# SpringBoot （https://spring.io/projects）

### Spring Boot核心功能

1.独立运行的Spring项目

Spring Boot可以以jar包的形式独立运行，运行一个Spring Boot项目只需通过java–jar xx.jar来运行。

2.内嵌Servlet容器

Spring Boot可选择内嵌Tomcat、Jetty或者Undertow。

3.提供starter简化Maven配置

Spring提供了一系列的starter pom来简化Maven的依赖加载

4.自动配置Spring

Spring Boot会根据在类路径中的jar包、类，为jar包里的类自动配置Bean，而Spring Boot没有提供支持，则可以自定义自动配置。

5.准生产的应用监控

Spring Boot提供基于http、ssh、telnet对运行时的项目进行监控。

1. 无代码生成和xml配置

### 构建Spring Boot项目

1.Maven项目构建

我们可以用任意开发工具新建空的Maven项目。

2.修改pom.xml

（1）添加Spring Boot的父级依赖，这样的项目就是Spring Boot项目了。**spring-boot-starter-parent**是一个特殊的starter，它用来提供相关的Maven默认依赖，使用它之后，常用的包依赖可以省去version标签。

<**parent**>

　　　　 <groupId>org.springframework.boot</groupId>

<artifactId>**spring-boot-starter-parent**</artifactId>

　　　　 <version>1.3.0.M1</version>

　　　　 <relativePath/>

</parent>

（2）在**dependencies**添加Web支持的starter pom，这样就添加了Web的依赖。

　　　　 <dependency>

　　　　　　 <groupId>org.springframework.boot</groupId>

　　　　　　 <artifactId>**spring-boot-starter-web**</artifactId>

　　　　 </dependency>

（3）添加Spring Boot的**编译插件**。

<**build**>

　　　　 <plugins>

　　　　　　 <plugin>

　　　　　　　　 <groupId>org.springframework.boot</groupId>

　　　　　　　　 <artifactId>**spring-boot-maven-plugin**</artifactId>

　　　　　　 </plugin>

　　　　 </plugins>

</build>

简单演示：

1. 新建Spring Boot项目后，生成的项目的根包目录下会有一个artifactId+Application命名规则的入口类



1. 在入口类直接编写代码

package com.wisely.ch5\_2\_2;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

@**RestController** //@RestController = @Controller + @ResponseBody

@SpringBootApplication //Spring Boot项目的核心注解，主要目的是开启自动配置

public class Ch522Application {

　　 @RequestMapping("/")

　　　　 String index() {

　　　　　　 return "Hello Spring Boot";

　　　　 }

　　 public static void main(String[] args) { // 项目启动的入口

**SpringApplication**.**run**(**Ch522Application**.class, args);

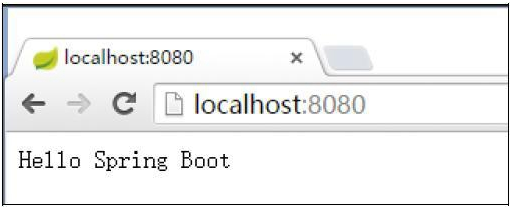
　 }

}

3.运行效果

我们可以通过Maven命令，运行项目。

mvn spring-boot:run 或单击Ch522Application右键，在右键菜单中选择以Spring Boot APP或Java Application运行项目



# Spring Boot 核心

## 基本配置

1. 入口类和**@SpringBootApplication**

Spring Boot通常有一个名为**\*Application**的入口类，入口类里有一个main方法，这个main方法其实就是一个标准的Java应用的入口方法。在main方法中使用SpringApplication.run（xxx.class，args），启动Spring Boot应用项目。

@SpringBootApplication是Spring Boot的核心注解，它是一个组合注解，源码如下：

@Target(ElementType.TYPE)

@Retention(RetentionPolicy.RUNTIME)

@Documented

@Inherited

@Configuration

@EnableAutoConfiguration

@ComponentScan

public @interface SpringBootApplication {

　　 Class<?>[] exclude() default {};

　　 String[] excludeName() default {};

}

@SpringBootApplication注解主要组合了@Configuration、@EnableAutoConfiguration、@ComponentScan；若不使用@SpringBootApplication注解，则可以在入口类上直接使用@Configuration、@EnableAutoConfiguration、@ComponentScan。

Spring [Boot会自动扫描@SpringBootApplication所在类的**同级包**以及**下级包里的Bean（若为JPA项目还可以扫描标注@Entity的实体类）**。建议入口类放置的位置在groupId+arctifactID组合的包名下。](mailto:Boot会自动扫描@SpringBootApplication所在类的同级包（如com.wisely.ch5_2_2）以及下级包里的Bean（若为JPA项目还可以扫描标注@Entity的实体类）。建议入口类放置的位置在groupId+arctifactID组合的包名下。)

2.关闭特定的自动配置

关闭特定的自动配置应该使用@SpringBootApplication注解的**exclude参数**，例如：

@SpringBootApplication(exclude ={DataSourceAutoConfiguration.class})

