메이븐 기본 이해

(Basic Maven)

목차

introduction

소개

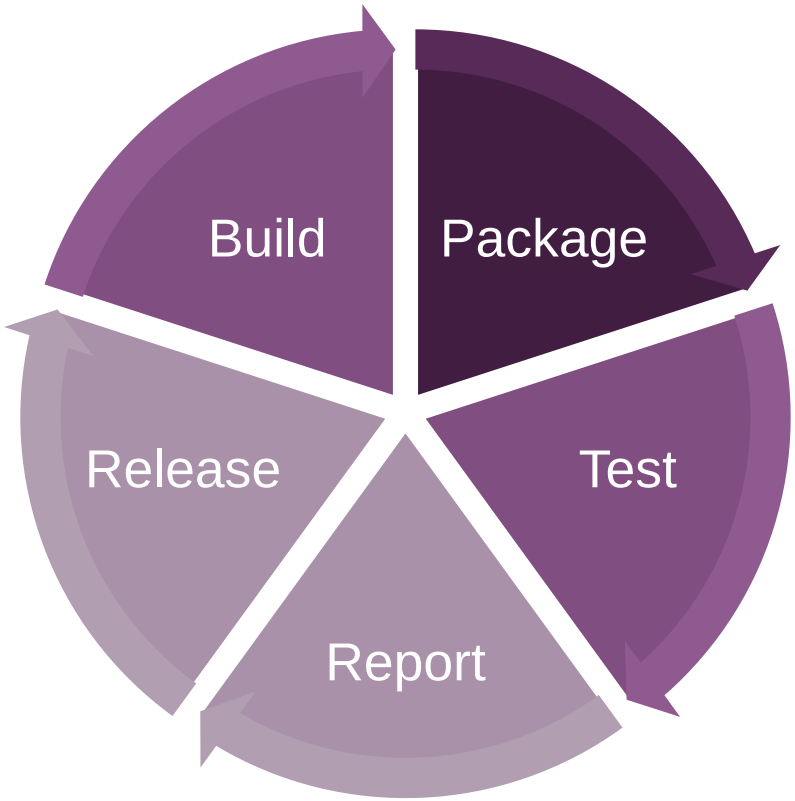
메이븐(Maven)이란 무엇인가?

- 질문에 대한 답은 관점에 달려 있다. 대다수의 메이븐 사용자들은 메이븐을 소스 코드로부터 배포 가능한 산출물 (artifact)을 빌드(build)하는 '**빌드 툴(build tool)**' 이라고 한다.
- 빌드 엔지니어 들과 프로젝트 관리자 들은 무엇인가 조금 더 편리한 '**프로젝트 관리 툴**'이라고 할 것이다.

- 앤트(Ant)와 어떠한 차이가 있는가?
  - 앤트(Ant)와 같은 빌드 툴은 전적으로 전처리(preprocessing), 컴파일(compilation), 패키징(packaging), 테스팅(testing), 배포(distribution) 하는데 초점이 맞추어져 있다.
  - 메이븐과 같은 프로젝트 관리 툴은 빌드 툴을 바탕으로 여러 기능들을 종합적으로 제공한다.
  - 덧붙이면 메이븐은 빌드에 관한 기능들과 보고서 작성, 웹 사이트 생성, 작업 팀의 구성원 간 소통 기능을 제공한다.

☞ "Maven : Sonatype 이 만든 Maven 핵심 가이드 | 팀 오브라이언 저 | 장선진 역 | 지앤선 출판" 인용

메이븐 활용 패턴



일반적인 활용 패턴은 5단계의 흐름으로 구성된다.

- Build
  - 소스 코드를 컴파일 한다.
  - 테스트 코드를 컴파일 한다.
  - 기타 패키지 생성을 위한 바이너리를 생성한다.
- Package
  - 배포 가능한 jar, war, exe 파일 등을 생성한다.
- Test
  - 단위 테스트(Unit Test) 등을 실행한다.
  - 빌드 결과가 정상적인지 점검한다.
- Report
  - 빌드/패키지/테스트 결과를 정리하고, 빌드 수행 리포트를 생성한다.
- Release
  - 빌드 후 생성된 아티팩트(artifact)를 로컬 혹은 원격 저장소에 저장(배포)한다.

