

Anyframe CXF JAX-RS 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. RESTful Services	2
1. Overview	3
2. JAX-RS Frontend	5
2.1. JAX-RS 활용한 RESTful 서비스 구현	5
2.1.1. RESTful Web Service 작성	5
2.1.2. Spring Configuration XML - <jaxrs:server/> tag 사용	10
2.1.3. Spring Configuration XML - <jaxrs:client/> tag 사용	12
2.2. Resources	14
3. WAS(Web Application Server) Configuration	15
3.1. Tomcat	15
3.1.1. 5.5.23, 6.0.x, 7.0.x	15
3.2. JEUS	15
3.2.1. 5.0	15
3.2.2. 6.0	15
3.3. WebLogic	16
3.3.1. 9.2	16
3.3.2. 10.1	17
3.3.3. 10.3.3	18
4. Resources	20

I.Introduction

웹 서비스 구현을 위해 많이 사용하고 있는 오픈소스 프레임워크에는 Apache CXF, Apache Axis/Axis2, Spring Web Services 등등 여러가지가 존재한다. cxf-jaxrs plugin은 이 중 Apache CXF [<http://cxf.apache.org/>]의 JAX-RS (JSR-311) Frontend 활용 방법을 가이드하기 위한 샘플 코드와 이 오픈 소스를 활용하는데 필요한 참조 라이브러리들로 구성되어 있다.

Installation

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

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

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

Dependent Plugins

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

II.RESTful Services

Anyframe은 **Apache CXF 2.3.0 버전**을 이용하여 웹 서비스 기능을 제공하고 있는데 SOAP 프로토콜 방식의 웹서비스와 함께 HTTP 프로토콜 상에서 REST 아키텍처 스타일의 웹서비스 기능을 함께 제공하고 있다. SOAP 프로토콜 방식의 웹서비스는 Anyframe cxf-jaxws plugin 매뉴얼을 참고한다.

1. Overview

REST에 대한 자세한 내용은 **Anyframe Spring REST Plugin 매뉴얼**을 참고하도록 한다. 여기서는 REST에 대해 간단히 소개만 하도록 하고 JAX-RS 기반으로 RESTful 서비스를 구현하는 방법에 대해서 설명하고 있다.

Representational state transfer (REST)는 World Wide Web처럼 분산 하이퍼미디어 시스템을 위한 소프트웨어 아키텍처 스타일이다. 요즘 웹 서비스를 논할 때마다, 화제가 되는 REST는 Roy T Fielding이 제안한, 네트워크 기반 어플리케이션을 위한 아키텍처 *스타일*로 표준이나 스펙이 아님에 유의한다. Representational state transfer은 마치 리소스(Resource)가 요청(Request)에 따라 상태가 변화하는 것처럼 동작한다는 의미에서 작성된 용어이다. RESTful하다는 의미는 REST의 원칙을 따른다는 뜻이며, RESTful Web Service는 RESTful 아키텍처 스타일을 사용하여 작성한 서비스이다. RESTful 방식의 웹 서비스 작성은 SOAP 프로토콜을 기반으로 하여 인터넷에서 서비스를 배포하는 방법의 대안으로 주목받고 있다. SOAP 기반 웹서비스에 비해 가벼울 뿐 아니라 직접 HTTP를 통해 데이터를 보낼 수 있기 때문이다.

REST(XML over HTTP) 특징 을 살펴보면 다음과 같다.

- **HTTP와 XML 이용**

HTTP와 XML을 이용하여 데이터를 주고 받는 웹 서비스를 이용한다.

