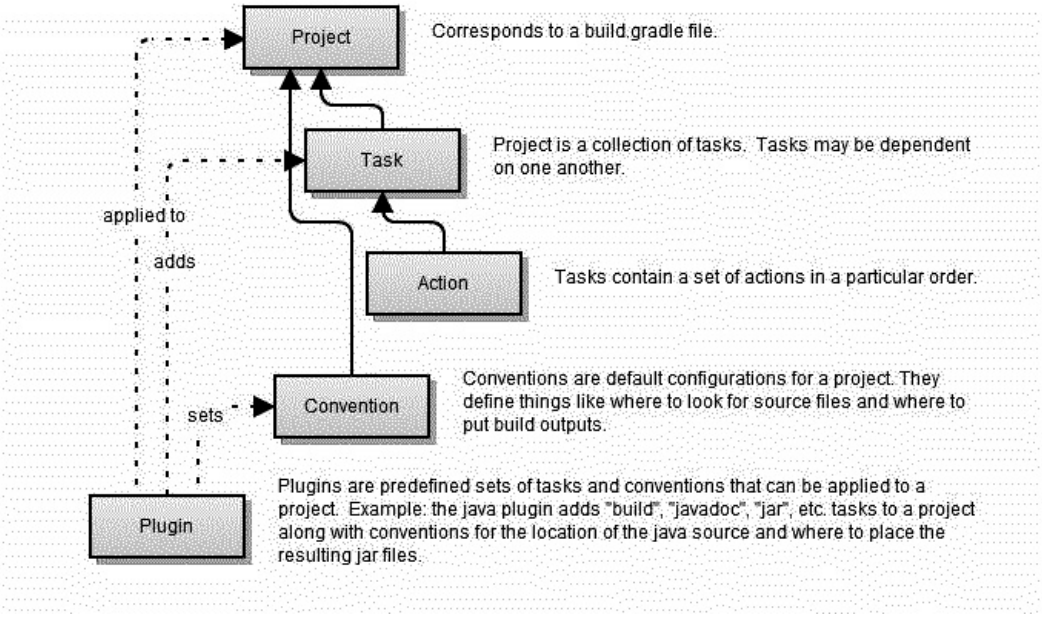
09.9주차Gradle기본스크립트와의존성관리

Table of Contents

Gradle 기본 스크립트

Gradle 빌드 스크립트 구조



이미지 출처:http://blogs.perficient.com/businessintelligence/2012/08/01/

- Projects : Gradle에서 Project는 Tasks (이하 태스크) 의 집합체이다. build.gradle을 일컫기도 하며, 단일 혹은 멀티 프로젝트로 구성할 수 있다.
- Tasks : 프로젝트에는 하나 이상의 태스크가 필요하다. 각각의 태스크는 빌드를 수행하는 작업의 일부라고 볼 수 있으며 상호 의존한다. 하

나의 태스크는 특정 순서대로 실행되는 Action(행위)들을 포함한다.

- Plugin : 태스크와 Convention(관례)의 집합으로 프로젝트에 적용 될수 있다. 예를 들어 프로젝트에 java 플러그인을 적용하면 class 컴파일, JAR 만들기, javadoc생성, 혹은 빌드 결과물을 repository에 배포하는 작업 등을 할 수 있다.

본 문서에서는 **Projects**와 **Tasks**를 중심으로 프로젝트를 빌드할 때 사용하는 간단한 태스크를 만들어 보면서 빌드 스크립트를 작성하는 법을 배워 본다.

1. Hello world 태스크

Gradle 빌드는 **gradle [option..] [task..]**를 명령어를 통해서 동작한다. gradle 명령어를 사용하면 현재 디렉토리에서 build.gradle 이라는 파일을 찾게되는데 이 build.gradle 파일을 **빌드 스크립트**라고 부른다. 엄밀하게 말하면 빌드 구성 스크립트(Build Configuration Script) 라고도 한다.
그러면 빌드 스크립트라고 불리우는 build.gradle 파일을 생성해서 아래와 같이 작성한다.

```
task hello {
    doLast {
        println 'Hello world!'
    }
}
```

파일을 저장하고 다음 명령어를 실행해본다.

```
gradle -q hello
```

Hello wolrd!

