# Springboot

## SpringBoot入门

### springboot简介

长期以来，java的开发一直让人所诟病：

* Java项目开发复杂度极高
* Java项目维护困难
* 在云时代如何实现项目的快速部署以及快速启动
* 即便使用了大量的开发框架，发现我们的开发也没有少多少
* 当所有人认为spring不再前进的时候，spring推出了微架构实现的两个重要开发框架：SpringBoot、SpingCloud。

1. java开发复杂度是最高的？

* Java里面提供的开发支持都属于原生操作的代码，例如：JDBC为例，如果使用java的原生代码，会重复编写大量的内容。例如：prepareStatement操作。
* Java进行Web项目开发的时候，必需要求按照严格的格式进行WEB项目的创建，以及每当修改web程序的时候，有需要进行tomcat的重新启动；
* Java中虽然提供了开发标准，但是所有的公司几乎都有可能有自己的标准，比如最初时代，jvm的标准就有三个，而且许多公司，由于版本的不同，会造成部署环境的不同；
* Java严格要求使用mvc设计模式完成
* 以web开发为例，jsp程序里面不应该包含有任何的Script程序代码。但是要做到这步会非常麻烦，有各种的实现标准，例如：JSTL+EL、SpringTaglib、StrutsTaglib、JSF
* 如果现在使用Node.js、Python开发一个控制器程序类可能只需要几行代码，而java

需要写一堆的代码，而且要部署。

1. java后期发展使用了大量的Maven技术作为开发，会发现传统的开发处理中并没有逃离掉web的身影，所有的项目需要打包成war文件，而后上传到系统中。使用maven最大的痛是：如果是开发框架，那一堆的maven配置依赖库。
2. Rest技术开始在行业中广为流传，而java要想实现Rest架构的开发（基于：Spring），那么也是相当麻烦。
3. 现在行业中，spring已经作为绝对的java架构，但是如果要想在Spring中整合RabbitMQ、KafKa、ActiveMQ、Druid、Redis、Shiro,需要编写一堆的.xml配置文件。

所以在这样的历史背景下，很多人开始寻求更加简便的开发，而遗憾的是这种简便的开发没有被JDK所支持、被JavaEE所支持，因为这些知识平台，平台能够提供的只是最原始的技术支持。这一时刻终于由于Spring框架的升级而得到了提升，SpringBoot的出现，改变了java开发的困境，SpringBoot的最终奉行的宗旨是：废除掉所有复杂的开发，废除掉所有的配置文件。让开发变得更简单纯粹。

### Springboot快速入门

1. 如果要想开发SpringBoot程序，只需要按照管方的要求配置一个父pom即可。

<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>

<parent>

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

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

<version>1.5.4.RELEASE</version>

</parent>

<groupId>cn.mldn</groupId>

<artifactId>bootfirst</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>bootfirst</name>

<url>http://maven.apache.org</url>

<properties>

<jdk.version>1.8</jdk.version>

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

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<scope>test</scope>

</dependency>

<dependency>

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

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

</dependency>

</dependencies>

<build>

<finalName>bootfirst</finalName>

<plugins>

<plugin>

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

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

<configuration>

<source>${jdk.version}</source><!-- 源代码使用的开发版本 -->

<target>${jdk.version}</target><!-- 需要生成的目标class文件的编译版本 -->

<encode>${project.build.sourceEncoding}</encode>

</configuration>

</plugin>

</plugins>

</build>

</project>

1. 编写一个具体的程序：

@Controller  
@EnableAutoConfiguration  
**public class** SampleController {  
 @RequestMapping(**"/"**)  
 @ResponseBody  
 **public** String home(){  
  
 **return "杨幂"**;  
 }  
  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(SampleController.**class**, args);  
 }  
}

直接运行java程序“Spring Boot APP”。

如果是一个Maven的普通项目，最简单的做法是在maven运行输入“Spring-boot:run”。

## SpringBoot基础知识

### 建立统一父pom

1. 建立一个microboot的maven项目

*<?***xml version="1.0" encoding="UTF-8"***?>*<**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**>cn.cn.ym.com</**groupId**>  
 <**artifactId**>microboot</**artifactId**>  
 <**packaging**>pom</**packaging**>  
 <**version**>1.0-SNAPSHOT</**version**>  
  
