

# Anyframe CXF JAX-WS Plugin



Version 1.0.2

저작권 © 2007-2011 삼성SDS

본 문서의 저작권은 삼성SDS에 있으며 Anyframe 오픈소스 커뮤니티 활동의 목적하에서 자유로운 이용이 가능합니다. 본 문서를 복제, 배포할 경우에는 저작권자를 명시하여 주시기 바라며 본 문서를 변경하실 경우에는 원문과 변경된 내용을 표시하여 주시기 바랍니다. 원문과 변경된 문서에 대한 상업적 용도의 활용은 허용되지 않습니다. 본 문서에 오류가 있다고 판단될 경우 이슈로 등록해 주시면 적절한 조치를 취하도록 하겠습니다.

---

I. Introduction .....	1
II. JAX-WS .....	2
1. JAX-WS Frontend .....	3
1.1. Web Service 작성 .....	4
1.1.1. Samples .....	4
1.2. Spring Configuration XML - jaxws:endpoint tag 사용 .....	5
1.2.1. Samples .....	5
1.3. Server: JAX-WS Frontend API 사용 .....	7
1.3.1. Samples .....	7
1.4. Spring Configuration XML - jaxws:client tag 사용 .....	7
1.4.1. Samples .....	7
1.5. Client: JAX-WS Frontend API 사용 .....	8
1.5.1. Samples .....	8
1.6. Annotation 작성 .....	9
1.6.1. @WebService (javax.jws.WebService) .....	9
1.6.2. @WebParam (javax.jws.WebParam) .....	10
1.6.3. @WebMethod (javax.jws.WebMethod) .....	10
1.6.4. @OneWay (javax.jws.OneWay) .....	11
1.6.5. @WebResult (javax.jws.WebResult) .....	11
1.6.6. Samples .....	11
1.7. Resources .....	13
2. Asynchronous Invocation .....	14
2.1. Server Configuration .....	14
2.1.1. Samples .....	14
2.2. Client Configuration .....	18
2.2.1. Samples .....	18
2.3. Resources .....	19
3. WAS(Web Application Server) Configuration .....	21
3.1. Tomcat .....	21
3.1.1. 5.5.23, 6.0.x, 7.0.x .....	21
3.2. JEUS .....	21
3.2.1. 5.0 .....	21
3.2.2. 6.0 .....	21
3.3. WebLogic .....	22
3.3.1. 9.2 .....	22
3.3.2. 10.1 .....	23
3.3.3. 10.3.3 .....	24
4. Resources .....	26

---

# I.Introduction

웹 서비스 구현을 위해 많이 사용하고 있는 오픈소스 프레임워크에는 Apache CXF, Apache Axis/Axis2, Spring Web Services 등등 여러가지가 존재한다. cxf-jaxws plugin은 이 중 Apache CXF [<http://cxf.apache.org/>]의 JAX-WS Frontend 활용 방법을 가이드하기 위한 샘플 코드와 이 오픈 소스를 활용하는데 필요한 참조 라이브러리들로 구성되어 있다.

## Installation

Command 창에서 다음과 같이 명령어를 입력하여 cxf-jaxws plugin을 설치한다.

```
mvn anyframe:install -Dname=cxf-jaxws
```

installed(mvn anyframe:installed) 혹은 jetty:run(mvn clean jetty:run) command를 이용하여 설치 결과를 확인해볼 수 있다.

## Dependent Plugins

Plugin Name	Version Range
<a href="http://dev.anyframejava.org/docs/anyframe/plugin/optional/query/1.1.2/reference/htmlsingle/query.html">query [http://dev.anyframejava.org/docs/anyframe/plugin/optional/query/1.1.2/reference/htmlsingle/query.html]</a>	2.0.0 > *
<a href="http://dev.anyframejava.org/docs/anyframe/plugin/optional/cxf/1.0.2/reference/htmlsingle/cxf.html">cxf [http://dev.anyframejava.org/docs/anyframe/plugin/optional/cxf/1.0.2/reference/htmlsingle/cxf.html]</a>	2.0.0 > *

---

# II.JAX-WS

---

# 1.JAX-WS Frontend

Web Services 표준 API인 JAX-WS를 사용하여 Annotation 설정을 통해 Web Services를 구현할 수 있게 해주는 Frontend 모델이다.

JAX-WS(Java API for XML Web Services)는 웹 서비스를 작성하는 자바 API로써, Java EE의 일부이다. 다른 Java EE의 자바 API와 같이, JAX-WS는 Java SE 5에서 도입된 어노테이션(Annotation)을 사용하여 웹 서비스 클라이언트 및 서버 모듈의 개발 및 배포를 쉽게 하고 있다. JAX-WS는 JAX-RPC 표준을 발전 시킨 개념으로 XML의 바인딩을 위한 JAXB 표준과 표준 스트리밍 파서를 위한 SAX 표준, 기능이 향상된 새로운 SAAJ 표준을 기반으로 통합, 발전된 웹 서비스 기술 표준이다. JAX-WS의 특징을 살펴보면 다음과 같다.

- JAX-RPC에 비해 Web Service 작성 편리

JAX-RPC와 비교해보았을때 Web Service 작성이 매우 편리해진 장점을 갖고 있다. Annotation 설정을 통해 Web Service Endpoint 작성 및 자바 타입과 WSDL 간의 매핑 등을 명시적으로 수행할 수 있다. 기존의 JAX-RPC 환경에서 Web Service를 생성하기 위해 작성해야만 했던 Web Service 배치 서술자 등을 모두 Annotation으로 대체 가능함으로써 Web Service 작성의 어려움을 크게 없애준다.

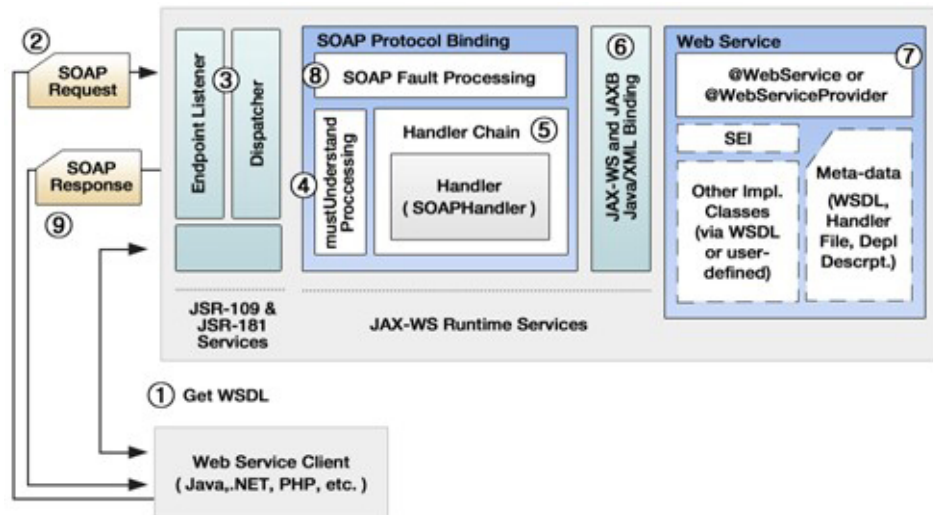
- Annotation을 통해 하는 작업

자바와 WSDL의 매핑 및 자바 타입과 WSDL에서 참조하는 메시지의 Schema 타입으로의 매핑을 담당한다. 실제 Web Service 메시지를 처리하는 Runtime Process 중에 필요한 정보를 제공하며 Web Service 호출에 응답하는데 사용된다.

- Annotation 관련 Spec.

JAX-WS 2.x에서 사용하게 되는 Annotation은 Web Services Metadata(JSR-181) Spec.과 JAX-WS 2.x Spec.에 분리 정의되어 있다. 이외의 메시지 타입 매핑을 위해 사용되는 Annotation은 JAXB 2.x Spec.에 정의되어 있다. Web Services Metadata(JSR-181) Spec.에 정의된 대표적인 Annotation으로는 다음과 같은 것들이 있다.(javax.jws.WebService, javax.jws.WebMethod, javax.jws.WebParam, javax.jws.WebResult, javax.jws.soap.SOAPBinding)

JAX-WS의 프로세스를 그림으로 살펴보면 다음과 같다.



