @Value("${application.name}")  
 private String name;  
 @Value("${application.version}")  
 private String version;  
}

@Data注释的作用是减少属性的set和get方法

其@Data的使用需要引入依赖

<dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <optional>true</optional>  
</dependency>

@Value注释的作用就是取出注释中的application.name等属性

application:  
 name: prod环境 @artifactId@  
……

* 编写controller

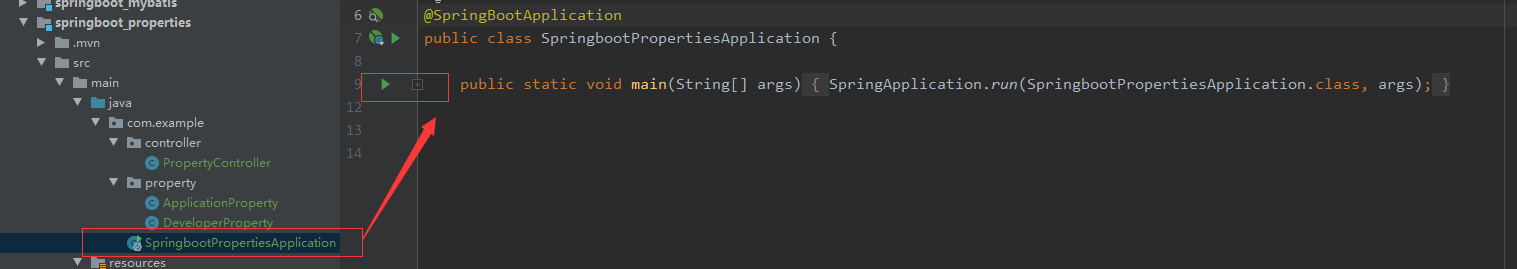
@RestController  
public class PropertyController {  
 private final ApplicationProperty applicationProperty;  
 private final DeveloperProperty developerProperty;  
  
 @Autowired  
 public PropertyController(ApplicationProperty applicationProperty, DeveloperProperty developerProperty) {  
 this.applicationProperty = applicationProperty;  
 this.developerProperty = developerProperty;  
 }  
  
 @GetMapping("/property")  
 public String index() {  
 return applicationProperty.toString()+"/////"+developerProperty.toString();  
 }  
}

这里@Autoired大家应该很熟悉是spring IOC中的自动注入

现在我们就可以开始运行了！

当然运行的时候还会有意想不到的事情发生

### 1.3 开始运行



老方法我们还是点击这里开始运行

## 运行结果

. \_\_\_\_ \_ \_\_ \_ \_

/\\ / \_\_\_'\_ \_\_ \_ \_(\_)\_ \_\_ \_\_ \_ \ \ \ \

( ( )\\_\_\_ | '\_ | '\_| | '\_ \/ \_` | \ \ \ \

\\/ \_\_\_)| |\_)| | | | | || (\_| | ) ) ) )

' |\_\_\_\_| .\_\_|\_| |\_|\_| |\_\\_\_, | / / / /

=========|\_|==============|\_\_\_/=/\_/\_/\_/

:: Spring Boot :: (v2.2.6.RELEASE)

2020-04-09 19:49:18.213 INFO 28688 --- [ main]

2020-04-09 19:49:18.213 INFO 28688 --- [ main] c.e.SpringbootPropertiesApplication : The following profiles are active: prod

2020-04-09 19:49:20.261 INFO 28688 --- [ main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)

2020-04-09 19:49:20.281 INFO 28688 --- [ main] o.apache.catalina.core.StandardService : Starting service [Tomcat]

2020-04-09 19:49:20.281 INFO 28688 --- [ main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.33]

2020-04-09 19:49:20.401 INFO 28688 --- [ main] o.a.c.c.C.[Tomcat].[localhost].[/demo] : Initializing Spring embedded WebApplicationContext

