05.5주차.Gant와Grails명령어(TheCommandLine)

Gant 와 Grails 명령어(The Command Line)

Table of Contents

Gant

Gant는 Apache Ant를 간단하게 Groovy로 포장한 것(wrapper)이다. Gant 특성은 Ant script로 불가능한 작업을 Groovy script와 Groovy를 직접적으로 사용하여 할 수 있다. Grails 명령어 시스템은 Gant로 만들었다. Grails는 이것을 편하게 사용할 수 있도록 만들어 단순한 Gant가 아닌 grails 명령어로 취급한다. 명령어는 다음 형식으로 입력한다.

```
includeTargets << gant.targets.Clean
cleanPattern << ['**/*~',  '**/*.bak']
cleanDirectory << 'build'

target(stuff: 'A target to do some stuff.') {
  println 'Stuff'
  depends clean
  echo message: 'A default message from Ant.'
  otherStuff()
}

target(otherStuff: 'A target to do some other stuff') {
  println 'OtherStuff'
  echo message: 'Another message from Ant.'
  clean()
}
```

이 스크립트는 두개의 타겟이 있다. 이 빌드의 기본목표는 타겟이 없는 파라미터 커맨드 라인으로 Gant가 실행 되어질때 stuff를 지정하고 타겟을 실행 시킨다.

- Gant는 자신의 강아지 밥을 먹을수 있다.

Gant는 스스로 building과 install을 하는데 사용되고 Groovy building과 Java Programs의 빌드, Grails의 일부에도 사용된다.XML과 Groovy script를 쉽게 사용할 수 있게 한다.

- Gant는 진짜 빌드 프레임웍이 아니다.

Gant는 Groovy AntBuilder의 가벼운 외관이다. 단지 Groovy를 사용하여 Ant 작업을 하는 방법이다. Gant는 build task를 위해 이용될 수 있지만 통합의존성관리, 프로젝트 라이프사이클 관리 멀티모듈/서브 프로젝트를 완전히 독립적인 빌드 프레임웍으로 지원하진 않는다. 반면에 Gradle은 Groovy와 Ivy에 따라 완전한 빌드 프레임 워크이다. Gant는 Ant task script 툴이지만 Ant와 Maven을 대신하기에는 부족하고 그러기 위해서는 Gradle을 고려해야 한다.Gant는 Gradle build에 의해 관리된다.

명령어

Grails 명령어 시스템은 Gant로 만들었다. Gant는 Apache Ant를 간단하게 Groovy로 포장한 것(wrapper)이다.
Grails는 이것을 편리하게 사용할 수 있도록 만들어 단순한 Gant가 아닌 grails 명령어로 취급한다. 명령어는 다음 형식으로 입력한다.

```
grails [command name]
```

Grails는 명령어를 실행하기 위해 아래 디렉토리를 찾는다.

- USER_HOME/.grails/scripts
- PROJECT_HOME/scripts
- PROJECT_HOME/plugins/*/scripts
- GRAILS_HOME/scripts

또한 Grails는 run-app와 같은 소문자 형태의 이름을 camel case로 변환할 것이다.. 따라서 아래와 같이 명령어를 입력한다.

```
1 grails run-app
```

그러면 다음과 같은 파일들을 찾아낼 것이다.

- USER_HOME/.grails/scripts/RunApp.groovy
- PROJECT_HOME/scripts/RunApp.groovy
- PROJECT_HOME/plugins/*/scripts/RunApp.groovy
- GRAILS_HOME/scripts/RunApp.groovy

만약 매치되는 것이 한 개가 아니라면 Grails는 실행할 것을 하나만 고른다. Gant 스크립트가 실행될 때 "default" 타겟이 실행된다.몇몇 사용 가능한 명령어에 대한 도움말의 목록을 얻으려면 다음과 같이 명령어를 입력한다.

```
1 grails help
```

그러면 간단한 사용법과 Grails가 가지고 있는 명령어들이 화면에 출력된다.

