由 GROOVY 到 GRADLE

qrtt1

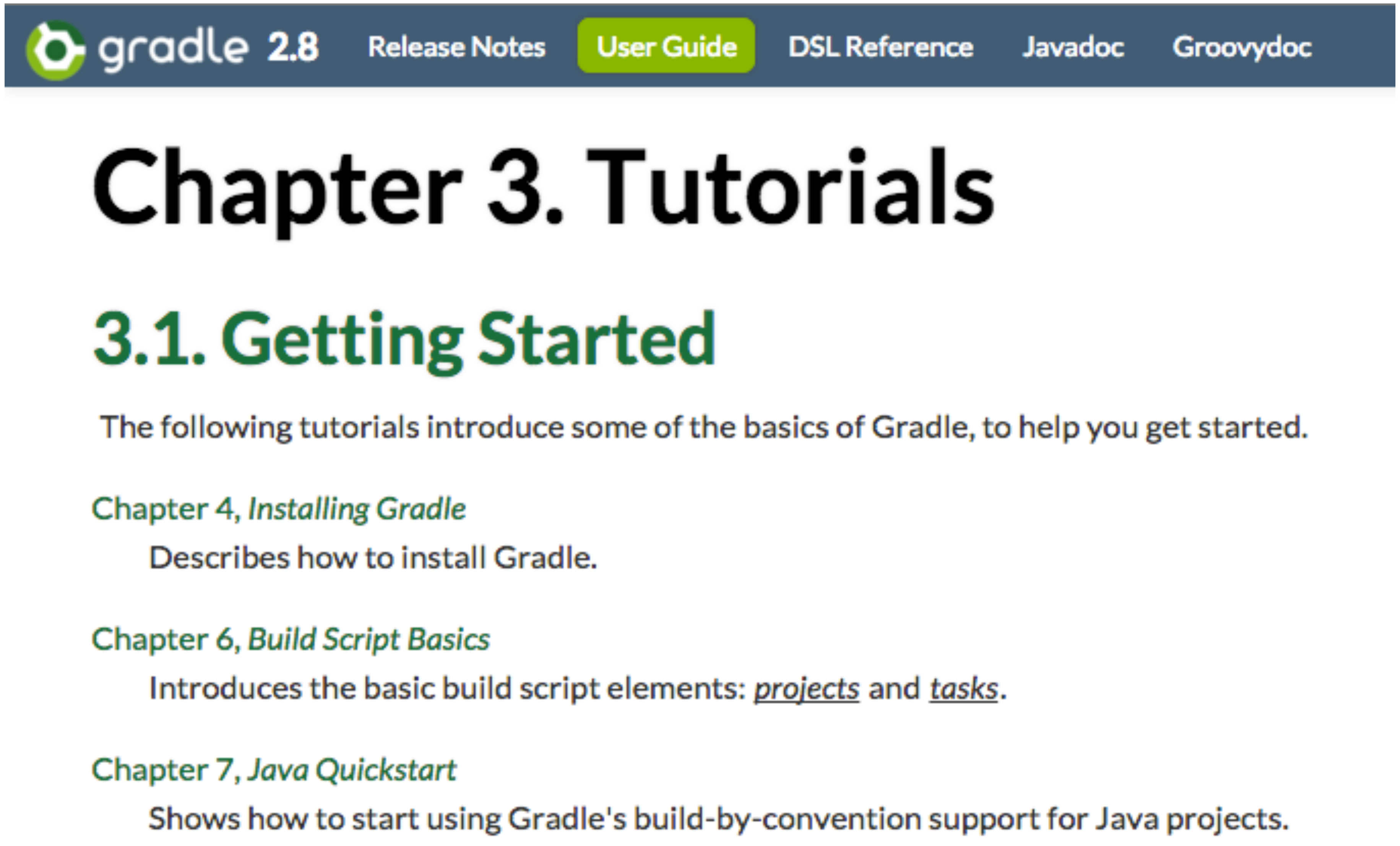
JCConf Taiwan 2015

#JCConf

忙碌的開發者

- 哪有時間看書呢！？看官網應該就夠了唄！
- 換上 Android Studio 它就在了！什麼？你說那東西叫 Gradle ？
- 總是在網路上的茫茫大海找 Gradle 秘技
- Gradle 文件看了，好像懂了，又好像沒懂

試著學一下 Gradle



Chapter 3. Tutorials

3.1. Getting Started

The following tutorials introduce some of the basics of Gradle, to help you get started.

Chapter 4, *Installing Gradle*
Describes how to install Gradle.

Chapter 6, *Build Script Basics*
Introduces the basic build script elements: *projects* and *tasks*.

Chapter 7, *Java Quickstart*
Shows how to start using Gradle's build-by-convention support for Java projects.

簡單來說

- 知道怎麼安裝（Android Studio User：竟然要安裝！）
- 知道怎麼寫 task
- 知道怎麼編譯 java 專案
- 懂得相依管理
- (那 Android Developer 咧！？)
- (那 Android Developer 咧！？)
- (那 Android Developer 咧！？)

CH6.1 基本 Script 教學

- Gradle Script 由二個基本概念構成
  - project：含 1 個或多個 project
  - task：每個 project 可以有 0 個或多個 task

CH6.2 Hello World

用 task 關鍵字，定義新的 task

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

CH6.2 Hello World

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

理解困難 >”<

再放一組 {} 再寫個 doLast 再一個 {} …..

CH6.3 Hello World

同樣用 task 關鍵字 ...

再加上 <<

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

阿鬼！你還是說中文吧！？

CH6.4 Script are code

然後咧！？

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

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

CH6.5 Task Dependencies

獲得新技能，設定 task 相依關係

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

CH6.6 Dynamic Task

《重新認識你的 int》 Groovy 萬物皆物件

4. repeat method 與 groovy closure

```
times { counter ->

    task "task$counter" << {

        println "I'm task number $counter"

    }   能教點實用的東西嗎！？

}
```

CH6.7 Dynamic Task

```
task hello << {
    println 'Hello Earth'
}
hello.doFirst {
    println 'Hello
}
hello.doLast {
    println 'Hello Mars'
}
hello << {
    println 'Hello Jupiter'
}
```

初學者們！
還撐得下去嗎？

CH6.9 Extra Task properties

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

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

CH6.10 Using Ant Task

現在

字太小！就是不重要

```
task loadfile << {
    def files = file('../antLoadfileResources').listFiles().sort()
    files.each { File file ->
        if (file.isFile()) {
            ant.loadfile(srcFile: file, property: file.name)
            println " *** $file.name ***"
            println "${ant.properties[file.name]}"
        }
    }
}
```

CH6.11 Using Methods

```
task checksum << {
    fileList('../antLoadfileResources').each {File file ->
        ant.checksum(file: file, property: "cs_$file.name")
        println "$file.name Checksum: ${ant.properties["cs_$file.name"]}"
    }
}

task loadfile << {
    fileList('../antLoadfileResources').each {File file ->
        ant.loadfile(srcFile: file, property: file.name)
        println "I'm fond of $fi
    }
}
```