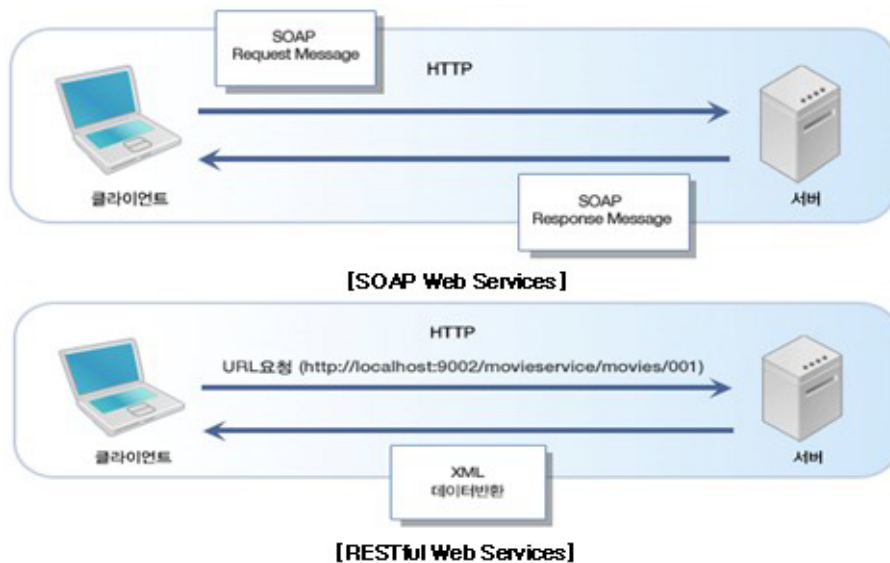
- **소프트웨어 아키텍처 스타일**

표준 기술이 아닌, 소프트웨어 아키텍처 스타일이다. 아키텍처 스타일은 아키텍처적인 제한 및 제약 조건을 정의해놓은 집합이다. HTTP와 같은 기존의 기술을 이용하여 잘 설계된 분산 웹 어플리케이션을 만들수 있도록 하는 설계 가이드에 해당한다.

- **WWW(웹), Open API**

적용 예로 WWW(웹), Open API(ex. Google, Microsoft, Amazone, eBay, Yahoo등)가 있다. 더 자세한 내용은 REST 적용 예 를 참고한다.

다음은 SOAP을 이용한 웹 서비스와 REST 방식의 웹 서비스를 비교한 그림(예제)이다. 그림에서와 같이 "http://***/movieservice/movies/001" 형태의 URL을 요청한 경우, 그 결과 데이터는 XML 형식으로 반환된다.



Apache CXF을 이용하여 **RESTful** 서비스를 구현하는 방법 에는 다음과 같이 3가지 방법이 있다. 이중 Anyframe에서는 표준 API인 JAX-RS (JSR-311)를 사용하는 JAX-RS Frontend 모델을 채택하여 사용하도록 권고하고 있으므로 HTTP Binding, JAX-WS Provider/Dispatch API를 사용하여 RESTful 서비스를 구현하는 방법에 관한 상세한 내용은 매뉴얼에서 언급하고 있지 않다.

- **JAX-RS Frontend**

JAX-RS(JSR-311) Spec. 을 이용하여 RESTful 서비스를 구현하는데, Apache CXF에서 Spec.에 대한 구현체를 제공하여 표준 방식으로 RESTful 서비스를 작성할 수 있게 한다. JAX-RS에서 제공하는 Annotation 설정을 이용한다.

- **HTTP Binding**

표준 방식이 아닌 방법으로 가장 쉽고 용이한 방식으로 RESTful 서비스를 작성할 수 있게 한다. JRA(Java REST Annotation) 설정 또는 Naming Convention 기반의 매핑 방식을 이용한다.

- **JAX-WS Provider/Dispatch API**

간단한 RESTful 서비스 작성을 지원한다. 표준 API를 사용하나 HTTP Binding 방식에 비해 유연한 구조가 아니다.

REST 적용 예는 아래와 같다.

- WWW(웹)

REST 적용 예시로 가장 많이 사용되는 것이 바로 WWW(웹)이다.

설명
상태를 유지하지 않는 클라이언트/서버 구조를 가진다.
어디에서나 적용되는 인터페이스를 가진다. (e.g., GET, POST, PUT, DELETE)
모든 자원은 URI를 이용하여 유일하게 지칭될 수 있다.
자원들의 표현(Representation)들이 URI을 통해 서로 연결되어 있다.

- Open API

설명
특정 기능 또는 콘텐츠를 가진 서비스 업체가 자신들의 서비스에 접근할 수 있도록 외부에 접근방법을 공개하는 것을 Open API라고 한다.
Open API를 이용하여 새로운 서비스를 개발할 수 있다. 즉 Open API는 해당 서비스로 접근하기 위한 규약 또는 표준적인 인터페이스를 의미한다.
Google, Microsoft, Amazon, eBay, Yahoo에서는 이미 여러 Open API를 공개하고 있다.

2.JAX-RS Frontend

JAX-RS (JSR-311)를 사용하여 Annotation 설정을 통해 RESTful Web Services를 구현할 수 있게 해주는 Frontend 모델이다.

