Tomcat On Linux

주효식nogadax@kldp.org 2차 수정일 : 2000,10,10 / 3차 수정일 : 2000,11,11 / 4차 수정일 : 2001,01/11

Tomcat은 **The Jakarta Project**의 산물이며 홈페이지는 *http://jakarta.apache.org* <http://jakarta.apache.org> 이다. **Jakarta Project 의 목표**는 오픈소스화되어 개발된 Java 플랫폼을 기반으로 하여 상용 품질의 서버 솔루션(Server Solution) 을 제공하기 위함이다. Tomcat 은 **Java Servlet 2.2** 와 **JavaServer Page 1.1** 의 명세를 구현한 형태이며 이를 "Servlet Container" 라고 한다. Tomcat은 기타의 Web Server 와 개발 툴에서 사용될 수 있을 뿐만아니라 Apache 에서 Plug-In 형태로 사용될 수 있다. 기타 Tomcat 은 그 자체가 Web Server 처럼 사용될 수 있으나 아직 안정적이지 못해 권장되지 않으며 기존의 Web Server와 같이 사용하길 권장한다. Tomcat의 개발진은 SUN,IBM 및 Italio 같은 신생 회사와 개인의 개발자들이며 개발에 관심있는 그 누구도 동참할 수 있으며 관심있는 분들은 Jakarata 홈을 찾기바란다. 기타 XML 관련 패키지인 cocoon 의 홈페이지는 *http://xml.apache.org* <http://xml.apache.org> 이다. cocoon 은 XML 파일을 서버측에서 처리하여 이를 브라우져에 보여주는 패키지로 설치도 간편하므로 Tomcat 설치시에 추가로 설치하면 좋을듯하다.

목 차

제 1 절 Tomcat Overview

Tomcat 3.1 버젼과 달리, 톰캣 3.2 에는 많은 변화가 이루어진 듯 하다. 톰캣 3.1까지는 servlet 및 JSP spec 에 대한 기능의 구현 차원이라면 3.2 버젼부터는 퍼포먼스 튜닝 ,보안관련 보완(SSL등..), 톰캣 자체에서의 쓰레드 풀링 지원 및 mod_jserv 같은 web-server-plugin의 재작성(mod_jk) 등 웹 어플리케이션 서버로의 도약 과정으로 보여진다. 앞으로 나올 4.0 버젼부터는 많은 기대를 걸어도 좋을듯 하다. 더욱 자세한 사항은 Tomcat 관련 문서를 참조하기 바란다.

1.1 Release Notes For Tomcat

Tomcat 3.1

- Servlet 및 JSP spec 기본 구현
- SSL 지원 안됨
- Automatic servlet reloading 지원
- WAR 파일 지원
- Netscape 및 IIS 지원
- 기타..

Tomcat 3.2

- Apache 1.3 버젼 부터 지원
- MS IIS 4.0 이후 부터 지원
- MS Personal Web Server 4.0 버젼부터 지원
- Netscape Enterprise server 3.0 부터 지원
- Tomcat 3.1 이후 많은 버그 수정
- Tomcat 3.1 이후 처음으로 퍼포먼스 튜닝
- 톰캣 자체에서 Threaded Pooling 지원
- mod_jserv의 재작성판 mod_jk 지원
- mod_jserv 및 mod_jk 모두 지원
- mod_jk를 통한 Ajpv13 지원
- 기타..

Tomcat 3.2.1

- 톰캣 3.2 버젼에 기반한 버그 수정판
- 톰캣 3.2 버젼에 대한 보안(security) 관련 업데이트판
- SSL 지원
- WEB-INF 디렉토리내의 JSP 요청 허용 안됨(예: /WEB-INF/dummy.jsp)
- JDBC 예외 처리에 대한 디버깅을 보조하기위해 JDBC 예외에 대한 기록 처리(JDBCRealm)
- 기타..

Tomcat 4.0

- 2001년 1월 현재 개발중
- 톰캣 3.x 버젼과는 매우 다른 구조로 개발중(카탈리나 아키텍쳐)
- Servlet 2.3 및 JSP 1.2 지원
- 기타..

1.2 톰캣의 실행 모드

A. Stand_alone

```
a. 톰캣 혼자서 웹 서버 및 Servlet 컨테이너 기능 모두 처리
   즉, 독자적인 웹 어플리케이션 서버이다.
b. 톰캣이 Stand-alone 일 경우 Static File등에 대한 처리는 CPU 낭비임.
   static file 예 : HTML, Image files
c. Apache 처럼 확장성이(perl,php) 없고 정적화일등의 처리가 견고하지 못함.
d. 그러므로 Apache 같은 웹서버와 병행하여 처리하는 것이 효율적임.
```

B. In-Process

```
a. Apache2.0 과 같은 Multi-Threaded-Server 가 필요함.
b. out-process 에 비해 비교적 빠른 편임.
c. apache등의 웹서버 프로세스 내에서 JVM 등이 실행됨.
   이는 web-server-plugin에 의해 구현됨.
d. JNI 사용
```

C. Out-Process

```
a. TCP/IP소켓을 사용하는 IPC 사용
b. 웹서버의 프로세스밖에서 JVM이 실행
c. 프로토콜로는 Ajpv12 와 Ajpv13 를 사용
```

1.3 Web Server Plugin

* Web-Server-Plugin은 톰캣이 웹서버와 병행하여 사용할 수 있도록 만들어진 모듈

* 클라이어트측의 Servlet 혹은 JSP대한 요청은 웹서버가 Web-Server-plugin 에게 전송

* 또한, Web-Server-Plugin 은 전달받은 메시지를 톰캣에게 전송.

* 즉, Web-Server-Plugin은 웹서버(Apache)와 톰캣사이의 의사소통 당담자임.

* 톰캣3.1 버젼의 web-server-plugin은 mod_jserv.so 만 지원한다.

* 톰캣3.2 버젼의 Web-Server-Plugin은 mod_jserv.so 와 mod_jk.so 등 두가지를 지원한다.

* 통신을 돕기 위해 web-server-plugin가 필요한 것은 포트와 프로토콜(Ajp..)임.

* 아파치 및 톰캣사이에서 위의 포트와 프로토콜로 인해 통신이 가능하며 기타 호스트 정보도 필요.

* 다음은 mod_jserv.so 와 mod_jk.so 의 비교이다.

a. mod_jserv.so

```
-구조가 복잡하다.
-SSL을 지원하는 HTTPS 지원 안됨
-Ajpv12 만 지원
-서블릿 지원 버젼 낮음 ..
-Tomcat 3.1, 3.2에서 모두 지원
-기타..
```

b. mod_jk.so

```
-구조가 복잡하지 않다.
-mod_jserv.so 를 약간만 변경한 것이 아니다.
-mod_jserv 를 완전히 재작성한 것이다.
-아파치, 넷스케이프등을 지원
-아파치 1.3.x 및 아파치 2.x.x 지원
-Tomcat 3.2에서만 지원
-SSL 지원
-Ajpv 12 및 Ajpv13 모두 지원
-기타
```

제 2 절 Tomcat 3.1 Getting Started

자바 기반인 tomcat 은 아파치 진영의 또 다른 산물이다. JSDK(servlet)의 설치 없이 tomcat만의 설치로 servlet과 jsp를 바로 사용할 수 가 있다.

Tomcat은 JSP 환경을 지원하는 Servlet Container 이다. Servlet Container는 사용자를 대신하여 Servlet 을 관리하고 Invoke 하는 실시간 쉘(Runtime Shell)이다.

이는 기존의 Servlet 을 지원하는 Jserv 와는 다르며 Tomcat 과 대표적인 차이는 JSDK 가 필요없다는 것이며 Jserv는 Servlet 2.0 까지만을 지원하나 Tomcat은 Servlet 2.2와 JSP 1.1을 지원한다는 것이다.

Servlet Container은 크게 세가지로 분류되며 다음과 같다.

1.Stand-alone Servlet Container : Tomcat 자체가 자바 기반의 Web Server로 사용된다. 이는 Tomcat 에서 Default mode 이다.

