技術筆記

筆記更新紀錄

|  |  |  |
| --- | --- | --- |
| 日期 yyyy/MM/dd | 更新人員 | 更新備註 |
| 2019/03/21 | Miles | 建立文件，添加技術目錄Apache Ant、Apache Maven、Gradle、Simple Factory…等等 |
| 2019/03/22 | Miles | 更新Gradle內容 |
| 2019/07/21 | Miles | 更新 設計模式- Simple Factory Pattern 內容 |
| 2019/07/22 | Miles | 更新 設計模式- Factory Method Pattern 內容 |
| 2019/07/23 | Miles | 更新 設計模式- Abstract Factory Pattern 內容 |
| 2019/07/23 | Miles | 更新 設計模式- SOLID 內容 |
|  |  |  |

目錄

[1建構工具 Apache Ant 4](#_Toc14768196)

[2建構工具 Apache Maven 5](#_Toc14768197)

[3建構工具 Gradle 7](#_Toc14768198)

[4設計模式- SOLID 14](#_Toc14768199)

[5設計模式- Simple Factory Pattern 15](#_Toc14768200)

[6設計模式- Factory Method Pattern 18](#_Toc14768201)

[7設計模式- Abstract Factory Pattern 22](#_Toc14768202)

[8設計模式- Builder Pattern 26](#_Toc14768203)

[8設計模式- Reflection Factory Pattern 26](#_Toc14768204)

[9設計模式- Singleton 26](#_Toc14768205)

[6技術框架- Hibernate 29](#_Toc14768206)

[8技術框架- NodeJS 30](#_Toc14768207)

[9自動化打包工具- Jenkins 31](#_Toc14768208)

[11 Server- Oracle WebLogic Server 33](#_Toc14768209)

[12 Server- WildFly JBoss 34](#_Toc14768210)

[13 Server- Apache Tomcat 35](#_Toc14768211)

[14 HTTP概念 36](#_Toc14768212)

[14 UML概念 36](#_Toc14768213)

## 1建構工具 Apache Ant

|  |
| --- |
| Apache Ant |
| 簡介 |
| 早期使用XML撰寫的建構技術 |
| 優缺點 |
|  |
| 基本結構 |
|  |

## 2建構工具 Apache Maven

|  |
| --- |
| Apache Maven |
| 簡介 |
| 自動化編譯工具、Java專案的標準化、Java專案函示庫的統一 |
| 優缺點 |
|  |
| pom.xml基本模板 |
| <?xml version="1.0" encoding="UTF-8"?>  <!-- pom的模板文件，包含內容節點的定義 -->  <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <!-- 模板版本 -->  <modelVersion>4.0.0</modelVersion>  ...  ...  ...  </project> |
| 註解 |
| 所有的pom文件需要<project>元素與三個子元素:groupId、artifactId、version。 |
| 子元素訊息 |
| 1. <groupId></groupId>:組織標識。 2. <artifactId></artifactId>:項目名稱(專案名稱) 。 3. <version></version>:版本號碼。 4. <packaging></packaging>:打包的格式，可以為:pom、jar、maven-plugin、ejb、war、ear、rar、par。 5. <name></name>:項目名稱，maven產生文件檔使用。 6. <url></url>:項目主頁的URL，maven產生文件檔使用。 7. <description></description>: |
| 基本目錄結構 |
| MavenProject/  ├── pom.xml  └── src      ├── main      │   └── java      │       └── tw      │           └── com      │               └── codedata      │                   └── App.java      └── test          └── java              └── tw                  └── com                      └── codedata                          └── AppTest.java    11 directories, 3 files |
| 註解 |
| 1.平行的目錄結構 src/main/java 與 src/test/java 下的 package 即為 groupdId。  2.以上使用maven-archetype-quickstart樣板產生 |
| 其他 |
| 1. super pom:父pom |
| 建構生命週期 |
| 什麼是建構生命週期，建構的階段，其中每一個階段都是生命周期的一部分。  以下舉例基本的序列:   |  |  |  | | --- | --- | --- | | 階段 | 處理 | 描述 | | prepare-resource | 拷貝資源 | 本階段可以定義需要拷貝的資源。 | | compile | 編譯 | 本階段完成程式碼編譯。 | | pagkage | 打包 | 本階段根據pom.xml所描述的配置創建jar/war檔。 | | install | 安裝 | 本階段在本地/遠程位置中安裝打包檔案。 | |

