SpringBoot整合CXF
1. 服务提供(服务方)
1. pom.xml中引入cxf依赖
2. yaml配置
3. 编写接口及实现类
3.1 编写接口
3.2 接口的实现类
4. CXF配置及服务发布
1. 使用JDK自带的 JAX-WS 发布WebService服务
2. 使用CXF发布WEbService服务
5. 打开浏览器访问
6. 动态发布多个WS服务
1. 实现方式
2. 编码
2. 服务消费(客户端)
1.生成代码调用(jdk)
2. 生成代码调用(cxf)
3. CXF动态调用WS服务(不生成Java代码)
4. 生成代码调用WS常见问题分析
1. jdk生成WS调用代码时错误
2. CXF生成WS调用代码时错误
3. 原因分析
4. 解决方法
4. 接口与接口的实现类中标注javax.jws注解的区别
1. 接口上标注jws注解
1. 接口类
2. 接口的实现类
3. 发布后访问效果
2. 接口的实现类标jws注解
1. 接口
2. 接口的实现类
3. 发布后访问效果
3. 接口及接口实现类上都标注jws注解
4. 结论

时间	状态	作者	内容	版本	备注
2022-01-28	创建	jinshengyuan	Springboot整合cxf	v1.0	

SpringBoot整合CXF

> WebService技术实现
>
> - jdk自带的JAX-WS
> - Apache Axis1
> - Apache Axis2
> - Apache CXF

> Springboot官网：https://spring.io/projects/spring-boot
>
> CXF官网：http://cxf.apache.org/
>
> 官网文档： http://cxf.apache.org/docs/index.html

1. 服务提供(服务方)

1. pom.xml中引入cxf依赖

```
<!--WebService CXF依赖 begin-->
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-spring-boot-starter-jaxws</artifactId>
            <version>3.5.0</version>
        </dependency>

        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-frontend-jaxws</artifactId>
            <version>3.5.0</version>
        </dependency>

        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-transports-http</artifactId>
            <version>3.5.0</version>
        </dependency>
        <!--WebService CXF依赖 end-->
```

完整的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/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.6.3</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.yuan</groupId>
    <artifactId>yuan-demo</artifactId>
    <version>1.0.0</version>
    <name>yuan-boot-cxf</name>
    <description>Spring Boot Integrate Apache CXF</description>
    <properties>
        <java.version>1.8</java.version>
    </properties>
    <dependencies>

        <!--WebService CXF依赖 begin-->
        <dependency>
```

```
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-spring-boot-starter-jaxws</artifactId>
            <version>3.5.0</version>
        </dependency>

        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-frontend-jaxws</artifactId>
            <version>3.5.0</version>
        </dependency>

        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-transports-http</artifactId>
            <version>3.5.0</version>
        </dependency>
        <!--WebService CXF依赖 end-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </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>
                    <excludes>
                        <exclude>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

2. yaml配置

```yaml
server:
  port: 8101
spring:
  application:
    name: yuan-boot-cxf
  #遇到相同类名覆盖注册
  main:
    allow-bean-definition-overriding: true
# 集成cxf,默认：cxf.path=/services
cxf:
  path: /ws
```

3. 编写接口及实现类

3.1 编写接口

```java
package com.yuan.cxf.serivce;

/**
 * description: HelloService接口
 *
 * @author:jinshengyuan
 * @date: 2022-1-28
 */
public interface HelloService {
    String sayHello(String name);
}
```

3.2 接口的实现类

```java
package com.yuan.cxf.serivce.impl;

import com.yuan.cxf.serivce.HelloService;
import org.springframework.stereotype.Service;

import javax.jws.WebMethod;
```

```java
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;

/**
 * description:  HelloService的实现
 *
 * @author:jinshengyuan
 * @date: 2022-1-28
 */
@WebService
@Service
public class HelloServiceImpl implements HelloService {
    /**
     * description: sayHello方法
     *
     * @param name 姓名
     * @return String
     * @author:jinshengyuan
     * @date: 2022-1-28
     */
    @WebMethod
    @WebResult(name = "resultName")
    @Override
    public String sayHello(@WebParam(name = "name") String name) {
        return "Hello ," + name;
    }
}
```