2.In-process Servlet Container : Web Server Plugin과 Java Container를 조합한 구현 형태이며 Apache 등의 Web Server 에 추가하여 사용하는 형태이다. 하나의 프로세스내에 멀티 쓰레드를 구현한 형태이며 이는 좋은 퍼포먼스(performance)를 가지나 하나의 프로세스내에서 각 서블릿이 멀티 쓰레드화되어 구현되므로 쓰레드의 동시 실행 개수가 제한된다(limited Scalability). 구현 형태는 Web Server Plugin이 JVM을 Web Server의 주소(Address) 영역내에 open 한 후 그 주소 영역내에서 JVM 이 실행될 수 있도록 하게 한다. 서블릿을 실행하려는 어떤 요청이 오면 Plugin은 제어권을 받아 Java Container에게 건네준다.

3.**Out-of-process Servlet Container :** In-process Servlet Container 와 비슷하나 JVM 은 Web Server 주소 영역의 외부에서 실행된다. Web Server Plugin 과 Java Container(JVM) 은 IPC 메커니즘(대개 TCP/IP 소켓)을 사용하여 통신한다. 서블릿을 실행하려는 어떤 요청이 들어오면 plugin 은 제어권을 받아 IPC 메카니즘을 사용하여 Java Container(JVM) 에게 건네준다. Out-of-process 에서는 요청에 대한 응답시간(response time) 이 In-process보다는 좋지 않지만 scalability 나 stability 등에서 좋다.

기타 servlet 의 경우 매번 컴파일할때마다 아파치의 재기동없이 바로 실행 결과를 볼 수 있어 개발자에게는 더없이 즐겁다. (하지만 대체로 재컴파일된 서블릿 은 바로 로딩이 안되고 브라우져를 죽였다가 다시 시작하면 되는데 이게 왜이런지 아시는 분은 메일이나 제 홈에 글좀 남겨주세요)

기타 이 문서는 Tomcat에 대한 최소한의 문서이므로 Tomcat 에 대한 다른 문서들도 참조하기 바란다. 가장 적절한 참고 사이트는*Jakarta 의 홈페이지* `<http://jakarta.apache.org>`이며 기타 *http://kldp.org* `<http://kldp.org>` 에서 "Tomcat" 으로 검색을 하여 김상수님의 Tip문서를 찾거나 *http://webdox.co.kr* `<http://webdox.co.kr>` 에서 김민식님과 김세곤님이 함께 쓴 문서를 찾아 참조하기 바란다.

(필자의 홈페이지는 *http://user.chollian.net/~nogadax* `<http://user.chollian.net/~nogadax>` 이고 메일을 통한 질문은 될수 있으면 삼가해주세요. 제 홈의 질문 게시판을 이용해주시기 바랍니다.)

제 3 절 Tomcat 3.1 설치

필자의 테스트환경은 다음과 같다

- Accel Linux 6.1
- postgreSQL 7.0.2
- JDK 1.2.2
- Apache 1.3.12
- Tomcat 3.1
- cocoon 1.8

여기에서 소개하는 설치는 Apache Plugin 형태이다.

3.1 Tomcat 3.1 설치

1. 설치전에 아파치와의 연동을 위해 아파치는 DSO 로 컴파일되어 있어야 한다.

Apache Configure 및 컴파일 예

```
[root@nogadax src]$ ./configure --prefix=/usr/local/apache \
--enable-module=most \
--enable-shared=max
[root@nogadax src]$ make
[root@nogadax src]$ make install
```

2. JDK1.1.x 버젼 이상이 설치되어 있어야 한다.

3. *http://jakarta.apache.org/downloads/binindex.html* `<http://jakarta.apache.org/downloads/binindex.html>` 에서 이미 컴파일된 **바이너리 패키지(jakarta-tomcat.tar.gz)**를 다운받고 **mod_jserv.so** 를 다운받는다.

4. 적당한 디렉토리에 바이너리 패키지의 압축을 풀면 설치 완료이다.

3.2 설정하기

1. 다운받은**mod_jserv.so** 를 (아파치 디렉토리의) **apache/libexec** 디렉토리에 복사한다.

2. **TOMCAT_HOME**과 **JAVA_HOME** 환경변수를 설정한다.

만약 tomcat 의 경로가 /usr/local/jakarta-tomcat 이면 /etc/profile 등의 파일에 다음처럼 설정한다.

```
export TOMCAT_HOME=/usr/local/jakarta-tomcat
```

JAVA_HOME 또한 마찬가지로 jdk의 경로가 /usr/local/jdk1.2.2 이면 다음과 같다.

```
export JAVA_HOME=/usr/local/jdk1.2.2
```

그런 후 설정 화일을 실행하여 변수를 메모리로 올리면 된다.

(예 : [root@nogadax local]$. /etc/profile)

3. tomcat 디렉토리의 conf 디렉토리에 존재하는**tomcat.conf** 를 아래처럼 아파치 디렉토리 밑의 conf 디렉토리내의 **httpd.conf** 파일의 제일 마지막에 추가하면 된다.

Include /usr/local/jakarta-tomcat/conf/tomcat.conf

그런 후 tomcat.conf를 수정한다. 아래의 라인을 추가하면 된다.(tomcat_test는 예제 디렉토리임)

ApJServMount /tomcat_test /root

tomcat.conf 수정 후 webapps 디렉토리밑에 tomcat_test 디렉토리를 만든 후 tomcat_test 디렉토리에 WEB-INF 디렉토리를 만든다. 그런 후 **TOMCAT_HOME/conf/web.xml**을 새로 만든 WEB-INF 디렉토리에 복사한다.

참고로 Tomcat 3.1에 Apache1.3.12 버전이면 위의 설정 후 Tomcat을 실행하면 자동으로 tomcat-apache.conf 파일이 생성된다. 그러면 Tomcat 종료 후 tomcat.conf를 적절히 백업 후 tomcat-apache.conf를 tomcat.conf로 rename 하여 사용해도 된다.

3.3 Tomcat 3.1 실행하기

먼저 tomcat 을 실행한 후에 아파치를 실행한다.

1. tomcat 실행은 tomcat 의 bin 디렉토리의 "**tomcat.sh start**" 나 "**startup.sh**" 를 실행한다.

2. 아파치를 실행한다.(apachectl start)

3.4 종료하기

1. tomcat을 종료한다. 실행때와 마찬가지로 tomcat 의 bin 디렉토리의 "**tomcat.sh stop**" 나 "**shutdown.sh**" 를 실행한다.

2. 아파치를 종료한다.(apachectl stop)

3.5 Tomcat 3.1 테스트하기

테스트는 URL 이 http://210.110.144.235 일때에 다음과 같다.

```
http://210.110.144.235/examples/servlets
```

```
http://210.110.144.235/examples/jsp
```

3.6 Servlet 컴파일

Servlet 의 컴파일을 위해 tomcat 디렉토리 밑의 lib 디렉토리의 "**servlet.jar**" 을 CLASSPATH 에 등록하면 된다. (이외에 다른 방법이 있으면 저의 홈이나 메일로 보내주시면 고맙겠습니다.)

예 : export CLASSPATH=$CLASSPATH:/usr/local/jakarta-tomcat/lib/servlet.jar

그런 후 servlet 프로그램들을 컴파일하면 된다.

제 4 절 Jakarta-tomcat/conf 디렉토리의 server.xml 보기

tomcat에서**server.xml** 은 중요한 화일이다. 화일내의 중요한 설정요소로는 Server,Logger,ContextManager ,ContextInterceptor와 RequestInterceptor,Connector,Context 등이 있으며 그중에 기초적으로 알아야 할 부분은 Context 가 아닌가 싶다. 필자가 모든 요소에 대해 알고 있지는 않은 상태이며 최소한의 부분만 짐작하고 있을 뿐이다.

Server.xml 에서 Context 설정부분은 Web Application(Servlet이나 JSP)이 위치할 장소를 설정하며 WEB 상의 PATH(경로)를 설정한다. 기타 재컴파일된 서블릿을 자동으로 재로딩한다.

참고로 Context(webapps,work)의 base 디렉토리는 ContextManager 의 Home이며 TOMCAT_HOME 하고는 의미가 다르다. 만약 ContextManager 의 Home 이 /usr/local/jakarta-tomcat 이면 샘플용의 Web Application 이 포함된 webapps 의 절대경로는 /usr/local/jakarata-tomcat/webapps 이다. 실제로 샘플용의 web application(servlets,JSPs) 가 들어 있는 경로는 /usr/local/jakarta-tomcat/webapps/examples/WEB-INF/classes 이다.

