

# Spring Boot自定义starters

## Spring Boot自定义starters

### 一、简介

### 二、如何自定义starter

### 三、自定义starter实例

## 一、简介

SpringBoot 最强大的功能就是把我们将常用的场景抽取成了一个个starter（场景启动器），我们通过引入springboot 为我们提供的这些场景启动器，我们再进行少量的配置就能使用相应的功能。即使是这样，springboot也不能囊括我们所有的使用场景，往往我们需要自定义starter，来简化我们对springboot的使用。

## 二、如何自定义starter

### 1.实例

如何编写自动配置？

我们参照@WebMvcAutoConfiguration为例，我们看看们需要准备哪些东西，下面是WebMvcAutoConfiguration的部分代码：

```
1  @Configuration
2  @ConditionalOnWebApplication
3  @ConditionalOnClass({Servlet.class, DispatcherServlet.class, WebMvcConfigurerAdapter.class})
4  @ConditionalOnMissingBean({WebMvcConfigurationSupport.class})
5  @AutoConfigureOrder(-2147483638)
6  @AutoConfigureAfter({DispatcherServletAutoConfiguration.class, ValidationAutoConfiguration.class})
7  public class WebMvcAutoConfiguration {
8
9      @Import({WebMvcAutoConfiguration.EnableWebMvcConfiguration.class})
10     @EnableConfigurationProperties({WebMvcProperties.class, ResourceProperties.class})
11     public static class WebMvcAutoConfigurationAdapter extends WebMvcConfigurerAdapter {
12
13         @Bean
14         @ConditionalOnBean({View.class})
15         @ConditionalOnMissingBean
16         public BeanNameViewResolver beanNameViewResolver() {
17             BeanNameViewResolver resolver = new BeanNameViewResolver();
18             resolver.setOrder(2147483637);
19
20             return resolver;
21         }
22     }
23 }
```

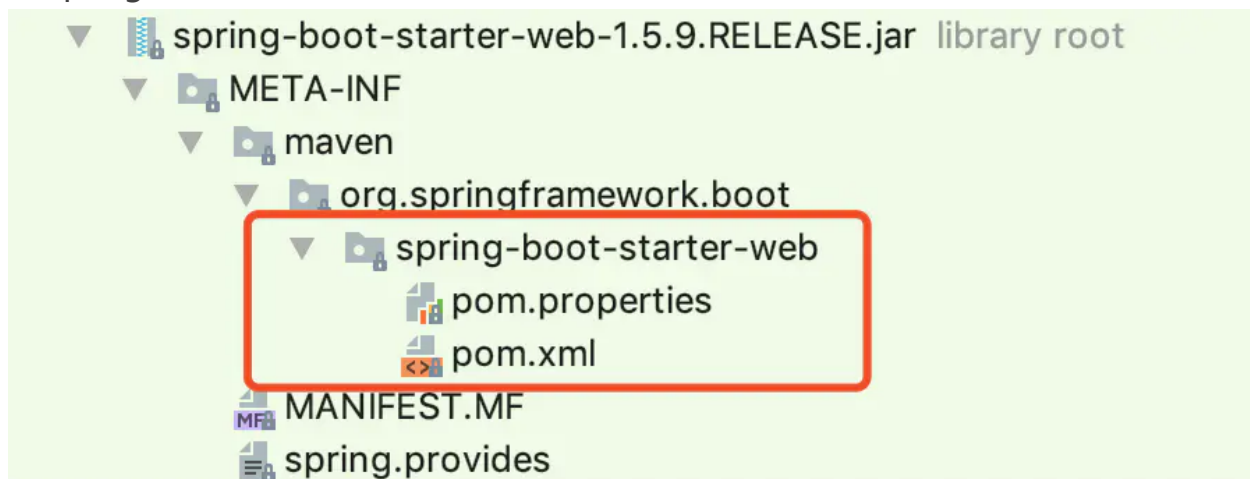
```
20     }  
21 }  
22 }
```

我们可以抽取到我们自定义starter时同样需要的一些配置。

```
1 @Configuration //指定这个类是一个配置类  
2 @ConditionalOnXXX //指定条件成立的情况下自动配置类生效  
3 @AutoConfigureOrder //指定自动配置类的顺序  
4 @Bean //向容器中添加组件  
5 @ConfigurationProperties //结合相关xxxProperties来绑定相关的配置  
6 @EnableConfigurationProperties //让xxxProperties生效加入到容器中  
7  
8 自动配置类要能加载需要将自动配置类，配置在META-INF/spring.factories中  
9 org.springframework.boot.autoconfigure.EnableAutoConfiguration=  
10 org.springframework.boot.autoconfigure.admin.SpringApplicationAdminJmxAutoConfiguration,  
11 org.springframework.boot.autoconfigure.aop.AopAutoConfiguration,\
```