要寫 method 也行滴！

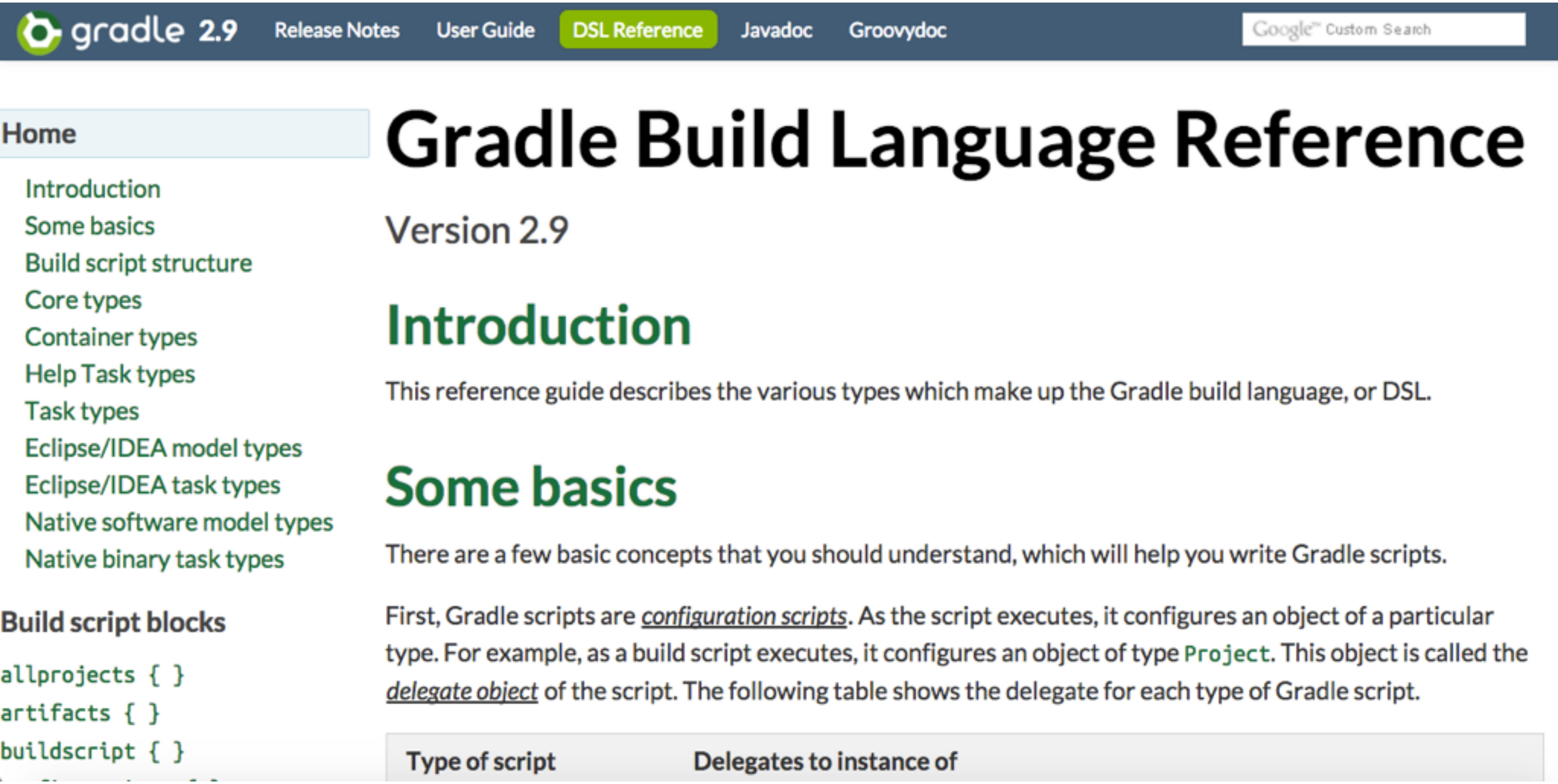
```
File[] fileList(String dir) {
    file(dir).listFiles({
        file -> file.isFile()
    }as FileFilter).sort()
}
```

CH6.13 Configure by DAG

```
gradle.taskGraph.whenReady {taskGraph ->
    if (taskGraph.hasTask(release)) {
        version = '1.0'
    } else {
        version = '1.0-SNAPSHOT'
    }
}
```

你會 Gradle 了！

還有 Gradle DSL



Home

Build script blocks

```
allprojects { }
artifacts { }
buildscript { }
```

Gradle Build Language Reference

Version 2.9

Introduction

This reference guide describes the various types which make up the Gradle build language, or DSL.

Some basics

There are a few basic concepts that you should understand, which will help you write Gradle scripts.

First, Gradle scripts are *configuration scripts*. As the script executes, it configures an object of a particular type. For example, as a build script executes, it configures an object of type `Project`. This object is called the *delegate object* of the script. The following table shows the delegate for each type of Gradle script.

