Part04 Enterprise 서비스

Chapter 11 OXM (Object XML Mapping)

1. SPRING OXM

: 시스템 간 정보 연동을 위해 WEB 서비스가 사용되는 비중이 높아짐에 따라 정보를 주고 받기 위한 데이터 포맷으로 XML이 사용되는 비중도 점차 높아지고

있다. 이에 따라, XML 문서와 Java 객체 사이의 Mapping을 처리해 주어야 할 필요성도 높아지고 있다. 이에 따라, XML 문서와 Java 객체 사이의 Mapping을 처리해

주어야 할 필요성도 높아지고 있다. SPRING 3버전은 이런 흐름에 발 맞추어 Java 객체와 XML 사이의 Mapping을 처리해 주는 OXM Module을 추가했으며, SPRING

Framework를 사용하는 개발자들은 구현 기술에 상관없이 SPRING OXM Module이 제공하는 Interface를 이용해서 Java 객체와 XML 사이의 Mapping을 처리할 수 있다.

SPRING OXM Module을 사용할 경우의 장점은 다음과 같다.

- SPRING이 제공하는 빈 팩토리 Class를 이용해서 손쉽게 설정할 수 있다.

- 사용하는 구현 기술에 상관 없이 Marshaller와 Unmarshaller Interface를 통해서 OXM을 처리하기 때문에, 약간의 노력(설정 파일 변경 및 일부 Code 변경)만으로

구현 기술을 변경할 수 있다.

- 사용하는 구현 기술에 상관 없이 SPRING이 제공하는 XmlMappingException을 발생시킨다. 따라서, 구현 기술이 변경되더라도 예외 처리를 변경할 필요가 없다.

본 장에서는 OXM의 주요 API를 살펴보고, SPRING과 JAXB2를 연동하는 방법을 살펴보도록 하겠다.

1.1 Marshaller와 Unmarshaller Interface

: OXM Module의 핵심은 Marshaller와 Unmarshaller Interface이다. 먼저 Marshaller Interface는 Java 객체를 XML 객체로 변환해주는 기능을 정의한 Interface로서

다음과 같이 정의되어 있다.

package org.springframework.oxm;

import java.io.IOException;

import javax.xml.transform.Result;

public interface Marshaller{

boolean supports(Class<?> class);

void marshall(Object graph, Result result) throws IOException, XmlMappingException;

}

marshall() 메서드는 첫 번째 파라미터로 전달받은 객체(graph)를 XML 출력을 표�h하는 javax.xml.transform.Result 객체로 변환해준다.

Unmarshaller는 XML을 Java 객체로 변환해 주는 기능을 정의한 Interface로서 다음과 같이 정의되어 있다.

package org.springframework.oxm;

import java.io.IOException;

import javax.xml.transform.Source;

public interface Unmarshaller{

boolean supports(Class<?> class);

void unmarshall(Object graph, Source source) throws IOException, XmlMappingException;

}

unmarshaller() 메서드는 XML입력을 표현하는 Source 객체를 전달받아 Java 객체로 변환해준다.

Marshaller와 Unmarshaller를 사용하려면 각 구현 기술에 알맞은 Marshaller/Unmarshaller 구현 객체를 SPRING 설정으로 생성한 뒤, 각 객체를 이용해서

Java 객체와 XML 사이의 Mapping을 처리하면 된다. OXM을 위한 구현 기술로는 JAXB2, XStream, JiBX, XMLBeans 등 다양한 API가 존재하는데, 본장에서는

JAXB2를 이용해서 OXM API를 사용하는 방법을 사용하도록 하겠다.

1.2 XmlMappingException 예외

: Marshaller와 Unmarshaller 구현 Class는 내부적으로 사용되는 OXM 기술에서 발생시킨 예외를 SPRING OXM Module에 정의된 XmlMappingException 예외 타입으로

변환해 준다. XmlMappingException Class는 런타임 예외이며, 의미 있는 하위 예외 타입을 추가로 제공하고 있다.

2. JAXB 2를 이용한 OXM

: OXM Module은 기본적으로 JAXB(Java Architecture for XML Binding) 2버전을 쉽게 연동할 수 있는 Class를 제공하고 있다. 본 절에서는 JAXB2를 이용해서

OXM Module을 사용하는 방법에 대해서 살펴보도록 하겠다. JAXB2 자체에 대한 내용은 다루지 않을 것이며, JAXB2에 대한 자세한 내용은

https://jaxb.dev.java.net를 참고하기 바란다.