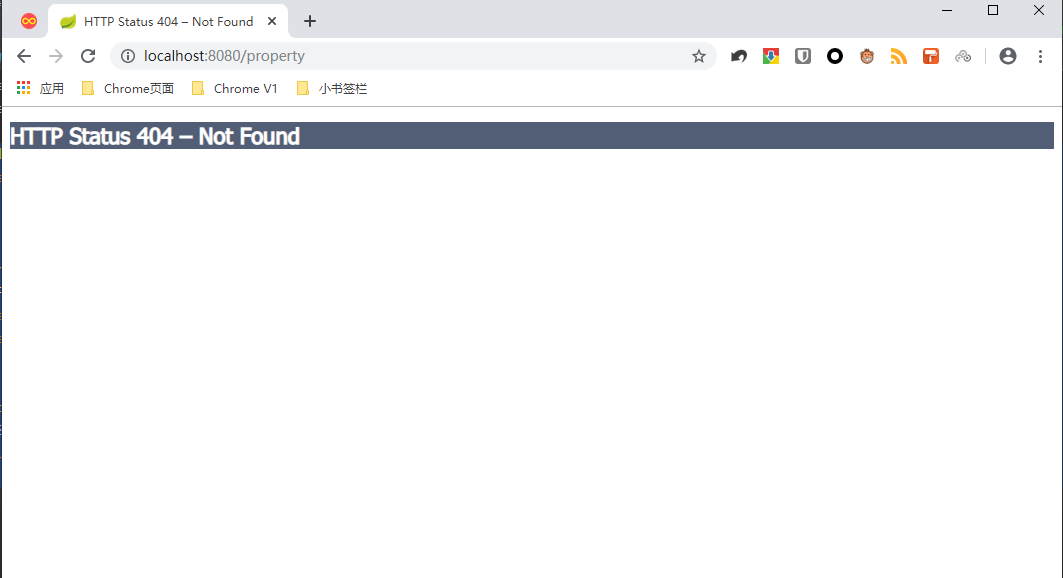
2020-04-09 19:49:20.401 INFO 28688 --- [ main] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization completed in 2048 ms

2020-04-09 19:49:20.731 INFO 28688 --- [ main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'

2020-04-09 19:49:21.071 INFO 28688 --- [ main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path '/demo'

2020-04-09 19:49:21.081 INFO 28688 --- [ main] c.e.SpringbootPropertiesApplication : Started SpringbootPropertiesApplication in 4.315 seconds (JVM running for 7.815)

上浏览器



？？？？？？？？？？？

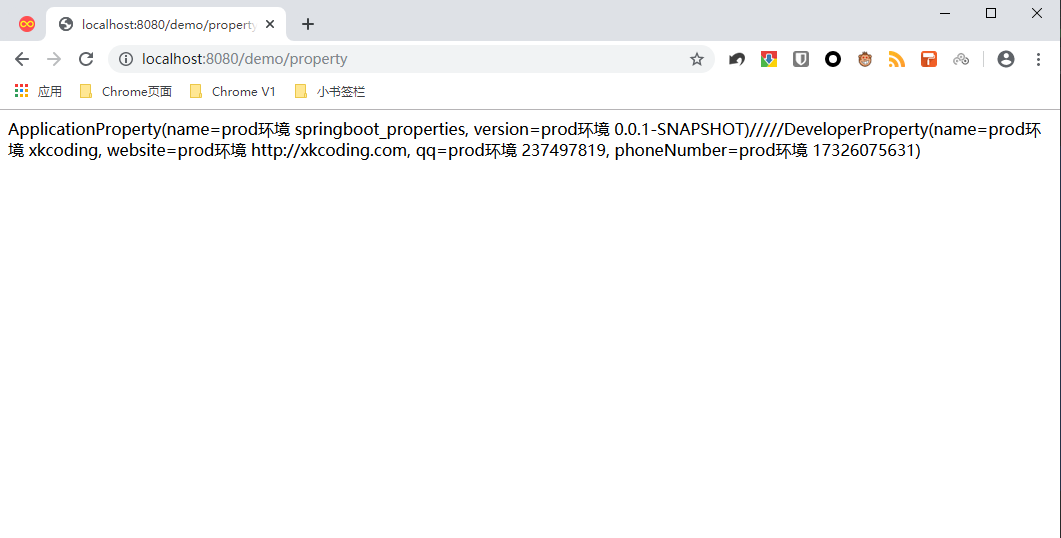
为什么404了

其实大家有没有在注意到spring Cnsole(控制台所输出的)

2020-04-09 19:49:21.071 INFO 28688 --- [ main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path '/demo'

/demo ?????????????????哪里来的

不管了，我们先试一下



成功了，这是为什么呢

## 3. 为什么

原因就在于我们另一个的配置文件大家还记得吗

* application.yml

server:  
 port: 8080  
 servlet:  
 context-path: /demo  
spring:  
 profiles:  
 active: prod

在这里配置的context-path:/demo

## 4. SpringBoot的配置文件

### 4.1 SpringBoot配置文件类型

#### 4.1.1 SpringBoot配置文件类型和作用

SpringBoot是基于约定的，所以很多配置都有默认值，但如果想使用自己的配置替换默认配置的话，就可以使用

application.properties或者application.yml（application.yaml）进行配置。

SpringBoot默认会从Resources目录下加载application.properties或application.yml（application.yaml）文件

