JEUS Client Application
안내서

Document info

Document name: JEUS Client Application 안내서

Document date: 2005-06-06

Manual release version: 3

Software Version: JEUS 5

차례

그림 목차

매뉴얼에 대해서

매뉴얼의 대상

본 문서에는 JEUS 에서의 Client Application 의 개념과 사용법 그리고 JNLP 의 개념, CAS, J2COM 에 대한 내용을 담고 있다.

클라이언트 어플리케이션이란 다음과 같은 실행 가능한 형태의 프로그램을 말한다.

- Java application client
- Applet
- JNLP client
- CAS 를 이용해서 JEUS EJB 를 호출하는 COM client

매뉴얼의 전제 조건

본 문서를 읽기 전에 JEUS Server 안내서와 JEUS EJB 안내서를 읽기 바란다.

그리고 J2EE Application client, Applet, JNLP, COM application 에 대해서 잘 알고 있어야 한다.

매뉴얼의 구성

JEUS 매뉴얼 문서들처럼 본 매뉴얼도 다음과 같이 구성되어 있다.

1. 각 장 마지막에 다음 장에서 다룰 내용에 대해 설명했다.
2. 각 장은 각각 독립적으로 구성되어 있어 필요한 장만 골라 볼 수 있다.
3. 부록 부분만을 레퍼런스로 사용할 수 있다. 본 매뉴얼을 모두 이해한 후에는 이렇게 레퍼런스로 사용하길 권한다.

JEUS Client Application 안내서는 7 개의 장으로 나누어져 있다. 아래 참조

1. 소개
2. JEUS Application Client
3. JEUS Applet Client
4. JEUS JNLP
5. JEUS CAS
6. JEUS-COM Connector (J2COM)
7. 결론

부록에는 2 개의 JEUS 설정 파일에 대한 설명이 2 개의 부록으로 구성 되어 있다.

A jeus-client-dd: JEUS Application Client Deployment Descriptor 레퍼런스

B JNLPMain.xml: JNLP Resource 설정 파일 레퍼런스

관련된 문서들

다음은 본 매뉴얼과 관련된 문서들이다.

- JEUS Server 안내서
- JEUS EJB 안내서
- The J2EE 1.4 Specification
- The JNLP Specification
- The Java WebStart Secification
- JEUS Web Container 안내서

일러두기

표기 예	내용
텍스트	본문, 12 포인트, 바탕체 Times New Roman
텍스트	본문 강조
CTRL+ C	CTRL 와 동시에 C 를 누름
`public class myClass { }`	Java 코드
`<system-config>`	XML 문서
참조 / 주의	참조 사항과 주의할 사항
Configuration 메뉴를 연다	GUI 의 버튼 같은 컴포넌트
JEUS_HOME	JEUS 가 실제로 설치된 디렉토리 예)c:\jeus
`jeusadmin nodename`	콘솔 명령어와 문법
[파 라 미 터]	옵션 파라미터
< xyz >	'<'와 '>' 사이의 내용이 실제 값으로 변경됨. 예)<node name>은 실제 hostname 으로 변경해서 사용
|	선택 사항. 예) A|B: A 나 B 중 하나
…	파라미터 등이 반복되어서 나옴
?, +, *	보통 XML 문서에 각각 "없거나, 한 번", "한 번 이상", "없거나, 여러 번"을

표기 예	내용
	나타낸다.
. . .	XML 이나 코드등의 생략
<<*FileName.ext*>>	코드의 파일명
그림 1.	그림 이름이나 표 이름

OS 에 대해서

본 매뉴얼에서는 모든 예제와 환경 설정을 Microsoft Windows™의 스타일을 따랐다. 유닉스같이 다른 환경에서 작업하는 사람은 몇 가지 사항만 고려하면 별무리 없이 사용할 수 있다. 대표적인 것이 디렉토리의 구분자인데, Windows 스타일인 "\"를 유닉스 스타일인 "/"로 바꿔서 사용하면 무리가 없다. 이외에 환경 변수도 유닉스 스타일로 변경해서 사용하면 된다.

그러나 Java 표준을 고려해서 문서를 작성했기 때문에, 대부분의 내용은 동일하게 적용된다.

용어 설명

다음에 나오는 용어들은 이 매뉴얼에서 주로 사용되는 것들이다. 이해하지 못하거나 의미가 불분명한 것들은 이 리스트에서 제공된 설명을 통하여 이해하도록 하기 바란다.

용어	설명
CAS	J2EE™ CAS (Client Access Services)는 COM Application 이 J2EE EJB 와 로컬에 있는 Java 라이브러리를 액세스할 수 있도록 해준다.
Com Bridge	CAS 를 이용하기 위해서 Sun 에서 제공하는 소프트웨어.

Java Web Start	클라이언트가 JNLP 리소스를 액세스하거나 다운받을 수 있도록, Sun 에서 제공하는 소프트웨어.
JEUS-COM connector	Java/JEUS application 과 COM application 을 연결하는 브릿지.
JNLP	JNLP 는 Java Network Launching Protocol 을 나타내며, 소프트웨어 컴포넌트를 배포하는 프로토콜이다.

연락처

Korea
Tmax Soft Co., Ltd
18F Glass Tower, 946-1, Daechi-Dong, Kangnam-Gu
Seoul 135-708
South Korea
Email: info@tmax.co.kr
Web (Korean): http://www.tmax.co.kr

USA
Tmax Soft, Inc.
560 Sylvan Ave, Englewood Cliffs NJ 07632
USA
Email: info@tmaxsoft.com
Web (English): http://www.tmaxsoft.com

Japan
Tmax Soft Japan Co., Ltd.
6-7 Sanbancho, Chiyoda-ku, Tokyo 102-0075
Japan
Email: info@tmaxsoft.co.jp
Web (Japanese): http://www.tmaxsoft.co.jp

China
Beijing Silver Tower, RM 1507, 2# North Rd Dong San Huan,
Chaoyang District, Beijing, China, 100027
E-mail : info@tmaxchina.com.cn
Web (Chinese): http://www.tmaxchina.com.cn

1 소개

이 매뉴얼은 표준 클라이언트 모듈과 Applet, JNLP 모듈, CAS 모듈을 JEUS 에서 사용하기 위한 내용을 설명한다. JEUS 에 접속하는 서비스를 이용하는 클라이언트는 대부분 웹 브라우저에서 호출하는 JSP 이다. 그러나 이에 대한 내용은 너무 일반적인 내용이라서 여기서는 다루지 않는다. 이 매뉴얼에서 다루고자 하는 주제는 다음과 같다.

- 클라이언트가 EJB 에 직접 접근하여 서비스를 제공받기 위해서는 JEUS 어플리케이션을 사용한다. 이에 대한 설명은 2 장에서 살펴 보기 바란다.
- 그래픽 환경의 클라이언트가 오랜시간동안 동작한다면, Applet 을 사용한다. 이에 자세한 사항은 3 장에서 설명한다.
- 어플리케이션의 사이즈가 크고 버전이 자주 변경된다면 JNLP 를 사용한다. 이에 대한 자세한 사항은 4 장에서 설명한다.
- VC++, ASP, Visual Basic 과 같은 어플리케이션이 JEUS 로 접속한다면 JEUS CAS 를 사용한다. 이에 대한 자세한 설명은 5 장에서 설명한다.

2 JEUS Application Client

2.1 소개

JEUS 어플리케이션 클라이언트란 JEUS 전용의 XML(Deployment Descriptor)을 사용해서 JEUS 에서 제공하는 환경으로 Deploy 되는 클라이언트를 일컫는다. JEUS 어플리케이션 클라이언트는 Client Container 를 사용해서 Naming Service, Scheduler, Security 등과 같은 JEUS 서비스를 모두 사용할 수 있다.

2.2 JEUS Application Client 의 개요

2.2.1 소개

이 절에서는 JEUS Client-Server 환경의 아키텍쳐에 대해 알아보고, 간단한 샘플 코드를 테스트해 보기로 한다.

2.2.2 아키텍쳐

어플리케이션 클라이언트는 JVM 에서 실행되는 클라이언트 프로그램이다. 어플리케이션 클라이언트는 main() 메소드를 호출해서 실행하고, 가상 머신이 종료되면 실행을 마친다. 다른 J2EE 어플리케이션 컴포넌트처럼 어플리케이션 클라이언트는 시스템 서비스를 제공하는 Client Container 위에서 동작한다. Client Container 는 다른 J2EE Container 에 비해서 매우 작은 양의 시스템 리소스를 사용한다 [그림 1].

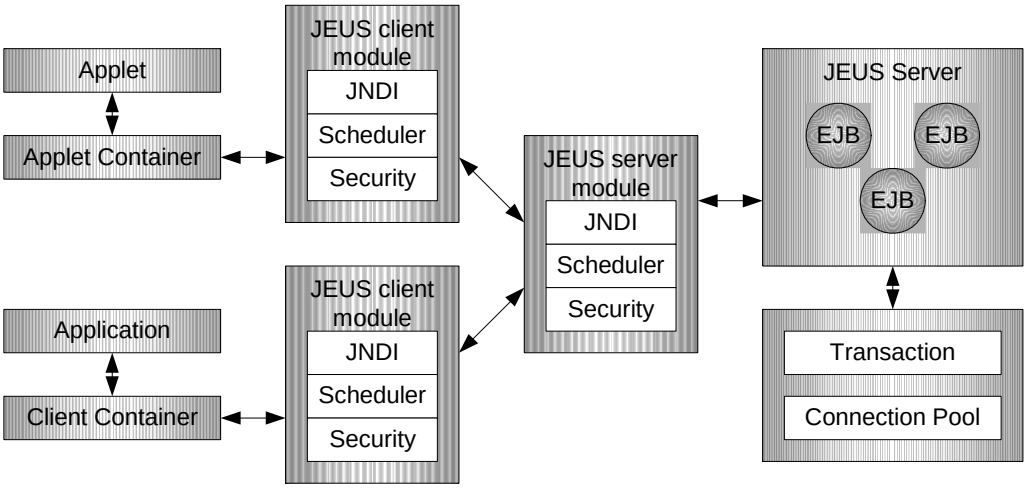

그림 1. Application/Applet Client 아키텍처

Client Container 는 일반 클라이언트 어플리케이션에서 JEUS 의 서비스를 이용할 수 있는 환경을 제공한다.

Client Container 는 JEUS Client Module 을 이용해서 JEUS Server Module 과 통신을 해서 JEUS 의 여러가지 서비스(JNDI, Scheduler, Security)를 이용한다. 그리고 JNDI 서비스를 통해서 JEUS 에 바인딩이 되어 있는 컴포넌트(EJB) 및 System Resource 들(JDBC DataSource, JMS Connection Factory 등)을 이용할 수 있다. 이처럼 일반적인 클라이언트 어플리케이션을 Client Container 위에서 실행함으로써 JEUS 에서 제공하는 여러가지 서비스를 손쉽게 사용할 수 있다.

2.2.3 어플리케이션 클라이언트 코드

Client application 은 일반적인 Java 프로그램과 마찬가지로 반드시 public 으로 선언된 main 메소드를 가지고 있어야 한다. 아래의 간단한 예제는 “Salary”라는 EJB 를 lookup 하는 코드이다.

<<SalaryClient.java>>

```
package SimpleBean;

import javax.ejb.*;
import SimpleBean.*;
import javax.naming.InitialContext;
import java.sql.*;


public class SalaryClient
```

```
{
    public static void main(String[] args)
    {
        try
        {
            InitialContext ctx = new InitialContext();
            SalaryHome home = (SalaryHome)ctx.lookup("Salary");
            Salary bean = home.create();

            System.out.println("Monthly net salary: " +
               bean.calculateSalary(30000, 28, 31000));
        }
        catch (javax.naming.NamingException ne)
        {
            System.out.println("Naming Exception caught: " + ne);
        }
        catch (javax.ejb.CreateException ce)
        {
            System.out.println("Create Exception caught: " + ce);
        }
        catch (java.rmi.RemoteException re)
        {
            System.out.println("Remote Exception caught: " + re);
        }
    }
}
```

2.2.4 J2EE Deployment Descriptor

J2EE 스펙에서는 Client Module 에 대한 Deployment Descriptor 를 정의하고 있으므로, 모든 Vendor 에서 사용할 수 있다. 자세한 사항은 J2EE 스펙을 참고하기 바란다.

2.2.5 JEUS Deployment Descriptor

서버와 클라이언트 어플리케이션이 통신할 때 client 모듈에 대한 정보가 필요한데, Deployment Descriptor(이하 DD 파일)는 이러한 정보들을 가지고 있는 XML 문서이다. 디스크립터를 이용하여 각각의 클라이언트 어플리케이션을 Client Container 에 배치할 때에 어떠한 서비스를 사용하게 할 지 결정할 수 있으며, 실행 시에도 별도의 수정 없이 디스크립터만 수정하여 해당하는 클라이언트 어플리케이션에게 적용할 수 있다.

다음은 SalaryClient client 의 모듈의 디스크립터 XML 문서이다.

<<*jeus-client-dd.xml*>>

```
<?xml version="1.0"?>
<jeus-client-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <module-info>
        <module-name>client_module</module-name>
        <app-main-class>SimpleBean.SalaryClient</app-main-class>
    </module-info>
</jeus-client-dd>
```

참고: 자세한 설명은 부록 A 를 참고하기 바란다.

2.2.6 결론

이번 장에서는 JEUS 에서 어플리케이션 클라이언트 프로그램이 어떻게 사용되는지 그리고 Client Container 의 아키텍쳐에 대해서 알아 보았다. 다음 장에서는 클라이언트 모듈에 대한 package, deploy 의 방법에 대해서 알아보도록 하겠다.

2.3 Application Client Module 의 Packaging

2.3.1 소개

패키징은 다음 방법 중 하나를 사용한다.

1. 사용자의 컴퓨터상에 설치되어 있는 텍스트 에디터나 XML 에디터를 사용해서 Deployment Descriptor XML 파일을 생성하고, 필요한 파일을 모아서 Java 에서 제공하는 jar 유틸리티를 이용하여 클라이언트 모듈에 대한 jar 파일을 생성한다.

2. 디스크립터, 패키지와 관련된 파일들을 JEUS Builder Tool 을 이용하여 생성(수정)한다. JEUS Builder Tool 을 사용하는 방법은 JEUS Builder 안내서를 참조하기 바란다.

주의: 만약 JEUS Client deployment descriptor 에 대해 잘 알지 못한다면 JEUS Builder Tool 을 이용할 것을 권장한다.

2.3.2 수동으로 module 패키징

1. 클라이언트 Application 을 패키징하기 위해서는 Application 을 구성하고 있는 class 파일과 application-client.xml, jeus-client-dd.xml 파일이 포함되어 있어야 한다.

2. console 에서는 jar 명령어를 이용하여 클라이언트 모듈에 대한 jar 파일을 생성한다.

예:

```
jar cvf product-client.jar *.*
```

2.3.3 결론

패키징 과정이 끝났으면 다음 절에서는 클라이언트 모듈을 deploy 하는 과정에 대해서 설명하도록 하겠다.

2.4 Application Client Module 의 Deploy

2.4.1 소개

Application 모듈은 "수동"으로 직접 deploy 를 진행하거나 JEUS 웹 관리자를 통해서 이루어지게 되며 모듈 안에는 J2EE 표준 디스크립터 파일인 application-client.xml 과 JEUS 에서 제공하는 jeus-client-dd.xml 이 생성 된다. 이번장에서는 수동으로 직접 deploy 를 하는 과정을 설명하게 되며, 웹 관리자를 이용하는 방법에 대해서는 JEUS 웹 관리자 안내서를 참조하기 바란다.

2.4.2 JEUS Deployment Descriptor 의 생성

JEUS 에서 클라이언트 모듈을 deploy 하기 전에 jeus-client-dd.xml 파일을 작성해야 한다. 이 파일은 JEUS Builder 에서 모듈을 Packaging 하는 단계에서 생성된다. 물론 텍스트 에디터나 XML 에디터를 이용하여 직접 작성할 수도 있다. 이 파일의 각각의 element 에 대한 설명은 부록 A 를 참고하기 바란다.

XML 파일의 예를 들면 다음과 같다.

<<jeus-client-dd.xml>>

```
<jeus-client-dd>
    <module-info>
        <module-name>client_application1</module-name>
        <app-main-class>simpleBean.SalaryClient</app-main-class>
        <app-argument>125000</app-argument>
    </module-info>
    <env>
        <name>year</name>
        <type>java.lang.Integer</type>
        <value>2002</value>
```

```
    </env>
    <ejb-ref>
        <jndi-info>
            <ref-name>count</ref-name>
            <export-name>count_bean</export-name>
        </jndi-info>
    </ejb-ref>
    <res-ref>
        <jndi-info>
            <ref-name>datasource</ref-name>
            <export-name>Oracle_DataSource</export-name>
        </jndi-info>
    </res-ref>
    <res-env-ref>
        <jndi-info>
            <ref-name>jms/SalaryInfo</ref-name>
            <export-name>jms/salary_info_queue1</export-name>
        </jndi-info>
    </res-env-ref>
</jeus-client-dd>
```

2.4.3 Application Client Module 의 Deploy

Application Client 에 대한 모듈파일을 생성한 후 해당 파일을 JEUS_HOME\webhome\client_home\client container.jar 파일을 참조할 수 있는 작업디렉토에 가져다 놓는다. 이것으로써 Client 모듈에 대한 Deploy 작업이 완료된다.

2.4.4 결론

위의 예제에서는 단순하게 “수동”으로 deploy 하는 방법만 보여주었다. JEUS Builder Tool 을 사용하면 좀 더 자세하고 세밀한 설정들을 할 수 있다. JEUS Builder Tool 을 사용해보면서 다른 옵션들도 사용해 보길 권장한다.

2.5 Application Client Module 의 실행

2.5.1 Console 에서의 사용

runclient.cmd 스크립트는 client application 모듈을 실행하기 위해 제공되며, 일부 내용을 수정함으로써 모듈을 실행 시킬 수 있다.

1. EJB 모듈 디플로이시 생성되는 스텁/스켈레톤 클래스들을 client.jar 로 묶어서 runclient.cmd 스크립트의 APP_CLASSPATH 에 추가한다. (EJB 모듈을 디플로이 할 때 생성되는 스텁/스켈레톤 클래스는 %JEUS_HOME% \webhome\<node name>_container1 아래에 생성 된다)

2. 커멘드 라인에서 runclient.cmd 스크립트를 실행 시키면 화면에 다음과 같은 결과를 출력한다.

   예)

   C:\>runclient –client c:\SalaryClient.jar -jar

   [2005.03.02 15:57:43][2] [Client-0028] client container is started

   [2005.03.02 15:57:43][2] [Client-0038] starting client application [simpleBean.SalaryClient:]

   [SalaryClient] SalaryEJB is retrieved ..

   [SalaryClient] Monthly net salary: 280002500

참고 1: client application 이 정상적으로 실행되기 위해서 Salary EJB 가 디플로이 되어있어야 하며, Salary EJB 가 사용하는 테이블이 생성되어 있어야 한다.

참고 2: samples 디렉토리에 ant 를 위한 build.xml 파일을 제공되며, Salary EJB 모듈을 생성, deploy 하고 application client 모듈을 실행 할 수 있다.

2.5.2 결론

client application 은 기본적으로 main 메소드가 호출되면서 실행되며, client container 가 classpath 에 있어야 한다. 여기서는 간단한 Client application 에 대해서 알아보았지만 복잡한 Client application 도 위와 동일한 과정을 따른다.

2.6 결론

어플리케이션 클라이언트 모듈은 JEUS 안에 포함된 클라이언트의 한 형태이다. 이 형태는 Client/Server 시스템이나 Test/Debuging 등에 널리 사용되고 있는 형태이다. J2EE 기반의 Client application 에 대해서 자세히 알기를 원할 때는 J2EE Spec 을 참조하길 바란다. 부록 A 에는 JEUS XML Schema 에 대한 설명이 있으니 참고하기 바란다.

3 JEUS Applet Client

3.1 소개

Applet 은 그래픽 기능이 막강하여 각광 받고 있다. 만약 어플리케이션이 그래픽을 처리하고, 오랫동안 동작하는 프로그램이라면, Applet 을 사용하는 것이 좋다. 예를 들면, 주식 시세 프로그램을 들 수 있다. 이 프로그램은 많은 그래픽을 가지고 있으며, 일반적으로 한 번 실행시키면 쉽게 종료하지 않고 오랫동안 사용하게 된다. 만약 이런 어플리케이션이 아니면 Applet 을 권장하지 않는다. 왜냐하면 로딩과 실행에서 상대적으로 많은 시간과 시스템 리소스가 소요되기 때문이다.

이번 장에서는 JEUS 에서 Applet 프로그램을 어떻게 작성하고 설정 및 실행하는지 알아본다.

3.2 JEUS Applet Client 의 개요

3.2.1 소개

이번 절에서는 사용자가 참고할 수 있는 샘플 예제를 설명한다. 샘플 예제는 JEUS_HOME\samples\clientContainer\에 있다.

3.2.2 Applet Client 코드

Applet 어플리케이션은 Applet 클래스를 상속받고, start() 메소드를 구현해야 한다. 다음은 Applet 의 예제 코드이다. 컴파일해서 사용해 보길 바란다.

