

保姆级教程，手把手教你实现SpringBoot自定义starter

 developer.aliyun.com/article/893073

开发者社区 开发与运维 文章 正文

版权

简介：保姆级教程，手把手教你实现SpringBoot自定义starter

引言

上篇文章《天天用SpringBoot，它的自动装配原理却说不出来》我们有说springBoot的自动装配（建议最好先看下篇文章，因为前后有关系），这篇文章的话我们就继续来看下SpringBoot的starter吧。什么是Spring Boot Starter呢？我们直接来看看官网是怎么介绍的吧

Starters are a set of convenient dependency descriptors that you can include in your application. You get a one-stop shop for all the Spring and related technologies that you need without having to hunt through sample code and copy-paste loads of dependency descriptors. For example, if you want to get started using Spring and JPA for database access, include the spring-boot-starter-data-jpa dependency in your project.

纳尼，一大堆的英文，这还有兴趣接着往下看吗？是不是看到这直接退出了。都到门口了，不进来喝杯茶再走嘛？看都看到这了还是接着继续往下看吧。我们先不解释这一段话是什么意思，我们可以看看starter的出现给我们解决了什么问题。

我们还是以上述官网的例子来进行说明比如说我们需要在Spring中适应JPA来操作数据库。

在没有springBoot-starter之前，我们需要引入jpa的步骤

- 通过maven引入jdbc的依赖、以及jpa相关的各种依赖
- 编写jpa相关的配置文件
- 网上各种查询找资料进行调试，调试的过程对于新手可能会有点奔溃会遇到各种奇奇怪怪的问题，jar包冲突啊，这个jar包下载不下来，缺少某个jar包。

终于在经历千辛万苦，哼次哼次的解决各种问题之后终于把项目跑起来了，然后有的人就会总结下，把这次整合jpa遇到的问题，以及整合的步骤都一一的详细记录下来。方便下次在需要整合jpa的时候直接copy就好了。当你另外一个项目也需要引入jpa的时候，这时候打开整理笔记三下五除二全部copy到项目里，然后就等待项目的成功启动。如果项目里面比如spring的版本和你引入jpa的版本不匹配的话，你又会遇到奇奇怪怪的问题哦。

我们以前在没有starter之前是不是都是这么玩的。这样的缺点是不是也非常显著，比如过程复杂、需要不停的粘贴复制（不过这是程序员经常干的事情了，也不在乎多一两次了）、整合其它组件到自己的项目变的困难，效率低下。

SpringBoot Starter的出现

我们可以看下SpringBoot 现在都为我们提供有哪些starter，我这边这截图了部分starter，

..		
spring-boot-starter-activemq	Require dependency on s-b-dependencies to use its constraints	last month
spring-boot-starter-actuator	Require dependency on s-b-dependencies to use its constraints	last month
spring-boot-starter-amqp	Require dependency on s-b-dependencies to use its constraints	last month
spring-boot-starter-aop	Require dependency on s-b-dependencies to use its constraints	last month
spring-boot-starter-artemis	Require dependency on s-b-dependencies to use its constraints	last month
spring-boot-starter-batch	Require dependency on s-b-dependencies to use its constraints	last month
spring-boot-starter-cache	Require dependency on s-b-dependencies to use its constraints	last month
spring-boot-starter-data-cassandra-reactive	Require dependency on s-b-dependencies to use its constraints	last month
spring-boot-starter-data-cassandra	Require dependency on s-b-dependencies to use its constraints	last month
spring-boot-starter-data-couchbase-reactive	Require dependency on s-b-dependencies to use its constraints	last month
spring-boot-starter-data-couchbase	Require dependency on s-b-dependencies to use its constraints	last month
spring-boot-starter-data-elasticsearch	Require dependency on s-b-dependencies to use its constraints	last month
spring-boot-starter-data-jdbc	Require dependency on s-b-dependencies to use its constraints	last month
spring-boot-starter-data-jpa	Require dependency on s-b-dependencies to use its constraints	last month
spring-boot-starter-data-ldap	Require dependency on s-b-dependencies to use its constraints	last month
spring-boot-starter-data-mongodb-reactive	Require dependency on s-b-dependencies to use its constraints	last month
spring-boot-starter-data-mongodb	Require dependency on s-b-dependencies to use its constraints	last month
spring-boot-starter-data-neo4j	Require dependency on s-b-dependencies to use its constraints	last month
spring-boot-starter-data-r2dbc	Require dependency on s-b-dependencies to use its constraints	last month
spring-boot-starter-data-redis-reactive	Require dependency on s-b-dependencies to use its constraints	last month