결과를 보면 Hello wolrd가 출력된 것을 알 수 있다. (-q 옵션은 에러 로그만 출력하라는 의미이다. 자세한건 gradle —help를 통해 알 수 있다.)
gradle hello를 실행하면 Gradle는 hello 태스크를 행한다. 그리고 태스크에 작성된 action을 수행하게 되는데 이 action은 그루비 코드를 실행할 수 있는 클로저이다.

2. 태스크 축약하기

태스크를 축약해서 간결하게 정의할 수 있다.

```
task hello << {
    println 'Hello world!'
}
```

Gradle은 위처럼 단일 클로저로 이루어진 task를 task definition style 이라고 한다.

3. 빌드 스크립트는 그루비 코드이다.

Gradle의 빌드 스크립트는 그루비의 코드로 이루어진다. task에 그루비를 사용해보자.

```
task upper << {
    String someString = 'mY_nAmE'
    println "Original: " + someString
    println "Upper case: " + someString.toUpperCase() // toUppperCase 메소드 사용.
}
```

gradle -q upper 의 결과는 다음과 같다.

Original: mY_nAmE
Upper case: MY_NAME

다음은 그루비의 times 연산자를 사용해본다.

```
task count << {
    4.times { print "$it " }
}
```

gradle -q count 결과는 다음과 같다.

0 1 2 3

4. 태스크 의존성 (Tasks Dependencies)

태스크 간의 의존관계를 설정할 수 있다. task taskName(depdendsOn: 다른태스크 | [task1, task2, ...]) ... 형태로 만든다.

```
task hello << {
    println 'Hello world!'
}
task intro(dependsOn: hello) << {
    println "I'm Gradle"
}
```

gradle -q intro의 결과는 다음과 같다. intro 태스크를 수행하면 의존하고 있는 hello task가 동작한 후 자신이 동작된다.

```
Hello world!
I'm Gradle
```

한 가지 더 알아둘 점은 의존성을 추가할 때 해당 태스크는 미리 존재할 필요가 없다. Lazy 하게 구동되기 때문이다.
아래 코드를 보면:

```
task taskX(dependsOn: 'taskY') << {
    println 'taskX'
}
task taskY << {
    println 'taskY'
}
```

taskY가 코드상에서 나중에 정의되어있어도 결과는 제대로 나온다.

```
gradle -q taskX
```

```
taskY
taskX
```

5. 동적 태스크 (dynamic tasks)

그루비의 특성을 이용하여 태스크를 동적으로 생성할 수 있다.

```
4.times { counter ->
    task "task$counter" << {
        println "I'm task number $counter"
    }
}
```

gradle -q task1 를 실행해보면 다음 처럼 출력된다. task0~3이 생성된 것이다.

```
I'm task number 1
```

```
gradle -q task0 task1 task2 task3
```

```
I'm task number 0
I'm task number 1
I'm task number 2
I'm task number 3
```

6. 태스크 조작 (Manipulating existing tasks)

이미 생성해놓은 태스크는 API를 통해 접근할 수 있다. 다음은 task를 미리 정의해놓고 의존성을 추가하는 예제이다.

```
4.times { counter ->
    task "task$counter" << {
        println "I'm task number $counter"
    }
}
task0.dependsOn task2, task3 // dependsOn를 사용하여 의존성 추가
```

gradle -q task0 의 결과는 아래와 같다.

I'm task number 2
I'm task number 3
I'm task number 0

의존성 뿐만 아니라 행위도 추가할 수 있다. 아래 예제는 API를 통해서 행위를 추가한다.

```
// hello  태스크를 정의한다.
task hello << {
    println 'hello world!'
}

// task 시작 전에 수행하는 행위 추가
hello.doFirst {
    println 'before'
}

// task 마지막 에 수행하는 행위 추가
hello.doLast {
    println 'after'
}

// doLast(<<) 를 추가 (append 개념으로 이해하면 편해요.)
hello << {
    println 'added'
}
```

gradle -q hello 출력은 다음과 같다.

```
before
hello world!
after
added
```

doFirst와 doLast는 여러 번 실행될 수 있다. 이들을 이용해 task 수행의 시작과 마지막에 행위를 추가할 수 있다. 태스크가 수행되면 action list 의 action들이 순서대로 수행하게 된다.
마지막의 << 는 doLast와 동일한 기능으로 alias라고 보면 된다.