<<*SalaryAppletClient.java*>>

```
package SimpleBean;

import javax.ejb.*;
import java.sql.*;
import javax.sql.*;
import java.util.*;
import java.applet.*;
import javax.naming.*;
```

```
public class SalaryAppletClient extends Applet
{
    public void start()
    {
        try
        {
            InitialContext ctx = new InitialContext();

            SalaryHome home = (SalaryHome)ctx.lookup("Salary");
            Salary bean = home.create();

            System.out.println("Monthly net salary: " +
                bean.calculateSalary(28000, 2, 500));
        } catch (javax.naming.NamingException ne) {
            System.out.println("Naming Exception caught: " + ne);
        } catch (javax.ejb.CreateException ce) {
            System.out.println("Create Exception caught: " + ce);
        } catch (java.rmi.RemoteException re) {
            System.out.println("Remote Exception caught: " + re);
        }
    }
}
```

3.2.3 HTML 코드

<<Salary.html>>

```
<html>
<body>
<applet codebase = "."
archive="jnlpcontainer.jar,simple.jar,simpleBean.jar,simpleClient
.jar,simpleClient2.jar" code = "SimpleBean.SalaryAppletClient"
width = 300 height = 300>
</applet>
</body>
</html>
```

참고: 좋은 성능을 원한다면 JDK 의 htmlconverter 를 이용하여 HTML 코드를 전환하기 바란다. 자세한 내용은 다음 사이트를 참고 하기 바란다.

http://java.sun.com/products/plugin/1.3/docs/htmlconv_01.html#using_command

3.2.4 결과

다음 절 에서는 Applet 의 Deploy 및 실행에 대해서 알아 보도록 한다.

3.3 Applet Client 의 deploy

3.3.1 소개

Applet 은 다른 어플리케이션의 deploy 와 다르게 시스템의 환경에 맞게 설정을 해줘야 한다.

3.3.2 Servlet Context 생성

웹 브라우저에서 HTML 페이지를 보려면, Servlet Context 에 해당 HTML 페이지가 포함 되어 있어야 한다.

예:

default Context 디렉토리에 "applet"이라는 폴더를 만든다. 그리고 http://localhost:8080/applet/ 에 접속한다.

3.3.3 Servlet Context 에 파일 설치하기

우선 다음 파일이 필요하다.

- <applet> 태그를 가지고 있는 HTML 파일
- Applet 에서 EJB 를 사용할 경우 EJB 의 Home 인터페이스와 Remote 인터페이스 파일
- Applet 클래스
- jnlpcontainer.jar (CLEINT_HOME 디렉토리에 있음)

참고 1: Home 과 Remote 인터페이스는 반드시 applet 의 codebase 아래에 위치하고 있어야 한다.

참고 2: JEUS base port 인 "9736"이 아닌 다른 base port 를 사용하고자 한다면, JAR 파일안에 있는 META-INF 디렉토리에 "jeus.properties" 파일을 만들어 추가 한다 (이처럼 JEUS base port 을 다르게 사용하고자 한다면, 반드시 JAR 파일 안에 "jeus.properties"를 추가 해야 하며 default 인 "9736"을 사용시에는 위의 과정을 생략할 수 있다).

```
jeus.baseport=<non-default JEUS base port>
```

예:

```
jeus.baseport=9222
```

.

3.3.4 결론

Applet 클라이언트는 특별한 Deploy 과정을 가졌다. 왜냐하면 Applet 은 J2EE 표준이 아니기 때문이고, 광범위하게 사용되지 않기 때문이다. 만약 Applet 이 널리 사용되어 J2EE 표준에 추가된다면, 보다 간편하게 Deploy 할 수 있게 될 것으로 본다.

3.4 Applet Client 의 실행

3.4.1 소개

Applet 의 실행은 매우 간단하다. Appletviewer 또는 web browser 에서 URL 을 통해 접근 할 수 있다. Web browser 에서 테스트 하는 것이 좀더 쉽지만 appletviewer 는 에러가 발생할 경우 자세한 에러 메시지를 보여준다.

URL 예제는 다음과 같다. http://localhost:8080/applet/salary.html.

3.4.2 web browser 에서의 실행

Web browser 는 HTML 페이지안에 있는 applet 태그로 접속한다.

3.4.3 appletviewer 의 실행

appletviewer 는 JDK 를 통해서 접속되며, JDK 디렉토리안에 있는 bin 디렉토리의 실행파일로 실행을 할 수 있다.

예 :

```
appletviewer  -J -Djava.naming.factory.initial=jeus.jndi.
JEUSContextFactory Salary.html
```

3.4.4 결론

Applet 을 실행 하는데 시간이 다소 걸린다. Applet 의 사이즈가 너무 크면 사용하기가 불편하므로 이런 점을 고려해서 사용하기 바란다.

3.5 결론

Applet 은 흥미로운 기능이 장점일 수 도 있지만, 반대로 불편 할 수도 있다. 이 기능을 이용 하려면 여러가지 상황과 제반환경을 주의깊게 생각하고 사용하기 바란다.

4 JEUS JNLP

4.1 소개

JNLP 는 Java Network Launching Protocol 의 약자로 소프트웨어 컴포넌트의 배포 및 실행에 관한 프로토콜이다. JNLP 를 통해서 어플리케이션이 J2EE 서버로 접속할 수 있는 환경이 자동적으로 구성된다.

JNLP 는 클라이언트 머신에 별다른 설정이나 설치를 할 필요가 없다. 프로세스가 쉬는 동안에 필요한 리소스를 다운받아 사용한다. 다운받을 때는 네트웍 환경에 따라서 시간이 걸릴 수 있다. 초기화 작업 후에는 다운로드한 리소스를 사용하는데, 웹 브라우저를 사용해서 어플리케이션을 실행하는 것보다 더 빠르다.

대표적인 예로, 오늘날 일상적으로 사용되는 Web 메일을 들 수 있다. 메일 계정이 적은 경우에는 JNLP 를 사용하는 것이 느릴지 모르나, 메일 계정이 증가할수록 상대적으로 빨라진다. 왜냐하면 유휴 시간에 필요한 리소스를 모두 다운받아서 사용하기 때문이다.

참고: 자세한 내용을 알고 싶다면 다음 URL 을 참고 하기 바란다. http://java.sun.com/products/javawebstart/

주의: 이 매뉴얼에서는 JNLP 에 대한 상세한 정보를 제공하지 않는다. 그러므로, 이번 장을 읽기 전에 JNLP 에 대해서 어느 정도 알아두길 바란다.

4.2 JEUS JNLP 의 개요

4.2.1 소개

클라이언트에 JNLP 서비스를 사용하기 위해서는 몇가지 절차가 필요하다. 다음 내용을 주의 깊게 보기 바란다.

4.2.2 JNLPMain.xml

JNLPMain.xml 은 JNLP 서비스를 사용하는데 중요한 설정 파일이다. 이 파일은 client 에서 리소스를 얻고자 할 때 제공해 주는 리소스가 등록되어 있다. 이 파일은 실제로 리소스를 등록하는 곳이다. JEUS 는 이 파일에 등록되지

않는 리소스는 접근 할 수 없다. 이 XML 파일에 대한 내용은 부록 B.4 를 참고 하기 바란다.

4.2.3 JNLP 디렉토리 구조

- **JEUS_HOME\config\<Node-Name>\JNLPServer**: JNLPMain.xml 파일이 있다.
- **CLIENT_HOME**: JNLP 파일과 리소스 파일들이 있다.

4.2.4 결론

JEUS JNLP 서비스는 Sun 의 JNLP 스펙을 준수한다. 사용자가 설정할 것은 JNLPMain.xml 뿐이다. jar 파일이나 JNLP 와 관련된 파일을 JNLP 서비스에 반드시 등록해야 한다. 자세한 내용은 다음 절에서 설명하도록 하겠다.

4.3 JEUS JNLP 서비스 활성화

4.3.1 소개

JEUS 에서 JNLP 서비스는 기본적으로 설정이 되어 있지 않다. 그러므로 서비스를 사용하려면 활성화시켜 줘야만 한다. 간단히 JEUSMain.xml 에 태그를 추가시켜주는 것으로 활성화된다.

4.3.2 JEUSMain.xml 에서 서비스 활성화하기

JNLP 서비스를 실행하기 위해서는 JNLP 를 활성화해야 한다. 아래 예제는 JEUSMain.xml 에서 JNLP 서비스를 어떻게 활성화하는지 보여준다. 파일을 수정한 후 JEUS 를 다시 시작시킨다.

<<*JEUSMain.xml*>>

```
<jeus-system>
    <node>
        . . .
        <enable-jnlp>true</enable-jnlp>
        . . .
    </node>
    . . .
</jeus-system>
```

참고: JNLP 서비스는 기본적으로 9744(JEUS_BASEPORT+8)이다.

4.3.3 결론

위의 예제 처럼 JEUSMain.xml 안에 JNLP 태그를 추가시켜 주는 것으로 JNLP 서비스가 활성화된다. JNLP 를 사용하지 않으려면 태그 전체를 지우거나 “false”로 설정한다.

4.4 JNLP Application 의 생성

4.4.1 소개

JNLP 어플리케이션이란 JNLP 서비스가 호출한 어플리케이션을 말한다. 어플리케이션이 설치 되어있다면 이번 절은 무시해도 된다. 그렇지 않을 경우는 샘플 어플리케이션 코드를 사용하기 바란다.

4.4.2 JNLP Application 코드의 예제

<<HelloJeus.java>>

```
import javax.swing.*;
import java.awt.Font;
import java.awt.BorderLayout;
import java.awt.event.*;
public class HelloJeus
{
    public static void main(String[] args)
    {
        new HelloJeusFrame();
    }
}
class HelloJeusFrame extends JFrame
{
    JLabel label = new JLabel(“Hello Jeus”);
    public HelloJeusFrame()
    {
        label.setFont( new Font(  “Helevetica” , Font.BOLD,70));
        getContentPane().setLayout( new BorderLayout() );
        getContentPane().add( label , BorderLayout.CENTER);
        setSize(500,250);
        setVisible( true );
    }
}
```

참고: 위의 코드를 컴파일해서, 클래스를 HelloJeus.jar 파일로 압축한다.

예:

```
jar cvf HelloJeus.jar *.*
```

4.4.3 결론

JNLP 서비스에 사용할 어플리케이션을 만들어 보았다. 다음 절에서는 JNLP 파일을 생성하는 방법에 대해 설명하도록 하겠다.

4.5 JNLP 설정 파일 생성

4.5.1 소개

JNLP 파일은 JNLP 어플리케이션과 리소스에 대한 정보를 가지고 있는 XML 파일이다. 아래는 JNLP 예제 어플리케이션을 위한 JNLP 예제 파일이다. 만약 JNLP 어플리케이션을 가지고 있다면, JNLP 파일을 작성해야 한다.

4.5.2 JNLP 파일 예제

<<*HelloJeus.jnlp*>>

```
<?xml version="1.0"?>
<jnlp codebase="http://localhost:9744/jnlp/HelloJeus"
href="HelloJeus.jnlp" spec="1.0+">
    <information>
        <title>HelloJeus Test</title>
        <vendor>Tmax</vendor>
        <homepage href="http://www.tmax.co.kr"/>
        <description kind="short">Hello Test</description>
    </information>
    <resources>
        <j2se version="1.4+"/>
        <jar href="HelloJeus.jar" main="true"/>
    </resources>
    <application-desc main-class="HelloJeus"/>
</jnlp>
```

HelloJeus.jnlp 이라는 이름으로 저장한다.

4.5.3 결론

JNLP 어플리케이션을 위한 JNLP 설정파일을 만들었다. 이것으로 JNLP 서비스를 등록할 준비는 완료된 것이다. 다음 절에서는 JNLP 서비스의 등록 절차에 대해서 설명하도록 하겠다.

4.6 JNLP Application 등록

4.6.1 소개

JNLP 어플리케이션을 등록하기 위해서는 JEUS_HOME\config\<node name>\JNLPServer\JNLPMain.xml 파일을 수정해야 한다. JEUS Webadmin 이나 텍스트 에디터, 또는 XML 에디터에서 JNLPMain.xml 을 수정할 수 있다. 이 방법에 대해서 알아보자.

주의: JNLPMain.xml 에 대해 이해가 안되면, JEUS Webadmin 을 사용하여 수정할 것을 추천한다. JEUS Webadmin 에 대한 자세한 설명은 JEUS 웹관리자 안내서를 참조하기 바란다. JNLPMain.xml 의 Schema 에 대한 자세한 설명은 부록 B JNLPMain.xml: JNLP Resource 설정 파일 레퍼런스를 참고하기 바란다.

4.6.2 client_home 폴더에 resource 복사

리소스(JAR 파일과 JNLP 파일)를 CLIENT_HOME\<package path> 디렉토리에 복사한다. 여기서 CLIENT_HOME 은 JEUS_HOME\webhome\client_home 을 나타내는 환경 변수이다.

예:

예제인 HelloJeus 를 사용하기 위해서는 CLIENT_HOME\HelloJeus 폴더를 만들고, 그 폴더에 JNLPServer 디렉토리에 있는 필요한 리소스를 복사한다.

4.6.3 수동으로 어플리케이션 등록하기

예제 어플리케이션을 등록하려면 아래 예제 처럼 JNLPMain.xml 에 두 개의 <jnlp-resource> 태그를 설정을 해야 한다.

<<*JNLPMain.xml*>>

```
<?xml version="1.0" encoding="UTF-8"?>
<jeus-system xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
<jnlp-resource-config>
    ...
    <jnlp-resource>
```

```
        <rsc-name>HelloJeus/HelloJeus.jar</rsc-name>
        <os-name/>
        <arch/>
        <locale/>
        <version/>
        <rsc-path>HelloJeus/HelloJeus.jar</rsc-path>
    </jnlp-resource>
    <jnlp-resource>
        <rsc-name>HelloJeus/HelloJeus.jnlp</rsc-name>
        <os-name/>
        <arch/>
        <locale/>
        <version/>
        <rsc-path>HelloJeus/HelloJeus.jnlp</rsc-path>
    </jnlp-resource>
    ...
</jnlp-resource-config>
```

태그 및 XML Schema 에 대한 설명은 부록 A jeus-client-dd: JEUS Application Client Deployment Descriptor 레퍼런스를 참고 하기 바란다.

4.7 JNLP Client (Java Web Start) 설치

4.7.1 소개

클라이언트 머신에는 Java Web Start 라는 JNLP 클라이언트가 설치 되어 있어야 JNLP 서비스를 가져 올 수 있다. JDK1.4 이상에서는 기본적으로 Java Web Start 가 포함되어 있으나, 이전 JDK 에는 없으므로 다운로드 받아서 설치 해야 한다.

4.7.2 Java Web Start 설치

Java Web Start 는 다음의 URL http://java.sun.com/products/javawebstart/에서 다운 받아 설치하면 된다.

Java Web Start 에서 제공하는 설명서를 참고 하기 바란다.

4.7.3 결론

JEUS 에 접속 하여 JNLP 서비스의 모든 리소스를 다운로드 받기 위해서 JNLP Client 를 설치 해야한다. 다음 절에서는 JNLP 서비스의 실행에 대해서 설명하도록 하겠다.

4.8 JNLP Service 의 실행

4.8.1 소개

이번 장에서는 JNLP 서비스를 실행해 본다. 우선 앞에서 다루었던 과정을 완료하기 바란다.

JNLP 를 실행하려면 웹 브라우저에서 JEUS 의 JNLP 파일을 액세스하면 된다. 그러면 Java Web Start 가 자동으로 실행되면서 서비스를 실행시킨다. 이에 대해서 자세히 알아보자.

JNLP 서비스의 실행은 Java Web Start 의 호출과 web browser 로 JEUS 의 JNLP 파일을 접근하는것이다. 다음 절에서 좀 더 자세히 설명하도록 하겠다.

4.8.2 Web Browser 을 이용한 실행

1. web browser 를 열어서 JNLP 파일이 위치한 URL 을 기입한다. URL 은 다음과 같이 설정한다.

   http://localhost:9744/jnlp/HelloJeus/HelloJeus.jnlp

2. Java Web Start 는 아래 그림과 같이 시작되어서 로딩을 한다[그림 2].

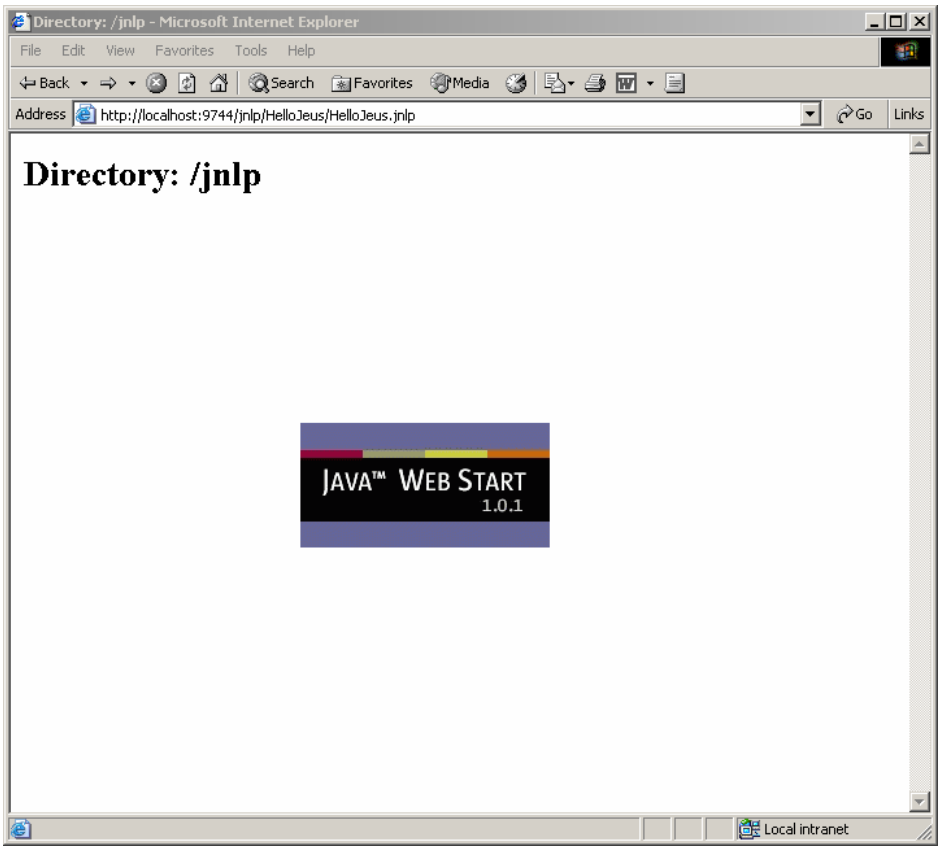

그림 2. browser 에서 Web Start 실행.

3. Java Web Start 는 실행시 파일이 필요하며 자동적으로 다운로드 받는다.

4. 다음은 “Hello Jeus” 의 결과 화면이다[그림 3].

그림 3. JNLP- 실행 화면

4.8.3 결론

JNLP 어플리케이션은 Web 기반 어플리케이션처럼 사용하기 매우 편하다. 위 예제는 JNLP 의 개념을 보여주지만, 장점을 보여주기에는 너무 부족하다. 대량의 사용자와 상호작용하는 어플리케이션을 사용할 때에만, JNLP 의 장

점을 알 수 있다. JNLP 및 Java Webstart 에 대해서 좀더 알고 싶다면 다음 URL 을 참고하기 바란다. http://java.sun.com/products/javawebstart/demos.html

4.9 Application Version 업데이트

4.9.1 소개

JNLP 는 어플리케이션을 업데이트한 후 리소스를 부분적으로 다운로드 받을 수 있다. 이 장에서는 어플리케이션 버전이 어떻게 업데이트 되는지에 대해서 설명하도록 하겠다.

4.9.2 초기 어플리케이션 코드

아래 코드는 볼드체 라인 부분만 제외하고는 이전에 만들었던 예제와 동일하다.

<<HelloJeus.java>>

```
import javax.swing.*;
import java.awt.Font;
import java.awt.BorderLayout;
import java.awt.event.*;
public class  HelloJeus
{
    public static void main(String[] args)
    {
        new HelloJeusFrame();
    }
}
class HelloJeusFrame extends JFrame
{
    JLabel label = new JLabel(“Hello Jeus 1.0”);
    public HelloJeusFrame()
    {
        label.setFont( new Font(  “Helevetica” , Font.BOLD,70));
        getContentPane().setLayout( new BorderLayout() );
        getContentPane().add( label , BorderLayout.CENTER);
        setSize(500,250);
        setVisible( true );
    }
}
```

참고: 위의 소스를 컴파일한 후 HelloJeus1.0.jar 로 묶는다. 그리고 CLIENT_HOME\Update 디렉토리에 복사한다.

4.9.3 JNLP File 초기화

다음 JNLP 파일은 초기화를 위한 어플리케이션이다.