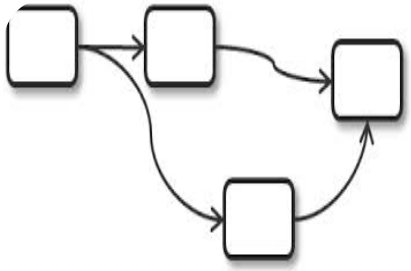
Maven의 다섯 가지 핵심 개념 이해

핵심 개념들

5가지 핵심 개념 (Key Concepts)

플러그인
(Plugin)

라이프사이클
(Lifecycle)



의존성
(Dependency)

프로필 (Profile)

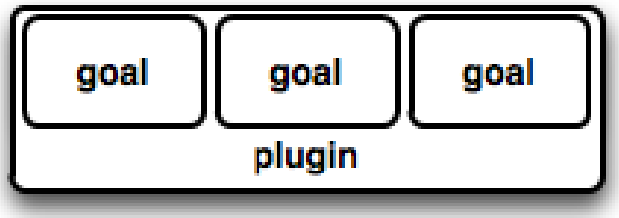
POM

① Maven 을 제대로 활용하기 위해서는 5가지 용어에 대한 개념을 익혀야 한다.

② 단어 자체는 외우기 쉽지만, 다른 도구나 언어에 적용되는 것과 다른 의미를 가지고 있으므로, 혼동하지 않도록 주의 깊게 학습해야 한다.

플러그인 (Plugin)

- 메이븐(Maven)은 플러그인 실행 프레임워크이다.
- 메이븐의 플러그인 메커니즘에 의해 기능이 확장된다.
  (모든 작업은 플러그인이 수행한다)
- 사용자 관점에서는 앤트(Ant) 의 태스크(Task) 혹은
  타겟(Target, 정확히 일치하지 않지만)과 유사하다.
- 플러그인은 다른 산출물(artifacts)와 같이 저장소에서 관리된다.
- 플러그인은 골(goal)의 집합이다.



모조(Mojo)

- 모조(Mojo) = 메이븐 ~~Commands~~ Goals
- 메이븐은 여러 가지 플러그인으로 구성되어 있으며, 각각의 플러그인은 하나 이상의 goal (명령, 작업)을 포함하고 있다.
  - 다른 의미로 '플러그인'은 새로운 '마력(mojo)'을 추가(설치)한다는 의미
- Goal은 플러그인과 goal 명칭의 조합으로 실행할 수 있다.
  - 형식 : <plugin>:<goal>
  - 예시 : mvn archetype:generate
- 메이븐은 여러 goal을 묶어 "라이프사이클 단계(lifecycle phases)"로 만들고 실행한다.
  - 형식 : mvn <phase name>
  - 예시 : mvn install

플러그인 선언 예시

- Ant (target)

```
<target name="compile" depends="init" description="compile the source ">
    <!-- Compile the java code from ${src} into ${build} -->
    <javac srcdir="${src}" destdir="${build}" source="1.6" target="1.6" />
</target>
```

- Maven (plugin)