다음은 JAX-WS Frontend를 Server와 Client 단에서 어떻게 사용해야 하는지에 대한 사용법이다. JAX-WS Frontend는 크게 JAX-WS Frontend API를 사용하여 서버와 클라이언트를 작성하는 방식과 Spring Configuration을 이용하는 방식으로 구분하여 사용될 수 있다.

- Server Configuration

## 1. Web Service 작성

## 2. 서버 구동(2가지 방식 중 택1)

- Spring Configuration XML - <jaxws:endpoint/> tag 사용 (추천 방식)
- [Server] JAX-WS Frontend API 사용

### • Client Configuration

## 1. 클라이언트 작성(2가지 방식 중 택1)

- Spring Configuration XML - <jaxws:client/> tag 사용 (추천 방식)
- [Client] JAX-WS Frontend API 사용

### • [참고] Web Services 작성 시 Annotation 설정 방법

# 1.1. Web Service 작성

Web Service로 노출시킬 서비스를 작성하는데, 일반 서비스 Bean 개발 방식과 거의 동일하다. 단, 인터페이스 클래스에 @WebService Annotation이 설정되어야 함에 유의하도록 한다.

## 1.1.1. Samples

다음은 Web Service로 노출시킬 MovieFinder Service에 대한 예제이다. 서비스는 일반 Spring Bean 개발과 동일하게 인터페이스 클래스, 구현 클래스, DAO(Data Access Object) 클래스, VO(Value Object) 클래스들로 구성되어 있다.

### • Interface Class

다음은 Movie Service의 인터페이스 클래스를 작성한 MovieService.java 의 일부이다. 인터페이스 클래스 상단에 @WebService Annotation을 작성해줘야 한다. @WebService Annotation의 속성 값들도 다양하게 존재하고, 각 메소드와 파라미터 별로 정의할 수 있는 Annotation의 종류도 여러가지가 있으나 필수 사항으로 필요한 Annotation은 @WebService뿐이다. 다른 종류의 Annotation에 대해서 추가 작성하는 방법에 대해서는 [참고] Web Services 작성 시 Annotation 설정 방법을 참고하도록 한다.

```
@WebService
public interface MovieService {

    @XmlJavaTypeAdapter(CXFMapAdapter.class)
    Map<String, Movie> get(String movieId) throws Exception;

    Page getPagingList(Movie movie, int pageIndex) throws Exception;

    중략...
}
```

### • Implementation Class

Interface Class를 구현한 클래스로 Web Service 구현과 관련된 부분 없이 작성될 수 있다. 다음은 Movie Service의 인터페이스 클래스를 구현한 MovieServiceImpl.java 의 일부이다. 내부적으로 MovieDao를 사용하여 Movie 정보를 관리하고 있다. 구현 클래스 상단에 @WebService Annotation을 작성해줄 수도 있다. 이미 인터페이스 클래스 상단에 @WebService Annotation을 작성해주었으므로 여기서는 생략시킨다. Annotation 설정 방법에 대해서는 [참고] Web Services 작성 시 Annotation 설정 방법을 참고하도록 한다.

```
@Service("cxfJaxwsMovieService")
@Transactional(rollbackFor = { Exception.class }, propagation = Propagation.REQUIRED)
public class MovieServiceImpl implements MovieService {
```

```

@Inject
@Named("cxfJaxwsMovieDao")
private MovieDao movieDao;

public Map<String, Movie> get(String movieId) throws Exception {
    return this.movieDao.get(movieId);
}

public Page getPagingList(Movie movie, int pageIndex) throws Exception {
    return this.movieDao.getPagingList(movie, pageIndex);
}
    중략...
}
    
```

## 1.2.Spring Configuration XML - jaxws:endpoint tag 사용

작성된 서비스를 Web Service로 노출시키는 서버를 구동하기 위해서 2가지 방식이 지원된다. 이중 Spring Configuration XML - <jaxws:endpoint/> tag를 사용하여 서버를 구동시켜보도록 한다. (Apache CXF에서는 <jaxws:server/> tag를 사용하여 서버를 구동시키는 방법도 제공하고 있으나 Anyframe에서는 <jaxws:endpoint/> tag 사용을 채택하고 있으므로 <jaxws:server/> tag에 대한 내용은 매뉴얼에서 언급하고 있지 않다.)

<jaxws:endpoint/> tag의 각 속성값에 대한 설명은 다음 표와 같다. 아래 표에 나와있지 않은 속성들도 여러 가지가 존재한다. 단, 여기서는 필수적으로 작성해야 하는 속성값에 대한 설명을 작성해놓은 것으로 나머지 속성값들에 대한 설명은 JAX-WS Configuration [<http://cxf.apache.org/docs/jaxws-configuration.html>]을 참고하도록 한다.

Property Name	Description	Required	Default Value
id	spring bean id를 작성한다.	Y	N/A
implementor	구현 클래스를 작성한다. 클래스명 대신에 spring bean id로 대체하고자 하면 bean id 앞에 #을 붙여서 작성하면 된다.	Y	N/A
address	서비스가 동작할 주소를 상대 경로로 작성한다.	Y	N/A

### 1.2.1.Samples

다음은 Spring Configuration XML - <jaxws:endpoint/> tag를 사용하여 Movie Service를 Web Service로 노출시키는 서버를 구동하는 예제이다.

- 다음은 비즈니스 레이어의 서비스를 Web Service로 노출시키는 <jaxws:endpoint/> tag 속성을 정의한 context-cxfjaxws.xml 의 일부이다.

Configuration

```

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns: xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:cxf="http://cxf.apache.org/core"
    xmlns:jaxws="http://cxf.apache.org/jaxws"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://cxf.apache.org/core http://cxf.apache.org/schemas/core.xsd
        http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd">
    
```

```

<!-- Load CXF modules from cxf.jar -->
<import resource="classpath:META-INF/cxf/cxf.xml" />
<import resource="classpath:META-INF/cxf/cxf-extension-soap.xml" />
<import resource="classpath:META-INF/cxf/cxf-servlet.xml" />

<!-- Enable message logging using the CXF logging feature -->
<cxf:bus>
  <cxf:features>
    <cxf:logging />
  </cxf:features>
</cxf:bus>

<!-- JAX-WS Frontend to expose movieService using endpoint tag -->
<jaxws:endpoint id="cxfJaxwsServerMovieService" implementor="#cxfJaxwsMovieService"
address="/ws"/>

```

Jetty 혹은 Tomcat 서버 등의 WAS를 이용하여 웹 어플리케이션을 구동하고 비즈니스 레이어의 서비스를 Web Service로 노출시키게 되는데, 이때 web.xml 파일에 Spring 속성 정의 XML 파일을 org.springframework.web.context.ContextLoaderListener를 이용하여 등록시켜 줘야 한다. 다음은 CXFServlet과 ContextLoaderListener를 정의한 web.xml의 일부이다.

```

<web-app
  종략...
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
      classpath:/spring/context-*.xml
    </param-value>
  </context-param>

  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-
class>
  </listener>
  종략...
  <!-- cxf-configuration-START -->
  <servlet>
    <servlet-name>CXF-JAXWS-Servlet</servlet-name>
    <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>CXF-JAXWS-Servlet</servlet-name>
    <url-pattern>/cxf-jaxws/*</url-pattern>
  </servlet-mapping>
  <mime-mapping>
    <extension>wsdl</extension>
    <mime-type>text/xml</mime-type>
  </mime-mapping>
  <mime-mapping>
    <extension>xsd</extension>
    <mime-type>text/xml</mime-type>
  </mime-mapping>
  종략...
</web-app>

```

## 1.3.Server: JAX-WS Frontend API 사용

작성된 서비스를 Web Service로 노출시키는 서버를 구동하기 위해서 2가지 방식이 지원된다. 이중 JAX-WS Frontend API를 이용하여 직접 자바코드를 작성하여 서버를 구동시켜보도록 한다. Apache CXF에서 제공하는 JAX-WS Frontend API를 직접 호출하여 작성하는 것이 가능하며, JaxWsServerFactoryBean 클래스를 Spring Configuration 파일의 Bean으로 등록하여 설정하는 것도 가능하다.

