SpringBoot

简介：

SpringBoot是由Pivotal团队在2013年开始研发、2014年4月发布第一个版本的全新开源的轻量级框架。它基于Spring4.0设计，不仅继承了Spring框架原有的优秀特性，而且还通过简化配置来进一步简化了Spring应用的整个搭建和开发过程。另外SpringBoot通过集成大量的框架使得依赖包的版本冲突，以及引用的不稳定性等问题得到了很好的解决。

优点：

1. 快速构建项目
2. 对主流开发框架的无配置集成
3. 项目可以独立运行，内嵌了servlet容器
4. 提高了开发和部署的效率
5. Cloud
6. **快速入门**

如何在pom中引用springboot. Springboot的项目必须要将parent设置为springboot的parent，该parent包含了大量默认的配置，大大简化了我们的开发。

<parent>

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

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

<version>2.1.6.RELEASE</version>

</parent>

1. **添加Springboot的插件**

<plugin>

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

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

</plugin>

第一个springboot程序

先去springboot主页创建一个工程。

Pom.xml

**<?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>**

**<parent>**

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

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

**<version>**2.1.6.RELEASE**</version>**

**<relativePath/>** *<!-- lookup parent from repository -->*

**</parent>**

**<groupId>**com.example.**</groupId>**

**<artifactId>**demo**</artifactId>**

**<version>**0.0.1-SNAPSHOT**</version>**

**<name>**demo**</name>**

**<description>**Demo project for Spring Boot**</description>**

**<properties>**

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

**</properties>**

**<dependencies>**

**<dependency>**

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

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

**</dependency>**

**<dependency>**

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

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

**<scope>**test**</scope>**

**</dependency>**

**<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>**

下面是Springboot主程序。

**package** **com.example.demo**;

**import** **org.springframework.boot.SpringApplication**;

**import** **org.springframework.boot.autoconfigure.SpringBootApplication**;

*/\*\**

*\* 用@SpringBootApplication 来标注一个主程序类，说明这是一个SpringBoot应用*

*\*/*

**@SpringBootApplication**

**public** **class** **DemoApplication** {

**public** **static** void main(String[] args) {

*//Spring应用启动起来*

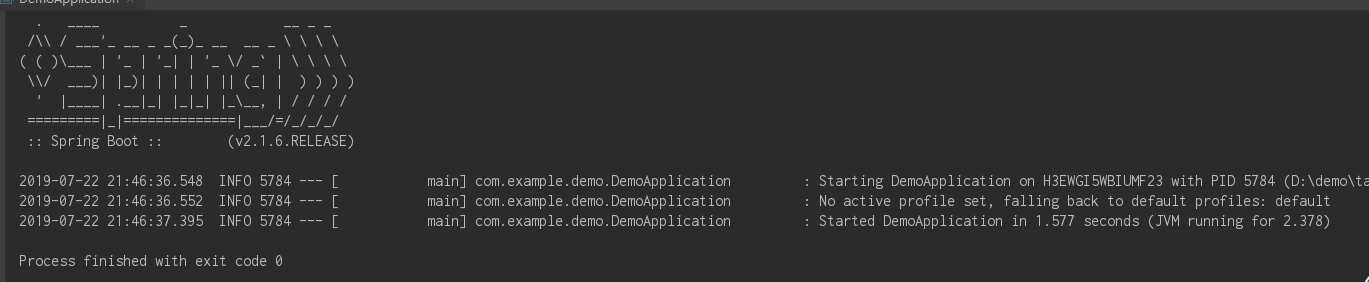
*//调用run方法来创建Springapplication对象，参数就是当前类本身。或者是包含@SpringBootApplication标签的类。*

SpringApplication.run(DemoApplication.class, args);