其中，application.properties文件是键值对类型的文件，之前一直在使用，所以此处不在对properties文件的格式进行阐述。除了properties文件外，SpringBoot还可以使用yml文件进行配置，下面对yml文件进行讲解。

#### 4.1.2 application.yml配置文件

##### 4.1.2.1 yml配置文件简介

YML文件格式是YAML (YAML Aint Markup Language)编写的文件格式，YAML是一种直观的能够被电脑识别的的数据数据序列化格式，并且容易被人类阅读，容易和脚本语言交互的，可以被支持YAML库的不同的编程语言程序导入，比如： C/C++, Ruby, Python, Java, Perl, C#, PHP等。YML文件是以数据为核心的，比传统的xml方式更加简洁。

YML文件的扩展名可以使用.yml或者.yaml。

##### 4.1.2.2 yml配置文件的语法

###### 4.1.2.2.1 配置普通数据

语法： key: value

示例代码：

name: haohao

注意：value之前有一个空格

###### 4.1.2.2.2 配置对象数据

语法：

key:

key1: value1

key2: value2

或者：

key: {key1: value1,key2: value2}

示例代码：

person:

name: haohao

age: 31

addr: beijing

#或者

person: {name: haohao,age: 31,addr: beijing}

注意：key1前面的空格个数不限定，在yml语法中，相同缩进代表同一个级别

###### 4.1.2.2.3 配置Map数据

同上面的对象写法

###### 4.1.2.2.3 配置数组（List、Set）数据

语法：

key:

- value1

- value2

或者：

key: [value1,value2]

示例代码：

city:

- beijing

- tianjin

- shanghai

- chongqing

#或者

city: [beijing,tianjin,shanghai,chongqing]

#集合中的元素是对象形式

student:

- name: zhangsan

age: 18

score: 100

- name: lisi

age: 28

score: 88

- name: wangwu

age: 38

score: 90

注意：value1与之间的 - 之间存在一个空格

#### 4.1.3 SpringBoot配置信息的查询

上面提及过，SpringBoot的配置文件，主要的目的就是对配置信息进行修改的，但在配置时的key从哪里去查询呢？我们可以查阅SpringBoot的官方文档

文档URL：<https://docs.spring.io/spring-boot/docs/2.0.1.RELEASE/reference/htmlsingle/#common-application-properties>

常用的配置摘抄如下：

# QUARTZ SCHEDULER (QuartzProperties)

spring.quartz.jdbc.initialize-schema=embedded # Database schema initialization mode.

spring.quartz.jdbc.schema=classpath:org/quartz/impl/jdbcjobstore/tables\_@@platform@@.

sql # Path to the SQL file to use to initialize the database schema.

spring.quartz.job-store-type=memory # Quartz job store type.

我们可以通过配置application.poperties 或者 application.yml 来修改SpringBoot的默认配置

例如：

application.properties文件

application.yml文件

spring.quartz.properties.\*= # Additional Quartz Scheduler properties.

# ----------------------------------------

# WEB PROPERTIES

# ----------------------------------------

# EMBEDDED SERVER CONFIGURATION (ServerProperties)

server.port=8080 # Server HTTP port.

server.servlet.context-path= # Context path of the application.

server.servlet.path=/ # Path of the main dispatcher servlet.

# HTTP encoding (HttpEncodingProperties)

spring.http.encoding.charset=UTF-8 # Charset of HTTP requests and responses. Added to

the "Content-Type" header if not set explicitly.

# JACKSON (JacksonProperties)

spring.jackson.date-format= # Date format string or a fully-qualified date format

class name. For instance, `yyyy-MM-dd HH:mm:ss`.

# SPRING MVC (WebMvcProperties)

spring.mvc.servlet.load-on-startup=-1 # Load on startup priority of the dispatcher

servlet.

spring.mvc.static-path-pattern=/\*\* # Path pattern used for static resources.

spring.mvc.view.prefix= # Spring MVC view prefix.

spring.mvc.view.suffix= # Spring MVC view suffix.

# DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)

spring.datasource.driver-class-name= # Fully qualified name of the JDBC driver. Auto

detected based on the URL by default.

spring.datasource.password= # Login password of the database.

spring.datasource.url= # JDBC URL of the database.

spring.datasource.username= # Login username of the database.

# JEST (Elasticsearch HTTP client) (JestProperties)

spring.elasticsearch.jest.password= # Login password.

spring.elasticsearch.jest.proxy.host= # Proxy host the HTTP client should use.

spring.elasticsearch.jest.proxy.port= # Proxy port the HTTP client should use.

spring.elasticsearch.jest.read-timeout=3s # Read timeout.

spring.elasticsearch.jest.username= # Login username.