4. CXF配置及服务发布

1. 使用JDK自带的 JAX-WS 发布WebService服务

1. WSPublish.java

```java
import serivce.impl.HelloServiceImpl;

import javax.xml.ws.Endpoint;
/**
 * description: JAX-WS发布WebService
 *
 * @author:jinshengyuan
 * @date: 2022-1-28
 */
public class WSPublish {
    public static void main(String[] args) {
        String address = "http://localhost:8888/ws";
        Endpoint.publish(address,new HelloServiceImpl());
        System.out.println("服务已发布");
    }
}
```

2. 打开浏览器输入 http://localhost:8888/ws?wsdl 如下:

2. 使用CXF发布WEbService服务

CxfConfig.java

```
package com.yuan.cxf.config;

import com.yuan.cxf.serivce.impl.HelloServiceImpl;
import org.apache.cxf.Bus;
import org.apache.cxf.jaxws.EndpointImpl;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import javax.xml.ws.Endpoint;

/**
 * description: CXF服务发布配置
 *
 * @author:jinshengyuan
 * @date: 2022-1-28
 */
@Configuration
public class CxfConfig {

    @Autowired
    private Bus bus;

    @Autowired
    private HelloServiceImpl helloService;

    /**
     * description: 发布服务
     *
     * @return Endpoint
     * @author:jinshengyuan
     * @date: 2022-1-28
     */
    @Bean
    public Endpoint helloServer() {
```

```
        EndpointImpl endpoint = new EndpointImpl(bus, helloService);
        endpoint.publish("/hello");
        return endpoint;
    }
}
```

5. 打开浏览器访问

访问地址：http://localhost:8101/ws/hello?wsdl

6. 动态发布多个WS服务

> 业务场景: 当要集成多方的WebService接口时，每次都要单独注册一个类似下面的Bean来发布WS接口,如果多了非常繁琐及代码冗余

```
@Bean
public Endpoint helloServer() {
 EndpointImpl endpoint = new EndpointImpl(bus, helloService);
 endpoint.publish("/hello");
 repackage com.yuan.cxf.config;

import com.yuan.cxf.service.impl.DeptInfoWSServiceImpl;
import com.yuan.cxf.service.impl.HelloServiceImpl;
import com.yuan.cxf.service.impl.UserInfoWSServiceImpl;
import org.apache.cxf.Bus;
import org.apache.cxf.jaxws.EndpointImpl;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import javax.xml.ws.Endpoint;

@Configuration
public class CxfConfig {
 @Autowired
 private Bus bus;
```

```
    @Autowired
    private HelloServiceImpl helloService;


    @Autowired
    private UserInfoWSServiceImpl userInfoWSService;


    @Autowired
    private DeptInfoWSServiceImpl deptInfoWSService;

    @Bean
    public Endpoint helloServer(){
        EndpointImpl endpoint = new EndpointImpl(bus,helloService);
        endpoint.publish("/hello");
        return endpoint;
    }

    @Bean
    public Endpoint userInfo(){
        EndpointImpl endpoint = new EndpointImpl(bus,userInfoWSService);
        endpoint.publish("/userInfo");
        return endpoint;
    }


    @Bean
    public Endpoint deptInfo(){
        EndpointImpl endpoint = new EndpointImpl(bus,deptInfoWSService);
        endpoint.publish("/deptInfo");
        return endpoint;
    }
}
turn endpoint;
}
```

1. 实现方式

> `ApplicationRunner`或`CommandLineRunner`使用场景：在Springboot容器启动完毕后要执行一些特定的业务代码，如读取Redis缓存数据、特定的配置文件、数据库等等相关数据
>
> - 可以同时配置多个`ApplicationRunner`或`CommandLineRunner`,如果配置了多个则启动的优先级可以通过`@Order(1)`来指定，值越小优先级越高
> - `ApplicationRunner`中的run方法接收一个`ApplicationArguments`类型的参数`run(ApplicationArguments args)`，如：当把应用打成一个jar包运行的时候，后面跟着的命令行参数可以通过ApplicationArguments拿到
> - `CommandLineRunner`中的run方法接收一个String类型的可变参数列表`run(String... args)`

1. 自定义动态发布接口的注解
2. 在业务类中标注自定义动态发布WebService接口的注解
3. 使用Springboot提供的`ApplicationRunner`或`CommandLineRunner`接口来实现当容器启动完毕后动态发布多个webService接口

