FRÅN MAVEN TILL GRADLE

JESPER HOLMBERG

2015-01-28 | CALLISTAENTERPRISE.SE

CALLISTA
— ENTERPRISE —

VARFÖR GRADLE?

- Är enkelt att lära sig för den som arbetat med Maven.
- Kan använda befintliga maven-repon för beroenden.
- Kan reducera antalet rader i byggfiler med 75-80%.
- Kan konfigureras med ett riktigt JVM-språk.
- Är designat för att anpassas efter projektets behov – inte tvärtom.
- Kan uttrycka komplexa beroenden på ett flexibelt sätt.
- Kan bygga ett stort projekt parallellt på flera kärnor.
- Utvecklas snabbt just nu.
- Är roligt att arbeta med!

KOD-EXEMPEL 1

LIKHETER MELLAN MAVEN OCH GRADLE

- Fördefinierad filstruktur (“src/main/java”, “src/test/java”)
- Maven-repositories
- Klassificering av beroenden (grupp, modul, version)
- Förinstallerade byggsteg (gradle tasks) för det man vill göra (compile, test, jar, …)
- Beroende-kedja mellan byggsteg
- Flera uppsättningar classpaths: scope i maven, configuration i gradle.
- Funktionaliteten kan utökas genom pluginer.

SKILLNADER MELLAN MAVEN OCH GRADLE

- Gradle är språk-agnostiskt: t.o.m. java-stöd måste laddas genom en plugin.
- Gradle har inte mavens *faser*, istället en beroende-graf av *tasks* som liknar ants *targets*.
- Gemensam funktionalitet modelleras explicit i underprojekt, snarare än genom arv till ett föräldra-projekt.
- Många problem som man i maven löser med en plugin löser man i gradle med några rader groovy-kod.
- Gradle har ett strukturerat API i flera lager som gör det lätt att lösa specifika problem med elegant programmatisk kod.

SKILLNADER MELLAN MELLAN MAVEN OCH GRADLE

- Groovy! Gradle är språk-baserat snarare än plugin-baserat.
- Många saker som man i maven löser med en plugin löser man i gradle med några rader groovy-kod.
- Deklarativ kod är bättre än programmatisk kod, men i gradle är även programmatisk kod OK.
- Gradle har ett strukturerat API i flera lager som gör det lätt att lösa specifika problem med elegant programmatisk kod.

