



Gradle 이란

ant와 maven의 장점을 모은 빌드 도구 / ant와 maven의 단점을 보완한 빌드 도구
유연함과 성능에 초점을 둔 오픈소스 빌드도구

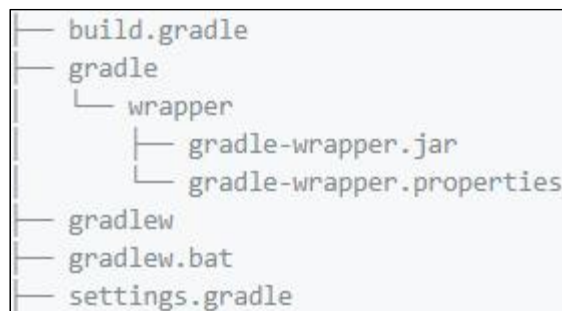
왜 Gradle 인가

	
실제 설정 내용보다 XML 뼈대가 더 많다	Groovy DSL로 작성 1) 가독성이 좋고 2) Groovy 언어를 몰라도 금방 습득 후 구성가능
빌드와 테스트 실행 속도 Gradle에 비해 느리다	캐시를 사용하기 때문에 속도가 빠르다

코드 작성의 효율성 및 속도 측면에서 Gradle을 사용해야 한다!!

Gradle 프로젝트 구조

- 1) .gradle : 의존성 파일들 다운로드 cache 저장되는 폴더
- 2) gradle : local에 gradle 설치되어 있지 않고, 버전 상관없이 빌드할 수 있는 wrapper 관련 파일들 존재
- 3) gradlew : 유닉스용 실행 스크립트
 - Gradle로 컴파일이나 빌드 할 때, “gradle build” 하면 로컬에 설치된 gradle 사용
 - Gradle로 컴파일이나 빌드 할 때, “./gradlew build” 하면 wrapper 아래의 jar를 이용해서 gradle task실행
- 4) gradlew.bat : Windows용 실행 배치 스크립트
- 5) **build.gradle** : 의존성이나 플러그인 설정 등을 위한 스크립트 파일
- 6) settings.gradle : 프로젝트의 구성정보를 기록하는 파일



[Gradle 프로젝트의 전체 구조]

Gradle 특징

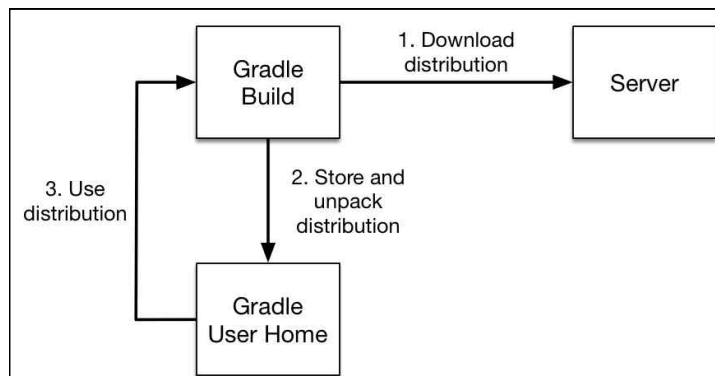
1) 멀티프로젝트 지원

- 하위 프로젝트에 대한 정의를 최상위 프로젝트(Root Project)에서 `allprojects`, `subprojects` 속성(Property)을 통해 각 프로젝트의 공통 속성과 작동을 정의할 수 있다

```
allprojects {  
    //모든 프로젝트 공통사항  
}  
  
subprojects {  
    //서브 프로젝트의 공통된 빌드 구성 설정  
}  
  
project(':module-1') {  
    //module-1 프로젝트에 대한 빌드 구성 설정  
}  
  
- project(':module-2') {  
    //module-2 프로젝트에 대한 빌드 구성 설정  
}
```

2) 래퍼(Wrapper) 지원

- 래퍼(Wrapper)는 빌드도구를 실행할 수 있는 jar 파일과 이를 실행할 수 있는 스크립트를 함께 등록하여 관리하는 방식이다



[래퍼를 이용한 빌드 과정]

래퍼의 버전을 업그레이드하고 변경사항을 커밋한 후 원격저장소에 푸시하면 이후에는 팀원들에게 업그레이드된 래퍼가 공유된다.

```
// ./gradlew wrapper --gradle-version={version}  
$ ./gradlew wrapper --gradle-version=5.4
```

[래퍼 버전 업그레이드 방법]

3) 의존성 옵션

Gradle 3.0부터 의존 라이브러리 수정시 재빌드가 필요한 라이브러리를 선택적으로 할 수 있도록 compile대신 api와 implementation으로 나눠 필요없는 경우 재빌드 하지 않도록 함