Type of script	Delegates to instance of

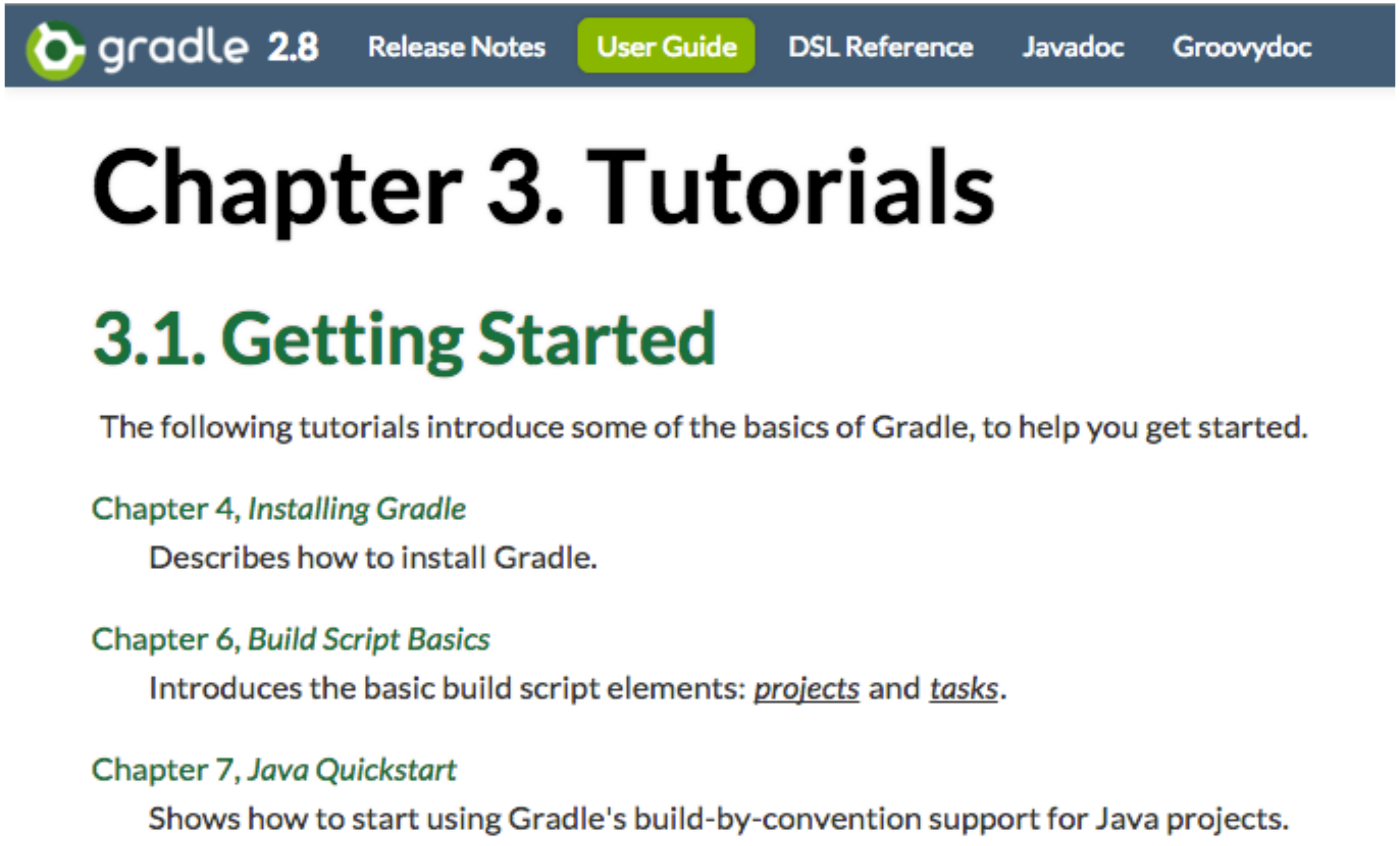
要繼續下一關嗎？

GOTO CH7

狀態顯示：隊友已陣亡

我不想懂，它能動就好

不要在官網上學 Gradle



Chapter 3. Tutorials

3.1. Getting Started

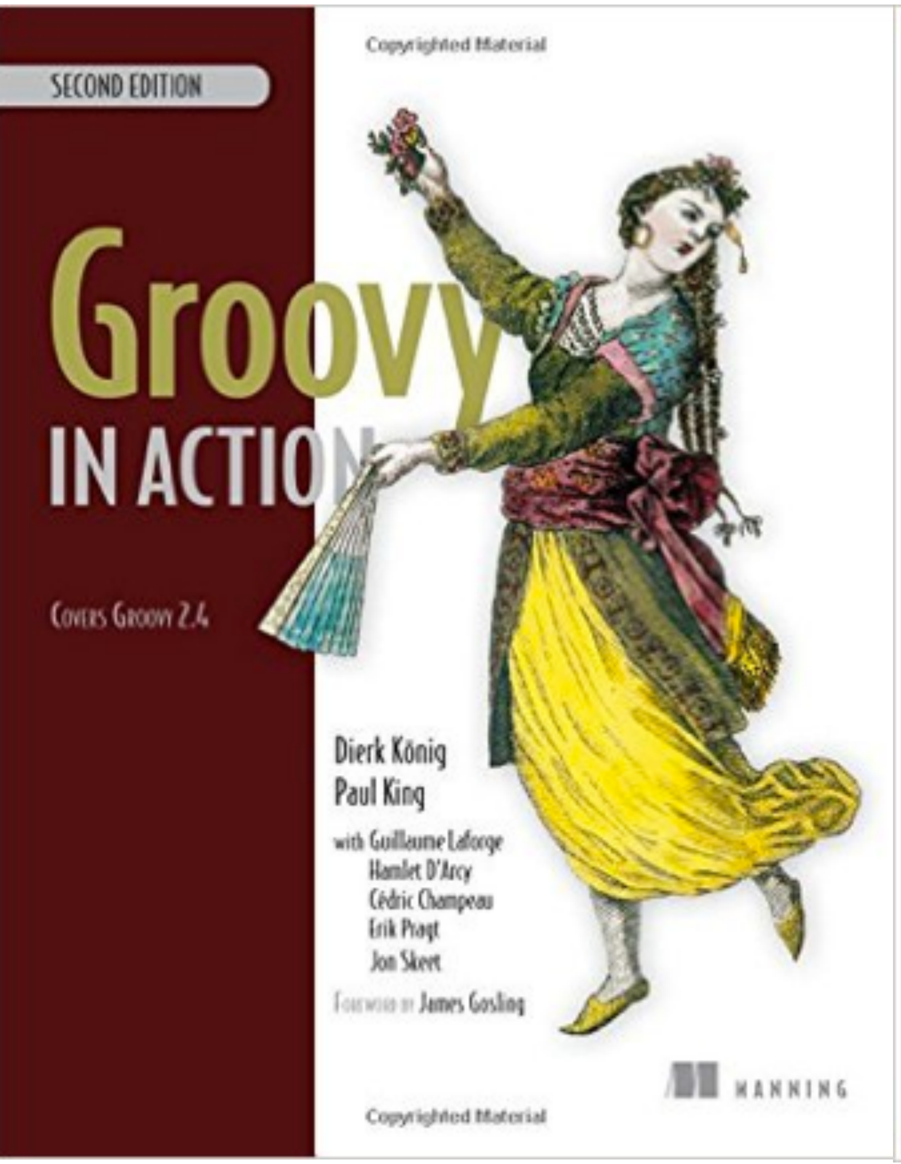
The following tutorials introduce some of the basics of Gradle, to help you get started.

Chapter 4, *Installing Gradle*
Describes how to install Gradle.

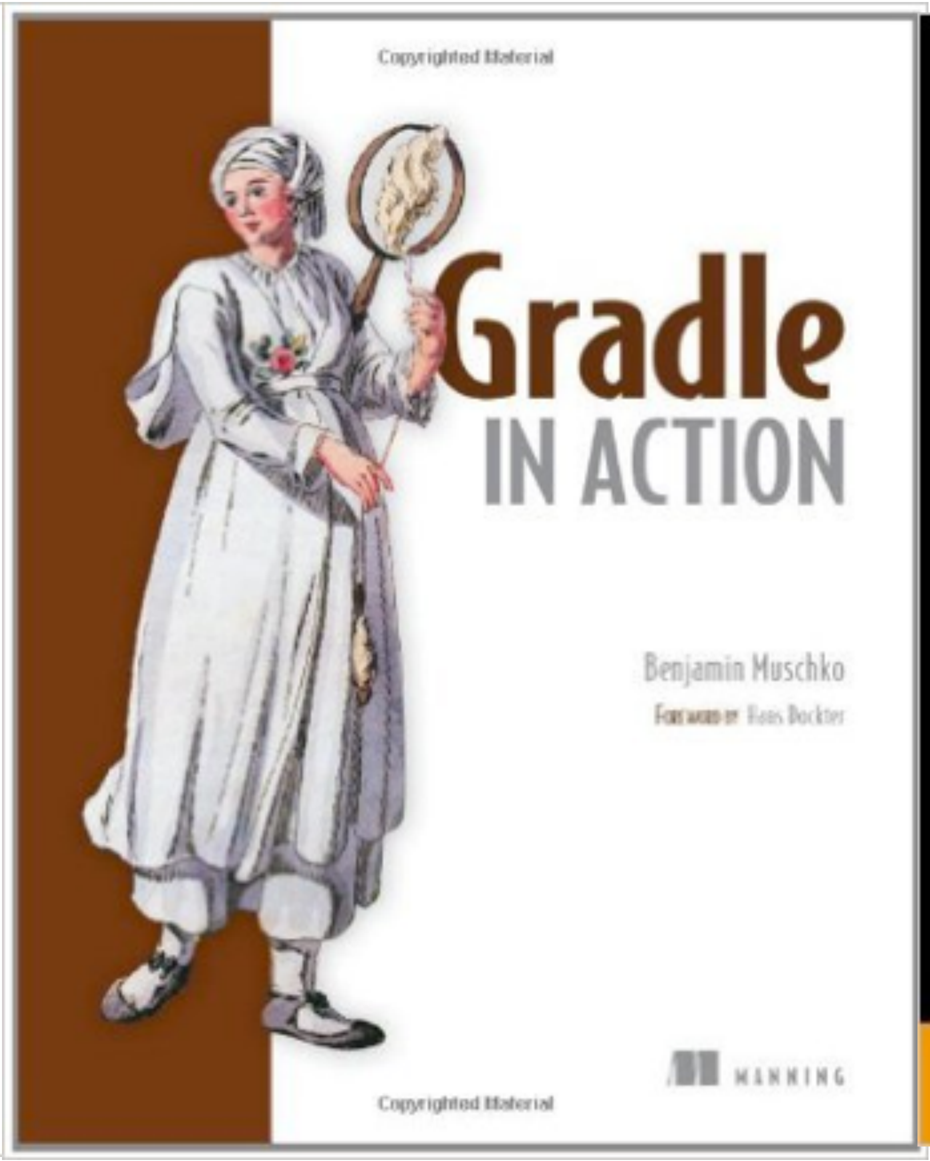
Chapter 6, *Build Script Basics*
Introduces the basic build script elements: *projects* and *tasks*.

Chapter 7, *Java Quickstart*
Shows how to start using Gradle's build-by-convention support for Java projects.

