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。

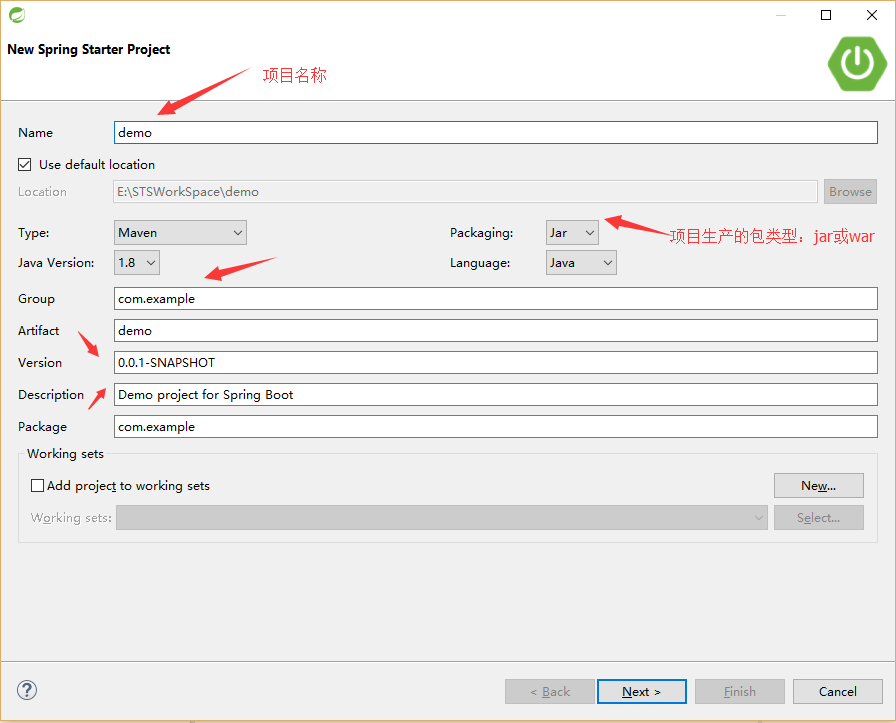
二．Hello World

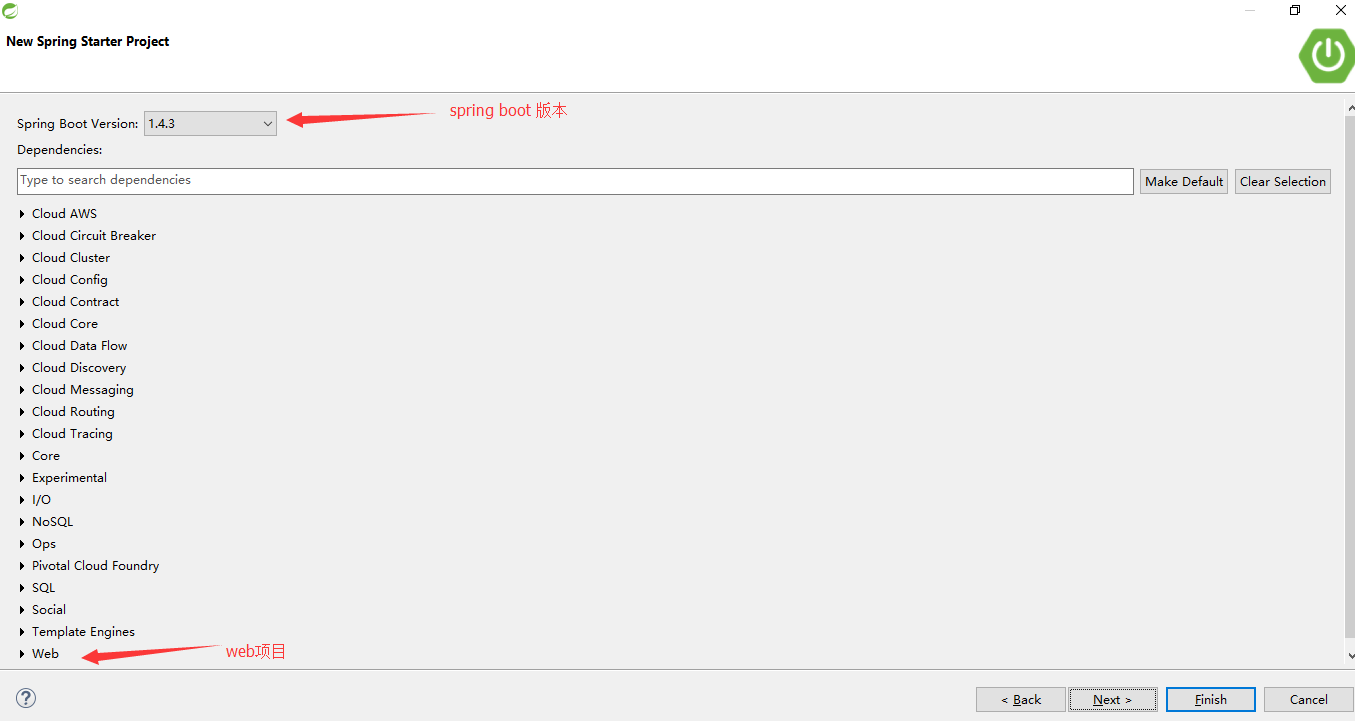
使用工具：Spring Tools Suite

下载地址：<http://spring.io/tools>

2.1.项目创建

File->New->Spring Start Project



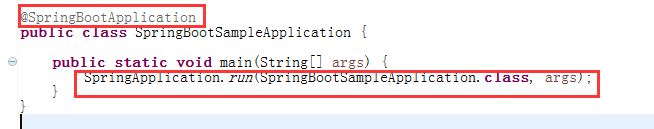


2.2.pom.xml文件

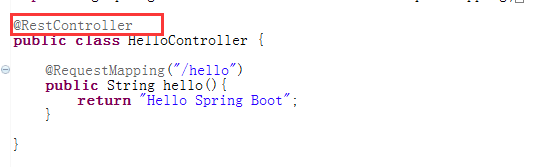


默认继承的父项目是可以修改的。

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。

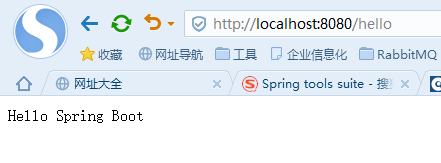
2.4 Hello Controller



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)。

