빌드도구

Maven을 활용한 빌드 환경

빌드환경의 필요성

빌드(Build)란?

소스 코드 파일을 동작하는 **독립적인 소프트웨어 산출물**로 만드는 과정이다. 빌드의 가장 중요한 단계 중 하나는 소스 코드 파일을 실행 코드로 변환하는 컴파일 과정이다. 간단한 프로그램은 하나의 파일만 컴파일 해도 되겠지만, 많은 소스코드와 다양한 버전닝을 필요로하는 복잡한 프로젝트는 어떻게 할 것인가? **빌드툴(Build Tool)**이 이런 수고로움을 덜어주고 있다!!

빌드 툴(Build Tool)

- 새로운 버전의 프로그램을 빌드 할 때 사용하는 툴
- 일반적인 빌드 툴이 제공하는 기능
    - 전처리(preprocessing), 컴파일(compilation), 패키징(packaging), 테스팅(testing), 배포(distribution)

빌드 자동화란?

개발자가 매일 진행하는 일을 자동화 하는 행동들, 빌드툴들이 이런 자동화를 도와준다

- 소스 코드를 바이너리 코드로 컴파일
- 바이너리 코드 패키징
- 테스트 수행
- 실서비스 시스템 배포
- 도큐먼트 및 릴리즈 노트 생성

개발에 있어 빌드란?

- 애플리케이션 개발에 있어 빌드는 개발하는 애플리케이션을 완성 시켜주는 마지막 단계
- 개발한 것들을 산출물, 결과물을 내는 과정
- **빌드를 잘하는 것은 애플리케이션을 만드는데 중요한 과정이다**
- 좋은 빌드 툴을 선택하는 것은 개발의 기본 요건이다!

빌드 툴 소개

Apache Ant

- 자바로 구현됨, 자바 프로젝트 빌드에 적합
- XML 파일(build.xml)을 사용하여 빌드 프로세스와 의존성 정의

Apache Maven

- Apache Ant와 비슷하나 컨셉과 동작방식이 다름
- C#, Ruby, Scala등 Java가 아닌 프로젝트도 빌드와 관리가 가능
- XML 파일(pom.xml)에 정의함(의존성, 빌드 순서, plug-in등)
- Maven Repository - 자바 라이브러리나 플러그인을 다운받을수 있음

Gradle

- Ant나 Maven과 비슷한 컨셉
- XML대신 DSL을 통해 프로젝트 정의

Maven 소개

Maven

- 빌드툴? 프로젝트 관리 툴?
- 소스 코드로 부터 배포 가능한 산출물을 빌드하는 '빌드툴'
- **프로젝트 관리 도구**
    - 프로젝트 오브젝트 모델, 표준 집합, 프로젝트 라이프사이클, 의존성관리 시스템, 라이프사이클에 정의된 단계에서 플러그인 골을 실행하기 위한 로직을 포함하는 관리 툴이다
- Archetype 통해 적합한 구조의 프로젝트 생성,

애플리케이션 명칭과 버전 및 연관된 라이브러리에 대한 정보 등을 종합적으로 관리

서브 프로젝트간의 관계 손쉽게 설정

Nexus를 통해 저장소 관리

Maven이 제공하는 기능

- Builds
- Documentation
- Reporting

- Dependencies
- SCMs
- Releases
- Distribution

Maven의 시작

메이븐은 Jakarta Turbine 프로젝트의 빌드 프로세스를 단순화 하기 위해 시작되었다.
여러 프로젝트 마다 자신의 ANT 빌드 파일과 JAR파일들이 존재 하고 있었다. 이런 환경은 빌드를 복잡하게 만든다.
점차 프로젝트가 커짐에 따라 프로젝트를 빌드하는 표준방법 필요하게 되었다. 이로 인해 나온것이 바로 메이븐이다.

Maven의 목적

개발자가 짧은 기간에 개발의 전체 상태를 이해 할 수 있도록 한다

프로젝트 공통 구조

src/main/java	Application/Library sources
src/main/resources	Application/Library resources
src/main/filters	Resource filter files
src/main/assembly	Assembly descriptors
src/main/config	Configuration files
src/main/scripts	Application/Library scripts
src/main/webapp	Web application sources
src/test/java	Test sources
src/test/resources	Test resources
src/test/filters	Test resource filter files
src/site	Site
LICENSE.txt	Project's license
NOTICE.txt	Notices and attributions required by libraries that the project depends on
README.txt	Project's readme

자바 웹 애플리케이션 배포

웹어플리케이션을 개발하고 배포할 때 각자의 컴퓨터에서 소스를 코딩하고 github과 같은 버전관리 시스템에 push한뒤 배포용 서버 컴퓨터에서 이를 pull받아 war파일로 package한뒤 이를 tomcat에 배포하는 과정으로 진행

서블릿 컨테이너 및 TOMCAT

개발 서버, 실 서버로 넘어가는 시점부터는 Tomcat 서버에 직접 접속해 시작하고 디버깅 로그 파악해야 하기 때문에 Tomcat에 대한 기본적인 이해가 필요하다.