## 模式

我们参照 spring-boot-starter 我们发现其中没有代码：



我们在看它的pom中的依赖中有个 springboot-starter

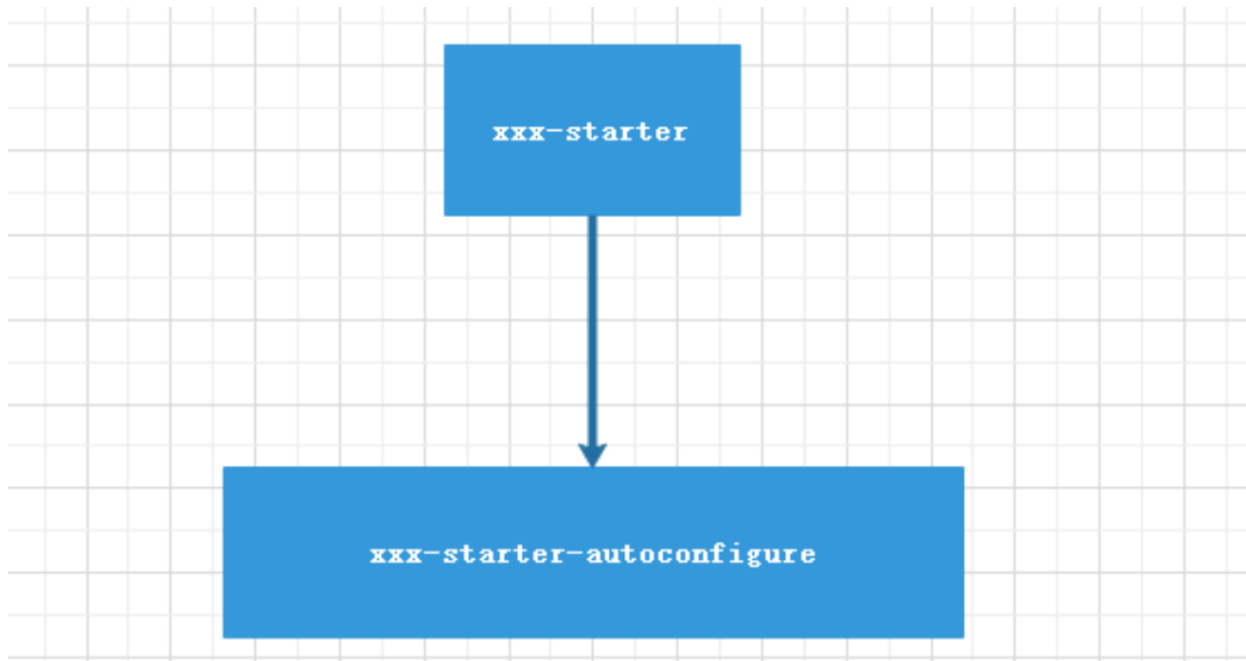
```
1 <dependency>  
2     <groupId>org.springframework.boot</groupId>  
3     <artifactId>spring-boot-starter</artifactId>  
4 </dependency>
```

我们再看看 spring-boot-starter 有个 spring-boot-autoconfigure

```
1 <dependency>
```

```
2 <groupId>org.springframework.boot</groupId>
3 <artifactId>spring-boot-autoconfigure</artifactId>
4 </dependency>
```

关于web的一些自动配置都写在了这里，所以我们有总结：



- 启动器（starter）是一个空的jar文件，仅提供辅助性依赖管理，这些依赖可能用于自动装配或其他类库。
- 需要专门写一个类似spring-boot-autoconfigure的配置模块
- 用的时候只需要引入启动器starter，就可以使用自动配置了

## 命名规范

### 官方命名空间

- 前缀：spring-boot-starter-
- 模式：spring-boot-starter-模块名
- 举例：spring-boot-starter-web、spring-boot-starter-jdbc