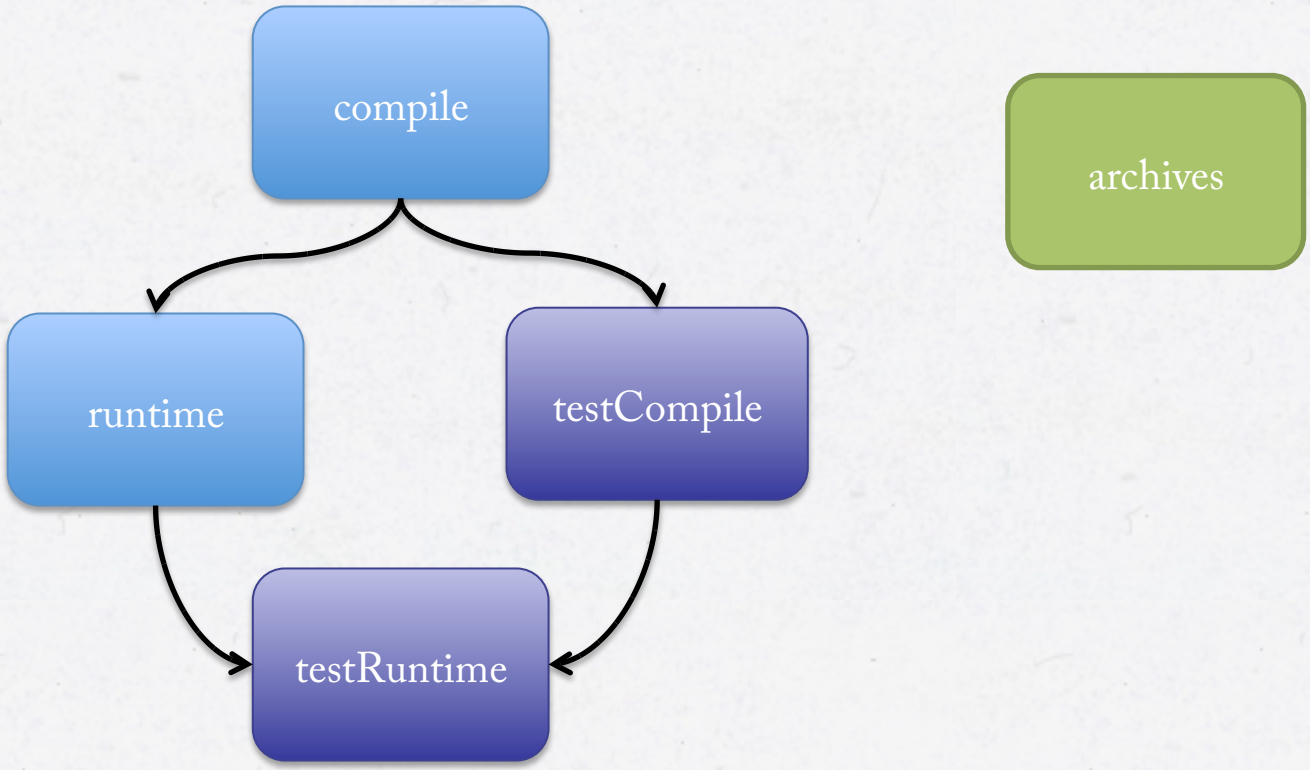
GRADLE BASICS: CONFIGURATIONS

- *Configuration* är ett viktigt begrepp i gradle-modellen. Begreppet liknar *scope* i maven.
- En configuration är en gruppering av beroenden, liknande en classpath.
- Olika configurations används vid olika uppgifter i bygget.
- Varje configuration har sin egen anpassning, t.ex. om dess beroenden är transitiva.
- En configuration skapas antingen via en plugin eller direkt i byggfilen.

CONFIGURATIONS

- Java-pluginen definierar ett antal viktiga configurations: *compile*, *runtime*, *testCompile*, *testRuntime* och *archives*
  - compile används vid kompilering av produktionskoden.
  - runtime används vid körning av produktionskoden.
  - testCompile används vid kompilering av tester.
  - testRuntime används för körning av tester.
  - archives representerar de byggda artefakterna (jar-fil, ear-fil etc)

CONFIGURATIONS DEFINIERADE I JAVA-PLUGINEN



KOD-EXEMPEL 2

FRÅN MAVEN TILL GRADLE

- Automatkonverteringen av vårt projekt har bara givit oss en första skiss att fylla i.
- Skissen innehåller ett par grundläggande gradle-pluginer som *java* och *war* men inga maven-pluginer har översatts.
- Funktionalitet från mavens pluginer kan vi antingen översätta med en gradle-plugin eller genom groovy-kod i bygg-filerna.
- Första exemplet är java-kods-generering från wsdl-filer.

WSDL-GENERERING I ANT

```
<target name="WSDLToJava">
   <java classname="org.apache.cxf.tools.wsdlto.WSDLToJava" fork="true">
      <arg value="-validate"/>
      <arg value="-d"/>
      <arg value="src/generated"/>
      <arg value="resources/hsa.wsdl"/>
      <classpath>
         <path refid="cxf.classpath"/>
      </classpath>
   </java>
</target>

<path id="cxf.classpath">
   <pathelement location="lib/cxf-2.7.9.jar"/>
</path>
```