<<*HelloJeus.jnlp*>>

```
<?xml version=”1.0”?>
<jnlp codebase="http://localhost:9744/jnlp" spec="1.0">
    <information>
      <locale>en_US</locale>
        <title>HelloJeus Test</title>
        <vendor>Tmax</vendor>
        <homepage href="http://www.tmax.co.kr"/>
        <description kind="short">Hello Test</description>
    </information>
    <resources>
      <j2se version="1.4+"/>
        <jar href="HelloJeus.jar" version=1.0 main="true" />
    </resources>
    <application-desc main-class="HelloJeus"/>
</jnlp>
```

참고: CLIENT_HOME 디렉토리에 HelloJeus.jnlp 파일로서 저장한다.

4.9.4 초기 어플리케이션 등록

앞에서의 어플리케이션 등록과 같지만 이번에는 버전 값을 수정하였다.

<<*JNLPMain.xml*>>

```
<?xml version="1.0" encoding="UTF-8"?>
<jeus-system xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
<jnlp-resource-config>
    ...
    <jnlp-resource>
        <rsc-name>HelloJeus/HelloJeus.jar</rsc-name>
         <version>1.0</version>
        <rsc-path>HelloJeus/HelloJeus.jar</rsc-path>
    </jnlp-resource>
    <jnlp-resource>
        <rsc-name>HelloJeus/HelloJeus.jnlp</rsc-name>
```

```
        <version>1.0</version>
        <rsc-path>HelloJeus/HelloJeus.jnlp</rsc-path>
    </jnlp-resource>
    ...
</jnlp-resource-config>
```

이 어플리케이션을 실행하게 되면 결과는 아래 그림과 같다[그림 4].

그림 4. application 업데이트.

4.9.5 어플리케이션 코드 업데이트

<<HelloJeus.java>>

```
import javax.swing.*;
import java.awt.Font;
import java.awt.BorderLayout;
import java.awt.event.*;
public class  HelloJeus
{
    public static void main(String[] args)
    {
        new HelloJeusFrame();
    }
}
class HelloJeusFrame extends JFrame
{
    JLabel label = new JLabel(“Hello Jeus 2.0”);
```

```
    public HelloJeusFrame()
    {
        label.setFont( new Font(  “Helevetica” , Font.BOLD,70));
        getContentPane().setLayout( new BorderLayout() );
        getContentPane().add( label , BorderLayout.CENTER);
        setSize(500,250);
        setVisible( true );
    }
}
```

참고: 위의 소스를 컴파일한 후 HelloJeus2.0.jar 로 묶는다.

4.9.6 JNLP 파일 업데이트

<<*HelloJeus.jnlp*>>

```
<?xml version="1.0"?>
<jnlp codebase="http://localhost:9744/jnlp" spec="1.0">
    <information>
      <locale>en_US</locale>
        <title>HelloJeus Test</title>
        <vendor>Tmax</vendor>
        <homepage href="http://www.tmax.co.kr"/>
        <description kind="short">Hello Test</description>
    </information>
    <resources>
      <j2se version="1.4+"/>
        <jar href="HelloJeus.jar" version=2.0 main="true" />
    </resources>
    <application-desc main-class="HelloJeus"/>
</jnlp>
```

참고: 이미 등록된 JNLP 파일을 수정한다. 수정하고 나서 다시 등록할 필요는 없다.

4.9.7 Updating the Application 버전 업데이트

같은 리소스 이름으로 새로 등록한다. 그러나 **version** 항목을 2.0 으로 설정한다.

참고: 이전 버전을 지울 필요는없다. Java Web Start 에서 jnlp 파일을 분석해서 자동으로 2.0 의 리소스를 다운받는다.

최종적으로 수정된 JNLPMain.xml 을 보면 다음과 같다.

```
<?xml version="1.0" encoding="UTF-8"?>
<jeus-system xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
<jnlp-resource-config>
    <jnlp-resource>
        <rsc-name>HelloJeus.jar</rsc-name>
        <rsc-path>Update/HelloJeus2.0.jar</rsc-path>
        <version>2.0</version>
    </jnlp-resource>
    <jnlp-resource>
        <rsc-name>HelloJeus.jar</rsc-name>
        <rsc-path>Update/HelloJeus1.0.jar</rsc-path>
        <version>1.0</version>
    </jnlp-resource>
    <jnlp-resource>
        <rsc-name>UpdateTest.jnlp</rsc-name>
        <rsc-path>HelloJeus.jnlp</rsc-path>
        <version>1.0</version>
    </jnlp-resource>
</jnlp-resource-config>
```

어플리케이션을 실행하게 되면 결과는 아래와 같다 [그림 5].

그림 5. JNLP 어플리케이션의 버전 2.0 최신 업데이트.

4.9.8 결론

지금까지 JNLP 어플리케이션의 버전을 어떻게 업데이트 하는지에 대해 알아보았다. JNLP 는 어플리케이션의 업데이트를 위해 부분적인 다운로드가 가능하므로, 자주 업데이트 되는 어플리케이션 환경에 적합하다.

4.10 결론

이상으로 JEUS 에서 JNLP 어플리케이션을 어떻게 개발하고 설정하는지 알아보았다. JNLP 에는 장단점이 있으므로, JNLP 서비스를 적용하기 전에 다운로드 타임을 측정해 보길 바란다. JNLP 를 적절하게 사용한다면, 어플리케이션의 성능을 많이 향상시킬 수 있다.

5 JEUS CAS

5.1 소개

J2EE™ CAS (Client Access Services)는 COM 어플리케이션 에서 J2EE Enterprise JavaBeans™ 뿐 아니라 Local Java Libraries 를 이용할 수 있는 서비스를 제공한다. COM 을 사용하는 Client 가 JEUS 서버에 액세스하려면 JEUS CAS 가 필요하다. 자세한 설명은 다음 http://java.sun.com 을 참고 하기 바란다.

5.2 JEUS CAS 개요

5.2.1 아키텍쳐

COM 어플리케이션을 사용하는 Client 가 JEUS 서버에 접근하여 EJB Bean 의 정보를 가져오기 위해서 COM 객체를 통해 필요한 요청을 한다. 요청을 받으면 COM Application 내부에서는 자신의 Address Space 내에 JVM 을 실행하여 EJB 서버에 메시지를 보내게 되는데 이를 담당하는 것이 COM-Bridge 이다. 이러한 COM-Bridge 가 생성한 JVM 이 JEUS EJB 서버에 접속하도록 인터페이스를 제공하는 것이 JEUS-CAS 인 것이다[그림 6].

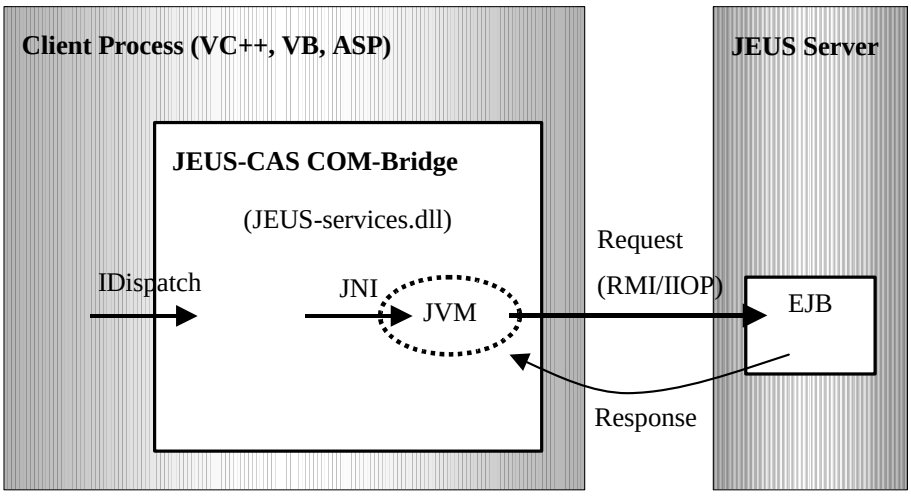

그림 6. JEUS CAS 아키텍처.

5.2.2 서비스

- **Java Service**: JVM 에서 동작되는 클래스로딩, 객체 생성, Static 클래스 메소드 사용, 타입 캐스팅등의 Service 를 사용할 수 있게 하는 Module 이다. 즉 COM 어플리케이션에서 JVM 을 이용할 수 있도록 Service 를 제공하는 부분이다.

- **JEUS Service**: Enterprise Service 에 연결할 때에 필요한 기능으로 JEUS 에 관련된 Service 를 제공하는 Module 이다. 이것은 EJB 에 접속하는 것을 포함해서, JEUS 관련 기능을 제공한다.

5.2.3 디렉토리

JEUS 를 설치하게 되면 JEUS_HOME\samples\CAS 디렉토리에 JEUS CAS 관련 예제가 설치되고 JEUS_HOME\lib\system 디렉토리에 실행 파일, DLL 파일 등의 JEUS CAS Component 들이 생기게 된다.

- **samples\CAS 디렉토리**

  - jeus-services Directory: JEUS CAS application 을 위한 jeus-services.dll 에 대한 소스 코드가 포함되어있다. 추가적으로 Interface 를 넣고자 할 때에는 이 파일을 수정하면 된다.

  - j2eecas-1_0-ea4-win-combridge.exe: JEUS CAS 를 실행하기 위해서는 먼저 J2EE™ CAS 를 설치하여 Com-Bridge 에 필요한 기본 파일들이 시스템에 설정 되어 있어야 한다.

  - examples 디렉토리: JEUS CAS 의 동작을 확인하기 위한 예제 프로그램들이 들어있다

    - asp 디렉토리: ASP 를 이용한 예제.

    - vb 디렉토리: Visual Basic 를 이용한 예제.

    - vc 디렉토리: Visual C++ 를 이용한 예제.

    - Classes 디렉토리: 예제 Program 에서 사용될 Comaccount EJB 의 클래스 파일.

    - src 디렉토리: 예제 EJB 인 Comaccount 를 deploy 하기 위해 필요한 Java 소스 파일들과 client 에서의 접속을 테스트 하기 위해 필요한 Java 소스 파일

    - setting 디렉토리: 예제를 동작시키기 위해 필요한 파일들

- **lib\system 디렉토리**

  - jeus-services.dll: 기본으로 제공되는 JEUS CAS 어플리케이션. DLL 파일로 제공되며, JEUS CAS 를 이용할 COM Application 에서 Link 시켜 사용된다. JEUS 가 설치 될 때 JEUS_HOME\lib\system 디렉토리에 복사된다(예: c:\jeus\lib\system\jeus-services.dll).

5.2.4 결론

COM 어플리케이션이 JEUS 의 EJB 를 호출할때 JEUS CAS 가 사용된다. JEUS CAS 는 J2EE CAS 의 아키텍쳐를 준수하고 있다.

5.3 JEUS CAS 설치

5.3.1 소개

J2EE CAS 는 JEUS CAS 의 핵심 컴포넌트이므로 반드시 설치해야 한다. 설치 파일은 JEUS_HOME\samples\CAS 디렉토리 안에 j2eecas-1_0-ea4-win-combridge.exe 의 형태로 존재한다.

참고: CAS 와 JEUS 의 연결이 가능하도록 클라이언트에 JEUS 를 설치 해야 한다.

5.3.2 설치

1. j2eecas-1_0-ea4-win-combridge.exe 실행.
2. Installer 가 실행되면 설정 화면이 나온다.
3. J2EE CAS 가 설치될 Directory 를 설정한다[그림 7].

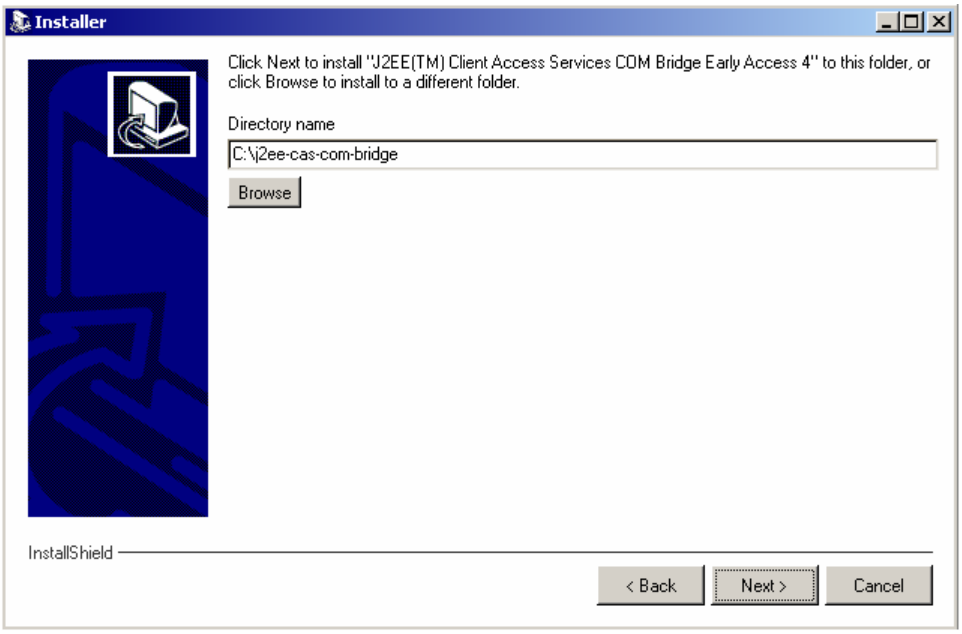

그림 7. J2EE CAS installer 가 설치될 디렉토리 선택.

4. **Next >** 버튼을 클릭하면 디렉토리에 파일들이 복사된다.

5. 모든 파일의 복사가 끝나면 설정 다이얼로그 화면이 나온다[그림 8]. 만약 JDK 1.4 를 사용한다면 JVM 을 Other 로 선택하고, JVM 의 경로를 입력한다.

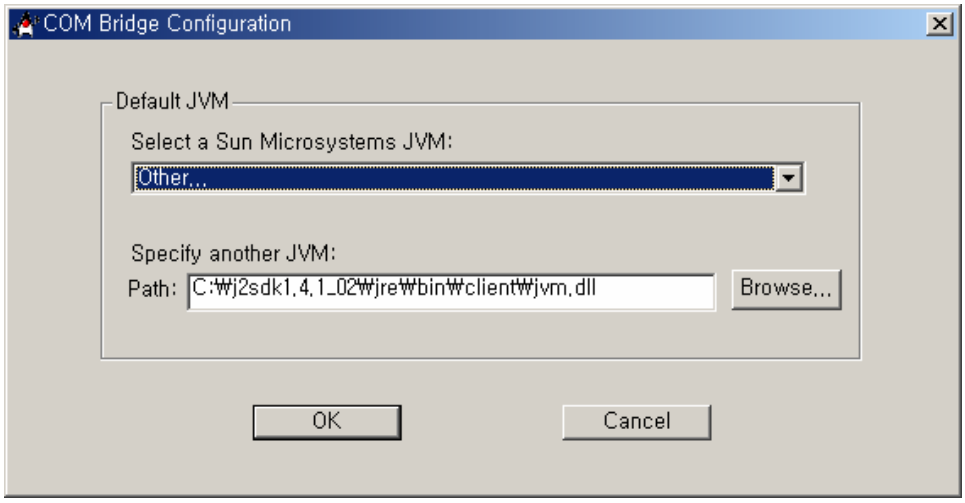

그림 8. COM Bridge 설정 다이얼로그.

6. J2EE CAS 에서 사용하는 JDK 를 선택한다.

5.3.3 환경 변수 설정

다음 환경 변수는 클라이언트 머신에서 설정해야 할 작업이다.

- **COMBRIDGE_HOME**: J2EE CAS 가 설치된 디렉토리로 기본값은 C:\j2ee-cas-com-bridge 이다.
- **JAVA_HOME:** JDK 가 설치된 디렉토리로서, JDK 1.4 를 사용할 경우 기본값은 C:\jdk1.4 이다.
- **JEUS_HOME:** JEUS 가 설치된 디렉토리로, 기본값은 C:\jeus 이다.

참고: JAVA_HOME 과 JEUS_HOME 은 JEUS 가 설치되는 과정에서 자동으로 설정된다.

5.3.4 결론

J2EE CAS 설치를 알아보았다. 다음 장에서는 EJB 를 호출하는 COM 어플리케이션의 개발 방법을 알아본다.

5.4 EJB 예제

5.4.1 소개

COM 어플리케이션 예제를 테스트하려면 EJB 가 하나 필요하다. 이 EJB 는 ASP, Visual C++ 같은 모든 클라이언트 어플리케이션에서 사용된다. EJB 가 준비되었다면 다음 내용을 보기 바란다.

5.4.2 COMAccount EJB

Comaccount JEUS CAS 를 이용하여 EJB 컴포넌트에 접속하는 과정을 보여주는 예제 프로그램이다. 이 예제는 J2EE CAS 매뉴얼에서 제공하는 예제 중 하나이다.

COMAccount 예제를 이용한 COM application 은 다음 절차로 동작한다.

1. JEUS EJB Server 에 Deploy 된 Account EJB Bean 에 접속한다.
2. JEUS EJB component 들과 연결하기 위해 필요한 JEUS EJB 의 클래스 파일이 들어있는 JEUS_HOME\lib\system\jeus.jar 파일과 Stub 클래스들이 들어있는 클라이언트 JAR 파일을 classpath 에 설정한다.
3. JEUS Naming 서비스를 이용하여 COMAccount EJB 의 Home Interface 를 Look up 한다.

4. Account EJB 의 findAll method 와 findByPrimaryKey method 를 불러낸다.

5.4.3 EJB 실행

1. 사용하는 컴퓨터에 몇가지 환경변수 설정을 한다. 앞에서 J2EE CAS 를 설치한 후 설정한 환경 변수인 COMBRIDGE_HOME 과 JAVA_HOME 이 설정되어 있어야 한다. 그리고, EJB 클래스들을 읽어오기 위해 JEUS_HOME 이 설정되어 있어야 한다.

```
C:\>set COMBRIDGE_HOME=C:\j2ee-cas-com-bridge
C:\>set JAVA_HOME=C:\jdk1.4
C:\>set JEUS_HOME=c:\jeus
```

2. example\setting 디렉토리에 있는 createComaccountTypeLibs.bat 파일을 실행시킨다. 이 파일은 COMAccount 예제에서 사용하는 Java 클래스 파일들을 Type Library 파일로 변환시켜주는 gentypelib.exe 가 포함되어 있다. 위의 환경 변수 설정이 제대로 되어 있을 경우, 정상적으로 Typelib 파일이 생성된다.

```
C:\j2ee-cas-com-bridge\doc\guide\examples\typelib>dir/2
...
[.]
[..]
comaccount_Account.tlb
comaccount_AccountHome.tlb
java_util_Collection.tlb
java_util_Iterator.tlb
```

3. samples\CAS\examples\src\bean\comaccont 디렉토리에 있는 create.sql, insert.sql 파일을 사용해서 JEUS 에 연결된 DB 에 입력한다. 이 파일들은 COM Account Example 에서 읽어오는 COM Account Data 들이다. create.sql 파일에는 Account Table 을 생성하는 SQL 문이 들어있고, insert.sql file 에는 Table 의 Data 를 Insert 하는 SQL 문들이 들어있다.

```
C:\jeus\samples\CAS\examples\src\bean\comaccount>type
   create.sql
drop table comaccount;
create table comaccount
(comaccount_id varchar(3) constraint pk_comaccount primary
   key,
first_name varchar(24),
```

```
last_name varchar(24),
type varchar(24),
balance double precision,
credit_limit double precision);
exit;
```

4. 이제 JEUS EJBServer 에 COMAccount EJB bean 을 등록시킨다. samples\CAS\examples\classes 디렉토리와 samples\CAS\examples\src 디렉토리에 있는 COM Account Bean 클래스 파일과 Java 소스 파일들을 이용하여 EJB Bean 을 만들어서 JEUS 에 등록시키고 deploy 시키면 된다. 아래는 테스트를 위해 AccountClient.class 를 실행해 본 것이다.

   참고: COMAccountEJB 의 등록과 deploy 에 대한 사용법은 JEUS EJB 안내서를 참고하기 바란다.

```
c:\jeus\samples\CAS\examples\classes>java -
   classpath .;c:\jeus\lib\system\jeus.jar;c:\jeus\client\comaccoun
   t2.jar -
   Djava.naming.factory.initial=jeus.jndi.JEUSContextFactory  -
   Djava.security.policy= c:\jeus\samples\CAS\examples
   \src\client\comaccount\policy comaccount.AccountClient
[ErrorMsgManager] Message Manager is initialized
...
lookup done
Account ID: 100
Account Type: Checking
First Name: Phil
Last Name: Smith
Balance: 1230.0
Credit Limit: 0.0
Account ID: 555
Account Type: Credit Card
First Name: Sally
Last Name: Smith
Balance: 33.0
Credit Limit: 1000.0
Account ID: 490
Account Type: Savings
First Name: Rupert
```

```
Last Name: Smith
Balance: 33000.0
Credit Limit: 0.0
Account ID: 296
Account Type: Money Market
First Name: Jose
Last Name: De Anza
Balance: 1200.0
Credit Limit: 0.0
Account ID: 807
Account Type: Checking
First Name: Lee
Last Name: Addison
Balance: 10.0
Credit Limit: 0.0
print done
```

5. JEUS CAS 프로그램을 등록한다. JEUS CAS 프로그램은 DLL 형태로 제공된다. DLL 파일을 사용하기 위해서는 우선 등록을 해야 한다. jeus-services.dll 파일이 들어있는 디렉토리에서 다음을 실행시키면 파일이 등록된다.

```
c:\jeus\lib\system>regsvr32 jeus-services.dll
```

5.4.4 결론

EJB COMAccount는 예제 샘플인 COM 어플리케이션에서 호출한다. 다른 COM 어플리케이션 예제를 테스트 하기 전에 EJB를 JEUS EJB Container에 deploy 해야 한다는 것을 명심하기 바란다.