기타 work 디렉토리는 Tomcat 실행중 중간 형태의 파일이 위치하며(예를 들어 컴파일된 JSP 파일) 이 디렉토리가 Tomcat 실행중에 삭제된다면 JSP 가 실행이 되지 않는다.

아래는 화일은 모든 내용은 아니지만 다음의 부분만 보자.

```
<Context path="/examples" docBase="webapps/examples" debug="0" reloadable="true">
</Context>

<Context path="/pgsql" docBase="webapps/pgsql" debug="0" reloadable="true">
</Context>

<Context path="/websql" docBase="webapps/websql" debug="0" reloadable="false">
</Context>
```

Context는 이전의 jserv 에서 리포지터리 역활을 하는 ZONE 과 같은 것으로 보인다.

제일 위의 examples 는 샘플용의 예제가 있으며 tomcat 의 설치 후 web 상에서 바로 실행할 수가 있다. 샘플용의 예제는 examples/WEB-INF/classes 에 있다.

샘플용인 examples 의 Hello 서블릿을 실행하기 위한 위의 WEB 상의 주소는 http://your_host_address/**examples/servlet**/Hello 이다.

web상에서 examples를 /examples/servlet 로 사용하는 이유는 다음과 같다.

tomcat 이 시작되면 conf 디렉토리의 **server.xml** 을 참조한 후 **tomcat-apache.conf**를 자동 설정한다. 그 중에 Context 에 설정된 PATH 를 참조하여 자동으로 적절하게 설정한다. 바로 이파일(tomcat-apache.conf)에 examples 를 들어가는 경로가 examples/servlet 로 설정된다.(여기에 대한 사항은 tomcat 실행후 이 파일을 참조하기 바란다.) tomcat-apache.conf 가 설정된 후에 아파치를 기동하면 아파치는 Tomcat에 의해 자동 설정된 tomcat-apache.conf 를 참조하여 실행된다.

reloadable="true"는 **Auto-reloadable** 이며 false이면 웹 어플리케이션의 Auto-reloadable 이 되지 않는다.

기타 **pgsql** 과 **websql** 은 필자가 테스트를 위해 만든 것이다. 이 두개의 디렉토리는 webapps 디렉토리에 생성하였고 각각의 디렉토리에는 또 다른 하위 디렉토리를 만들어야 한다.(WEB-INF/classes) 다음은 그 예이다.

```
webapps/pgsql/WEB-INF/classes
webapps/websql/WEB-INF/classes
```

최종 디렉토리인 **classes** 에 서블릿 클래스들이 들어가면 되며 tomcat 실행후 tomcat 디렉토리의 "**tomcat-apache.conf**" 를 보라

다음은 pgsql을 브라우져에서 사용하는 URL 예이다.

```
http://210.110.144.235/pgsql/servlet/example_servlet
```

위의 example_servlet 는 서블릿(example_servlet.class)이며 "pgsql/WEB-INF/classes" 이 있다.

제 5 절 server.dtd 설명

이장은 보다 전문적인 내용이나 XML 에 관련된 내용이므로 필독사항이 아니다. 관심있는 분만 읽어보기 바란다.

또한 이 장은 server.dtd 에 대해 완벽한 설명이 아니므로 관심있는 분은 XML 관련 문서를 참고하길 바란다.

server.dtd은 XML 마크업 언어의 메타 파일인 DTD(Document Type Declarations)파일이며, tomcat 설정 파일인 server.xml파일에서 사용되는 마크업언어를 정의하는 파일인 TOMCAT_HOME/conf/server.dtd 파일이다. 즉, server.dtd는 server.xml 의 문법을 정의 하는 파일이 되는 셈이다.

참고로 마크업 언어의 메타 파일이란 새로운 마크업 언어를 정의하는 파일이다. 또한, tomcat은 server.dtd 에서 정의된 내용을 참조하여 server.xml 의 내용을 이해 할 수 있게 되며, 사용자 또한 server.dtd 에서 정의된 문법(?)대로 server.xml에 각 설정등을 하게 된다.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

위의 라인은 XML 파일임을 나타낸다.

```
<!ELEMENT Server (ContextManager+)>
```

Server 라는 요소(element)를 정의 하며 (ContextManager) 는 server 라는 요소에 포함된다. (ContextManager+) 에서 "+"는 하나 이상을 의미하며 여기서는 ContextManager 이라는 요소가 Server 요소내에 최소한 하나가 있어야 하며 하나 이상 있어도 된다는 의미이다. 다음은 사용 예이다.

```
example 1)

<Server>
   <ContextManager></ContextManager>
</Server>

example 2)

<Server>
   <ContextManager></ContextManager>
   <ContextManager></ContextManager>
   <ContextManager></ContextManager>
</Server>
```

```
<!ATTLIST Server
    adminPort NMTOKEN "-1"
    workDir CDATA "work">
```

ATTLIST 는 속성을 의미하며 Server 요소의 속성을 정의한다. 다음은 예이다.

```
<Server adminPort="8808" workDir="workdirect" >
   <ContextManager></ContextManager>
</Server>
```

```
<!ELEMENT ContextManager (Context+, Interceptor*, Connector+)>
```

위의 예는 ContextManager 요소의 하위 요소를 정의 한다. Interceptor* 의 *은 0개 이상을 의미하므로 Interceptor 가 없어도 되고 하나 이상 있어도 된다. 다음은 예이다.

```
<Server>
   <ContextManager>
    <Context></Context>
    <Connector></Connector>
    <Connector></Connector>
   </ContextManager>
</Server>
```

```
 <!ATTLIST ContextManager
     port NMTOKEN "8080"
     hostName NMTOKEN ""
     inet NMTOKEN "">
```

위의 내용은 요소 ContextManager 의 속성을 정의 한다. NMTOKEN 은 문자열의 형으로 보아도 되며 XML 의 이름 설정에 대한 제한에 맞아 떨어져야 한다. ”8080”, ”” 등은 각 속상 값의 디폴트 값으로서 값을 설정하지 않을 경우를 대비한 것이다. 다음은 사용 예이다.

```
<Server>
   <ContextManager port="8007"></ContextManager>
</Server>
```

```
 <!ELEMENT Context EMPTY>
```

Context 요소를 정의 한다. 값은 없다(EMPTY).

```
 <!ATTLIST Context
     path CDATA #REQUIRED
     docBase CDATA #REQUIRED
     defaultSessionTimeOut NMTOKEN "30"
     isWARExpanded (true | false) "true"
     isWARValidated (false | true) "false"
     isInvokerEnabled (true | false) "true"
     isWorkDirPersistent (false | true) "false">
```

Context요소의 속성을 정의한다. CDATA 는 문자열을 의미하며 #REQUIRED 는 무조건 값이 있어야 됨을 의미하며 (true | false) 는 true 나 false 중에서 값이 나와야 함을 의미한다. 그 다음의 ”true” 나 ”false”는 디폴트 값이다. 다음은 사용 예이다.

```
<Server>
   <ContextManager>
    <Context path="/test" docBase="webapps/test"></Context>
   </ContextManager>
</Server>
```

```
 <!ELEMENT Interceptor EMPTY>
```

위의 내용은 Interceptor 요소를 정의한다.

```
 <!ATTLIST Interceptor
     className NMTOKEN #REQUIRED
     docBase   CDATA #REQUIRED>
```

위의 내용은 Interceptor의 속성을 정의한다.

이하 생략

제 6 절 server.xml 및 tomcat.conf 설정 예

다음의 예제는 메일 및 게시판을 통해 들어온 질문을 정리한 것입니다.

server.xml 과 tomcat.conf 파일은 TOMCAT_HOME/conf 디렉토리에 있습니다.

6.1 예제 1

```
***server.xml***

 <Context path="/pgsql" docBase="webapps/pgsql" debug="0" reloadable="true">
 </Context>

***tomcat.conf***

Alias /pgsql  /usr/local/tomcat/webapps/pgsql

<Directory "/usr/local/tomcat/webapps/pgsql">
   Options Indexes FollowSymLinks
</Directory>

ApJServMount /pgsql/servlet  /pgsql  (이부분을 주의 깊게 보세요)

<Location /pgsql/WEB-INF/ >
   AllowOverride None
   deny from all
</Location>
```

/usr/local/tomcat/webapps/pgsql 디렉토리에 WEB-INF/classes를 만들어 테스트용 웹 어플리케이션을 저장한다.