### 1.3.1.Samples

다음은 Apache CXF에서 제공하는 JaxWsServerFactoryBean을 직접 이용하여 MovieFinder Service를 Web Service로 노출시키는 서버를 구동하는 예제이다.

- Apache CXF JaxWsServerFactoryBean을 사용한 서버 구동

다음은 비즈니스 레이어의 서비스를 Web Service로 노출시키는 서버를 구동하는 코드 작성 예이다.

```
MovieServiceImpl implementor = new MovieServiceImpl();
JaxWsServerFactoryBean svrFactory=new JaxWsServerFactoryBean();
svrFactory.setServiceClass(MovieService.class);
svrFactory.setAddress("http://localhost:8080/myproject/cxf-jaxws/ws");
svrFactory.setServiceBean(implementor);
svrFactory.create();
```

## 1.4.Spring Configuration XML - jaxws:client tag 사용

Web Services에 접근하기 위한 클라이언트 작성 방식에는 2가지 방식이 지원된다. 이중 Spring Configuration XML - <jaxws:client/> tag를 사용하여 클라이언트를 작성하여 Web Services에 접근해 보도록 한다. <jaxws:client/> tag의 각각의 Attribute 속성값에 대한 설명은 다음 표와 같다. 아래 표에 나와있지 않은 속성들도 여러 가지가 존재한다. 단, 여기서는 필수적으로 작성해야 하는 Attribute 속성값에 대한 설명을 작성해놓은 것으로 나머지 속성 값들에 대한 설명은 JAX-WS Configuration [http://cxf.apache.org/docs/jax-ws-configuration.html]을 참고하도록 한다.

Property Name	Description	Required	Default Value
id	spring bean id를 작성한다.	Y	N/A
serviceBean	서비스의 인터페이스 클래스를 작성한다.	Y	N/A
address	서비스 접근 URL Address를 절대 경로로 작성한다.	Y	N/A

### 1.4.1.Samples

다음은 Spring Configuration XML - <jaxws:client/> tag를 사용하여 Movie Service에 접근하는 예제이다.

- Configuration

다음은 Web Services로 노출된 Movie Service에 접근하는 <jaxws:client/> tag 속성을 정의한 cxf-jaxws-servlet.xml의 일부이다.

```
<beans xmlns="http://www.springframework.org/schema/beans"
```

```

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:cxf="http://cxf.apache.org/core"
xmlns:jaxws="http://cxf.apache.org/jaxws"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://cxf.apache.org/core http://cxf.apache.org/schemas/core.xsd
http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd">
<jaxws:client id="cxfJaxwsClientMovieService"
serviceClass="myproject.cxf.jaxws.moviefinder.service.MovieService"
address="http://localhost:8080/myproject/cxf-jaxws/ws" />

```

- Controller

다음은 앞서 정의한 속성 설정 파일들을 기반으로 하여 Web Services로 노출된 Movie Service에 접근하는 Controller 클래스인 MovieController.java의 일부이다.

```

@Controller("cxfJaxwsMovieController")
@RequestMapping("/cxfJaxwsMovie.do")
@SessionAttributes(types = Movie.class)
public class MovieController {

    @Inject
    @Named("cxfJaxwsClientMovieService")
    private MovieService movieService;

    @RequestMapping(params = "method=get")
    public String get(@RequestParam("movieId") String movieId, Model model)
        throws Exception {
        Map<String, Movie> resultMap = this.movieService.get(movieId);
        model.addAttribute(resultMap.get("movie"));

        return "cxf-jaxws/moviefinder/movie/form";
    }
    중략...

```

## 1.5.Client: JAX-WS Frontend API 사용

Web Service에 접근하기 위한 클라이언트 작성 방식에는 2가지 방식이 지원된다. 이중 JAX-WS Frontend API를 직접 이용한 클라이언트를 작성하여 Web Service에 접근해보도록 한다. Apache CXF에서 제공하는 JAX-WS Frontend API를 직접 호출하여 작성하는 것이 가능하며, JaxWsProxyFactoryBean 클래스를 Spring Configuration 파일의 Bean으로 등록하여 설정하는 것도 가능하다.

### 1.5.1.Samples

다음은 Apache CXF에서 제공하는 JaxWsProxyFactoryBean를 직접 이용하여 Web Services로 노출된 Movie Service에 접근하는 예제이다.

- Apache CXF JaxWsProxyFactoryBean 사용한 클라이언트

다음은 Apache CXF에서 제공하는 JaxWsProxyFactoryBean 클래스를 사용하여 Web Services로 노출된 Movie Service에 접근하는 코드 작성 예이다.

```

JaxWsProxyFactoryBean factory = new JaxWsProxyFactoryBean();
factory.setServiceClass(MovieService.class);
factory.setAddress("http://localhost:8080/myproject/cxf-jaxws/ws");

MovieService movieService = (MovieService) factory.create();
Map<String, Movie> resultMap = movieService.get("MV-00003");

```

## 1.6.Annotation 작성

@WebService()만 필수 사항이고 대부분 annotation의 경우 디폴트 값이 제공되므로 작성이 불필요 하다.

### 1.6.1.@WebService (javax.jws.WebService)

Annotation을 포함하고 있는 자바 클래스가 Web Service Endpoint 인터페이스 클래스 혹은 구현 클래스임을 명시할 때 사용하며 SEI와 구현 클래스에 설정한다.(필수 사항) 모든 Property의 값을 작성할 필요는 없으나 되도록 많은 정보를 작성하는 것을 권장한다. SEI에 더 많은 Annotation 정보를 작성할 수록 잘 정의된 WSDL을 생성시킬 수 있기 때문이다. Property에 대한 세부 내용은 다음 표와 같다.

Property Name	Type	필수	WSDL Mapping	Default	설명
name	String	X	wsdl:portType	클래스명	웹 서비스 명으로 wsdl:portType으로 매핑됨, 기본값은 인터페이스 클래스 혹은 구현 클래스의 패키지를 제외한 명
targetNamespace	String	X	N/A	패키지 명	웹 서비스로부터 생성된 XML element와 WSDL에 사용될 XML namespace
serviceName	String	X	wsdl:service	클래스명+"Service"	SEI에 설정 불가, 서비스 명으로 wsdl:service로 매핑됨, 기본값은 인터페이스 클래스 혹은 구현 클래스의 패키지명을 제외한 명+"Service"
wsdlLocation	String	X	N/A	서비스 디폴로이 URI	미리 정의된 WSDL의 위치로 WSDL에 정의된 portType, binding은 서비스 구현 클래스에 작성된 값들과 일치해야 함

Property Name	Type	필수	WSDL Mapping	Default	설명
endpointInterface	String	X	N/A	SEI의 패키지 이름을 포함한 클래스명, 구현 클래스명 + "Port"	SEI에 설정 불가, 구현 클래스의 SEI명
portName	String	X	wsdl:port	구현 클래스명 + "Port"	SEI에 설정 불가, wsdl:port로 매핑됨, 기본값은 구현클래스명 + "Port"

- @WebService annotation 설정이 적용된 클래스 내의 메소드 규칙
  - 구현 클래스의 @WebService annotation이 SEI를 참조하는 경우 구현 클래스에 @WebMethod annotation은 없어야 한다.
  - SEI의 모든 공용 메소드는 @WebMethod annotation 설정 여부와 관계 없이 Web Service로 노출된 메소드로 간주된다.
  - SEI를 참조하지 않는 구현 클래스의 경우 @WebMethod annotation이 exclude=true 값으로 지정되면 해당 메소드가 Web Service로 노출되지 않는다.

## 1.6.2. @WebParam (javax.jws.WebParam)

Web Service의 자바 메소드의 입력 파라미터와 WSDL 파일에서 파라미터를 표현하는 XML element 간의 매핑을 설정하며 Property에 대한 세부 내용은 다음 표와 같다.