서블릿 컨테이너

웹 컨테이너(web container, 또는 서블릿 컨테이너)는 웹 서버의 컴포넌트 중 하나로 자바 서블릿과 상호작용한다. 웹 컨테이너는 서블릿의 생명주기를 관리하고, URL과 특정 서블릿을 맵핑하며 URL 요청이 올바른 접근 권한을 갖도록 보장한다.

웹 컨테이너는 서블릿, 자바서버 페이지(JSP) 파일, 그리고 서버-사이드 코드가 포함된 다른 타입의 파일들에 대한 요청을 다룬다. 웹 컨테이너는 서블릿 객체를 생성하고, 서블릿을 로드와 언로드하며, 요청과 응답 객체를 생성하고 관리하고, 다른 서블릿 관리 작업을 수행한다.

웹 컨테이너는 웹 컴포넌트 자바 EE 아키텍처 제약을 구현하고, 보안, 병행성, 생명주기 관리, 트랜잭션, 배포 등 다른 서비스를 포함하는 웹 컴포넌트의 실행 환경을 명세한다.

오픈 소스 웹 컨테이너

- **아파치 톰캣** (예전 자카르타 톰캣) 은 아파치 소프트웨어 라이선스 하에 사용할 수 있는 오픈 소스 웹 컨테이너다.
- **아파치 제로니모**는 아파치 소프트웨어 재단에서 자바 EE 6를 완전히 구현한 웹 컨테이너다.
- Lutris Technologies 사의 **Enhydra**.
- 오라클의 **글래스피시** (웹 컨테이너를 포함하는 애플리케이션 서버)
- **제이보스** (현재 WildFly) 는 레드햇의 제이보스에서 자바 EE 를 완전히 구현한 웹 컨테이너다.
- 이클립스 재단의 **제티**. SPDY와 웹소켓 프로토콜을 지원한다.
- **Jaminid** 는 서블릿의 추상화된 개념을 포함하고 있다.
- **Winstone** 은 v2.5 (현재 0.9) 명세를 지원하고, 최소한의 설정에 초점을 맞추고 원하는 기능만 남도록 추려내는 기능을 가지고 있다.
- **Tiny Java Web Server** (TJWS) 2.5 는 작고 모듈화 가능하도록 설계되어있다.

- 이클립스 재단의 **Virgo**는 모듈화된 OSGi 기반의 웹 컨테이너로 내장 톰캣과 제티를 구현하고 있다. Virgo는 이클립스 공용 라이선스 하에 사용할 수 있다.

상용 웹 컨테이너

- 오라클의 iPlanet Web Server.
- 레드햇, 제이보스의 JBoss Enterprise Application Platform 는 서브스크립션 커머스와 오픈소스로 된 자바 EE 기반의 애플리케이션 서버다.
- 어도비 시스템즈의 JRun (과거 Allaire Corporation에서 개발).
- 오라클의 WebLogic Application Server (과거 BEA Systems에서 개발).
- IronFlaer의 Orion Application Server.
- Caucho Technology의 Resin Pro.
- New Atlanta Communications의 ServletExec.
- IBM WebSphere Application Server.
- SAP NetWeaver.
- SpringSrouce Inc. 의 tc Server.

Tomcat 디렉토리 구조 설명

bin/
톰캣 서버 실행과 관련된 쉘 스크립트 파일(.sh, bat등)을 모아둔 곳으로 윈도우즈의 경우 startup.bat파일을 더블 클릭하여 톰캣서버를 가동시킬 수 있습니다. 물론 명령창으로 실행할 수 있습니다.

conf/
톰켓 서버를 실행할 때 참조할 **설정파일**을 모아둔 곳입니다.

lib/
톰켓 서버를 구성하는 **자바 클래스 라이브러리**들을 모아둔 곳입니다.

logs/
톰켓 서버를 실행하는 동안 실행 또는 오류 정보를 기록한 파일을 모아둔 곳으로서, 언제 누가 접속했는지, get, post요청을 했는지, ip address 등이 세세히 기록됩니다.

temp/
톰켓 서버가 실행하는 동안 임시 데이터를 보관하는 폴더입니다.

