Spring Boot

1. Spring Boot简介

1.1.简介

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 的作用在于创建和启动新的基于 Spring 框架的项目。它的目的是帮助开发人员很容易的创建出独立运行和产品级别的基于 Spring 框架的应用。Spring Boot 会选择最适合的 Spring 子项目和第三方开源库进行整合。大部分 Spring Boot 应用只需要非常少的配置就可以快速运行起来。

1.2.Spring Boot特性

创建可以独立运行的 Spring 应用。

直接嵌入 Tomcat 或 Jetty 服务器，不需要部署 WAR 文件。

提供推荐的基础 POM 文件来简化 Apache Maven 配置。

尽可能的根据项目依赖来自动配置 Spring 框架。

提供可以直接在生产环境中使用的功能，如性能指标、应用信息和应用健康检查。

没有代码生成，也没有 XML 配置文件。

1.3.开发环境配置

如果使用Eclipse，可以安装Spring Tool Suite插件，详情参考：<http://www.roncoo.com/article/detail/124661>

或者直接下载Spring Tool Suite开发工具。

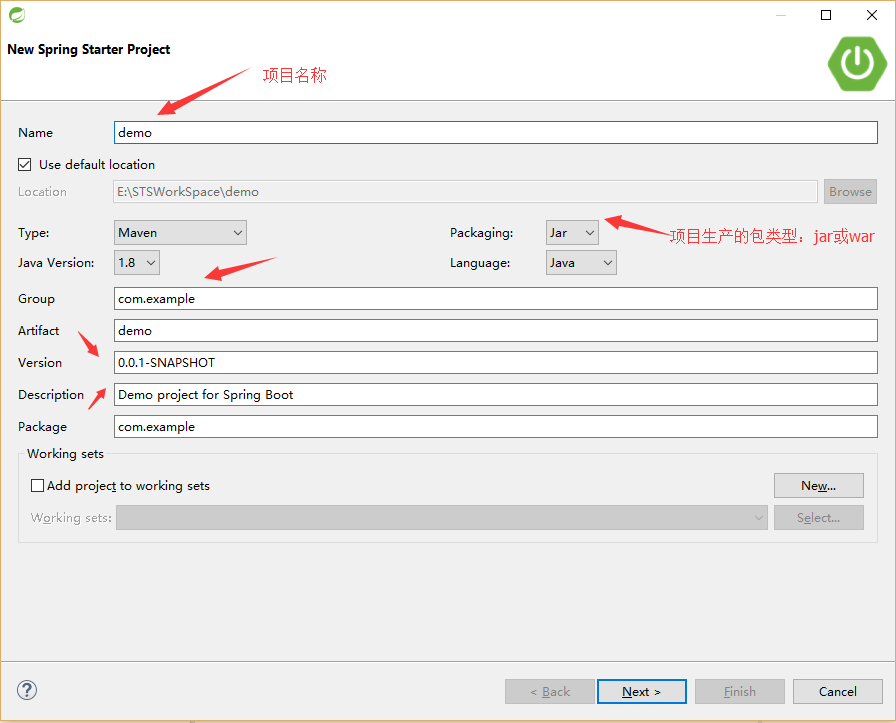
二．Hello World

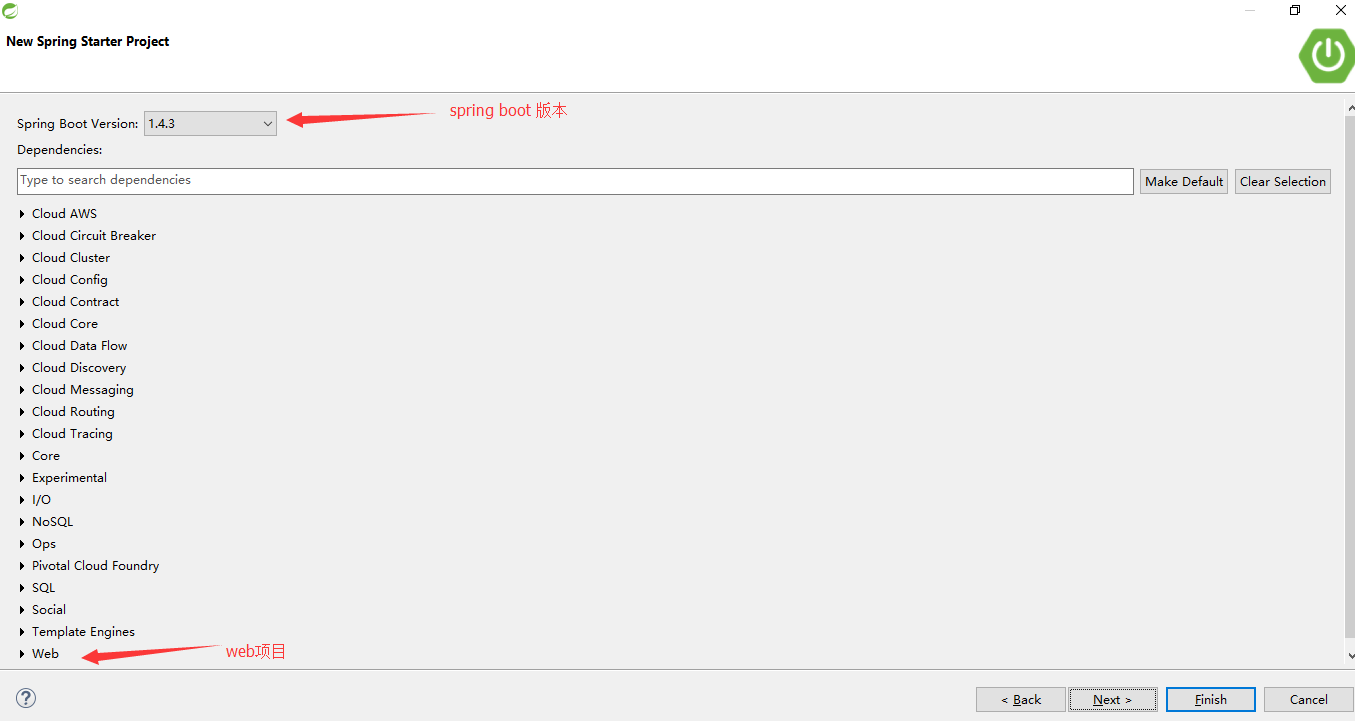
使用工具：Spring Tools Suite

下载地址：<http://spring.io/tools>

2.1.项目创建

File->New->Spring Start Project



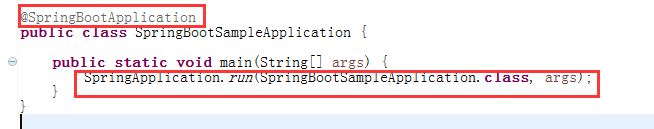