Property	Type	필수	설명
name	String	X	파라미터명으로 RPC style의 Web Service의 경우 파라미터를 나타내는 wsdl:part으로 매핑되고, Document style의 Web Service의 경우 파라미터를 나타내는 XML element의 로컬 네임이 됨. 기본값은 메소드의 파라미터명
targetNamespace	String	X	파라미터의 XML namespace로 파라미터가 XML element로 대응되는 Document style의 Web Service에서만 사용됨. 기본값은 Web Service의 targetNamespace
mode	Enum	X	파라미터가 전달되는 방향으로 IN, OUT 혹은 INOUT 중 하나 선택. 기본값은 IN
header	Boolean	X	true로 설정되면 파라미터를 SOAP 메시지 BODY가 아닌 SOAP 메시지 헤더로부터 가져옴. 기본값은 false
partName	String	X	RPC 혹은 Document style의 매개변수 양식이 BARE인 경우에만 사용됨

## 1.6.3. @WebMethod (javax.jws.WebMethod)

Web Service 메소드로 공개되는 메소드를 설정할 때 사용하며 @WebMethod annotation은 @WebService annotation 설정이 된 클래스에서만 지원된다. Property에 대한 세부 내용은 다음 표와 같다.

Property	Type	필수	설명
operationName	String	X	공개되는 메소드 명으로 wsdl:operation으로 매핑됨. 기본값은 자바 메소드 명

Property	Type	필수	설명
action	String	X	메소드에 적용되는 action 속성으로 SOAPBinding의 경우 SOAP 메시지에서 SOAPAction 헤더의 값을 결정
exclude	Boolean	X	Web Service에서 메소드를 제외시킬지 여부 지정함. 기본값은 false

## 1.6.4.@OneWay (javax.jws.OneWay)

입력 값은 있으나 리턴 값이 없는 단방향 메소드를 정의할 때 사용한다. @WebMethod annotation과 함께 사용되며 별도의 속성 정의가 필요없다.

## 1.6.5.@WebResult (javax.jws.WebResult)

Web Service 메소드로 공개되는 메소드의 리턴 값과 WSDL의 리턴 값을 표현하는 XML 요소 간의 매핑을 설정할 때 사용한다. Property에 대한 세부 내용은 다음 표와 같다.

Property	Type	필수	설명
name	String	X	WSDL에서 리턴 값을 나타내는 element 명, RPC style의 Web Service에서는 리턴 값을 나타내는 wsdl:part 에 매핑되며, Document style의 Web Service에서는 리턴 값을 나타내는 XML element의 로컬
targetNamespace	String	X	리턴값의 XML namespace로 RPC style의 Web Service 혹은 Document style의 매개변수 양식이 BARE인 Web Service에서만 사용됨. 기본값은 Web Service의 targetNamespace값
header	String	X	결과를 헤더에 저장할 것인지 여부 지정. 기본값은 false
partName	String	X	RPC 혹은 Document style의 매개변수 양식이 BARE인 경우, 결과의 part name을 지정함. 기본값은 @WebResult.name.

## 1.6.6.Samples

다음은 Movie Service의 인터페이스 클래스에 여러 가지 Annotation을 설정해 본 예제이다.

- Interface Class

다음은 Movie Service의 인터페이스 클래스를 작성한 MovieService.java의 일부이다. 인터페이스 클래스 상단에 @WebService Annotation 작성은 필수 사항이다. 이외에 @WebMethod, @Oneway, @WebParam Annotation을 설정해보고 어떻게 동작하는지 살펴해보도록 한다.

```
@WebService
public interface MovieService {
    // ===== method for Annotation tests
    @WebMethod(exclude = true)
    public void testAnnotationMethodExclude();

    @WebMethod(operationName = "testAnnotationMethodInclude")
    public void testAnnotationMethod();

    @Oneway
    public String testAnnotationOneway();

    public String testAnnotationWebParam(
        @WebParam(name = "movieAnnotationWebParam") String input);
    중략...
```

- Implementation Class

다음은 Movie Service의 인터페이스 클래스를 작성한 MovieServiceImpl.java 의 일부이다.

```
@Service("cxfJaxwsMovieService")
public class MovieServiceImpl implements MovieService {

    @Inject
    @Named("cxfJaxwsMovieDao")
    private MovieDao movieDao;

    // ===== method for Annotation tests
    public void testAnnotationMethodExclude() {
        MovieService.LOGGER.debug("testAnnotationMethodExclude method is called.");
    }

    public void testAnnotationMethod() {
        MovieService.LOGGER.info("testAnnotationMethod method is called.");
    }

    public String testAnnotationOneway() {
        MovieService.LOGGER.debug("testAnnotationOneway method is called.");
        return "testAnnotationOneway";
    }

    public String testAnnotationWebParam(String input) {
        MovieService.LOGGER.debug("testAnnotationWebParam method is called with the input
parameter="
+ input);
        return "testAnnotationWebParam";
    }
}
종략...
```

- Test case

다음은 Apache CXF에서 제공하는 JaxWsProxyFactoryBean 클래스를 사용하여 Web Services로 노출된 Movie Service에 접근하는 테스트케이스 JaxWsFrontendServerFactoryAnnotationTest.java의 일부이다. WebServices Annotation 설정 값에 따라 어떻게 동작하는지 확인해보도록 한다.

```
@RunWith(JUnit4.class)
public class JaxWsFrontendServerFactoryAnnotationTest extends ServerRunner {
    /**
     * [Flow #-1] Positive Case : @WebMethod annotation을 이용하여
     *                             특정 method를 Web Service method로 노출되지 않도록 한다.
     *                             @WebMethod의 속성 값 중 exclude 값을 true로 설정한다.
     *                             (ex. @WebMethod(exclude=true) )
     */
    @Test
    public void testAnnotationMethodExclude() {
        Client client = new JaxwsClient();
        MovieService movieService =
            (MovieService) client.getClient(new ClientInfo(MovieService.class,
                "http://localhost:9002/Movie", false));

        try {
            movieService.testAnnotationMethodExclude();
            Assert.fail();
        } catch (Exception e) {
            // Exception should be occurred.
            if (!(e instanceof WebserviceException))
                Assert.fail();
        }
    }
}
```

```

    }
}

/**
 * [Flow #-2] Positive Case : @webMethod annotation을 이용하여
 *                               특정 method name을 다른 name으로 변경하여 호출 가능하도록
 *                               한다.
 *                               @webMethod의 속성 값 중 operationName 값을 다른 name으로
 *                               설정한다.
 *                               (ex.
 * @webMethod(operationName="testAnnotationMethodInclude") )
 *                               실제 SEI에서 제공하는 method는 testAnnotationMethod 뿐이
 *                               고,
 *                               @webMethod 속성 값 중 operationName값을
 * testAnnotationMethodInclude로 설정해놓았으므로,
 *                               Client에서 Web Service method를 호출할 때에는
 * testAnnotationMethodInclude method를 갖는
 *                               Service Interface class를 제정의에서 호출하여 사용하도록
 *                               한다.
 */
@Test
public void testAnnotationMethodInclude() {
    Client client = new JaxWSClient();
    org.anyframe.sample.cxf.jaxws.moviefinder.service.MovieServiceWebMethod
    movieService =
        (org.anyframe.sample.cxf.jaxws.moviefinder.service.MovieServiceWebMethod)
    client.getClient(
        new
    ClientInfo(org.anyframe.sample.cxf.jaxws.moviefinder.service.MovieServiceWebMethod.class,
        "http://localhost:9002/Movie", false));

    try {
        movieService.testAnnotationMethodInclude();
    } catch (Exception e) {
        // It should not be failed.
        Assert.fail();
    }
}
}
중략...