starter的实现：虽然不同的starter实现起来各有差异，但是他们基本上都会使用到两个相同的内容：ConfigurationProperties和AutoConfiguration。因为Spring Boot坚信“约定大于配置”这一理念，所以我们使用ConfigurationProperties来保存我们的配置，并且这些配置都可以有一个默认值，即在我们没有主动覆写原始配置的情况下，默认值就会生效，这在很多情况下是非常有用的。除此之外，starter的ConfigurationProperties还使得所有的配置属性被聚集到一个文件中（一般在resources目录下的application.properties），这样我们就告别了Spring项目中XML地狱。

starter的出现帮我们把各种复杂的配置都封装起来了，让我们真正的可以达到了开箱即用。不仅降低了我们使用它的门槛，并且还大大提高了我们的开发效率，正如前面所说

《SpringBoot自动装配》让我们有更多的时间去陪女朋友。

实现自己的SpringBoot Starter

命名规范

如果你快有孩子了，出生前你比较急的一定是起个名字。孩子的姓名标识着你和你爱人的血统，一定不会起隔壁老王的姓氏，肯定会招来异样的眼光。在maven中，groupId代表着姓氏，artifactId代表着名字。Spring Boot也是有一个命名的建议的。所以名字是不能够随便取得，可以按照官方的建议来取。

What's in a name

All official starters follow a similar naming pattern; `spring-boot-starter-where` is a particular type of application. This naming structure is intended to help when you need to find a starter. The Maven integration in many IDEs lets you search dependencies by name. For example, with the appropriate Eclipse or STS plugin installed, you can press ctrl-space in the POM editor and type “spring-boot-starter” for a complete list.

As explained in the “Creating Your Own Starter” section, third party starters should not start with spring-boot, as it is reserved for official Spring Boot artifacts. Rather, a third-party starter typically starts with the name of the project. For example, a third-party starter project called `thirdpartyproject` would typically be named `thirdpartyproject-spring-boot-starter`.

大概意思是

官方的 `starter` 的命名格式为 `spring-boot-starter-{xxxx}` 比如 `spring-boot-starter-activemq`

第三方我们自己的命名格式为 `{xxxx}-spring-boot-starter`。比如 `mybatis-spring-boot-starter`。

如果我们忽略这种约定，是不是会显得我们写的东西不够“专业”。

自定义一个Starter

下面我们就来实现一个自定义的发送短信的starter，命名为 `sms-spring-boot-starter`。

1. 引入pom

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-configuration-processor</artifactId>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.16.18</version>
    <scope>provided</scope>
</dependency>
```

1. 编写配置文件

发短信我们需要配置一些账号信息，不同的短信供应商，账户信息是不一样的，所以我们需要定义一个 `XXXXProperties` 来自动装配这些账户信息。下面我们就以腾讯云和阿里云两家供应商为例；

```

@ConfigurationProperties(prefix = "sms")
@Data
public class SmsProperties {

    private SmsMessage aliyun = new SmsMessage();

    private SmsMessage tencent = new SmsMessage();

    @Data
    public static class SmsMessage{

        /**
         * 用户名
         */
        private String userName;

        /**
         * 密码
         */
        private String passWord;

        /**
         * 秘钥
         */
        private String sign;

        /**
         *
         */
        private String url;

        @Override
        public String toString() {
            return "SmsMessage{" +
                "userName='" + userName + '\'' +
                ", passWord='" + passWord + '\'' +
                ", sign='" + sign + '\'' +
                ", url='" + url + '\'' +
                '}';
        }
    }
}

```

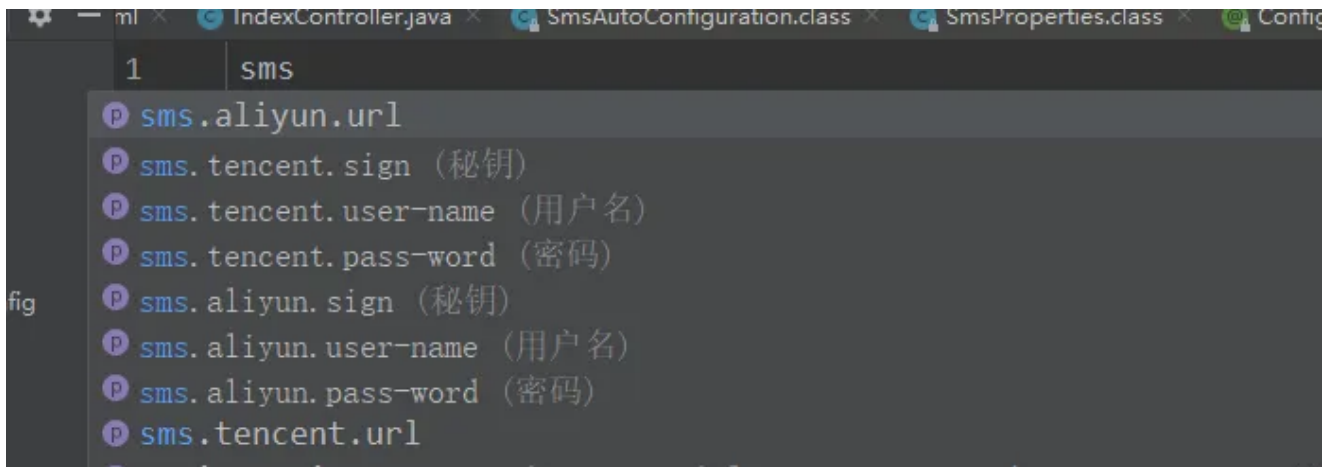
如果需要在其他项目中使用发送短信功能的话，我们只需要在配置文件(application.yml) 中配置SmsProperties 的属性信息就可以了。比如：