5.5 JEUS CAS 프로그래밍

5.5.1 소개

COM 어플리케이션을 개발 실행하기 위해서는 JEUS CAS API 와 J2EE CAS 뿐만 아니라 JVM control, classpath 설정, JEUS Naming lookup 등에 대해 이해하고 있어야만 한다. J2EE CAS가 설치된 디렉토리에 있는 J2EE CAS 매뉴얼을 유용하게 사용하기 바란다.

JEUS CAS의 예제로는 COM을 이용한 Visual C++, Visual Basic, ASP 등을 제공한다.

5.5.2 VC++

COM-Bridge 설치할 때 생성되는 jvm-control.dll (예: C:\j2ee-cas-com-bridge\bin\jvm-control.dll)과 jeus-service.dll, gentypelib.exe 에 의해 생성된 typelib file 에 해당하는 Comaccount_AccountHome.tlb 을 include 한다.

```
void Initialize(void)
{
// Initialize the JVM
    IJvmControlPtr jvmCtl(__uuidof(JvmControl));

      jvmCtl-
>put_Classpath(L"C:\\jeus\\lib\\system\\jeus.jar;C:\\jeus\\lib\\s
ystem\\jmxri.jar;C:\\jeus\\lib\\system\\jmxremote.jar;C:\\jeus\\l
ib\\system\\jmxtools.jar;C:\\jeus\\samples\\CAS\\examples\\classe
s\\deployexample\\build");

      //When system classpath is set
      //jvmCtl-
>put_Classpath(L"%JEUS_HOME%\\lib\\system\\jeus.jar;%JEUS_HOME%\\
lib\\system\\jmxri.jar;%JEUS_HOME%\\lib\\system\\jmxremote.jar;%J
EUS_HOME%\\lib\\system\\jmxtools.jar;%JEUS_HOME%\\samples\\CAS\\e
xamples\\classes\\deployexample\\build");

      jvmCtl->put_JvmOptions(L"-Djeus.baseport=2100");
    jvmCtl->StartJvm();

    // Create an enterprise services object for EJB access
      jeus.CreateInstance(__uuidof(JeusServices));
    jeus->put_ProviderURL(L"localhost");

    //Retrieve reference to EJB home interface using enterprise
services
      acctHome = jeus->LookupEjbHome(L"MyComAccount",
L"comaccount.AccountHome");
}
```

JEUS->LookupEjbHome(L"MyComaccount", L"comaccount.AccountHome")은 export-name 이 “MyComAccount”인 comaccount.AccoutHome 클래스를 lookup 한다.

아래 결과는 examples\vc 에 있는 VC++의 프로젝트 파일(Examples6.dsw)을 열어서 실행시킨 것이다.

```
C:\jeus\samples\CAS\examples\vc>Debug\example6
[ErrorMsgManager] Message Manager is initialized
...
Account ID: 100
Account Type: Checking
First Name: Phil
Last Name: Smith
Balance: 1230
Credit Limit: 0

Account ID: 555
Account Type: Credit Card
First Name: Sally
Last Name: Smith
Balance: 33
Credit Limit: 1000

Account ID: 490
Account Type: Savings
First Name: Rupert
Last Name: Smith
Balance: 33000
Credit Limit: 0

Account ID: 296
Account Type: Money Market
First Name: Jose
Last Name: De Anza
Balance: 1200
Credit Limit: 0

Account ID: 807
Account Type: Checking
First Name: Lee
Last Name: Addison
Balance: 10
Credit Limit: 0
```

5.5.3 Visual Basic

Visual Basic 에서 JEUS-CAS 를 사용하기 위해서 Type Library 를 프로젝트 참조에 추가해야 한다. 그래서 jvm-control.dll 과 jeus-services.dll 이 실행할 수 있도록 한다.

다음 코드는 Visual Basic 객체가 EJBHome 을 look up 하는 예제 샘플이다.

```
'Initialize the JVM
 Dim JvmCtl As New JvmControl
 'when system classpath is not set
 JvmCtl.Classpath =
"C:\jeus\lib\system\jeus.jar;C:\jeus\lib\system\jmxtools.jar;C:\j
eus\lib\system\jmxri.jar;C:\jeus\lib\system\jmxremote.jar;C:\jeus
\samples\CAS\examples\classes\deployexample\build"
 'JvmCtl.Classpath =
"%JEUS_HOME%\lib\system\jeus.jar;%JEUS_HOME%\lib\system\jmxtools.
jar;%JEUS_HOME%\lib\system\jmxri.jar;%JEUS_HOME%\lib\system\jmxre
mote.jar;%JEUS_HOME%\samples\CAS\examples\classes\deployexample\b
uild"

 JvmCtl.JvmOptions = "-Djeus.baseport=2100"

 JvmCtl.StartJvm

'Create an enterprise services object for EJB access
Set Jeus = New JeusServices
Jeus.ProviderURL = "localhost"

'Retrieve reference to EJB home interface using enterprise
services
Set AcctHome = Jeus.LookupEjbHome("MyComAccount",
        "comaccount.AccountHome")
```

JEUS.LookupEjbHome("MyComaccount", "Comaccount.AccountHome") 를 통해 commaccount.AccountHome 타입이면서 Export 이름이 MyComaccount 인 EJB bean 을 찾아낸다.

examples\vb 디렉토리 내에 들어 있는 project file(Example6.vbp)을 열어서 프로그램을 실행해 보면 결과값을 알 수 있다.

5.5.4 ASP

참고:ASP Program 을 실행하기 위해서는 Windows 2000 의 Internet Information Services 5.0 이상이 설치되어 있어야 한다. ASP 프로그램이 실행되지 않을 경우 IIS server 가 제대로 동작중인지를 확인해야 한다

다음 ASP 샘플 코드는 EJBHome 객체가 look up 하는 과정을 보여 주고 있다.

```
<SCRIPT LANGUAGE=VBSCRIPT RUNAT=SERVER>
Sub Application_OnStart
    '*** Initialize the JVM
    Dim JvmCtl
    Set JvmCtl = Server.CreateObject("J2EECAS.JvmControl")
   'when system classpath is not set
   JvmCtl.Classpath =
"C:\jeus\lib\system\jeus.jar;C:\jeus\lib\system\jmxtools.jar;C:\j
eus\lib\system\jmxri.jar;C:\jeus\lib\system\jmxremote.jar;C:\jeus
\samples\CAS\examples\classes\deployexample\build"
   'JvmCtl.Classpath =
"%JEUS_HOME%\lib\system\jeus.jar;%JEUS_HOME%\lib\system\jmxtools.
jar;%JEUS_HOME%\lib\system\jmxri.jar;%JEUS_HOME%\lib\system\jmxre
mote.jar;%JEUS_HOME%\samples\CAS\examples\classes\deployexample\b
uild"

   JvmCtl.JvmOptions = "-Djeus.baseport=2100"
    JvmCtl.StartJvm

    '*** Create an enterprise services object for EJB access
    Dim JEUS
    Set JEUS = Server.CreateObject("J2EECAS.JEUSServices")
    JEUS.ProviderURL = "localhost"

    '*** Retrieve reference to EJB home interface
    'using enterprise services

    Dim AcctHome
    Set AcctHome = JEUS.LookupEjbHome("MyComAccount",
                                      "comaccount.AccountHome")
    Set Application("AcctHome") = AcctHome
End Sub
```

```
</SCRIPT>
```

JEUS.LookupEjbHome("MyComaccount", "Comaccount.AccountHome") 을 통해서 Comaccount.AccountHome 타입이면서 Export 이름이 MyComaccount 인 EJB bean 을 가져오게 된다. 이후 ASP 코드에서 이 객체를 이용해서 JEUS 에 request 를 보내게 된다.

ASP 프로그램을 실행시키기 위해 소스 코드가 들어있는 examples\asp 를 IIS 에서 실행 가능하게 Virtual 디렉토리로 등록한다. 그리고 Internet Explorer 를 통해서 실행시키면 ASP 프로그램이 실행되는 화면을 볼수 있을 것이다.

주의: 웹 브라우저에서 ASP 프로그램 실행시 IIS 의 "Application Protection" 설정에 의해 문제가 발생하여 Error 가 생기는 경우가 있는데, 이때는 등록시킨 Virtual Directory 의 Property 를 "High(Isolated)"로 설정하면 문제가 해결된다.

5.5.5 결론

샘플 예제는 EJB Home 객체를 어떻게 얻어오는 것만 보여주었다. 이 객체를 얻고나서는 EJB Remote 객체를 생성하면, 필요한 메소드를 실행시킬 수 있다.

5.6 결론

JEUS CAS 를 사용하려면 실제 구현해서 사용하기 전에 몇가지 단계가 필요하다. 이 단계를 거친 후에 COM 어플리케이션에서 CAS 서비스를 사용할 수 있다. J2EE CAS 는 JEUS CAS 의 핵심이므로 J2EE CAS 매뉴얼을 읽어보길 권한다. J2EE CAS 에 대한 자세한 사항은 http://java.sun.com 을 참조하길 바란다.

JEUS CAS 는 COM 어플리케이션에서 Java 어플리케이션으로 호출을 가능하게 하지만, JEUS-COM Connector 모듈("J2COM")은 위의 사항과는 반대로 Java 어플리케이션에서 COM 어플리케이션으로 호출이 가능하다.

다음 장에서는 JEUS-COM Connector bridge 에 대해 설명하도록 하겠다.

6 JEUS-COM Connector (J2COM)

6.1 소개

JEUS J2COM 은 Java 기반의 어플리케이션에서 COM 으로 구성된 서비스 및 라이브러리의 호출을 위해 개발된 단방향 인터페이스 모듈이다. J2COM 을 이용하면 JEUS 를 통하여 이루어지는 웹 서비스에 기존에 개발된 COM 기반 서비스들을 별 다른 수정없이 사용할 수 있게 되며, 또한 Java Swing 과 같은 기술을 사용하지 않더라도 손쉬운 GUI 개발이 가능한 MS 윈도우 기반의 다양한 어플리케이션을 프리젠테이션 로직으로 활용 할 수 있게 된다.

또한 이것은 Microsoft 가 제공하는 개발환경에 익숙한 개발자들이 다시 Java 언어를 배우기 위한 추가적인 시간과 노력을 필요로 하지 않는다는 것을 의미하며, 앞서의 장점과 더불어 직접적인 개발 기간의 단축 및 개발 비용 절감의 효과도 얻을 수 있다. 이외에도 필요에 따라 핵심적인 비지니스 로직을 Java 가 아닌 COM 으로 개발하여 핵심 로직이 외부에 노출되는 것을 원천 봉쇄하는 방법도 생각해 볼 수 있다.

즉 JEUS 가 제공하는 EJB, Servlet, JMS 등 Java 기반의 서비스에 덧붙여 VC 혹은 VB 로 개발된 COM 을 서비스 루틴으로 활용 할수 있게 됨으로써 윈도우 플랫폼 환경만이 제공하는 장점을 살릴 수 있게 되었으며, 전체 프로젝트에 각 언어들의 장점만을 살리는 유연한 개발이 가능해지게 된다

6.2 JEUS J2COM 개요

6.2.1 소개

이 절에서는 J2COM 인터페이스 모듈과 2 개의 기본폼(Local Invocation 과 Remote Invocation)에 대해서 설명하도록 하겠다.

6.2.2 아키텍쳐

JEUS J2COM 은 마샬/언마샬 작업을 수행하는 J2COM 코어 라이브러리와 COM 라이브러리의 원격 수행을 위한 COM Manager 로 이루어진다.

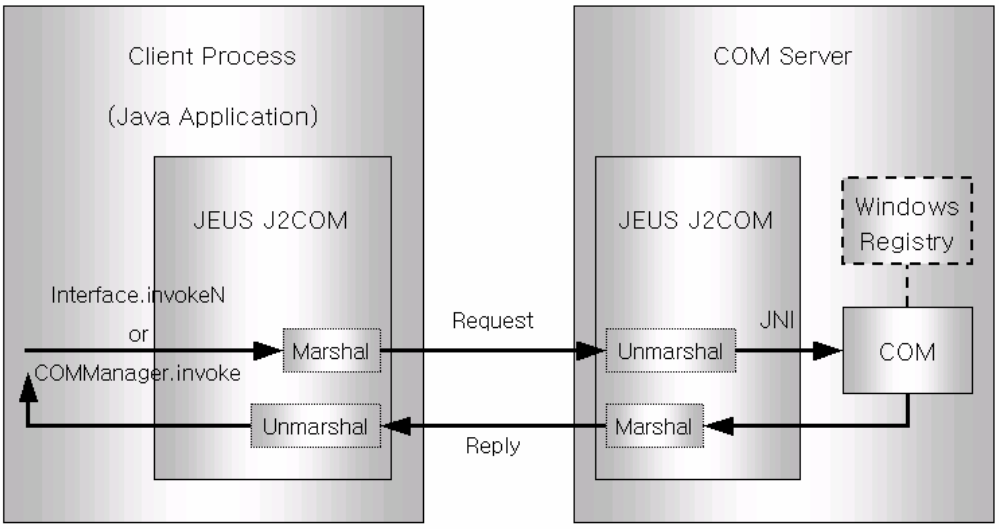

그림 9. J2COM 아키텍처.

J2COM 코어 라이브러리는 실제 COM 인스턴스의 생성 및 메소드의 호출을 담당하며 이를 위해 Java 클라이언트가 넘긴 Java primitive 형 데이터 혹은 String 형 데이터를 COM 에서 인식할 수 있는 데이터로 마샬/언마샬 하는 작업을 수행한다. 이 과정에서 사용되는 직렬화 데이터는 J2COM 고유의 것으로 Java 의 직렬화와는 무관하다.

COM Manager 는, 원격으로 COM 을 호출하고 그에 대한 응답을 받아 J2COM 코어 라이브러리로 넘기는 간단한 클라이언트/서버 프로세스이다. 로컬 영역의 COM 을 직접 호출하는 경우에는 필요하지 않다.

COM Manager 를 이용하여 메소드를 호출하는 경우, 각 호출에 대하여 인스턴스를 새로 생성해서 사용하므로, 하나의 작업에 계속적인 상태를 유지할 필요가 있는 경우에는 사용 할 수 없다.

6.2.3 Service Category

J2COM 은 COM 클라이언트(COM 을 호출하는 Java 어플리케이션)와 COM 서비스를 제공하는 서버의 위치에 따라 두 가지 방식으로 사용할 수 있다.

Local Invocation

J2COM Local Invocation 은 COM 을 로컬 영역에서 직접 호출하는 것으로 이 경우 COM 서비스는 COM 클라이언트의 JVM 내에서 JNI 를 통해 직접 호출된다.

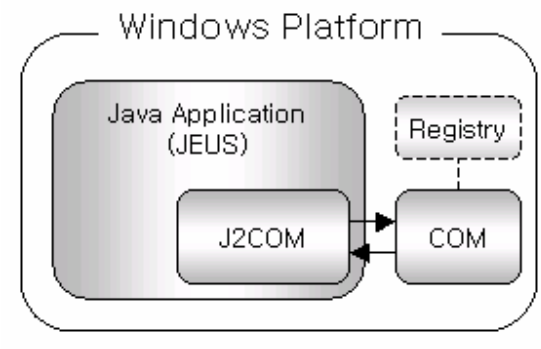

그림 10. Local Invocation.

위의 [그림 10] 처럼 Java 어플리케이션은 J2COM 과 같이 제공되는 tool(j2comGen) 을 이용하여 생성된 인터페이스를 통하여 COM 인스턴스를 생성하거나 메소드를 호출하는 작업을 하게 된다.

참고: J2comGen Tool 에 대한 설명은 6.4 절에서 설명하도록 한다.

클라이언트 프로그램과 COM 이 동일한 서버에 존재하는 경우에도 COM Manager 를 이용한 Remote Invoke 가 가능하지만 이 경우 메소드의 수행 결과는 가져올 수 있어도 생성된 인스턴스를 직접 제어할 수는 없으므로 이 점을 유의하도록 한다.

Remote Invocation

J2COM Remote Invocation 은 J2COM Manager 를 이용하여 COM 인스턴스 생성 및 메소드 호출을 위임하는 것으로 리모트에 위치한 COM 을 사용하는 것이 가능하다

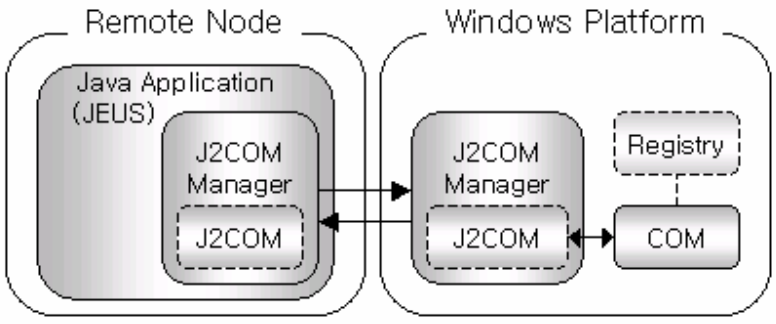

그림 11. Remote Invocation.

위의 [그림 11] 처럼 Java 어플리케이션은 COM 이 위치한 서버의 이름과 COM 이 제공하는 메소드의 이름 및 파라미터를 로컬의 COM Manager 에게 넘겨주며 로컬 COM Manager 는 이를 해당 서버에 위치한 COM Manager 를

통해 COM 인스턴스를 생성하고 메소드를 호출하며, 그 결과값을 받아 클라이언트에게 전달하는 작업을 하게 된다.

COM 이 위치한 서버에는 미리 COM Manager 가 시작되어 있어야 하며 클라이언트 측은 메소드를 호출할 때 직접 COM Manager 를 사용하므로 별다른 설정을 필요로 하지 않는다.

J2COM Manager 간의 통신은 소켓을 이용하여 구현되므로 J2COM Manager 가 위치한 서버의 위치 정보와 소켓 타임 아웃 시간 등을 지정할 필요가 있다.

6.3 JEUS J2COM 설정

6.3.1 소개

이번 절에서는 J2COM 의 구성 설정 및 설치에 대해서 예제를 통해서 설명하도록 하겠다. 다음 절에서는 J2COM 의 개발에 대해서 설명하도록 하겠다.

6.3.2 설치

J2COM 은 기본적으로 jeus.com.j2com 패키지의 클래스 파일과 J2COM 라이브러리(j2com.dll) 파일을 사용하며 환경파일 파싱을 위해 jxerces.jar 파일을 사용한다.

또한 Java 클라이언트 의 요청을 주고 받기 위해 jeus.util.io 패키지의 클래스 파일과 NSStream.dll (또는 NSStream.so) 라이브러리 파일을 필요로 한다. 이 파일들은 JEUS 설치 과정에 같이 설치되지만 J2COM 을 사용하기 위해 JEUS 가 반드시 필요한 것은 아니다.

라이브러리 파일들은 각 시스템의 라이브러리 검색 경로에 추가 하거나 컴파일 및 실행시 –Djava.library.path 로 설정하도록 하고 클래스 파일과 jar 파일은 마찬가지의 경우에 해당 플랫폼의 CLASSPATH 에 포함 되도록 위치시킨다.

6.3.3 설정

Local Invocation

이 경우 툴(j2comGen)을 이용하여 생성된 인터페이스를 사용하여 직접 호출하게 되므로 별도의 설정이 필요하지 않다.

Remote Invocation

리모트에 위치한 COM 을 호출하기 위해서는 클라이언트의 요청을 받아 리모트 서버로 보내고 이를 받아들여 COM 호출을 대행해 주는 J2COM Manager 를 사용해야 할 필요가 있다. J2COM Manager 는 리모트 서버들에 대한 정보를 갖고 있으며 이를 통해 리모트 서버의 서비스를 호출한다. 클라이언트의 호출 전에 리모트 서버에는 J2COM Manager 가 동작하고 있어야 한다.

설정 파일 형식은 다음과 같다.

```
<j2com-severs>
   <j2com-server>
       <server-name>COMServer1</server-name>
      <ip-address>143.248.1.1</ip-address>
      <port>1004</port>
      <timeout>10000</timeout>
   </j2com-server>
</j2com-severs>
```

`<j2com-server>` 태그에 com 이 위치한 서버를 지정한다. 하나 이상 지정 할 수 있으며 COM 서버의 경우 자신에 대한 정보를 반드시 지정해야 한다. 다음은 하위 엘리먼트이다.