2. 编码

1. 自定义动态发布WS的注解

AutoPublishWS.java

```
package com.yuan.cxf.annotation;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
 * description: 动态发布WebService的自定义注解
 *
 * @author:jinshengyuan
 * @date: 2022-1-29
 */
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.TYPE})
public @interface AutoPublishWS {
    /**
     * description: 定义WebService的发布路径属性
     *
     * @return String
     * @author:jinshengyuan
     * @date: 2022-1-29
     */
    String publishPath();
}
```

2. 在业务接口的实现类上标注动态发布WS的自定义注解
3. 定义自动发布WS的配置类

AutoPublishWSEndpoint.java

```
package com.yuan.cxf.config;

import com.yuan.cxf.annotation.AutoPublishWS;
import lombok.extern.slf4j.Slf4j;
import org.apache.cxf.Bus;
import org.apache.cxf.jaxws.EndpointImpl;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.core.annotation.Order;
import org.springframework.core.env.Environment;
import org.springframework.stereotype.Component;
import org.springframework.web.context.WebApplicationContext;

import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.ArrayList;
import java.util.List;
```

```java
/**
 * description: 自动发布
 *
 * @author:jinshengyuan
 * @date: 2022-1-29
 */
@Slf4j
@Component
@Order(1)
public class AutoPublishWSEndpoint implements ApplicationRunner {
    //取出WS的根路径
    @Value("${cxf.path}")
    public String cxfPath;
    //取出AppName
    @Value("${spring.application.name}")
    public String appName;

    //注入CXF Bus
    @Autowired
    private Bus bus;
    //注入Spring Web中Servlet上下文Web容器
    @Autowired
    private WebApplicationContext webApplicationContext;

    @Override
    public void run(ApplicationArguments args) throws Exception {
        log.info("AutoPublishWSEndpoint===发布开始");
        //根据注解获取beanNames
        String[] beanNames =
webApplicationContext.getBeanNamesForAnnotation(AutoPublishWS.class);
        EndpointImpl endpoint;
        List<String> cxfPathList = new ArrayList<>();
        for (String beanName : beanNames) {
            //根据beanName获取属性名
            String publishPath =
webApplicationContext.getType(beanName).getAnnotation(AutoPublishWS.class).publi
shPath();
            //发布WebService接口
            endpoint = new EndpointImpl(bus,
webApplicationContext.getBean(beanName));
            endpoint.publish(publishPath);
            cxfPathList.add(publishPath);
            log.info(String.format("%s", publishPath));
        }
        log.info("AutoPublishWSEndpoint===发布结束");
        Environment environment = webApplicationContext.getEnvironment();
        address(environment, cxfPathList);
    }

    /**
     * description: 打印访问路径
     *
     * @param environment 运行环境
     * @param cxfPathList 发布的WS路径
     * @return String
     * @author:jinshengyuan
     * @date: 2022-1-29
     */
```

```
    public void address(Environment environment, List<String> cxfPathList) {
        try {
            String ip = InetAddress.getLocalHost().getHostAddress();
            String port = environment.getProperty("server.port");
            String path = environment.getProperty("server.servlet.context-
path");
            if (path == null) {
                path = "";
            }
            log.info("\n----------------------------------------------------------
--\n\t" +
                    "Application " + appName + " is running! Access URLs:\n\t" +
                    "Local: \t\thttp://localhost:" + port + path + "/\n\t" +
                    "External: \thttp://" + ip + ":" + port + path + "/" +
                    "\n----------------------------------------------------------
");
            if (cxfPathList.size() > 0) {
                StringBuilder logStr = new StringBuilder();
                logStr.append("\n\tWSDL URLS:\n\t");
                String finalPath = path;
                cxfPathList.forEach(s -> {
                    logStr.append("\t\thttp://localhost:" + port + finalPath +
cxfPath + "/" + s + "?wsdl\n\t");
                    logStr.append("\t\thttp://" + ip + ":" + port + finalPath +
cxfPath + "/" + s + "?wsdl\n\t");
                });
                log.info(logStr.toString());
            }
        } catch (UnknownHostException e) {
            e.printStackTrace();
        }
    }
}
```

2. 服务消费(客户端)

WSDL文档

- 直接通过http请求获取wsdl文档
- 将请求后的wsdl另存为本地文件

```
wsimport -keep -encoding utf-8 -verbose    http://localhost:8101/ws/hello?wsdl

wsimport -p com.test -keep -encoding utf-8 -verbose
http://localhost:8101/ws/autoUserInfo?wsdl
```

1.生成代码调用(jdk)