JAX-RS(Java API for RESTful Web Services)는 자바 플랫폼 상에서 REST 방식의 웹 서비스 구현을 지원하는 자바 API로 Java SE 5에서 도입된 어노테이션(Annotation)을 사용하여 RESTful Web Services 서버 모듈의 개발 및 배포를 쉽게 하고 있다. 오픈 소스 중 대표적인 구현체들로 Apache CXF [http://cxf.apache.org/docs/restful-services.html], Jersey [http://jersey.java.net/], Restlet [http://www.restlet.org/], JBoss RESTEasy [http://www.jboss.org/resteasy] 등이 있다.

2.1.JAX-RS 활용한 RESTful 서비스 구현

JAX-RS(JSR-311) Spec [http://jcp.org/en/jsr/detail?id=311]. 을 이용하여 RESTful 서비스를 구현하는데, Apache CXF에서 Spec.에 대한 구현체를 제공하여 표준 방식으로 RESTful 서비스를 작성할 수 있게 한다. JAX-RS에서 제공하는 Annotation 설정을 이용한다.

Web Service로 노출시킬 서비스 인터페이스 클래스의 각 method별로 JAX-RS에서 제공하는 Annotation(javax.ws.rs.*)을 설정하여 RESTful Web Service를 손쉽게 구현할 수 있다. (ex. @Path, @GET, @PUT, @DELETE, @Produces 등)

다음은 JAX-RS를 활용하여 RESTful 서비스 구현 시 Server와 Client 단에서 어떻게 사용해야 하는지에 대한 사용법이다.

- Server Configuration
 1. RESTful Web Service 작성
 2. 서버 구동
 - Spring Configuration XML - <jaxrs:server/> tag 사용
- Client Configuration
 1. 클라이언트 작성
 - Spring Configuration XML - <jaxrs:client/> tag 사용

2.1.1.RESTful Web Service 작성

Movie Service를 JAX-RS Annotation을 사용하여 RESTful Web Services로 노출시켜보도록 한다.

RESTful Web Services를 통해 접근할 수 있게 웹 리소스를 정의하는데, 웹 리소스는 리소스 클래스로 구현하며 리소스 메소드를 이용하여 요청을 처리하도록 한다. 이때 리소스 메소드는 public 지시자로 작성하며 리턴 타입으로 void나 Response 혹은 다른 자바 타입 등을 사용할 수 있다.

Annotation 설정 정보는 다음 표와 같다.

Annotation	Description	Example
@Path	리소스를 URI와 결합시키기 위해 @Path Annotation을 사용하며 상대 경로로 표현하며 이 Annotation은 클래스와 메소드에 설정할 수 있다. @Path Annotation 설정 값은 정규 표현식을 사용할 수도 있다.	@Path("/movies/"), @Path("/{movieid}/"), @Path("/subresource/{n1:.}*")
@HttpMethod	JAX-RS는 @GET, @PUT, @POST, @DELETE와 같은 다양한 HTTP	@Target({ElementType.METHOD}) @Retention(RetentionPolicy.RUNTIME)