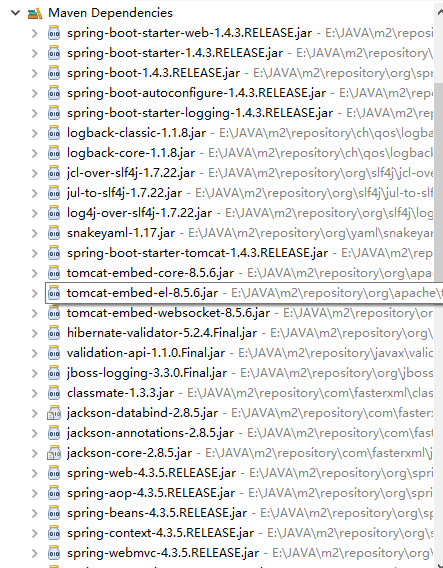
2.5.运行程序

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。

3.4 读取properties配置

若是将配置项配置在application.properties文件中，在程序中可以通过@Value(**"${配置项}"**)将值注入进去，当配置项很多的时候，通过@Value进行注入的方式会很麻烦，这个时候可以将配置项绑定到实体，如:

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

新建对应的配置信息类AppInfo,通过@ConfigurationProperties注解进行配置的处理：

@ConfigurationProperties(prefix = **"app"**)  
**public class** AppInfo {  
 */\*\*  
 \* App的名称  
 \*/* **private** String **name**;  
 */\*\*  
 \* App的描述  
 \*/* **private** String **description**;