1. wsimport命令使用
2. 解析WSDL文档

```
wsdl2java -keep -encoding utf-8 -verbose  http://localhost:8101/ws/hello?wsdl

wsdl2java -p com.test -keep -encoding utf-8 -verbose
http://localhost:8101/ws/autoDeptInfo?wsdl
```

2. 生成代码调用(cxf)

1. 下载cxf二进制包
2. 快速下载地址：https://www.apache.org/dyn/closer.lua/cxf/3.5.0/apache-cxf-3.5.0.zip
3. wsdl2java命令使用
4. 生成java代码解析WSDL文档

3. CXF动态调用WS服务(不生成Java代码)

> 注意：仅限第三方用java语言发布的WebService接口动态调用，不支持用.net发布的WebService接口

1. 添加frontend调用依赖

```
<!-- https://mvnrepository.com/artifact/org.apache.cxf/cxf-rt-frontend-jaxws -->
<dependency>
    <groupId>org.apache.cxf</groupId>
    <artifactId>cxf-rt-frontend-jaxws</artifactId>
    <version>3.5.0</version>
</dependency>
```

4. 生成代码调用WS常见问题分析

> 注意：不管是JDK自带的wsmport命令还是cxf的wsdl2java生成`非java语言(如.net)`开发的WebService接口调用代码时，都会出现`解析组件 's:schema' 时出错。`的错误，比如以生成接口`http://www.webxml.com.cn/WebServices/WeatherWebService.asmx?wsdl`代码为例
>
> 1. wsimport命令生成代码时提示错误：`无法将名称 's:schema' 解析为 'element declaration' 组件。`
> 2. cxf的wsdl2java命令生成代码时提示错误：`undefined element declaration 's:schema'`

1. jdk生成WS调用代码时错误

```
wsimport -keep -verbose -encoding utf-8 WeatherWebService.wsdl

wsimport -keep -verbose -encoding utf-8
http://www.webxml.com.cn/WebServices/WeatherWebService.asmx?wsdl
```

1. 生成源代码问题：`无法将名称 's:schema' 解析为 'element declaration' 组件。`

```
I:\yuan-boot-integrate\yuan-ws-jdk\src\main\java>wsimport -keep -verbose -
encoding utf-8 http://www.webxml.com.cn/WebServices/WeatherWebService.asmx?wsdl

正在解析 WSDL...
```

```
[WARNING] src-resolve.4.2: 解析组件 's:schema' 时出错。在该组件中检测到 's:schema' 位于名称空间 'http://www.w3.org/2001/XMLSchema' 中，但无法从方案文档 'http://www.webxml.com.cn/WebServices/WeatherWebService.asmx?wsdl#types?schema1' 引用此名称空间的组件。如果这是不正确的名称空间，则很可能需要更改 's:schema' 的前缀。如果这是正确的名称空间，则应将适当的 'import' 标记添加到 'http://www.webxml.com.cn/WebServices/WeatherWebService.asmx?wsdl#types?schema1'。
  http://www.webxml.com.cn/WebServices/WeatherWebService.asmx?wsdl#types?schema1 的第 44 行

[WARNING] src-resolve: 无法将名称 's:schema' 解析为 'element declaration' 组件。
  http://www.webxml.com.cn/WebServices/WeatherWebService.asmx?wsdl#types?schema1 的第 44 行

[ERROR] undefined element declaration 's:schema'
  http://www.webxml.com.cn/WebServices/WeatherWebService.asmx?wsdl的第 44 行

[ERROR] undefined element declaration 's:schema'
  http://www.webxml.com.cn/WebServices/WeatherWebService.asmx?wsdl的第 85 行

Exception in thread "main" com.sun.tools.internal.ws.wscompile.AbortException
        at com.sun.tools.internal.ws.processor.modeler.wsdl.JAXBModelBuilder.bind(JAXBModelBuilder.java:129)
        at com.sun.tools.internal.ws.processor.modeler.wsdl.WSDLModeler.buildJAXBModel(WSDLModeler.java:2283)
        at com.sun.tools.internal.ws.processor.modeler.wsdl.WSDLModeler.internalBuildModel(WSDLModeler.java:183)
        at com.sun.tools.internal.ws.processor.modeler.wsdl.WSDLModeler.buildModel(WSDLModeler.java:126)
        at com.sun.tools.internal.ws.wscompile.WsimportTool.buildWsdlModel(WsimportTool.java:429)
        at com.sun.tools.internal.ws.wscompile.WsimportTool.run(WsimportTool.java:190)
        at com.sun.tools.internal.ws.wscompile.WsimportTool.run(WsimportTool.java:168)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at com.sun.tools.internal.ws.Invoker.invoke(Invoker.java:159)
        at com.sun.tools.internal.ws.WsImport.main(WsImport.java:42)
```