我们可以通过配置application.poperties 或者 application.yml 来修改SpringBoot的默认配置

例如：

application.properties文件

server.port=8888

server.servlet.context-path=demo

application.yml文件

server:

port: 8888

servlet:

context-path: /demo

### 4.2 配置文件与配置类的属性映射方式

#### 4.2.1 使用注解@Value映射

我们可以通过@Value注解将配置文件中的值映射到一个Spring管理的Bean的字段上

例如：

application.properties配置如下：

person:

name: zhangsan

age: 18

或者，application.yml配置如下：

person:

name: zhangsan

age: 18

实体Bean代码如下：

@Controller

public class QuickStartController {

@Value("${person.name}")

private String name;

@Value("${person.age}")

private Integer age;

@RequestMapping("/quick")

@ResponseBody

public String quick(){

return "springboot 访问成功! name="+name+",age="+age;

}

}

#### 4.2.2 使用注解@ConfifigurationProperties映射

通过注解@ConfifigurationProperties(prefifix="配置文件中的key的前缀")可以将配置文件中的配置自动与实体进行映

射

application.properties配置如下：

person:

name: zhangsan

age: 18

或者，application.yml配置如下：

person:

name: zhangsan

age: 18

实体Bean代码如下：

@Controller

@ConfigurationProperties(prefix = "person")

public class QuickStartController {

private String name;

private Integer age;

@RequestMapping("/quick")

@ResponseBody

public String quick(){

return "springboot 访问成功! name="+name+",age="+age;

}

public void setName(String name) {

this.name = name;

}

public void setAge(Integer age) {

this.age = age;

}

}

注意：使用@ConfifigurationProperties方式可以进行配置文件与实体字段的自动映射，但需要字段必须提供set方法才可以，而使用@Value注解修饰的字段不需要提供set方法

# Springboot\_actuator

## SpringBoot actuator简介

Spring Boot Actuator可以帮助你监控和管理Spring Boot应用，比如健康检查、审计、统计和HTTP追踪等。所有的这些特性可以通过JMX或者HTTP endpoints来获得。

Actuator同时还可以与外部应用监控系统整合，比如 [Prometheus](https://prometheus.io/), [Graphite](https://graphiteapp.org/), [DataDog](https://www.datadoghq.com/), [Influx](https://www.influxdata.com/), [Wavefront](https://www.wavefront.com/), [New Relic](https://newrelic.com/)等。这些系统提供了非常好的仪表盘、图标、分析和告警等功能，使得你可以通过统一的接口轻松的监控和管理你的应用。

Actuator使用[Micrometer](http://micrometer.io/" \t "_blank)来整合上面提到的外部应用监控系统。这使得只要通过非常小的配置就可以集成任何应用监控系统。

说了一大堆官方文档也不知道是什么

总的来说Spring Boot Actuator提供一系列HTTP端点来暴露项目信息，用来监控和管理项目。其实就是可以通过网页的访问去进行对运行项目的监控。

你想在项目部署之后，我们不能Debug,没有打印台，对项目的运行情况和其他的比如说占用内存啊磁盘的等等情况，都可以使用web网页访问的形式进行查看。

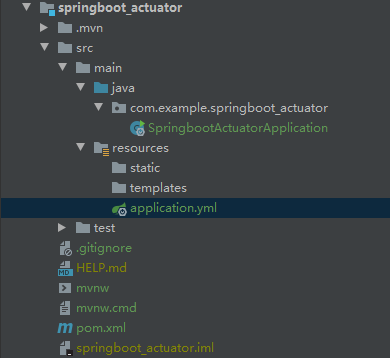
更多内容请移步[官方文档](https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#production-ready)。

## SpringBoot actuator快速入门

### 2.1项目搭建

和以前一样，我们还是选择一个SpringBoot框架的web工程。不了解的同学可以去看我SpringBoot\_Helloword文章，这里就不在演示项目是如何搭建的了。

首先，我们先来看一下我们最终的项目目录，看看需要添加那些文件



？？？？？？？？？？？

好像什么都没有添加？

你说对了，这次我们什么类啊，接口啊都不用写只需要导入依赖项和配置配置文件就好了

### 2.2代码实现

* 我们先看看我们需要导入那些依赖

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.2.6.RELEASE</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
 <groupId>com.example</groupId>  
 <artifactId>springboot\_actuator</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>springboot\_actuator</name>  
 <description>Demo project for Spring Boot</description>  
  
 <properties>  
 <java.version>1.8</java.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-actuator</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.security</groupId>  
 <artifactId>spring-security-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

在这些依赖中我们今天的主角就是

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-actuator</artifactId>  
</dependency>

