**Maven 정리**

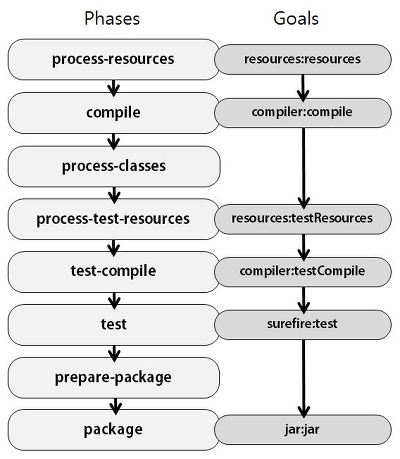
**1. 메이븐이란**

**1.1 메이븐**

메이븐은 프로젝트 구조와 내용을 기술하는 선언적 접근방식의 오픈 소스 빌드 툴. 쉽게 말하면 **라이브러리 자동관리 플러그** 인이다. 컴파일과 동시에 빌드를 수행할 수 있으며 테스트를 병행하거나 서버 측 자원을 관리할 수 있는 환경을 제공한다. 하지만 아무래도 개발자들에게 가장 큰 장점은 프로젝트의 종속 라이브러리 들과 그 라이브러리에 영향을 미치는 Dependency자원까지 관리할 수 있다는 점이다.  
즉, jar파일을 다운받아 프로젝트에 추가할 경우 그것과 연관된 다른 종속 라이브러리 또한 다 찾아야 하는 불편함을 maven을 통해서 일관성 있는 라이브러리간의 의존관계를 관리할 수 있다. 이는 단순히 라이브러리뿐 아니라 프로젝트 별 모듈의 의존성 또한 관리가 가능하다.

주요 기능으로는빌드, 문서화, 리포팅, 의존관계 관리, 소스코드 관리, 릴리즈, 배포가 있다.  
  
**1.2.1 메이븐 사용의 장점**  
  - **의존성 관리** (오픈 소스 라이브러리뿐만 아니라 프로젝트 혹은 모듈간의 의존성 또한 관리가 가능함)  
  - 잘 정의된 convention에 의해서 **일관된 프로젝트 구성**을 가질 수 있다.  
  - IDE에 **종속적인 부분을 제거**할 수 있다. (Eclipse, IntelliJ, NetBeans, JBuilder, JCreator, JDveloper)  
  - 이클립스를 사용하는 경우 maven 이클립스 플러그 인을 사용하면 의존성을 갖는 오픈 소스의 소스 코드를 F3키를 사용하여 아주 쉽게 확인할 수 있다.  
  - Maven을 알면 Java를 이용한 많은 오픈 소스 프로젝트들이 Maven 프로젝트로 구성되어 있어 **오픈 소스를 분석할 때 편리**하다.  
  - Maven Profile 기능을 사용하면 배포환경에 따른 **설정 파일을 관리하고 배포 파일을 생성할 수 있다**.   
  - 의존 라이브러리를 pom.xml 파일을 통해서 관리하므로 버전(형상)관리 시스템으로 공유할 **파일의 크기가 줄어든다.**   
 **1.2.2 메이븐 사용의 단점**

- maven은 현재 3.x.x 버전까지 왔지만 일부의 기능에서 예상외로 동작하는 경우가 있다.   
  - 이클립스의 maven 플러그 인인 m2e가 가끔 올바르게 실행이 잘 되지 않는다.  
**2. Maven 기초  
2.1 Maven 기본 명령어**  
 - maven [options] [<goal(s)>] [<phase(s)>]