}

其中prefix指定配置的前缀，此处为app.APPInfo配置类的属性名与前缀通过.连接后构成配置项的名称。

此外，为了使该配置类能正常使用，需要在配置类开启对该类的扫描：

@EnableConfigurationProperties(value = { AppInfo.**class** })

这样，就可以在需要使用的地方，通过@Autowired进行注入：

@Autowired  
**private** AppInfo **appInfo**;

Remark：@EnableConfigurationProperties注解将指定的value属性的值配置成一个bean，这样才能被Spring容器所管理，但是，当我们有很多配置类的时候，这种写法比较麻烦并且耦合度较高，这个时候可以通过在配置类添加@Component注解，进行自动扫描。

此外，还需要引入spring-boot-configuration-processor依赖

很多时候，为了配置模块化，需要将一些相同的信息单独配置在一个properties文件中，这个时候，如果通过配置类的方式，需要加@PropertySource指定配置文件，如：

@Component  
@ConfigurationProperties  
@PropertySource(value = **"test.properties"**)  
**public class** Company {  
 **private** String **company**;  
 **private int year**;  
 **private** String **address**;

}

test.properties的内容为：

**company**=**gl  
year**=**2017  
address**=**guangzhou**

Remark：默认Spring Boot程序只会加载application.properties属性文件，如需要加载其他配置文件，需要通过@PropertySource注解进行引入。这样，不管是@value注解还是配置类，就都能正确解析了。

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。

*# 最佳实践：  
# 在application.properties中配置通用内容  
# application-{profile}.properties配置各个环境不同的内容  
# 通过命令行方式去激活不同环境的配置：--spring.profiles.active=*

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

*# 日志文件每达到10M就会进行分割，  
# 配置日志级别，默认记录ERROR/WARN/INFO日志***logging.level.root**=**INFO***# 配置指定日志记录器的日志级别***logging.level.org.springframework.web**=**DEBUG***# 记录到具体的文件  
# logging.file=D:\\logs\\springboot-quickstart.log  
  
# 记录到具体目录下的spring.log文件  
# logging.path=D:\\logs  
  
# 指定日志输出样式  
# 默认样式为:[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]***logging.pattern.console**=**%d{YYYY-MM-DD HH:mm:ss.SSS} - [%t] %-5level [%logger{36}:%line] - %msg%n  
logging.pattern.file**=**%d{HH:mm:ss.SSS} - [%t] %-5level [%logger{36}:%line] - %msg%n***# 自定义日志配置  
# 由于日志初始化早于ApplicationContext创建  
# 自定义的配置需要通过系统属性或命令行等方式指定*

2.自定义配置文件

默认使用logback类库进行日志记录，所以可以通过在classpath编写logback.xml或logback-spring.xml(官方推荐)，

Spring Boot官方在Spring-boot-{Version}.RELEASE.jar的

org.springframework.boot.logging.\*包下提供了一些默认配置文件，如logback的base.xml/defaults.xml/console-appender.xml/file-appender.xml

通过在application.properties配置文件通过属性**logging.config**指定自定义配置文件。

3.更换日志框架

spring-boot-starter默认集成了spring-boot-starter-logging依赖，该依赖集成了logback日志框架，若要更换logback，需要先将其排除掉，然后再集成需要的日志框架，如log4j2：

*<!-- 排除默认的logback日志框架 -->*<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter</**artifactId**>  
 <**exclusions**>  
 <**exclusion**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-logging</**artifactId**>  
 </**exclusion**>  
 </**exclusions**>  
</**dependency**>  
*<!-- log4j2日志框架 -->*<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-log4j2</**artifactId**>  
</**dependency**>

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

使用Spring Boot Maven插件生成的jar包，会将外部依赖也打包在其中，导致更新的时候需要上传很多额外的东西，对于依赖没有更改的情况下，每次更新程序还需要更新自己的代码程序即可，可以使用其他maven插件将依赖打包在外面，然后自己的程序单独成jar，每次只需要更新自己的程序jar即可。