当然还有小伙伴会问

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
</dependency>

<dependency>  
 <groupId>org.springframework.security</groupId>  
 <artifactId>spring-security-test</artifactId>  
 <scope>test</scope>  
</dependency>

这两个依赖是干什么用的,其实这有关于spring框架中的spring-security问题，如果想了解详情的小伙伴可以去我的另一篇文章仔细看一下，在这里我们就不多做赘述了

就算不了解的小伙伴也不用着急，等我们看一下运行结果时你就知道他们时干什么用的了。

* 文件配置

在application.yml中添加配置

server:  
 port: 8080  
 servlet:  
 context-path: /demo  
*# 若要访问端点信息，需要配置用户名和密码*spring:  
 security:  
 user:  
 name: liyingjie  
 password: 123456  
management:  
 *# 端点信息接口使用的端口，为了和主系统接口使用的端口进行分离* server:  
 port: 8090  
 servlet:  
 context-path: /sys  
 *# 端点健康情况，默认值"never"，设置为"always"可以显示硬盘使用情况和线程情况* endpoint:  
 health:  
 show-details: *always  
 # 设置端点暴露的哪些内容，默认["health","info"]，设置"\*"代表暴露所有可访问的端点* endpoints:  
 web:  
 exposure:  
 include: '\*'

我们来解释一下这些配置信息都有什么含义

server:  
 port: 8080  
 servlet:  
 context-path: /demo  
8080端口为我们整个项目的访问端口，/demo为默认路径。

*# 若要访问端点信息，需要配置用户名和密码*spring:  
 security:  
 user:  
 name: liyingjie  
 password: 123456  
这是我们在访问端点信息时需要填入的用户名和密码

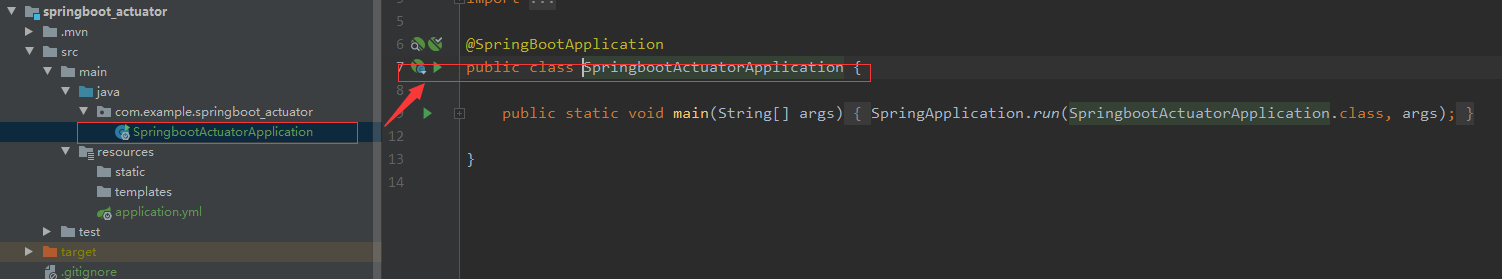
*# 端点信息接口使用的端口，为了和主系统接口使用的端口进行分离* server:  
 port: 8090  
 servlet:  
 context-path: /sys  
这是我们在使用actuator时访问的接口与默认路径

*# 端点健康情况，默认值"never"，设置为"always"可以显示硬盘使用情况和线程情况* endpoint:  
 health:  
 show-details: *always  
 # 设置端点暴露的哪些内容，默认["health","info"]，设置"\*"代表暴露所有可访问的端点* endpoints:  
 web:  
 exposure:  
 include: '\*'