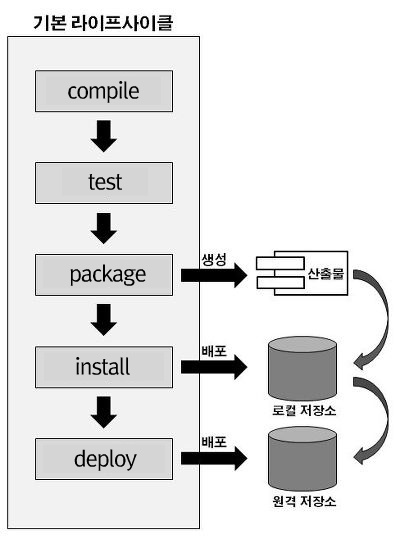
**2.1.1 MAVEN GOAL**

goal은 내가 실행할 명령. 메이븐에서 기본 명령어는 이런 식이다. mvn [-option] [<goal(s)>] [<phase(s)>] 의 형태이다. 예를 들어 'mvn clean compiler:compile' 의 형태로 빌드를 실행했다면 이런 의미이다. clean phase를 실행하고, compiler 플러그인의 compile 골을 실행한다. phase에 goal이 연결되어 있기 때문에 phase를 통해 메이븐 빌드를 실행하면 해당 phase에 연결되어 있는 goal이 실행된다.

**2.1.2 MAVEN PHASE**

phase의 사전적 의미는 단계이다. 메이븐에서도 phase는 단계를 의미한다. phase는 빌드 라이프사이클에서 빌드 단계와 각 단계의 순서만을 정의하고 있는 개념으로, 빌드 과정에서 phase가 빌드 작업을 하지는 않는다. 아까도 말했다시피 실질적인 빌드 작업은 각 phase에 연결되어 있는 플러그인의 goal이 한다.

**2.2 메이븐 POM 설정 파일**  
 - 프로젝트 기본 정보  
  + 프로젝트 이름, URL, 개발자, 라이선스 등등  
 - 빌드 설정  
  + 기본 빌드 설정을 변경하기 위한 카테고리 (소스, 리소스 디렉토리 변경, 플러그인 설정 변경 및 플러그인 추가)  
 - 프로젝트 관계 설정  
  + 각 프로젝트간의 관계 관리 혹은 모듈간의 의존성 관리  
 - 빌드 환경  
  + 다양한 환경에 따라 달라지는 설정 정보를 관리 (프로파일 기능)  
 - 속성 정보 관리  
  + pom/project 속성, settings 속성, 시스템 환경 변수 속성, 자바 시스템 속성 등을 정의 또는 사용할 수 있다.  
 - 메이븐은 기본적인 프로젝트 디렉토리 구조와 플러그인 정보를 포함하고 있는 POM설정파일을 제공하며 모든 POM 설정 파일은 해당 설정파일을 상속한다. (해당 설정 파일은 메이븐에 포함되어 있는 jar 파일을 통하여 배포된다.)  
  
**2.3 메이븐 라이프사이클**



 - 메이븐은 미리 정의하고 있는 빌드 순서를 라이프사이클 이라고 하며, 라이프사이클의 각 빌드 단계를 **페이즈**라고 한다.  
 - 라이프사이클은 여러 단계(페이즈)로 나뉘어져 있으며, 각 페이즈는 의존 관계를 갖는다.

|  |
| --- |
| - 기본 라이프 사이클   + **compile** : 소스 코드를 컴파일 한다   + **test** : 단위 테스트 실행 (기본설정은 단위 테스트가 실패하면 빌드 실패로 간주함)   + **package** : 컴파일 된 클래스 파일과 리소스 파일들을 war 혹은 jar와 같은 파일로 패키징   + **install** : 패키징한 파일을 로컬 저장소에 배포 (USER\_HOEM/.m2/)   + **deploy** : 패키징한 파일을 원격 저장소에 배포 (nexus 혹은 maven central 저장소) |
| - clean 라이프 사이클   + **clean** : 메이븐 빌드를 통하여 생성된 모든 산출물을 삭제한다. |
| - site 라이프 사이클   + **site** : 메이븐 설정파일 정보를 활용하여 프로젝트에 대한 문서 사이트를 생성한다.   + **site**-deploy : 생성한 문서 사이트를 설정되어 있는 서버에 배포 |

**2.4 메이븐 페이즈와 플러그인**  
 - 메이븐에서 제공하는 모든 기능은 플러그인 기반으로 동작한다. 메이븐 라이프사이클에 포함되어 있는 페이즈 또한 플러그 인을 통하여 실직적인 작업이 실행된다.   
 - <bild>/<plugins>/<plugin> 엘리먼트를 사용하여 사용하고자 하는 플러그 인을 추가 및 설정 할 수 있다.   
 - 메이븐 플러그 인은 하나의 플러그 인에서 여러 작업을 수행할 수 있도록 지원하며, 플러그 인에서 실행할 수 있는 각각의 작업을 goal 이라고 정의한다.