### 自定义命名空间

- 后缀：-spring-boot-starter
- 模式：模块-spring-boot-starter
- 举例：mybatis-spring-boot-starter

## 三、自定义starter实例

我们需要先创建一个父maven项目:springboot\_custome\_starter

两个Module: tulingxueyuan-spring-boot-starter 和 tulingxueyuan-spring-boot-starter-autoconfigurer

springboot\_custome\_starter

pom.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema"
3     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4     <modelVersion>4.0.0</modelVersion>
5     <modules>
6         <module>tulingxueyuan-spring-boot-starter</module>
7         <module>tulingxueyuan-spring-boot-autoconfigure</module>
8     </modules>
9     <parent>
10         <groupId>org.springframework.boot</groupId>
11         <artifactId>spring-boot-starter-parent</artifactId>
12         <version>2.3.6.RELEASE</version>
13         <relativePath/> <!-- lookup parent from repository -->
14     </parent>
15     <packaging>pom</packaging>
16     <groupId>com.tulingxueyuan.springboot</groupId>
17     <artifactId>springboot_custome_starter</artifactId>
18     <version>0.0.1-SNAPSHOT</version>
19     <name>springboot_custome_starter</name>
20     <description>SpringBoot自定义starter</description>
21
22     <properties>
23         <java.version>1.8</java.version>
24     </properties>
25
26     <dependencies>
27         <dependency>
28             <groupId>org.springframework.boot</groupId>
29             <artifactId>spring-boot-starter</artifactId>
30         </dependency>
31     </dependencies>
32
33 </project>
```

## 1. tulingxueyuan-spring-boot-starter

### 1.pom.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xs
5     <parent>
6         <artifactId>springboot_custome_starter</artifactId>
7         <groupId>com.tulingxueyuan.springboot</groupId>
8         <version>0.0.1-SNAPSHOT</version>
9     </parent>
10    <modelVersion>4.0.0</modelVersion>
11    <description>
12        启动器（starter）是一个空的jar文件，
13        仅提供辅助性依赖管理，
14        这些依赖需要自动装配或其他类库。
15    </description>
16
17    <artifactId>tulingxueyuan-spring-boot-starter</artifactId>
18
19    <dependencies>
20        <!--引入autoconfigure-->
21        <dependency>
22            <groupId>com.tulingxueyuan.springboot</groupId>
23            <artifactId>tulingxueyuan-spring-boot-autoconfigure</artifactId>
24            <version>0.0.1-SNAPSHOT</version>
25        </dependency>
26
27        <!--如果当前starter 还需要其他的类库就在这里引用-->
28    </dependencies>
29
30 </project>

```

如果使用spring Initializr创建的需要删除 启动类、resources下的文件，test文件。

## 2. tulingxueyuan-spring-boot-starter-autoconfigurer

### 1. pom.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xs
5     <parent>
6         <artifactId>springboot_custome_starter</artifactId>
7         <groupId>com.tulingxueyuan.springboot</groupId>
8         <version>0.0.1-SNAPSHOT</version>
9     </parent>
10    <modelVersion>4.0.0</modelVersion>
11
12    <artifactId>tulingxueyuan-spring-boot-autoconfigure</artifactId>
13    <dependencies>
14        <dependency>
15            <groupId>org.springframework.boot</groupId>
16            <artifactId>spring-boot-starter-web</artifactId>
17        </dependency>
18        <!--导入配置文件处理器，配置文件进行绑定就会有提示-->
19        <dependency>
20            <groupId>org.springframework.boot</groupId>
21            <artifactId>spring-boot-configuration-processor</artifactId>
22            <optional>true</optional>
23        </dependency>
24    </dependencies>
25
26
27 </project>
28

```

## 2. HelloProperties

```

1 package com.starter.tulingxueyuan;
2
3 import org.springframework.boot.context.properties.ConfigurationProperties;
4
5 /**
6  * @Author 徐庶    QQ:1092002729
7  * @Slogan 致敬大师，致敬未来的你

```

```

8  */
9  @ConfigurationProperties("tuling.hello")
10 public class HelloProperties {
11     private String name;
12
13     public String getName() {
14         return name;
15     }
16
17     public void setName(String name) {
18         this.name = name;
19     }
20 }
21

```

### 3. IndexController

