# @VenCano

# SpringBoot使用讲解：

## 1简单介绍：

随着Spring 4新版本的发布，Spring Boot这个新的子项目得到了广泛的关注，Spring Boot充分利用了JavaConfig的配置模式以及“约定优于配置”的理念，能够极大的简化基于Spring MVC的Web应用和REST服务开发。

Spring 4倡导微服务的架构，针对这一理念，近来在微博上也有一些有价值的讨论，如这里和这里。微服务架构倡导将功能拆分到离散的服务中，独立地进行部署，Spring Boot能够很方便地将应用打包成独立可运行的JAR包，因此在开发模式上很契合这一理念。

       现在的web项目几乎都会用到spring框架，而要使用spring难免需要配置大量的xml配置文件，而springboot的出现解决了这一问题，一个项目甚至不用部署到服务器上直接开跑，真像springboot所说：“just run”。

springboot的很多默认编码方式都是utf-8，真是福利啊。

org.spring 2013年新开发的框架springboot , 它让一个单独项目的创建变得更加的简单，让所有依赖spring的程序可以做到“just run”。springboot提供大量第三方libraries让我们可以非常轻松的开始创建一个spring工程，甚至不需要再去配置一些繁琐的xml配置文件；

官网地址：**<http://projects.spring.io/spring-boot/>**

**框架特点：**

    1：创建独立的spring应用。

    2：嵌入Tomcat, Jetty Undertow 而且不需要部署他们。

    3：提供的“starters”poms来简化Maven配置

    4：尽可能自动配置spring应用。

    5：提供生产指标,健壮检查和外部化配置

    6：绝对没有代码生成和XML配置要求

## 2简单的实例

### (1)新建一个maven项目

### (2)配置pom.xml文件：（简单的配置）

|  |
| --- |
| <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 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <groupId>spring.boot.demo</groupId>  <artifactId>springBootDemo</artifactId>  <version>1.0.0-SNAPSHOT</version>  <packaging>war</packaging>  <name>spring.boot.demo</name>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.2.4.RELEASE</version>  </parent>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <!-- 这里一定要配置上java的版本，如果是1.7版本的可不用配置 -->  <java.version>1.7</java.version>  <!-- <tomcat.version>7.0.55</tomcat.version> -->  </properties>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

配置java.version的版本或者添加java1.7的版本插件：

<!-- 默认会出现：Dynamic Web Module 3.0 requires Java 1.6 or newer.bdpline 1 Maven Java EE Configuration Problem类似的错误

maven的编译级别为1.5或者更低，导入与你编译器相同级别的jdk版本即可，见下面的配置：引入jdk1.7版本的依赖-->

<!-- <plugin> -->

<!-- <groupId>org.apache.maven.plugins</groupId> -->

<!-- <artifactId>maven-compiler-plugin</artifactId> -->

<!-- <configuration> -->

<!-- <source>1.7</source> -->

<!-- <target>1.7</target> -->

<!-- </configuration> -->

<!-- </plugin> -->

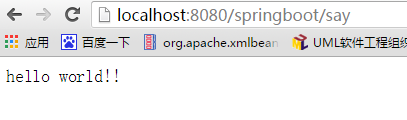
### (3)主入口Application.java

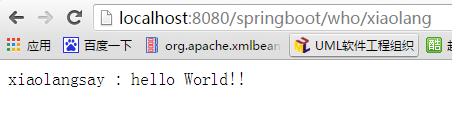
|  |
| --- |
| **package** springBootDemo;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  **public** **class** Application {    **public** **static** **void** main(String[] args) {  SpringApplication.*run*(Application.**class**, args);  }  } |

### (4)业务Action，Controller

|  |
| --- |
| package springBootDemo.action;  import org.springframework.web.bind.annotation.PathVariable;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.ResponseBody;  import org.springframework.web.bind.annotation.RestController;  @RestController  @RequestMapping("springboot")  public class HelloWorldAction {  @RequestMapping("/say")  @ResponseBody  public String sayHelloWorld() {  return "hello world!!";  }    @RequestMapping("/who/{name}")  @ResponseBody  public String whoSayHelloWorld(@PathVariable("name")String name) {  return name+"say : hello World!!";  }  } |

### (5)url请求：





## 具体讲解：

### \*注意事项：

1.开发第一个springboot程序最好使用maven来搭建，文档全程也是maven构建。

2.springboot因为是一个最新开发的框架，所以只支持java6以上，java7最好，官方推荐java8。

3.需要maven3.2以上版本支持。

4. 支持的servlet的类型：

5.官方文档都是java1.7运行的，不配置<java.version>1.6</java.version>的话可能 会报版本异常的错误。

6.如果也不配置tomcat版本的话springboot默认会使用8.x版本的tomcat。所以要加一个

<tomcat.version>7.0.55</tomcat.version>来指定你所使用的tomcat版本(视你CATALINA\_HOME配 置的所定)。



也可以将springboot程序部署到所有支持servlet3.0以上的容器

### 配置文件：

**1.spring-boot-starter-parent**

Springboot的核心，在配置文件中引入，该父依赖；

注意：可以通过parent方式，去继承spring-boot-starter-parent依赖；这样就不用在引入：spring-boot-maven-plugin插件了；