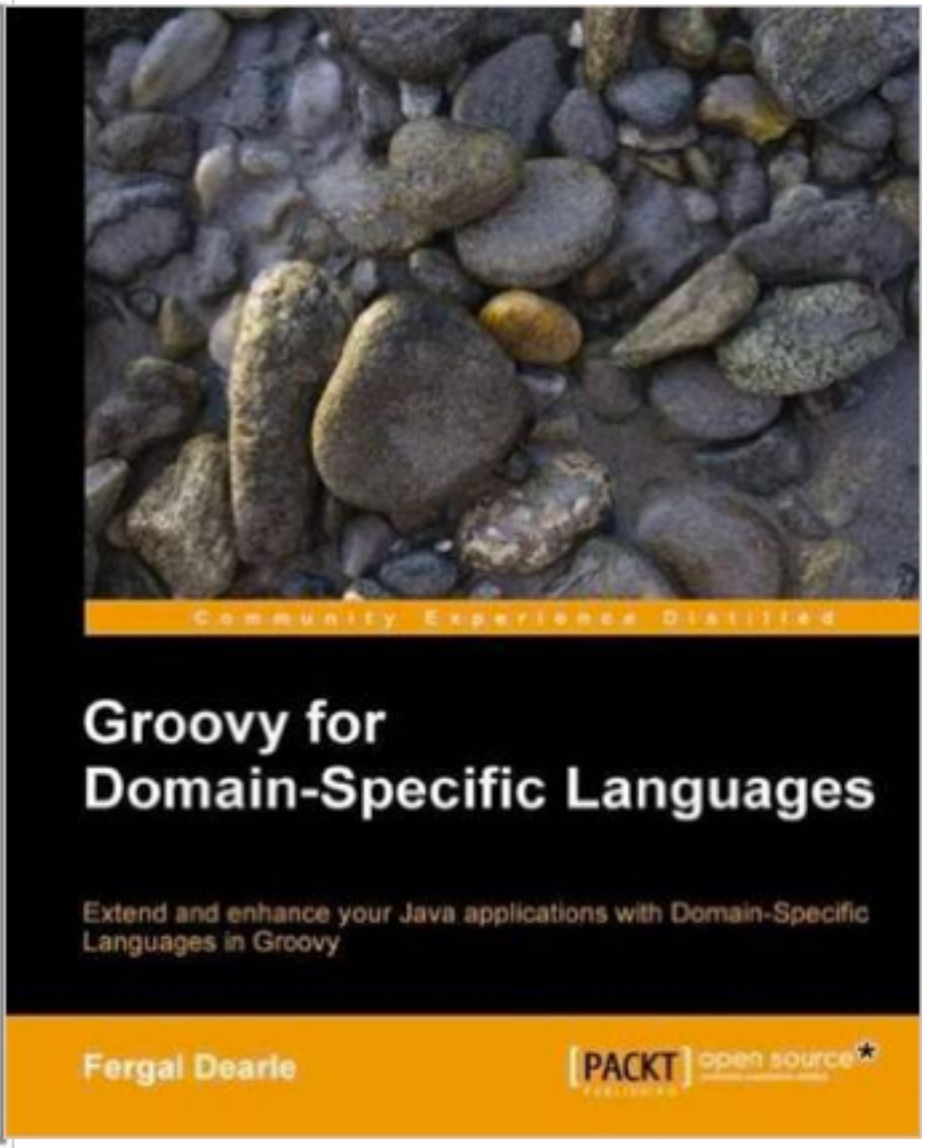
如果有時間…



Copyrighted Material
SECOND EDITION
Groovy
IN ACTION
Covers Groovy 2.4
Dierk König
Paul King
with Guillaume Laforge
Hamlet D'Arcy
Cédric Champeau
Erik Pragt
Jon Skeet
Foreword by James Gosling
MANNING
Copyrighted Material



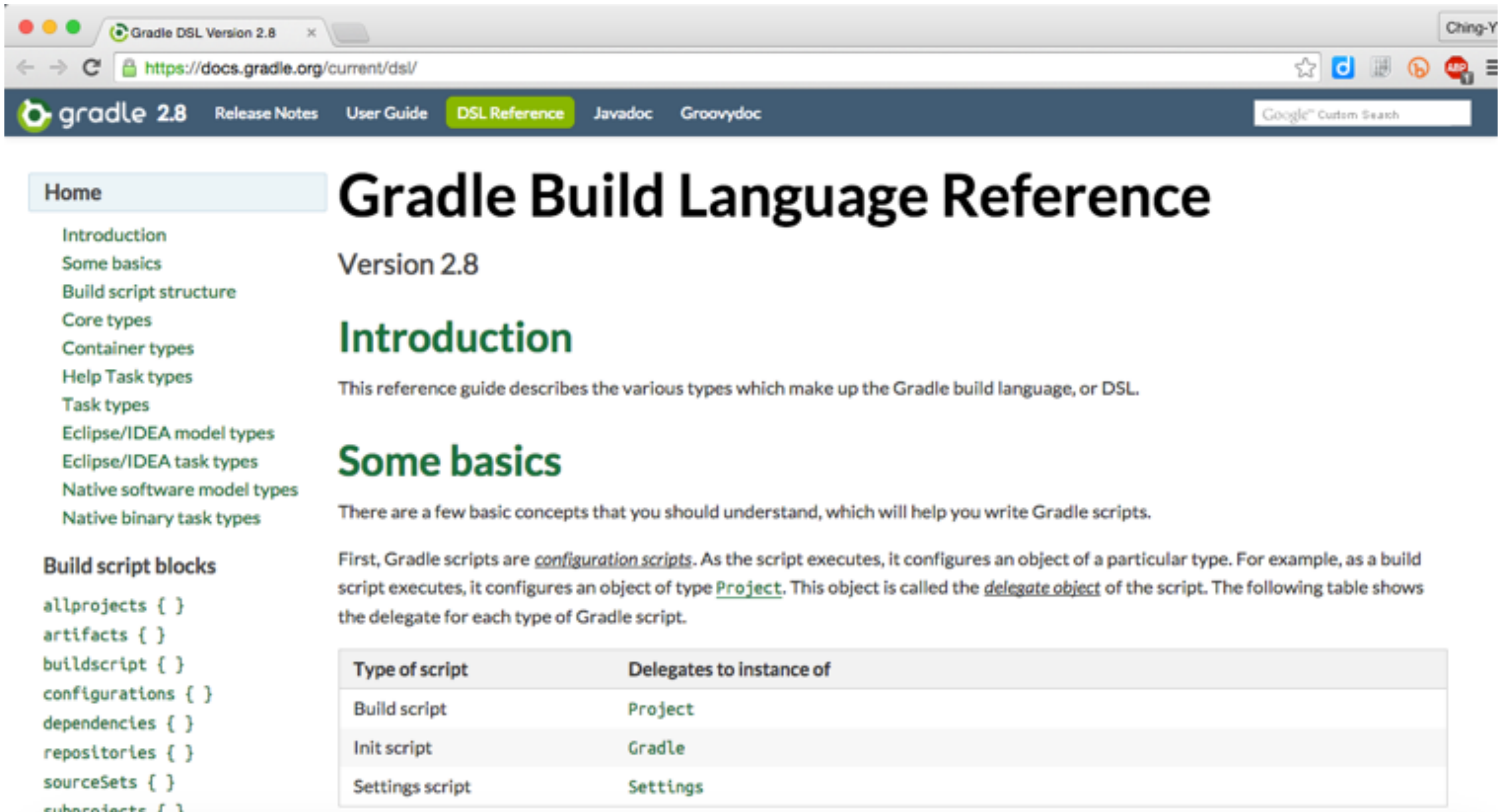
Copyrighted Material
Gradle
IN ACTION
Benjamin Muschko
Foreword by Hans Dockter
MANNING
Copyrighted Material



Community Experience Distilled
Groovy for
Domain-Specific Languages
Extend and enhance your Java applications with Domain-Specific Languages in Groovy
Fergal Dearle
PACKT open source

讀後心得

- Groovy 真是個兼容各家懶人語法的語言
- 理解 Gradle DSL 可以先由 Groovy Feature 開始


Gradle DSL Version 2.8

https://docs.gradle.org/current/dsl/

gradle 2.8 Release Notes User Guide DSL Reference Javadoc Groovydoc

Home

Introduction
Some basics
Build script structure
Core types
Container types
Help Task types
Task types
Eclipse/IDEA model types
Eclipse/IDEA task types
Native software model types
Native binary task types

Build script blocks

```
allprojects { }
artifacts { }
buildscript { }
configurations { }
dependencies { }
repositories { }
sourceSets { }
```

Gradle Build Language Reference

Version 2.8

Introduction

This reference guide describes the various types which make up the Gradle build language, or DSL.

Some basics

There are a few basic concepts that you should understand, which will help you write Gradle scripts.

First, Gradle scripts are *configuration scripts*. As the script executes, it configures an object of a particular type. For example, as a build script executes, it configures an object of type `Project`. This object is called the *delegate object* of the script. The following table shows the delegate for each type of Gradle script.