 <**properties**>  
 <**java.version**>1.8</**java.version**>  
 <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  
 </**properties**>  
 <**dependencyManagement**>  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-dependencies</**artifactId**>  
 <**version**>2.0.2.RELEASE</**version**>  
 <**type**>pom</**type**>  
 <**scope**>import</**scope**>  
 </**dependency**>  
 </**dependencies**>  
 </**dependencyManagement**>  
 <**build**>  
 <**finalName**>microboot</**finalName**>  
 <**plugins**>  
 <**plugin**>  
 <**groupId**>org.apache.maven.plugins</**groupId**>  
 <**artifactId**>maven-compiler-plugin</**artifactId**>  
 <**configuration**>  
 <**source**>${java.version}</**source**>*<!-- 源代码使用的开发版本 -->* <**target**>${java.version}</**target**>*<!-- 需要生成的目标class文件的编译版本 -->* <**encoding**>${project.build.sourceEncoding}</**encoding**>  
 </**configuration**>  
 </**plugin**>  
 </**plugins**>  
 </**build**>  
 <**modules**>  
 <**module**>microboot-base</**module**>  
 <**module**>microboot-base</**module**>  
 </**modules**>  
</**project**>

1. 建立microboot-base的子模块，实现之前同样的操作功能

*<?***xml version="1.0" encoding="UTF-8"***?>*<**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"**>  
 <**parent**>  
 <**artifactId**>microboot</**artifactId**>  
 <**groupId**>cn.ym.com</**groupId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
 </**parent**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
  
 <**artifactId**>microboot-base</**artifactId**>  
  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-web</**artifactId**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>junit</**groupId**>  
 <**artifactId**>junit</**artifactId**>  
 <**scope**>test</**scope**>  
 </**dependency**>  
 </**dependencies**>  
</**project**>

### SpringBoot代码测试

1、【修改microboot-base模块】修改pom.xml配置文件，追加SpringBoot测试支持类

<dependency>

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

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

<scope>test</scope>

</dependency>

2、【修改microboot-base模块】建立一个测试程序类

@SpringBootTest(classes = SampleController.**class**)  
@RunWith(SpringJUnit4ClassRunner.**class**)  
@WebAppConfiguration  
**public class** TestSampleController {  
 @Autowired  
 **private** SampleController **sampleController**;  
  
 @Test  
 **public void** home() {  
 TestCase.*assertEquals*(**this**.**sampleController**.home(), **"www.ym.cn"**);  
 }  
}

### SpringBoot启动注解分析

|  |  |  |
| --- | --- | --- |
| No | 注解 | 说明 |
| 1 | @Controller | 进行控制器的配置注解，这个注解所在的类就是控制器类 |
| 2 | @EnableAutoConfiguration | 开启自动配置处理 |
| 3 | @RequestMapping("/") | 表示访问路径，此时的路径为“/”,访问地址：“http://localhost:8080/” |
| 4 | @ResponseBody | 在Restful架构中，该注解表示直接将返回的数据以字符串或json的形式获得 |

在给出的几个直接中，“EnableAutoConfiguration”为整个SpringBoot的启动注解配置，也就是说这个注解应该随着程序的主类一起定义。

而对于控制器程序类，由于项目中会有许多的控制器，那么最好将这些类统一保存在一个包中。

下面将所有的程序类保存在“xxx.comtroller”中，是”xxx”的子包，强烈建议（Spring官方建议）：如果要想进行简单方便的开发，所有的程序类一定要在启动类所在包的子包下。

1. 【microboot-base】模块建立一个HelloController程序类：

@Controller

public class HelloController {

@RequestMapping("/")

@ResponseBody

public String home() {

return "www.mldn.cn";

}

}

1. 【microboot-base】模块启动程序主类：

@SpringBootApplication // 启动SpringBoot程序，而后自带子包扫描

public class SampleController {

public static void main(String[] args) throws Exception {

SpringApplication.run(SampleController.class, args);

}

}

1. 【microboot-base】以上的做法只是传统程序的开发做法，因为现在为止毕竟是两个程序类，其彼此之间的联系需要有一个连接点，程序中配置的“@ComponentScan”注解就是负责这个连接处理。但是SpringBoot考虑到了此类的配置问题，提出了一个更简化策略，该策略核心思想：既然程序主类会在所有开发包的父包里面，那么能不能简化点取得配置呢？为此在实际开发中，会使用一个特殊的符合注解：@SpringBootApplication（启动SpringBoot程序，而后自带**子包**扫描）。

如果开发的程序类和启动主类不在一个子包下，那么就启动类必须加上“@ComponentScan("cn.ym.com")”。

@SpringBootApplication=@EnableAutoConfiguration+@ComponentScan+其他配置。正式因为它有这样的特点，所以当以后使用Bean实现配置处理的时候将会非常的容易。

注意：程序类和启动类在一个包下不行，必须是在启动类所在的子包下：

### 路径访问控制

在实际开发中，控制器的路径可能会有许多个，而且在进行控制器编写的时候也会有两种运行模式：跳转配置、Restful显示。那么下面来观察关于路径的详细描述。

1. 在之前所编写的控制器里面有如下两种注解配置使用：

* @Controller在类上定义表示的是一个控制器；
* @ResponseBody：将控制器中方法的返回值变为rest内容。如果不加这个注解，表示发挥到“”内的页面。