2. CXF生成WS调用代码时错误

```
wsdl2java -keep -verbose -encoding utf-8
http://www.webxml.com.cn/WebServices/WeatherWebService.asmx?wsdl
wsdl2java -keep -verbose -encoding utf-8 WeatherWebService.wsdl
```

1. 生成源代码的问题: `undefined element declaration 's:schema'`

```
I:\yuan-boot-integrate\yuan-ws-jdk\src\main\java>wsdl2java -keep -verbose -
encoding utf-8 http://www.webxml.com.cn/WebServices/WeatherWebService.asmx?wsdl
Loading FrontEnd jaxws ...
Loading DataBinding jaxb ...
wsdl2java -keep -verbose -encoding utf-8
http://www.webxml.com.cn/WebServices/WeatherWebService.asmx?wsdl
wsdl2java - Apache CXF 3.5.0

二月 02, 2022 10:04:35 上午 org.apache.cxf.wsdl11.WSDLServiceBuilder
checkForWrapped
信息: Operation {http://WebXml.com.cn/}getSupportCity cannot be unwrapped, input
message must reference global element declaration with same localname as
operation
.....
WSDLToJava Error: http://www.webxml.com.cn/WebServices/WeatherWebService.asmx?
wsdl [44,19]: undefined element declaration 's:schema'
http://www.webxml.com.cn/WebServices/WeatherWebService.asmx?wsdl [85,13]:
undefined element declaration 's:schema'

org.apache.cxf.tools.common.ToolException:
http://www.webxml.com.cn/WebServices/WeatherWebService.asmx?wsdl [44,19]:
undefined element declaration 's:schema'
http://www.webxml.com.cn/WebServices/WeatherWebService.asmx?wsdl [85,13]:
undefined element declaration 's:schema'
        at
org.apache.cxf.tools.common.ToolErrorListener.throwToolException(ToolErrorListen
er.java:87)
        at
org.apache.cxf.tools.wsdlto.WSDLToJavaContainer.execute(WSDLToJavaContainer.java
:158)
        at
org.apache.cxf.tools.wsdlto.WSDLToJavaContainer.execute(WSDLToJavaContainer.java
:402)
        at
org.apache.cxf.tools.common.toolspec.ToolRunner.runTool(ToolRunner.java:105)
        at org.apache.cxf.tools.wsdlto.WSDLToJava.run(WSDLToJava.java:113)
        at org.apache.cxf.tools.wsdlto.WSDLToJava.run(WSDLToJava.java:86)
        at org.apache.cxf.tools.wsdlto.WSDLToJava.main(WSDLToJava.java:184)
        Suppressed: org.apache.cxf.tools.common.ToolException:
http://www.webxml.com.cn/WebServices/WeatherWebService.asmx?wsdl [44,19]:
undefined element declaration 's:schema'
                ... 7 more
        Caused by: org.xml.sax.SAXParseExceptionpublicId:
http://www.webxml.com.cn/WebServices/WeatherWebService.asmx?wsdl; systemId:
http://www.webxml.com.cn/WebServices/WeatherWebService.asm
x?wsdl; lineNumber: 44; columnNumber: 19; undefined element declaration
's:schema'
```

```
                at
com.sun.xml.xsom.impl.parser.ParserContext$1.reportError(ParserContext.java:150)
                at
com.sun.xml.xsom.impl.parser.NGCCRuntimeEx.reportError(NGCCRuntimeEx.java:149)
                at
com.sun.xml.xsom.impl.parser.DelayedRef.resolve(DelayedRef.java:80)
                at
com.sun.xml.xsom.impl.parser.DelayedRef.run(DelayedRef.java:55)
