自定义 starter　spring-boot-starter-weather

创建自定义的 starter，有两个重要的部分，一个是 resources/META-INF/spring.factories 文件，在 springboot 启动时通过扫描该文件来自动加载配置类。另一个是**AutoConfigure 类，这个类就是自动配置类，在 springboot 启动时被自动加载配置。

可以参考 DataSourceAutoConfiguration ，现在编写一个 天气服务的自动配置类 WeatherAutoConfiguration

1，新建 starter 项目

新建一个 module ，artifactId 为
spring-boot-starter-weather （可以在 spring boot V2.X.X 版本申请加入吗？^_^ ）

```
<groupId>com.allen.spring.src.learning</groupId>
<artifactId>spring-boot-starter-weather</artifactId>
<version>1.0.0-SNAPSHOT</version>
```

2，定义属性 WeatherProperties

```
@ConfigurationProperties(prefix = "spring.weather")
public class WeatherProperties {
    private String url;
    private String day;
}
```

3，定义要提供的服务 WeatherService 以及 服务的实现 WeatherServiceImpl

```
public interface WeatherService {
    String getWeather(String city);
}

public class WeatherServiceImpl implements WeatherService {

    private WeatherProperties weatherProperties;

    public WeatherServiceImpl(WeatherProperties weatherProperties) {
        this.weatherProperties=weatherProperties;
    }


    @Override
    public String getWeather(String city) {
        String url=weatherProperties.getUrl();
        String day=weatherProperties.getDay();
        return request(url,day,city);
    }

    public String request(String url,String day,String city)
    {
            return new StringBuilder(city).append(" cloudy ,23-27 ℃
        ").append(day).toString();
    }

}
```

4，导入自动装配依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-autoconfigure</artifactId>
    <version>1.5.4.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-configuration-processor</artifactId>
    <version>2.0.3.RELEASE</version>
    <optional>true</optional>
</dependency>
```

5，定义 WeatherAutoConfiguration 的实现

里面的主要逻辑 是判断 类路径是否有 WeatherService.class，如果有，则定义一个 bean WeatherService

```
@Configuration
@EnableConfigurationProperties(WeatherProperties.class)
@ConditionalOnClass(WeatherService.class)
@ConditionalOnProperty(prefix = "weather",value = "enable",matchIfMissing = false)
public class WeatherAutoConfiguration {

    @Autowired
    private WeatherProperties weatherProperties;

    @Bean
    @ConditionalOnMissingBean(WeatherService.class)
    public WeatherService weatherService(){
        WeatherService weatherService = new
    WeatherServiceImpl(weatherProperties);
        return weatherService;
    }

}
```

6，在/META-INF/spring.factories 里面添加 自动配置

```
org.springframework.boot.autoconfigure.EnableAutoConfiguration=com.allen.spring.
src.learning.weather.WeatherAutoConfiguration
```

7，另外新建一个工程 testWeatherService，依赖这个 starter

```
<groupId>com.allen.spring.src.learning</groupId>
<artifactId>testWeatherService</artifactId>
<version>1.0.0-SNAPSHOT</version>

<dependencies>
```

```
    <dependency>
        <groupId>com.allen.spring.src.learning</groupId>
        <artifactId>spring-boot-starter-weather</artifactId>
        <version>1.0.0-SNAPSHOT</version>
    </dependency>
</dependencies>
```

8，注入 spring-boot-starter-weather 天气服务 WeatherService，调用里面的函数

```
@RestController
public class WeatherController {

@Autowired
private WeatherService weatherService;

@RequestMapping("/get")
public String getWeather(@RequestParam("city") String city)
{
return weatherService.getWeather(city);
}
}
```

9，在 testWeatherService 工程里面 添加属性文件配置

```
weather.enable=true
spring.weather.url=http://www.weather.com.cn/
spring.weather.day=2019-03-02
```

或者 也可以定义一个 runner

```
@Component
@Slf4j
public class GetWeatherRunner implements ApplicationRunner{

    @Autowired
    private WeatherService weatherService;

    @Override
    public void run(ApplicationArguments args) throws Exception {

        String city="shenzheng";
        String weather=weatherService.getWeather(city);

        log.info("weather={}",weather);

    }
}
```

测试： 在 IE 上输入 ：http://localhost:8080/get?city=beijing
显示 ： beijing cloudy ,23-27 ℃ 2019-03-02