@RequestMapping(**"/"**)  
**public** String home() {  
 **return "www.baidu.cn"**;  
}

访问后跳转项目中“www.baidu.com”这个页面，但是项目中是没有这个页面。

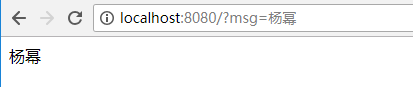
但是一个项目里面可能控制器之中返回的全部是是restful信息，这样分别定义就态麻烦了，为此在SpringBoot里面又提供一个符合注解：@RestController。

@RestController  
**public class** HelloController {  
 @RequestMapping(**"/"**)  
 **public** String home() {  
 **return "www.baidu.cn"**;  
 }  
}

从MVC的实际标准来讲，控制器需要传递一些属性到页面显示，按照这样的原则并不是所有的开发都会以rest的结构返回，但是rest结构是SpringCloud的实现核心技术。

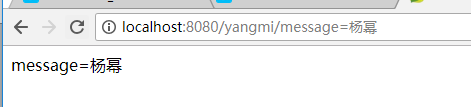
2、现在所给出的控制器类里面只是负责了简单的信息返回，那么实际上也可以进行参数的接收处理，传递参数到控制器中最简单的做法是使用地址重写传递“xx?参数名称=内容”。

@ResponseBody  
@SpringBootApplication  
**public class** SampleController {  
 **public static void** main(String[] args) **throws** Exception {  
 SpringApplication.*run*(SampleController.**class**, args);  
 }  
}



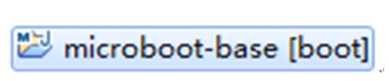
3、由于Springboot支持rest分割处理，所以此时对于参数的接收可以采用路径参数的形式完成。

@ResponseBody  
@SpringBootApplication  
**public class** SampleController {  
 **public static void** main(String[] args) **throws** Exception {  
 SpringApplication.*run*(SampleController.**class**, args);  
 }  
}



4、在springboot里面，可以追加自动加载配置的依赖。

* 如果项目只是一个简单的springBoot的配置项目，则显示的项目信息如下：



* 【mircoo-base模块】如果现在添加有如下的两个依赖库：

<dependency>

<groupId>org.springframework</groupId>

<artifactId>springloaded</artifactId>

</dependency>

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

</dependency>



这就表示该项目重新启动后以后可以自动加载。

### 使用内置对象

在springboot中控制器的形式和SpringMVC是一样的，所以要想使用jsp内置对象，也可以按照springmvc的方式进行。

@RequestMapping("/object")

public String object(HttpServletRequest request,HttpServletResponse response) {

System.out.println("\*\*\* 客户端IP地址：" + request.getRemoteAddr());

System.out.println("\*\*\* 取得客户端响应编码：" + response.getCharacterEncoding());

System.out.println("\*\*\* 取得SessionID：" + request.getSession().getId());

System.out.println("\*\*\* 取得真实路径：" + request.getServletContext().getRealPath("/upload/"));

return "www.mldn.cn" ;

}

### 项目打包

在项目中配置好插件，以及打包就可以执行了，并且这个执行不需要特别复杂的配置。1、【microo项目】修改Pom.xml,追加新的插件：

<plugin> <!-- 该插件的主要功能是进行项目的打包发布处理 -->

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

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

<configuration> <!-- 设置程序执行的主类 -->

<mainClass>cn.mldn.microboot.StartSpringBootMain</mainClass>

</configuration>

<executions>

<execution>

<goals>

<goal>repackage</goal>

</goals>

</execution>

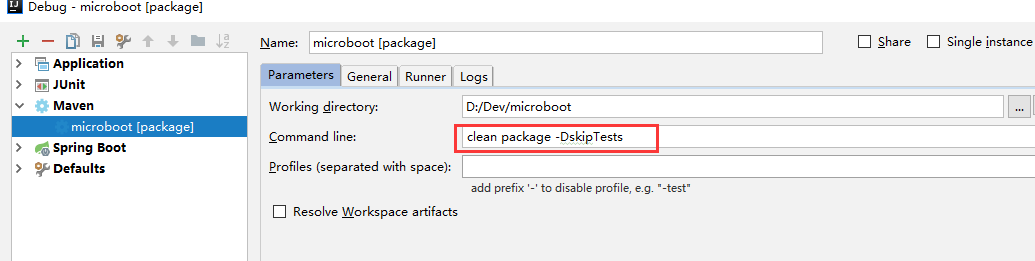
</executions>

</plugin>

</plugins>

这样每一个模块都可以使用这个配置，如果每个模块不一样则单独配置即可

1. 【microboot-base模块】将当前的项目模块打包处理：clean package;



1. 将“microbbot.jar”文件随意拷贝到一个路径中，在命令行下执行：java -jar xxx.jar（linux环境同理）。