Annotation	Description	Example
	method Annotation을 제공하고 있다. @HttpMethod Annotation을 이용하면 커스텀 Annotation을 만들어 사용할 수 있다.	@HttpMethod("PATCH") public @interface PATCH {}
@Produces	[Media Type] HTTP Response의 MIME Type을 지정한다. 클라이언트에 반환되는 타입을 무엇으로 할 것인지 지정하기 위한 Annotation이다. Accept Header 정보를 통해 타입을 지정할 수 있다.	@Produces("application/xml"), @Produces("application/json")
@Consumes	[Media Type] HTTP Request의 MIME Type을 지정한다. Request body에 대한 타입을 무엇으로 할 것인지 지정하기 위한 Annotation이다. Content-Type Header 정보를 통해 타입을 지정할 수 있다.	@Consumes({"application/xml", "application/json"}), @Consumes("application/x-www-form-urlencoded")
@GET	[HTTP Method] 리소스 메소드에 설정하는 Annotation으로 HTTP 요청 메소드 타입을 표시한다. 목록 및 상세 조회 메소드에 사용한다.	@GET @Path("/{movieId}/") Response get(@PathParam("movieId") String movieId)
@POST	[HTTP Method] 리소스 메소드에 설정하는 Annotation으로 HTTP 요청 메소드 타입을 표시한다. 생성 메소드에 사용한다.	@POST Response create(Movie movie)
@PUT	[HTTP Method] 리소스 메소드에 설정하는 Annotation으로 HTTP 요청 메소드 타입을 표시한다. 수정 메소드에 사용한다.	@PUT Response update(Movie movie)
@DELETE	[HTTP Method] 리소스 메소드에 설정하는 Annotation으로 HTTP 요청 메소드 타입을 표시한다. 삭제 및 취소 메소드에 사용한다.	@DELETE @Path("/{movieId}/") Response remove(@PathParam("movieId") String movieId)
@PathParam	[PARAMETER] URI template에 명시되어 있는 값을 얻는다.	get(@PathParam("movieId") String movieId)
@QueryParam	[PARAMETER] URI query 파라미터 값을 얻는다. 객체로 바로 매핑하여 사용하는 경우 @QueryParam Annotation 값에 빈 스트링을 입력한다.	http://.../movies? title=Avatar&pageIndex=1와 같은 형 태로 요청이 된 경우, getPagingList(@QueryParam("") Movie movie, @QueryParam("pageIndex") int pageIndex) 와 같이 title을 멤버변수로 갖 는 Movie 객체로 바로 값이 바인딩될 수 있다.
@FormParam	[PARAMETER] Form Submit이 발생한 경우 Form에 저장된 값을 읽어낼 때 사용되는 Annotation이다. 메소드 파라미터 중, Annotation이 없는 파라미터를 엔티티 파라미터라고 하는데 @FormParam Annotation을 사용하는 경우 엔티티 파라미터의 타입은 MultivaluedMap<String,String>이어야 한다.	create(@FormParam("movieId") String movieId, MultivaluedMap<String, String> form)
@HeaderParam	[PARAMETER] header의 값을 얻는다.	Book getBookByHeader(@HeaderParam("BOOK2") String id)

Annotation	Description	Example
@MatrixParam	[PARAMETER] matrix URI로부터 값을 얻는다. matrix URI란 http://.../movies/color;lat=50;long=20;scale=32000와 같이 세미콜론을 사용하여 작성된 URI를 말한다.	Book getBookByMatrixParams(@MatrixParam("first") String s1, @MatrixParam("second") String s2)
@CookieParam	[PARAMETER] 쿠키 값을 얻는다.	Book getTheBook2(@CookieParam("n5") String name5)
@DefaultValue	[PARAMETER] @PathParam, @QueryParam, @HeaderParam, @CookieParam Annotation에서 기본값을 설정할 때 사용한다.	getPagingList(@QueryParam("pageIndex") @DefaultValue("1") int pageIndex)
@Encoded	[PARAMETER] @PathParam, @QueryParam, @FormParam, @MatrixParam Annotation 사용 시 파라미터 값을 자동으로 디코딩시키지 않을 때 사용한다. 이 Annotation은 클래스와 메소드에 설정할 수 있다. 클래스에 설정한 경우, 모든 메소드의 모든 파라미터 값에 대해서 디코딩시키지 않게 된다.	@Encoded
@Context	HTTP Request Header나 URI 등의 Inject된 정보들을 이용할 때 사용한다.	get(@Context UriInfo uri) { MultivaluedMap<String,String> params = uri.getQueryParameters();}, get(@Context HttpHeaders header) { Map<String,Cookie> map = header.getCookies();}

2.1.1.1.Samples

다음은 Movie Service의 인터페이스 클래스 정의에 대한 예제이다.

- Interface Class

다음은 Movie Service의 인터페이스 클래스를 작성한 MovieService.java 의 일부이다. Annotation 설정에 유의하도록 한다.

```
import javax.ws.rs.Path;
import javax.ws.rs.Produces;

@Path("/movies/")
@Produces("application/xml")
public interface MovieService {

    @POST
    Response create(Movie movie) throws Exception;

    @GET
    @Path("/{movieId}")
    Response get(@PathParam("movieId") String movieId)
        throws Exception;

    @PUT
    Response update(Movie movie) throws Exception;
}
```

```

@DELETE
@Path("/{movieId}")
Response remove(@PathParam("movieId") String movieId)
    throws Exception;

@GET
Response getPagingList(@QueryParam("") Movie movie,
    @QueryParam("pageIndex") @DefaultValue("1") int pageIndex)
    throws Exception;
중략...

