Introduction To Gradle

John Engelman

Object Partners Inc

@johnrengelman
github.com/johnrengelman

Topics

1. What is Gradle?
2. Getting Started With Gradle
3. Building JVM Projects

What is Gradle?

1. Expressive, declarative, & maintainable build language
2. Dependency Resolver & Manager
3. Build Task Scheduler & Executor
4. Build By Convention

Gradle is an opinionated framework on top of an unopinionated toolkit
- Szczepan Faber

What Gradle is NOT!

It is **NOT** Groovy Ant!

(That tool exists -> GANT)

Core Gradle Features

1. Build-By-Convention w/ Flexibility
2. Project & Build Groovy DSL
3. Support for Ivy & Maven Dependencies
4. Multi-Project Builds
5. Easy to add custom logic
6. 1st class integration w/ Ant builds
7. Extensive public API and plugin ecosystem
8. Task UP-TO-DATE checking

The Quick & Dirty

A Typical Maven Build

- 11.28s (`mvn package`)
- 2.061s (`rm -r target && mvn package`)
- ~35 lines

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.o
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.o
  <modelVersion>4.0.0</modelVersion>
  <groupId>de.uulm.vs</groupId>
  <artifactId>netty-example</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>netty-example</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.jboss.netty</groupId>
      <artifactId>netty</artifactId>
      <version>3.2.2.Final</version>
```

The Same Build w/ Gradle

- 11.07s (`gradle build`)
- 2.161s (`rm -r build/ && gradle build`)
- ~13 lines

```
apply plugin: 'java'
apply plugin: 'maven'

group = 'de.uulm.vs'
version = '1.0-SNAPSHOT'

repositories {
  jcenter()
}

dependencies {
  compile 'org.jboss.netty:netty:3.2.2.Final'
  testCompile 'junit:junit:3.8.1'
}

targetCompatibility = '1.6'
sourceCompatibility = '1.6'
```

Getting Started w/ Gradle

Installing Gradle

1. Install GVM - http://gvmtool.net/
2. `gvm install gradle 2.0`

Starting a project

1. `mkdir todo && cd todo`
2. Initialize project
    - Create & edit `build.gradle`
    - `gradle init --type groovy-library`
    - Convert existing Maven pom.xml: `gradle init`

Command Line Gradle

- List Available tasks

```
$ gradle tasks
```

```
:tasks

------------------------------------------------------------
All tasks runnable from root project
------------------------------------------------------------

Build tasks
-----------
assemble - Assembles the outputs of this project.
build - Assembles and tests this project.
buildDependents - Assembles and tests this project and all projects that depe
buildNeeded - Assembles and tests this project and all projects it depends on
classes - Assembles classes 'main'.
clean - Deletes the build directory.
jar - Assembles a jar archive containing the main classes.
testClasses - Assembles classes 'test'.
...
```

- Using the Gradle wrapper

```
$ ./gradlew tasks
```

- Passing Properties
  - `-Dmyprop=myvalue`: JVM System Properties for Gradle JVM
  - `-Pprojectprop=projectval`: Gradle Project properties
- Specify Build File (default: `build.gradle`)
  - `-b <path to build file>`
- Logging
  - `-i, --info`: Log more Gradle information
  - `-d, --debug`: Log more information than Info
  - `-s, --stacktrace`: Log stacktrace on error
  - `-q, --quiet`: Log errors only (or printlns)

Gradle Tasks

- Single, atomic piece of work
- Consists of a list of "actions"
  - Each "action" is a `org.gradle.api.Action`
  - `Closure` is coerced into `Action`

```
task helloWorld { //defines a new task with name 'helloWorld'
  doLast { //add action to the end of the action list
    println 'Hello World!'
  }
}
```

- Build script is backed by a Gradle `Project` instance
  - `task` is method from Gradle DSL (`org.gradle.api.Project.task(String name, Closure configure)`)
- `<< {..}` is shorthand for `doLast {..}`