- **server-name** [(required) default : null]: 이 속성은 서비스 요청을 할 COM 서버의 위치를 지정하기 위해 클라이언트 프로그램 내에서 사용된다.
- **ip-address** [(required) default : null]: COM 서버의 IP 주소 설정.
- **port** [(required) default : 0]: COM 서버의 port 설정
- **timeout** [(optional) default : -1]: COM 서버와 통신 간의 timeout 시간을 ms 단위로 설정한다

COM 서버들에 대한 설정파일의 위치는 J2COM Manager 실행시 –Dj2com.configfile 옵션으로 지정한다.

J2COM 의 시작은 다음과 같다.

```
java -Dj2com.configfile=<J2COMConfigFileName>
     -Dj2com.COMManager.traceLog=<TraceLog>
```

```
-classpath <ClassPaths> jeus.com.j2com.COMManager
```

위의 예제에서 이탤릭체의 의미는 다음과 같다.

- **J2COMConfigFileName** [(required)]: J2COM 설정 파일의 위치 지정
- **TraceLog** [(optional) true | false]: J2COM Manager 가 클라이언트의 요청을 받고 결과를 리턴하는 과정을 화면에 기록으로 남길 것인지 지정한다.
- **ClassPaths** [(required)]: J2COM Manager 가 필요로 하는 클래스 파일에 대한 경로를 지정한다. jxerces.jar, j2com.jar 파일과 jeus.util.io 패키지의 클래스 파일들을 지정하도록 한다

6.4 JEUS J2COM 개발

6.4.1 COM Registration/Unregistration

COM 을 J2COM 을 통해서 사용하기 위해서 개발 과정에 유의할 사항은 특별히 존재하지 않는다. COM 의 개발 및 컴파일이 끝나면 이를 윈도우 레지스트리에 등록한다.

```
Regsvr32 Email.dll
```

"/u" 옵션을 사용해서 등록된 것을 해제 한다.

```
Regsvr32 /u Email.dll
```

6.4.2 Java Client (Local Invocation)

일반적으로 로컬에서 COM 을 호출하는 경우 tool(j2comGen)로 생성한 인터페이스를 사용하여 직접 COM 인스턴스를 제어하는 것이 보통이다.

앞서 설명한 대로 COM Manager 를 통한 메소드 호출은 각각의 호출에 대하여 인스턴스를 새로 생성하게 되므로 직접 인스턴스를 이용하여 메소드 호출 하는 것은 불가능 함을 기억해 두기 바란다.

다음은 j2comGen tool 을 이용하여 Java 인터페이스를 생성하는 과정이다.

```
j2comGen –l <TypeLibraryPath> -d <TargetDirectory>
         –p <PackageName> –v <verbose> –c <javaCapitalize>
         –o <OptionalParameterOverloading>
```

- **TypeLibraryPath** [(required) default : null]: Type Library (idl) 혹은 이를 포함하는 dll/olb file 을 지정한다.
- **TargetDirectory** [(required) default : null]: 생성된 인터페이스들이 위치할 디렉토리 이름이다.
- **PackageName** [(required) default : null]: 생성된 인터페이스들의 패키지 이름이다. 인터페이스 파일들은 <TargetDirectory>/src/<PackageName> 디렉토리 아래에 생성된다.
- **verbose** [(optional) 1 | 0]: 인터페이스 생성 과정에 대한 로그를 출력한다.
- **javaCapitalize** [(optional) 1 | 0]: 일반적인 Java 방식대로 메소드의 첫 글자를 소문자로 치환한다.
- **OptionalParameterOverloading** [(optional) 1 | 0]: [optional] 로 설정된 parameter 에 대해서도 메소드를 생성한다.

마지막으로 지정한 세가지 옵션값은 윈도우 레지스트리에 저장되며 다음 j2comGen 이 실행될때 기본적으로 적용된다.

아래 샘플 예제는 j2comGen tool 을 사용해서 만든 인터페이스이다.

```
package email;
import com.tmax.J2COM.*;
import com.tmax.J2COM.constant.*;
import com.tmax.J2COM.io.*;
import java.io.*;

public class InewMail extends DispatchPtr
{
   public InewMail(String progid)throws COMException
   { super(progid);}

   public InewMail(IUnknown other) throws COMException
   { super(other);}

   public InewMail(GUID ClsID)throws COMException
   { super(ClsID);}
```

```
   public void mail(String company, String name) throws
COMException
   {
      invokeN("mail", new Object[] {company, name});
   }
}
```

아래의 샘플 예제는 위의 인터페이스를 사용해서 만든 클라이언트 프로그램이다.

```
package email;
import jeus.com.j2com.*;

public class emailTest
{
   public static void main(String[] args)
   {
      try{
         InewMail newMail = new InewMail("Email.newMail");
         newMail.mail("Tmax", "navis");
      }catch (COMException e) {
         e.printStackTrace();
      }
   }
}
```

```
InewMail newMail = new InewMail("Email.newMail")
```

프로그램 ID “Email.newMail” 에 해당하는 COM 객체에 대한 인터페이스를 생성하여 리턴한다. 다음과 같이 GUID 를 이용하는 방법도 있다.

```
GUID guid_InewMail = new GUID("{16B17112-D1A1-4971-93CC-
5D42D77F95F3}");
InewMail newMail = new InewMail(guid_InewMail);
```

참고: GUID 는 COM 라이브러리 소스파일(헤더파일 및 IDL 파일)을 참조하면 알 수 있으며 윈도우 레지스트리 HKEY_CLASSES_ROOT 에 프로그램 ID 를 키값으로 하여 저장되어 있다.

```
newMail.mail("Tmax", "navis")
```

newMail 인터페이스에 정의 되어 있는 mail 메소드를 호출한다. mail 메소드는 두개의 String 을 parameter 로 받는다. J2COM 은 이 값을 COM 에 넘기며 결과값을 리턴한다.

```
catch (COMException e)
```

COM 사용중의 모든 에러는 COMException 을 발생시킨다.

6.4.3 Java Client (Remote Invocation)

Local Invoke 와 달리 Remote Invoke 는 인터페이스를 사용하지 않고 COM Manager 를 이용하며, 개발자는 COM 이 제공하는 메소드의 이름과 사용되는 파라미터의 내용을 미리 알고 있어야 한다.

COM Manager 를 이용한 클라이언트 프로그램의 예는 다음과 같다.

```
package email;
import jeus.com.j2com.*;
public class emailTest
{
   public static void main(String[] args)
   {
      String[] arg = { "Tmax", "navis" };
      try {
         String progid = "Email.newMail";
         COMManager.invoke("COMServer1", progid, "mail", arg );
      } catch ( ComException e) {
         e.printStackTrace();
      }
   }
}
```

```
COMManager.invoke("COMServer1", progid, "mail", arg )
```

프로그램 ID “Email.newMail” 에 해당하는 COM 객체에 대한 인터페이스를 생성하여 “arg” 파라미터를 “mail” 메소드에 넘겨 결과값을 리턴한다. 파라미터는 반드시 Object 타입의 Array 로 만들어서 넘겨야 한다

앞에서 언급한것과 마찬가지로 GUID 를 이용할 수도 있다.

```
GUID guid_InewMail = new GUID("{16B17112-D1A1-4971-93CC-
5D42D77F95F3}");
COMManager.invoke("COMServer1", guid_InewMail, "mail", arg );
```

6.5 GUID guid_InewMail JEUS J2COM 예제

6.5.1 소개

이 절에서는 다양한 환경에서 Local Invocation 과 Remote Invocation 을 어떻게 개발하는지 예제을 보여준다.

6.5.2 Local Invocation (A)

로컬 영역에서 COM 라이브러리를 호출하여 결과 값을 가져오는 간단한 예이다. COM Manager 를 사용하지 않고 tool(j2comGen) 로 생성된 인터페이스를 이용하여 직접 호출한다.

COM Library (Part)

VC++의 ATL Wizard 로 만든 간단한 COM 라이브러리이다. 출력 상태를 위해 일부 편집하였다. 두 개의 정수 값을 넘겨 받아 두 값의 합을 리턴한다.

<<*Math.idl*>>

```
// Math.idl : IDL source for Math.dll
//
// This file will be processed by the MIDL tool to
// produce the type library (Math.tlb) and marshalling code.

import "oaidl.idl";
import "ocidl.idl";
   [
      object,
      uuid(9DA6E3FD-0A77-489A-82D9-FACDB294A4E5),
      dual,
      helpstring("IArithmetic Interface"),
      pointer_default(unique)
   ]
   interface IArithmetic : IDispatch
   {
      [id(1), helpstring("method Sum")]
      HRESULT Sum(int x, int y, [out, retval] int *sum);
   };

[
   uuid(E318B006-A72E-429E-92BB-4C59CD2D53B9),
   version(1.0),
```

```
    helpstring("Math 1.0 Type Library")
]

library MATHLib
{
    importlib("stdole32.tlb");
    importlib("stdole2.tlb");

    [
        uuid(50354DC9-E6FF-4A88-9CB7-8C1C4BE1E5BE),
        helpstring("Arithmetic Class")
    ]
    coclass Arithmetic
    {
        [default] interface IArithmetic;
    };
};
```

<<Arithmetic.h>>

```
// Arithmetic.h : Declaration of the CArithmetic

#ifndef __ARITHMETIC_H_
#define __ARITHMETIC_H_

#include "resource.h"       // main symbols

/////////////////////////////////////////////////////////////////////////////
// CArithmetic
class ATL_NO_VTABLE CArithmetic :
    public CComObjectRootEx<CComSingleThreadModel>,
    public CComCoClass<CArithmetic, &CLSID_Arithmetic>,
    public IDispatchImpl<IArithmetic, &IID_IArithmetic,
&LIBID_MATHLib>
{
public:
    CArithmetic()
    {
    }
```

```
DECLARE_REGISTRY_RESOURCEID(IDR_ARITHMETIC)

DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CArithmetic)
   COM_INTERFACE_ENTRY(IArithmetic)
   COM_INTERFACE_ENTRY(IDispatch)
END_COM_MAP()

// IArithmetic
public:
   STDMETHOD(Sum)(int x, int y, /*[out retval]*/ int *sum);
};

#endif //__ARITHMETIC_H_
```

<<Arithmetic.cpp>>

```
// Arithmetic.cpp : Implementation of CArithmetic
#include "stdafx.h"
#include "Math.h"
#include "Arithmetic.h"

/////////////////////////////////////////////////////////////////////////
// CArithmetic

STDMETHODIMP CArithmetic::Sum(int x, int y, int *sum)
{
   // TODO: Add your implementation code here
   *sum = x + y;
   return S_OK;
}
```

J2COM Interface

이 인터페이스는 j2comGen tool 에 의해 생성된 파일 이다.

```
package sample;

import jeus.com.j2com.*;
import jeus.com.j2com.constant.*;
```

```
import jeus.com.j2com.io.*;
import java.io.*;

public class IArithmetic extends DispatchPtr
{
   public IArithmetic(String progid) throws COMException
   { super(progid);}

   public IArithmetic(IUnknown other) throws COMException
   { super(other);}

   public IArithmetic(GUID ClsID) throws COMException
   { super(ClsID);}

   public int sum(int x, int y) throws COMException
   {
      return ((Integer)invokeN("Sum", new Object[]{new
Integer(x), new Integer(y)} )).intValue();
   }
}
```

J2COM Client

프로그램 id 를 이용하여 COM 을 호출하는 클라이언트 프로그램이다.

```
package sample;

import jeus.com.j2com.*;

public class SumTester
{
   public static void main(String[] args)
   {
      try {
      IArithmetic sumInterface = new
IArithmetic("Math.Arithmetic");
      int sum = sumInterface.sum(3,4);
      System.out.println("3 + 4 = " + sum);
      } catch ( COMException e) {
         e.printStackTrace();
```

```
        }
    }
}
```

결과

다음과 같이 실행시킨 결과이다.

```
d:\>java –
classpath .;d:\jeus\lib\system\j2com.jar;d:\jeus\lib\system
\jeus.jar sample.SumTester

3 + 4 = 7
```

6.5.3 Local Invocation (B)

MS PowerPoint 라이브러리를 사용하여 Java 클라이언트로 프리젠테이션을 만드는 예이다.

COM Library

D:\Program Files\Microsoft Office\Office 디렉토리에 위치한 MSPPT9.OLB 태그 라이브러리를 사용한다.

J2COM Interface

위의 라이브러리를 j2comGen 을 이용하여 인터페이스 파일을 생성한다. 이때 javaCapitalize 옵션은 false 로 지정하도록 한다

J2COM Client

위에서 생성된 인터페이스를 사용하여 프리젠테이션 페이지를 만드는 Java 클라이언트이다.

```
import ms.*;
import jeus.com.j2com.*;
import jeus.com.j2com.win32.*;

public class PptDemo {

public static void main(String[] args) {
    try {
        Ole32.CoInitialize();
        _Application app = new__Application("PowerPoint.Application");
        app.put("Visible", true);
```

```
      Presentations preses = app.getPresentations();
      _Presentation pres = preses.Add(-1);
      _Slide slide = pres.getSlides().Add(1,1);
      Shapes shapes = slide.getShapes();
      Shape shape = shapes.Item(new Integer(1));
      TextFrame frame = shape.getTextFrame();
      TextRange range = frame.getTextRange();
      range.setText("Welcome to JEUS J2COM");

      slide.getShapes().Item(new Integer(2)).getTextFrame().
getTextRange().setText("J2COM Demonstration");

      Ole32.CoUninitialize();
      }
      catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

결과

아래 그림은 예제를 실행하였을 때 나오는 결과이다[그림 12].

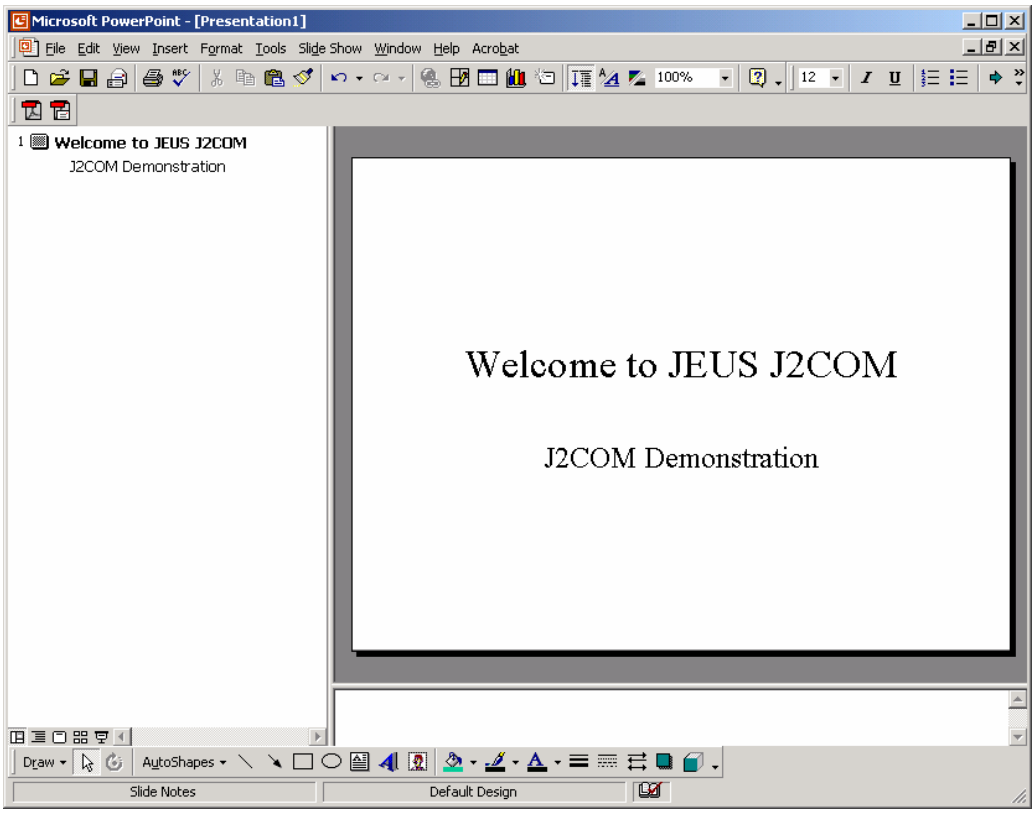

그림 12. MS PowerPoint Slide 예제.

6.5.4 Remote Invocation

Java 클라이언트로부터 두개의 스트링 값을 받아 MS Word 문서를 만드는 예이다

COM Library (Part)

MS Word 테이블 라이브러리를 이용하여 Word 문서를 만들어 주는 간단한 COM 라이브러리 이다. VC++ 6.0 의 ATL Wizard 를 이용하였다.

<<Email.idl>>

```
// Email.idl : IDL source for Email.dll
//
// This file will be processed by the MIDL tool to
// produce the type library (Email.tlb) and marshalling code.

import "oaidl.idl";
import "ocidl.idl";
[
   object,
```

```
    uuid(2434BC8E-56C9-4B47-9884-703300A3AB27),
    dual,
    helpstring("InewMail Interface"),
    pointer_default(unique)
]
interface InewMail : IDispatch
{
    [id(1), helpstring("method mail")]
    HRESULT mail([in] BSTR company, [in] BSTR name);
};

[
    uuid(F757774E-C679-4673-916A-73DD9EED1FE3),
    version(1.0),
    helpstring("Email 1.0 Type Library")
]
library EMAILLib
{
    importlib("stdole32.tlb");
    importlib("stdole2.tlb");

    [
        uuid(16B17112-D1A1-4971-93CC-5D42D77F95F3),
        helpstring("newMail Class")
    ]
    coclass newMail
    {
        [default] interface InewMail;
    };
};
```

<<*newMail.h*>>

```
// newMail.h : Declaration of the CnewMail

#ifndef __NEWMAIL_H_
#define __NEWMAIL_H_

#include "resource.h"       // main symbols
```

```
/////////////////////////////////////////////////////////////
// CnewMail
class ATL_NO_VTABLE CnewMail :
    public CComObjectRootEx<CComSingleThreadModel>,
    public CComCoClass<CnewMail, &CLSID_newMail>,
    public IDispatchImpl<InewMail, &IID_InewMail, &LIBID_EMAILLib>
{
public:
    CnewMail()
    {
    }

DECLARE_REGISTRY_RESOURCEID(IDR_NEWMAIL)

DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CnewMail)
    COM_INTERFACE_ENTRY(InewMail)
    COM_INTERFACE_ENTRY(IDispatch)
END_COM_MAP()

// InewMail
public:
    STDMETHOD(mail)(/*[in]*/ BSTR company, /*[in]*/ BSTR name);
};

#endif //__NEWMAIL_H_
```

<<newMail.cpp>>

```
// newMail.cpp : Implementation of CnewMail
#include "stdafx.h"
#include "Email.h"
#include "newMail.h"
#include "msword9.h"

/////////////////////////////////////////////////////////////
// CnewMail

STDMETHODIMP CnewMail::mail(BSTR company, BSTR name)
```

```
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState())

    COleVariant vOpt(DISP_E_PARAMNOTFOUND, VT_ERROR);

    _Application oApp;
    oApp.CreateDispatch("Word.Application");
    if(!oApp)
    {
        AfxMessageBox("Cannot start Word.");
        return S_FALSE;
    }

    Documents oDocs = oApp.GetDocuments();
    _Document oDoc = oDocs.Add(vOpt, vOpt, vOpt, vOpt);

    //Add the field codes and text to the document
    Selection oSel = oApp.GetSelection();
    Range oRange;

    CHAR msg[128];

    oSel.TypeText("Dear ");

    WideCharToMultiByte(CP_ACP, 0, (LPWSTR)name, -1, msg, 128, NULL, NULL);
    oSel.TypeText(msg);

    oSel.TypeParagraph();
    oSel.TypeParagraph();
    oSel.TypeText("Welcome to Tmax!");
    oSel.TypeParagraph();
    oSel.TypeText("Work hard, play hard and enjoy Life!!");
    oSel.TypeParagraph();
    oSel.TypeParagraph();

    WideCharToMultiByte(CP_ACP, 0, (LPWSTR)company, -1, msg, 128, NULL, NULL);
    oSel.TypeText(msg);

    oApp.SetVisible(TRUE);
```

```
    return S_OK;
}
```

J2COM Client

COM Manager 를 이용하여 원격의 COM 을 호출하는 Java 클라이언트 프로그램이다.