Type of script	Delegates to instance of
Build script	`Project`
Init script	`Gradle`
Settings script	`Settings`

基本認知

- Gradle Script 就是 Groovy Script
- Gradle Script 即為 Groovy DSL
- 蝦毀！要學新的語言！？
  - Groovy 相容於『大部分』Java 語法
  - Groovy 能省略 () 與 ;
  - 支援 Closure (Code Block)

In Gradle 新

Groovy 常用語法

- Bean 自動產生 setter/getter
- 常用容器 List 與 Map 直接在語法上支援
- 具有 Closure 的設計，能取代匿名類別的常用情景
- 配合 () 省略，讓 Closure 寫起來像 method body

In Gradle 新

Groovy 常用語法

optional typing 懶得寫 type
就用 def (其實就是 Object)

new HashMap()

```
def map = [:]
```

看到 [] 就是容器，
看到 : 就是 Map

```
def colors = [
    red: '#FF0000',
    green: '#00FF00',
    blue: '#0000FF']
```

http://www.groovy-lang.org/syntax.html

In Gradle

Groovy 常用語法 新

new ArrayList()

```
def numbers = [1, 2, 3]
```

看到 [] 就是容器，看到，又沒寫 type 就是 List

```
String[] arrStr =
        ['Ananas', 'Banana', 'Kiwi']
```

http://www.groovy-lang.org/syntax.html

In Gradle

新

Groovy 常用語法

{} 放在參數列或被當成變數內容，就是 Closure 物件

```
["hello", "world"].each({ println it })
```

it 是預設的參數名稱

```
["hello", “world"].each({
   elem -> println elem
})
```

可以變更參數名稱，用 -> 隔開就行了

http://www.groovy-lang.org/syntax.html

In Gradle 新

Groovy 常用語法

語法省略 () 的效果，讓 Closure 看起來像 Method Body

```
file("build.gradle").withReader { reader ->
    reader.eachWithIndex { it, line ->
        println "${line+1} $it"
    }
}
```

所以，實作 DSL 時，常把 Closure 參數放在最後 1 個

In Gradle

新

Groovy 常用語法

這是 Map

```
apply plugin: 'java'

sourceSets {
  main {
    java {
      exclude 'some/unwanted/package/**'
    }
  }
}
```

sourceSets 調整是 Closure & Method Invoke

《欣賞一下 Groovy Code》

```
001_bean.groovy
002_collection.groovy
```

Groovy DSL Features

- Closure 支援 delegate 機制

{} closure 將實作 delegate 給 Copy

```
task copyDocs(type: Copy) {
    from 'src/main/doc'
    into 'build/target/doc'
}
```

https://docs.gradle.org/current/javadoc/org/gradle/api/tasks/Copy.html

《欣賞一下 Groovy Code》

`003_closure.groovy`

Groovy DSL Features

- Compiler 提供可客製化的 AST Transformations

```
task helloworld << {
    println 'Hello World'
}
```

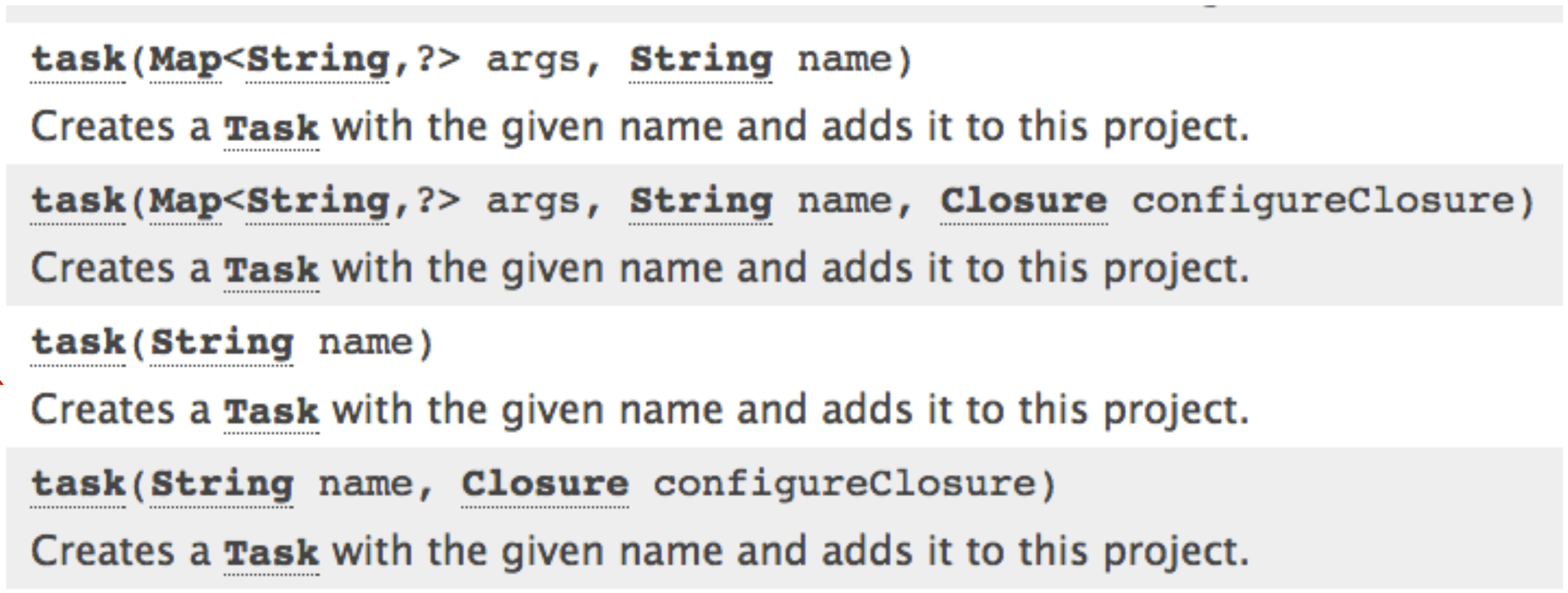
- 對 gradle 來說
  - task 是 keyword 用來宣告新的 task
  - helloworld 是 task 名稱
- 對 groovy 來說
  - task 是個 method invoke (呼叫 Script 的 BaseClass)
  - helloworld 是 method 的參數，也是未定義的變數

Groovy DSL Features

- AST Transformations：處理 task method invoke

```
task helloworld << {
    println 'Hello World'
}
```

透過 AST Transformation 轉成 task("helloworld")



task(Map<String,?> args, String name)
Creates a Task with the given name and adds it to this project.

task(Map<String,?> args, String name, Closure configureClosure)
Creates a Task with the given name and adds it to this project.

task(String name)
Creates a Task with the given name and adds it to this project.

task(String name, Closure configureClosure)
Creates a Task with the given name and adds it to this project.

https://github.com/gradle/gradle/blob/master/subprojects/core/src/main/groovy/org/gradle/groovy/scripts/internal/TaskDefinitionScriptTransformer.java

MetaObjectProtocol

Groovy DSL Features

- 支援 Meta Programming (透過 MOP 的 method hooking 機制)。讓你在 build script 可以 access 到 plugin 新增的 method 或是 properties

有想過為什麼這麼寫能動嗎？

```
apply plugin: 'java'

sourceCompatibility = 1.8
targetCompatibility = 1.8
manifest {}
sourceSets {}
```

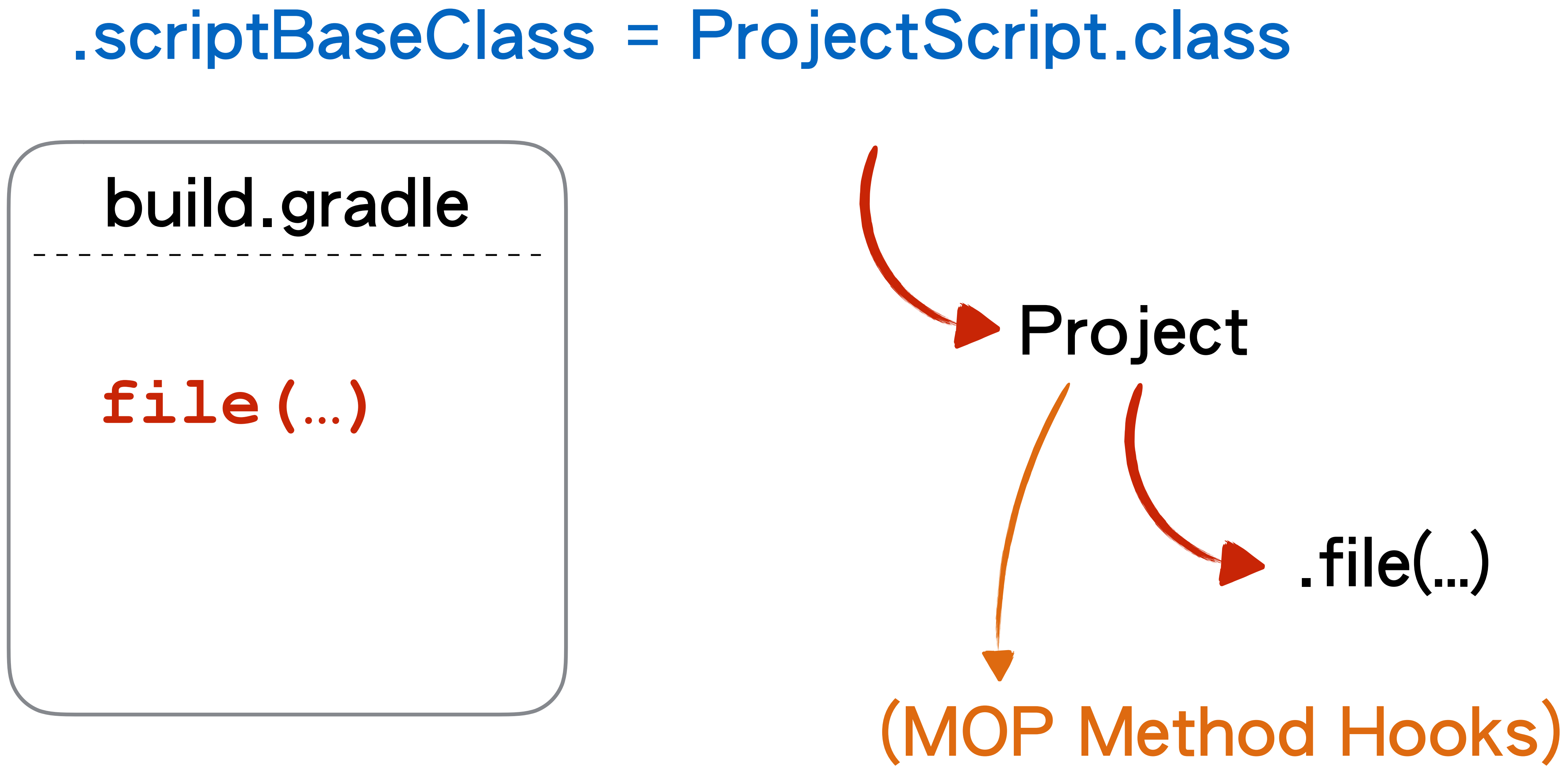
https://docs.gradle.org/current/javadoc/org/gradle/api/Project.html#property(java.lang.String)

