链滴

自定义 SpringBoot starter

作者：kyrie92
原文链接：https://ld246.com/article/1675136631566
来源网站：链滴

自定义SpringBoot starter

1、了解SpringBoot starter

SpringBoot中的starter是一种非常重要的机制(自动化配置)，能够抛弃以前繁杂的配置，将其统一集进starter，应用者只需要在maven中引入starter依赖，SpringBoot就能自动扫描到要加载的信息并动相应的默认配置。

starter让我们摆脱了各种依赖库的处理，需要配置各种信息的困扰。SpringBoot会自动通过classpat路径下的类发现需要的Bean，并注册进IOC容器。SpringBoot提供了针对日常企业应用研发各种场的spring-boot-starter依赖模块。所有这些依赖模块都遵循着约定成俗的默认配置，并允许我们调整些配置，即遵循“约定大于配置”的理念。

2、创建Starter的优点

- 包含了许多我们需要的依赖项，以使项目快速启动和运行，并且具有一致的、被支持的一组管传递依赖项
- 不再需要担心依赖关系，它们将由Spring Boot Starters自动管理
- 封装特定的应用程序Starter，只需要通过简单的依赖引用即可，与实际业务代码解耦合

3、常用的Starter

Spring中已经提供了很多Starter给我们使用，常用的包括以下：

- spring-boot-starter-data-elasticsearch：快速集成Elasticsearch 的Starter
- spring-boot-starter-data-jpa：快速集成数据库的Starter
- spring-boot-starter-data-redis：快速集成Redis的Starter
- spring-boot-starter-aop：快速集成AOP切面编程的Starter
- spring-boot-starter-log4j2：快速集成log4j2的Starter
- spring-boot-starter-mail：快速集成邮件功能的Starter
- spring-boot-starter-quartz：定时任务的Starter

Spring还提供了很多Starter给我们使用，使用这些约定好的Starter，可以更方便的集成第三方组件使用Starter中提供的通用操作方法进行操作，不需要再很麻烦的通过一个个依赖去导入再配置来实现件集成。

4、自定义SpringBoot Starter

4.1 开发流程

1. 创建Start项目
2. 定义Starter需要的配置类（Properties）
3. 编写Starter项目的业务功能
4. 编写自动配置类

5. 编写spring.factories文件来加载自动配置类
6. 打包安装Starter依赖
7. 其他项目引用starter并使用

4.2 案例一：自定义实现通用短信发送功能的starter

1、创建 sms-spring-boot-starter 项目（使用IDEA创建）

- 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/
MLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xs
/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.4.10</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>org.apache.mf</groupId>
    <artifactId>sms-spring-boot-starter</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>sms-spring-boot-starter</name>
    <description>sms-spring-boot-starter</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-configuration-processor</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
```

```
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <classifier>exec</classifier>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

2、定义Starter需要的配置类：SmsProperties

```
package org.apache.mf.config;

import lombok.Data;
import org.springframework.boot.context.properties.ConfigurationProperties;

import java.io.Serializable;

/**
 * @Author: kyrie  @date: 2023/1/28 16:40
 * @Description:
 * @Package: org.apache.mf.config
 */
@ConfigurationProperties("global.sms")
public class SmsProperties implements Serializable {
    private static final long serialVersionUID = -3014361972477357324L;

    /**
     * 访问ID
     */
    private String accessKeyId;
    /**
     * 访问凭证
     */
    private String accessKeySecret;

    public String getAccessKeyId() {
        return accessKeyId;
    }

    public void setAccessKeyId(String accessKeyId) {
        this.accessKeyId = accessKeyId;
    }

    public String getAccessKeySecret() {
        return accessKeySecret;
```

```
    }

    public void setAccessKeySecret(String accessKeySecret) {
        this.accessKeySecret = accessKeySecret;
    }
}
```

3、编写Starter项目的业务功能：发送短信

- 接口：ISmsService

```
package org.apache.mf.service;

/**
 * @Author: kyrie  @date: 2023/1/28 16:44
 * @Description:
 * @Package: org.apache.mf.service
 */
public interface ISmsService {

    /**
     * 发送短信
     * @param phone          手机号
     * @param areaCode       手机区号
     * @param smsTemplateCode 短信模版code
     * @param data           发送数据
     */
    void send(String phone, String areaCode, String smsTemplateCode, String data);
}
```