```
package email;

import jeus.com.j2com.*;

public class emailTest
{
    public static void main(String[] args)
    {
        String[] arg = { "Tmax", "navis" };
        String progid = "Email.newMail";

        try {
            COMManager.invoke("COMServer1", progid, "mail", arg );
        } catch ( COMException e) {
            e.printStackTrace();
        }
    }
}
```

J2COM 설정 파일

```
<?xml version="1.0"?>
<j2com-severs>
    <j2com-server>
        <server-name>COMServer1</server-name>
        <ip-address>143.248.1.177</ip-address>
        <port>4444</port>
        <timeout>10000</timeout>
    </j2com-server>
</j2com-severs>
```

결과

예제를 실행 하였을 때 나오는 결과이다[그림 13].

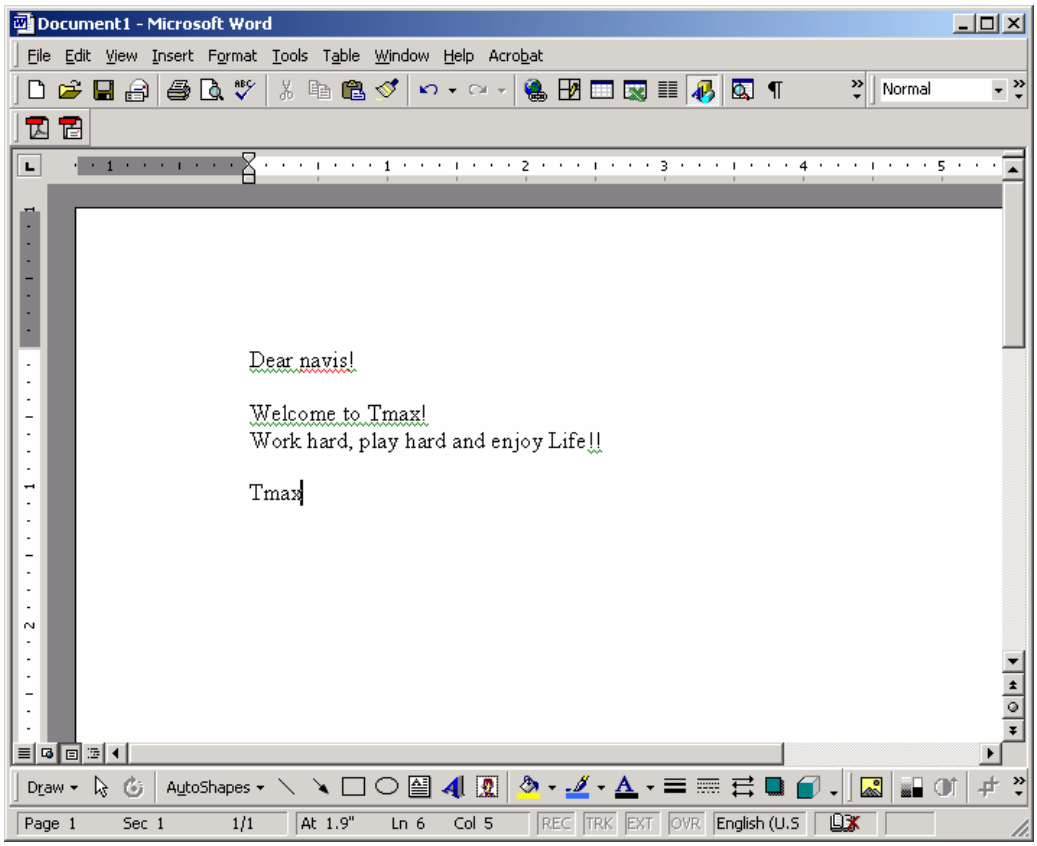

그림 13. MS Word Document 예제.

6.6 결론

이 장에서는 JEUS-COM bridge 에 대해서 설명하였으며 JEUS 와 Java 어플리케이션은 COM 어플리케이션에 대한 연결 방법에 대해서 설명하였다.

7 결론

지금까지 본 매뉴얼에서는 JEUS 의 클라이언트 어플리케이션과 관계있는 중요한 기술들에 대해서 알아보았다. 이 기술들은 아래 5 가지이다.

- JEUS/J2EE Application Client
- JEUS Applet Client
- JEUS JNLP (Java Network Launching Protocol)
- JEUS/J2EE CAS (COM-to-JEUS interoperation)
- JEUS-COM Connector (J2COM: JEUS-to-COM 상호운용)

자세한 설명은 J2EE 공식 사이트인 http://java.sun.com/j2ee 를 참고 하기 바란다.

클라이언트와 관계있는 것 중에는 JEUS Server Management 가 있다. 이것은 JEUS 가 JMX 를 통해서 지원하고 있다. 관련 내용은 JEUS Server 안내서 및 JEUS JMX 안내서에 소개되어 있다.

A jeus-client-dd: JEUS Application Client Deployment Descriptor 레퍼런스

A.1 소개

본 부록의 레퍼런스는 Client Container 의설정 파일인 jeus-client-dd.xml 의 모든 태그에 대해서 설명하고 있다. 이 파일의 schema 파일은 "JEUS_HOME\config\xsds" 디렉토리의 "jeus-client-dd.xsd" 파일이다.

본 레퍼런스는 3 부분으로 나눠져 있다.

1. **XML Schema/XML 트리**: XML 설정 파일의 모든 태그 리스트를 정리했다. 각 노드의 형식은 다음과 같다.
    a. 태그 레퍼런스로 빨리 찾아보기 위해서 각 태그마다 인덱스 번호(예: (11))를 붙여 놓았다. 태그 레퍼런스에서는 이 번호 순서로 설명한다.
    b. schema 에서 정의한 XML 태그명을 <**tag name**> 형식으로 표시한다.
    c. schema 에서 정의한 Cardinality 를 표시한다. "?" = 0 개나 1 개의 element, "+" = 1 개 이상의 element, "*" = 0 개 이상의 element, (기호가 없음) = 정확히 1 개의 element.
    d. 몇몇 태그에는 "P" 문자를 붙여 놓았는데, 해당 태그는 성능에 관계되는 태그라는 것을 뜻한다. 이 태그는 설정을 튜닝할 때 사용된다.
2. **태그 레퍼런스**: 트리에 있는 각 XML 태그를 설명한다.
    a. **Description**: 태그에 대한 간단한 설명
    b. **Value Description**: 입력하는 값과 타입

c. **Value Type**: 값의 데이터 타입. 예) String

d. **Default Value**: 해당 XML 을 사용하지 않았을 때 기본적으로 사용되는 값

e. **Defined values**: 이미 정해져 있는 값

f. **Example**: 해당 XML 태그에 대한 예

g. **Performance Recommendation**: 성능 향상을 위해서 추천하는 값

h. **Child Elements**: 자신의 태그 안에 사용하는 태그

3. **Example XML 파일**: “jeus-client-dd.xml”에 대한 완전한 예제

A.2 XML Schema/XML 트리

```
(1) <jeus-client-dd>
   (2) <module-info>
       (3) <module-name>
       (4) <app-main-class>?
       (5) <app-argument>?
   (6) <system-logging>?
       (7) <level>? P
       (8) <use-parent-handlers>? P
       (9) <filter-class>?
       (10) <handler>?
           (11) <console-handler>
               (12) <name>
               (13) <level>? P
               (14) <encoding>?
               (15) <filter-class>?
           (16) <file-handler>
               (17) <name>
               (18) <level>? P
               (19) <encoding>?
               (20) <filter-class>?
               (21) <file-name>?
               (22) <valid-day>
```

```
            (23) <valid-hour>
            (24) <buffer-size>? P
            (25) <append>? P
        (26) <smtp-handler>
            (27) <name>
            (28) <level>? P
            (29) <encoding>?
            (30) <filter-class>?
            (31) <smtp-host-address>
            (32) <from-address>
            (33) <to-address>
            (34) <cc-address>?
            (35) <bcc-address>?
            (36) <send-for-all-messages>? P
        (37) <socket-handler>
            (38) <name>
            (39) <level>? P
            (40) <encoding>?
            (41) <filter-class>?
            (42) <address>
            (43) <port>
        (44) <user-handler>
            (45) <handler-class>
            (46) <name>
            (47) <level>? P
            (48) <encoding>?
            (49) <filter-class>?
            (50) <handler-property>?
                (51) <property>*
                    (52) <key>
                    (53) <value>
            (54) <formatter-class>? P
            (55) <formatter-property>?
                (56) <property>*
                    (57) <key>
                    (58) <value>
(59) <security-info>?
    (60) <provider-node-name>
    (61) <user>
```

```
    (62) <passwd>
(63) <scheduler>?
    (64) <default-lookup-name>?
    (65) <thread-pool>?
        (66) <min>? P
        (67) <max>? P
        (68) <period>? P
    (69) <job-list>?
        (70) <job>*
            (71) <class-name>
            (72) <name>?
            (73) <description>?
            (74) <begin-time>?
            (75) <end-time>?
            (76) <interval>?
                (77) <millisecond>
                (78) <minutely>
                (79) <hourly>
                (80) <daily>
            (81) <count>? P
(82) <env>*
    (83) <type>
    (84) <name>
    (85) <value>
(86) <ejb-ref>?
    (87) <jndi-info>*
        (88) <ref-name>
        (89) <export-name>
(90) <res-ref>?
    (91) <jndi-info>*
        (92) <ref-name>
        (93) <export-name>
(94) <res-env-ref>?
    (95) <jndi-info>*
        (96) <ref-name>
        (97) <export-name>
(98) <service-ref>?
    (99) <service-client>*
        (100) <service-ref-name>
```

```
(101) <port-info>*
    (102) <service-endpoint-interface>?
    (103) <wsdl-port>?
    (104) <stub-property>*
        (105) <name>
        (106) <value>
    (107) <call-property>*
        (108) <name>
        (109) <value>
    (110) <security>?
        (111) <request-sender>?
            (112) <action-list>
            (113) <password-callback-class>?
            (114) <user>
            (115) <timeToLive>?
            (116) <userNameToken>?
                (117) <passwordType>?
                (118) <userTokenElements>?
            (119) <signature-infos>?
                (120) <signature-info>+
                    (121) <signatureParts>?
                    (122) <keyIdentifier>
                    (123) <keystore>
                        (124) <key-type>
                        (125) <keystore-password>
                        (126) <keystore-filename>
            (127) <encryption-infos>?
                (128) <encryption-info>+
                    (129) <encryptionParts>?
                    (130) <encryptionSymAlgorithm>?
                    (131) <encryptionUser>?
                    (132) <keyIdentifier>
                    (133) <keystore>?
                        (134) <key-type>
                        (135) <keystore-password>
                        (136) <keystore-filename>
                    (137) <embeddedKey>?
                        (138)
<embeddedKeyCallbackClass>
```

```
                        (139) <key-name>
                (140) <response-receiver>?
                    (141) <action-list>
                    (142) <password-callback-class>?
                    (143) <timeToLive>?
                    (144) <decryption>?
                        (145) <keystore>
                            (146) <key-type>
                            (147) <keystore-password>
                            (148) <keystore-filename>
                    (149) <signature-verification>?
                        (150) <keystore>
                            (151) <key-type>
                            (152) <keystore-password>
                            (153) <keystore-filename>
            (154) <service-impl-class>?
            (155) <wsdl-override>?
            (156) <service-qname>?
            (157) <call-property>*
                (158) <name>
                (159) <value>
```

A.3 Element Reference

(1) <jeus-client-dd>

Description 단일 JEUS Client 모듈의 최상위 element. 각각의 jeus-client-dd.xml 파일에는 이 태그가 반드시 존재한다.

Child Elements

(2)module-info
(6)system-logging?
(59)security-info?
(63)scheduler?
(82)env*
(86)ejb-ref?
(90)res-ref?
(94)res-env-ref?
(98)service-ref?

(2) <jeus-client-dd> <module-info>

Description 클라이언트 어플리케이션에 대한 정보

Child Elements
(3)module-name
(4)app-main-class?
(5)app-argument?

(3) <jeus-client-dd> <module-info> **<module-name>**

Description 클라이언트 어플리케이션에 지정되는 module 이름

Value Type token

(4) <jeus-client-dd> <module-info> **<app-main-class>**

Description 클라이언트 어플리케이션의 Main class 의 fully qualified class name 이다.

Value Type token

(5) <jeus-client-dd> <module-info> **<app-argument>**

Description 클라이언트 어플리케이션이 실행될 때 사용되는 application argument 를 지정한다.

Value Type token

(6) <jeus-client-dd> **<system-logging>**

Description 클라이언트 어플리케이션 컨테이너가 사용하는 logger 를 설정한다.

Child Elements
(7)level?
(8)use-parent-handlers?
(9)filter-class?
(10)handler?

(7) <jeus-client-dd> <system-logging> **<level>**

Description logging 의 level 을 설정한다. 각 level 의 의미는 J2SE 의 logging API 의 Level class 를 참고하기 바란다.

Value Type loggingLevelType

Default Value INFO

Defined Value FATAL
SEVERE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

NOTICE
WARNING 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

INFORMATION
INFO 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

DEBUG
FINE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

SEVERE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

WARNING
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

INFO
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

CONFIG
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINER
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINEST
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

(8) <jeus-client-dd> <system-logging> **<use-parent-handlers>**

Description parent logger 의 handler 들을 이 logger 에서도 사용할지를 결정한다.

Value Type boolean

Default Value false

(9) <jeus-client-dd> <system-logging> **<filter-class>**

Description logger 에 지정할 filer class 의 fully qualified class name 을 설정한다.

Value Type token

Example
```
<filter-
class>com.tmax.logging.filter.MyFilter</filter-
class>
```

(10) <jeus-client-dd> <system-logging> **<handler>**

Description logger 에서 사용할 handler 를 설정한다.

Child Elements
```
(11)console-handler
(16)file-handler
(26)smtp-handler
(37)socket-handler
(44)user-handler
```

(11) <jeus-client-dd> <system-logging> <handler> **<console-handler>**

Description logging 을 화면에 남기고자 하는 경우에 사용하는 handler 이다.

Child Elements
```
(12)name
(13)level?
(14)encoding?
(15)filter-class?
```

(12) <jeus-client-dd> <system-logging> <handler> <console-handler> **<name>**

Description handler 의 이름을 설정한다. 이때 이 이름은 하나의 logger 내에서만 unique 하게 설정되면 된다. 이 이름은 tool 등에서 handler 를 지칭할 때 사용한다.

Value Type token

Example `<name>handler1</name>`

(13) <jeus-client-dd> <system-logging> <handler> <console-handler> **<level>**

Description 이 handler 의 level 을 설정한다. logger 를 통과한 message 의 level 이

이 handler level 에 포함될 경우에만 이 handler 에 의해 출력된다.

Value Type loggingLevelType

Default Value FINEST

Defined Value

FATAL
SEVERE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

NOTICE
WARNING 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

INFORMATION
INFO 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

DEBUG
FINE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

SEVERE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

WARNING
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

INFO
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

CONFIG
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINER
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINEST
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

(14) <jeus-client-dd> <system-logging> <handler> <console-handler> **<encoding>**

Description 이 handler 이 message 를 남길때 사용할 encoding 을 설정한다.

Value Type token

(15) <jeus-client-dd> <system-logging> <handler> <console-handler> **<filter-class>**

Description 이 handler 에 지정할 filer class 의 fully qualified class name 을 설정한다.

Value Type token

Example
```
<filter-
class>com.tmax.logging.filter.MyFilter</filter-
class>
```

(16) <jeus-client-dd> <system-logging> <handler> **<file-handler>**

Description logging 을 file 로 출력하고자 하는 경우에 사용하는 handler 이다.

Child Elements
(17)name
(18)level?
(19)encoding?
(20)filter-class?
(21)file-name?
(22)valid-day
(23)valid-hour
(24)buffer-size?
(25)append?

(17) <jeus-client-dd> <system-logging> <handler> <file-handler> **<name>**

Description handler 의 이름을 설정한다. 이때 이 이름은 하나의 logger 내에서만 unique 하게 설정되면 된다. 이 이름은 tool 등에서 handler 를 지칭할 때 사용한다.

Value Type token

Example `<name>handler1</name>`

(18) `<jeus-client-dd> <system-logging> <handler> <file-handler> <level>`

Description 이 handler 의 level 을 설정한다. logger 를 통과한 message 의 level 이 이 handler level 에 포함될 경우에만 이 handler 에 의해 출력된다.

Value Type loggingLevelType

Default Value FINEST

Defined Value

FATAL
SEVERE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

NOTICE
WARNING 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

INFORMATION
INFO 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

DEBUG
FINE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

SEVERE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

WARNING
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

INFO
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

CONFIG
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINE

J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINER
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINEST
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

(19) <jeus-client-dd> <system-logging> <handler> <file-handler> **<encoding>**	
Description	이 handler 이 message 를 남길때 사용할 encoding 을 설정한다.
Value Type	token

(20) <jeus-client-dd> <system-logging> <handler> <file-handler> **<filter-class>**	
Description	이 handler 에 지정할 filer class 의 fully qualified class name 을 설정한다.
Value Type	token
Example	`<filter-class>com.tmax.logging.filter.MyFilter</filter-class>`

(21) <jeus-client-dd> <system-logging> <handler> <file-handler> **<file-name>**	
Description	이 handler 가 사용할 file name 을 설정한다. 만약 user 가 이 설정을 하지 않으면 각 logger 의 default file name 이 사용된다. 각각의 default file name 은 JEUS Server 메뉴얼을 참고하기 바란다.
Value Type	token
Example	`<file-name>C:\logs\mylog.log</file-name>`

(22) <jeus-client-dd> <system-logging> <handler> <file-handler> **<valid-day>**	
Description	이 handler 가 사용하는 file 을 이 설정에서 지정된 기간동안만 사용하고 계속 갱신할 경우에 사용한다. 이 설정은 날짜 단위로 file 을 바꿀때 사용한다. 이 경우 handler 가 사용하는 file 이름 뒤에

	file 이 사용된 날짜가 자동으로 붙게 된다.
Value Description	day
Value Type	off-intType
Value Type Description	기본적으로 non-negative int 형이지만 -1 인 경우에는 설정을 하지 않은게 된다.즉, off 된다.
Example	`<valid-day>1</valid-day>`

`(23) <jeus-client-dd> <system-logging> <handler> <file-handler>` **`<valid-hour>`**

Description	이 handler 가 사용하는 file 을 이 설정에서 지정된 기간동안만 사용하고 계속 갱신할 경우에 사용한다. 이 설정은 시간 단위로 file 을 바꿀때 사용한다. 이 경우 handler 가 사용하는 file 이름 뒤에 file 이 사용된 날짜와 시간이 자동으로 붙게 된다.
Value Description	시간을 나타내며 24 의 약수 + n*24 (n 은 0 이상의 정수)의 값을 가져야 한다.
Value Type	off-intType
Value Type Description	기본적으로 non-negative int 형이지만 -1 인 경우에는 설정을 하지 않은게 된다.즉, off 된다.
Example	`<valid-hour>3</valid-hour>`

`(24) <jeus-client-dd> <system-logging> <handler> <file-handler>` **`<buffer-size>`**

Description	이 handler 가 file 에 출력할때 사용하는 buffer 의 크기를 지정한다.
Value Description	byte 단위이다. [Performance Recommandation]: 이 값이 클수록 file 에 출력되는 message 는 지연되어 출력되지만 logging 성능은 좋아진다.
Value Type	nonNegativeIntType
Value Type Description	0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.
Default Value	1024

`(25) <jeus-client-dd> <system-logging> <handler> <file-handler>` **`<append>`**

Description 이 handler 가 사용하는 file 이 이미 존재하는 경우 file 뒤에 덧붙여 쓸지를 결정한다. false 로 설정되어 있다면 기존의 file 은 제거된다.

Value Type boolean

Default Value true

(26) <jeus-client-dd> <system-logging> <handler> **<smtp-handler>**

Description logging 을 email 로 보내고자 하는 경우에 사용하는 handler 이다. [Performance Recommandation]: logging message 하나가 하나의 email 로 전송되므로 적절한 filter 없이 사용하는 것은 엄청난 양의 email 을 발생시켜 아주 위험하므로 주의를 요한다.

Child Elements
(27)name
(28)level?
(29)encoding?
(30)filter-class?
(31)smtp-host-address
(32)from-address
(33)to-address
(34)cc-address?
(35)bcc-address?
(36)send-for-all-messages?

(27) <jeus-client-dd> <system-logging> <handler> <smtp-handler> **<name>**

Description handler 의 이름을 설정한다. 이때 이 이름은 하나의 logger 내에서만 unique 하게 설정되면 된다. 이 이름은 tool 등에서 handler 를 지칭할 때 사용한다.

Value Type token

Example
```
<name>handler1</name>
```

(28) <jeus-client-dd> <system-logging> <handler> <smtp-handler> **<level>**

Description 이 handler 의 level 을 설정한다. logger 를 통과한 message 의 level 이 이 handler level 에 포함될 경우에만 이 handler 에 의해 출력된다.

Value Type loggingLevelType

Default Value FINEST

Defined Value FATAL

SEVERE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

NOTICE
WARNING 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

INFORMATION
INFO 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

DEBUG
FINE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

SEVERE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

WARNING
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

INFO
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

CONFIG
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINER
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINEST
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

```
(29) <jeus-client-dd> <system-logging> <handler> <smtp-handler>
<encoding>
```

Description 이 handler 이 message 를 남길때 사용할 encoding 을 설정한다.

Value Type token

```
(30) <jeus-client-dd> <system-logging> <handler> <smtp-handler>
<filter-class>
```

Description 이 handler 에 지정할 filer class 의 fully qualified class name 을 설정한다.

Value Type token

Example

```
<filter-
class>com.tmax.logging.filter.MyFilter</filter-
class>
```

```
(31) <jeus-client-dd> <system-logging> <handler> <smtp-handler> <smtp-
host-address>
```

Description email 을 보낼 smtp server 의 주소를 지정한다.

Value Type token