《欣賞一下 Groovy Code》

`004_mop.groovy`

Groovy DSL Features

- Compiler 支援 DSL 設定 scriptBaseClass



目前為止的新知

- Gradle DSL 出現的 {} 大部分都是 Closure
- Closure 可透過指定 delegate 來委派實作
- 看到 [] 就想到容器，看到 [] 內有：就是個 Map 容器，單獨看到：也要想到是個 Map

目前為止的新知

- build script 的 scriptBaseClass 是 ProjectScript
- ProjectScript 透過 MOP 委派工作給 Project 物件
- Project 提供常用 method 並透過 MOP 委派 method invoke 或 properties access 給其它物件
- task 關鍵字會被轉換為 task method，而 task name 轉為字串傳入

讓我們再重來一次

GOTO CH6

CH6.2 Hello World

用 task 關鍵字，定義新的 task

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

CH6.2 Hello World

用

```
task hello {

    doLast {

    }

}
```

task 關鍵字，會被 AST 轉為 task method。

hello 會被轉為字串，作為 task method 的參數

CH6.2 Hello World

用

```
task hello {
```

依據 build script 的 baseClass 最終將工作委派給 Project 物件，預期能在它上面找到相關 method

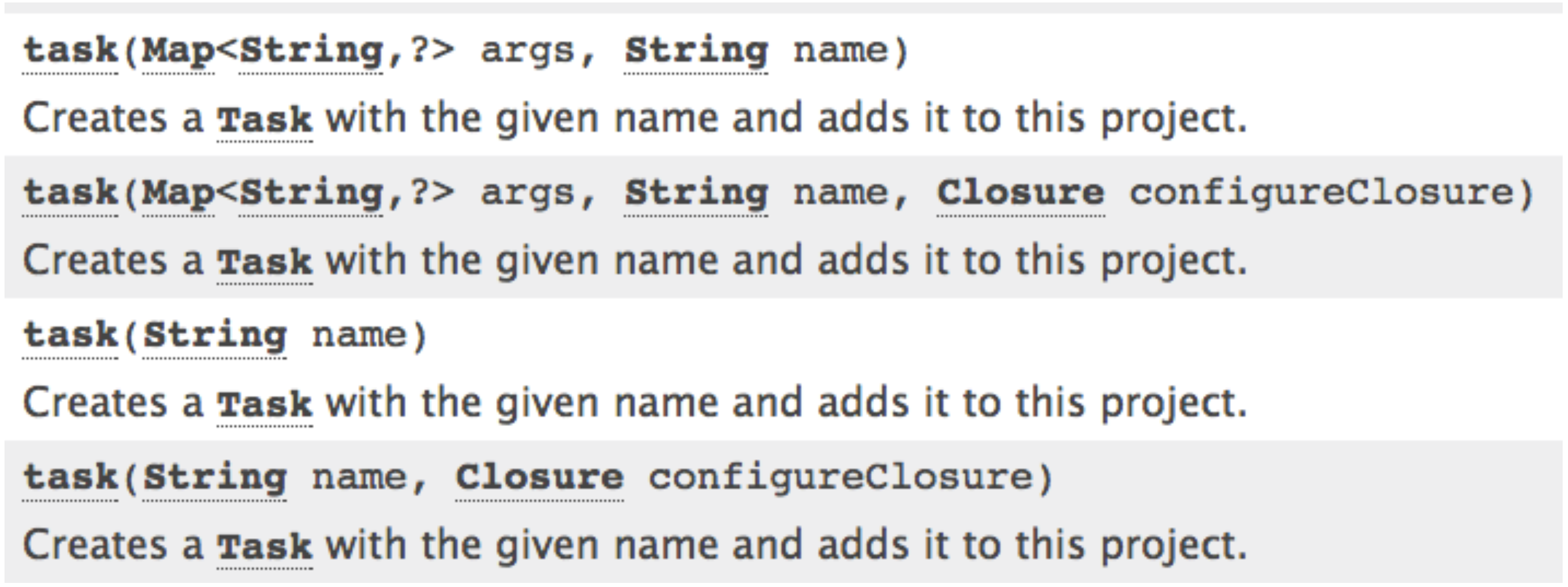
```
        }

}
```

CH6.2 Hello World

用

```
task hello {
```

它應該對應到下面哪一個 task method 呢？



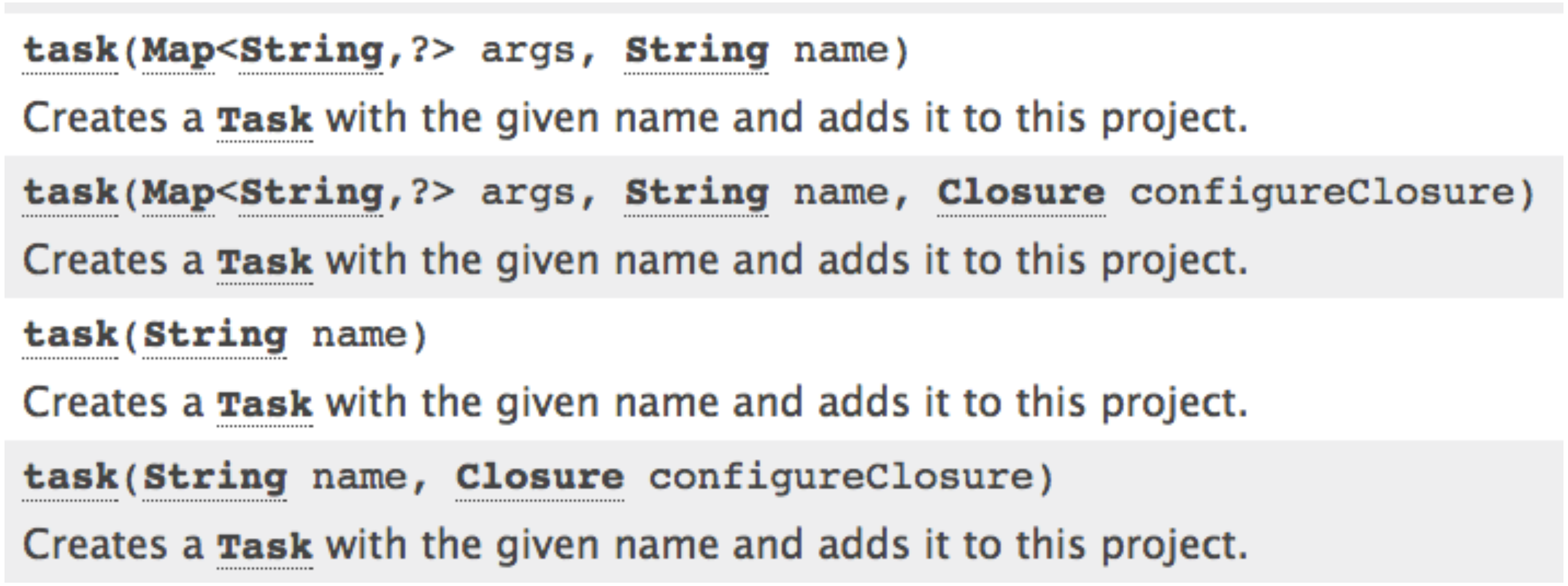
`task(Map<String,?> args, String name)`	Creates a `Task` with the given name and adds it to this project.
`task(Map<String,?> args, String name, Closure configureClosure)`	Creates a `Task` with the given name and adds it to this project.
`task(String name)`	Creates a `Task` with the given name and adds it to this project.
`task(String name, Closure configureClosure)`	Creates a `Task` with the given name and adds it to this project.