## 3建構工具 Gradle

|  |
| --- |
| Gradle |
| 簡介 |
| 1. Gradle是一個參考Apache Ant與Apache Maven概念的專案自動化建構工具。 2. Gradle使用Groovy語言來聲明設定，而不是使用XML。 3. 支援Java、Groovy、Scala等。 4. Gradle與Ant有很緊密的結合，甚至在建構時可以把Ant建構指令碼直接匯入。 |
| 特點 |
| 1. 自動處理套件相依關係 – 取得Maven Repos的概念。 2. 自動處理部屬問題 – 取自Ant的概念。 3. 條件判斷寫法直覺 –使用Groovy語言。 4. 容易辨識與直觀。 |
|  |
|  |
| 建構基本的Gradle Java Project(範例) |
| 1. 建立一個資料夾(Project)      1. 建立 build.gradle 檔案，內容為      1. 建立一個java檔案(測試用) \src\main\java\com\example\Main.java   4   1. 進行編譯 語法為 gradle build      1. 測試 |
| 建構基本的Gradle Java Project(說明) |
| 1. 基本上Gradle Project目錄結構與Maven相似，所以可以參考Maven目錄結構 2. 步驟4解釋當中，建立流程如同Ant的task流程，每一個都是一個任務。 |
| Gradle 基本指令操作 |
| 1. 如果不知道要下哪一個task，可以直接下tasks。     執行後可以看到許多的 task，它們以特定的群組為單位集合在一起(以下僅有部分) |
| 其他語法參考 |
| 如果有在學到或者碰到會再補上   |  |  | | --- | --- | | 語法 | 說明 | | gradle | 主要是help與歡迎Gradle。 | | gradle tasks | 提示可以使用那些task。 | | gradle build | 編譯gradle專案，並且產生一個jar檔案(預設)，jar檔位置在根目錄/build/…。 | | gradle build run | 編譯gradle專案，並且run該專案的main的入口。 | | gradle eclipse | 產生eclipse專案。 | | gradle clean | 清理專案。 | |  |  | |  |  | |
| build.gradle基本結構(範例) |
| 使用gradle結構  // 引用java plugin 獲得編譯 java 專案相關的 task  apply plugin: 'java'  //引用 application plugin 獲得執行 java 專案相關的 task  apply plugin:'application'  //執行 application plugin 用到的參數  mainClassName = "com.example.Main"  //設定 repository server  repositories {  mavenCentral()  }  //宣告專案的相依函式庫  dependencies {  compile group: 'commons-logging', name: 'commons-logging', version: '1.1.1'  compile group: 'log4j', name: 'log4j', version: '1.2.16'  } |
| build.gradle基本結構(說明) |
| 1. apply plugin : 引用插件。   Gradle Plugins 插件 Gradle 在它的核心中有意地提供了一些小但有用的功能，用于在真實世界中的自動化。 所有有用的功能，例如以能夠編譯Java 代碼为例，都是通過插件進行添加的。   1. mainClassName: 入口程式的名稱(需要加上package)。 2. repositories{}: lib來源server，可以當作你要從哪裡抓lib，上面範例是從maven官方函式庫抓lib。 3. dependencies{}: 專案需要使用的函示來倒入。 |
| Closure(閉包) |
| Gradle 支援Closure，而Closure(封閉、閉包)是一個擁有閒置變數(Free Variable)的運算式。  閒置變數真正扮演的角色依當時詞彙環境而定。支援閉包的程式語言通常具有一級函式（First-class function）。建立函式不等於建立閉包。如果函式的閒置變數與當時語彙環境綁定，該函式才稱為閉包。  function doSome(){  var x = 10;  function f(y){  return x+y;  }  return f;  }  var foo = doSome(); //建立一個函式物件  foo(20); //結果為30  foo(40); //結果為50  從上面例子來看，f建立了一個閉包，f當中的x參數看似好像沒定義，其實x是從外部函式doSome()捕捉而來。閉包是個捕捉了外部函式變數(或使之繼續存活)的函式。以上範例，函式f建立了閉包，因為它將變數x關入了自己的函式當中。如果形式閉包的函式物件持續存活，被關閉的變數x也會繼續存活。就像是延續了變數x的生命週期。  看起來閉包應該是{}的x+y兩者產生的一種形式。 |
| Gradle 支援Closure |
|  |
| Build Script 與 Project 物件 |
| 我們寫的build.gradle檔案，它是Gradle Build Script的預設檔名。  Build Script檔案被Gradle載入後轉換成BuildScript物件，本質上它是一個Groovy Script。  Groovy Script可以設定Base Class，它的效果就如同替Closure指定delegate(代表)一般，任何你在Build Script內使用的方法、屬性都會交給Base Class處理，對Gradle來說它將這個Base Class封裝成Project物件。  根據Gradle官網提示: There is a one-to-one relationship between a Project and a “build.gradle" file. During build initialisation, Gradle assembles a Project object for each project which is to participate in the build.  (Project 與 build.gradle文件之間存在一對一的關係。在建構初始化期間，Gradle為每個參與建構的項目組裝一個Project物件)。 |
| Build Script 與 Project 物件嘗試解讀 |
| apply plugin: 'java'  apply plugin: 'application'    mainClassName = "tw.com.codedata.HelloWorld"    repositories {      mavenCentral()  }    dependencies {      compile group: 'commons-logging', name: 'commons-logging', version: '1.1.1'      compile group: 'log4j', name: 'log4j', version: '1.2.16'  }  以上build.gradle本質上就是一個java object。所以它將會有對應的method能使用。  1. apply plugin: 'java'  Javadoc URL: <https://docs.gradle.org/current/javadoc/org/gradle/api/plugins/PluginAware.html#apply-groovy.lang.Closure->  可以找到三種方法      (2)  (3)  在文件中可以查到三個apply方法，根據以上範例，我們使用的是Map那組，也可以知道有三種Key值，from、plugin、to。 |
| Gradle DSL文件導讀 |
| 可以參考URL: <https://docs.gradle.org/current/dsl/>     1. Build script => build.gradle   對應一個project   1. Settings script => settings.gradle   看起來比較偏向多個project的導入運作。     1. Init script => 尚未研究 |
| 參考網站 |
| 1. 基本說明：   <http://www.codedata.com.tw/java/understanding-gradle-3-getting-started/>   1. 插件說明：   <https://www.kancloud.cn/kancloud/gradle2-user-guide/52725>   1. Grable Project API: <https://docs.gradle.org/current/javadoc/org/gradle/api/Project.html> |