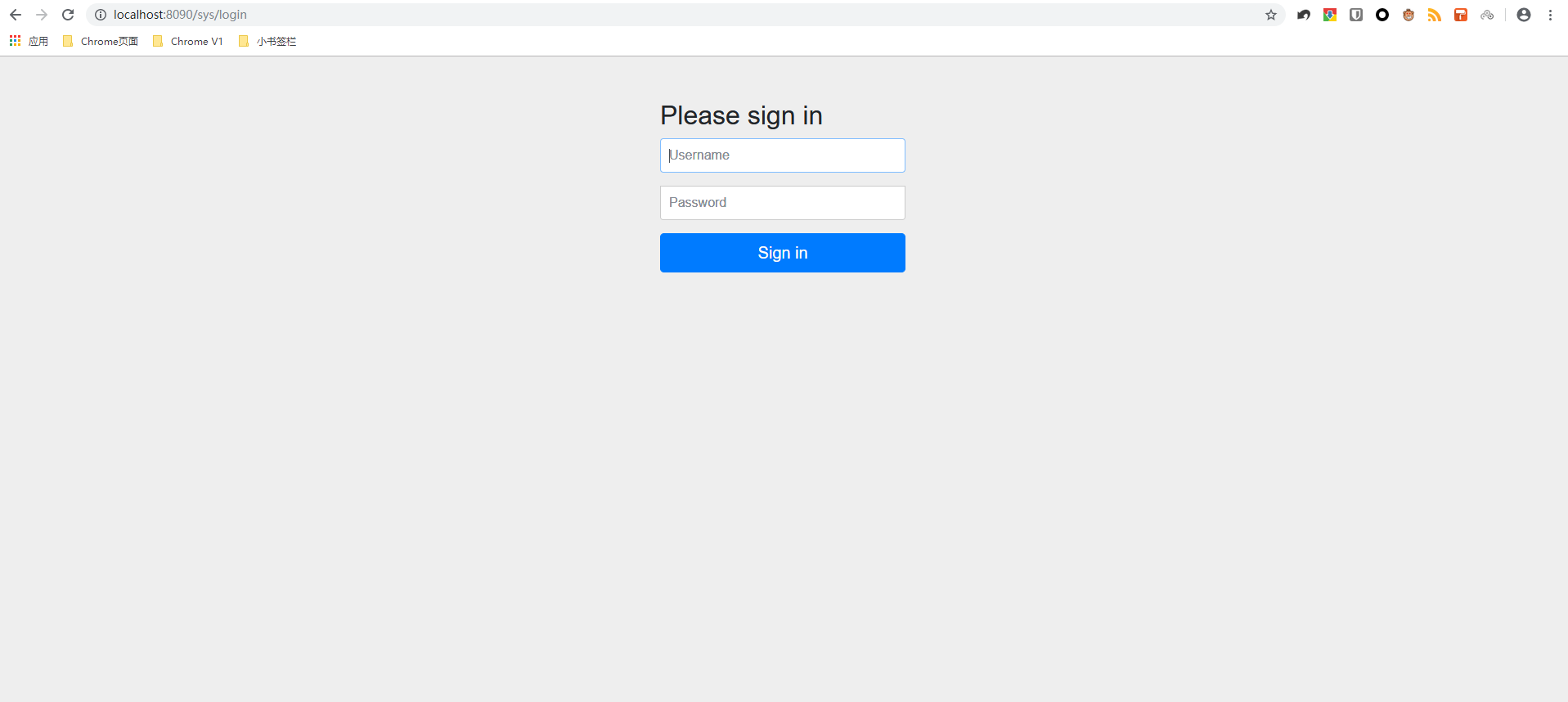
剩下的这些为actuator的配置，后面会详细叙述

### 2.3运行结果

* 老样子，我们点击起步类开始运行



* 我们访问网址localhost:8090/sys/actuator



我们进入到了登录页面

* 可能由小伙伴们会问了，这个登录页面哪里来的呢，我们也并没有编写HTML或者JP文件，怎么会突然跑出个登录页面来呢。

还记得我们一开始引入的依赖项里有两个依赖没有给大家详细解释吗

就是

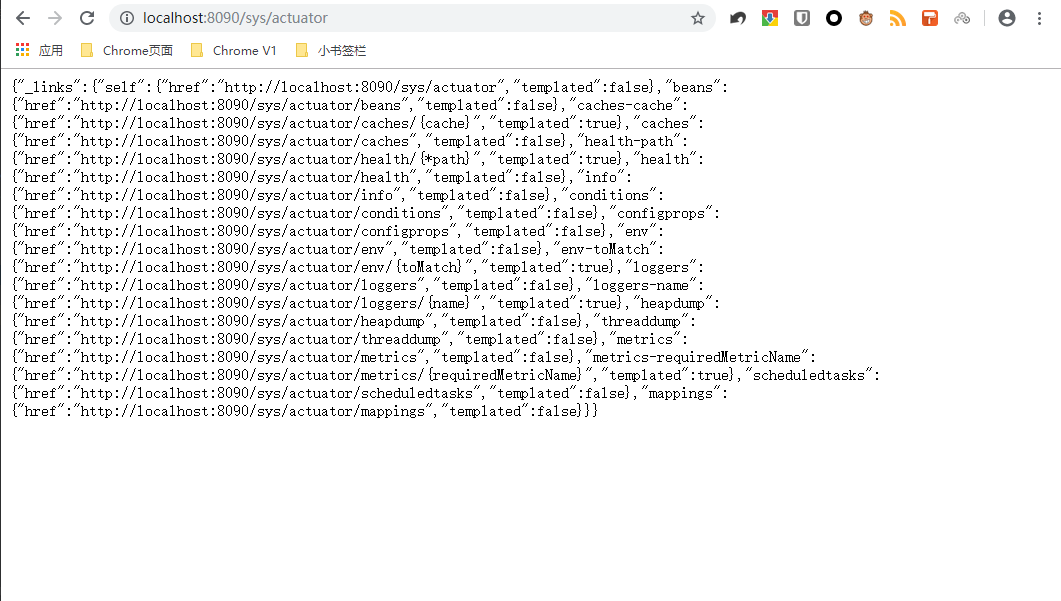
<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
</dependency>