```

- Implementation Class

Interface Class를 구현한 클래스로 JAX-RS 관련 Annotation 설정 부분 없이 작성될 수 있다. 이때 서비스 메소드 별로 Response 객체를 리턴할 수도 있고 리턴하지 않을 수도 있다. cxf-jaxrs plugin 샘플 코드로 제공되는 예제에서는 Response 객체를 리턴하는 것으로 구현되어 있는데 이는 Response 객체에 Status Code 값을 설정하여 보내줄 수 있기 때문이다.

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

```

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

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

    public Response create(Movie movie) throws Exception {
        movieDao.create(movie);
        return Response.status(Status.CREATED).build();
    }

    public Response get(String movieId) throws Exception {
        Movie movie = movieDao.get(movieId);
        if (movie == null)
            return Response.status(Status.NOT_FOUND).build();
        return Response.ok(movie).build();
    }

    public Response update(Movie movie) throws Exception {
        this.movieDao.update(movie);
        return Response.status(Status.NO_CONTENT).build();
    }

    public Response remove(String movieId) throws Exception {
        this.movieDao.remove(movieId);
        return Response.status(Status.NO_CONTENT).build();
    }

    public Response getPagingList(Movie movie, int pageIndex) throws Exception {
        ResultPage resultPage = new ResultPage();
        resultPage.setPage(this.movieDao.getPagingList(movie, pageIndex));
        return Response.ok(resultPage).build();
    }
중략...

```

- Java Beans Class

다음은 Movie Service의 인터페이스 클래스에서 상세 조회 시 리턴 값으로 사용하는 Movie.java 의 일부이다. Movie 클래스 정의 시 작성한 @XmlElement Annotation 설정에 유의하도록 한다. JAXB 를 이용하여 XML을 JavaBeans 객체로 변환 시 이 Annotation 정보를 이용한다.

```
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class Movie implements Serializable {
    private String movieId;

    @NotNull
    @Size(min = 1, max = 50)
    private String title = "";

    @NotNull
    @Size(min = 1, max = 50)
    private String director;

    중략...
```

다음은 Movie Service의 인터페이스 클래스에서 목록 조회 시 리턴 값으로 사용하는 ResultPage.java 의 일부이다. ResultPage 클래스 정의 시 작성한 @XmlElement와 @XmlSeeAlso(Movie.class) Annotation 설정에 유의하도록 한다. JAXB를 이용하여 XML을 JavaBeans 객체로 변환 시 이 Annotation 정보를 이용한다. 특히 목록 조회 결과 시 사용되는 리턴 값은 내부 멤버 변수로 Page 객체를 사용하는데 이때 이 Page 클래스는 org.anyframe.pagination.Page 클래스로 내부에 Collection 객체를 가지고 있다. 이 Collection 객체내에 저장된 JavaBeans 객체를 JAXB Databinding 시키기 위해 @XmlSeeAlso(Movie.class) Annotation을 설정하고 있다.

```
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlSeeAlso;
import org.anyframe.pagination.Page;

@XmlRootElement
@XmlSeeAlso(Movie.class)
public class ResultPage implements Serializable {
    private Page page;

    public Page getPage() {
        return page;
    }

    public void setPage(Page page) {
        this.page = page;
    }

    중략...
```

- Exception Transfer Aspect for error handling

Movie Service의 비즈니스 메소드 수행 중 에러가 발생하는 경우, Response 객체에 Internal Server Error(500) Code 값을 전달해 주기 위해 ExceptionTransfer Aspect 클래스를 작성한다.

다음은 ExceptionTransfer.java 의 일부이다.

```
@Aspect
@Service("cxfJaxRsExceptionTransfer")
public class ExceptionTransfer {

    @Pointcut("execution(* myproject.cxf.jaxrs.*Impl.*(..))")
    public void serviceMethod() {
```

```

}

@Around("serviceMethod()")
public Object aroundExecutesServiceMethod(ProceedingJoinPoint thisJoinPoint)
throws Throwable {
    Object target = thisJoinPoint.getTarget();
    String className = target.getClass().getSimpleName().toLowerCase();
    String opName = (thisJoinPoint.getSignature().getName()).toLowerCase();

    Log logger = LogFactory.getLog(target.getClass());

    logger.debug("***** Around Advice of ExceptionTransfer [" + className + "."
        + opName + "()");

    // before logic
    Object retVal = null;
    try {
        retVal = thisJoinPoint.proceed();
    } catch (Exception e) {
        return Response.serverError().build();
    }
    // after logic
    return retVal;
}
종막...