2.2.pom.xml文件

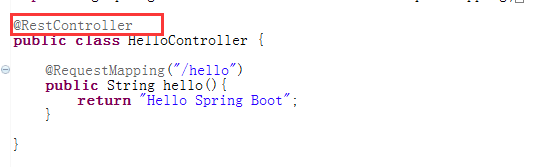


默认继承的父项目是可以修改的。

2.3.启动类

Spring Boot Application注解表明这个程序是个Spring Boot程序。通过Spring Application的run方法启动程序。

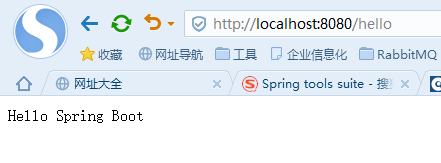
2.4 Hello Controller



Rest Controller是继承了Controller和ResponseBody注解，默认会返回json数据。所有标有Spring注解的类所在的包都必须是程序启动类所在包的子包(或同级)。

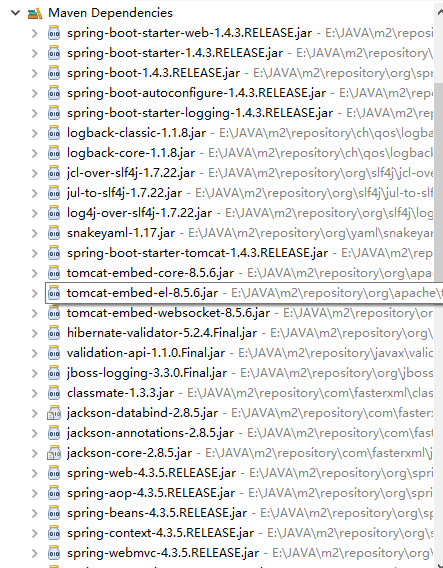
2.5.运行程序

点击项目，右键Run As->Spring Boot App则可以启动项目，在浏览器上访问: <http://localhost:8080/hello>



会启动默认的内嵌容器。

2.6.项目依赖说明



展开maven依赖节点，可以看到默认依赖了Spring MVC,Jackson,tomcat容器等。这是Spring官方推荐的依赖。

Spring Boot三种启动方式

@EnableAutoConfiguration  
@ComponentScan  
@Configuration  
public class App   
{  
 public static void main( String[] args )  
 {  
 // 运行SpringApplication  
 // SpringApplication.run(App.class,args);  
  