                at
com.sun.xml.xsom.impl.parser.ParserContext.getResult(ParserContext.java:105)
                at
com.sun.xml.xsom.parser.XSOMParser.getResult(XSOMParser.java:184)
                at
com.sun.tools.xjc.ModelLoader.createXSOM(ModelLoader.java:479)
                at
com.sun.tools.xjc.api.impl.s2j.SchemaCompilerImpl.bind(SchemaCompilerImpl.java:2
40)
                at
com.sun.tools.xjc.api.impl.s2j.SchemaCompilerImpl.bind(SchemaCompilerImpl.java:6
7)
                at
org.apache.cxf.tools.wsdlto.databinding.jaxb.JAXBDataBinding.initialize(JAXBData
Binding.java:445)
                at
org.apache.cxf.tools.wsdlto.WSDLToJavaContainer.generateTypes(WSDLToJavaContaine
r.java:711)
                at
org.apache.cxf.tools.wsdlto.WSDLToJavaContainer.processWsdl(WSDLToJavaContainer.
java:259)
                at
org.apache.cxf.tools.wsdlto.WSDLToJavaContainer.execute(WSDLToJavaContainer.java
:156)
                ... 5 more
        Suppressed: org.apache.cxf.tools.common.ToolException:
http://www.webxml.com.cn/WebServices/WeatherWebService.asmx?wsdl [85,13]:
undefined element declaration 's:schema'
                ... 7 more
        Caused by: org.xml.sax.SAXParseExceptionpublicId:
http://www.webxml.com.cn/WebServices/WeatherWebService.asmx?wsdl; systemId:
http://www.webxml.com.cn/WebServices/WeatherWebService.asm
x?wsdl; lineNumber: 85; columnNumber: 13; undefined element declaration
's:schema'
                at
com.sun.xml.xsom.impl.parser.ParserContext$1.reportError(ParserContext.java:150)
                at
com.sun.xml.xsom.impl.parser.NGCCRuntimeEx.reportError(NGCCRuntimeEx.java:149)
                at
com.sun.xml.xsom.impl.parser.DelayedRef.resolve(DelayedRef.java:80)
                at
com.sun.xml.xsom.impl.parser.DelayedRef.run(DelayedRef.java:55)
                at
com.sun.xml.xsom.impl.parser.ParserContext.getResult(ParserContext.java:105)
                at
com.sun.xml.xsom.parser.XSOMParser.getResult(XSOMParser.java:184)
                at
com.sun.tools.xjc.ModelLoader.createXSOM(ModelLoader.java:479)
```

```
                at
com.sun.tools.xjc.api.impl.s2j.SchemaCompilerImpl.bind(SchemaCompilerImpl.java:2
40)
                at
com.sun.tools.xjc.api.impl.s2j.SchemaCompilerImpl.bind(SchemaCompilerImpl.java:6
7)
                at
org.apache.cxf.tools.wsdlto.databinding.jaxb.JAXBDataBinding.initialize(JAXBData
Binding.java:445)
                at
org.apache.cxf.tools.wsdlto.WSDLToJavaContainer.generateTypes(WSDLToJavaContaine
r.java:711)
                at
org.apache.cxf.tools.wsdlto.WSDLToJavaContainer.processWsdl(WSDLToJavaContainer.
java:259)
                at
org.apache.cxf.tools.wsdlto.WSDLToJavaContainer.execute(WSDLToJavaContainer.java
:156)
                ... 5 more

I:\yuan-boot-integrate\yuan-ws-jdk\src\main\java>
```

