# Spring Boot介绍

什么是Spring boot？

Spring Boot是由Pivotal团队提供的全新框架，其设计目的是用来简化新Spring应用的初始搭建以及开发过程。该框架使用了特定的方式来进行配置，从而使开发人员不再需要定义样板化的配置

Spring Boot特性

1. 创建独立的Spring应用程序

2. 嵌入的Tomcat，无需部署WAR文件

3. 简化Maven配置

4. 自动配置Spring

5. 提供生产就绪型功能，如指标，健康检查和外部配置

6.开箱即用，没有代码生成，也无需XML配置。

Spring Boot特性理解

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特性，如嵌入式服务器、安全、指标，健康检测、外部配置等。

Spring Boot并不是对Spring功能上的增强，而是提供了一种快速使用Spring的方式。

# 创建工程步骤

官网下载带springboot的Eclipse地址：https://spring.io/tools3/sts/all 选择windows版本 下载zip（它就是一个eclipse版本）

## Eclipse开发springboot

第一步 创建一个普通的maven工程

就是和之前的操作不变

第二步 pom.xml的设置

<!-- spring boot 父节点依赖,引入这个之后相关的引入就不需要添加version配置，spring boot会自动选择最合适的版本进行添加。-->

<parent>

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

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

<version>1.4.1.RELEASE</version>

</parent>

<!-- 指定一下jdk的版本 ，这里我们使用jdk 1.8 ,默认是1.6 -->

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

<!-- spring-boot-starter-web: 等于加入了webMVC,AOP等spring基本依赖包....-->

<dependency>

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

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

<!-- <version></version> 由于我们在上面指定了 parent(spring boot) 不用配版本号-->

</dependency>

</dependencies>

<build>

<plugins>

<!-- 这是spring boot devtool plugin -->

<plugin>

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

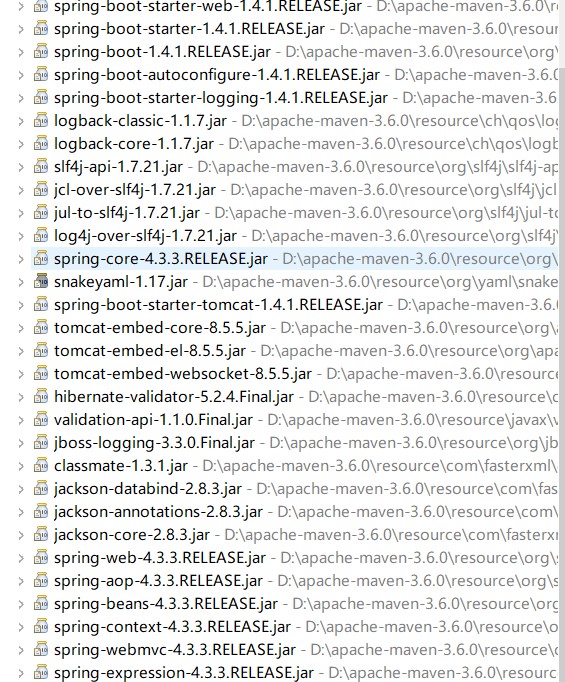
<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

加载的java如下



第三步 在同包目录下创建一个App.jar和UserController.java

UserController.java内容：主要是springmvc的Controller类

@RestController //等同于 Controller和ResponseBody的结合体，spring版本4.3以后支持

public class UserController {

@RequestMapping("nihao")