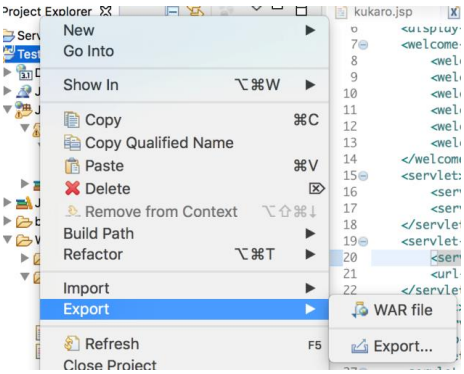
wepapps/
웹 애플리케이션을 모아둔 곳입니다.

work/
톰켓 서버가 JSP를 실행할 때 그 중간 파일을 보관하는 곳입니다.

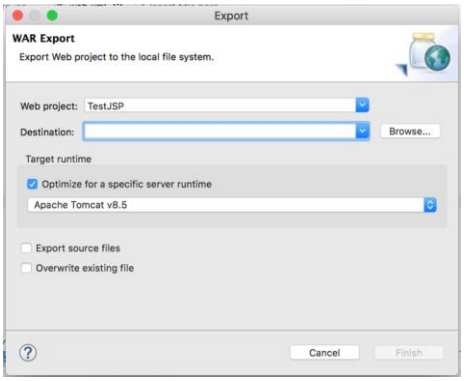
war 파일 빌드, 설정, Tomcat 서버 패포

war는 web archive를 줄인 말이다. 웹 애플리케이션은 war 파일로 만들어 배포할 수 있다. war 파일은 메이븐 빌드 도구의 package 명령을 실행해 가능하다. 배포 형태를 디렉토리를 통째로 복사하거나 war 파일로 배포하는 두가지 형태가 있다. 환경에 따라 두 가지 방법 중 하나를 사용하면 된다.

war 파일 생성



프로젝트를 우 클릭한 후 Export->WAR file을 눌러준다.



Destination, 즉 export할 경로를 선택해 준다.

war 파일을 Tomcat 서버에 배포

배포하고자 하는 WAR파일을 EC2서버로 업로드 한 후, 톰캣의 webapps 폴더에 넣고 재실행 하면 톰캣이 알아서 war 파일에서 패키지를 추출합니다.

Tomcat의 webapps의 경로 : **/usr/share/tomcat8/webapps**

war 파일업로드는 FileZilla 프로그램을 이용해서 업로드.

실제 서비스 폴더는 /var/lib/tomcat8/webapps/ 입니다. 이 폴더는 바로 업데이트가 불가하기 때문에 사용자 홈 디렉토리에 업데이트 후 파일 이동을 해주면 됩니다.

ex> # mv OpenProject.war /var/lib/tomcat8/webapps/open.war

OpenProject.war 파일을 /var/lib/tomcat8/webapps/ 폴더에 open.war 이름으로 이동

개발 서버에 소스 코드 배포 과정

소스 코드를 빌드/배포하는 과정에서 수 많은 반복 작업이 발생한다. 반복 작업은 사람의 실수를 유발하고 이는 대규모 장애로 이어진다. 따라서 반복 작업은 컴퓨터를 통해 자동화하는 것이 가장 안전하다.

배포 과정에서 발생하는 반복 과정에 대한 설명

(1) git clone하기
우리는 작업은 각자의 로컬에서 하고 그 내용을 GitHub에 push 하면 배포 서버에서는 그 저장소를 pull받아와 빌드하여 사용하는 식으로 배포를 할 계획이다.
우선 배포용 서버라는 가정하에 로컬에 위와 같이 git clone을 받을 디렉토리를 하나 만든다.
- 작업하던 프로젝트의 github 저장소의 url 을 복사한다.
- apps 디렉토리에서 clone을 받는다.

(2) clone받은 프로젝트를 배포하기 위해 war파일로 package 한다.(이클립스는 export)
- 먼저 clone받은 저장 디렉토리로 이동한다.
- mvn clean package 명령어로 이전의 빌드 내용을 지우고 다시 war파일을 생성해준다.
- package 결과로 target 디렉토리가 생성되었다.
- target 디렉토리 안에는 package한 war파일과 그 디렉토리가 생성되어있다.

(3) 현재 빌드한 war파일을 톰캣에 복사 하기 위해 잠시 서버를 중단한다.
- tomcat 디렉토리/bin 에서 shutdown.bat파일 실행

(4) 생성된 package 결과를 배포할 톰캣의 webapps 디렉토리에 복사한다.
- 결과에 대한 복사는 cp 명령어를 사용하거나 rsync 명령어를 사용하면 변경된 내용을 자동으로 동기화 하여 좀더 편리하게 관리할 수 있다.

(5) 서버를 실행시킨다.
- tomcat 디렉토리/bin 에서 startup.bat 실행

(6) 브라우저를 이용해 접속해본다.