```
task helloWorld << {
  println 'Hello World!'
}
```

Task Dependencies & Ordering

- `dependsOn` creates an execution dependency
- All execution dependencies of a task must also be executed and completed before the task

```
task a << { println 'a' }
task b(dependsOn: a) << { println 'b' }
```

```
$ gradle b
:a
a
:b
b
```

- `finalizedBy` creates a finalization dependency
- The finalizer task is added if the finalized task is present in the task graph
  - finalizer will execute after the finalized even when finalized fails
  - finalizer will not execute if finalized did no work or was UP-TO-DATE

```
task cleanup << { println 'cleanup' }
task run << println 'run'

run.finalizedBy cleanup // 'run' is the "finalized" task, 'cleanup' is the "f
```

```
$ gradle run
:run
run
:cleanup
cleanup
```

- Task ordering allows you to specify ordering w/ creating an execution dependency
- `mustRunAfter`
  - Task A runs after Task B only if both are in the task graph
  - Always respected
- `shouldRunAfter`
  - Same as `mustRunAfter` but less strict
  - Ignored if
    - Creates an ordering cycle
    - When executing in parallel and all other dependencies are completed except the `shouldRunAfter`

```
task first << { println 'first' }
task second << { println 'second' }
second.mustRunAfter first
```

```
$ gradle second
:second
second

$ gradle first
:first
first

$ gradle second first
:first
first
:second
second
```

- Gradle is Groovy!

```
task ready() << {
  println 'Ready'
}

3.times { num ->
  task "count${num+1}" << {
    println num+1
  }
}

task go() << {
  println 'Go!'
}

count3.dependsOn ready
count2.dependsOn count3
count1.dependsOn count2
go.dependsOn count1

task countdown(dependsOn: go)
```

```
$ gradle countdown -q
Ready
3
2
1
Go!
```

- Task name shortening

```
$ gradle hW
:helloWorld
Hellow World!

BUILD SUCCESSFUL

Total time: 0.673 secs
```

- Excluding Tasks

```
$ gradle countdown -q -x ready
3
2
1
Go!
```

- Task Rules
  - Dynamically creates tasks based on the requested task name

```
tasks.addRule('Pattern: countdown<From>') { String taskName ->
  if (taskName.startsWith('countdown')) {
    task(taskName) << {
      ((taskName - 'countdown').toInteger()..0).each {
        println it
      }
    }
  }
}
```

```
$ gradle countdown5
:countdown5
5
4
3
2
1
0

$ gradle countdown2
:countdown2
2
1
0
```

Task Inputs & Outputs

- Gradle tracks the inputs & outputs of a task
- Compares current inputs & outputs against previous runs
  - If the same, task is considered UP-TO-DATE and is skipped
- Inputs consist of files and map of properties (String: Object)
- Outputs consists of files
- Tasks with no outputs, are never considered UP-TO-DATE and always executed
- Tasks with outputs but no inputs is considered UP-TO-DATE if the output hasn't changed

Gradle Properties

- Gradle Properties are loaded by the project
  - From `gradle.properties` in the rootDir of the project
  - From the command line w/ `-Pproperty=value`
- Gradle Properties are inherited by child project and merged with child's properties

```
//gradle.properties
currentVersion=1.0

//build.gradle
apply plugin: 'java'
version = currentVersion
```

Extra Properties

- Every enhanced object in the Gradle Domain Model can be extends via Extra Properties
- Each object has a `ext` property to access the space
- Initially define using `ext.<propertyName>`, then treat like project property

```
ext.foo = 'bar'
task echoFoo << { println foo }

task baz {
  ext.foo = 'baz'
  doLast {
    println foo
  }
}
```

```
$ gradle echoFoo echoFoo2
:echoFoo
bar
:echoFoo2
baz
```

Variables

Build scripts are Groovy, so define variables normally