WSDL-GENERERING I MAVEN

```
<plugin>
  <groupId>org.apache.cxf</groupId>
  <artifactId>cxf-codegen-plugin</artifactId>
  <version>2.7.9</version>
  <executions>
    <execution>
      <id>generate-sources</id>
      <phase>generate-sources</phase>
      <configuration>
        <sourceRoot>src/generated</sourceRoot>
        <wsdlOptions>
          <wsdlOption>
            <extraargs>
              <extraarg>-validate</extraarg>
            </extraargs>
            <wsdl>src/main/resources/hsa.wsdl</wsdl>
          </wsdlOption>
        </wsdlOptions>
      </configuration>
      <goals>
        <goal>wsdl2java</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

WSDL-GENERERING I GRADLE (PLUGIN)

```
apply plugin: 'no.nils.wsdl2java'

wsdl2java {
    generatedWsdlDir = file("src/generated")
    wsdlsToGenerate = [ ['src/main/resources/hsa.wsdl'], ['-validate', '-d', 'src/generated'] ]
    cxfVersion = "2.7.9"
}
```

WSDL-GENERERING I GRADLE (“FÖR HAND”)

```
task wsdl2Java(type: JavaExec) {
    classpath configurations.wsdlgen
    main "org.apache.cxf.tools.wsdlto.WSDLToJava"
    args "-validate", '-d', "src/generated", "src/main/resources/hsa.wsdl"
}

compileJava.dependsOn(wsdl2Java)

sourceSets.main.java { srcDir "src/generated” }

configurations { wsdlgen }

dependencies { wsdlgen "org.apache.cxf:cxf-tools-wsdlto-frontend-jaxws:2.7.9" }
```

FITNESSE

- Ett exempel på en plugin som inte finns (än) är Fitnesse, som är ett verktyg för automatiserade acceptanstester.
- För att skriva Fitnesse-tester startar man en Fitnesse-wiki-server. Från wiki-sidan kan man sedan starta testerna.
- Vi behöver två classpather:
  - För att starta Fitnesse-servern behöver vi en fitnesse-configuration.
  - För testerna behöver vi en classpath med vår egen kod och alla beroenden, i en systemvariabel vars värde ser ut så här:
    ”!path /opt/project/…/project.jar
    !path /opt/project/…/lib1.jar
    !path /opt/project/…/lib2.jar”

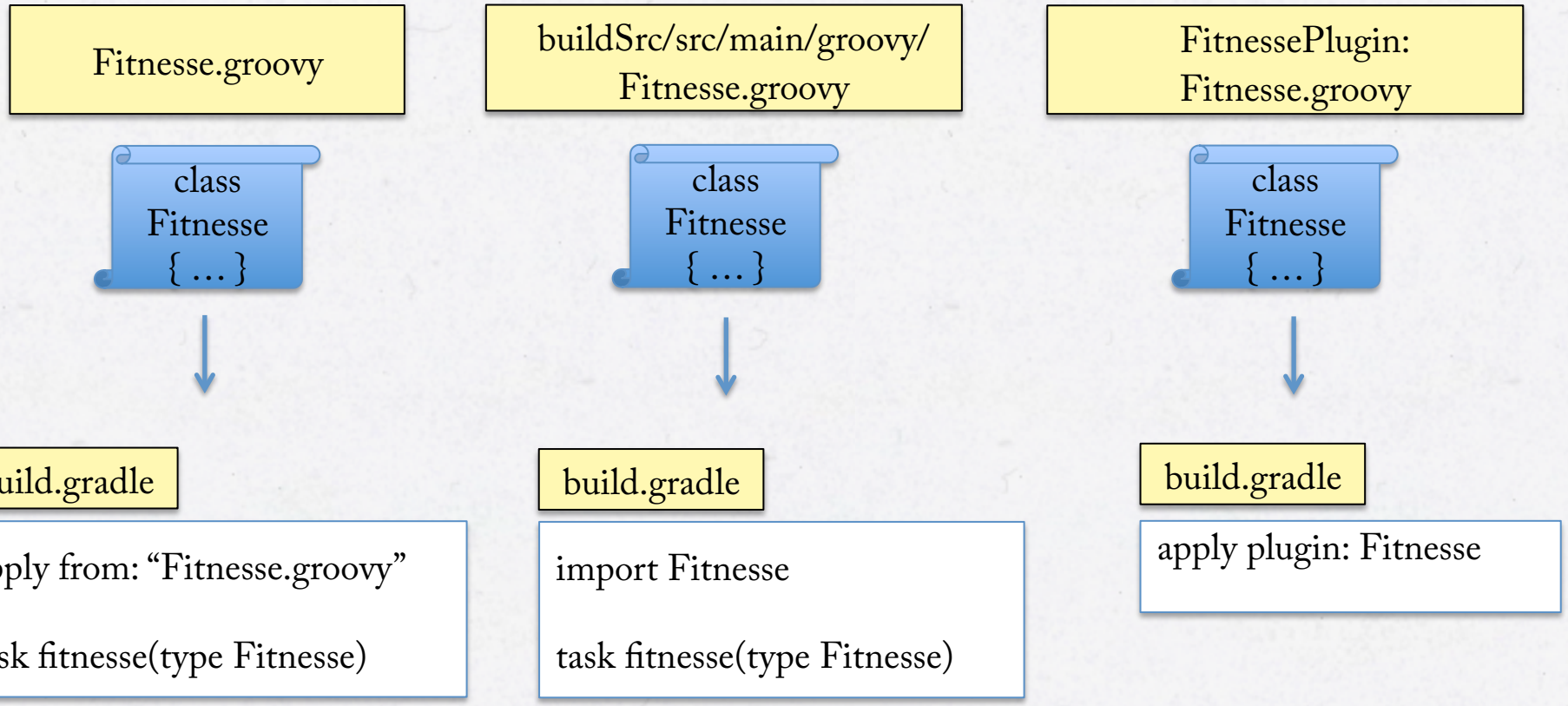
build.gradle