CH6.2 Hello World

用

```
task hello {
```

前 2 組有 **Map<String, ?>**，但在 Script 沒出現：



`task(Map<String,?> args, String name)`	Creates a **Task** with the given name and adds it to this project.
`task(Map<String,?> args, String name, Closure configureClosure)`	Creates a **Task** with the given name and adds it to this project.
`task(String name)`	Creates a **Task** with the given name and adds it to this project.
`task(String name, Closure configureClosure)`	Creates a **Task** with the given name and adds it to this project.

CH6.2 Hello World

用

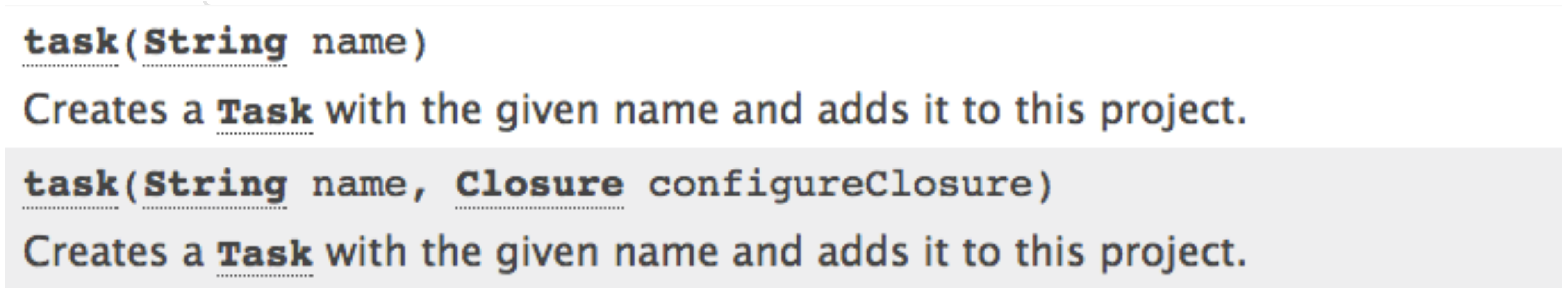
```
task hello {
```

第 3 組只有唯一的 name 參數，

而第 4 組有 name, closure。

rld!'

符合看到 {} 幾乎是 closure



`task(String name)`

Creates a `Task` with the given name and adds it to this project.

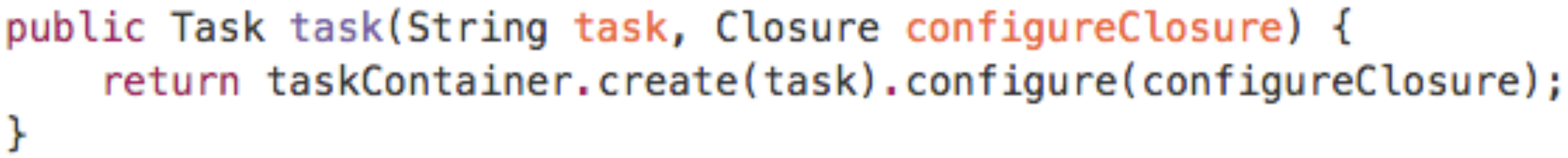
`task(String name, Closure configureClosure)`

Creates a `Task` with the given name and adds it to this project.

CH6.2 Hello World

接著，我們來搞定這組 closure

```
task hello {
```



```
public Task task(String task, Closure configureClosure) {
    return taskContainer.create(task).configure(configureClosure);
}
```

project 建 1 個 task 物件後，呼叫 configure 方法

```
}
```

CH6.2 Hello World

這是 configure closure

doLast 是誰家的 method 呢？
回想一下 closure delegate

```
task hello {

    doLast {

        println 'Hello world!'

    }

}
```

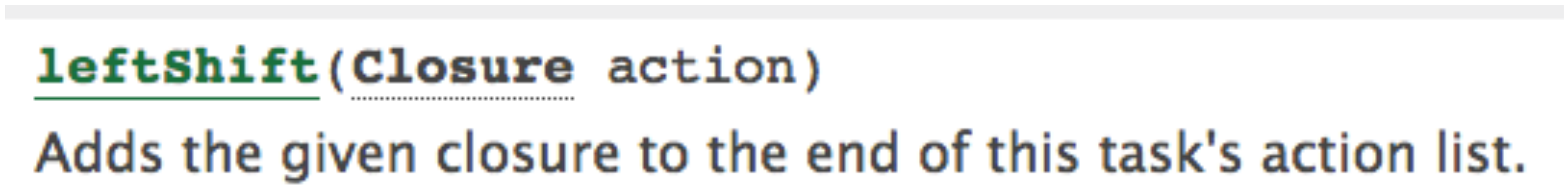
《用 gradle 做個小實驗》

CH6.3 Hello World

同樣用 task 關鍵字 ...

再加上 <<

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



leftShift(Closure action)

Adds the given closure to the end of this task's action list.

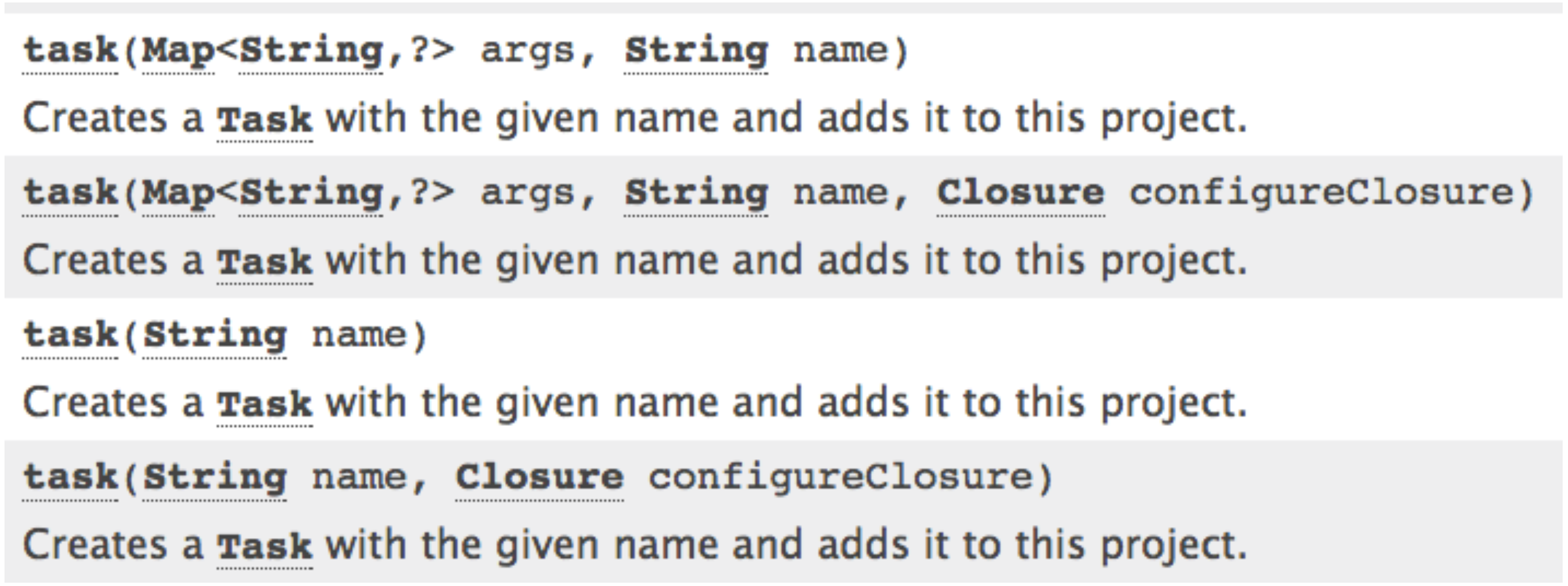
《繼續看 gradle 實作》

CH6.5 Task Dependencies

這是 Map

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

它應該對應到下面哪一個 task method 呢？



`task(Map<String,?> args, String name)`

Creates a Task with the given name and adds it to this project.