URL 실행 예 : http://www.xxx.com/pgsql/servlet/xxx

6.2 예제 2

```
***server.xml***


<Context path="/test" docBase="webapps/test" debug="0" reloadable="true">
</Context>

***tomcat.conf***

Alias  /test  /usr/local/tomcat/webapps/test

>Directory "/usr/local/tomcat/webapps/test">
 Options  Indexes  FollowSymLinks
>/Directory>

ApJServMount  /test  /test  (이부분을 주의 깊게 보세요)

<Location /test/WEB-INF/ >
   AllowOverride  None
   deny  from  all
```

```
</Location>
```

URL 실행 예 : http://www.xxx.com/test/xxx

6.3 예제 3

1. 일단은 tomcat디렉토리 밑의 conf디렉토리의server.xml을 다음처럼 편집합니다. (대충 끝 부분에 추가하면 되겠죠.)

```
<Context path="/jj" docBase="/usr/local/apache/htdocs/jsp" debug="0" reloadable="true" >
</Context>
```

2. 그리고 동일한 디렉토리의 tomcat.conf 파일에 아래처럼 추가해주고요.

```
Alias /jj /usr/local/apache/htdocs/jsp

<Directory "/usr/local/apache/htdocs/jsp">
   Options Indexes FollowSymLinks
</Directory>

ApJServMount /jj /jj

<Location /jsp/WEB-INF/ >
   AllowOverride None
   deny from all
</Location>
```

3. 그 다음은 /usr/local/apache/htdocs 디렉토리에 jsp 디렉토리를 만들면 되고요. jsp 이하 디렉토리는 webapps 디렉토리밑의 examples 를 보고 참조해서 디렉토리를 구성하면 될 것 같네요.

저같은 경우는 테스트를 위해 examples 디렉토리를 모두 복사했고요. 그중 examples/jsp/dates/date.html을 테스트해봤는데 잘되네요.

실행은 http://210.1x0.xx4.xx1/jj/jsp/dates/date.html로 했습니다.

제 7 절 Web Application,war,web.xml

이 부분은 Java Servlet Specification Version 2.2 를 참조하였다. 이 장에 소개된 내용외는 이 문서를 참조하기 바란다.

Web Application 이란 Servlet,JSP,HTML문서,기타 이미지나 압축된 자료 및 다른 데이타를 포함하는 Web 관련 리소스이다.

이러한 Web Application 을 압축하여 배포하기 위한 파일이 **WAR** 파일이다. **WAR** 파일은 **Web Application Archive**이며 디렉토리를 포함하여 압축 혹은 패키지된 하나의 파일이다.

web.xml은 Deployment Descriptor 이며 XML 을 알고 있는 분은 $TOMCAT_HOME/conf/**web.dtd**를 보기 바란다. web.xml의 요소는 deployment 정보 및 환경 설정 등이며 다음과 같다.

- ServletContext Init Parameters
- Session Configuration
- Servlet/JSP Definitions

- Servlet/JSP Mappings
- Mime Type Mappings
- Welcom File list
- Error Pages
- Security

web.xml 파일은 $TOMCAT_HOME/webapps/**user_app/WEB-INF** 디렉토리에 위치하며 user_app 디렉토리 이하에 있는 Web Application 의 환경 설정 부분을 담담한다고 보면 된다. 즉, 모든 Web Application 이 아닌 특정 Web Application 만의 환경 설정만을 하며, WAR 파일로 패키지 될 때 같이 포함된다.

WEB-INF 디렉토리에는 web.xml 파일과 classes 디렉토리 기타 lib 디렉토리가 위치한다. 참고로 classes 디렉토리에 servlet 과 utility class 등이 위치 할 수 있다.

참고로 web.xml은 각각의 사용자 디렉토리를 만든 후 해당 디렉토리에 대한 적절한 설정 후 **$TOMCAT_HOME/conf/web.xml** 을 **$TOMCAT_HOME/webapps/해당_디렉토리/WEB-INF/**에 복사하여 사용한다. web.xml설정은 각자의 web application 에 맞게 설정한다.

다음은 web.dtd 및 web.xml의 간단한 설명과 예를 보이겠다. 또한 web.dtd 및 web.xml의 이해를 돕기 위해 xml 에 대한 간단한 설명을 하도록 하겠다.

다음은 $TOMCAT_HOME/conf/web.dtd 의 일부분이다.

```
<!--(주석) The session-timeout element defines the default session
timeout interval for all sessions created in this web application.
The specified timeout must be expressed in a whole number of minutes.-->
<!ELEMENT session-timeout (#PCDATA)>
<!ATTLIST session-timeout id ID #IMPLIED>
...........
...........
<!--(주석) The welcome-file-list contains an ordered list of
welcome files elements. -->
<!ELEMENT welcome-file-list (welcome-file+)>
<!ATTLIST welcome-file-list id ID #IMPLIED>
<!--(주석) The welcome-file element contains file name to use
as a default welcome file, such as index.html-->
<!ELEMENT welcome-file (#PCDATA)>
<!ATTLIST welcome-file id ID #IMPLIED>
```

위의 <!ELEMENT session-timeout (#PCDATA)> session 의 디폴트 Time-Out 시간을 설정하도록 하는 요소로서 요소 정의 부분이다. (#PCDATA)는 단말 노드를 의미하며 스트링이 오면 된다. (참고로 xml 은 트리 구조이다.)

<!ATTLIST session-timeout id ID #IMPLIED> 은 session-timeout 요소에 대한 속성의 정의 부분이다. #IMPLIED는 session-timeout 요소의 id 가 생략가능함을 의미한다. 즉, <session-timeout id="123dc56"> 처럼 되어야 하나 #IMPLIED 로 인해 <session-timeout> 가 되어도 된다.

welcome-file 은 web-application 의 welcome-file을 정의하며 아파치의 DocumentRoot의 index.html 과 같다고 보면 된다. (welcome-file+) 은 welcome-file-list 요소의 자식노드를 정의 하며 "+" welcome-file 요소가 하나 이상 있어도 된다는 의미이다. 아래의 web.xml 의 예의 welcome-file-list 요소내의 welcome-file요소들처럼 다수가 올 수 있다는 의미이다.

다음은 $TOMCAT_HOME/conf/web.xml의 일부분이다.

```
<session-timeout>
  30
</session-timeout>
..........
..........
<welcome-file-list>
<welcome-file>
   index.jsp
</welcome-file>
<welcome-file>
   index.html
</welcome-file>
<welcome-file>
   index.htm
</welcome-file>
</welcome-file-list>
```

제 8 절 PostgreSQL 의 JDBC 에 연동 및 한글처리

8.1 PostgreSQL 의 JDBC 연동

Tomcat 3.1 에 postgreSQL의 JDBC 연동은 의외로 간단하다. 환경변수 CLASSPATH 에 등록하기만 하면 된다. 그러면 Tomcat 실행시 자동으로 읽어서 JDBC 를 로딩한다.

CLASSPATH 설정 예

export CLASSPATH=$CLASSPATH:/usr/local/pgsql7.0.2/jdbc/postgresql.jar

8.2 Tomcat 에서의 한글 처리

Tomcat 에서의 한글처리는 Apache-Jserv 와 동일하다.

아래는 개략적인 소스입니다.

```
 ..............
 ..............

 public void doGet(HttpServletRequest req , HttpServletResponse res)
 throws ServletException,IOException
 {
   .......
   .......
   PrintWriter out;
   res.setContentType("text/html;charset=euc-kr");
   out=new PrintWriter(new OutputStreamWriter(
                    res.getOutputStream(),"KSC5601"));
   .......
   .......
}
  .....
  .....
 public  void doPost(HttpServletRequest req, HttpServletResponse res)
 throws ServletException,IOException
 {
```

```
    res.setContentType("text/html;charset=euc-kr");
    PrintWriter out=new PrintWriter(new OutputStreamWriter(
                                res.getOutputStream(),"KSC5601"));
    ....
    String id=ksc(req.getParameter("id"));
    String name=ksc(req.getParameter("name"));
    String juso=ksc(req.getParameter("juso"));
    String tel=ksc(req.getParameter("tel"));
    String memo=ksc(req.getParameter("memo"));
    ....
    ....
  }
    .....
    .....

  public String ksc(String kscstr)
  throws UnsupportedEncodingException
  {
   if(kscstr==null) return null;
   return new String(kscstr.getBytes("8859_1"),"KSC5601");
  } //  ksc() 함수는 문자를 KSC5601로 변환한다.

  ............................................................
```

8.3 PostgreSQL 7.0.2's JDBC 에서의 한글처리

또한 포스트그레스7.0.2에 한글을 삽입하기전에 KSC5601로 변환하여야 한다. 그렇지 않으면 포스트그레스에 들어간 한글은 바로 깨진다. 기타 다른 버전은 아마 비슷할 거로 추측되며 아직 테스트는 못해 보았다.