```

sms:
  aliyun:
    pass-word: 12345
    user-name: java金融
    sign: 阿里云
    url: http://aliyun.com/send
  tencent:
    pass-word: 6666
    user-name: java金融
    sign: 腾讯云
    url: http://tencent.com/send

```

还记的`@ConfigurationProperties`注解里面是不是有个`prefix` 属性，我们配置的这个属性是`sms`，配置这个的主要一个作用的话是主要用来区别各个组件的参数。这里有个小知识点需要注意下当我们在配置文件输入`sms`我们的`idea`会提示这个`sms`有哪些属性可以配置，以及每个属性的注释都有标记，建议的话注释还是写英文，这样会显得你比较专业。



这个提示的话，是需要引入下面这个jar的。

```

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-configuration-processor</artifactId>
  <optional>true</optional>
</dependency>

```

引入这个jar之后，我们编译之后就会在META-INF文件夹下面生成一个`spring-configuration-metadata.json`的文件。



我们可以看到这个文件其实 是根据SmsProperties类的成员属性来生成的。

1. 然后在编写短信自动配置类：

```

@EnableConfigurationProperties(value = SmsProperties.class)
@Configuration
public class SmsAutoConfiguration {
    /**
     * 阿里云发送短信的实现类
     * @param smsProperties
     * @return
     */
    @Bean
    public AliyunSmsSenderImpl aliYunSmsSender(SmsProperties smsProperties){
        return new AliyunSmsSenderImpl(smsProperties.getAliyun());
    }
    /**
     * 腾讯云发送短信的实现类
     * @param smsProperties
     * @return
     */
    @Bean
    public TencentSmsSenderImpl tencentSmsSender(SmsProperties smsProperties){
        return new TencentSmsSenderImpl(smsProperties.getTencent());
    }
}

```

编写我们的发送短信实现类：

```

public class AliyunSmsSenderImpl implements SmsSender {

    private SmsMessage smsMessage;

    public AliyunSmsSenderImpl(SmsMessage smsProperties) {
        this.smsMessage = smsProperties;
    }

    @Override
    public boolean send(String message) {
        System.out.println(smsMessage.toString()+"开始发送短信==》短信内容："+message);
        return true;
    }
}

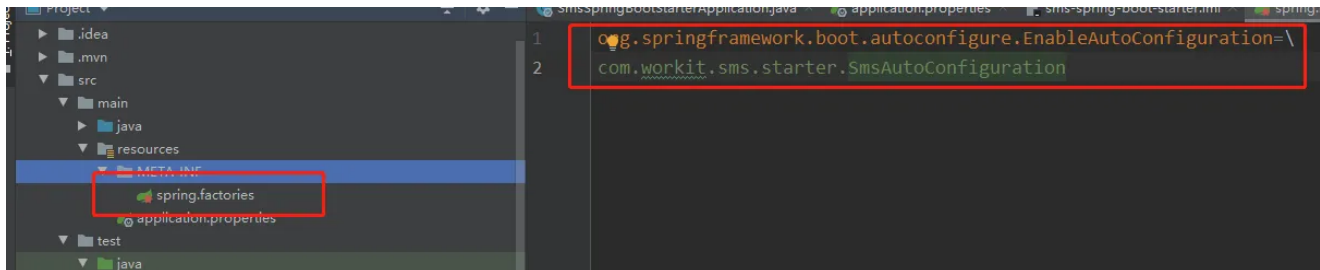
```

1. 让starter生效

starter集成应用有两种方式：

被动生效

我们首先来看下我们熟悉的方式，通过SpringBoot的SPI的机制来去加载我们的starter。我们需要在META-INF下新建一个spring.factories文件key为org.springframework.boot.autoconfigure.EnableAutoConfiguration，value是我们的SmsAutoConfiguration 全限定名（记得去除前后的空格，否则会不生效）。



主动生效

在starter组件集成到我们的Spring Boot应用时需要主动声明启用该starter才生效，通过自定义一个@Enable注解然后在把自动配置类通过Import注解引入进来。