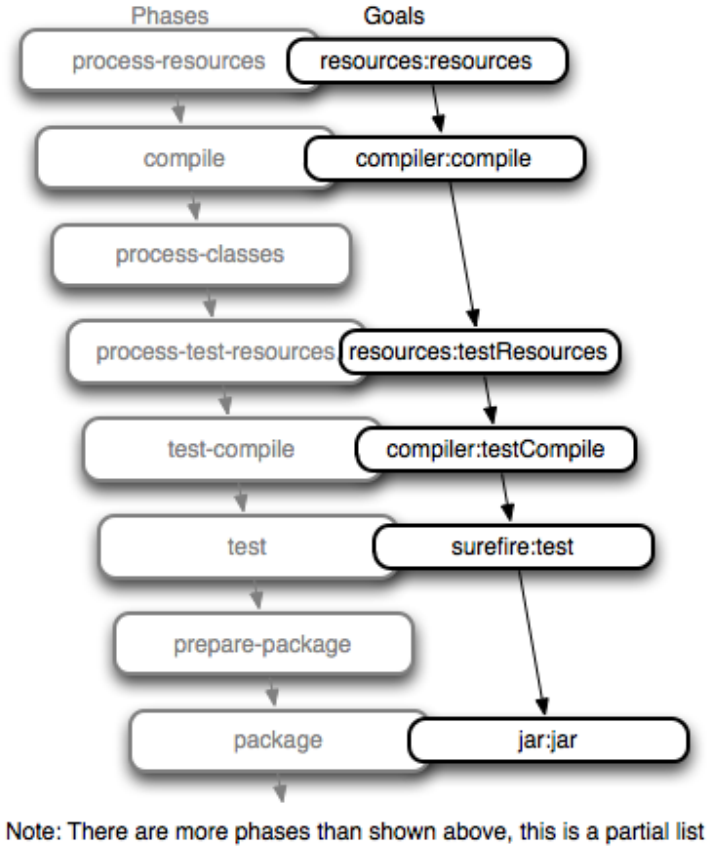
```
<build>
                        …
    <plugins>
        <plugin>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>2.1</version>
            <configuration>
                <source>1.6</source>
                <target>1.6</target>
            </configuration>
        </plugin>
    </plugins>
                        …
</build>
```

라이프사이클 (lifecycle)

- 메이븐의 동작 방식은 일련의 단계(phase)에 연계된 goal 을 실행하는 것이며, 논리적인 작업 흐름인 단계의 집합이 라이프사이클이다.
  - 빌드 단계(build phases)들은 사전 정의된 순서대로 실행된다.
  - 모든 빌드 단계는 이전 단계가 성공적으로 실행되었을 때, 실행된다.
- 빌드 단계는 goal들로 구성된다.
  - Goal은 특정 작업, 최소한의 실행 단위(task)이다.
  - 각 단계는 0개 이상의 goal과 연관(associate)된다.
- 메이븐은 3개의 표준 라이프사이클을 제공한다.
  - clean : 빌드 시 생성되었던 산출물을 지운다.
  - default : 일반적인 빌드 프로세스를 위한 모델이다.
  - site : 프로젝트 문서와 사이트 작성을 수행한다.

단계 (phase)

- 단계(phase)는 논리적인 개념이며, 실질적인 작업을 수행하는 것은 각각의 단계에 연결(associate) 플러그인 goal 이다.
- 패키지 타입(package type : jar, war, ear 등)에 따라 각 단계에서 수행되는 goal이 달라질 수 있다.
- default 라이프사이클은 jar 를 생성하는 작업들을 수행하며, 다음과 같은 goal 들로 구성된다 :



Note: There are more phases than shown above, this is a partial list

내장(build-in) 라이프 사이클 (1/2)

[default 라이프사이클]

validate	프로젝트의 상태가 정상인지 여부와 빌드에 필요한 모든 정보가 존재하는지 검사한다.
initialize	빌드 상태를 초기화 한다. 속성을 설정하거나, 작업 디렉터리 등을 생성하는 작업을 수행한다.
generate-sources	컴파일에 필요한 소스를 생성한다.
process-sources	필요한 모든 값에 대한 필터링(filtering) 처리 등 소스 코드를 가공한다.
generate-resources	패키지에 포함될 자원(resources)들을 생성한다.
process-resources	패키지 작업을 준비하기 위해 자원들을 대상 디렉터리로 복사하고 가공한다.
compile	프로젝트의 소스 코드를 컴파일 한다.
process-classes	컴파일 된 파일들에 대한 후처리(post-process)를 수행한다. (예를 들어, 자바 클래스에 대한 byte-code enhancement)
generate-test-sources	컴파일 하기 위해 테스트를 위한 소스 코드를 생성한다.
process-test-sources	필요한 값에 대한 필터링 처리 등 테스트 소스 코드를 가공한다.
generate-test-resources	테스트 수행을 위한 자원을 생성한다.
process-test-resources	테스트 대상 디렉터리에 자원을 복사하고 가공한다.
test-compile	테스트 소스 코드를 컴파일하고 대상 디렉터리에 담는다.
process-test-classes	컴파일 된 테스트 파일들에 대한 후처리(post-process)를 수행한다.
test	적합한 단위 테스트 프레임워크를 이용해 테스트를 수행한다. 테스트 코드는 패키지 되거나 배포된 패키지 없이 실행될 수 있어야 한다.
prepare-package	실제 패키지 생성을 수행하기 전에 필요한 사전 작업을 수행한다.
Package	컴파일 된 코드 등을 이용해 jar 등의 패키지 파일을 생성한다.
pre-integration-test	통합 테스트(integration test)를 실행하기 위한 사전 처리를 수행한다. 환경 설정 작업 등을 예로 들 수 있다.
integration-test	통합 테스트를 수행할 수 있는 환경이라면 패키지를 배포하고 가공한다.
post-integration-test	통합 테스트 수행 이후의 후처리 작업을 수행한다. 환경 설정 해제 등의 작업을 예로 들 수 있다.
verify	패키지가 품질 기준에 적합한지 여부를 검사하는 작업을 수행한다.
install	다른 프로젝트에서 참조하고 사용할 수 있도록 패키지를 로컬 저장소에 설치한다.
deploy	다른 개발자 혹은 프로젝트와 공유할 수 있도록 원격 저장소에 최종 패키지를 배포한다.