}

}

上面这个程序没有任何功能。运行结果如下：



SpringBoot的项目一般都会有\*Application的入口类，入口类中会有main方法，这是一个标准的Java应用程序的入口方法。

现在我们增加一些内容

我们在POM.xml中增加web的支持：

1. **导入springboot的web支持**

<dependency>

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

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

</dependency>

这里不需要设置版本号。

在DemoApplication.java中增加以下内容：

**package** **com.example.demo**;

**import** **org.springframework.boot.SpringApplication**;

**import** **org.springframework.boot.autoconfigure.SpringBootApplication**;

**import** **org.springframework.context.annotation.Configuration**;

**import** **org.springframework.stereotype.Controller**;

**import** **org.springframework.web.bind.annotation.RequestMapping**;

**import** **org.springframework.web.bind.annotation.ResponseBody**;

**@Controller**

*/\*\**

*\* 用@SpringBootApplication 来标注一个主程序类，说明这是一个SpringBoot应用*

*\*/*

**@SpringBootApplication**

**@Configuration**

**public** **class** **DemoApplication** {

**@RequestMapping**("hello")

**@ResponseBody**

**public** String hello(){

**return**"hello world！";

}

**public** **static** void main(String[] args) {

*//Spring应用启动起来*

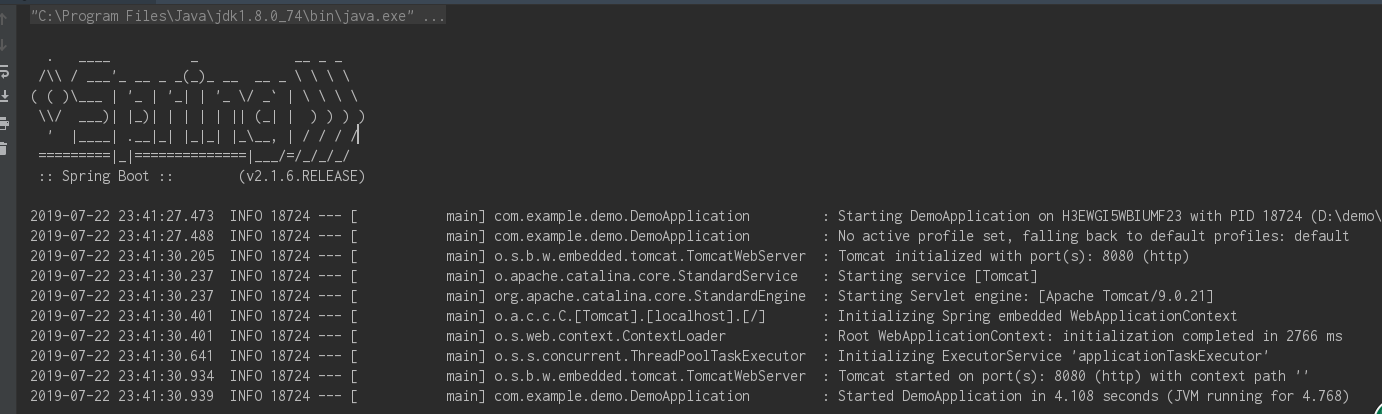
*//调用run方法来创建Springapplication对象，参数就是当前类本身。或者是包含@SpringBootApplication标签的类。*

SpringApplication.run(DemoApplication.class, args);

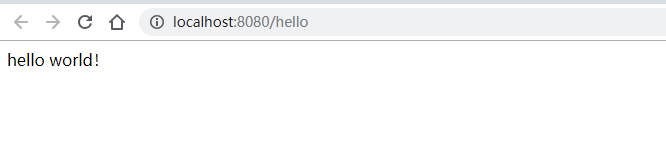
}

}

在次运行代码：



这时访问localhost:8080/hello



代码说明：

1、@SpringBootApplication：SpringBoot项目的核心注解，主要目的是开启自动配置。；

2、@Configuration：这是一个配置Spring的配置类；

3、@Controller：标明这是一个SpringMVC的Controller控制器；

4、main方法：在main方法中启动一个应用，即：这个应用的入口；

一共有4种方法注入bean – 注册组件

* @Controller 接口层
* @Service 业务层
* @Repository 数据层(DAO)
* @Component 不属于上面任何一层的都可以放在这里