```

2.1.2.Spring Configuration XML - <jaxrs:server/> tag 사용

작성된 서비스를 Web Service로 노출시키는 서버를 Spring Configuration XML - <jaxrs:server/> tag를 사용하여 구동시켜보도록 한다.

<jaxrs:server/> 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 다양한 사용법에 대한 설명은 JAX-RS Spring Configuration [<http://cxf.apache.org/docs/jax-rs.html#JAX-RS-ConfiguringJAXRSServicesincontainerwithSpringconfigurationfile>.] 내용을 참고하도록 한다.

Property Name	Description	Required	Default Value
id	spring bean id를 작성한다.	Y	N/A
address	서비스가 동작할 주소를 상대 경로로 작성한다.	Y	N/A
[Child Tag] serviceBeans	RESTful Web Services로 노출할 구현 클래스를 작성하는데 클래스명을 직접 작성하거나 해당 클래스에 대한 spring bean id를 작성한다. (ex. <bean class="myproject.cxf.jaxrs.moviefinder.service.impl.MovieServiceImpl"/> or <ref bean="cxfJaxRsMovieService" />)	Y	N/A

2.1.2.1.Samples

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

- 다음은 비즈니스 레이어의 서비스를 Web Service로 노출시키는 <jaxrs:server/> tag 속성을 정의한 context-cxf-jaxrs.xml 의 일부이다.

Configuration

```

<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:cxfr="http://cxf.apache.org/core"
  xmlns:jaxrs="http://cxf.apache.org/jaxrs"
  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/jaxrs http://cxf.apache.org/schemas/jaxrs.xsd">

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

  <jaxrs:server id="restCxfJaxRsMovieService" address="/movie">
  <jaxrs:serviceBeans>
  <ref bean="cxfJaxRsMovieService" />
  </jaxrs:serviceBeans>
  <!-- Enable message logging using the CXF logging feature -->
  <jaxrs:features>
  <cxfr:logging />
  </jaxrs:features>
  </jaxrs:server>
  종략...

```

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-JAXRS-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-JAXRS-Servlet</servlet-name>
    <url-pattern>/cxf-jaxrs/*</url-pattern>
  </servlet-mapping>
  종략...
</web-app>

```

2.1.3.Spring Configuration XML - <jaxrs:client/> tag 사용

RESTful Web Services에 접근하기 위한 클라이언트를 작성한다. Apache CXF에서 제공하는 WebClient API를 이용하면 손쉽게 클라이언트 코드를 작성할 수 있다. 여기서는 간단한 예제를 중심으로 설명하고 있으므로 상세한 내용은 Apache CXF 매뉴얼 내용 중 JAX-RS Client API 부분 [<http://cxf.apache.org/docs/jax-rs.html#JAX-RS-ClientAPI>]을 참고하도록 한다.

2.1.3.1.Samples

다음은 Spring MVC Controller에서 <jaxrs:client/> tag와 WebClient API를 이용하여 RESTful Web Services로 노출된 Movie Service에 접근하는 예제이다. 이때, RESTful Webservice의 결과 값이 XML 형태로 리턴되는데 이를 JAXB Databinding을 통해 JavaBeans로 객체 변환하여 클라이언트 코드에서 사용할 수 있다.

- Spring Configuration XML Client 설정

RESTful 서비스에 접근하기 위한 클라이언트를 구현할 때 Spring 설정 파일을 이용할 수 있다.

다음은 RESTful 서비스에 접근하기 위한 클라이언트를 구현하기 위해 <jaxrs:client/> tag 속성을 정의한 cxfjaxrs-servlet.xml의 일부이다.

Configuration

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

  <jaxrs:client id="cxfJaxRsClientMovieService"
    address="http://localhost:8080/myproject/cxf-jaxrs"
    serviceclass="myproject.cxf.jaxrs.moviefinder.service.MovieService">
  </jaxrs:client>
  종략...
```

- Exception Handling

RESTful Web Services를 호출하여 사용 시, 에러가 발생하는 경우 WebClient에서 javax.ws.rs.WebApplicationException을 throw하고 있으므로 이 Exception을 처리하는 Exception Resolver를 Spring Configuration XML에 등록하여 사용하도록 한다.