- 实现类：SmsServiceImpl

```
package org.apache.mf.service.impl;

import lombok.extern.slf4j.Slf4j;
import org.apache.mf.service.ISmsService;

/**
 * @Author: kyrie  @date: 2023/1/28 16:47
 * @Description:
 * @Package: org.apache.mf.service.impl
 */
@Slf4j
public class SmsServiceImpl implements ISmsService {

    /**
     * 访问ID
     */
    private String accessKeyId;
    /**
     * 访问凭证
```

```
     */
    private String accessKeySecret;

    public SmsServiceImpl(String accessKeyId, String accessKeySecret) {
        this.accessKeyId = accessKeyId;
        this.accessKeySecret = accessKeySecret;
    }

    /**
     * 发送短信
     *
     * @param phone          手机号
     * @param areaCode       手机区号
     * @param smsTemplateCode 短信模版code
     * @param data           发送数据
     */
    @Override
    public void send(String phone, String areaCode, String smsTemplateCode, String data) {
        log.info("##### 短信发送鉴权信息：accessKeyId={}, accessKeySecret={}", accessKeyId, ac
essKeySecret);
        log.info("##### 短信发送: 接收手机号：{}，接收手机区号：{}，短信模版code：{}，短信内容
：{}", phone, areaCode, smsTemplateCode, data);
        // todo 具体业务逻辑自行实现
    }
}
```

4、编写自动配置类：SmsAutoConfig

```
package org.apache.mf.config;

import org.apache.mf.service.impl.SmsServiceImpl;
import org.springframework.boot.context.properties.EnableConfigurationProperties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import javax.annotation.Resource;

/**
 * @Author: kyrie  @date: 2023/1/28 16:51
 * @Description: 配置加载类
 * @Package: org.apache.mf.config
 */
@Configuration
@EnableConfigurationProperties({SmsProperties.class})
public class SmsAutoConfig {
    @Resource
    private SmsProperties smsProperties;

    @Bean
    public SmsServiceImpl smsServiceImpl() {
        return new SmsServiceImpl(smsProperties.getAccessKeyId(), smsProperties.getAccessKey
ecret());
    }
```

```
}
```

5、编写spring.factories文件来加载自动配置类

在resources目录下创建META-INF/spring.factories文件，编写自动配置信息

```
org.springframework.boot.autoconfigure.EnableAutoConfiguration=org.apache.mf.config.SmAutoConfig
```

6、打包安装Starter依赖

使用IDEA Maven工具 clean 和 install将依赖安装到本地（有私服可以安装到私服）

7、其他项目引用starter并使用

```
<dependency>
    <groupId>org.apache.mf</groupId>
    <artifactId>sms-spring-boot-starter</artifactId>
    <version>0.0.1-SNAPSHOT</version>
</dependency>

@Service
@Slf4j
@RefreshScope
public class OrderAppServiceImpl implements OrderAppService {

    @Value("${order.orderNo}")
    private String orderNo;
    @Resource
    private UserAppApiRemoteFacade userAppApiRemoteFacade;

    @Resource
    private ISmsService smsService;

    @Override
    public UserOrderResDTO getOrderNo() {
        log.info("读取配置中心数据为: {}", orderNo);
        UserInfoResDTO userInfoResDTO = userAppApiRemoteFacade.getUserName();
        return UserOrderResDTO.builder().orderNo(orderNo).userName(userInfoResDTO.getNicName()).headUrl(userInfoResDTO.getHeadUrl()).build();
    }

    @Override
    public String send(SmsSendDTO smsSendDTO) {
        smsService.send(smsSendDTO.getPhone(), smsSendDTO.getAreaCode(), smsSendDTO.geSmsTemplateCode(), smsSendDTO.getData());
        return "发送成功！";
    }
}
```

4.2 案例二：自定义实现AOP切面日志的Starter

详细请参考源码：

- 短信starter： https://gitee.com/kyrie_code/sms-spring-boot-starter.git
- aop日志starter： https://gitee.com/kyrie_code/log-aspect-spring-boot-starter.git