내장(build-in) 라이프 사이클 (2/2)

[clean 라이프사이클]

pre-clean	clean 작업을 수행하기에 앞서 필요한 사전 작업을 수행한다.
clean	이전 빌드에서 생성된 모든 파일들을 삭제한다.
post-clean	프로젝트 clean 작업을 끝내기 위해 필요한 작업을 수행한다.

[site 라이프사이클]

pre-site	사이트(site) 생성을 수행하기에 앞서 필요한 사전 작업을 수행한다.
site	프로젝트 사이트 문서를 생성한다.
post-site	프로젝트 사이트 생성을 끝내기 위해 필요한 작업을 수행하고, 배포를 위한 사전 작업을 수행한다.
site-deploy	생성된 사이트 문서를 대상 웹 서버로 배포한다.

의존성 (dependency)

- 라이브러리 다운로드 자동화
  - 더 이상 필요한 (의존성 있는) 라이브러리를 하나씩 다운로드 받을 필요가 없다. 필요하다고 선언만 하면 메이븐이 자동으로 다운로드 받아준다.
- 메이븐은 선언적 (명령식이 아니다) :
  - 사용되는 jar 파일들을 어디서 다운로드 받고, 어느 릴리즈(버전)인지 명시하면, 코딩을 하지 않아도 메이븐이 알아서 관리한다. (재 다운로드, 최신 버전 설치 등)
- 메이븐이 관리한다.
  - 라이브러리 (lib) 디렉터리를 생성할 필요가 없다.
  - 이클립스 내에서 라이브러리, 클래스패스 환경 설정을 할 필요도 없다.

프로파일 (profile)

- 서로 다른 대상 환경(target environment)를 위한 다른 빌드 설정
  - 다른 운영체제
  - 다른 배포 환경
- 동작 방식 (Activation)
  - -P 명령행 실행환경(CLI) 옵션
  - 환경 변수(environment variable) 기반...
- 메이븐은 정상 절차(step) 이외에 프로파일을 위한 절차를 추가로 수행한다.

프로파일 - 예시

```
<profiles>
	<profile>
		<id>cn-dev</id>
		<activation>
			<property>
				<name>target</name>
				<value>cn-dev</value>
			</property>
		</activation>
		<properties>
			<encoding>gb2312</encoding>
		</properties>
	</profile>
	<profile>
		<id>us-deploy</id>
		<activation>
			<property>
				<name>target</name>
				<value>us-deploy</value>
			</property>
		</activation>
		<properties>
			<skipTests>true</skipTests>
		</properties>
	</profile>
</profiles>
```

POM

- POM = Project Object Model, 프로젝트 객체 모델
- 프로젝트 당 하나의 pom.xml
  - 각각의 프로젝트는 pom.xml 파일을 하나씩 가진다.
  - pom은 프로젝트 자체와 의존성에 대한 설정 및 정보를 포함한다.
  - 메이븐을 pom.xml 을 읽어, 프로젝트를 가공하는 방법을 이해한다.
- 3 가지 “coordinates”를 이용해 자원을 식별한다.
  - 그룹 ID (Group ID) : com.acme
  - 아티펙트 ID (Artifact ID) : common
  - 버전 (Version) : 1.0