public String test() {

return "成功"; //这里作用就是响应客户端text文本信息了

}

}

页面显示：成功 //不知道为啥它不是乱码。Springmvc没有配置ContentType就是乱码，应该底层变成了utf-8了

App.jar此名字都可以可以随便取的，这里的内容重要，在这里它就是一个启动口

1、@SpringBootApplication 指定springboot是一个应用程序

2、SpringApplication.run(App.class, args);：这样就可以使用普通的java启动程序直接运行工程

/\*\*

\* 我们使用@SpringBootApplication指定这是一个spring boot的应用程序

\* @author WWL

\*

\*/

@SpringBootApplication

public class App {

//在main方法中进行启动我们的应用程序

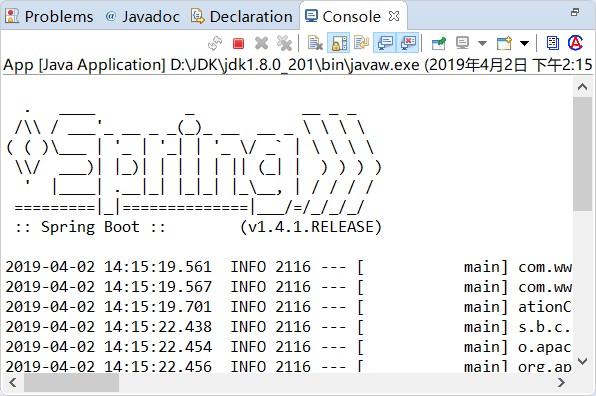
public static void main(String[] args) {

SpringApplication.run(App.class, args);

}

}

第四步 启动工程：在App.java页面直接右键 运行普通工程即可



# 热部署

在编写代码的时候，你会发现我们只是简单把打印信息改变了，就需要重新部署，如果是这样的编码方式，那么我们估计一天下来就真的是打几个Hello World就下班了。那么如何解决热部署的问题呢？那就是springloaded

使用方式

第一步 在pom.xml文件添加依赖包：

<plugin>

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

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

<dependencies>

<!--springloaded hot deploy -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>springloaded</artifactId>

<version>1.2.4.RELEASE</version>

</dependency>

</dependencies>

<executions>

<execution>

<goals>

<goal>repackage</goal>

</goals>

<configuration>

<classifier>exec</classifier>

</configuration>

</execution>

</executions>

</plugin>

第二步 启动方式第一种 使用maven工程启动 注意 此方式会造成关闭不了服务进程，要手动在任务管理器进行关闭(已测试)

命令由tomcat7:run变成spring-boot:run即可

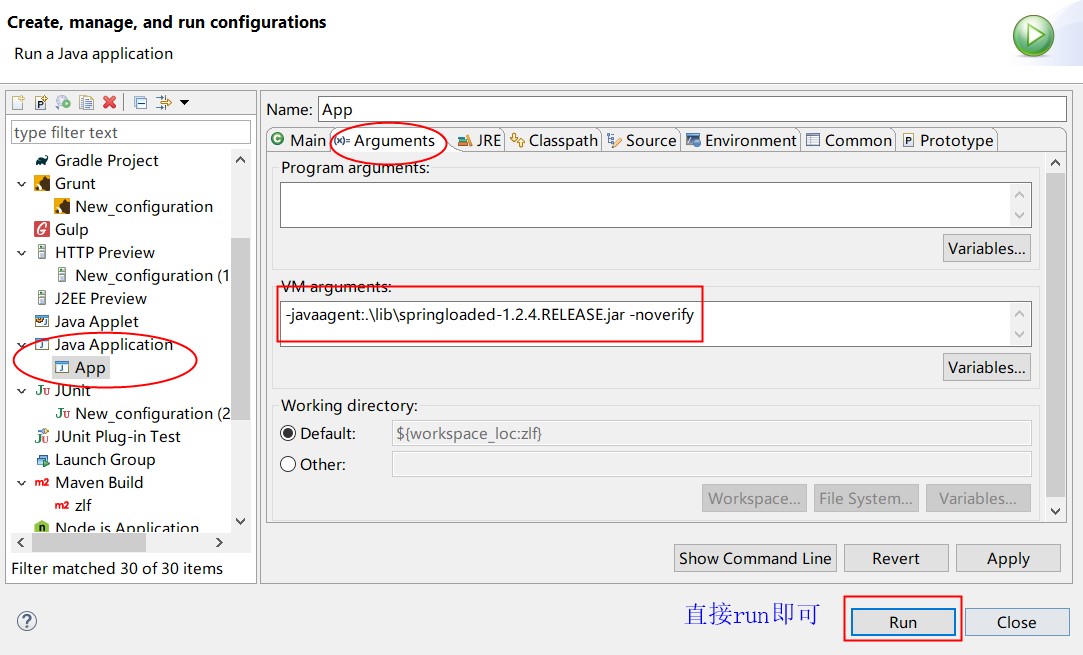
题外话：这样启动又会造成乱码，比如json数据没有配置contentType：application/json;charset=utf-8,返回浏览器又变成了乱码