```
def taskNames = ['foo', 'bar', 'baz']

taskNames.each { name ->
  tasks.create(name) << { println name }
}
```

Gradle Daemon

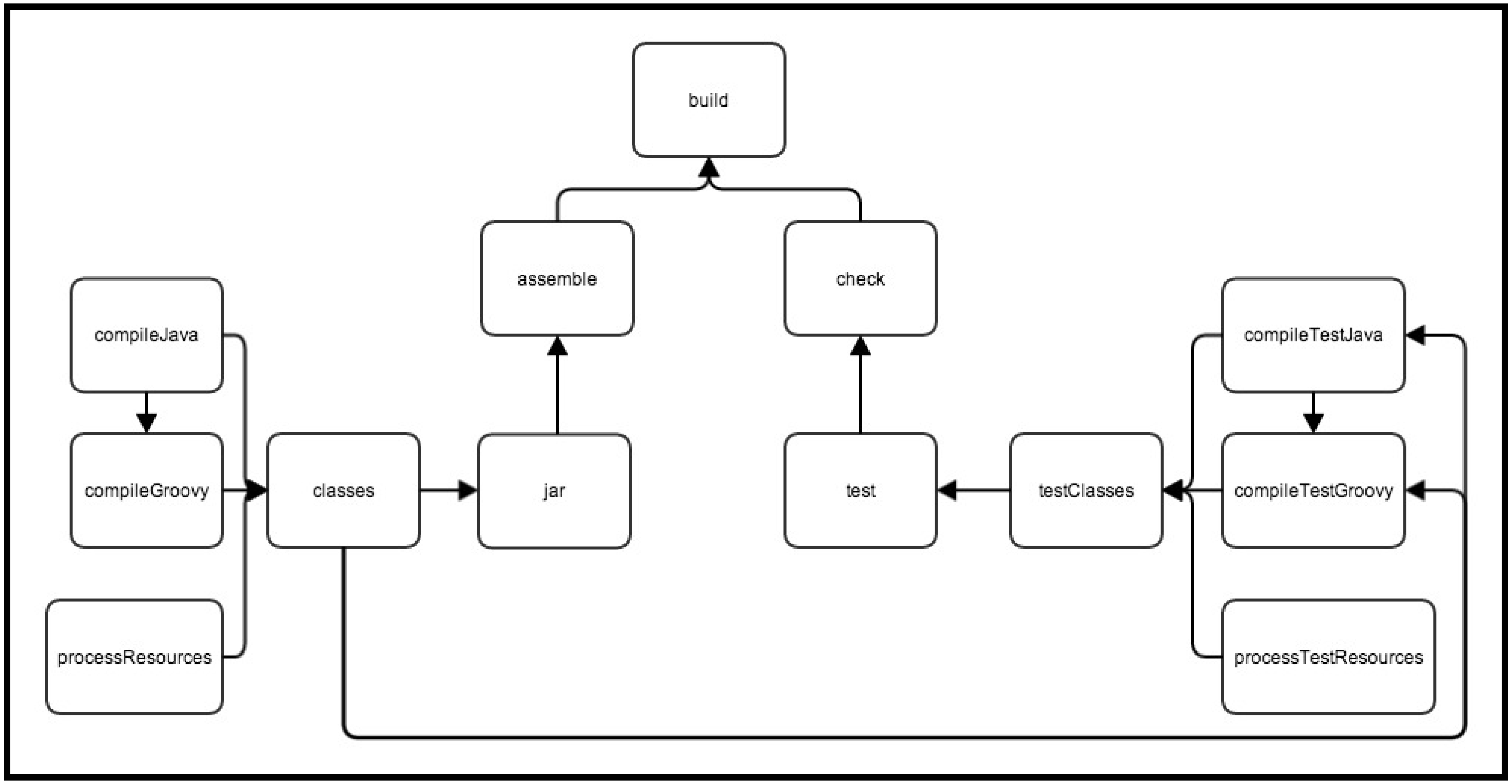
- Reduce startup hit by keeping a live JVM
- Expires after 3 hours
- `gradle --daemon`
- `echo "org.gradle.daemon=true" >> ~/.gradle/gradle.properties`

Debugging Gradle builds