一共有3种方法

* @Autowired
* @Inject
* @Resource

@Controller注解。

* Controller是用来处理http请求的。
* 一般来说必须配合@ResponseBody来使用，否则必须使用模板，例如Thymeleaf
* @Controller用于标记在一个类上，使用它标记的类就是一个SpringMVC Controller对象。分发处理器将会扫描使用了该注解的类的方法，并检测该方法是否使用了@RequestMapping注解。@Controller只是定义了一个控制器类，而使用@RequestMapping注解的方法才是真正处理请求的处理器.
* @RequestMapping是配置URL映射

修改上面的代码：

在com.example.demo包下，新建一个package-controller

在controller包下新建文件HelloController.java

**package** **com.example.demo.controller**;

**import** **org.springframework.stereotype.Controller**;

**import** **org.springframework.web.bind.annotation.RequestMapping**;

**import** **org.springframework.web.bind.annotation.ResponseBody**;

**@Controller**

**public** **class** **HelloController** {

*/\**

*//这段代码先注释掉，以后再用*

*@RequestMapping("hello")*

*@ResponseBody*

*public String hello(){*

*return"hello world！";*

*}\*/*

**public** void getName(){

System.out.println("hello~");

}

}

在程序运行的时候将Spring 扫描到@Controller标签后会声明一个**HelloController**的实例。

在src/test/java/com/example/demo 下DemoApplicationTests.java文件中通过@Autowired注入上面的bean。

代码如下：

**package** **com.example.demo**;

**import** **com.example.demo.controller.HelloController**;

**import** **org.junit.Test**;

**import** **org.junit.runner.RunWith**;

**import** **org.springframework.beans.factory.annotation.Autowired**;

**import** **org.springframework.boot.test.context.SpringBootTest**;

**import** **org.springframework.test.context.junit4.SpringRunner**;

**@RunWith**(SpringRunner.class)

**@SpringBootTest**

**public** **class** **DemoApplicationTests** {

**@Autowired**

HelloController helloController;

**@Test**

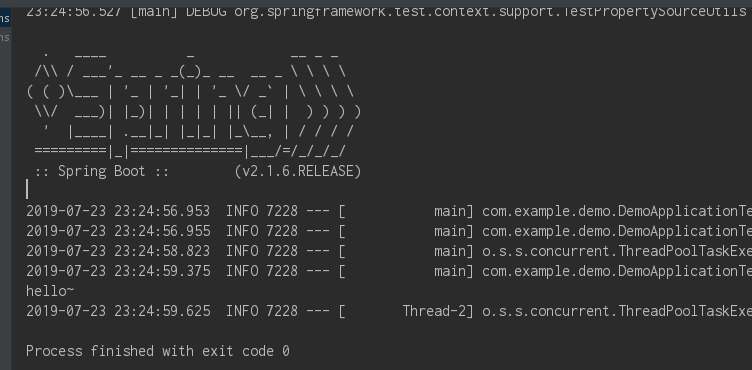
**public** void contextLoads() {

helloController.getName();

}

}

运行程序： 打印出 hello~,说明注入成功！！！



\*虽然@Controller是用来处理URL的请求，但是上面的代码并没有对请求做任何回应，所以访问localhost:8080不会有任何反应。

如果我们放开注释的部分在运行springboot主程序，然后访问localhost:8080/hello,浏览器中将返回hello world!

代码部分：

**package** **com.example.demo.controller**;

**import** **org.springframework.stereotype.Controller**;

**import** **org.springframework.web.bind.annotation.RequestMapping**;

**import** **org.springframework.web.bind.annotation.ResponseBody**;

**@Controller**

**public** **class** **HelloController** {

**@RequestMapping**("hello")

**@ResponseBody**

**public** String hello(){

**return**"hello world！";

}

**public** void getName(){

System.out.println("hello~");

}

}

@Service 业务层

在com.example.demo 下新建包services

新建HelloService.java

**package** **com.example.demo.services**;

**import** **org.springframework.stereotype.Service**;

**@Service**

**public** **class** **HelloService** {

**public** void getService(){

System.out.println("This is service~~!");