第三步 启动方式第二种 使用java 这个没有后遗症

1、如果使用的run as – java application（其实就是普通工程的启动方式）的话，那么还需要做一些处理。

2、把spring-loader-1.2.4.RELEASE.jar下载下来，放到项目的lib（没有自己创建）目录中，然后把IDEA的run参数里VM参数设置为：

-javaagent:.\lib\springloaded-1.2.4.RELEASE.jar –noverify



3、然后启动就可以了，这样在run as的时候，也能进行热部署

# Json数据

其实看创建工程步骤完后：可以看到依赖进来很多jar包其中就有jackson给依赖进来了，

Pom.xml中会引入此jackson的配置

<!-- spring-boot-starter-web: webMVC,AOP等spring基本依赖包.... -->

<dependency>

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

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

</dependency>

Springboot底层已配置好加载jackson类，直接用即可

@RestController //等同于 Controller和ResponseBody的结合体，spring版本4.3以后支持

public class UserController {

@RequestMapping("ni")

public Demo testJson() {

Demo demo= new Demo(1, "@RestController就已实现ResponseBody，spring底层配置jackson类");

return demo; //

}

我们需要使用第三方的json解析框架：FastJson解析JSON数据

1. 引入FastJson包

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>fastjson</artifactId>

<version>1.2.15</version>

</dependency>

这里要说下很重要的话，官方文档说的1.2.10以后，会有两个方法支持HttpMessageconvert，一个是FastJsonHttpMessageConverter，支持4.2以下的版本，一个是FastJsonHttpMessageConverter4支持4.2以上的版本，具体有什么区别暂时没有深入研究。这里也就是说：低版本的就不支持了，所以这里最低要求就是1.2.10+。

有两种方式

第一种方法就是：

（1）启动类继承extends WebMvcConfigurerAdapter

（2）覆盖方法configureMessageConverters

@SpringBootApplication

public class App extends WebMvcConfigurerAdapter{

// 1、需要先定义一个 convert 转换消息的对象;

FastJsonHttpMessageConverter fastConverter = new FastJsonHttpMessageConverter();

//2、添加fastJson 的配置信息，比如：是否要格式化返回的json数据;

FastJsonConfig fastJsonConfig = new FastJsonConfig();

fastJsonConfig.setSerializerFeatures(

SerializerFeature.PrettyFormat

);

//3、在convert中添加配置信息.

fastConverter.setFastJsonConfig(fastJsonConfig);

//4、将convert添加到converters当中.

converters.add(fastConverter);

}

第二种方法 还是在启动类中进行操作

（1）在App.java启动类中，

注入Bean : HttpMessageConverters

/\*\*

\* 在这里我们使用 @Bean注入 fastJsonHttpMessageConvert

\* @return

\*/

@Bean

public HttpMessageConverters fastJsonHttpMessageConverters() {

// 1、需要先定义一个 convert 转换消息的对象;

FastJsonHttpMessageConverter fastConverter = new FastJsonHttpMessageConverter();

//2、添加fastJson 的配置信息，比如：是否要格式化返回的json数据;

FastJsonConfig fastJsonConfig = new FastJsonConfig();

fastJsonConfig.setSerializerFeatures(SerializerFeature.PrettyFormat);

//3、在convert中添加配置信息.

fastConverter.setFastJsonConfig(fastJsonConfig);

HttpMessageConverter<?> converter = fastConverter;

return new HttpMessageConverters(converter);

}