아래처럼 문자를 ksc()함수로 ksc5601 로 변환한 후에 포스트그레스에 insert 하면 된다.

```
String id=ksc(req.getParameter("id"));
String name=ksc(req.getParameter("name"));
String juso=ksc(req.getParameter("juso"));
String tel=ksc(req.getParameter("tel"));
String memo=ksc(req.getParameter("memo"));
```

제 9 절 cocoon 1.8

Cocoon은 official 버전과 unofficial 버전을 사용하여 설치를 할 수가 있다. official버전은 까다로운 설치 및 설정을 피하기 위해 구성된 바이너리 패키지이여서 다운받아 압축을 풀고 자바 클래스만 필요한 곳에 복사를 하면 된다. 반대로 unofficial은 그렇지 않다.

cocoon 은 XML 기반의 Web publishing 을 하기 위한 100% 순수 자바 publishing Framework이다. XML을 완벽하게 지원하지 않는 브라우저를 위해 XML 관련 파일을 서버측에서 처리하여 HTML 로 변환하여 그 결과를 사용자측의 브라우저에 보여준다.

Cocoon1.8 을 다운받기를 권장하며 다운은 *http://xml.apache.org* `<http://xml.apache.org>` 에서 하기 바란다.

또한 이 문서에서는 리눅스 기반에서 Tomcat-apache 가 설치된 상태에서 Cocoon 1.8의 official 버전을 기준으로 설치 방법을 설명한다.

기타 필자가 XML에 관해 문외한이므로 XML 자체에 대한 설명은 불가하며 질문 또한 답변이 불가하니 참조하기 바란다.

9.1 cocoon 설치

1. 다운 받은 Cocoon 1.8 바이너리 패키지를 적당한 곳에서 압축을 푼다.

2. Cocoon 의 bin 디렉토리의 cocoon.jar 과 lib 디렉토리의 모든 *.jar 파일을 tomcat 디렉토리의 lib 디렉토리에 복사한다.

3. tomcat 디렉토리밑의 conf 디렉토리로 이동하여 server.xml을 편집한다. 아래의 라인을 적절히 추가하면 된다.

```
<Context path="/cocoon" docBase="webapps/cocoon" debug="0" reloadable="true">
</Context>
```

4. tomcat 디렉토리밑의 conf 디렉토리의 tomcat.conf에 다음의 예를 추가한다.

```
Alias /cocoon $TOMCAT_HOME/webapps/cocoon
<Directory "$TOMCAT_HOME/webapps/cocoon">
  Options Indexes FollowSymLinks
</Directory>
ApJServMount /cocoon /cocoon
<Location /cocoon/WEB-INF/ >
  AllowOverride None
  deny from all
</Location>
```

5. 아래의 예처럼 tomcat 디렉토리의 webapps 디렉토리에 디렉토리를 만든다.

```
mkdir $TOMCAT_HOME/webapps/cocoon
mkdir $TOMCAT_HOME/webapps/cocoon/WEB-INF
```

6. 아래의 예처럼 cocoon 소스 디렉토리내의 의 web.xml 파일과 cocoon.properties 파일을 적당히 복사한다.

```
cp  cocoon_소스_압축푼_디렉토리/src/WEB-INF/web.xml $TOMCAT_HOME/webapps/cocoon/WEB-INF
cp  cocoon_소스_압축푼_디렉토리/conf/cocoon.properties $TOMCAT_HOME/webapps/cocoon/WEB-INF
```

7. 복사된 web.xml을 수정한다.

```
<init-param>
...
<param-value>[path-to-cocoon]/conf/cocoon.properties</param-value>
...
```

위의 부분에서 <param-value>[path-to-cocoon]/conf/cocoon.properties 을 <param-value>/WEB-INF/cocoon.properties 로 수정한다.

8. 마지막으로 아래 예처럼 cocoon 소스 디렉토리내의 samples 디렉토리를 tomcat 디렉토리애로 복사한다.

```
cp  cocoon_소스_압축푼_디렉토리/samples $TOMCAT_HOME/webapps/cocoon/servlets
```

9.2 cocoon 테스트

cocoon 에 대한 설치 및 설정은 끝났다. 간단한 테스트를 위해 tomcat 과 apache를 기동한 후 아래의 예처럼 브라우저에 url 을 입력한다.

```
http://localhost/cocoon/Cocoon.xml
http://localhost/cocoon/servlets/index.xml
```

참고로 Cocoon.xml 은 디스크상에 존재하는 물리적인 파일이 아니라 설정에 의해 테스트를 목적으로 하는 가상파일이니 참고하기 바란다.

테스트가 실패할 경우 위의 순서를 다시 한번 확인하기 바라며, 기타, tomcat 기동시 tomcat 의 lib 디렉토리에 있는 xerces.jar 의 클래스 로드 순서를 xml.jar 보다 먼저 로드하기 위해 xml.jar 을 zxml.jar 로 rename 하도록 하자.

더욱 자세한 내용은 *http://xml.apache.org* `<http://xml.apache.org>` 에서 참조하기 바란다.

제 10 절 Cocoon 1.8 기타사항

10.1 Cocoon 1.8 의 euc-kr Encoding

cocoon에서 한글을 보기 위해 $TOMCAT_HOME/webapps/cocoon/WEB-INF/**cocoon.properties**을 수정해야 한다. vi 로 cocoon.properties를 열고 XML Formatters 부분에 있는 HTML 4.0(strict) 부분을 수정하면 된다. 다음은 예이다.

```
# HTML 4.0 (strict)
formatter.text/html.doctype-public = ...
formatter.text/html.doctype-system = ...
```

위의 부분을 찾아서 아래의 라인을 추가하면 된다.

formatter.text/html.encoding = euc-kr

즉, 다음처럼 추가하면 된다.

```
# HTML 4.0 (strict)
formatter.text/html.doctype-public = ...
formatter.text/html.doctype-system = ...
formatter.text/html.encoding = euc-kr
```

위의 한글 문제를 해결하기 위해 엄청난 삽질을 했다. 아파치 등의 사이트의 메일링 리스트등을 찾아보면 formatter.text/html/loose 부분을 수정하라고 나와 있었기 때문인데..

혹시 이부분에 대해 다른 대안이나 방법을 알고 있다면 (`nogadax@kldp.org`)에게 메일을 보내주었으면 한다.

10.2 간단한 xml 만들기

이분은 그렇게 권장할 만한 부분이 아니다. 필자가 만든 간단한 예제가 있어 그냥 올려 놓은 것이므로 관심 있는 분만 참조하기 바란다.

아래의 예제를 테스트하기 위해 $TOMCAT_HOME/webapps/cocoon/servlets/ngd 디렉토리를 만들었다. 테스트를 목적으로 작성한 파일은 두개이며 ngd-book.xml 과 ngd-book.xsl 이다. ngd-book.xml은 간단한 xml 파일이며 ngd-book.xsl 파일은 xml 문서를 브라우저에 디스플레이하기 위한 파일이다.

테스트를 위한 URL은 http://210.110.1xx.xx6/cocoon/servlets/ngd/ngd-book.xml 이다.