```

## 1.7.Resources

- 참고자료
  - JAX-WS 2.0 Specification [<http://jcp.org/en/jsr/detail?id=224>]
  - Building Web Services with JAX-WS [<http://download.oracle.com/javase/5/tutorial/doc/bnayl.html>]
  - JAX-WS Configuration [<http://cxf.apache.org/docs/jax-ws-configuration.html>]

---

## 2.Asynchronous Invocation

보통의 synchronous한 호출 방식과 함께, Apache CXF는 JAX-WS Spec.에 정의된 2가지(Polling approach, Callback approach) 형태의 비동기(asynchronous) 호출 방식을 지원한다. 즉, 클라이언트 사이드에서 서버 사이드의 Web Services를 호출 시 비동기적으로 호출하여 사용할 수 있게 하는 기능이다.

**Polling approach**의 특징 을 살펴보면 다음과 같다.

- 서버) 서비스 인터페이스에 메소드(메소드명: 대상 메소드 명 + "Async")를 추가 작성하는데, 이때 리턴타입이 Response인 메소드를 작성한다.
- 클라이언트) 원격에 존재하는 메소드를 호출하기 위해, output 파라미터 없이 javax.xml.ws.Response 객체를 리턴하는 특정 메소드를 호출한다.
- javax.util.concurrent.Future 인터페이스를 상속받은 Response 객체는 응답 메시지가 도착했는지 여부를 확인하는 표를 받는다.

**Callback approach**의 특징 을 살펴보면 다음과 같다.

- 서버) 서비스 인터페이스에 메소드(메소드명: 대상 메소드 명 + "Async")를 추가 작성하는데, 이때 리턴 타입이 Future<?>이며 AsyncHandler 파라미터를 추가로 갖는 메소드를 작성한다.
- 클라이언트) AsyncHandler 클래스 구현이 필요하다.
- 클라이언트) 원격에 존재하는 메소드를 호출하기 위해, 파라미터 중 하나가 javax.xml.ws.AsyncHandler 타입인 callback 객체에 참조 관계가 있는 위에서 작성한 메소드를 호출한다.
- 응답 메시지가 클라이언트에 도착하자마자, Apache CXF 런타임 환경은 응답 메시지의 콘텐츠에 응답 메시지를 전달해주기 위해 AsyncHandler 객체를 재호출한다.

다음은 Asynchronous Method Invocation 기능을 사용하기 위해서 Server와 Client 단에서 어떤 작업을 수행해야 하는지에 대한 내용이다. 2가지(Polling approach와 Callback approach) 형태의 asynchronous 한 호출 방식에 대해서 하나의 예제를 가지고 살펴보도록 한다.

### 2.1.Server Configuration

Apache CXF에서 제공하는 Tool을 이용하여 비동기적으로 Web Services의 메소드를 호출할 수 있도록 지원하고 있다. 아래 예제를 통해서 Tool 사용 모습을 포함하여 어떻게 비동기적인 호출을 가능하게 하는지 살펴보도록 한다.

Tool을 아직 준비하지 못했다면 Apache CXF를 다운로드 페이지 [<http://cxf.apache.org/download.html>]에서 내려 받아서 압축을 풀고 루트 폴더 하위의 bin 폴더 내에 존재하는 Tool을 사용하도록 한다.

#### 2.1.1.Samples

Web Service로 노출시킬 Movie Service의 인터페이스 클래스를 대상으로 java2ws Tool을 사용하여 WSDL 파일을 생성해낸다. WSDL 파일을 생성한 이후, wsdl2java Tool을 통해 WSDL 파일과 asynch\_binding.xml 파일을 이용하여 asynchronous한 호출을 가능하게 하는 Java 소스 코드들(org.anyframe.sample.cxf.jaxws.moviefinder.asynch.service.asynch\_soap\_http package 하위의 클래스들(SEI, WebServiceClient, complex type classes))을 생성한다. 이때 이미 WSDL 파일을 가지고 있다면 java2ws Tool을 사용할 필요는 없다.

- Temporary Interface Class

다음은 Movie Service의 인터페이스 클래스를 작성한 MovieServiceAsynch.java의 일부이다. JAX-WS Frontend 모델을 이용하여 Web Service를 구현하므로, 인터페이스 클래스 상단에 @WebService Annotation을 작성해줘야 한다. 인터페이스 메소드로는 findMovieListAll() 메소드를 정의하고, 클라이언트

언트에서 asynchronous하게 이 메소드를 호출하고자 한다. 실제로는 이 인터페이스를 사용하지 않는다. WSDL 파일을 생성시키고자 하는 목적에서 작성한 것이다. 실제로는 WSDL 파일로부터 생성된 인터페이스 클래스를 사용하게 됨에 유의하도록 한다.

```
@WebService(targetNamespace = "http://
service.asynch.moviefinder.jaxws.cxf.sample.anyframe.org/asynch_soap_http")
public interface MovieServiceAsynch {

    public List<Movie> findMovieListAll() throws Exception;

}
...중략
```

- WSDL 파일 생성

인터페이스 클래스를 기준으로 WSDL 파일을 생성시키도록 한다. 이때 Apache CXF에서 제공하는 Tool 중 java2ws를 이용하여 생성시킨다. Command Prompt 상에서 아래와 같은 명령어를 수행시키도록 한다. 수행 시키는 위치는 프로젝트가 존재하는 루트 폴더이다.

```
command>java2ws -wsdl -cp target/classes
org.anyframe.sample.cxf.jaxws.moviefinder.asynch.service.MovieServiceAsynch
```

다음은 생성된 WSDL 파일(MovieServiceAsynchService.wsdl)의 일부이다. 생성된 후 soap:address location 정보 등 수정하고 싶은 정보가 있는 경우 변경하도록 한다.

```
<wsdl:definitions name="MovieServiceAsynchService"
targetNamespace="http://service.asynch.moviefinder.jaxws.cxf.sample.anyframe.org/
asynch_soap_http"
중략...
<wsdl:binding name="MovieServiceAsynchServiceSoapBinding"
type="tns:MovieServiceAsynch">
<soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
<wsdl:operation name="findMovieListAll">
<soap:operation soapAction="" style="document"/>
<wsdl:input name="findMovieListAll">
<soap:body use="literal"/>
</wsdl:input>
<wsdl:output name="findMovieListAllResponse">
<soap:body use="literal"/>
</wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="MovieServiceAsynchService">
<wsdl:port name="MovieServiceAsynchPort"
binding="tns:MovieServiceAsynchServiceSoapBinding">
<soap:address location="http://localhost:9002/Movie"/>
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
...중략
```

- Configuration

Asynchronous 기능을 제공하는 Java 소스 코드를 생성하기 위해서는 WSDL 파일 뿐아니라 asynch\_binding.xml 파일도 함께 필요하다. 다음은 asynch\_binding.xml 의 일부이다. 여기서 wsdlLocation 부분에 유의하여 작성하도록 한다.

```
<bindings
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
```

```

wsdlLocation="MovieServiceAsynchService.wsdl"
xmlns="http://java.sun.com/xml/ns/jaxws">
<bindings node="wsdl:definitions">
  <enableAsyncMapping>true</enableAsyncMapping>
</bindings>
</bindings>
중략...

```

- Java 코드 생성

지금까지 준비된 WSDL 파일과 asynch\_binding.xml 파일을 이용하여 Asynchronous 기능을 제공하는 Java 소스 코드를 생성하도록 한다. 이때 Apache CXF에서 제공하는 Tool 중 wsdl2java을 이용하여 생성시킨다. Command Prompt 상에서 아래와 같은 명령어를 수행시키도록 한다. 수행 시키는 위치는 프로젝트가 존재하는 루트 폴더이다.

```

command>wsdl2java -d src -b src/test/resources/jaxws/asynch/wsdl/asynch_binding.xml
src/test/resources/jaxws/asynch/wsdl/MovieServiceAsynchService.wsdl