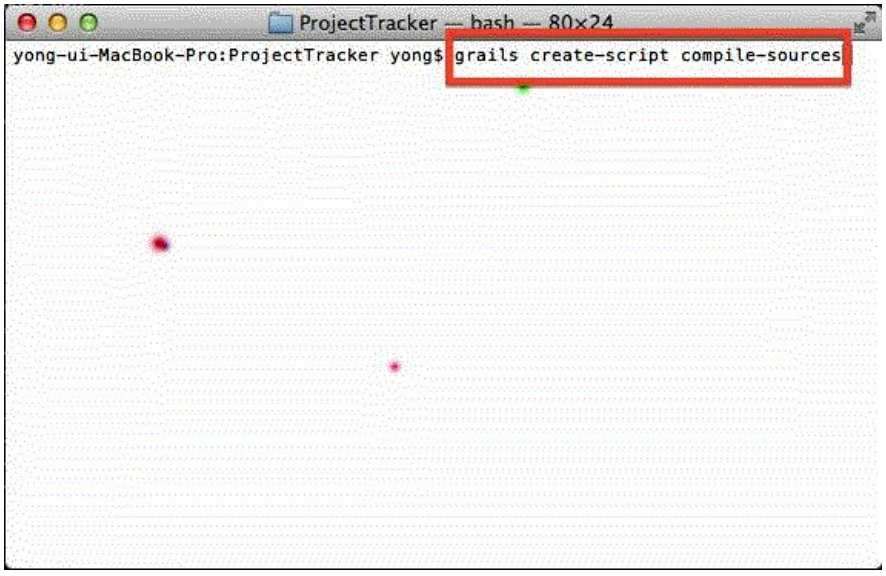
Usage (optionals marked with *):
grails [environment]* [target] [arguments]*
Examples:
grails dev run-app
grails create-app books

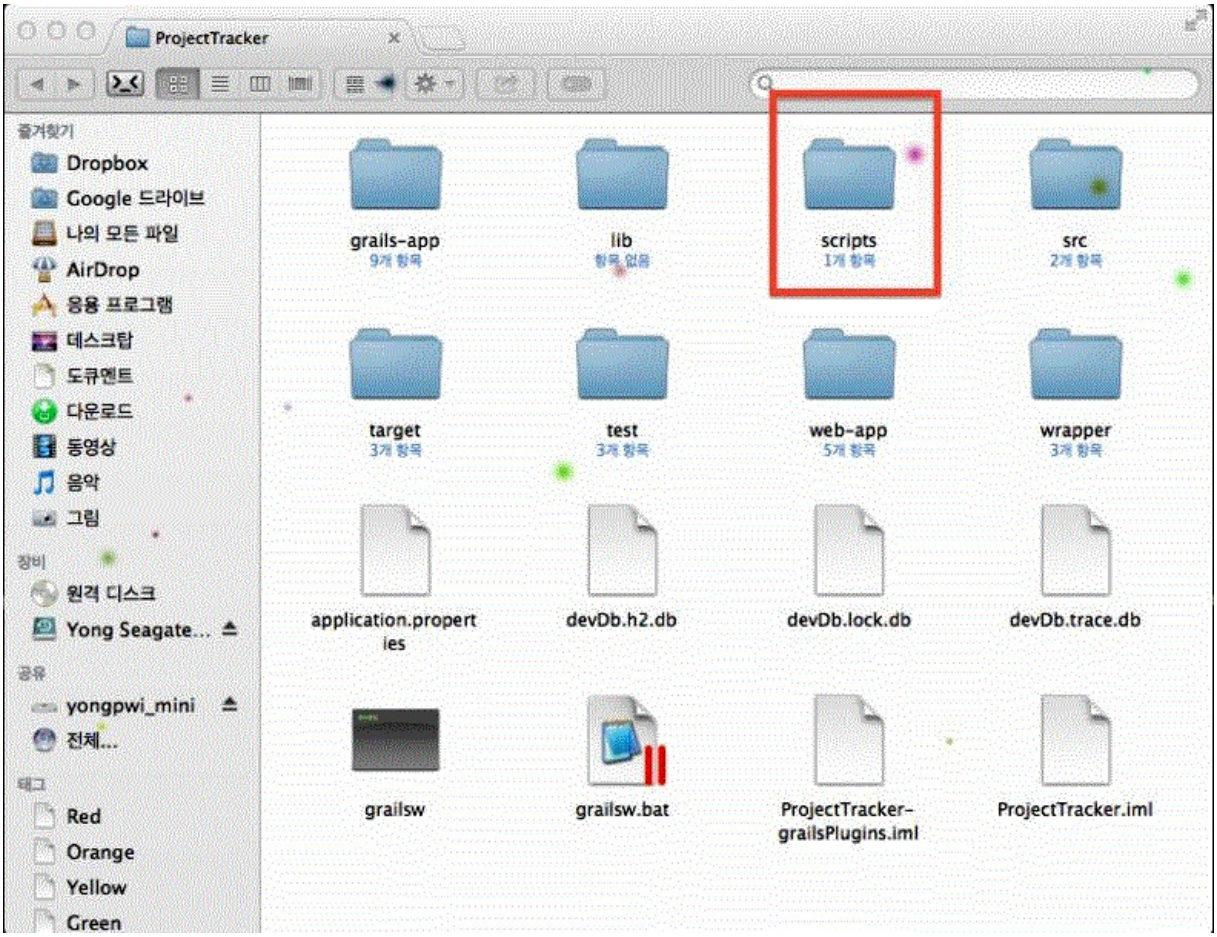
Available Targets (type grails help 'target-name' for more info):
grails bootstrap
grails bug-report
grails clean
grails compile