 */\*\*  
 \* 自定义SpringBootApplication  
 \* <br/> 传递给 SpringApplication 的构造器参数将作为spring beans的配置源  
 \*/*// SpringApplication app = new SpringApplication(App.class);  
// app.setBannerMode(Banner.Mode.OFF);  
// app.setWebEnvironment(true);  
// app.run(args);  
  
 */\*\*  
 \* 流式操作  
 \*/* new SpringApplicationBuilder()  
 .sources(App.class) // 父上下文  
 .bannerMode(Banner.Mode.*OFF*)  
 .web(true)  
 .run(args);  
 }  
}

三．Spring Boot外部配置介绍

可以使用properties文件、Yaml文件、环境变量、命令行参数来为Spring Boot应用提供外部配置。

Spring Boot按照下面方式对属性进行设值：

1) 命令行参数

2) spring.application.json

3) ServletConfig 初始化参数

4) ServletContext 初始化参数

5) Java系统属性( -D 参数)

6) 操作系统环境变量

7) 没有打进jar包application.properties/Yaml变量

8) 打进jar包application.properties/Yaml变量

9) @Configuration 类上的 @PropertySource 注解

10) 默认属性（使用 SpringApplication.setDefaultProperties 指定）

命令行参数

默认情况下， SpringApplication 会将所有命令行配置参数（以'--'开头，比如 --server.port=9000 ） 转化成一个 property ，并将其添加到Spring Environment 中。正如上面提过的，命令行属性总是优先于其他属性源。

如果不想将命令行属性添加到 Environment ，你可以使用

SpringApplication.setAddCommandLineProperties(false) 来禁用它们。

Remark: Spring的Environment可以方便的访问property属性，包含系统属性，环境变量和自定义的。

3.1.application.properties：属性文件

Spring Boot默认支持properties和yaml两种格式的属性文件，优先取properties类型的文件。配置文件在下面的位置，优先级从高到低：

1) 当前目录的config目录下

2) 当前目录

3) classpath路径下的config目录

4) classpath根路径。

配置文件的默认名字是：application.properties，可以通过指定 spring.config.name 环境属性来切换其他的名称，也可以使用 spring.config.location 环境属性引用一个明确的路径。上面两个属性应该配置成系统环境变量、系统属性或命令行参数。

3.2 属性占位符

使用 application.properties 定义的属性时，可以引用事先定义的值，如：

app.name=MyApp  
app.description=${app.name} is a Spring Boot Application

3.3 使用Yaml替代Properties

YAML是Json的超集，只要将SnakeYAML库放到classpath下，SpringApplication就会自动支持YAML，以替代properties文件。

添加spring-boot-starter会自动加载SnakeYAML依赖。

四、Profiles

Spring Profiles提供了一种隔离应用程序配置的方式，并让这些配置只在特定的环

境下生效。任何 @Component 或 @Configuration 都能注解 @Profile ，从而限制加载它的时机。

可以通过spring.profiles.active指定哪个配置生效。

五、日志 Logging

默认使用的是slf4j + logback进行日志记录，记录的日志样式为：

[日期和时间 - 精确到毫秒] [日志级别] [Process ID] --- [线程名] [日志名] [日志信息]

输出的日志样例：

2018-01-19 10:44:57.153 INFO 2576 --- [restartedMain] org.apache.catalina.core.StandardEngine : Starting Servlet Engine: Apache Tomcat/8.0.33

1.application.properties关于日志的相关配置

六、Spring Boot程序发布

<http://blog.csdn.net/senpogml/article/details/52366518>

<http://blog.csdn.net/screengost/article/details/50506830>

<http://www.cnblogs.com/f-zhao/p/6929814.html>

63.Spring Boot Maven插件

Spring Boot Maven插件为Maven提供Spring Boot支持，允许将应用打包成可执行jar或war，然后就地允许应用。需要Maven 3.2及以上版本支持。

1.简单使用

<plugins>

<plugin>

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

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

<version> 1.5.0.RELEASE </version>

<executions>

<execution>

<goals>

<goal>repackage</goal>  
</goals>

</execution>  
</executions>

</plugin>  
</plugins>

上面配置会在Maven生命周期的 package 阶段重新打包一个jar或war，会把原始的jar包备份成.original后缀的文件。

2. 目标：Goals

Spring Boot Maven插件提供了5个Goal，分别是：

spring-boot:repackage默认goal。在mvn package之后，再次打包可执行的jar/war，同时保留mvn package生成的jar/war为.origin。

spring-boot:run运行Spring Boot应用

spring-boot:start在mvn integration-test阶段，进行Spring Boot应用生命周期的管理

spring-boot:stop在mvn integration-test阶段，进行Spring Boot应用生命周期的管理

spring-boot:build-info生成Actuator使用的构建信息文件build-info.properties