}

}

修改DemoApplicationTests.java，增加HelloService的引用

**package** **com.example.demo**;

**import** **com.example.demo.controller.HelloController**;

**import** **com.example.demo.services.HelloService**;

**import** **org.junit.Test**;

**import** **org.junit.runner.RunWith**;

**import** **org.springframework.beans.factory.annotation.Autowired**;

**import** **org.springframework.boot.test.context.SpringBootTest**;

**import** **org.springframework.test.context.junit4.SpringRunner**;

**@RunWith**(SpringRunner.class)

**@SpringBootTest**

**public** **class** **DemoApplicationTests** {

**@Autowired**

HelloController helloController;

**@Autowired**

HelloService helloService;

**@Test**

**public** void contextLoads() {

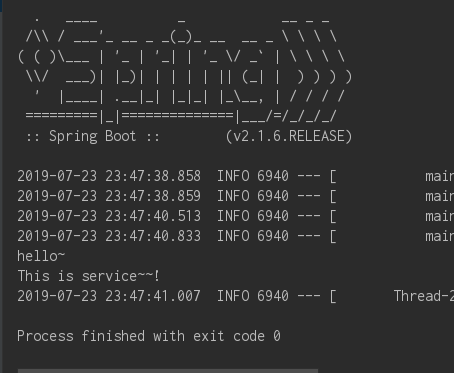
helloController.getName();

helloService.getService();

}

}

运行：



@Component 如果新增的模块不属于任何一层，无法明确归类的话，就可以放在这里。

在com.example.demo 下新建包components

新建HelloComponent.java

**package** **com.example.demo.components**;

**import** **org.springframework.stereotype.Component**;

**@Component**

**public** **class** **HelloComponent** {

**public** void getComponent(){

System.out.println("Hello Component!");

}

}

修改DemoApplicationTests.java，增加HelloComponent的引用

**package** **com.example.demo**;

**import** **com.example.demo.components.HelloComponent**;

**import** **com.example.demo.controller.HelloController**;

**import** **com.example.demo.services.HelloService**;

**import** **org.junit.Test**;

**import** **org.junit.runner.RunWith**;

**import** **org.springframework.beans.factory.annotation.Autowired**;

**import** **org.springframework.boot.test.context.SpringBootTest**;

**import** **org.springframework.test.context.junit4.SpringRunner**;

**@RunWith**(SpringRunner.class)

**@SpringBootTest**

**public** **class** **DemoApplicationTests** {

**@Autowired**

HelloController helloController;

**@Autowired**

HelloService helloService;

**@Autowired**

HelloComponent helloComponent;

**@Test**

**public** void contextLoads() {

helloController.getName();

helloService.getService();

helloComponent.getComponent();

}

}

@Configuration配置又称java配置

通过@Configuration来声明当前类是一个配置类，然后通过@Bean注解在方法上，来声明当前方法的返回值是一个Bean。本质上也是@Component

什么时候用@Configuration，什么时候用注解配置呢？

原则： 全局配置使用java配置（如数据库相关，MVC相关），业务BEAN的配置使用注解配置。

@Configuration中所有带@Bean注解的方法都会被动态代理，调用该方法返回的都是同一个实例。

@Bean一般是用在导入第三方的包

新建一个package – user

新建文件User.java

**package** **com.example.demo.user**;

**public** **class** **User** {

**@Override**

**public** String toString() {

**return** "User{" +

"name='" + name + '\'' +

", age=" + age +

", address='" + address + '\'' +

'}';

}

**public** String name;

**public** int age;

**public** String address;

}

新建一个package – congiguration

新建文件Configurations.java

**package** **com.example.demo.configuration**;

**import** **com.example.demo.user.User**;

**import** **org.springframework.context.annotation.Bean**;

**import** **org.springframework.context.annotation.Configuration**;

**@Configuration**

**public** **class** **Configurations** {

**@Bean**

**public** User user(){

**return** **new** User();

}

}

修改DemoApplicationTests.java

**package** **com.example.demo**;

**import** **com.example.demo.components.HelloComponent**;

**import** **com.example.demo.controller.HelloController**;

**import** **com.example.demo.services.HelloService**;

**import** **com.example.demo.user.User**;

**import** **org.junit.Test**;

**import** **org.junit.runner.RunWith**;

**import** **org.springframework.beans.factory.annotation.Autowired**;

**import** **org.springframework.boot.test.context.SpringBootTest**;

**import** **org.springframework.context.annotation.AnnotationConfigApplicationContext**;

**import** **org.springframework.test.context.junit4.SpringRunner**;

**@RunWith**(SpringRunner.class)

**@SpringBootTest**

**public** **class** **DemoApplicationTests** {

**@Autowired**

HelloController helloController;