```
task fitnesseWiki << {
    project.javaexec {
        main = "fitnesse.FitNesse"
        classpath = configurations.fitnesse
        systemProperties = ['test.classpath': testPathForWiki()]
        args = ['-p', 9125, '-d', 'src/test/fitnesse', '-o']
    }
}

def testPathForWiki() {
    (configurations.archives.artifacts.files + configurations.runtime.asFileTree).collect
        { file -> "!path ${file.path}" }.join("\n")
}

configurations { fitnesse }

dependencies { fitnesse "org.fitnesse:fitnesse:20140901" }
```

build.gradle

```
ext.fitnessePort = 9125
ext.fitnesseDir = "src/test/fitnesse"

class Fitnesse extends DefaultTask {
    @TaskAction def runFitnesse() {
        project.javaexec {
            main = "fitnesse.FitNesse"
            classpath = project.configurations.fitnesse
            systemProperties = ["test.classpath": testPathForWiki()]
            args = ['-p', project.fitnessePort, '-d', project.fitnesseDir, '-o']
        }
    }

    def testPathForWiki() { ... }
}

task fitnesseWiki(type: Fitnesse)

configurations { fitnesse }

dependencies { fitnesse "org.fitnesse:fitnesse:20140901" }
```

EN TASK-TYP KAN ÅTERANVÄNDAS



BUILDSRC-KATALOGEN

- All kod som läggs i rot-projektets buildSrc-kod kompileras automatiskt och finns tillgänglig i bygg-filerna.
- Koden i buildSrc kan vara skriven i det JVM-språk man föredrar.
- Med koden i buildSrc kan gemensam funktionalitet delas på många ställen i bygg-filerna.

buildSrc/src/main/groovy/Fitnesse.groovy

```groovy
import org.gradle.api.DefaultTask
import org.gradle.api.tasks.TaskAction

class Fitnesse extends DefaultTask {
    @TaskAction def runFitnesse() {
        project.javaexec {
            main = "fitnesse.FitNesse"
            classpath = project.configurations.fitnesse
            systemProperties = ["test.classpath": testPathForWiki()]
            args = ['-p', project.fitnessePort, '-d', project.fitnesseDir, '-o']
        }
    }

    def testPathForWiki() { ... }
}
```

build.gradle

```
import Fitnesse

task fitnesseWiki(type: Fitnesse)