다음은 SimpleMappingExceptionHandler 클래스를 정의한 cxfjaxrs-servlet.xml의 일부이다. javax.ws.rs.WebApplicationException이 발생하는 경우 default error view가 아닌 cxfJaxRsError view로 이동하도록 설정한다.

```
<bean id="cxfJaxRsExceptionHandler"
  class="org.springframework.web.servlet.handler.SimpleMappingExceptionHandler">
  <property name="exceptionMappings">
  <props>
    <prop key="javax.ws.rs.WebApplicationException">cxf-jaxrs/common/error</prop>
  </props>
  </property>
  <property name="defaultErrorView" value="forward:/sample/common/error.jsp" />
  <property name="order" value="1" />
</bean>
  종략...
```

- Test Case

다음은 RESTful Web Services로 노출된 Movie Service에 접근하는 클라이언트 코드를 작성한 Controller 클래스의 일부이다.

```
@Controller("cxfJaxRsMovieController")
@DependsOn("cxfJaxRsClientMovieService")
@RequestMapping("/cxfJaxRsMovie.do")
public class MovieController {
    @Inject
    @Named("cxfJaxRsClientMovieService")
    private MovieService movieService;

    private WebClient client;

    private WebClient getClient() {
        if (this.client == null)
            client = WebClient.fromClient(WebClient.client(movieService));
        return client.reset();
    }

    @RequestMapping(params = "method=create")
    public String create(Movie movie, BindingResult results,
        SessionStatus status) throws Exception {
        if (results.hasErrors()) {
            return "cxf-jaxrs/moviefinder/movie/form";
        }
        Response response = getClient().path("/movies").post(movie);
        if (response.getStatus() == Status.INTERNAL_SERVER_ERROR
            .getStatusCode()) {
            throw new Exception("Fail to create : Movie Title="
                + movie.getTitle());
        }

        status.setComplete();
        return "redirect:/cxfJaxRsMovie.do?method=list";
    }

    @RequestMapping(params = "method=get")
    public String get(@RequestParam("movieId") String movieId, Model model)
        throws Exception {
        Movie movie = getClient().path("/movies/" + movieId).get(Movie.class);
        model.addAttribute("movie", movie);

        return "cxf-jaxrs/moviefinder/movie/form";
    }

    @RequestMapping(params = "method=list")
    public String list(
        @RequestParam(value = "pageIndex", defaultValue = "1") int pageIndex,
        Movie movie, BindingResult result, Model model) throws Exception {
        ResultPage resultPage = getClient().path("/movies").query("title",
            movie.getTitle()).query("nowPlaying", movie.getNowPlaying())
            .query("pageIndex", pageIndex).get(ResultPage.class);
        Page page = resultPage.getPage();

        model.addAttribute("movie", movie);
        model.addAttribute("movies", page.getList());
        model.addAttribute("resultPage", page);

        return "cxf-jaxrs/moviefinder/movie/list";
    }
}
```

중략...

2.2.Resources

- 참고자료
 - Architectural Styles and the Design of Network-based Software Architectures, Roy Thomas Fielding, 2000 [<http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>]
 - Sun Article: RESTful Web Services [<http://java.sun.com/developer/technicalArticles/WebServices/restful/>]
 - RESTful Web Services slides [<http://www.stefan-marr.de/downloads/RESTful-Web-Services.slides.pdf>]
 - REST Wiki FrontPage [<http://rest.blueoxen.net/cgi-bin/wiki.pl?FrontPage>]
 - RESTful Web Services, John Cowan, 2005 [<http://mercury.ccil.org/~cowan/restws.pdf>]
 - Building Web Services the REST Way [<http://www.xfront.com/REST-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 서버에 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 cxf-jaxrs-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

cxfjaxrs 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 폴더로 웹어플리케이션을 구성하는 방법

- Anyframe에서 제공하는 Plugin들은 Jetty 기반으로 구성되어 있으므로 기본적으로 8080 포트를 이용한다. 따라서, WebLogic을 활용할 경우에는 샘플 어플리케이션의 cxf-jaxrs-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

cxfjaxrs 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 cxfjaxrs-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

cxfrjars 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 cxfrjars-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-jaxrs 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-jaxrs.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-jaxrs.zip	Download [http://dev.anyframejava.org/docs/anyframe/plugin/optional/cxf-jaxrs/1.0.2/reference/sample/anyframe-sample-cxf-jaxrs.zip]