POM – 중요한 속성들

- <artifactId/>
  - 아티팩트의 명칭 (Artifact's name), groupId 범위 내에서 유일해야 한다.
- <groupId/>
  - 일반적으로 프로젝트의 패키지 명칭
- <version/>
  - 아티팩트(artifact)의 현재 버전
- <name/>
  - 어플리케이션의 명칭
- <packaging/>
  - 아티팩트(artifact) 패키징 유형
  - POM, jar, WAR, EAR, EJB, bundle, ... 중에서 선택 가능
- <distributionManagement/>
  - 아티팩트(artifact)가 배포될 저장소 정보와 설정
- <parent/>
  - 프로젝트의 계층 정보
- <version/>
  - 아티팩트(artifact)의 현재 버전
- <scm/>
  - 소스 코드 관리 시스템 정보
- <dependencyManagement/>
  - 의존성 처리에 대한 기본 설정 영역
- <dependencies/>
  - 의존성 정의 및 설정 영역

CoC

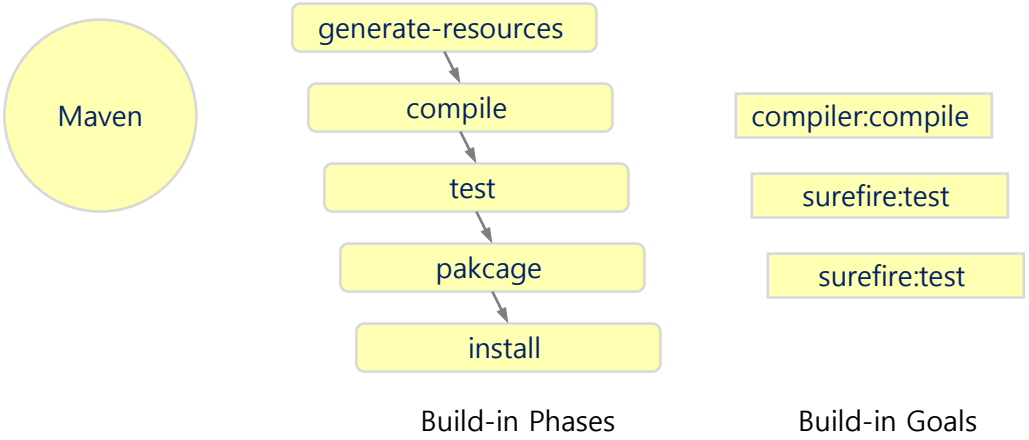
- CoC : Convention over Configuration 의 약어
- 메이븐의 큰 철학이며 명확한 '관습'으로 인해 더 편해진다는 의미이다.
  - 개발자는 소스나 실행 파일 디렉터리 명칭을 '그냥' 보고 알 수 있다.
  - 개발자들의 관습 혹은 암묵적으로 알고 있는 디렉터리나 위치 정보를 빌드 도구들도 똑같이 사용하자....
  - 설정 작업이 좀더 간결해지고 쉬워진다. (관습과 다른 내용만 서술/명시)
- 루비 온 레일스 (Ruby on Rails) 등으로 인해 유명해짐
- 관습 혹은 기본 값은 (Conventions, defaults) 거의 다, 보이지 않는 "super pom"에 선언되어 있다.