```
$ gradle build -Dorg.gradle.debug=true
```

- Attach debugger to port 5005.

Basics of a JVM Project Build



- Build a Java/Groovy project

```
$ gradle build
```

```
:compileJava UP-TO-DATE
:compileGroovy
:processResources UP-TO-DATE
:classes
:jar
:assemble
:compileTestJava UP-TO-DATE
:compileTestGroovy
:processTestResources UP-TO-DATE
:testClasses
:test
:check
:build

BUILD SUCCESSFUL

Total time: 6.213 secs
```

Adding Dependencies

- Dependencies are added to a `Configuration`
- JVM projects have
  - compile, runtime, testCompile, testRuntime
- Configurations can extend other repositories
  - compile extends runtime

```
dependencies {
  compile 'org.codehaus.groovy:groovy-all:2.3.3'
  runtime 'mysql:mysql-connector-java:5.1.31'

  testCompile 'org.spockframework:spock-core:0.7-groovy-2.0'
  testRuntime 'com.h2database:h2:1.4.180'
}
```

Locating Dependencies

- Gradle supplies some default repositories to search:
  - jcenter, mavenCentral, mavenLocal
- Can also look at custom maven, ivy, and file system paths

```
repositories {
  mavenLocal()
  jcenter()
  mavenCentral()
  maven {
    url 'http://maven.myhost.com/'
    credentials {
      username 'username'
      password 'password'
    }
  }
  ivy {
    url 'http://ivy.myhost.com'
  }
  flatDir {
    dir file('repo')
  }
}
```

Project structure

- Follows the Maven convention

```
+ <projectDir>/
+-- src/
    +-- main/
    |   +-- java/
    |   +-- groovy/
    |   +-- resources/
    +-- test/
        +-- java/
        +-- groovy/
        +-- resources/
```

- But can be configured

```
sourceSets.main.java.srcDirs = ['src']
sourceSets.test.java.srcDirs = ['test']
```

- Project structure is tied to `SourceSets`
- JVM projects have 2 - `main` & `test`
  - Each `SourceSet` starts w/
    - `java`
    - `resources`

```
apply plugin: 'java'

sourceSets {
  main {
    java { ... }
    resources { ... }
  }
  test {
    java { ... }
    resources { ... }
  }
}
```

- Plugins can extends the SourceSet
  - Gradle plugin adds `<sourceSet>/groovy`
  - Scala adds `<sourceSet>/scala`

```
apply plugin: 'groovy'
apply plugin: 'scala'

sourceSets {
  main {
    groovy { ... }
    scala { ... }
  }
  test {
    groovy { ... }
    scala { ... }
  }
}
```

- Can declare additional `SourceSets` (i.e "intTest")

```
sourceSets {
  intTest
}
```

- Gradle automatically creates compile/runtime configurations for source sets

```
// No need to declare this
configurations {
  intTestCompile
  intTestRuntime
}
```

Scripting Builds

- Extend builds through plugins
- Plugins come in 3 flavors
    - Core Plugins: shipped w/ Gradle
    - Script Plugins
    - 3rd Party Plugins: resolved from outside sources

Creating Script Plugins

- Configure project, create tasks and set up dependencies in a filed
- Apply the file to your project

```
//docs.gradle
task javadocJar(type: Jar, dependsOn: javadoc) {
    classifier = 'javadoc'
    from 'build/docs/javadoc'
}

task sourcesJar(type: Jar) {
    classifier = 'sources'
    from sourceSets.main.allSource
}

build.dependsOn javadocJar, sourcesJar

//build.gradle
apply plugin: 'groovy'
apply from: file('docs.gradle')
```

Adding 3rd party plugins

- Plugins must be available to Gradle itself
  - They are not project dependencies, need something else.
  - Configure Gradle's executiong using `buildscript {}`
  - Similar to configure a normal project
- Find plugins at Gradle Plugin Portal - http://plugins.gradle.org