2.1 Jaxb2Marshaller를 이용한 Marshaller/Unmarshaller 설정

: OXM Module은 Marshaller Interface와 Unmarshaller Interface를 모두 구현한 Jaxb2Marshaller Class를 제공하고 있다. 이 Class를 이용하면 손쉽게

JAXB2를 SPRING과 연동할 수 있다. 예를 들어, 다음과 같이 JAXB2 Annotation을 이용해서 XML과 Mapping될 Class를 작성했다고 하자.

import javax.xml.bind.annotation.XmlAccessType;

import javax.xml.bind.annotation.XmlAccessorType;

import javax.xml.bind.annotation.XmlAttribute;

import javax.xml.bind.annotation.XmlElement;

import javax.xml.bind.annotation.XmlRootElement;

@XmlAccessorType(XmlAccessType.FIELD)

@XmlRootElement(name = "config")

public class Config {

@XmlAttribute

private String description;

@XmlElement(name = "server-list")

private ServerConfigList serverConfigList;

public String getDescription() {

return description;

}

public void setDescription(String description) {

this.description = description;

}

public ServerConfigList getServerConfigList() {

return serverConfigList;

}

public void setServerConfigList(ServerConfigList serverConfigList) {

this.serverConfigList = serverConfigList;

}

}

import javax.xml.bind.annotation.XmlAccessType;

import javax.xml.bind.annotation.XmlAccessorType;

import javax.xml.bind.annotation.XmlType;

@XmlAccessorType(XmlAccessType.FIELD)

@XmlType(name = "", propOrder = { "host", "port" })

public class ServerConfig {

private String host;

private int port;

public ServerConfig() {

}

public ServerConfig(String host, int port) {

this.host = host;

this.port = port;

}

public String getHost() {

return host;

}

public void setHost(String host) {

this.host = host;

}

public int getPort() {

return port;

}

public void setPort(int port) {

this.port = port;

}

@Override

public String toString() {

return "ServerConfig [" + host + ":" + port + "]";

}

}

위 Code와 같이 JAXB2 Annotation을 이용한 Class가 존재할 경우, Jaxb2Marshaller Class의 classesToBeBound 프로퍼티를 이용해서 XML과

Mapping될 Class 목록을 지정해주면 된다. 아래 Code는 Jaxb2Marshaller Class의 설정 예이다.

<bean id="jaxb2Marshaller" class="org.springframework.oxm.jaxb.Jaxb2Marshaller">-->

<property name="classesToBeBound">-->

<list>

<value>madvirus.spring.chap11.Config</value>-->

</list>

</property>

</bean>

<bean id="configHandler" class="madvirus.spring.chap11.ConfigHandler">

<property name="marshaller" ref="jaxb2Marshaller" />

<property name="unmarshaller" ref="jaxb2Marshaller" />

</bean>

Jaxb2Marshaller Class는 Marshaller Interface와 Unmarshaller Interface를 모두 구현하고 있기 때문에, 위 Code에서와 같이 Marshaller와

Unmarshaller로 Jaxb2Marshaller 빈을 사용할 수 있다.

Jaxb2Marshaller Class는 classesToBeBound 프로퍼티 뿐만 아니라 아래와 같은 프로퍼티를 설정할 수 있다.

프로퍼티 타입 설 명

=================================================================================

classesToBeBound Class 배열 XML과 바인딩 될 Class 목록

=================================================================================

contextPath String 바인딩 관련 Mapping 정보를 담고 있는 패키지(예, ObjectFactory Class가

위치한 패키지)

=================================================================================

contextPaths String 배열 바운딩 관련 Mapping 정보를 담고 있는 패키지 목록

=================================================================================

jaxbContextProperties Map JAXBContext의 프로퍼티를 설정

=================================================================================

marshallerProperties Map JAXB의 Marshaller 프로퍼티를 설정

=================================================================================

unmarshallerProperties Map JAXB의 Unmarshaller 프로퍼티를 설정

=================================================================================

schema Resource XML 문서를 검증할 때 사용할 스키마를 지정한다.

=================================================================================

mtomEnabled boolean MTOM 가능 여부, 기본 값은 false

=================================================================================

Jaxb2Marshaller를 설정하는 또 다른 방법은 <oxm:jaxb2-marshaller> 태그를 사용하는 것이다. 아래 Code는 <oxm:jaxb2-marshaller> 태그를

사용해서 바인딩 될 Class 목록을 설정하는 예를 보여주고 있다.

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

xmlns:oxm="http://www.springframework.org/schema/oxm"

xmlns:p="http://www.springframework.org/schema/p"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/oxm

http://www.springframework.org/schema/oxm/spring-oxm-3.0.xsd">

<oxm:jaxb2-marshaller id="jaxb2Marshaller">

<oxm:class-to-be-bound name="madvirus.spring.chap11.Config" />

</oxm:jaxb2-marshaller>

...