다음은 간단한 예제이다.

```
<?xml version="1.0" encoding="euc-kr"?>
<?xml-stylesheet href="ngd-book.xsl" type="text/xsl"?>
<?cocoon-process type="xslt"?>
<!DOCTYPE ngd-book [
 <!ELEMENT ngd-book (title?, name, dt, abstract, sect1)>
 <!ELEMENT title (#PCDATA)>
 <!ELEMENT name (#PCDATA)>
 <!ELEMENT dt (#PCDATA)>
 <!ELEMENT abstract (#PCDATA)>
 <!ELEMENT sect1 (#PCDATA)>
]>

<ngd-book>
 <title>Hello</title>
 <name>nogadax</name>
 <dt>/2000/11/11</dt>
 <abstract>Tomcat은 The Jakarta Project의 산물이며 홈페이지는
 http://jakarata.apache.org 이다. Jakarta Project의 목표는
 오픈 소스화되어 개발된 Java 플랫폼을 기반으로 하여 상용 품질의
 서버 솔루션을 제공하기 위함이다. 기타 XML 관련 패키지인
 cocoon 의 홈 페이지는 http://xml.apache.org이다. Cocoon은
 XML 파일을 서버측에서 처리하여 이를 브라우져에 보여주는
 패키지로 설치가 간편하다.
 </abstract>
 <sect1>
 Tomcat은 The Jakarta Project의 산물이며 홈페이지는
 http://jakarata.apache.org 이다. Jakarta Project의 목표는
 오픈 소스화되어 개발된 Java 플랫폼을 기반으로 하여 상용 품질의
 서버 솔루션을 제공하기 위함이다. 기타 XML 관련 패키지인
 cocoon 의 홈 페이지는 http://xml.apache.org이다. Cocoon은 XML
 파일을 서버측에서 처리하여 이를 브라우져에 보여주는
 패키지로 설치가 간편하다.
 </sect1>
</ngd-book>

<?xml version="1.0" encoding="euc-kr"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="ngd-book">
   <xsl:processing-instruction name="cocoon-format">type="text/html"</xsl:processing-instruction>
   <html>
    <head>
     <title>
      <xsl:value-of select="title"/>
     </title>
    </head>
    <body bgcolor="#ffffff">
      <xsl:apply-templates/>
    </body>
   </html>
  </xsl:template>

  <xsl:template match="title">
   <h2>
```

```
      <xsl:apply-templates/>
     </h2>
    </xsl:template>

    <xsl:template match="name">
      <h3>
        <xsl:apply-templates/>
      </h3>
    </xsl:template>

    <xsl:template match="dt">
      <h4>
        <xsl:apply-templates/>
      </h4>
    </xsl:template>
    <xsl:template match="abstract">
      <hr/>
      <p>
      <i>
        <xsl:apply-templates/>
      </i>
      </p>
      <hr/>
    </xsl:template>
    <xsl:template match="sect1">
      <p>
        <xsl:apply-templates/>
      </p>
    </xsl:template>

  </xsl:stylesheet>
```

제 11 절 Tomcat 3.2.1 설치

Tomcat 3.2 버젼은 크게 두가지 방법으로 설치될 수 있다. mod_jserv 를 이용한 설치와 mod_jk 를 이용한 설치가 있다. 두가지 모두 허용된 설치는 불가하며 각각 따로 설치되어야 한다.

11.1 Tomcat 3.2.1(mod_jserv,Ajpv12)

A. 설치 환경 및 주의 사항

```
- wow linux 6.2
- Tomcat 3.2.1
- Apache 1.3.14
- JDK 1.2.2
- 아래는 주의 사항이다.
  a. Apache 와 JDK 는 먼저 설치되어 있어야 한다.
  b. 아래의 설치는 mod_jserv에 의한 설치이다.
  c. 톰캣 3.2 버전대와 3.1버전대의 설치가 다를 수 있다.
  d. 또한 릴리즈될 톰캣 4.0 버전과의 설치가 다를 수 있다.
```

B. 설치

```
B-1

 -Tomcat 3.2.1 의 바이너리 버전과 소스 버전 모두 다움 받는다.
 -http://jakarta.apache.org 에서 다운받는다.

B-2

 -톰캣 바이너리 버젼은 /usr/local 에 ..
 -소스 버젼은 /usr/local/src 디렉토리에 압축을 푼다.
 -jakarata-tomcat-3.2.1.tar.gz 의 경우는 아래처럼 ..
 - "tar -zxvf jakarta-tomcat-3.2.1.tar.gz"하여 압축 해제

B-3

 -/usr/local/src/jakarta-tomcat-3.2.1-src/src/native/apache/jserv로 이동
 - "apxs -c -o mod_jserv.so *.c" 하여 mod_jserv.so 생성.

B-4

 -생성된 mod_jserv.so를 APACHE_HOME/libexec 로 copy 한다.
 -주의 : "APACHE_HOME" 은 설치된 아파치 디렉토리임

B-5

 -APACHE_HOME/conf/httpd.conf 파일의 마지막 라인에 다음 라인 추가
 - Include /usr/local/jakarata-tomcat/conf/tomcat.conf

B-6

 -/etc/profile 에 JAVA_HOME 및 TOMCAT_HOME 환경 변수 설정.
 -Tomcat 의 bin 도 환경 변수 PATH 에 등록함.
 -다음은 설정 예이다.
 - JAVA_HOME=/usr/local/jdk
 - TOMCAT_HOME=/usr/local/tomcat
 - PATH=$PATH:/usr/local/tomcat/bin
 - export JAVA_HOME TOMCAT_HOME PATH

B-7

 -/etc/profile 을 재실해하던지 (예: . /etc/profile)
 -로그아웃 후 다시 로그인

B-8

 -tomcat.sh start 하여 톰캣을 기동한다.
 -아파치보다 먼저 톰캣이 기동하여 함.
B-9

 -TOMCAT_HOME/conf/tomcat.conf를 다른 이름으로 적절히 백업 혹은 copy
 -TOMCAT_HOME/conf/tomcat-apache.conf 를 tomcat.conf로 copy 한다.

B-10

 - apachectl start 하여 아파치를 기동한다.
```

```
-아파치가 기동하면서 httpd.conf의 마지막 라인을 읽고 tomcat.conf를
 참조하게 된다.
-아파치 기동 후 브라우저를 띄우고 다음을 테스트한다.
- http://MY-URL/examples/jsp
- http://MY-URL/examples/servlets
```

11.2 tomcat.conf 예제 (Tomcat 3.2.1)

TOMCAT_HOME/conf 에 존재하는 tomcat.conf 와 톰캣이 기동할 때 자동적으로 생성되는 tomcat-apache.conf는 함께 동작하는 아파치 및 톰캣에 대한 환경 설정 부분을 담담한다.

기본적으로 jserv 에 대한 환경 설정 지시어가 사용되므로 web-server-plugin 중에 mod_jserv 만을 허용하며 디폴트 프로토콜로는 Ajpv12 를 사용하며 디폴트 포트로는 8007을 사용한다.

Tomcat 기동시 overwrite 되어 자동 생성되는 tomcat-apache.conf는 커스터마이징에 부적절하므로 생성된 tomcat-apache.conf를 tomcat.conf로 복사하여 tomcat.conf를 계속 수정하며 사용하는 것이 좋을 듯 하다. 하지만 tomcat-apache.conf를 사용하여도 큰 문제는 없을 듯 ...

```
################################################
#                                              #
#  TOMCAT_HOME/conf/tomcat.conf  (2001,01,08)   #
#                                              #
################################################

#tomcat.conf 파일은 아파치의 httpd.conf에 설정에
#의해 아파치에 의해 읽혀짐.
#아파치가 mod_jserv를 인식하도록..
LoadModule jserv_module libexec/mod_jserv.so

#mod_jserv 모듈에 대한 설정사항
<IfModule mod_jserv.c>

#mod_jserv 설정사항
############################
# Do not Edit!
############################
#아파치가 톰캣을 자동 실행 못하므로 manual on으로
ApJServManual on

#아파치와 톰캣간 통신에 쓰일 디폴트 프로토콜 설정(ajpv12)
ApJServDefaultProtocol ajpv12

#mod_jserv 는 보안키 지원안됨 DISABLED 로 ..
ApJServSecretKey DISABLED

#가상 호스팅 마운트 지원
ApJServMountCopy on

#Log Level (다른 레벨 : emerg,warn,crit,debug,,)
ApJServLogLevel notice
############################

#디폴트호스트 및 포트 설정
#아파치와 동일한 호스트(The same machine)이면 localhost
```

```
ApJServDefaultHost localhost

#아파치와 톰캣이 통신할 디폴트 포트 설정
ApJServDefaultPort 8007

#아파치에게 jsp 확장자를 인식시키고 ..
AddType test/jsp .jsp

#jsp 확장자에 대한 처리는 톰캣으로 ..
AddHandler jserv-servlet .jsp

#Alias 는 아파치가 사용할 가상 디렉토리 정의 .
#Static file(HTML,Image 파일)은 아파치가 처리하도록 설정.
#http://MY-URL/examples는 톰캣이 아닌 아파치가 처리
Alias /examples "/usr/local/tomcat/webapps/examples"

#아파치가 처리할 가상 디렉토리의 실제 디렉토리에 대한 설정사항
<Directory "/usr/local/tomcat/webapps/examples">
 Options Indexes FollowSymLinks
</Directory>

#톰캣이 사용할 URL-PATH를 위한 마운팅 설정 옵션
#http://MY-URL/examples/servlet/my_classes
#또한 아래의 마운트옵션과 다은은 등가일 것이다.
#ApJServMount /examples/servlet ajpv12://localhost:8007/examples
ApJServMount /examples/servlet /examples

#보안상 중요한 WEB-INF 디렉토리에 대한 보호 설정
#마찬가지로 META-INF도 똑같이 설정하면 좋을듯..
<Location "/examples/WEB-INF/">
 AllowOverride None
 deny from all
</Location>

..........
..........
</IfModule>
</verb></tscreen>
```

11.3 Tomcat 3.2.1 (mod_jk.so, Ajp13)

mod_jk.so는 mod_jserv.so 를 완전히 재작성한 web-server-plugin이다. web-server-plugin이란 아파치등의 웹서버와 톰캣사이의 통신을 당담하는 모듈로서 제공된 포트와 특정 프로토콜을 이용하여 웹서버와 톰캣간 통신을 지원한다.