## 4設計模式- SOLID

|  |
| --- |
| SOLID |
| 簡介與說明 |
| 1. S = Single Responsibility Principle(SRP) 單一職責(單一責任原則)   指一個類別只負責一件事情，從這樣的講解當中，我們要注意不能將功能切得太細導致過度設計(over design)的情況。  例如一個方法(method)就只做一件事情，一個類別(class)就只做一個功能該做的事情，多於的變數(value)、方法(method)就不需要出現。   1. O = Open/Close Principle(OCP) 開放/封閉原則   物件導向設計當中最重要的開放(擴充)與封閉(修改)原則。一套軟體必須保留高延展性，方便日後的更新與修改，必須減少過度的耦合(Coupling)發生。而延展模組就是物件導向中的繼承  ，而延展的方法相當多，例如直接繼承舊有的模組並且覆蓋新功能上去(Override)、繼承抽象類別、透過抽象類別的規範來實現新功能、透過多型(Overload)來實現新功能。   1. L = Liskov Substitution Principle(LSP) Liskov替換   在一個系統中，子類別應該要可以替換掉父類別而不會受到影響程式架構。  子類別應該要可以執行父類別想做的事情。   1. I = Interface Segregation Principle(ISP) 介面隔離   把不同功能的功能從介面中分離出來。   1. D = Dependency Inversion Principle(DIP) 依賴凡轉   高階模組不應依賴低階模組，兩個都應該依賴在抽象概念上;抽象概念不依賴細節，而是細節依賴在抽象概念。意思就是說: 話不能說的太死，盡量講一些概念性的東西。  子類別肯定依賴父類別，而父類別不能去依賴子類別實作的功能。  抽象類別不應該有太多細節，而實作類別依賴抽象細節，意思是說依賴應該是單向的。因為程式設計上最好能達到低耦合，可惜光繼承就是一個依賴，因此單向的依賴是最乾淨的，雙向的依賴就違反常理的依賴會導致程式碼不容易追蹤。 |