```

Tool을 통해 생성되는 Java 코드들은 모두 org.anyframe.sample.cxf.jaxws.moviefinder.asynch.service.asynch\_soap\_http package 하위에 위치한다. MovieService의 인터페이스 클래스인 MovieServiceAsynch.java, MovieService 접근을 위한 WebService Client 클래스인 MovieServiceAsynchService.java, JavaBeans 클래스인 Movie.java 클래스 등 여러 가지의 Java 소스 코드가 생성된다. 다음은 생성된 인터페이스 클래스인 MovieServiceAsynch.java의 일부이다. 이 인터페이스 클래스가 실제로 사용된다. Polling approach 방식 및 Callback approach에서 사용되는 2개의 findMovieListAllAsync() 메소드가 추가로 생성되어 있음을 확인할 수 있다.

```

@WebService(targetNamespace = "http://
service.asynch.moviefinder.jaxws.cxf.sample.anyframe.org/asynch_soap_http",
  name = "MovieServiceAsynchService")
@XmlSeeAlso( {ObjectFactory.class } )
public interface MovieServiceAsynch {

  @ResponseWrapper(localName = "findMovieListAllResponse",
    targetNamespace = "http://service.asynch.moviefinder.jaxws.cxf.sample.anyframe.org/
asynch_soap_http",
    className =
      "org.anyframe.sample.cxf.jaxws.moviefinder.asynch.service.asynch_soap_http.FindMovieListAllResponse")
  @RequestWrapper(localName = "findMovieListAll",
    targetNamespace = "http://service.asynch.moviefinder.jaxws.cxf.sample.anyframe.org/
asynch_soap_http",
    className =
      "org.anyframe.sample.cxf.jaxws.moviefinder.asynch.service.asynch_soap_http.FindMovieListAll")
  @WebResult(name = "return", targetNamespace = "")
  @WebMethod
  public
  java.util.List<org.anyframe.sample.cxf.jaxws.moviefinder.asynch.service.asynch_soap_http.Movie>
  findMovieListAll() throws Exception;

  @ResponseWrapper(localName = "findMovieListAllResponse",
    targetNamespace = "http://service.asynch.moviefinder.jaxws.cxf.sample.anyframe.org/
asynch_soap_http",
    className =
      "org.anyframe.sample.cxf.jaxws.moviefinder.asynch.service.asynch_soap_http.FindMovieListAllResponse")
  @RequestWrapper(localName = "findMovieListAll",
    targetNamespace = "http://service.asynch.moviefinder.jaxws.cxf.sample.anyframe.org/
asynch_soap_http",
    className =
      "org.anyframe.sample.cxf.jaxws.moviefinder.asynch.service.asynch_soap_http.FindMovieListAll")
  @WebMethod(operationName = "findMovieListAll")

```

```

public
Response<org.anyframe.sample.cxf.jaxws.moviefinder.asynch.service.asynch_soap_http.FindMovieListAllResponse>
    findMovieListAllAsync();

    @ResponseWrapper(localName = "findMovieListAllResponse",
        targetNamespace = "http://service.asynch.moviefinder.jaxws.cxf.sample.anyframe.org/
asynch_soap_http",
        className =
"org.anyframe.sample.cxf.jaxws.moviefinder.asynch.service.asynch_soap_http.FindMovieListAllResponse")
    @RequestWrapper(localName = "findMovieListAll",
        targetNamespace = "http://service.asynch.moviefinder.jaxws.cxf.sample.anyframe.org/
asynch_soap_http",
        className =
"org.anyframe.sample.cxf.jaxws.moviefinder.asynch.service.asynch_soap_http.FindMovieListAll")
    @WebMethod(operationName = "findMovieListAll")
    public Future<?> findMovieListAllAsync(
        @WebParam(name = "asynchHandler", targetNamespace = "")

AsynchHandler<org.anyframe.sample.cxf.jaxws.moviefinder.asynch.service.asynch_soap_http.FindMovieListAllResponse>
    asynchHandler);
중략...

```

- Implementation Class

Interface Class를 구현한 구현 클래스로 2개의 findMovieListAllAsync() 메소드에 대해서 빈 내용으로 구현 메소드만 작성해놓는다. 실제로 호출되지는 않는다. 다음은 Movie Service의 인터페이스 클래스인 MovieServiceAsynch를 구현한 MovieServiceImpl.java 의 일부이다.

```

@WebService(serviceName = "MovieServiceAsynchService", portName =
"MovieServiceAsynchPort", 중략...
public class MovieServiceImpl implements MovieServiceAsynch {
    public List<Movie> findMovieListAll() throws Exception {
        return this.movieDao.findMovieListAll();
    }

    public Response<FindMovieListAllResponse> findMovieListAllAsync() {
        return null;
        /* not called */
    }

    public Future<?> findMovieListAllAsync(
        AsynchHandler<FindMovieListAllResponse> asynchHandler) {
        return null;
        /* not called */
    }
}
중략...

```

- Apache CXF JaxWsServerFactoryBean 사용한 서버 구동

다음은 서버 사이드의 서비스를 Web Services로 노출시키는 서버를 구동하는 코드 작성 예이다. 인터페이스 클래스, 구현 클래스의 인스턴스, Web Services 주소를 JaxWsServerFactoryBean 속성 정보로 설정해준다.

```

MovieServiceImpl implementor = new MovieServiceImpl();
JaxWsServerFactoryBean svrFactory=new JaxWsServerFactoryBean();
svrFactory.setServiceClass(MovieServiceAsynch.class);
svrFactory.setAddress("http://localhost:9002/Movie");
svrFactory.setServiceBean(implementor);
svrFactory.create();

```

## 2.2.Client Configuration

클라이언트에서 서버 사이트의 Web Services를 호출 시 비동기적으로 호출하여 사용해 보도록 한다. 이때, 위에서 Tool을 통해 생성된 Java 코드 중 MovieServiceAsynchService를 이용하여 Web Service에 접근한다.

### 2.2.1.Samples

다음은 Web Services로 노출된 Movie Service의 메소드를 비동기적으로 호출하여 사용하는 예제이다.

- Callback approach 사용 시 AsyncHandler 클래스 구현

Polling approach 방식을 사용하면 부가 작성해야 하는 클래스가 없지만, Callback approach 방식을 사용하려면 AsyncHandler 클래스를 추가 구현해줘야 한다. javax.xml.ws.AsyncHandler 인터페이스를 implement 하며, handleResponse method를 구현하여야 한다. 다음은 AsyncHandler 클래스를 구현한 MovieAsyncHandler.java의 일부이다.

```
public class MovieAsyncHandler implements AsyncHandler<FindMovieListAllResponse> {

    private FindMovieListAllResponse reply;

    public void handleResponse(Response<FindMovieListAllResponse> response) {
        try {
            LOGGER.debug("The handleResponse method of MovieAsyncHandler is called.");
            reply = response.get();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    public List<Movie> getResponse() {
        return reply.getReturn();
    }

    ...종락
```

- Test case

다음은 앞서 작성한 코드와 Tool을 통해 생성된 Java 코드들을 이용하여 Web Services로 노출된 Movie Service에 접근하는 JaxWsAsynchTest.java의 일부이다. Tool을 통해 생성된 WebService Client 코드인 MovieServiceAsynchService를 통해, Web Services로 노출된 Movie Service의 인터페이스 메소드를 asynchronous하게 호출한다. 이때 WSDL 파일과 서비스명을 이용하여 접근하도록 한다. Movie Service를 얻은 후에는 Movie Service에서 findMovieListAllAsync() 메소드를 비동기 방식으로 호출하여 동작이 올바른지 테스트해본다.

```
@RunWith(JUnit4.class)
public class JaxWsAsynchTest extends ServerRunner {

    private static final QName SERVICE_NAME =
        new QName("http://service.asynch.moviefinder.jaxws.cxf.sample.anyframe.org/
asynch_soap_http",
            "MovieServiceAsynchService");

    /**
     * [Flow #-2] Positive Case : Asynchronous method invocation 방식 중 Polling approach
     방법으
     *
     * List 형태의 Movie 전체 목록을 조회한다.
     * @throws Exception
     *
     * throws exception which is from service
```

```

*/
@Test
public void testFindMovieListAllPolling() throws Exception {

    File wsdl = new File("src/test/resources/jaxws/asynch/wsdl/
MovieServiceAsynchService.wsdl");

    MovieServiceAsynchService client =
        new MovieServiceAsynchService(wsdl.toURL(), SERVICE_NAME);
    MovieServiceAsynch movieService = client.getMovieServiceAsynchPort();

    // 1. find movie list all
    Response<FindMovieListAllResponse> response =
        movieService.findMovieListAllAsync();

    // 2. wait for response after asynchronous method invocation
    while (!response.isDone()) {
        Thread.sleep(100);
    }

    // 3. check the movie list count
    FindMovieListAllResponse reply = response.get();
    Assert.assertEquals(2, reply.getReturn().size());
}

/**
 * [Flow #-3] Positive Case : Asynchronous method invocation 방식 중 Callback approach
방법으로
 *
 * List 형태의 Movie 전체 목록을 조회한다.
 * @throws Exception
 *     throws exception which is from service
 */
@Test
public void testFindMovieListAllCallback() throws Exception {

    File wsdl = new File("src/test/resources/jaxws/asynch/wsdl/
MovieServiceAsynchService.wsdl");

    MovieServiceAsynchService client =
        new MovieServiceAsynchService(wsdl.toURL(), SERVICE_NAME);
    MovieServiceAsynch movieService = client.getMovieServiceAsynchPort();

    // 1. find movie list all
    MovieAsyncHandler asynchHandler = new MovieAsyncHandler();
    Future<?> response = movieService.findMovieListAllAsync(asynchHandler);

    // 2. wait for response after asynchronous method invocation
    while (!response.isDone()) {
        Thread.sleep(100);
    }

    // 3. check the movie list count
    List<Movie> reply = asynchHandler.getResponse();
    Assert.assertEquals(2, reply.size());
}
중략...