7. 단축 표기법 (shortcut notations)

이미 정의된 태스크에 접근할 수 있는 편리한 표기법이 있다. 바로 빌드 스크립트의 속성(property)을 이용한 것이다.
다음 코드는 빌드 스크립트의 속성을 이용하여 태스크에 접근한다.

```
task test << {
    println 'Hello world!'
}
test.doLast {
    println "Greetings from the $test.name task."
}
```

gradle -q test 의 결과는 다음과 같다. $test.name 처럼 name 속성을 통해 test 태스크의 이름을 출력했다.

```
Hello world!
Greetings from the test task.
```

8. 태스크 속성 확장하기 (extra task properties)

task에 자신만의 property(속성)을 추가할 수 있다. myProperty라는 이름의 속성을 추가하기 위해서는 ext.myProperty 라고 세팅한다. 이렇게 하면 미리 정의된 태스크의 속성처럼 사용할 수 있다.

```
task myTask {
    ext.myProperty = "myValue"
}

task printTaskProperties << {
    println myTask.myProperty
}
```

gradle -q printTaskProperties 결과는 다음과 같다.

myValue

9. 기본 태스크 (Defaults task)

Gradle은 빌드 할 때 기본 태스크를 제공한다.

```
defaultTasks 'clean', 'run'

task clean << {
    println 'Default Cleaning!'
}

task run << {
    println 'Default Running!'
}

task other << {
    println "I'm not a default task!"
}
```

gradle -q를 하면 출력은 다음과 같다.

Default Cleaning!
Default Running!

실행시 어떠한 태스크도 지정하지 않았지만 clean과 run이 동작한다. (gradle clean run 하고 동일하게 작동한다고 보면 된다.)
멀티 프로젝트 빌드에서 모든 서브 프로젝트는 자신의 기본 태스크를 가질 수 있는데 서브 프로젝트가 기본 태스크를 가지고 있지 않다면 부모 프로젝트의 기본 태스크를 사용하게 된다. 이를 활용하면 멀티프로젝트에서 공통으로 사용하는 태스크를 사용할 경우 유용하다.

의존성 관리

이번에는 기초적인 의존성 관리 방법을 java 샘플 프로젝트를 통하여 살펴본다.

java 프로젝트로 살펴보는 의존성 관리

우선 자바 프로젝트 초기화를 위해 gradle init --type java-library 명령어를 실행해본다.

init 태스크는 프로젝트를 처음 시작할때 Gradle에 필요한 파일들을 템플릿처럼 구성해주는 역할을 한다. 태스크의 옵션으로 프로젝트 타입(—type)을 지정할 수 있는데 종류는 다음과 같다.

- pom
- basic (default)
- java-library
- groovy-library
- scala-library

자세한 설명은 gradle help --task init 명령어로 확인할 수 있다.

빌드에 성공한 뒤 build.gradle 파일을 열어보면 아래처럼 빌드 스크립트가 3개의 섹션으로 나눠서 파일이 구성되는 것을 확인할 수 있다.

```
// 1.  java 플러그인 적용
apply plugin: 'java'

// 2.  라이브러리를 다운로드할 Maven central repository
repositories {
    mavenCentral()
}

// 3. 의존성  설정
dependencies {
    compile 'org.slf4j:slf4j-api:1.7.5'
    testCompile 'junit:junit:4.11'
}
```

1. java plugin 적용
java 프로젝트로 정의하기 위해 java 플러그인을 적용한다.

2. 레파지토리
라이브러리를 관리하는 레파지토리를 명시한다. 기본적으로 Maven central repository가 사용된다.
원격 메이븐 레파지토리를 사용하려면 다음과 같이 작성한다.

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

3. 의존성 설정
Gradle 의존성은 설정들이 모여서 구성된다. 다른 말로 의존성은 설정들의 집합이다.
java 플러그인에서는 아래와 같은 의존성 설정이 존재한다. 위 예제에서는 compile과 testCompile만 설정되어 있는 것이다.

- archives : 해당 프로젝트의 결과물(Artifact). uploadArchives 태스크에 의해 실행된다.
- compile : 프로젝트의 소스가 컴파일시 사용되는 설정
- default : 프로젝트에서 사용되는 기본 설정. artifacts와 프로젝트 런타임시 필요한 의존성을 포함한다.
- runtime : classes가 런타임될때 요구되는 설정. 기본적으로 compile 의존성을 포함한다.
- testCompile : 프로젝트의 테스트 소스가 컴파일될 때 요구되는 설정. compile, runtime 의존성을 포함한다.
- testRuntime 테스트가 동작할 때 요구되는 설정. compile,runtime, testCompile 의존성을 포함한다.