```

1  package com.starter.tulingxueyuan;
2
3  import org.springframework.beans.factory.annotation.Autowired;
4  import org.springframework.web.bind.annotation.RequestMapping;
5  import org.springframework.web.bind.annotation.RestController;
6
7  /**
8   * @Author 徐庶    QQ:1092002729
9   * @Slogan 致敬大师，致敬未来的你
10  */
11 @RestController
12 public class IndexController {
13
14     HelloProperties helloProperties;
15
16     public IndexController(HelloProperties helloProperties) {
17         this.helloProperties=helloProperties;
18     }
19
20     @RequestMapping("/")
21     public String index(){
22         return helloProperties.getName()+"欢迎您";
23     }
24 }

```

```
23     }
24
25 }
26
```

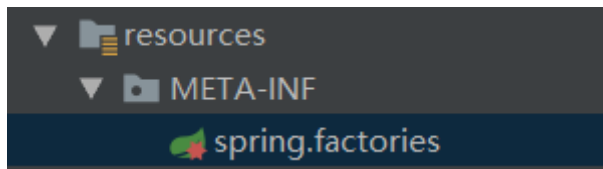
## 4. HelloAutoConfitguration

```
1  package com.starter.tulingxueyuan;
2
3  import org.springframework.beans.factory.annotation.Autowired;
4  import org.springframework.boot.autoconfigure.condition.ConditionalOnProperty;
5  import org.springframework.boot.context.properties.EnableConfigurationProperties;
6  import org.springframework.context.annotation.Bean;
7  import org.springframework.context.annotation.Configuration;
8
9  /**
10   * @Author 徐庶    QQ:1092002729
11   * @Slogan 致敬大师，致敬未来的你
12   *
13   * 给web应用自动添加一个首页
14   */
15  @Configuration
16  @ConditionalOnProperty(value = "tuling.hello.name")
17  @EnableConfigurationProperties(HelloProperties.class)
18  public class HelloAutoConfitguration {
19
20      @Autowired
21      HelloProperties helloProperties;
22
23      @Bean
24      public IndexController indexController(){
25          return new IndexController(helloProperties);
26      }
27  }
28
```

## 5. spring.factories

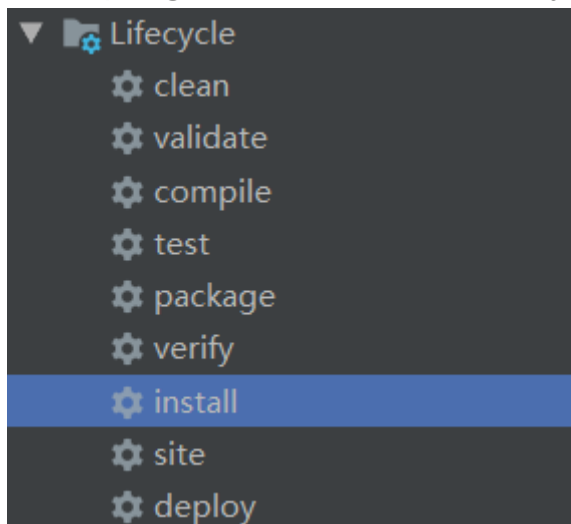
在 resources 下创建文件夹 META-INF 并在 META-INF 下创建文件 spring.factories ， 内容如下：





```
1
2 org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
3   com.starter.tulingxueyuan.HelloAutoConfitguration
```

到这儿，我们的配置自定义的starter就写完了，我们hello-spring-boot-starter-autoconfigurer、hello-spring-boot-starter 安装成本地jar包。



### 三、测试自定义starter

我们创建个Module: 12\_springboot\_starter，来测试系我们写的stater。

#### 1. pom.xml

```
1 <dependency>
2   <groupId>com.tulingxueyuan.springboot</groupId>
3   <artifactId>tulingxueyuan-spring-boot-starter</artifactId>
4   <version>0.0.1-SNAPSHOT</version>
5 </dependency>
6
7
```

#### 2.浏览

<http://localhost:8080/>

# Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Tue Dec 22 20:02:17 CST 2020

There was an unexpected error (type=Not Found, status=404).

由于在自动配置上设置了

```
1 @ConditionalOnProperty(value = "tuling.hello.name")
```

但我们还没有配置。so.....

## 3. application.properties

```
1 tuling.hello.name="图灵学院"
```

再次访问: <http://localhost:8080/>

---

# "图灵学院"欢迎您