3. 原因分析

> 不管是wsimport还是wsdl2java解析WSDL文档是，都是通过`JAXB`将xml文件转换到java类的，而.net发布的wsdl文档里面的复合类型中会包含`<s:element ref="s:schema" /><s:any />`的元素，而JAXB似乎不支持ref这种引用类型的属性,所以会导致生成文档是出现错误

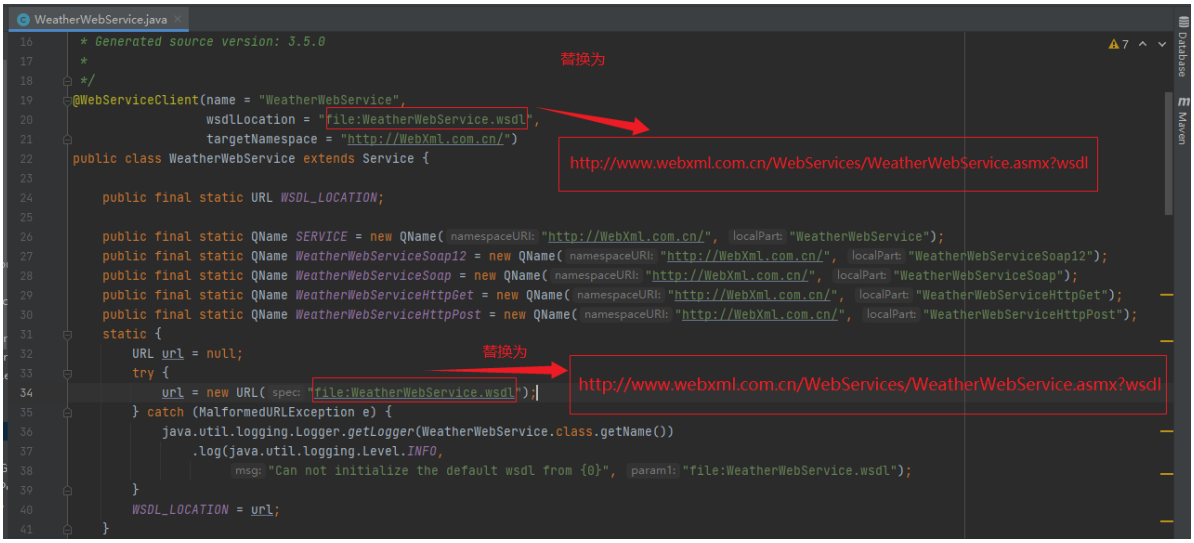
4. 解决方法

以获取天气预报的免费WebService接口
`http://www.webxml.com.cn/WebServices/WeatherWebService.asmx?wsdl`为例

1. 先访问上面接口，并在浏览器中另存为本地xml文件`WeatherWebService.xml`
2. 打开`WeatherWebService.xml`文件,查找所有的`<s:element ref="s:schema" /><s:any />`并替换为`<s:any minOccurs="2"  maxOccurs="2"/>`，一般都会查找到两处
3. 将`WeatherWebService.xml`重命名为`WeatherWebService.wsdl`
4. 将`WeatherWebService.wsdl`拷贝到指定的位置，如`D:\WeatherWebService.wsdl`
5. 生成java代码

```
wsimport -keep -verbose -encoding utf-8 D:\WeatherWebService.wsdl
# 或
wsdl2java -keep -verbose -encoding utf-8 D:\WeatherWebService.wsdl
```

6. 生成java代码后，将`WeatherWebService.java`中的`file:WeatherWebService.wsdl`替换为`http://www.webxml.com.cn/WebServices/WeatherWebService.asmx?wsdl`,如下图



7. 编写服务调用代码

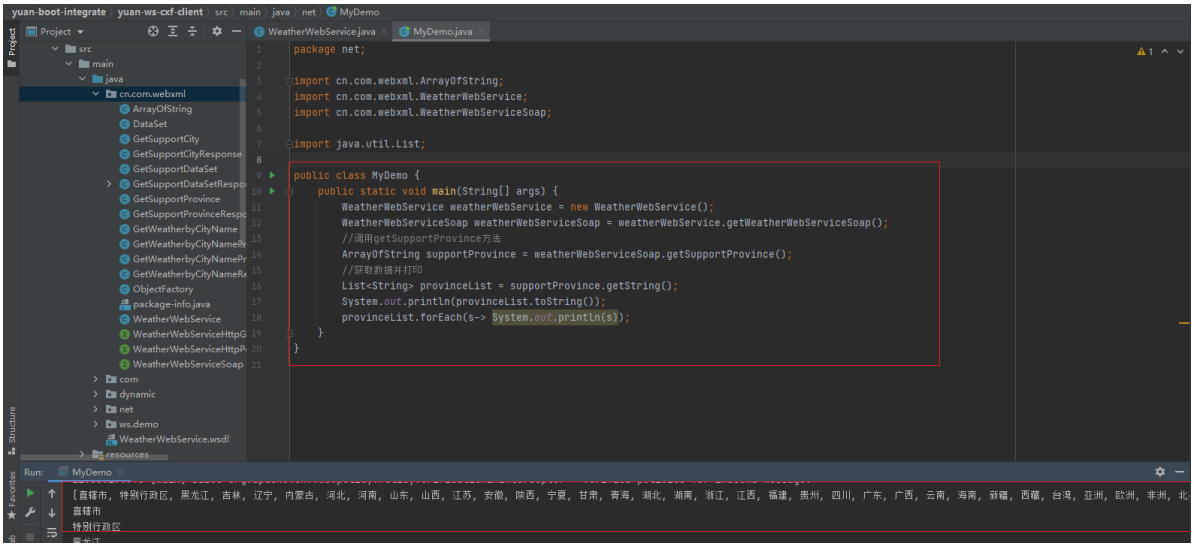
```
package net;

import cn.com.webxml.ArrayOfString;
import cn.com.webxml.WeatherWebService;
import cn.com.webxml.WeatherWebServiceSoap;

import java.util.List;

public class MyDemo {
    public static void main(String[] args) {
        WeatherWebService weatherWebService = new WeatherWebService();
        WeatherWebServiceSoap weatherWebServiceSoap =
weatherWebService.getWeatherWebServiceSoap();
        //调用getSupportProvince方法
        ArrayOfString supportProvince =
weatherWebServiceSoap.getSupportProvince();
        //获取数据并打印
        List<String> provinceList = supportProvince.getString();
        System.out.println(provinceList.toString());
        provinceList.forEach(s-> System.out.println(s));
    }
}
```