 - 페이즈와 플러그인 관계  
  + 각 페이즈의 작업은 메이븐에서 기본으로 포함하고 있는 각 플러그인의 골을 이용해서 수행한다.   
 - 메이븐 플러그 인을 검색하고 사용방법을 확인할 수 있는 곳  
  + http://maven.apache.org/plugins/index.html  
  + http://mojo.codehause.org/plugins.html   
  
**2.5 메이븐을 이용한 의존성 관리**

 - 메이븐 저장소 구성  
  + **중앙 저장소**: 오픈 소스 라이브러리, 메이븐 플러그인, 메이븐 타입을 관리하는 저장소이다. 중앙 저장소는 개발자가 임의로 라이브러리를 배포할 수 없다.   
  + **원격 저장소**: 메이븐 중앙 저장소 이 외에 각각의 회사 혹은 오픈 소스 재단에서 운영 관리하는 저장소   
   ex) http://maven.springframework.org (sprint source), http://mesir.googlecode.com/svn/trunk/mavenrepo (ojdbc), http://192.168.1.46:5050/nexus (사내 maven 저장소)  
  + **로컬 저장소** : 메이븐을 빌드할 때 다운로드 하는 라이브러리, 플러그 인을 관리하는 개발자 PC의 저장소 (USER\_HOME/.m2)  
 - 메이븐 저장소 설정 (xml 파일예제..)  
 - 메이븐 의존성 관리  
  + <dependencies>/<dependency> 엘리먼트를 사용하여 의존성을 관리하며, 의존 라이브러리의 groupId, artifactId, version, scope 정보들을 갖는다.  
 - 메이븐 의존성 정보에서 scope 설명  
  + **compile**: 기본 scope, 컴파일 및 배포할 때 같이 제공해야 하는 라이브러리  
  + **provided** : servlet.jar 와 같이 컴파일 시점에는 필요하지만 배포할 때에는 포함되지 말아야 하는 라이브러리  
  + **runtime**: 컴파일 시에는 사용되지 않지만 실행환경에서 사용되어지는 라이브러리  
  + **test** : JUnit과 같이 테스트 시점에만 사용되는 라이브러리  
  + **system** : provided와 비슷하다. 단지 우리가 직접 jar 파일을 제공해야 한다. 따라서 이 scope의 jar 파일은 저장소에서 관리되지 않을 수도 있다.  
  + **import**: 다른 POM설정 파일에 저의 되어 있는 의존 관계 설정을 현재 프로젝트로 가져온다.  
 - 의존성 전이  
  + 오픈 소스 프레임워크를 의존성에 추가하며, 해당 프레임워크가 의존하고 있는 오픈 소스 라이브러리 또한 의존 관계에 자동으로 포함된다.   
  + 의존성 전이 기능은 프로젝트의 의존성을 편리하게 관리할 수 있도록 도와주기도 하지만 불필요한 라이브러리가 추가되거나 의존성이 꼬이게 만드는 원인이 되기도 한다.   
 - 의존성 전이에 대한 설정 변경기능  
  + **의존성 중개**: 버전이 다른 두 개의 라이브러리가 의존 관계에 있다면 메이븐은 더 가까운 의존 관계에 있는 pom 설정의 버전과 의존관계를 갖는다. 예를 들어 A 프로젝트가 A -> B -> C -> D2.0 버전, A -> E -> D1.0 버전의 의존 관계가 발생한다면, A 프로젝트는 D1.0 버전과 의존 관계를 갖는다. 만약 D2.0 버전을 사용하고 싶다면 A 프로젝트의 메이븐 설정 파일에 명확하게 의존 관계를 명시해야 한다. (A -> D.20)  
  + **의존성 관리** : <dependency Management> 엘리먼트를 사용하여 의존 관계에 있는 라이브러리와 버전을 명시적으로 정의한다.   
  + **의존성 예외**: <exclusion> 엘리먼트를 활용하여 의존성 전이를 예외 처리한다.  
  + 기타: 의존성 스코프, 선택적 의존성 등의 기능이 있음

**2.7 메이븐 프로파일 기능**  
 - 메이븐은 서로 다른 환경에 따라 달라지는 설정을 다르게 관리할 수 있는 프로파일 기능을 제공한다.

 - 프로파일은 <profiles> -> <profile> 엘리먼트를 사용하여 설정할 수 있으며, <project> 루트 엘리먼터가 가질 수 있는 모든 엘리먼트를 설정할 수 있다.