<dependency>  
 <groupId>org.springframework.security</groupId>  
 <artifactId>spring-security-test</artifactId>  
 <scope>test</scope>  
</dependency>

引入这两个依赖就会出现登录页面

而账号密码就是我们在配置文件中设置的用户名和密码。

* 输入用户名和密码后我们就进入页面了



看到这里，这堆乱七八糟的东西都是什么？

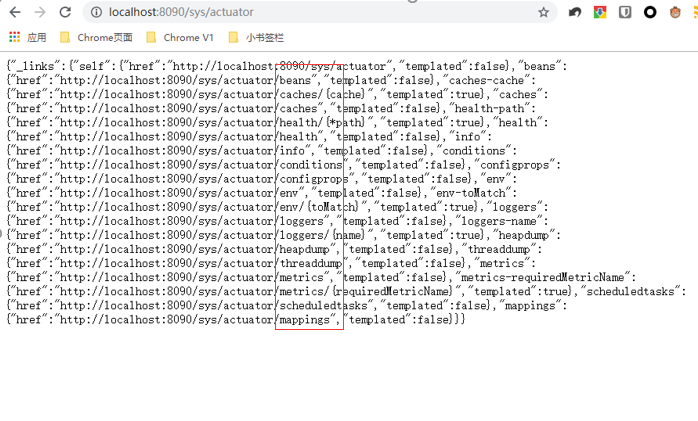
别着急，我们下面详细解释一下

## actuator

### 3.1 Actuator 端点说明

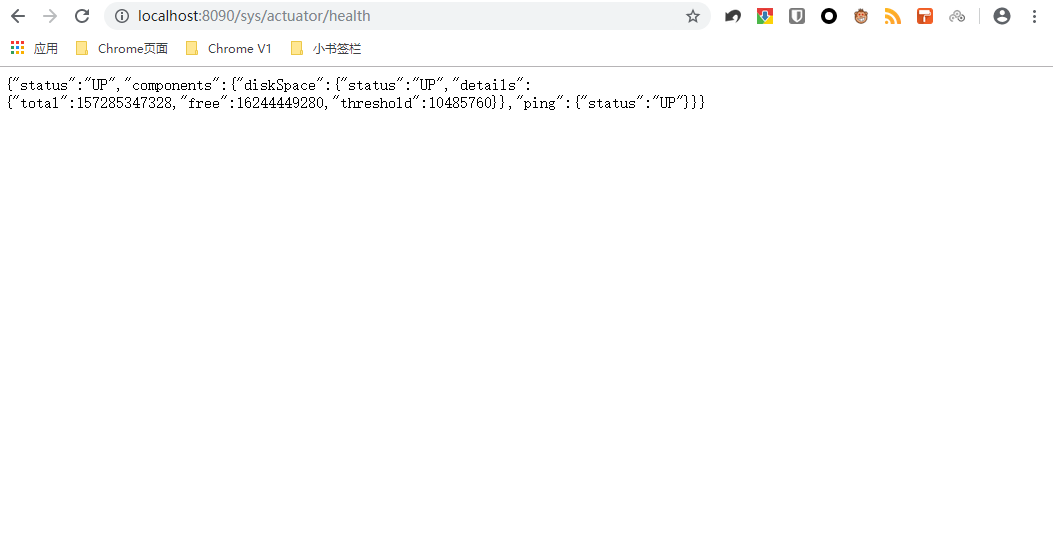
| **端点** | **描述** |
| --- | --- |
| auditevents | 获取当前应用暴露的审计事件信息 |
| beans | 获取应用中所有的 Spring Beans 的完整关系列表 |
| caches | 获取公开可以用的缓存 |
| conditions | 获取自动配置条件信息，记录哪些自动配置条件通过和没通过的原因 |
| configprops | 获取所有配置属性，包括默认配置，显示一个所有 @ConfigurationProperties 的整理列版本 |
| env | 获取所有环境变量 |
| flyway | 获取已应用的所有Flyway数据库迁移信息，需要一个或多个 Flyway Bean |
| liquibase | 获取已应用的所有Liquibase数据库迁移。需要一个或多个 Liquibase Bean |
| health | 获取应用程序健康指标（运行状况信息） |
| httptrace | 获取HTTP跟踪信息（默认情况下，最近100个HTTP请求-响应交换）。需要 HttpTraceRepository Bean |
| info | 获取应用程序信息 |
| integrationgraph | 显示 Spring Integration 图。需要依赖 spring-integration-core |
| loggers | 显示和修改应用程序中日志的配置 |
| logfile | 返回日志文件的内容（如果已设置logging.file.name或logging.file.path属性） |
| metrics | 获取系统度量指标信息 |
| mappings | 显示所有@RequestMapping路径的整理列表 |
| scheduledtasks | 显示应用程序中的计划任务 |
| sessions | 允许从Spring Session支持的会话存储中检索和删除用户会话。需要使用Spring Session的基于Servlet的Web应用程序 |
| shutdown | 关闭应用，要求endpoints.shutdown.enabled设置为true，默认为 false |
| threaddump | 获取系统线程转储信息 |
| heapdump | 返回hprof堆转储文件 |
| jolokia | 通过HTTP公开JMX bean（当Jolokia在类路径上时，不适用于WebFlux）。需要依赖 jolokia-core |
| prometheus | 以Prometheus服务器可以抓取的格式公开指标。需要依赖 micrometer-registry-prometheus |