```

@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Import({SmsAutoConfiguration.class})
public @interface EnableSms {
}

```

使用的时候需要在启动类上面开启这个注解。

```

@SpringBootApplication(exclude = {
    RestTemplateAutoConfiguration.class
})
@EnableSms
public class AutoconfigApplication {

```

1. 打包，部署到仓库

如果是本地的话，直接通过mvn install命令就可以了。

如果需要部署到公司的仓库话，这个就不说了。

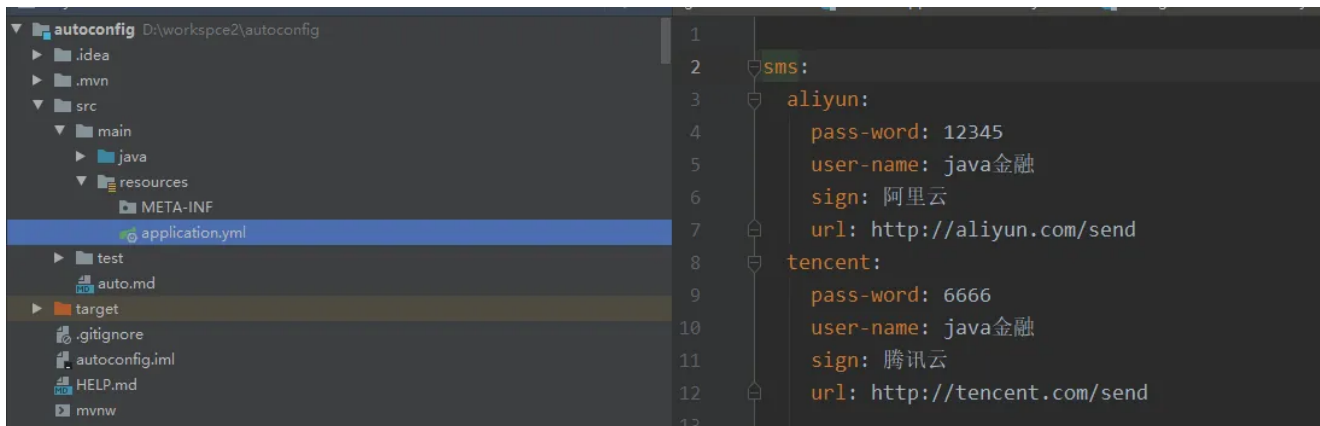
1. 新建一个新的SpringBoot项目引入我们刚写的starter

```

<dependency>
    <groupId>com.workit.sms</groupId>
    <artifactId>sms-spring-boot-starter</artifactId>
    <version>0.0.1-SNAPSHOT</version>
</dependency>

```

在项目配置文件配上短信账号信息



The screenshot shows an IDE with a project named 'autoconfig'. The project structure on the left includes 'src/main/resources/application.yml'. The right pane shows the content of 'application.yml' with the following configuration:

```

1 sms:
2
3   aliyun:
4     pass-word: 12345
5     user-name: java金融
6     sign: 阿里云
7     url: http://aliyun.com/send
8
9   tencent:
10    pass-word: 6666
11    user-name: java金融
12    sign: 腾讯云
13    url: http://tencent.com/send

```

测试代码


```

@SpringBootApplication
@EnableSms
public class AutoconfigApplication {
    public static void main(String[] args) {
        ConfigurableApplicationContext applicationContext =
SpringApplication.run(AutoconfigApplication.class, args);
        AliyunSmsSenderImpl aliyunSmsSender =
applicationContext.getBean(AliyunSmsSenderImpl.class);
        aliyunSmsSender.send("用阿里云发送短信");
        TencentSmsSenderImpl tencentSmsSender =
applicationContext.getBean(TencentSmsSenderImpl.class);
        tencentSmsSender.send("用腾讯云发送短信");
    }
}

```

运行结果：

```

SmsMessage{userName='java金融', password='12345', sign='阿里云',
url='http://aliyun.com/send'}开始发送短信==》短信内容：用阿里云发送短信
SmsMessage{userName='java金融', password='6666', sign='腾讯云',
url='http://tencent.com/send'}开始发送短信==》短信内容：用腾讯云发送短信

```

至此的话我们自定义的一个starter就已经完成了，这个starter只是一个演示的demo，代码有点粗糙。重点看下这个实现原理就好。

<https://www.cnblogs.com/tjudzj/p/8758391.html>

相关文章