 Ex) 프로파일에 따른 설정 예제

|  |
| --- |
| <profiles>  <profile>  <id>local</id>                 <!-- 기본 Profile 설정 -->  <activation>  <activeByDefault>true</activeByDefault>  </activation>  <properties>  <environment>local</environment>  </properties>  <dependencies>  <!-- JDBC Driver -->  <dependency>  <groupId>com.oracle</groupId>  <artifactId>ojdbc14</artifactId>  <version>10.2.0.4.0</version>  </dependency>  </dependencies>  </profile>  <profile>  <id>development</id>  <properties>  <environment>development</environment>  </properties>  <dependencies>  <!-- JDBC Driver -->  <dependency>  <groupId>com.oracle</groupId>  <artifactId>ojdbc14</artifactId>  <version>10.2.0.4.0</version>  </dependency>  </dependencies>  </profile>    <profile>  <id>service</id>  <properties>  <environment>service</environment>  </properties>  <dependencies>  <!-- JDBC Driver -->  <dependency>  <groupId>com.oracle</groupId>  <artifactId>ojdbc6</artifactId>  <version>11.1.0.7.0</version>  </dependency>  </dependencies>  </profile>  </profiles>  <build>          <!-- Profile 기능을 이용한 리소스 관리 -->  <resources>  <resource>  <directory>src\main\resources</directory>  </resource>  <resource>  <directory>src\main\resources-${environment}</directory>  </resource>  <resource>  <directory>src\main\java</directory>  <excludes>  <exclude>\*\*/\*.java</exclude>  </excludes>  </resource>  </resources>  </build> |

 - 메이븐 사용시 프로파일 지정  
  + mvn -P<profile\_id> pahse[s] goal[s]  
  ex) mvn -Pdevelopment clean package  // 개발서버에 배포할 패키지 파일 생성  
        mvn -Pservice clean package  // 상용 서버에 배포할 패키지 파일 생성  
   
**2.8 메이븐 모듈**  
 - 메이븐은 하나의 프로젝트에서 여러 모듈을 관리할 수 있는 기능을 지원한다.   
 - 일반적으로 서비스를 개발하게 되면 관리자 페이지 혹은 CMS(Content Management System) 과 고객 혹은 가입자에게 서비스 제공을 위한 Web Module이 분리가 되며, 두 개의 모듈이 핵심 비지니스 로직을 공유하게 된다. 이런 경우 메이븐의 모듈 기능을 이용하면 유용하다.

 - 메이븐의 모듈 개념을 이해하고 활용하려면 상속, 집합, 의존 관계 개념을 이해해야 한다.  
 - 상속  
  + 프로젝트에서 공통으로 사용하는 설정은 공통 POM 파일을 만들어 관리하고 하위 모듈에서 이 POM파일을 상속할 수 있다.   
 \* 부모 POM 파일

|  |
| --- |
| <project>    <modelVersion>4.0.0</modelVersion>    <groupId>net.javajigi</groupId>    <artifactId>wikibook</artifactId>   <packaging>pom</packaging>    <version>1.0-SNAPSHOT</version>    [...]  </project> |

 \* 부모 POM 파일을 상속하는 자식 POM 파일

|  |
| --- |
| <project>    <modelVersion>4.0.0</modelVersion>    <parent>      <groupId>net.javajigi</groupId>      <artifactId>wikibook</artifactId>      <version>1.0-SNAPSHOT</version>    </parent>    <groupId>net.javajigi</groupId>    <artifactId>wiki-web-parent</artifactId>    <version>1.0-SNAPSHOT</version>    [...]  </project> |

 출처: http://dimdim.tistory.com/entry/Maven-%EC%A0%95%EB%A6%AC.

<http://jokergt.tistory.com/164>

http://arihong218.tistory.com/entry/MAVEN-plugin-phase-goal-LifeCycle