```
buildscript {
  repositories {
    jcenter()
  }
  dependencies {
    classpath 'org.github.jengelman.gradle.plugins:shadow:1.0.2'
  }
}

apply plugin: 'java'
apply plugin: 'com.github.johnrengelman.shadow'
```

Gradle 2.1 Plugins DSL

- New plugin resolution via Gradle Plugin Portal & Bintray

```
plugins {
  id 'com.github.johnrengelman.shadow' version '1.0.2'
}
```

Classpath Isolation

- All plugins are evaluated & executed with an isolated classloader
- Each classloader inherits from its parent classloader
  - Apply a plugin in build.gradle and all subsequent script plugins can use classes from it
  - Apply a plugin in a script plugin, then those classes are isolated to that script

```
//shadow.gradle
buildscript {
  repositories {
    jcenter()
  }
  dependencies {
    classpath 'com.github.jengelman.gradle.plugins:shadow:1.0.2'
  }
}
import com.github.jengelman.gradle.plugins.shadow.ShadowPlugin
apply plugin: ShadowPlugin

//build.gradle
apply plugin: 'groovy'
apply from: file('shadow.gradle')

import com.github.jengelman.gradle.plugins.shadow.tasks.ShadowJar
task customShadow(type: ShadowJar) //NoClassDefFoundException or MissingPrope
```

Multi-Project Builds

Define sub-projects in `settings.gradle`

```
+<projectDir>
+-- api/
|   +-- build.gradle
+-- client/
|   +-- build.gradle
+-- server/
    +-- build.gradle
```

```
// settings.gradle
include "api", "client", "server"
```

```
$ gradle projects
------------------------------------------------------------
Root project
------------------------------------------------------------

Root project 'todo'
+--- Project ':api'
+--- Project ':client'
\--- Project ':server'
```

Declaring dependencies on projects

```
// server/build.gradle
dependencies {
  compile project(":api") //depends on artifacts of the 'default' configurati
}
```

Enabling Parallel Builds

```
$ gradle build --parallel
```

OR

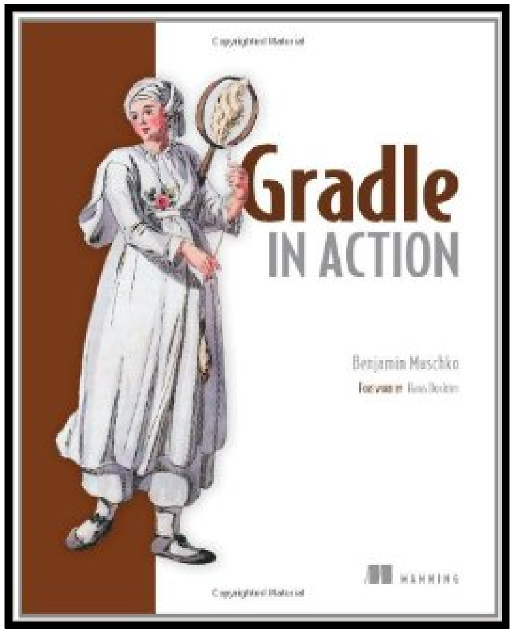
```
// gradle.properties or ~/.gradle/gradle.properties
org.gradle.parallel=true
```

Gradle References

Gradle User Guide:
http://www.gradle.org/docs/current/userguide/userguide.html

Gradle DSL Reference: http://www.gradle.org/docs/current/dsl/

Gradle In Action (Benjamin Muschko)



OPI

- Java, Groovy, Javascript, Mobile, Open Source
- ~ 100 Senior Consultants
  - Minneapolis, MN & Omaha, NE
  - Chicago, IL & Denver, CO
  - Average tenure > 5 years
- Founded in 1996

objectpartners.com
@objectpartners
objectpartners.com/blog