```

## 2.3.Resources

- 참고자료

- Developing a Consumer - Asynchronous Invocation Model [<http://cxf.apache.org/docs/developing-a-consumer.html>]
- Asynchronous Web Service Invocation with JAX-WS 2.0 [<http://today.java.net/pub/a/today/2006/09/19/asynchronous-jax-ws-web-services.html>]

---

# 3.WAS(Web Application Server) Configuration

Apache CXF 기반의 Web Services를 구현한 웹 어플리케이션을 WAS(Web Application Server)에 배포하여 구동시키게 되는데, 이때 각 WAS 별로 Apache CXF를 실행시키기 위해서 추가 작업이 필요할 수 있다. 본 장에서는 WAS 별 추가 환경 설정이 필요한 경우 어떻게 해야 하는지 설명하고 있다. Apache CXF는 JDK 1.5 이상을 지원하므로 설치 대상 WAS도 JDK 1.5 이상을 지원하는 WAS로 제한한다.

다음은 각 WAS 별 내용이다.

## 3.1.Tomcat

Apache CXF는 JDK 1.5 이상을 지원하므로 Tomcat 서버의 경우, Tomcat 5.5.x 버전 이상 서버가 대상이 된다. Tomcat 서버에 대한 설명 및 다운로드는 이곳 [<http://tomcat.apache.org/>]을 참고하도록 한다.

단, Plugin 설치로 생성된 샘플 어플리케이션일 경우, 오픈소스 활용을 위한 WAS별 조치 사항을 확인하기 위해 먼저 설치된 각 Plugin 매뉴얼 내의 WAS(Web Application Server)별 유의사항 을 참고하도록 한다.

### 3.1.1.5.5.23, 6.0.x, 7.0.x

추가 설정 없이 Apache CXF 사용이 가능하다.

## 3.2.JEUS

Apache CXF는 JDK 1.5 이상을 지원하므로 JEUS 서버의 경우, JEUS 5와 JEUS 6 버전의 서버가 대상이 되나, JEUS 5의 경우 JAXB 라이브러리의 충돌로 Apache CXF 사용이 불가능하다. JEUS 서버에 대한 설명 및 다운로드는 TmaxSoft 홈페이지를 참고하도록 한다.

단, Plugin 설치로 생성된 샘플 어플리케이션일 경우, 오픈소스 활용을 위한 WAS별 조치 사항을 확인하기 위해 먼저 설치된 각 Plugin 매뉴얼 내의 WAS(Web Application Server)별 유의사항 을 참고하도록 한다.

### 3.2.1.5.0

Apache CXF 사용이 불가능 하다. JEUS 서버 라이브러리에 배포된 JAXB API, IMPL 및 참조 라이브러리(JAXB 1.x)와 Apache CXF를 사용하여 구현한 웹 어플리케이션에 배포된 라이브러리들(JAXB 2.x)과 버전 차이로 동작하지 않는다.

### 3.2.2.6.0

추가 설정 없이 Apache CXF 사용이 가능하다. JEUS 서버 라이브러리에 배포된 JAXB API, IMPL 및 참조 라이브러리(JAXB 2.x)와 Apache CXF를 사용하여 구현한 웹 어플리케이션에 배포된 라이브러리들(JAXB 2.x)의 버전 일치로 문제없이 동작한다.

단, Anyframe에서 제공하는 Plugin들은 Jetty 기반으로 구성되어 있으므로 기본적으로 8080 포트를 이용한다. 따라서, JEUS를 활용할 경우에는 샘플 어플리케이션의 cxfjaxws-servlet.xml 파일을 열고, 'http://localhost:8080/...' 부분을 'http://localhost:{JEUS 사용 포트}/...'로 변경해주어야 한다.

## 3.3.WebLogic

Apache CXF는 JDK 1.5 이상을 지원하므로 WebLogic 서버의 경우, WebLogic 9.2(JDK 1.5), 10.1(JDK 1.5), 10.3(JDK 1.6) 버전의 서버가 대상이 된다. WebLogic 서버에 대한 설명 및 다운로드는 이곳 [<http://www.oracle.com/appserver/index.html>]을 참고하도록 한다.

단, Plugin 설치로 생성된 샘플 어플리케이션일 경우, 오픈소스 활용을 위한 WAS별 조치 사항을 확인하기 위해 먼저 설치된 각 Plugin 매뉴얼 내의 WAS(Web Application Server)별 유의사항 을 참고하도록 한다.

### 3.3.1.9.2

cxffaxws plugin 설치 시 라이브러리 문제로 인해 아래와 같이 추가 작업이 필요하며, EAR 파일로 웹어플리케이션을 구성하는 방법[방법 1] 혹은 geronimo-ws-metadata\_2.0\_spec-1.1.2.jar 파일을 복사하는 방법[방법 2] 중에서 선택할 수 있다.

[방법 1] EAR 폴더 형태로 웹어플리케이션을 구성하는 방법

- Anyframe에서 제공하는 Plugin들은 Jetty 기반으로 구성되어 있으므로 기본적으로 8080 포트를 이용한다. 따라서, WebLogic을 활용할 경우에는 샘플 어플리케이션의 cxf-jaxws-servlet.xml 파일을 열고, 'http://localhost:8080/...' 부분을 'http://localhost:{Weblogic 사용 포트}/...'로 변경해주어야 한다.
- EAR Folder를 구성한다. 예를 들어 myproject라는 이름의 프로젝트를 ear로 작업한다면 다음과 같이 2개의 폴더 형태로 구성할 수 있다.

```
myproject.ear/ META-INF
                / myproject
```

- META-INF 폴더에 application.xml을 배포한다. 아래 내용을 예시로 참고한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE application PUBLIC "-//Sun Microsystems,Inc.//DTD J2EE Application 1.3//EN"
    "http://java.sun.com/dtd/application_1_3.dtd">
<application>
  <display-name>Sample</display-name>
  <module>
    <web>
      <web-uri>myproject</web-uri>
      <context-root>/myproject</context-root>
    </web>
  </module>
</application>
```

- META-INF 폴더에 weblogic-application.xml 파일을 배포한다. javax.jws package에 대해서 WEB-INF/lib 폴더 내에 있는 라이브러리를 우선 참조하도록 설정한다. 아래 내용을 예시로 참고한다. weblogic-application.xml 작성 방법은 이곳 [<http://cwiki.apache.org/CXF20DOC/application-server-specific-configuration-guide.html#ApplicationServerSpecificConfigurationGuide-WebLogic>] 을 참고하도록 한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<weblogic-application xmlns="http://www.bea.com/ns/weblogic/90">
  <prefer-application-packages>
    <package-name>javax.jws.*</package-name>
  </prefer-application-packages>
</weblogic-application>
```