**@Autowired**

HelloService helloService;

**@Autowired**

HelloComponent helloComponent;

**@Autowired**

User user;

**@Test**

**public** void contextLoads() {

helloController.getName();

helloService.getService();

helloComponent.getComponent();

user.address = "hello ~~~~~~~~~~~~~~~~~~~~~~";

System.out.println(user.toString());

AnnotationConfigApplicationContext applicationContext = **new** AnnotationConfigApplicationContext(DemoApplication.class);

String[] defin = applicationContext.getBeanDefinitionNames();

**for**(String name:defin)

System.out.println(name);

}

}

运行上述代码，可以发现user已经被注入可以使用了。

**@ComponentScan注解是什么？**

其实很简单，@ComponentScan主要就是定义**扫描的路径**从中找出标识了**需要装配**的类自动装配到spring的bean容器中。

也就是说上面的组件可以被注入是因为他们和主程序DemoApplication在同一个包demo下面，所以可以被自动识别。如果需要注册的组件不在同一个包，就需要@ComponentScan来注册。

新建一个包com.example.another

新建文件ComponentTest.java

**package** **com.example.another**;

**import** **org.springframework.stereotype.Component**;

**@Component**

**public** **class** **ComponentTest** {

}

这时候运行测试程序会发现在IOC中，并没有这个组件。

修改Configurations.java 增加@ComponentScan

**package** **com.example.demo.configuration**;

**import** **com.example.demo.user.User**;

**import** **org.springframework.context.annotation.Bean**;

**import** **org.springframework.context.annotation.ComponentScan**;

**import** **org.springframework.context.annotation.Configuration**;

**@Configuration**

**@ComponentScan**(value="com.example.another")

**public** **class** **Configurations** {

**@Bean**

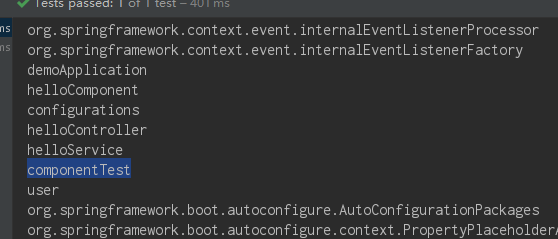
**public** User user(){

**return** **new** User();

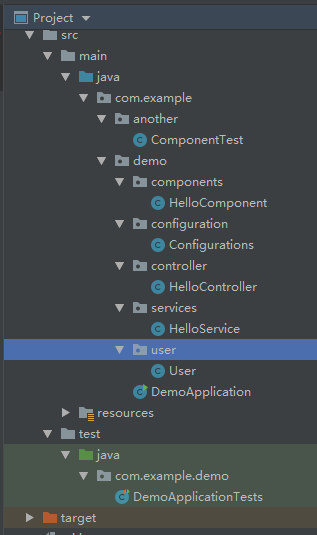
}

}

这是在运行测试代码，发现新的组件已经存在IOC中了。



代码目录结构



@RestController注解

@RestController注解相当于@ResponseBody ＋ @Controller合在一起的作用。返回json数据。

在Controller包下，新建HelloRestController.java

**package** **com.example.demo.controller**;

**import** **org.springframework.web.bind.annotation.RequestMapping**;

**import** **org.springframework.web.bind.annotation.RestController**;

**import** **java.util.HashMap**;

**import** **java.util.Map**;

**@RestController**

**public** **class** **HelloRestController** {

**@RequestMapping**(value="/helloworld")

**public** Map<String, String> hello(){

Map<String, String> helloWrold = **new** HashMap<>();

helloWrold.put("name", "sam");

helloWrold.put("age", "16");

helloWrold.put("sex", "male");

**return** helloWrold;

}

}

启动程序，访问localhost:8080/helloworld