执行结果如下：



4. 接口与接口的实现类中标注javax.jws注解的区别

1. 接口上标注jws注解

1. 接口类

在接口及接口方法上标注@WebService、 @WebMethod、 @WebResult等注解

- HelloService.java

```
package com.yuan.cxf.serivce;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;

/**
 * description:
 *
 * @author:jinshengyuan
 * @date: 2022-1-28
```

```
 */
@WebService
public interface HelloService {

    /**
     * description:
     *
     * @param name 用户名
     * @return
     * @author:jinshengyuan
     * @date: 2022-1-28
     * @since v1.0.0
     */
    @WebMethod
    @WebResult(name = "resultName")
    String sayHello(@WebParam(name = "name") String name);
}
```

2. 接口的实现类

- HelloServiceImpl.java

```
package com.yuan.cxf.serivce.impl;

import com.yuan.cxf.serivce.HelloService;
import org.springframework.stereotype.Service;

import javax.jws.WebParam;

/**
 * description:  HelloService的实现
 *
 * @author:jinshengyuan
 * @date: 2022-1-28
 */
@Service
public class HelloServiceImpl implements HelloService {
    /**
     * description: sayHello方法
     *
     * @param name 姓名
     * @return String
     * @author:jinshengyuan
     * @date: 2022-1-28
     */
    @Override
    public String sayHello(@WebParam(name = "name") String name) {
        System.out.println("服务被调用了ss," + System.currentTimeMillis());
        return "Hello ," + name;
    }
}
```

3. 发布后访问效果

访问地址：http://localhost:8101/ws/hello?wsdl

1. 访问后如下图

2. 再次访问 location中的地址 http://localhost:8101/ws/hello?wsdl=HelloService.wsdl ,如下图

2. 接口的实现类标jws注解

1. 接口

HelloService.java中未标注jws的相关注解

```
package com.yuan.ws.service;

/**
 * description: Helloservice接口
 *
 * @author:jinshengyuan
 * @date: 2022-1-28
 */
```

```java
public interface HelloService {
    /**
     * description:
     *
     * @param name 用户名
     * @return
     * @author:jinshengyuan
     * @date: 2022-1-28
     * @since v1.0.0
     */
    String sayHello(String name);
}
```

2. 接口的实现类

```java
package com.yuan.ws.service.impl;

import com.yuan.ws.service.HelloService;
import org.springframework.stereotype.Service;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;

/**
 * description: HelloService的实现
 *
 * @author:jinshengyuan
 * @date: 2022-1-28
 */
@WebService
@Service
public class HelloServiceImpl implements HelloService {
    @WebMethod
    @WebResult(name = "resultName")
    @Override
    public String sayHello(@WebParam(name = "name") String name) {
        return "hello," + name;
    }
}
```

3. 发布后访问效果

访问地址：http://localhost:8101/ws/hello?wsdl

3. 接口及接口实现类上都标注jws注解

> 结论：以接口标注的注解为主，即有两层wsdl地址，与**1. 接口上标注jws注解**显示效一致

4. 结论

1. 如果jws注解标注在接口上，则会出现两层wsdl文档
2. 如果jws注解标注在接口的实现类上，则只有一层wsdl文档
3. 如果接口及接口的实现类中都标注的jws注解，则以接口上标注的为主，也即会出现两层wsdl地址
4. 建议还是将jws注解标注在接口的实现类上，这样一次性看全所有的wsdl文档