3.Spring Boot的**配置文件**

Spring Boot使用一个全局的配置文件application.properties或application.yml，放置在src/main/resources目录或者**类路径**的/config下。

yaml是以数据为中心的语言，在配置数据的时候具有面向对象的特征。

Spring Boot的全局配置文件的作用是对一些默认配置的配置值进行修改。

(1)例如，将Tomcat的默认端口号8080修改为9090，并将默认的访问路径“/”修改为“/helloboot”。

可以在application.properties中添加：

server.port=9090

server.context-path=/helloboot

或在application.yml中添加：

server:

　 port: 9090

contextPath: /helloboot

从上面的配置可以看出，在Spring Boot中，context-path、contextPath或者CONTEXT\_PATH形式其实是通用的。

Spring Boot提倡零配置，即无xml配置，但是在实际项目中，可能有一些特殊要求你必须使用xml配置，这时我们可以通过Spring提供的@ImportResource来加载xml配置，例如：

@ImportResource({"classpath:some-context.xml","classpath:another-context.xml"})

## 外部配置

Spring Boot允许使用properties文件、yaml文件或者命令行参数作为外部配置。

1.**命令行参数**配置

Spring Boot可以是基于jar包运行的，打成jar包的程序可以直接通过下面命令运行：

java –jar xx.jar

可以通过以下命令修改Tomcat端口号：

*java –jar xx.jar --server.port=9090*

1. **常规属性**配置

在Spring Boot里，我们只需在application.properties定义属性，直接使用**@Value**注入即可。

如：

（1）application.properties增加属性：

book.author=wangyunfei

book.name=spring boot

1. 修改入口类：

package com.wisely.ch5\_2\_2;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

@RestController

@SpringBootApplication

public class Ch522Application {

*@Value("${book.author}")*

　　 private String bookAuthor;

*@Value("${book.name}")*

　　 private String bookName;

　　 @RequestMapping("/")

　　　　 String index() {

　　　　　　 return "book name is:"+bookName+" and book author is:" + bookAuthor;

　　　　 }

public static void main(String[] args) {

　　　　 SpringApplication.run(Ch522Application.class, args);

　　 }

}

1. 运行效果



3.**类型安全**的配置（基于properties）

Spring Boot提供了基于类型安全的配置方式，通过@ConfigurationProperties将properties属性和一个Bean及其属性关联，从而实现类型安全的配置。

如：

（1）添加配置，即在application.properties上添加：

author.name=wyf

author.age=32

除此之外，可以指定另一个properties配置文件，需要在入口类上配置指定路径。

@ConfigurationProperties(prefix = "author")

@PropertySource("classpath:Book.properties")

（2）**类型安全的Bean**：

package com.wisely.ch6\_2\_3.config;

import org.springframework.boot.context.properties.ConfigurationProperties;

@Component

**@ConfigurationProperties**(prefix = "author")

public class AuthorSettings {

　　 private String name;

　　 private Long age;

　　 public String getName() {

　　　　 return name;

　　 }

　　 public void setName(String name) {

　　　　 this.name = name;

　　 }

　　 public Long getAge() {

return age;

　　 }

　　 public void setAge(Long age) {

　　　　 this.age = age;

　　 }

}

①通过@ConfigurationProperties加载properties文件内的配置，通过prefix属性指定properties的配置的前缀，还可以通过locations指定properties文件的位置

（3）检验代码：

package com.wisely.ch6\_2\_3;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import com.wisely.ch6\_2\_3.config.AuthorSettings;

@RestController

@SpringBootApplication

public class Ch623Application {

　　 @Autowired

　　 private AuthorSettings authorSettings; //可以用@Autowired直接注入该配置

　　 @RequestMapping("/")

　　 public String index(){

　　　　 return "author name is "+ authorSettings.getName()+" and author age is "+authorSettings.getAge();

　　 }

　　 public static void main(String[] args) {

　　　　 SpringApplication.run(Ch623Application.class, args);

　　 }

}

1. 效果：



## 日志配置

Spring Boot支持Java Util Logging、Log4J、Log4J2和Logback作为日志框架，默认情况下，Spring Boot使用Logback作为日志框架。

配置**日志文件**：

logging.file=D:/mylog/log.log

配置**日志级别**，格式为logging.level.包名=级别

如：logging.level.org.springframework.web= DEBUG

## Profile配置

Profile是Spring用来针对不同的环境对不同的配置提供支持的，全局Profile配置使用application-{profile}.properties（如application-prod.properties）。

通过在application.properties中设置spring.profiles.active=prod来指定活动的Profile。

下面将做一个最简单的演示，如我们分为生产（prod）和开发（dev）环境，生产环境下端口号为80，开发环境下端口为8888.

（1）生产和开发环境下的配置文件如下：

application-**prod**.properties：

server.port=80

application-**dev**.properties：

server.port=8888

（2）运行。

application.properties增加：**spring.profiles.active**=dev 或者 **spring.profiles.active**=prod