Ant vs. Maven vs. Gradle

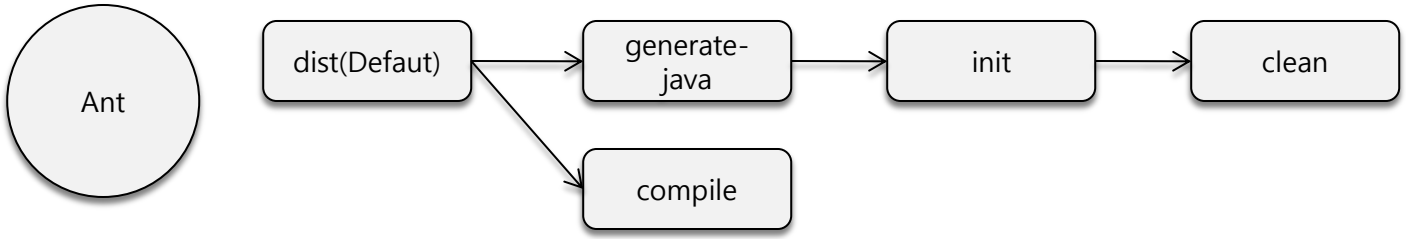
빌드 도구 비교

Ant vs. Maven

- Maven
  - 선언적, 관습과 설정



- Ant
  - 명령적, 설정과 스크립트 작성



<target name="dist" depends="generate-java, compile"/>

Maven vs. Ant

- 메이븐 (Maven)
  - 모듈화(modularization) 하기 좋다.
  - 의존성 관리를 수행해 준다.
  - 초보자가 이해하기 어렵다.
  - 버그와 이슈(문제)를 추적하기 어렵다.
    (관습을 이해하고 학습해야 한다.)
  - 가끔 느리다고 여겨지기도 한다.
- 앤트 (Ant)
  - 배우기 쉽다. (추상화가 많지 않다.)

메이븐 vs. 앤트 혹은 그래들

- vs. Ant
  - 표준화된 프로젝트 구조
  - 의존성 관리를 제공함
  - 내장된 보고서 및 문서 생성 기능 제공
  - 프로젝트에 상관없이 플랫폼 설치가 용이함.
- vs. Gradle
  - 좀 더 상세한 POM 파일 (장점 혹은 단점)
  - 프로그래밍에 대한 지식이 덜 필요함.
  - 좀 더 큰 커뮤니티의 지원
    - 사전 제작된 플러그인, 해법과 다양한 문서
  - 더 나은 통합개발환경 지원
    - 이클립스는 예외, 둘다 이클립스에서는 별로...

그리고 남은 것들

정리

용어 리뷰

- Project – 메이븐이 작업을 수행하는 대상
- Pom – 메이븐이 프로젝트를 처리하는 필요한 정보를 제공하는 파일.
- Artifact – 프로젝트에 필요한 jar, war, pom 혹은 다른 것들.
- Dependency – 프로젝트에 필요한 다른 프로젝트에 존재하는 파일들
- Coordinates – 아티팩트(artifact)를 식별하는데 필요한 속성들의 조합.
- Repository – 아티팩트 (artifact) 들이 위치하는(저장된) 장소.
- Lifecycle – 아티펙트(artifact)를 생성하는 절차들의 집합.
- Phase – 라이프사이클을 구성하는 하나 이상의 절차
- Plugin (Mojo) – 확장(추가) 가능한 기능
- Install – 아티펙트를 로컬 디렉토리에 저장하는 행위 ~/.m2/repository
- Deploy – 아티펙트를 로컬 저장소에 저장하는 행위
- Reactor – 의존성을 분석하고, 해결한 후 빌드를 수행하는 프로세스.

권고 사항

- 학습 곡선(learning curve)이 가파를 수 있다.
- 인내를 가져라 : 빌드 작업에 필요한 코드가 줄어들 것이다.
- 관습(convention)을 따르라
- 그러나, 관습을 깨는 것을 주저하지 말라.
- 또 다른 방법(혹은 의견)을 수용하라
- 에러 메시지는 밑에서부터(bottom-up) 읽어라.
- 진실로 기묘한 상황에 처하면, “mvn clean” 을 실행하고, 로컬 저장소를 지워라. “rm ~/.m2/repository”
- 레퍼런스를 읽어라.

  http://www.sonatype.com/books/mvnex-book/reference/public-book.html

참고 문서 및 사이트

- Maven 3 overview
  - http://www.slideshare.net/MikeEnsor/maven-3-overview-15845337
- Maven
  - http://www.slideshare.net/leefs/maven-7839125
- Basic Maven
  - http://www.slideshare.net/croeder6000/maven-7847270
- Maven : Sonatype 이 만든 Maven 핵심 가이드
  - 팀 오브라이언 저 | 장선진 역 | 지앤선 출판