在这个表格中的所有端口就是我们刚才访问网页中的



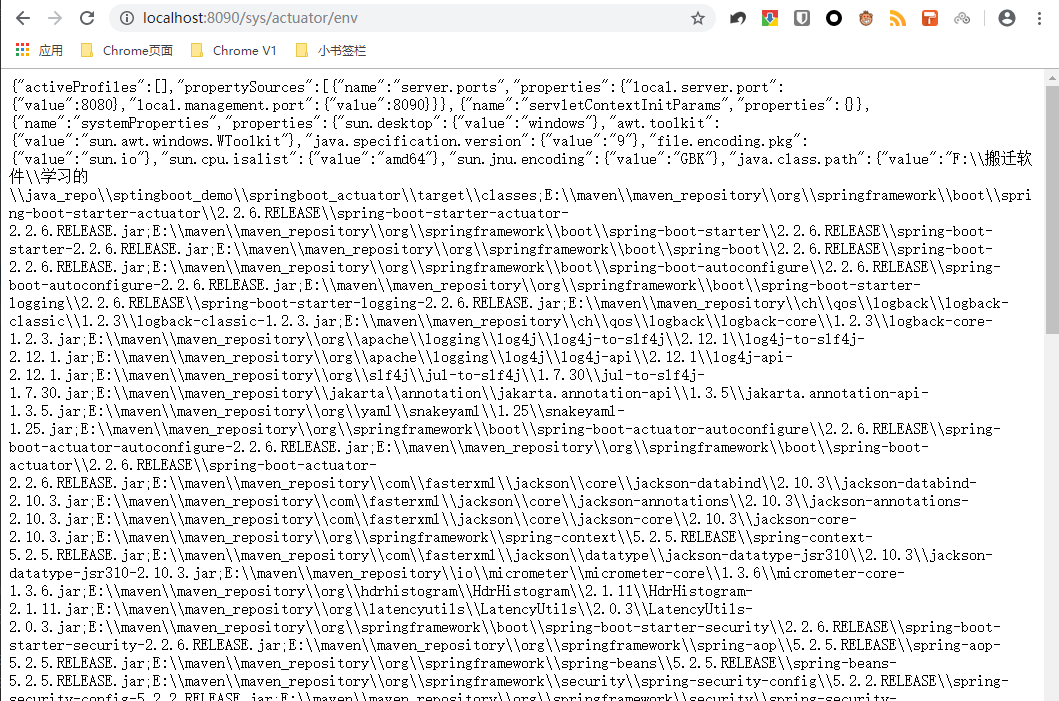
现在大家就明白这个网页是干什么的了吧，其实就是个引导页。

好的，我们现在进入一个看看，在web浏览器中输入localhost:8090/sys/actuator/health



health主要用来检测应用的运行状况，是使用最多的一个监控点。监控软件通常使用该接口实时监测应用运行状况，在系统出现故障时把报警信息推送给相关人员，如磁盘空间使用情况、数据库和缓存等的一些健康指标。

在web浏览器中输入localhost:8090/sys/actuator/env



通过 env 可以获取到所有关于当前 Spring Boot 应用程序的运行环境信息，如：操作系统信息（systemProperties）、环境变量信息、JDK 版本及 ClassPath 信息、当前启用的配置文件（activeProfiles）、propertySources、应用程序配置信息（applicationConfig）等。

好了，就举这两个例子吧。。。。其实剩下的我也没怎么用过。。。。