웹 브라우저를 통해 아파치 서버등의 웹서버로 전달된 클라이언트측의 사용자 요청은 톰캣으로 전달될 수 있으며 톰캣에서 처리된 결과를 아파치 서버로 전달할 수 있게 한다.

웹서버와 톰캣사이에서의 통신을 위해 요구되는 것은 포트번호와 프로토콜 및 기타 호스트에 대한 정보등이다. 호스트에 대한 것은 위의 tomcat.conf의 ApJServDefaultHost를 보라.

mod_jk에서 사용하는 프로토콜로는 크게 두가지가 있으며 **Ajpv12** 와 **Ajpv13** 이 있다.

A. 설치환경 및 주의사항

```
 - wow linux 6.2
```

```
 - Tomcat 3.2.1
 - Apache 1.3.14
 - JDK 1.2.2
 - 다음은 주의사항이다.
   a. Apache 와 JDK 는 먼저 설치되어 있어야 한다.
   b. 아래의 설치는 mod_jk에 의한 설치이다.
   c. 톰캣 3.2 버전대와 3.1버전대의 설치가 다를 수 있다.
   d. 또한 릴리즈될 톰캣 4.0 버전과의 설치가 다를 수 있다.
```

B. 설치

```
B-1

 -톰캣 3.2.1 바이너리 버전과 소스 버전 모두 다운받는다.
 -http://jakarta.apache.org 에서 다운받는다.

B-2

 -이전에 톰캣을 mod_jserv로 설치했다면
 -mod_jserv에 대한 모든 사항은 모두 삭제
 -httpd.conf 의 Include /usr/local/tomcat/conf/tomcat.conf등의 항 제거

B-3

 -톰캣 바이너리 버전은 /usr/local/ 에
 -소스 버전은 /usr/local/src/ 에 압축을 푼다.
 -jakarata-tomcat-3.2.1.tar.gz 의 경우는 아래처럼 ..
 -"tar -zxvf jakarta-tomcat-3.2.1.tar.gz"하여 압축 해제

B-4

 -mod_jk.so 를 생성하기 위해 다음처럼  소스 디렉토리로 이동.
 -"cd /usr/local/src/jakarta-tomcat-3.2.1-src/src/native/apache1.3"
   (or apache2.0)
 -"apxs -o mod_jk.so -I../jk -I/usr/local/jdk/include
   -I/usr/local/jdk/include/linux -c *.c ../jk/*.c"
 -위의 라인을 모두 한줄에서 입력 후 실행.
 -JDK PATH 는 자신의 JDK 경로에 맞게..
 -생성된 mod_jserv.so 를 APACHE_HOME/libexec/에 copy 한다.
 -APACHE_HOME 자신의 아파치 디렉토리 경로임.

B-5

 -/etc/profile에 TOMCAT_HOME 과 JAVA_HOME 및 톰캣의 bin 경로 설정
 -다음은 설정 예이다.
 - JAVA_HOME=/usr/local/jdk
 - TOMCAT_HOME=/usr/local/tomcat
 - PATH=$PATH:/usr/local/tomcat/bin
 - export JAVA_HOME TOMCAT_HOME PATH

B-6

 -/etc/profile 을 재실해하던지 (예: . /etc/profile)
 -로그아웃 후 다시 로그인
```

```
B-7

 -Ajpv13 설정
 -TOMCAT_HOME/conf/server.xml 에 다음 내용을 Ajp12 설정 다음 부분에 추가
 -<Connector className="org.apache.tomcat.service.PoolTcpConnector">
    <Parameter name="handler"
    value="org.apache.tomcat.service.connector.Ajp13ConnectionHandler"/>
    <Parameter name="port" value="8009">
  </Connector>
 -참고로 Ajpv12 에 대한 설정 부분은 그대로 나둔다.
 -Ajpv12 설정 부분은 톰캣의 종료에 대한 부분을 관여하기 때문이다.
 -대신 stand-alone 모드가 아니므로 http 에 대한 부분은 주석처리 가능하다.

B-8

 -아파치의 httpd.conf에 다음라인 추가
 -Include /usr/local/jakarta-tomcat/conf/mod_jk.conf-auto

B-9

 -TOMCAT_HOME/conf/workers.properties 수정하기
 -수정되어질 항목은 다음의 3개이다.
 -workers.tomcat_home, workers.java_home, ps
 -workers.tomcat_home 을 다음처럼
 -workers.tomcat_home=/usr/local/jakarta_tomcat
 -workers.java_home을 다음처럼
 -workers.java_home-/usr/local/jdk
 -ps 를 다음처럼
 -ps=/
 -각각 위의 예를 보며 자신에 맞게 적절히 수정한다.


B-10

 -실행 후 확인한다.
 -tomcat.sh start
 -apachectl start
 -브라우저를 띄우고 URL 창에 다음처럼 입력
 -http://MY-URL/examples/jsp
```

11.4 stand-alone 제거

server.xml 파일을 수정하여 stand-alone 동작을 위한 connector 부분은 주석 처리하거나 삭제하여도 무방하다. 아니 삭제하는 것이 더 좋을 듯하다. 하지만 AJPV13 을 사용하더라도 AJPV12 에 대한 Connector 설정부분은 삭제를 하면 안된다. Ajpv12 Connector 부분은 Tomcat 의 Shutdown 에 관여하기 때문이다.

다음은 server.xml 내의 stand-alone 의 operation을 위한 HTTP Connection 설정부분이다. 그냥 참고하기 바란다.

```
<Connector className="org.apache.tomcat.service.PoolTcpConnector">
 <Parameter name="handler"
   value="org.apache.tomcat.service.http.HttpConnectionHandler"/>
 <Parameter name="port"
   value="8080"/>
```

```
</Connector>
```

제 12 절 Thread Pool 지원 (Tomcat3.2.1)

12.1 Thread 생성으로 인한 문제점

Tomact 은 multi-thread 된 servlet container로서 이는 클라이언트측의 각각의 요청이 쓰레드에 의해 실행됨을 의미한다. Tomcat 3.2 이전 버젼에서는 클라이언트측의 각 요청이 도착할 때마다, 쓰레드가 생성되어 각 요청을 처리하곤 하였다. 이러한 절차는 많은 부하를 일으키는 문제를 야기하며, 이러한 문제들은 다음과 같다.

* 모든 요청에 대해 쓰레드를 생성하고 소멸하는 것은 OS와 JVM에게 필요없는 많은 부담을 안겨준다.

* 동시에 일정 이상의 다수의 요청이 들어올 경우 리소스(CPU 및 메모리 자원등) 소모에 대한 제한이 어렵다. 즉, 순간적으로 서버가 다운되거나 기타 동시 다발적인 요청을 처리못해 생기는 문제가 야기될 수 있다.