```
(32) <jeus-client-dd> <system-logging> <handler> <smtp-handler> <from-
address>
```

Description email 을 보내는 사람의 address 를 지정한다.

Value Type token

```
(33) <jeus-client-dd> <system-logging> <handler> <smtp-handler> <to-
address>
```

Description email 을 받는 사람의 address 를 지정한다.

Value Type token

```
(34) <jeus-client-dd> <system-logging> <handler> <smtp-handler> <cc-
address>
```

Description email 을 참조로 받는 사람의 address 를 지정한다.

Value Type token

```
(35) <jeus-client-dd> <system-logging> <handler> <smtp-handler> <bcc-
address>
```

Description email 을 숨은 참조로 받는 사람의 address 를 지정한다.

Value Type token

(36) <jeus-client-dd> <system-logging> <handler> <smtp-handler> **<send-for-all-messages>**

Description 이 handler 가 등록한 logger 의 log() method 를 통해 들어온 message 들이 이 handler 로 들어왔을때 이를 email 로 보낼 대상으로 여길지를 설정한다. 만약 false 로 설정되어 있으면 logger 의 특별한 send() method 로 호출된 message 들만 email 로 전송된다. 즉, 처음부터 email 로 보낼 의도로 지정된 message 들만 email 로 전송된다.

Value Type boolean

Default Value false

(37) <jeus-client-dd> <system-logging> <handler> **<socket-handler>**

Description logging 을 지정된 socket 으로 보내고자 하는 경우에 사용하는 handler 이다. [Performance Recommandation]: logging message 하나당 Socket 으로 전송이 되므로 적절한 filter 없이 사용하는 것은 성능 저하를 가져온다.

Child Elements
(38)name
(39)level?
(40)encoding?
(41)filter-class?
(42)address
(43)port

(38) <jeus-client-dd> <system-logging> <handler> <socket-handler> **<name>**

Description handler 의 이름을 설정한다. 이때 이 이름은 하나의 logger 내에서만 unique 하게 설정되면 된다. 이 이름은 tool 등에서 handler 를 지칭할 때 사용한다.

Value Type token

Example
```
<name>handler1</name>
```

(39) <jeus-client-dd> <system-logging> <handler> <socket-handler>

<level>

Description 이 handler 의 level 을 설정한다. logger 를 통과한 message 의 level 이 이 handler level 에 포함될 경우에만 이 handler 에 의해 출력된다.

Value Type loggingLevelType

Default Value FINEST

Defined Value

FATAL
SEVERE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

NOTICE
WARNING 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

INFORMATION
INFO 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

DEBUG
FINE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

SEVERE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

WARNING
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

INFO
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

CONFIG
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINER

J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINEST

J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

(40) <jeus-client-dd> <system-logging> <handler> <socket-handler> **<encoding>**

Description 이 handler 이 message 를 남길때 사용할 encoding 을 설정한다.

Value Type token

(41) <jeus-client-dd> <system-logging> <handler> <socket-handler> **<filter-class>**

Description 이 handler 에 지정할 filer class 의 fully qualified class name 을 설정한다.

Value Type token

Example

```
<filter-
class>com.tmax.logging.filter.MyFilter</filter-
class>
```

(42) <jeus-client-dd> <system-logging> <handler> <socket-handler> **<address>**

Description 이 handler 가 생성될때 message 들을 보낼 곳의 IP address 를 설정한다.

Value Type token

(43) <jeus-client-dd> <system-logging> <handler> <socket-handler> **<port>**

Description 이 handler 가 생성될때 message 들을 보낼 곳의 port 를 설정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

(44) <jeus-client-dd> <system-logging> <handler> **<user-handler>**

Description User 가 J2SE logging API 에 따라 만든 handler 를 사용할 경우의

설정이다.

Child Elements

```
(45)handler-class
(46)name
(47)level?
(48)encoding?
(49)filter-class?
(50)handler-property?
(54)formatter-class?
(55)formatter-property?
```

`(45) <jeus-client-dd> <system-logging> <handler> <user-handler> <handler-class>`

Description user 가 만든 handler 의 fully qualified class name 을 설정한다.이 클래스는 java.util.logging.Handler 를 상속받고 jeus.util.logging.JeusHandler 를 구현해야 한다.

Value Type token

Example

```
<handler-class>com.tmax.logging.handler.MyHandler</handler-class>
```

`(46) <jeus-client-dd> <system-logging> <handler> <user-handler> <name>`

Description handler 의 이름을 설정한다. 이때 이 이름은 하나의 logger 내에서만 unique 하게 설정되면 된다. 이 이름은 tool 등에서 handler 를 지칭할 때 사용한다.

Value Type token

Example

```
<name>handler1</name>
```

`(47) <jeus-client-dd> <system-logging> <handler> <user-handler> <level>`

Description 이 handler 의 level 을 설정한다. logger 를 통과한 message 의 level 이 이 handler level 에 포함될 경우에만 이 handler 에 의해 출력된다.

Value Type loggingLevelType

Default Value FINEST

Defined Value FATAL
SEVERE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로

compatibility 를 위해 지원한다.

NOTICE
WARNING 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

INFORMATION
INFO 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

DEBUG
FINE 에 해당하는 level 이다. JEUS 4.x 대의 logger 의 level 로 compatibility 를 위해 지원한다.

SEVERE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

WARNING
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

INFO
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

CONFIG
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINE
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINER
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

FINEST
J2SE Logging API 의 Level class documentation 을 참고하기 바란다.

(48) <jeus-client-dd> <system-logging> <handler> <user-handler> **<encoding>**

Description 이 handler 이 message 를 남길때 사용할 encoding 을 설정한다.

Value Type token

```
(49) <jeus-client-dd> <system-logging> <handler> <user-handler>
<filter-class>
```

Description 이 handler 에 지정할 filer class 의 fully qualified class name 을 설정한다.

Value Type token

Example
```
<filter-
class>com.tmax.logging.filter.MyFilter</filter-
class>
```

```
(50) <jeus-client-dd> <system-logging> <handler> <user-handler>
<handler-property>
```

Description handler 가 생성될 때 넘겨줄 property 를 설정한다. 이 property 들은 key-value 로 Map 객체에 저장되어 JeusHandler.setProperty() method 를 통해 handler 로 전달된다.

Child Elements
```
(51)property*
```

```
(51) <jeus-client-dd> <system-logging> <handler> <user-handler>
<handler-property> <property>
```

Description handler 등에게 전달할 property 들을 설정한다.

Child Elements
```
(52)key
(53)value
```

```
(52) <jeus-client-dd> <system-logging> <handler> <user-handler>
<handler-property> <property> <key>
```

Description property 의 key 값이다.

Value Type token

```
(53) <jeus-client-dd> <system-logging> <handler> <user-handler>
<handler-property> <property> <value>
```

Description property 의 value 값이다.

Value Type token

```
(54) <jeus-client-dd> <system-logging> <handler> <user-handler>
<formatter-class>
```

Description 이 handler 가 사용할 formatter 의 fully qualified class name 을 설정한다.이 클래스는 java.util.logging.Formatter 를 상속받고 jeus.util.logging.JeusFormatter 를 구현해야 한다.

Value Type token

Default Value jeus.util.logging.SimpleFormatter

Example
```
<formatter-
class>com.tmax.logging.handler.MyHandler</formatt
er-class>
```

```
(55) <jeus-client-dd> <system-logging> <handler> <user-handler>
<formatter-property>
```

Description handler 가 생성될 때 만들어진 formatter 에게 넘겨줄 property 를 설정한다. 이 property 들은 key-value 로 Map 객체에 저장되어 JeusFormatter.setProperty() method 를 통해 formatter 로 전달된다.

Child Elements
```
(56)property*
```

```
(56) <jeus-client-dd> <system-logging> <handler> <user-handler>
<formatter-property> <property>
```

Description handler 등에게 전달할 property 들을 설정한다.

Child Elements
```
(57)key
(58)value
```

```
(57) <jeus-client-dd> <system-logging> <handler> <user-handler>
<formatter-property> <property> <key>
```

Description property 의 key 값이다.

Value Type token

```
(58) <jeus-client-dd> <system-logging> <handler> <user-handler>
<formatter-property> <property> <value>
```

Description property 의 value 값이다.

Value Type token

(59) <jeus-client-dd> **<security-info>**

Description 클라이언트 어플리케이션이 실행될 때 security 인증을 위한 여러가지 정보를 설정한다.

Child Elements
(60)provider-node-name
(61)user
(62)passwd

(60) <jeus-client-dd> <security-info> **<provider-node-name>**

Description 클라이언트 어플리케이션이 실행될 때 security 인증을 수행할 server 를 지정한다.

Value Type token

(61) <jeus-client-dd> <security-info> **<user>**

Description 클라이언트 어플리케이션이 실행될 때 사용하는 security context 의 user name 을 지정한다.

Value Type token

(62) <jeus-client-dd> <security-info> **<passwd>**

Description 클라이언트 어플리케이션이 실행될 때 사용하는 security context 의 password 를 지정한다. 여기에는 base64 로 encoding 된 값이 지정된다.

Value Type token

(63) <jeus-client-dd> **<scheduler>**

Description Client Container 에서 사용할 scheduler 의 설정이다.

Child Elements
(64)default-lookup-name?
(65)thread-pool?
(69)job-list?

(64) <jeus-client-dd> <scheduler> **<default-lookup-name>**

Description jeus_service/Scheduler 로 lookup 할때 return 할 Scheduler 객체가 존재하는 node 나 container 의 이름을 지정한다. 지정이 되어 있지 않으면 lookup 하는 곳에 존재하는 Scheduler 객체가 사용된다.

Value Type token

(65) <jeus-client-dd> <scheduler> **<thread-pool>**

Description	scheduler 에서 multi-thread 로 job 을 실행할때 사용하는 thread pool 을 설정한다.
Child Elements	(66)min? (67)max? (68)period?

```
(66) <jeus-client-dd> <scheduler> <thread-pool> <min>
```

Description	pooling 되는 객체의 최소값을 지정한다.
Value Type	nonNegativeIntType
Value Type Description	0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.
Default Value	2

```
(67) <jeus-client-dd> <scheduler> <thread-pool> <max>
```

Description	pooling 되는 객체의 최대값을 지정한다.
Value Type	nonNegativeIntType
Value Type Description	0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.
Default Value	30

```
(68) <jeus-client-dd> <scheduler> <thread-pool> <period>
```

Description	pooling 되는 객체를 정리하는 시간을 지정한다.
Value Type	long
Default Value	3600000
Performance Recommendation	이 값이 클수록 정리하는 주기가 길어져 server 운영에는 부하가 적게 가해지나 그만큼 메모리가 누수될 수 있으므로 적당한 값으로 지정한다.

```
(69) <jeus-client-dd> <scheduler> <job-list>
```

Description	scheduler 에 등록할 job list 을 지정한다.
Child Elements	(70)job*

```
(70) <jeus-client-dd> <scheduler> <job-list> <job>
```

Description	scheduler 에 등록할 하나의 job 을 지정한다.

Child Elements

(71)class-name
(72)name?
(73)description?
(74)begin-time?
(75)end-time?
(76)interval?
(81)count?

(71) <jeus-client-dd> <scheduler> <job-list> <job> **<class-name>**

Description job 을 수행하는 class 의 fully qualified name 이다.

Value Type token

(72) <jeus-client-dd> <scheduler> <job-list> <job> **<name>**

Description 이 job 의 이름을 지정한다.

Value Type token

(73) <jeus-client-dd> <scheduler> <job-list> <job> **<description>**

Description 이 job 의 설명을 적을 수 있다.

Value Type string

(74) <jeus-client-dd> <scheduler> <job-list> <job> **<begin-time>**

Description 이 job 의 시작 시간을 지정한다.

Value Type dateTime

(75) <jeus-client-dd> <scheduler> <job-list> <job> **<end-time>**

Description 이 job 의 종료 시간을 지정한다.

Value Type dateTime

(76) <jeus-client-dd> <scheduler> <job-list> <job> **<interval>**

Description 이 job 이 수행되는 주기를 지정한다.

Child Elements

(77)millisecond
(78)minutely
(79)hourly
(80)daily

(77) <jeus-client-dd> <scheduler> <job-list> <job> <interval>

<millisecond>

Description 주기를 millisecond 단위로 지정한다.

Value Type long

(78) <jeus-client-dd> <scheduler> <job-list> <job> <interval> **<minutely>**

Description 주기를 분 단위로 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

(79) <jeus-client-dd> <scheduler> <job-list> <job> <interval> **<hourly>**

Description 주기를 시간 단위로 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

(80) <jeus-client-dd> <scheduler> <job-list> <job> <interval> **<daily>**

Description 주기를 날짜 단위로 지정한다.

Value Type nonNegativeIntType

Value Type Description 0 이상의 int type 이다. 즉, int 범위에서 0 이상의 값들을 포함한다.

(81) <jeus-client-dd> <scheduler> <job-list> <job> **<count>**

Description 이 job 이 수행되는 횟수를 지정한다.

Value Type long

Default Value -1

Defined Value -1
수행되는 횟수를 제한하지 않는다.

(82) <jeus-client-dd> **<env>**

Description environment entry 에 대한 정보

Child Elements (83)type
(84)name

(85)value

(83) <jeus-client-dd> <env> **<type>**

Description 환경 변수의 자바 타입.

Value Description 다음의 자바 타입 중 하나를 선택해야 한다. java.lang.Boolean, java.lang.String, java.lang.Integer, java.lang.Double, java.lang.Byte, java.lang.Short, java.lang.Long, java.lang.Float, java.lang.Character.

Value Type token

Example
```
<type>java.lang.Integer</type>
```

(84) <jeus-client-dd> <env> **<name>**

Description 코드에서 사용하는 환경 변수의 이름.

Value Type token

Example
```
<name>minAmount</name>
```

(85) <jeus-client-dd> <env> **<value>**

Description 이 값은 대응하는 wrapper 클래스 생성자의 파라미터로서 사용 된다.

Value Type token

Example
```
<value>100</value>
```

(86) <jeus-client-dd> **<ejb-ref>**

Description EJB reference 에 대한 정보

Child Elements (87)jndi-info*

(87) <jeus-client-dd> <ejb-ref> **<jndi-info>**

Description 이 element 는 코드에서 사용하는 EJB 참조를 실제 EJB JNDI 이름으로 bind 한다. 예를 들면 실제 JNDI 이름이 "ACCEJB"인 account EJB 를 코드상에서 "ejb/account"으로 lookup 할 수 있다.

Child Elements
(88)ref-name
(89)export-name

(88) <jeus-client-dd> <ejb-ref> <jndi-info> **<ref-name>**

Description 이 element 는 소스코드상에서 사용할 수 있는 참조 이름을 선언할 수

있다.

Value Description 실제 JNDI 이름에 bind 될 참조 이름. 이것은 해당하는 J2EE 표준 descriptor element 의 ref-name 에 대응된다.

Value Type token

Example
```
<ref-name>ejb/AccountEJB</ref-name>
```

(89) <jeus-client-dd> <ejb-ref> <jndi-info> **<export-name>**

Description JEUS DD 에 정의된 실제 JNDI 이름.

Value Type token

Example
```
<export-name>ACCEJB</export-name>
```

(90) <jeus-client-dd> **<res-ref>**

Description resource 관련 정보

Child Elements (91)jndi-info*

(91) <jeus-client-dd> <res-ref> **<jndi-info>**

Description 이 element 는 코드에서 사용하는 EJB 참조를 실제 EJB JNDI 이름으로 bind 한다. 예를 들면 실제 JNDI 이름이 "ACCEJB"인 account EJB 를 코드상에서 "ejb/account"으로 lookup 할 수 있다.

Child Elements (92)ref-name
(93)export-name

(92) <jeus-client-dd> <res-ref> <jndi-info> **<ref-name>**

Description 이 element 는 소스코드상에서 사용할 수 있는 참조 이름을 선언할 수 있다.

Value Description 실제 JNDI 이름에 bind 될 참조 이름. 이것은 해당하는 J2EE 표준 descriptor element 의 ref-name 에 대응된다.

Value Type token

Example
```
<ref-name>ejb/AccountEJB</ref-name>
```

(93) <jeus-client-dd> <res-ref> <jndi-info> **<export-name>**

Description JEUS DD 에 정의된 실제 JNDI 이름.

Value Type token

Example `<export-name>ACCEJB</export-name>`

`(94) <jeus-client-dd>` **`<res-env-ref>`**

Description resource environment 관련 정보

Child Elements `(95)jndi-info*`

`(95) <jeus-client-dd> <res-env-ref>` **`<jndi-info>`**

Description 이 element 는 코드에서 사용하는 EJB 참조를 실제 EJB JNDI 이름으로 bind 한다. 예를 들면 실제 JNDI 이름이 "ACCEJB"인 account EJB 를 코드상에서 "ejb/account"으로 lookup 할 수 있다.

Child Elements `(96)ref-name`
`(97)export-name`

`(96) <jeus-client-dd> <res-env-ref> <jndi-info>` **`<ref-name>`**

Description 이 element 는 소스코드상에서 사용할 수 있는 참조 이름을 선언할 수 있다.

Value Description 실제 JNDI 이름에 bind 될 참조 이름. 이것은 해당하는 J2EE 표준 descriptor element 의 ref-name 에 대응된다.

Value Type token

Example `<ref-name>ejb/AccountEJB</ref-name>`

`(97) <jeus-client-dd> <res-env-ref> <jndi-info>` **`<export-name>`**

Description JEUS DD 에 정의된 실제 JNDI 이름.

Value Type token

Example `<export-name>ACCEJB</export-name>`

`(98) <jeus-client-dd>` **`<service-ref>`**

Description JEUS 웹 서비스 클라이언트 설정 문서의 루트 엘리먼트(root element).

Child Elements `(99)service-client*`

`(99) <jeus-client-dd> <service-ref>` **`<service-client>`**

Description 배치되는 웹서비스 클라이언트를 위한 설정들을 표시한다.

Child Elements
```
(100)service-ref-name
(101)port-info*
(154)service-impl-class?
(155)wsdl-override?
(156)service-qname?
(157)call-property*
```

(100) <jeus-client-dd> <service-ref> <service-client> **<service-ref-name>**

Description WSDL 파일에서 관련된 웹서비스 엔드포인트 이름이다. <service-ref-name>은 표준 배치 서술자 web.xml 혹은 ejb-jar.xml 의 <service-ref-name>에 상응한다.

Value Type token

(101) <jeus-client-dd> <service-ref> <service-client> **<port-info>**

Description 배치되는 웹서비스 클라이언트가 호출하는 웹서비스 포트 정보를 표시한다.

Child Elements
```
(102)service-endpoint-interface?
(103)wsdl-port?
(104)stub-property*
(107)call-property*
(110)security?
```

(102) <jeus-client-dd> <service-ref> <service-client> <port-info> **<service-endpoint-interface>**

Description WSDL port 의 서비스 엔드포인드 인터페이스를 나타내는 클래스를 표시한다. <service-ref>에서 <port-component-ref>의 <service-endpoint-interface>에 상응한다.

Value Type token

(103) <jeus-client-dd> <service-ref> <service-client> <port-info> **<wsdl-port>**

Description <port-info>와 연결된 WSDL port 정의를 표시한다.

Value Type QName

(104) <jeus-client-dd> <service-ref> <service-client> <port-info> **<stub-property>**

Description 특정 port 에서 사용하는 javax.xml.rpc.Stub 객체에 설정하는 property 들을 표시한다.

Child Elements (105)name
(106)value

(105) <jeus-client-dd> <service-ref> <service-client> <port-info> <stub-property> **<name>**

Description javax.xml.rpc.Stub 또는 javax.xml.rpc.Call 에 프로퍼티를 설정하기 위한 키(key) 이름을 나타낸다.

Value Type string

(106) <jeus-client-dd> <service-ref> <service-client> <port-info> <stub-property> **<value>**

Description javax.xml.rpc.Stub 또는 javax.xml.rpc.Call 에 프로퍼티를 설정하기 위한 키(key)에 상응하는 값(value)이다.

Value Type string

(107) <jeus-client-dd> <service-ref> <service-client> <port-info> **<call-property>**

Description 특정 port 에서 사용하는 javax.xml.rpc.Call 객체에 설정하는 property 들을 표시한다.

Child Elements (108)name
(109)value

(108) <jeus-client-dd> <service-ref> <service-client> <port-info> <call-property> **<name>**

Description javax.xml.rpc.Stub 또는 javax.xml.rpc.Call 에 프로퍼티를 설정하기 위한 키(key) 이름을 나타낸다.

Value Type string

(109) <jeus-client-dd> <service-ref> <service-client> <port-info> <call-property> **<value>**

Description javax.xml.rpc.Stub 또는 javax.xml.rpc.Call 에 프로퍼티를 설정하기

위한 키(key)에 상응하는 값(value)이다.

Value Type string

```
(110) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security>
```

Description 웹서비스의 보안(WS-Security)을 위한 웹 서비스 클 라이언트 설정이다.

Child Elements
```
(111)request-sender?
(140)response-receiver?
```

```
(111) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <request-sender>
```

Description 웹서비스를 호출하는 메시지에 보안을 적용하기 위한 설정이 다.

Child Elements
```
(112)action-list
(113)password-callback-class?
(114)user
(115)timeToLive?
(116)userNameToken?
(119)signature-infos?
(127)encryption-infos?
```

```
(112) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <request-sender> <action-list>
```

Description 어떤 보안을 적용할 것인지를 String 으로 나열한다. Timestamp, Encrypt, Signature, UsernameToken 이 들어갈수 있다. 각각의 항목은 공백으로 분리한다.(예:UsernameToken Signature Encrypt)

Value Type string

```
(113) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <request-sender> <password-callback-class>
```

Description 패스워드를 설정하는 콜백 클래스의 풀 패키지 명이다.

Value Type string

```
(114) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <request-sender> <user>
```

Description UsernameToken 에 들어갈 이름과 서명에 들어갈 키의 별칭 을

설정한다.

Value Type string

```
(115) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <request-sender> <timeToLive>
```

Description 보내게 될 메시지의 유효기간을 초 단위로 설정한다. 기본값 은 300 초이다.

Value Type string

```
(116) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <request-sender> <userNameToken>
```

Description UsernameToken 을 설정한다.

Child Elements
```
(117)passwordType?
(118)userTokenElements?
```

```
(117) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <request-sender> <userNameToken> <passwordType>
```

Description UsernameToken 에 사용될 패스워드의 타입 설정이다. "PasswordDigest" 혹은 "PasswordText"를 사용할 수 있다.

Value Type passwordTypeType

Defined Value
PasswordDigest
UsernameToken 에 설정되는 암호가 base64 encoding 된 상태 로 메시지에 포함된다.

PasswordText
UsernameToken 에 설정되는 암호가 평이한 텍스트로 메시지에 포함된다.