ext.fitnessePort = 9125
ext.fitnesseDir = "src/test/fitnesse”

configurations { fitnesse }

dependencies { fitnesse "org.fitnesse:fitnesse:20140901" }
```

EN EGEN PLUGIN

- Genom att skriva en egen plugin kan man distribuera gemensam funktionalitet till många projekt.
- Ett plugin-projekt är ett vanligt gradle-projekt, skrivet i groovy eller java (eller scala, eller clojure)
- Några filer behöver finnas på plats för att gradle ska kunna koppla in pluginen på rätt sätt:
  - En klass som implementerar interfacet Plugin.
  - En fil som anger vad pluginen ska kallas i bygg-filerna.
  - Task-klasser som implementerar funktionaliteten.

KOD-EXEMPEL 3

build.gradle

```
buildscript {
    dependencies {
        classpath 'se.test:gradle-fitnesse-plugin:1.0-SNAPSHOT'
    }
}

apply plugin: 'fitnesse'

task fitnesseWiki(type: Fitnesse)

ext.fitnessePort = 9125
ext.fitnesseDir  = "src/test/fitnesse"

configurations { fitnesse }

dependencies { fitnesse "org.fitnesse:fitnesse:20140901" }
```

KOD-EXEMPEL 4

build.gradle

```
buildscript {
    dependencies {
        classpath 'se.test:gradle-fitnesse-plugin:1.0-SNAPSHOT'
    }
}

apply plugin: 'fitnesse'

fitnesse {
    port = 9125
    workingDir  = "src/test/fitnesse"
}
```

VAD ÅTERSTÅR?

- Ersätt all funktionalitet med plugin eller egna tasks.
- Samla gemensam funktionalitet i subprojects {}.
- Städa beroenden, ta bort duplicerade beroenden.
- Ladda upp artefakter till maven-repo.

VARFÖR GRADLE?

- Kan använda befintliga maven-repon för beroenden.
- Kan reducera antalet rader i byggfiler med 75-80%.
- Kan konfigureras med ett riktigt JVM-språk.
- Är designat för att enkelt kunna användas för olika behov.
- Kan uttrycka komplexa beroenden på ett kraftfullt och flexibelt sätt.
- Kan bygga ett stort projekt parallellt på flera kärnor.
- Utvecklas snabbt just nu.
- Är roligt att arbeta med!

AVANCERAD BEROENDE-HANTERING

- Genom gradles API är det enkelt att specialbehandla vissa beroenden.

```
configurations.all {
    resolutionStrategy.eachDependency { dependency ->
        if (dependency.requested.name == 'commons-logging') {
            dependency.useTarget 'org.slf4j:jcl-over-slf4j:1.7.7'
        }
    }
}
```

PROVIDED SCOPE

- Provided scope saknas i gradle, men kan enkelt skapas.

```
configurations {
    provided
}

sourceSets {
    main {
        compileClasspath += configurations.provided
    }
}

dependencies {
    provided "org.apache.geronimo.specs:geronimo-jms_1.1_spec:1.1.1"
}
```

PROFILER

- Stöd för profiler saknas i gradle, men kan enkelt skapas.

```
task loadConfiguration {
    addProjectPropertiesFromFile(file(”configs/${env}/build.properties"))
}

private void addProjectPropertiesFromFile(propfile) {
    def props = new Properties();
    propfile.withInputStream { props.load(it) }
    allprojects { subproject ->
        props.each { key, value ->
            subproject.ext.setProperty(key, value.toString())
        }
    }
}
```

ANT FINNS TILLGÄNGLIGT UTAN SÄRSKILD KONFIGURERING

- Alla ant-targets finns tillgängliga i gradle utan särskild konfigurering.

```
task createVersionPropertyFile(dependsOn: processResources) << {
    def propertyFile = "${buildDir}/resources/main/version.properties"
    ant.touch(file: propertyFile, mkdirs: "true")
    ant.propertyfile(file: propertyFile) {
        entry(key: 'project.version', default: project.version)
    }
}

jar.dependsOn createVersionPropertyFile
```