- myproject 폴더 내 WEB-INF에 weblogic.xml 파일을 배포한다. 웹 어플리케이션 내 라이브러리 및 클래스 파일들을 우선 참조하도록 설정한다. 아래 내용을 예시로 참고한다.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<weblogic-web-app xmlns="http://www.bea.com/ns/weblogic/90">
  <container-descriptor>
    <prefer-web-inf-classes>true</prefer-web-inf-classes>
  </container-descriptor>
</weblogic-web-app>
```

- myproject 폴더 내 WEB-INF/lib 폴더 내에 있는 xmlbeans-x.x.x.jar 파일을 제거한다.

[방법 2] geronimo-ws-metadata\_2.0\_spec-1.1.2.jar 파일을 복사하는 방법

- JDK\_HOME/jre/lib/endorsed 폴더에 geronimo-ws-metadata\_2.0\_spec-1.1.2.jar 파일을 복사한다.
- WebLogic 서버 설치 시 설정했던 JDK 1.5의 위치를 확인하여 JDK\_HOME/jre/lib 폴더 하위에 endorsed 폴더를 생성하고, 현재 배포하려고 하는 웹 어플리케이션의 WEB-INF/lib 폴더 하위의 geronimo-ws-metadata\_2.0\_spec-1.1.2.jar 파일을 endorsed 폴더 하위로 복사해 넣도록 한다.

[참고사항] 이 경우, 위 작업 내용이 WebLogic 서버 전체에 영향을 미치므로 주의하도록 한다.

### 3.3.2.10.1

cxfjaws plugin 설치 시 라이브러리 문제로 인해 아래와 같이 추가 작업이 필요하다.

- Anyframe에서 제공하는 Plugin들은 Jetty 기반으로 구성되어 있으므로 기본적으로 8080 포트를 이용한다. 따라서, WebLogic을 활용할 경우에는 샘플 어플리케이션의 cxf-jaws-servlet.xml 파일을 열고, 'http://localhost:8080/...' 부분을 'http://localhost:{Weblogic 사용 포트}/...'로 변경해주어야 한다.
- EAR Folder를 구성한다. 예를 들어 myproject라는 이름의 프로젝트를 ear로 작업한다면 다음과 같이 2개의 폴더 형태로 구성할 수 있다.

```
myproject.ear/ META-INF
                / myproject
```

- META-INF 폴더에 application.xml을 배포한다. 아래 내용을 예시로 참고한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE application PUBLIC "-//Sun Microsystems, Inc.//DTD J2EE Application 1.3//EN"
  "http://java.sun.com/dtd/application_1_3.dtd">
<application>
  <display-name>Sample</display-name>
  <module>
    <web>
      <web-uri>myproject</web-uri>
      <context-root>/myproject</context-root>
    </web>
  </module>
</application>
```

- META-INF 폴더에 weblogic-application.xml 파일을 배포한다. javax.jws package에 대해서 WEB-INF/lib 폴더 내에 있는 라이브러리를 우선 참조하도록 설정한다. 아래 내용을 예시로 참고한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<weblogic-application>
  <prefer-application-packages>
    <package-name>javax.persistence.*</package-name>
    <package-name>javax.jws.*</package-name>
  </prefer-application-packages>
</weblogic-application>
```

- myproject 폴더 내 WEB-INF에 weblogic.xml 파일을 배포한다. 웹 어플리케이션 내 라이브러리 및 클래스 파일들을 우선 참조하도록 설정한다. 아래 내용을 예시로 참고한다.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<weblogic-web-app>
  <container-descriptor>
    <prefer-web-inf-classes>true</prefer-web-inf-classes>
  </container-descriptor>
</weblogic-web-app>
```

- myproject 폴더 내 WEB-INF/lib 폴더 내에 있는 xmlbeans-x.x.x.jar 파일을 제거한다.

### 3.3.3.10.3.3

cxf-jaxws plugin 설치 시 라이브러리 문제로 인해 아래와 같이 추가 작업이 필요하다.

- Anyframe에서 제공하는 Plugin들은 Jetty 기반으로 구성되어 있으므로 기본적으로 8080 포트를 이용한다. 따라서, WebLogic을 활용할 경우에는 샘플 어플리케이션의 cxf-jaxws-servlet.xml 파일을 열고, 'http://localhost:8080/...' 부분을 'http://localhost:{Weblogic 사용 포트}/...'로 변경해주어야 한다.
- EAR Folder를 구성한다. 예를 들어 myproject라는 이름의 프로젝트를 ear로 작업한다면 다음과 같이 2개의 폴더 형태로 구성할 수 있다.

```
myproject.ear/ META-INF
                / myproject
```

- META-INF 폴더에 application.xml을 배포한다. 아래 내용을 예시로 참고한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE application PUBLIC "-//Sun Microsystems, Inc.//DTD J2EE Application 1.3//EN"
  "http://java.sun.com/dtd/application_1_3.dtd">
<application>
  <display-name>Sample</display-name>
  <module>
    <web>
      <web-uri>myproject</web-uri>
      <context-root>/myproject</context-root>
    </web>
  </module>
</application>
```

- META-INF 폴더에 weblogic-application.xml 파일을 배포한다. 아래 내용을 예시로 참고한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<weblogic-application>
  <prefer-application-packages>
    <package-name>javax.persistence.*</package-name>
  </prefer-application-packages>
</weblogic-application>
```

- myproject 폴더 내 WEB-INF에 weblogic.xml 파일을 배포한다. 웹 어플리케이션 내 라이브러리 및 클래스 파일들을 우선 참조하도록 설정한다. 아래 내용을 예시로 참고한다.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<weblogic-web-app>
  <container-descriptor>
    <prefer-web-inf-classes>true</prefer-web-inf-classes>
  </container-descriptor>
</weblogic-web-app>
```

- Apache CXF 2.3 사용 시: 현재 배포되는 Anyframe cxf-jaxws plugin은 Apache CXF 2.3 버전을 사용하고 있다. 이 경우 WEB-INF/lib 폴더 내에 있는 xercesImpl-x.x.x.jar 파일을 제거하고, 사용하려는 Weblogic 도메인 하위의 lib 폴더에 배포시키도록 한다.

[참고] Apache CXF 2.2.7 사용 시: 이전 버전의 Anyframe cxf plugin은 Apache CXF 2.2.7 버전을 사용하고 있다. 이 경우 WEB-INF/lib 폴더 내에 있는 xercesImpl-x.x.x.jar, stax-api-x.x.jar 파일을 제거하고 이 중 xercesImpl-x.x.x.jar 파일만을 사용하려는 Weblogic 도메인 하위의 lib 폴더에 배포시키도록 한다.

- myproject 폴더 내 WEB-INF/lib 폴더 내에 있는 xmlbeans-x.x.x.jar 파일과 geronimo-stax-api\_1.0\_spec-x.x.x.jar 파일을 제거한다.

---

## 4.Resources

- 다운로드

다음에서 sample 코드를 포함하고 있는 anyframe-sample-cxf-jaxws.zip 파일을 다운받은 후, 압축을 해제한다.

- Maven 기반 실행

Command 창에서 압축 해제 폴더로 이동한 후, mvn clean test 라는 명령어를 실행시켜 결과를 확인한다.

- Eclipse 기반 실행

Eclipse에서 압축 해제 프로젝트를 import한 후, src/test/java 폴더의 모든 Test 코드 각각에 대해 마우스 오른쪽 버튼을 클릭하고, 컨텍스트 메뉴에서 Run As > JUnit Test를 클릭한다. 그리고 실행 결과를 확인한다.

### 표 4.1. Download List

Name	Download
anyframe-sample-cxf-jaxws.zip	Download [ <a href="http://dev.anyframejava.org/docs/anyframe/plugin/optional/cxf-jaxws/1.0.2/reference/sample/anyframe-sample-cxf-jaxws.zip">http://dev.anyframejava.org/docs/anyframe/plugin/optional/cxf-jaxws/1.0.2/reference/sample/anyframe-sample-cxf-jaxws.zip</a> ]