```
(118) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <request-sender> <userNameToken> <userTokenElements>
```

Description UsernameToken 에 추가될 엘리먼트의 리스트이다. 각 항목은 공백으로 분리된다. "nonce" 혹은 "created"가 사용될 수 있다. passwordType 이 "PasswordText"일 경우에 사용가능하다.

Value Type string

```
(119) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <request-sender> <signature-infos>
```

Description 메시지에 서명을 하기 위한 설정이다.

Child Elements (120)signature-info+

```
(120) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <request-sender> <signature-infos> <signature-info>
```

Description 메시지의 서명을 위한 설정히다. 복수 설정이 가능하다.

Child Elements
(121)signatureParts?
(122)keyIdentifier
(123)keystore

```
(121) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <request-sender> <signature-infos> <signature-info>
<signatureParts>
```

Description 메시지의 특정 부분을 서명하고자 할 때 사용한다.
"{}{http://schemas.xmlsoap.org/soap/envelope/}Body; Token"과 같은 방식으로 열거할 수 있다. 기본적으로 설정하지 않았을 경우에는 SOAP 몸체 전체를 서명하게 되어 있다.

Value Type string

```
(122) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <request-sender> <signature-infos> <signature-info>
<keyIdentifier>
```

Description 서명에 사용될 키의 정보를 표현하는 방식이다. IssuerSerial, DirectReference, SKIKeyIdentifier, X509KeyIdentifier 중의 하나를 사용한다.

Value Type sigKeyIdentifierType

Defined Value
IssuerSerial
X509 인증서의 발급 번호를 메시지에 포함하여 서명을 검증하 기 위한 인증서를 지정한다.

DirectReference
X509 인증서를 메시지에 포함하고 그것을 메시지 내부에서 참조하는 방식이다.

SKIKeyIdentifier
Subject Key Identification 방식이다. X509 인증서의 버전이 3 이상이어야 한다.

X509KeyIdentifier
메시지에 X509 인증서를 포함하고 서명 검증을 위해 사용하도 록 한다.

```
(123) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <request-sender> <signature-infos> <signature-info>
<keystore>
```

Description 메시지의 서명을 위한 개인키를 저장하고 있는 키스토어의 설정이다.

Child Elements
```
(124)key-type
(125)keystore-password
(126)keystore-filename
```

```
(124) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <request-sender> <signature-infos> <signature-info>
<keystore> <key-type>
```

Description 키 스토어에 저장되는 키의 타입이다. (JKS 혹은 pkcs12)

Value Type string

```
(125) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <request-sender> <signature-infos> <signature-info>
<keystore> <keystore-password>
```

Description 키 스토어에 접근하기 위한 암호 설정이다.

Value Type string

```
(126) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <request-sender> <signature-infos> <signature-info>
<keystore> <keystore-filename>
```

Description 키 스토어의 파일 이름이다. 파일 이름을 적거나 절대 경로 를 포함하는 파일 이름을 적는다. 파일 이름만 적을 경우, 클래스 경로에서 찾게 된다.

Value Type string

```
(127) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <request-sender> <encryption-infos>
```

Description 메시지를 암호화 하기 위한 설정이다.

Child Elements (128)encryption-info+

```
(128) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <request-sender> <encryption-infos> <encryption-info>
```

Description 배치되는 웹서비스 클라이언트를 위한 설정들을 표시한다.

Child Elements
(129)encryptionParts?
(130)encryptionSymAlgorithm?
(131)encryptionUser?
(132)keyIdentifier
(133)keystore?
(137)embeddedKey?

```
(129) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <request-sender> <encryption-infos> <encryption-info>
<encryptionParts>
```

Description 특정 부분을 암호화 하기 위한 설정이다.
"{mode}{ns}{localname};{mode}{ns}{localname};…" 과 같은 형식이다.
기본 mode 값은 content 이다.
예:{Content}{http://example.org/payment}CreCard;{Element}{}UserName

Value Type string

```
(130) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <request-sender> <encryption-infos> <encryption-info>
<encryptionSymAlgorithm>
```

Description 암호화에 사용하는 알고리즘이다. AES_128, AES_256, TRIPLE_DES, AES_192 를 지원한다.

Value Type string

```
(131) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <request-sender> <encryption-infos> <encryption-info>
<encryptionUser>
```

Description 암호화에 사용되는 키의 별칭이다.

Value Type string

```
(132) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <request-sender> <encryption-infos> <encryption-info>
<keyIdentifier>
```

Description 암호화에 사용될 키의 정보를 표현하는 방식이다. IssuerSerial, DirectReference, SKIKeyIdentifier, X509KeyIdentifier EmbeddedKeyName 중의 하나를 사용한다.

Value Type encKeyIdentifierType

Defined Value

IssuerSerial
X509 인증서의 발급 번호를 메시지에 포함하여 서명을 검증하 기 위한 인증서를 지정한다.

DirectReference
X509 인증서를 메시지에 포함하고 그것을 메시지 내부에서 참조하는 방식이다.

SKIKeyIdentifier
Subject Key Identification 방식이다. X509 인증서의 버전이 3 이상이어야 한다.

X509KeyIdentifier
메시지에 암호화에 사용된 X509 인증서를 포함한다.

EmbeddedKeyName
웹서비스와 웹서비스 클라이언트가 공유하는 세션키를 사용할 때 사용한다. 웹서비스와 클라이언트는 키의 이름만을 주고 받음으로써 어떤 키를 사용했는지를 알 수 있다.

```
(133) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <request-sender> <encryption-infos> <encryption-info>
<keystore>
```

Description 암호화에 사용될 키의 저장소 설정이다.

Child Elements (134)key-type

(135)keystore-password
(136)keystore-filename

```
(134) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <request-sender> <encryption-infos> <encryption-info>
<keystore> <key-type>
```

Description 키 스토어에 저장되는 키의 타입이다. (JKS 혹은 pkcs12)

Value Type string

```
(135) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <request-sender> <encryption-infos> <encryption-info>
<keystore> <keystore-password>
```

Description 키 스토어에 접근하기 위한 암호 설정이다.

Value Type string

```
(136) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <request-sender> <encryption-infos> <encryption-info>
<keystore> <keystore-filename>
```

Description 키 스토어의 파일 이름이다. 파일 이름을 적거나 절대 경로 를 포함하는 파일 이름을 적는다. 파일 이름만 적을 경우, 클래스 경로에서 찾게 된다.

Value Type string

```
(137) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <request-sender> <encryption-infos> <encryption-info>
<embeddedKey>
```

Description 웹서비스와 웹서비스 클라이언트가 공유하고 있는 키를 설정한다. keyIdentifier 가 "EmbeddedKeyName"으로 설정되어야 사용할 수 있다.

Child Elements (138)embeddedKeyCallbackClass
(139)key-name

```
(138) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <request-sender> <encryption-infos> <encryption-info>
<embeddedKey> <embeddedKeyCallbackClass>
```

Description 세션 키를 사용하려 할 경우, 키의 바이트 정보를 가지고 있 는 콜백

클래스를 설정한다.

Value Type string

```
(139) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <request-sender> <encryption-infos> <encryption-info>
<embeddedKey> <key-name>
```

Description 세션 키의 이름을 설정한다.

Value Type string

```
(140) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <response-receiver>
```

Description 웹서비스 응답 메시지가 보안 적용이 되어있을 경우, 처리하 기 위한 설정이다.

Child Elements
```
(141)action-list
(142)password-callback-class?
(143)timeToLive?
(144)decryption?
(149)signature-verification?
```

```
(141) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <response-receiver> <action-list>
```

Description 받게 되는 메시지가 어떤 보안이 적용되어 있어야 하는지 설정한다. Timestamp, Encrypt, Signature, UsernameToken 이 들어갈수 있다. 각각의 항목은 공백으로 분리한다.(예:UsernameToken Signature Encrypt)

Value Type string

```
(142) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <response-receiver> <password-callback-class>
```

Description 패스워드 콜백 클래스의 이름을 풀 패키지 이름으로 입력한다.

Value Type string

```
(143) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <response-receiver> <timeToLive>
```

Description 받은 메시지의 유효기간을 설정한다(초단위). 기본값은 생성 시간으로 부터 300 초 동안이다.

Value Type string

```
(144) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <response-receiver> <decryption>
```

Description 받는 메시지의 암호화 된 부분을 해독하기 위한 설정이다.

Child Elements (145)keystore

```
(145) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <response-receiver> <decryption> <keystore>
```

Description 메시지의 암호를 해독하기 위한 키 스토어의 설정이다.

Child Elements (146)key-type
(147)keystore-password
(148)keystore-filename

```
(146) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <response-receiver> <decryption> <keystore> <key-type>
```

Description 키 스토어에 저장되는 키의 타입이다. (JKS 혹은 pkcs12)

Value Type string

```
(147) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <response-receiver> <decryption> <keystore> <keystore-
password>
```

Description 키 스토어에 접근하기 위한 암호 설정이다.

Value Type string

```
(148) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <response-receiver> <decryption> <keystore> <keystore-
filename>
```

Description 키 스토어의 파일 이름이다. 파일 이름을 적거나 절대 경로 를 포함하는 파일 이름을 적는다. 파일 이름만 적을 경우, 클래스 경로에서 찾게 된다.

Value Type string

```
(149) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <response-receiver> <signature-verification>
```

Description 받는 메시지의 서명을 검증하기 위한 설정이다.

Child Elements (150)keystore

```
(150) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <response-receiver> <signature-verification> <keystore>
```

Description 서명을 검증하기 위한 키 스토어 설정이다.

Child Elements
(151)key-type
(152)keystore-password
(153)keystore-filename

```
(151) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <response-receiver> <signature-verification> <keystore>
<key-type>
```

Description 키 스토어에 저장되는 키의 타입이다. (JKS 혹은 pkcs12)

Value Type string

```
(152) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <response-receiver> <signature-verification> <keystore>
<keystore-password>
```

Description 키 스토어에 접근하기 위한 암호 설정이다.

Value Type string

```
(153) <jeus-client-dd> <service-ref> <service-client> <port-info>
<security> <response-receiver> <signature-verification> <keystore>
<keystore-filename>
```

Description 키 스토어의 파일 이름이다. 파일 이름을 적거나 절대 경로 를 포함하는 파일 이름을 적는다. 파일 이름만 적을 경우, 클래스 경로에서 찾게 된다.

Value Type string

```
(154) <jeus-client-dd> <service-ref> <service-client> <service-impl-
class>
```

Description 웹서비스 클라이언트를 위한 서비스 구현체를 표시한다. 배치 시에 자동 생성되므로 웹서비스 배치자가 설정할 필요가 없다.

Value Type token

(155) <jeus-client-dd> <service-ref> <service-client> **<wsdl-override>**

Description <service-ref>의 <wsdl-file>을 대체하기 위한 WSDL 파일의 위치를 표시한다. 표시된 위치는 유효한 URL 이어야 한다.

Value Type string

(156) <jeus-client-dd> <service-ref> <service-client> **<service-qname>**

Description WSDL 의 WSDL service 정의를 표시한다.

Value Type QName

(157) <jeus-client-dd> <service-ref> <service-client> **<call-property>**

Description WSDL service 에서 사용하는 모든 javax.xml.rpc.Call 객체에 설정하는 property 들을 표시한다.

Child Elements (158)name
(159)value

(158) <jeus-client-dd> <service-ref> <service-client> <call-property> **<name>**

Description javax.xml.rpc.Stub 또는 javax.xml.rpc.Call 에 프로퍼티를 설정하기 위한 키(key) 이름을 나타낸다.

Value Type string

(159) <jeus-client-dd> <service-ref> <service-client> <call-property> **<value>**

Description javax.xml.rpc.Stub 또는 javax.xml.rpc.Call 에 프로퍼티를 설정하기 위한 키(key)에 상응하는 값(value)이다.

Value Type string

A.4 jeus-client-dd.xml 예제 파일

<<*jeus-client-dd.xml*>>

```
<?xml version="1.0" encoding="UTF-8"?>
<jeus-client-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <module-info>
```

```
    <module-name>client_application1</module-name>
    <app-main-class>SalaryClient</app-main-class>
    <app-argument>125000</app-argument>
</module-info>
<system-logging>
    <level>FINE</level>
    <handler>
        <file-handler>
            <name>fileHandler</name>
            <level>FINE</level>
        </file-handler>
    </handler>
</system-logging>
<security-info>
    <provider-node-name>jeus_node1</provider-node-name>
    <user>john</user>
    <password>MTExMTExMQ==</password>
</security-info>
<scheduler>true</scheduler>
<env>
    <name>year</name>
    <type>java.lang.Integer</type>
    <value>2002</value>
</env>
<ejb-ref>
    <jndi-info>
        <ref-name>count</ref-name>
        <export-name>count_bean</export-name>
    </jndi-info>
</ejb-ref>
<res-ref>
    <jndi-info>
        <ref-name>datasource</ref-name>
        <export-name>Oracle_DataSource</export-name>
    </jndi-info>
</res-ref>
<res-env-ref>
    <jndi-info>
        <ref-name>jms/SalaryInfo</ref-name>
```

```
            <export-name>jms/salary_info_queue1</export-name>
        </jndi-info>
    </res-env-ref>
</jeus-client-dd>
```

B JNLPMain.xml: JNLP Resource 설정 파일 레퍼런스

B.1 소개

본 부록의 레퍼런스는 JEUS JNLP 의 설정 파일인 JNLPMain.xml 의 모든 태그에 대해서 설명하고 있다. 이 파일의 schema 파일은 “JEUS_HOME\config\xsds” 디렉토리의 “jnlp-main.xsd” 파일이다.

본 레퍼런스는 3 부분으로 나눠져 있다.

4. **XML Schema/XML 트리**: XML 설정 파일의 모든 태그 리스트를 정리했다. 각 노드의 형식은 다음과 같다.
    a. 태그 레퍼런스로 빨리 찾아보기 위해서 각 태그마다 인덱스 번호(예: (11))를 붙여 놓았다. 태그 레퍼런스에서는 이 번호 순서로 설명한다.
    b. schema 에서 정의한 XML 태그명을 <**tag name**> 형식으로 표시한다.
    c. schema 에서 정의한 Cardinality 를 표시한다. “?” = 0 개나 1 개의 element, “+” = 1 개 이상의 element, “*” = 0 개 이상의 element, (기호가 없음) = 정확히 1 개의 element.
    d. 몇몇 태그에는 “P” 문자를 붙여 놓았는데, 해당 태그는 성능에 관계되는 태그라는 것을 뜻한다. 이 태그는 설정을 튜닝할 때 사용된다.
5. **태그 레퍼런스**: 트리에 있는 각 XML 태그를 설명한다.
    a. **Description**: 태그에 대한 간단한 설명
    b. **Value Description**: 입력하는 값과 타입
    c. **Value Type**: 값의 데이터 타입. 예) String

d. **Default Value**: 해당 XML 을 사용하지 않았을 때 기본적으로 사용되는 값

e. **Defined values**: 이미 정해져 있는 값

f. **Example**: 해당 XML 태그에 대한 예

g. **Performance Recommendation**: 성능 향상을 위해서 추천하는 값

h. **Child Elements**: 자신의 태그 안에 사용하는 태그

6. **Example XML 파일**: "JNLPMain.xml"에 대한 완전한 예제

B.2 XML Schema/XML 트리

```
(1) <jnlp-resource-config>
     (2) <jnlp-resource>+
           (3) <rsc-name>
           (4) <rsc-path>
           (5) <os-name>?
           (6) <arch>?
           (7) <locale>?
           (8) <version>?
```

B.3 Element Reference

(1) **<jnlp-resource-config>**	
Description	JNLPMain.xml 의 설정항목중 최상위 element. JNLPServer 에서 제공하는 resource 에 대한 모든 설정을 이 element 아래에 작성한다. JNLPMain.xml 파일에는 이 element 가 반드시 존재한다.
Child Elements	(2)jnlp-resource+

(2) <jnlp-resource-config> **<jnlp-resource>**	
Description	JNLPServer 에서 제공하는 resource 에 대한 정보를 제공한다. resource 에 대한 상세한 정보는 이 element 아래에 작성된다.
Child Elements	(3)rsc-name (4)rsc-path (5)os-name? (6)arch? (7)locale? (8)version?

(3) <jnlp-resource-config> <jnlp-resource> <rsc-name>

Description JNLPServer 에서 제공하는 resource 의 이름. client 에서는 해당 resource 를 요청할 경우 rsc-name 을 URL 에 사용한다. 예) http://localhost:9744/jnlp/HelloJeus/rsc-name

Value Type token

(4) <jnlp-resource-config> <jnlp-resource> <rsc-path>

Description JNLPServer 에서 제공하는 resource 파일에 대한 실제 물리적인 경로정보를 지정한다. 예) <rsc-path>c:₩jeus₩webhome₩client_home₩HelloJeus₩HelloJeus.jar</rsc-path>

Value Type token

(5) <jnlp-resource-config> <jnlp-resource> <os-name>

Description resource 가 고려해야할 os 를 지정한다. 이 값이 JVM 의 시스템 파라미터 os.name 의 prefix 라면 resource 를 사용할 수 있다. 만약 이 속성이 지정되어 있지 않다면 모든 OS 에서 사용할 수 있다. 이 값은 Java 의 시스템 프로퍼티인 os.name 과 일치해야 한다. <os-name>, <arch>, <locale> 값을 여러 개 넣을 때는 공백으로 구별한다. 만약 공백이 값의 일부로 사용되면 공백 앞에 "₩"를 넣는다. 예를 들면 "Windows₩ 95 Windows₩ 2000"는 "Windows 95"와 "Windows 2000"을 지원한다는 의미이다. 여기서 "95"와 "2000"앞에 공백을 넣기 위해서 "₩"가 사용되었다.

Value Type token

(6) <jnlp-resource-config> <jnlp-resource> <arch>

Description resource 항목이 사용되어야 하는 하드웨어 구조를 지정한다. 이 속성 값이 시스템 파라미터 os.arch 의 시작부분과 같으면 resource 항목은 사용할 수 있다. 만약 이 속성이 지정되어 있지 않다면 모든 하드웨어에서 사용할 수 있다. 이 값은 <os-name>과 같은 형식으로 기술한다.

Value Type token

(7) <jnlp-resource-config> <jnlp-resource> <locale>

Description 항목이 특정 지역에서 의존하는 resource 임을 지정한다. 만약 이 값이 지정되어 있으면 resource 항목은 지정된 locale 정보를 기본값으로 가지고 있는 JNLP 클라이언트에서만 사용할 수 있다. 이 값이 지정되어 있지 않으면 모든 클라이언트에서 사용 가능하다. 이 값은 <os-name>과 같은 형식으로 기술한다.

Value Type token

(8) <jnlp-resource-config> <jnlp-resource> <version>

Description 이 자원의 버전을 지정한다. 클라이언트 시스템이 원하는 버전과 일치할 경우에만 이 자원을 사용할 수 있고 지정된 값이 없는 경우 모든 클라이언트 시스템에서 사용 가능하다.

Value Type token

B.4 Sample JNLPMain.xml File

<<JNLPMain.xml>>

```
<?xml version="1.0" encoding="UTF-8"?>
<jnlp-resource-config
xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
   <jnlp-resource>
      <rsc-name>HelloJeus.jar</rsc-name>
      <rsc-path>
          c:\jeus\webhome\client_home\HelloJeus\HelloJeus.jar
      </rsc-path>
      <os-name>SunOS</os-name>
      <arch>sparc</arch>
      <locale>en_US</locale>
   </jnlp-resource>
</jnlp-resource-config>
```

색 인