自定义Spring Boot Starter

在Spring Boot中Starter是核心组成部分，Spring Boot已经提供了一系列的自动化配置的Starter插件，例如使用spring-boot-starter-web时只需要在pom.xml中添加依赖即可，下面自定义Starter并通过spring-boot-autoconfigure完成自动化配置

1. 构建项目，pom.xml中添加依赖，信息如下

*<groupId>com.fys</groupId>*

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

*<version>1.0-SNAPSHOT</version>*

*<packaging>jar</packaging>*

*<properties>*

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

*</properties>*

*<dependencies>*

*<dependency>*

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

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

*<version>1.5.4.RELEASE</version>*

*</dependency>*

*</dependencies>*

# 只添加spring-boot-autoconfigure模块

1. 配置映射参数实体，starter从application.yml文件获取自定义配置信息，类定义如下：

*@Getter*

*@Setter*

*@ConfigurationProperties(prefix = "hello")*

*public class HelloProperties {*

*private String msg = "Hello FYS";*

*private boolean show = true;*

*}*

Spring Boot提供注解

*@ConfigurationProperties*

该注解可以完成将application.properties或者application.yml配置文件中有规则的配置参数映射到实体的field内，在配置参数实体类中药提供Setter方法。

上例中使用属性prefix，该属性配置了读取参数的前缀，根据上面的实体属性对应配置文件内的配置则是：

*hello.msg*

*hello.show*

在示例中提供了默认值，配置文件不进行配置时则使用默认值。

1. 编写自定义业务

* 在Starter中提供Service，在类中定义sayHello方法返回配置的msg内容，代码如下：

*@Setter*

*public class HelloService {*

*private String msg;*

*private boolean show = true;*

*public String sayHello() {*

*return show ? "Hello." + msg : "Hidden";*

*}*

*}*

根据属性参数返回格式化后的字符串

* 实现自动化配置，提供实体bean的验证以及初始化，代码如下：

*@Configuration*

*@EnableConfigurationProperties(HelloProperties.class)*

*@ConditionalOnClass(HelloService.class)*

*@ConditionalOnProperty(*

*prefix = "hello",*

*value = "enabled",*

*matchIfMissing = true*

*)*

*public class HelloAutoConfiguration {*

*@Autowired*

*private HelloProperties helloProperties;*

*@Bean*

*@ConditionalOnMissingBean(HelloService.class)*

*public HelloService helloService() {*

*System.out.println(">>>> The HelloService Not Found, Execute Create New Bean.");*

*HelloService helloService = new HelloService();*

*helloService.setMsg(helloProperties.getMsg());*

*helloService.setShow(helloProperties.isShow());*

*return helloService;*

*}}*

核心是注解配置：

* @Configuration
* @EnableConfigurationProperties，开启使用配置参数的注解，value值为配置实体参数映射的ClassType，将配置实体作为配置来源
* @ConditionOnX，条件注解，是自动配置的关键，根据不同条件创建具体的Bean
* @ConditionOnBean，当SpringIoc容器内存在指定的Bean的条件
* @ConditionOnClass，当SpringIoc容器内存在指定Class的条件
* @ConditionalOnExpression：基于SpEL表达式作为判断条件  
  @ConditionalOnJava：基于JVM版本作为判断条件  
  @ConditionalOnJndi：在JNDI存在时查找指定的位置  
  @ConditionalOnMissingBean：当SpringIoc容器内不存在指定Bean的条件  
  @ConditionalOnMissingClass：当SpringIoc容器内不存在指定Class的条件  
  @ConditionalOnNotWebApplication：当前项目不是Web项目的条件  
  @ConditionalOnProperty：指定的属性是否有指定的值  
  @ConditionalOnResource：类路径是否有指定的值  
  @ConditionalOnSingleCandidate：当指定Bean在SpringIoc容器内只有一个，或者虽然有多个但是指定首选的Bean  
  @ConditionalOnWebApplication：当前项目是Web项目的条件

1. 自定义spring.factories

在src/main/resource目录下创建META-INF，并在目录中添加文件spring.factories，具体内如如下：

*org.springframework.boot.autoconfigure.EnableAutoConfiguration=com.fys.starter.HelloAutoConfiguration*

至此spring-boot-starter-fys创建完毕，编译上传到本地maven仓库

*在META-INF中可以定义spring.provides，例如spring-boot-starter-tomcat中定义如下：*

*provides: tomcat-embed-core*

配置了该模块的依赖关系，在构建中获取tomcat-embed-core依赖库，maven在构建期间解析该依赖关系。

1. 测试Spring Boot项目，创建测试项目中pom.xml配置文件内容如下：

*<dependencies>*

*<dependency>*

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

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

*</dependency>*

*<dependency>*

*<groupId>com.fys</groupId>*

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

*<version>1.0-SNAPSHOT</version>*

*</dependency>*

*<dependency>*

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

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

*</dependency>*

*<dependency>*

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

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

*<scope>test</scope>*

*</dependency>*

*</dependencies>*

编写测试控制器和启动类

|  |  |
| --- | --- |
| *@RestController*  *public class HelloController {*  *@Autowired*  *HelloService helloService;*  *@RequestMapping(value = "/hello")*  *public String sayHello() {*  *return helloService.sayHello();*  *}*  *}* | *@SpringBootApplication*  *public class SampleApplication {*  *public static void main(String[] args) {*  *SpringApplication.run(*  *SampleApplication.class, args);*  *}*  *}* |

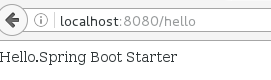
在spring.properties配置参数，格式如下：

*debug=true*

*hello.msg=Spring Boot Starter*

*hello.show=true*

启动项目后，访问REST，界面输出内容如下：



配置生效，至此完成了自定义Spring Boot Starter。

*附录：*

参考链接：

https://segmentfault.com/a/1190000011433487

https://github.com/lianggzone/springboot-action