`task(Map<String,?> args, String name, Closure configureClosure)`

Creates a Task with the given name and adds it to this project.

`task(String name)`

Creates a Task with the given name and adds it to this project.

`task(String name, Closure configureClosure)`

Creates a Task with the given name and adds it to this project.

重塑 Gradle 的世界觀

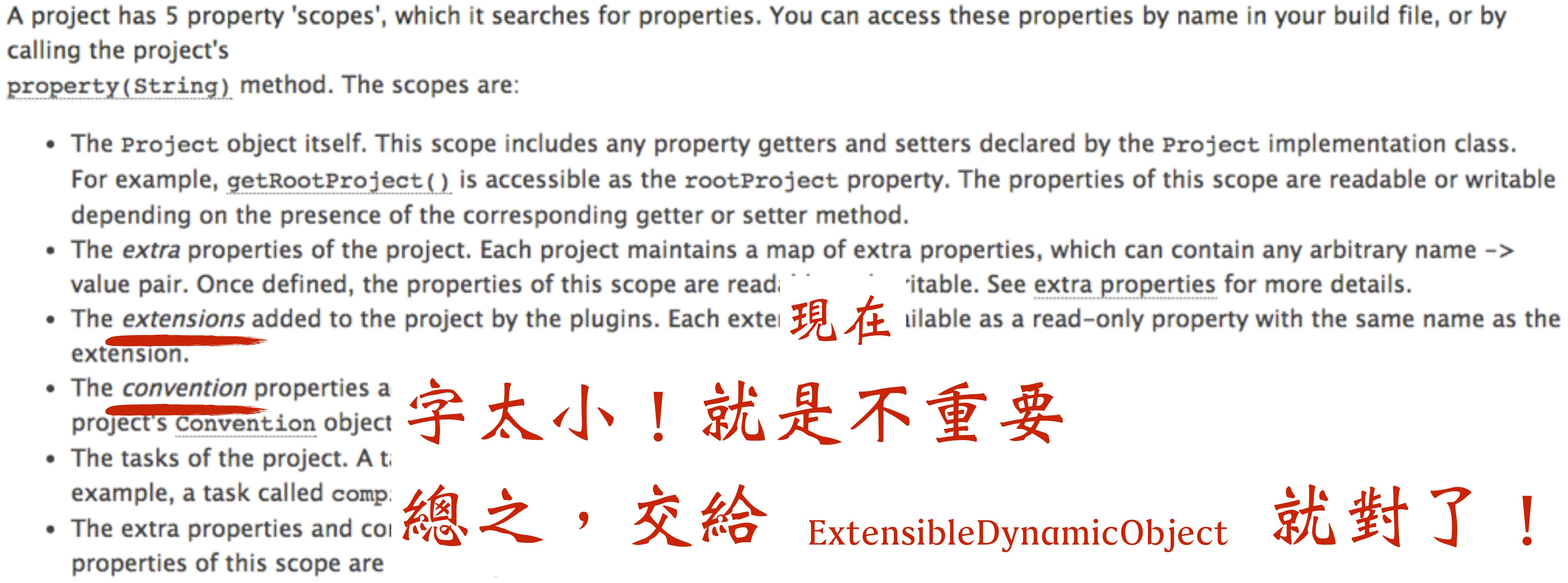
- build script 透過 baseClass 委派 ProjectScript
  - ProjectScript 透過 MOP 委派 Project
    - Project 透過 MOP 委派給「中介物件」
      - ExtensibleDynamicObject

Project 物件中的 property 或 method 的 resolver

《Property Scope：How gradle resolve property》

ExtensibleDynamicObject

Project 的 javadoc 有寫：



A project has 5 property 'scopes', which it searches for properties. You can access these properties by name in your build file, or by calling the project's
property(String) method. The scopes are:

- The Project object itself. This scope includes any property getters and setters declared by the Project implementation class. For example, getRootProject() is accessible as the rootProject property. The properties of this scope are readable or writable depending on the presence of the corresponding getter or setter method.
- The *extra* properties of the project. Each project maintains a map of extra properties, which can contain any arbitrary name -> value pair. Once defined, the properties of this scope are read ritable. See extra properties for more details.
- The *extensions* added to the project by the plugins. Each exte 現在 ilable as a read-only property with the same name as the extension.
- The *convention* properties a
project's Convention object
- The tasks of the project. A t
example, a task called comp
- The extra properties and co
properties of this scope are

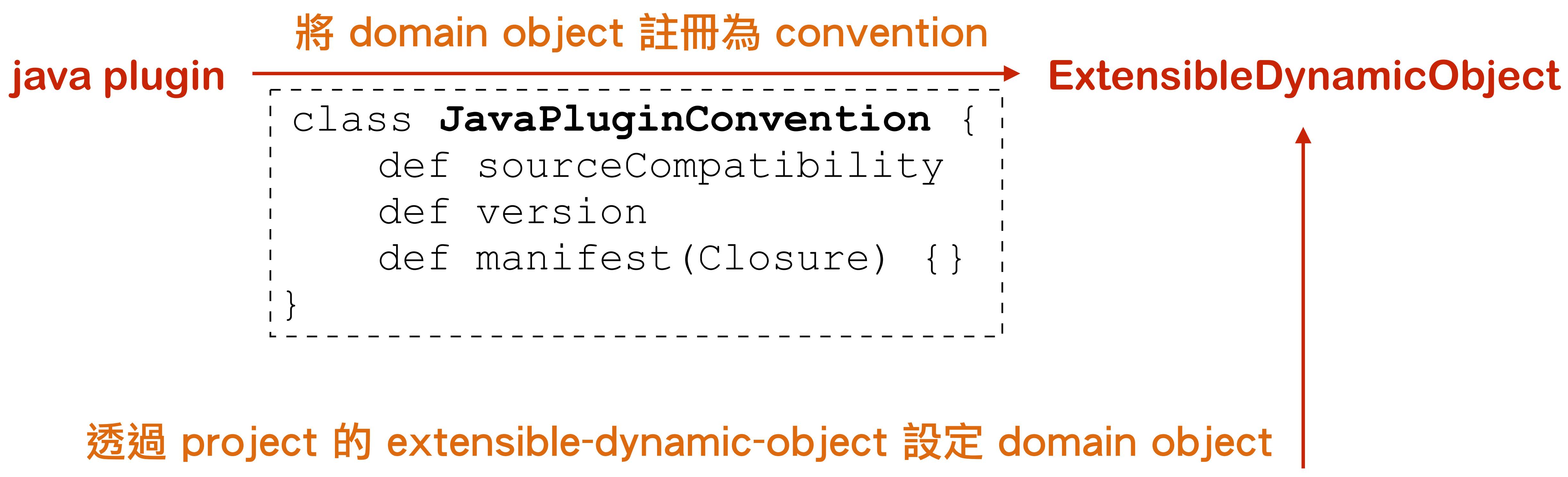
字太小！就是不重要

總之，交給 ExtensibleDynamicObject 就對了！

https://github.com/gradle/gradle/blob/master/subprojects/core/src/main/groovy/org/gradle/api/internal/ExtensibleDynamicObject.java

Chapter 7. Java Quickstart

CH7.2.3 Customizing the project


將 domain object 註冊為 convention
java plugin
ExtensibleDynamicObject
透過 project 的 extensible-dynamic-object 設定 domain object

```
class JavaPluginConvention {
    def sourceCompatibility
    def version
    def manifest(Closure) {}
}
```

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

你會 Gradle 了！

由 groovy 到 gradle

- 透過理解 groovy 語法與 DSL feature 培養另一種看待 gradle 的「審美觀」（視角）
- 透過閱讀 gradle 程式碼取得比「文件」更直接的訊息，而理解 gradle 的運作方式
- gradle 內還有許多精巧的設計，是 groovy DSL 之外的部分需深入研究，但只要把握著它最終會透過 groovy DSL 實現，就無需有太多的憂慮

Q&A