12.2 문제에 대한 해결책

이러한 문제에 대한 해결책으로는 Thread Pool을 사용하는 것이며 Tomcat3.2 부터는 디폴트로톰캣 자체에서 Thread Pool 을 사용한다. 다음은 톰캣의 Thread Pool 에 대한 설명이다.

```
a. 각 요청에 대해 쓰레드를 생성하여 사용하고 난 후 계속해서 재사용할
   수 있도록 사용된 쓰레드를 "open" 상태로 계속 유지하며 이를 관리한
   다. 관리되고 있는 쓰레드는 소멸되지 않고 계속 "open" 상태로 유지
   되고 있다가 다른 요청이 들어오면 관리되고 있는 쓰레드를 사용하여
   해당 요청을 처리하도록 한다. 다시 요청을 처리하고 난 쓰레드는 관리
   대상이 되어 관리되어 지며 또 다른 요청을 기다린다.

b. 이러한 방식은 각 요청에 대해 쓰레드를 생성하고 사용한 후 소멸되는
   반복적인 절차를 탈피하게 하며, 반복적인 쓰레드의 생성 및 소멸로
   인해 야기되는 문제를 해결한다.

c. Tomcat 3.2 부터는 Tomcat의 Admin이 단지 server.xml 파일에 Thread
   Pool 에 대한 설정만으로 위의 사항을 해결할 수 있도록 하며, 자신의
   사이트에 맞도록 쓰레드의 제한 사항등을 조정할 수 있게 한다.

d. 동시에 사용할 수 있는 쓰레드의 상한선, idle 상태의 쓰레드에 대한
   최대 쓰레드 갯수 및 톰캣 기동시 생성될 최소한의 쓰레드 수를 설정
   할 수 있다.
```

12.3 Thread Pool 설정 사항

다음은 server.xml의 Thread Pool 디폴트 설정 부분이다.

```
<Connector className="org.apache.tomcat.service.PoolTcpConnector">
 <Parameter name="handler"
  value="org.apache.tomcat.service.connector.Ajp12ConnectionHandler"/>
 <Parameter name="port"
  value="8007"/>
</Connector>
```

위의 부분이 디폴트 쓰레드 풀 설정 관련 부분이다. 위의 내용을 봐서는 아무것도 모르겠지만 쓰레드 풀에 대해 설정이 생략되면 바로 디폴트 값이 세팅된다.

디폴트 값은 동시 사용가능한 쓰레드가 50개까지 (**max_threads**) 이며, idle 상태의 쓰레드가 25개 이상이면 (**max_spare_threads**) 이면 쓰레드 삭제를 하고, 최초 풀 생성시 10개 쓰레드로 (**min_spare_threads**) 상태로 시작하며 최소한 10개의 쓰레드 이상을 유지하고자 한다.

참고로 **idle 상태의 쓰레드**란 빈둥빈둥 놀고 있는 쓰레드로서, 언제 올지 모르는 요청에 대해 대기중인 여유분의 쓰레드이다. 이러한 쓰레드들도 관리 대상으로서 관리되어 진다.

톰캣 기동 후 **"ps -aux"** 를 해보라. 상당히 많은 톰캣이 보일 것이다.

다음은 Thread Pool에 대한 **설정** 예이다.

```
<Connector className="org.apache.tomcat.service.PoolTcpConnector">
 <Parameter name="handler"
  value="org.apache.tomcat.service.connector.Ajp12ConnectionHandler"/>
 <Parameter name="port"
  value="8007"/>
 <Parameter name="max_threads"
  value="30"/>
 <Parameter name="max_spare_threads"
  value="20"/>
 <Parameter name="min_spare_threads"
  value="5"/>
</Connector>
```

제 13 절 Tomcat Workers

톰캣 워커는(Tomcat worker) 웹서버를 대신하여 클라이언트측에서 요청된 서블릿을 실행하기 위해 대기중인 **인스턴스(Instance)**로서, 쉽게 얘기하면 서블릿 요청을 대기중인 톰캣 프로세스이다.

즉, 아파치 같은 웹서버를 대신하여 클라이언트측의 서블릿 요청을 톰캣 프로세스(Worker)에게 전달하여 요청을 처리할 수 있게 한다. 바로 이러한 프로세스가 워커이다. 또한 워커는 단일 워커뿐만 아니라 특정의 웹서버를 대신해 서블릿을 처리하기 위한 다중 워커를 가질 수 있다.

예를 들면, 모든 개발자들이 동일한 웹 서버에서 작업을 하고 있지만 개발 환경으로 인해 서로 다른 톰캣 워커에 의해 서로 다른 context 가 서비스되길 원할 경우가 있을 것이고, 웹 호스팅의 업체의 경우 자신의 고객을 위해 가상 호스팅 처리를 위해 서로 다른 워커를 사용하고 싶을 때 도 있을 것이다. 또한, 서블릿에 대한 분산된 처리를 하는 load balancing을 원할 경우도 있을 것이다.

이러한 여러가지 이유로 워커가 필요할 것이며 이러한 워커는 TOMCAT_HOME/conf/workers.properties 파일을 통해 설정할 수 있다.

13.1 워커 설정하기

여기에서는 간단한 설명만 언급하고 자세한 사항은 관련 문서를 참조하기 바란다. 관련 문서는 Tomcat3.2.1의 소스 버젼이나 바이너리 버젼의 doc 디렉토리에 존재하므로 참조하기 바란다.

기본적으로 TOMCAT_HOME/conf/workers.properties 파일에서 다음 3가지 항목만 찾아서 설정하면 잘 작동한다.

- workers.tomcat_home
- workers.java_home

- ps

간단한 설정 예

```
workers.tomcat_home=/usr/local/tomcat_321
workers.java_home=/usr/local/jdk_122
ps=/
```

13.2 워커 타입

다음은 워커 타입 이다.

ajp12 : ajpv12 프로토콜을 사용하여 어떻게 요청을 Out-process 톰캣 워커에게 전달할지를 아는 워커이다. 즉, ajpv12 프로토콜을 사용하는 Out-process 톰캣 worker 를 지원하는 워커 타입이다.

ajp13 : ajpv12 프로토콜을 사용하여 어떻게 요청을 Out-process 톰캣 워커에게 전달할지를 아는 워커이다.

jni : jni 사용하여 어떻게 요청을 In-process 톰캣 워커에게 전달할지를 아는 워커이다.

lb : Load-balancing 워커이다. 이 워커는 어느 정도의 결함 허용도를 가지며 라운드 로빈 방식으로 load-balancing 를 처리하는 워커이다.

워커 정의 예

worker.local.type=ajp12 : 톰캣 프로세스에게 요청을 전달하기 위해 ajp12 프로토콜을 사용하는 local라는 워커

worker.fast.type=jni : 톰캣 프로세스에게 요청을 전달하기 위해 jni 를 사용하는 fast 라는 워커

13.3 기타

먼저 자신의 리눅스 박스에 설정된 workers.properties를 보라.

이 파일은 다음을 정의 하고 있다.

- localhost 인 호스트와 8007 인 포트를 사용하는 ajp12 워커
- localhost 인 호스트와 8009 인 포트를 사용하는 ajp13 워커
- jni worker
- ajp12 와 ajp13 워커에 대해 load balancing 하는 lb 워커

ajp12,ajp13 및 lb 워커는 파일의 별다른 수정없이 바로 사용할 수 있으며, jni 워커의 경우 workers.tomcat_home,workers.java_home 및 ps 의 값을 수정하여 하며 기타 설정 문서를 참고하여야 한다.

제 14 절 기타 사항

TOMCAT_HOME/doc 디렉토리를 참고하기 바란다.

참고할 사항들

```
-JDBC Realm
-Configuring for Multiple Tomcat JVMs In A Minimalistic User's Guide
-Configuring Virtual Hosting In a Minimalistic User's Guide
```

```
-Using the Java SecurityManager with Tomcat
-Tomcat workers.properties
-Tomcat and SSL
```

이 문서는 TOMCAT_HOME/doc 에 있는 문서를 참조하였다.

문서는 다음과 같다.

```
-JDBC Realm
-Configuring for Multiple Tomcat JVMs In A Minimalistic User's Guide
-Configuring Virtual Hosting In a Minimalistic User's Guide
-Using the Java SecurityManager with Tomcat
-Tomcat workers.properties
-Tomcat and SSL
-Release Notes for tomcat
-Working with mod_jk
-Tomcat_apache HOWTO
-Developing application with tomcat
-Faq on Tomcat
```

언급되지 않은 부분들에 대해서

```
이 문서에서 언급 되지 않은 부분은 위의 참고할 사항들을 찾아보기 바라며
이전에 작성된 톰캣 3.1문서도(KLDP나 DSN 에서..) 참조하기 바란다.
```