- compileOnly : compile시에만 빌드하고 빌드 결과물에는 포함하지 않음
- runtimeOnly : runtime시에만 필요한 라이브러리인 경우
- api : 의존 라이브러리 수정시 본 모듈을 의존하는 모듈들도 재빌드
- implementation : 의존 라이브러리 수정시 본 모듈까지만 재빌드

4) 공용저장소 종류

- mavenCentral(), jcenter(), gradlePluginPortal(), google()
- 뿐만 아니라 특정 저장소를 url 로 지정가능

```
maven {  
    url "http://repo.mycompany.com/maven2"  
}
```

[특정 url을 저장소로 지정한 예시]

프로젝트 적용

1) 프로젝트 최상위 폴더 내부에서 “gradle init --type pom” 커맨드 수행


```
C:\Users\User\Desktop\WorkArea\GPMS_MAVEN_TO_GRADLE\gooroom-admin>gradle init --type pom  
  
> Task :init  
Maven to Gradle conversion is an incubating feature.  
Get more help with your project: https://docs.gradle.org/5.6.1/userguide/migrating_from_maven.html  
  
BUILD SUCCESSFUL in 2s  
2 actionable tasks: 2 executed
```

- pom type : 이미 생성되어 있는 Maven 기반 프로젝트 파일을 gradle로 마이그레이션 할 때 사용하는 type
- pom type은 반드시 Maven 기반 프로젝트 디렉토리에서 실행되어야 함

[참조] : <https://blog.naver.com/PostView.nhn?blogId=sharpLee7&logNo=221411403165>

2) 프로젝트 전체 구조가 Gradle 프로젝트 구조로 변경됨

3) pom.xml에 dependency로 설정되어 있던 의존파일들 implementation 옵션 형태로 변경 되서 적용됨



Why no compile configuration?

The Java Plugin has historically used the `compile` configuration for dependencies that are required to both compile and run a project's production code. It is now deprecated — although it won't be going away any time soon — because it doesn't distinguish between dependencies that impact the public API of a Java library project and those that don't. You can learn more about the importance of this distinction in [Building Java libraries](#).

4) 기존의 pom.xml에 의존성으로 걸려있던 라이브러리 중 빠진 라이브러리 없는지 확인

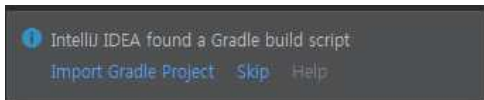
5) 변경완료

프로젝트 사용(IntelliJ IDE) Import

- 1) File -> New -> Project from Existing Sources
- 2) Select File or Directory to Import 다이어로그에서 import하려는 프로젝트 내의 build.gradle파일 선택
- 3) 열린 다이어로그에서 “open as project” 선택
- 4) 프로젝트 import 완료



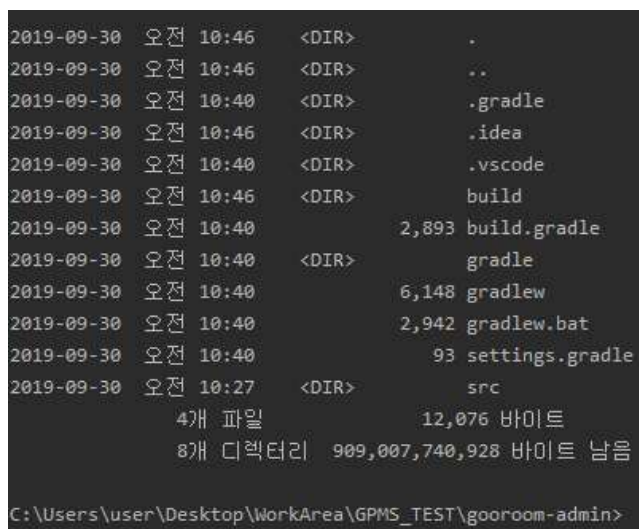
- 5) 프로젝트 import 후 IDE 오른쪽 아래



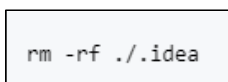
다이어로그에서 “Import Gradle Project” 수행

ReImport

- 1) 프로젝트 디렉터리로 이동



- 2) IntelliJ의 프로젝트 설정을 담고 있는 .idea 디렉터리 삭제



3) 디렉터리 삭제 후 프로젝트에 IntelliJ 설정이 없어서 다시 시작하면 Import가 진행됨

참고자료

1)https://docs.gradle.org/current/userguide/migrating_from_maven.html

2)https://docs.gradle.org/current/userguide/building_java_projects.html#sec:building_java_webapps

3)https://docs.gradle.org/current/userguide/war_plugin.html