</beans>

<oxm:class-to-be-bound> 태그를 이용해서 바운딩 될 Class 목록을 지정해 주면된다. contextPath를 설정하고 싶다면 다음과 같이

<oxm:jaxb2-marshaller> 태그의 contextPath 속성을 사용하면 된다.

<oxm:jaxb2-marshaller id="jaxb2Marshaller" contextPath="madvirus.spring.chap11"/>

2.2 Marshaller와 Unmarshaller 사용하기

: Jaxb2Marshaller에 대한 설정을 완료했다면, Marshaller Interface와 Unmarshaller Interface를 이용해서 Java 객체와 XML 사이의 Mapping을 처리하면 된다.

Marshaller Interface의 marshal() 메서드는 javax.xml.transform.Result 객체에 Java 객체를 XML로 변환한 결과를 저장한다.

import javax.xml.transform.Result;

public interface Marshaller{

void marshal(Object graph, Result result) throws IOException, XmlMappingException;

}

따라서, 다음과 같이 알맞은 Result 구현 Class를 이용해서 Java 객체를 XML로 저장할 수 있다. 아래 Code는 앞서 예로 작성했던 Config 객체를 XML 파일로

저장하기 위해 StreamResult를 사용하는 Code의 예를 보여주고 있다.

import javax.xml.transform.stream.StreamResult;

import org.springframework.oxm.Marshaller;

public class ConfigHandler {

private Marshaller marshaller;

private Unmarshaller unmarshaller;

public void setMarshaller(Marshaller marshaller) {

this.marshaller = marshaller;

}

public void setUnmarshaller(Unmarshaller unmarshaller) {

this.unmarshaller = unmarshaller;

}

public void save(Config config, File file) throws XmlMappingException,

IOException {

StreamResult result = new StreamResult(file);

marshaller.marshal(config, result);

}

public Config load(File file) throws XmlMappingException, IOException {

StreamSource source = new StreamSource(file);

return (Config) unmarshaller.unmarshal(source);

}

}

위 Code가 생성하는 파일은 아래와 같다. (실제로 생성되는 파일은 공백과 줄바꿈 없이 한 줄로 생성된다.)

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<config description="설명">

<server-list>

<server>

<host>localhost</host>

<port>1024</port>

</server>

<server>

<host>central</host>

<port>1088</port>

</server>

</server-list>

</config>

javax.xml.transform.Result Interface의 주요 구현 Class는 아래와 같으며, XML 데이터를 어떻게 표현할 지에 따라서 알맞은 구현 Class를 사용하면 된다.

구현 Class 설 명

=============================================================================

javax.xml.transform.stream.StreamResult XML Mapping 결과를 스트림 또는 파일에 출력한다.

=============================================================================

javax.xml.transform.dom.DOMResult XML Mapping 결과를 DOM 트리로 저장한다.

=============================================================================

javax.xml.transform.sax.SAXResult XML Mapping 결과를 SAX2의 ContentHandler를 통해서 전달받는다.

=============================================================================

javax.xml.transform.stax.StAXResult XML Mapping 결과를 StAX의 XMLEventWriter를 통해서 전달받는다.

=============================================================================

Unmarshaller Interface의 unmarshal() 메서드는 javax.xml.transform.Source 객체로부터 XML 데이터를 읽어와 Java 객체로 Mapping한 결과를 리턴한다.

import javax.xml.transform.Source;

public interface Unmarshaller{

void unmarshal(Object graph, Source source) throws IOException, XmlMappingException;

}

따라서, 다음과 같이 알맞은 Source 구현 Class를 이용해서 XML 데이터를 Java 객체로 Mapping할 수 있다. 아래 Code는 XML 파일로부터 앞서 작성했던

Config 객체를 생성해주기 위해 StreamSource를 사용하는 Code의 예를 보여주고 있다.

import javax.xml.transform.Source;

import org.springframework.oxm.Unmarshaller;

public class ConfigHandler {

private Unmarshaller unmarshaller;

public void setUnmarshaller(Unmarshaller unmarshaller) {

this.unmarshaller = unmarshaller;

}

public Config load(File file) throws XmlMappingException, IOException {

StreamSource source = new StreamSource(file);

return (Config) unmarshaller.unmarshal(source);

}

}

Source Interface의 주요 구현 Class는 다음과 같다.

Class 설 명

===============================================================================

javax.xml.transform.stream.StreamSource 스트림 또는 파일로 부터 XML 데이터를 가져온다.

===============================================================================

javax.xml.transform.dom.DOMSource DOM 트리로부터 XML 데이터를 가져온다.

===============================================================================

javax.xml.transform.sax.SAXSource InputSource로부터 XML 데이터를 가져온다.

===============================================================================

javax.xml.transform. stax.StAXSource XMLStreamReader나 XMLEventReader로부터 XML 데이터를 가져온다.

===============================================================================