1 Creating Gant Scripts

현재 프로젝트의 루트에서 create-script 명령어를 사용하여 Gant 스크립트를 만들 수 있다. 터미널에 다음과 같이 명령어를 입력하면

```
1 grails create-script compile-sources
```





상위 이미지와 같이 script/CompileSources.groovy 라는 스크립트가 만들어질 것이다. Gant 스크립트는 "타겟" 이라는 개념을 지원하고 그것에 의존한다는 것만 제외하면 Groovy 스크립트와 유사하다.

- 생성된 코드

```
includeTargets << grailsScript("_GrailsInit")

target(compileSources: "The description of the script goes here!") {
    // TODO: Implement script here
}

setDefaultTarget(compileSources)
```

- 사용 예제

```
target(default:"The default target is the one that gets executed by Grails") {
    depends(clean, compile)
}
target(clean:"Clean out things") {
    Ant.delete(dir:"output")
}
target(compile:"Compile some sources") {
    Ant.mkdir(dir:"mkdir")
    Ant.javac(srcdir:"src/java", destdir:"output")
}
```

위의 스크립트에서 보여준 예처럼 Gant 스크립트에는 암묵적으로 Apache Ant API 에 접근 할 수 있는 Ant의 변수가 사용된다.

2 Re-using Grails scripts

Grails에는 재사용하기 매우 좋은 명령어들이 내장돼 있다. 가장 유용한 것들은 compile, package, bootstrap 스크립트이다.
bootstrap 스크립트는 스프링의 ApplicationContext 인스턴스를 통해서 데이터 소스등에 접근할 수 있게 해준다.

```
Ant.property(environment:"env")
grailsHome = Ant.antProject.properties."env.GRAILS_HOME"
includeTargets << new File ( "${grailsHome}/scripts/Bootstrap.groovy" )


target ('default': "Load the Grails interactive shell") {
    depends( configureProxy, packageApp, classpath, loadApp, configureApp )

    Connection c
    try {
        // do something with connection
        c = appCtx.getBean('dataSource').getConnection()
    }
    finally {
        c?.close()
```

```
    }
}
```

3 Hooking into Events

Grails는 스크립트 이벤트를 가로채는 방법을 제공한다. Grails의 타겟과 플러그인 스크립트가 실행되는 동안 이벤트가 발생한다.
이 매커니즘은 꽤 단순하고 유연하게기술된다. 가로챌 이벤트의 목록은 정해진 것이 아니다. 중요한 타겟target 스크립트에는 동일한 이벤트가 없기 때문에 플러그인 스트립트가 발생시킨 이벤트들을 가로챌 수 있다.