프로젝트에 적용된 전체적인 의존성 구조를 확인하려면 dependencies 태스크를 실행해보면 된다.

```
gradle dependencies
```

결과는 다음과 같이 나온다. 각 설정에 의존하는 라이브러리들이 나타나는 것을 알 수 있다.

:dependencies

i. ----------

Root project

i. ----------

archives - Configuration for archive artifacts.
No dependencies

```
compile - Compile classpath for source set 'main'.
₩--- org.slf4j:slf4j-api:1.7.5

default - Configuration for default artifacts.
₩--- org.slf4j:slf4j-api:1.7.5

runtime - Runtime classpath for source set 'main'.
₩--- org.slf4j:slf4j-api:1.7.5

testCompile - Compile classpath for source set 'test'.
+--- org.slf4j:slf4j-api:1.7.5
₩--- junit:junit:4.11
₩--- org.hamcrest:hamcrest-core:1.3

testRuntime - Runtime classpath for source set 'test'.
+--- org.slf4j:slf4j-api:1.7.5
₩--- junit:junit:4.11
₩--- org.hamcrest:hamcrest-core:1.3
```

의존성 설정에 값을 넣을때는 group, name, version을 명시한다.

```
dependencies {
    compile group: 'org.hibernate', name: 'hibernate-core', version: '3.6.7.Final'
}
```

또한 다음과 같이 축약해서 쓸 수 있다.

```
dependencies {
    compile 'org.hibernate:hibernate-core:3.6.7.Final'
}
```

레파지토리에 배포하기

레퍼지토리는 기본적으로 ivy와 maven을 지원한다. 여기서는 maven을 사용해본다.

먼저 jar에 필요한 버전정보와 Manifest정보를 작성한다.

```
sourceCompatibility = 1.5
version = '1.0'
jar {
    manifest {
        attributes 'Implementation-Title': 'Gradle Quickstart', 'Implementation-Version': version
    }
}
```

그리고 레파지토리에 접근하기 위해 Gradle에서 지원하는 uploadArchives 태스크도 작성한다. 여기에 메이븐 레파지토리 정보를 추가해준다.

```
uploadArchives {
    repositories {
        maven {
            credentials {
                username “username"
                password “password"
            }
            url "http://nexus-server/nexus/content/repositories/develop"
        }
    }
}
```

저장 후 gradle uploadArchives 태스크를 실행해본다. 그러면 Gradle은 빌드를 마친 후 artifacts(JAR) 파일이 메이븐 레퍼지토리 서버로 업로드되는 것을 확인할 수 있다.

```
:compileJava UP-TO-DATE
:processResources UP-TO-DATE
:classes UP-TO-DATE
:jar UP-TO-DATE
:uploadArchives
```

```
Upload http://nexus-server/nexus/content/repositories/develop//helloGradle/1.0/helloGradle-1.0.jar
Upload http://nexus-server/nexus/content/repositories/develop//helloGradle/1.0/helloGradle-1.0.jar.sha1
Upload http://nexus-server/nexus/content/repositories/develop//helloGradle/1.0/ivy-1.0.xml
Upload http://nexus-server/nexus/content/repositories/develop//helloGradle/1.0/ivy-1.0.xml.sha1

BUILD SUCCESSFUL
```

참고

http://www.gradle.org/docs/current/userguide/tutorial_using_tasks.html
http://www.gradle.org/docs/current/userguide/artifact_dependencies_tutorial.html

---

링크 목록

- 이미지 출처:http://blogs.perficient.com/businessintelligence/2012/08/01/ - http://blogs.perficient.com/businessintelligence/2012/08/01/iterative-bi-gradle-tips-and-tricks-a-primer-on-gradle-objects/
- http://www.gradle.org/docs/current/userguide/tutorial_using_tasks.html - http://www.gradle.org/docs/current/userguide/tutorial_using_tasks.html
- http://www.gradle.org/docs/current/userguide/artifact_dependencies_tutorial.html - http://www.gradle.org/docs/current/userguide/artifact_dependencies_tutorial.html

관련 키워드

Gradle