**2.spring-boot-maven-plugin**

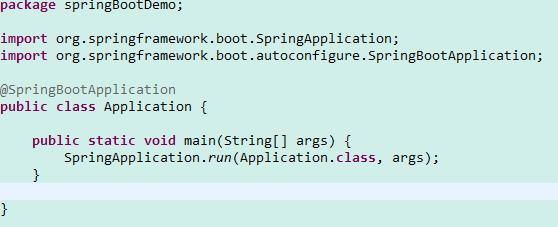
1. **导入web依赖：spring-boot-starter-web**

我们pom中配置了    spring-boot-starter-web所以spring会来创建一 个web应用来配置程序，@EnableAutoConfiguration注解用来自动配置，

  完成后运行main程序无报错则运行成功。在浏览器输入相应的路径即可获得@RequestMapping返回的数据。

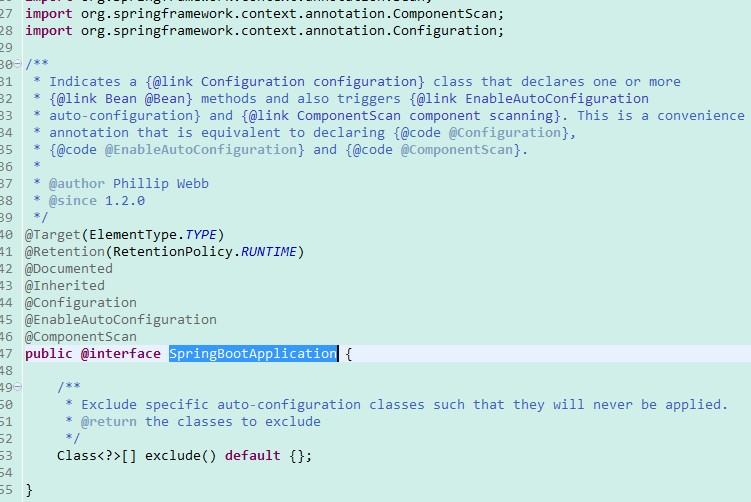
### 程序入口：

**@SpringBootApplication**



**SpringBootApplication的源码：**

@Configuration : 表示Application作为sprig配置文件存在   
@EnableAutoConfiguration: 启动spring boot内置的自动配置   
@ComponentScan : 扫描bean，路径为Application类所在package以及package下的子路径，这里为com.u51.lkl.springboot，在spring boot中bean都放置在该路径已经子路径下。

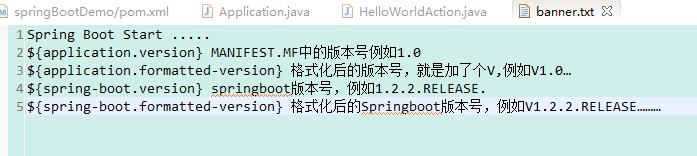


### Console输出：

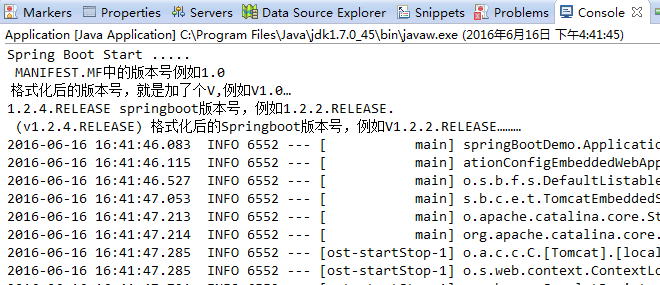
在resource包下新建：banner.txt文件



填充下面内容：



启动打印：



### 监听事件：

1. **创建监听:**

|  |
| --- |
| **public** **class** TestListener **implements** ApplicationListener<ApplicationStartedEvent>{  @Override  **public** **void** onApplicationEvent(ApplicationStartedEvent arg0) {  /\*\*  \* 做监听事件的业务处理  \* 1.ApplicationStartedEvent在初始化的时候发送事件  \* 2.ApplicationEnvironmentPreparedEvent 在创建上下文前发送事件  \* 3.ApplicationPreparedEvent 在应用准备之前，在bean加载之后发送事件  \* 4.ApplicationFailedEvent 启动失败事件发送  \*/  System.*out*.println("我是监听事件！在应用初始化时就发送事件！");  }  } |

1. **自定义启动（加监听）：**

|  |
| --- |
| @SpringBootApplication  **public** **class** Application\_Custom {  **public** **static** **void** main(String[] args) {    /\*\*  \* 自定义一些应用的配置启动(对应用添加监听，入口Action)  \*/  SpringApplication app = **new** SpringApplication(CustomApplication.**class**);  app.addListeners(**new** TestListener());  app.setShowBanner(**false**);  app.run(args);  }  } |

1. **启动入口CustomApplication**

|  |
| --- |
| @RestController  @EnableAutoConfiguration //自动加载配置  **public** **class** CustomApplication {  @RequestMapping("/entry")  **public** String entryHome() {  **return** "welcome to the home of CustomApplication!";  }  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(CustomApplication.**class**, args);  }  } |