Defining event handlers(이벤트 핸들러 정의하기)

이벤트 핸들러는 플러그인의 scripts/ 폴더나 USER_HOME 디렉토리의 .grails/scripts/ 폴더에 있는 Events.groovy라는 스크립트에 정의한다. 모든 이벤트 스크립트는 이벤트가 발생할 때마다 호출된다. 그래서 이벤트를 처리하는 플러그인을 10개를 만들 수도 있고 사용자마다 다르게 할 수도 있다.

```
eventCreatedArtefact = { type, name ->
    println "Created $type $name"
}
eventStatusUpdate = { msg ->
    println msg
}

eventStatusFinal = { msg ->
    println msg
}
```

Triggering events

Init.groovy script와 호출된 event()클로저 간단하게 추가된 이벤트 트리거? == 문장이 미묘

```
Ant.property(environment:"env")
grailsHome = Ant.antProject.properties."env.GRAILS_HOME"
includeTargets << new File ( "${grailsHome}/scripts/Init.groovy" )

event("StatusFinal", ["Super duper plugin action complete!"])
```

Common Events

Event	Parameters	Description
StatusUpdate	message	Passed a string indicating current script status/progress
StatusError	message	Passed a string indicating an error message from the current script
StatusFinal	message	Passed a string indicating the final script status message, i.e. when completing a target, even if the target does not exit the scripting environment
CreatedArtefact	artefactType,artefactName	Called when a create-xxxx script has completed and created an artefact
CreatedFile	fileName	Called whenever a project source filed is created, not including files constantly managed by Grails
Exiting	returnCode	Called when the scripting environment is about to exit cleanly
PluginInstalled	pluginName	Called after a plugin has been installed
CompileStart	kind	Called when compilation starts, passing the kind of compile - source or tests
CompileEnd	kind	Called when compilation is finished, passing the kind of compile - source or tests
DocStart	kind	Called when documentation generation is about to start - javadoc or groovydoc
DocEnd	kind	Called when documentation generation has ended - javadoc or groovydoc
SetClasspath	rootLoader	Called during classpath initialization so plugins can augment the classpath with rootLoader.addURL(...). Note that this augments the classpath after event scripts are loaded so you cannot use this to load a class that your event script needs to import, although you can do this if you load the class by name.
PackagingEnd	none	Called at the end of packaging (which is called prior to the Jetty server being started and after web.xml is generated)
ConfigureJetty	Jetty Server object	Called after initial configuration of the Jetty web server.

4 Ant and Maven

Ant Integration

create-app 명령어로 Grails 어플리케이션을 만들때 Grails는 자동으로 create-app의 build.xml 파일을 생성한다. 여기에는 다음과 같은 타겟들이 포함되어있ㅁ다.

- clean - Grails 어플리케이션을 청소한다.
- war - WAR 파일을 만든다.
- test - 유닛 테스트를 실행한다.
- deploy - 기본적으로 아무일도 하지 않지만 자동 설치deployment를 구현할 수 있다.

이 타겟들은 Ant에 의해 실행된다.

```
1 ant war
```

build.xml은 Grails의 일반적인 명령어들을 실행하고 CruiseControl이나 Hudson같은 CI(continuous integration) 서버와 통합하는데 사용될 수 있다.

Maven Integration

Grails는 공식적으로 Maven을 지원하지 않는다. 하지만 Grails를 위한 Maven Tools for Grails라는 프로젝트가 있다. 이 도구는 이 기존의 Grails 프로젝트를 위한 POM을 만들어줄 뿐만아니라 라이프사이클을 Grails에서 사용할 수 있게 해준다.
현재는 0.3 이후로 개발되지 않고 있다.

링크 목록

- create-app - http://dogfeet.github.io/grails-doc/ref/Command%20Line/create-app.html
- CruiseControl - http://cruisecontrol.sourceforge.net/
- Hudson - https://hudson.dev.java.net/
- Maven - http://maven.apache.org/
- Maven Tools for Grails - http://forge.octo.com/confluence/display/MTG/Home

---