## 5設計模式- Simple Factory Pattern

|  |
| --- |
| Simple Factory模式(工廠模式) |
| 簡介 |
| 1. 又稱 Static Factory。(此處要特別注意，不一定產生物件的方法要設為static方法，所以第一項 又稱Static Factory 其實有點衝突)。 2. 對客戶端隱藏產品產生的細節，物件如何生成，生成前是否與其他物件建立依賴關係，客戶端皆不用理會，用以將物件生成方式之變化 與客戶端程式碼隔離。 3. 根據需求，直接由工廠產生該物件出來。 |
| 適用時機 |
| 1. 統一建立物件。 2. 有效率的產生、管理、操作物件。 |
| 參考範例 |
| 1. 我們設計了一個介面，這個介面專門讀取不同格式的檔案      1. 我們建立了兩個實現這個介面的檔案類型(XML、JSON)    1. JSONLoader =>專門讀取JSON格式      * 1. XMLLoader =>專門讀取XML格式      1. 建立一個Factory      1. 當我們要使用某個類別時，直接使用了工廠來產生這個物件     我們可以從範例中發現，註解的地方為 未使用SimpleFactoryPattern的設計模式，如果我要讀取XML格式內容就必須產生一個新物件。  但是這樣有一個問題就是，如果我今天把XMLLoader類別名稱改成了XALLoader類別名稱的話  那麼範例中的new XMLLoader就會出現錯誤，因為這樣產生了高依賴性。  所以我們統一集中在一個類別(class)當中，並且由這個Factory幫我們產生。  這就是SimpleFactoryPattern的基本範例。  但是以上有一個問題，如果我今天再加入CSVLoader類別(class)的話，我就要去動到Factory當中的getLoader方法(method)，這個行為其實我們違反了SOLID設計原則的OCP原則，所以比較正統的工廠模式有另一種寫法，可以參考 工廠方法模式(Factory Method Pattern)。  我們可以從程式碼中看到  C:\Users\ASUS\Desktop\擷取.JPG  如果今天我們把JSON改成CSV，那麼代表著  C:\Users\ASUS\Desktop\擷取.png  我們必須把LoaderFactory內的getLoader方法進行修改，並且在LoaderType 的enum進行添加CSV相關的程式碼。以上行為就會違法SOLID當中的O(OCP)原則。   1. UML |

## 6設計模式- Factory Method Pattern

|  |
| --- |
| Factory Method模式(工廠方法模式) |
| 簡介 |
| 1. 定義一個建立物件(Object)的介面(inferface)，但讓實現(implements)這個介面的類別(class)來決定實體化哪個類別。(出處:維基百科) 2. 由子類別(subClass)來實現這個方法來建立具體類別的物件。(出處:維基百科) 3. 工廠方法帶來的方便在，無須修改原本的程式碼，在符合OCP原則情況之下，當產品發生改變時，只要在新增功能上去即可。 4. 缺點是只能建立單一類型的功能。 5. 與SimpleFactory不同的是，我們將工廠也一併切分了，而不是一個工廠統一生產多種不同類型物件。 |
| 適用時機 |
| 1. 建立物件需要大量重複的程式碼。 2. 有效率的產生、管理、操作物件。 |
| 參考範例 |
| 1. 我們建立了 TV 的規範，也就是介面(interface)或者抽象(abstract)，範例中使用介面。     這個規範就是，所有的TV都具備開與關的功能。   1. 並且建立了兩個類別來實現這個TV(練習用)。        1. 建立一個工廠的介面(interface)用來規範建立的標準。      1. 並且子類別來實現這個工廠規範。     這裡要特別注意，我們將產生TV的規範切分，不同類型的TV由不同的工廠(Factory)處理。     1. 最後由呼叫端來決定要建立哪一個物件。     範例當中我們決定產生ProductTV\_B 所以我們直接呼叫專門產生ProductTV\_B的工廠(Factory)來進行產生實體動作。   1. UML   C:\Users\ASUS\Desktop\Class Diagram3.jpg |

## 7設計模式- Abstract Factory Pattern

|  |
| --- |
| Abstract Factory模式(抽象工廠模式) |
| 簡介 |
| 1. 提供一個建立一系列相關或相互依賴物件的介面，而無需指定它們具體的類別。 2. 管理有關聯性的物件。 |
| 參考範例 |
| 1. UML     由子類別來產生對應的實體。   1. 假設:Audi和BMW都有生產休旅車，而我的工廠專精生產休旅車，不一定需要會做某品牌全系列的商品。在我想要生產其他車種時只需要抽換具體工廠即可(更換生產的產品)。 2. 定義Audi的介面     定義BMW的介面     1. 定義產品            1. 抽象工廠可以製作某品牌的Jeep或是SUV      1. 不同工廠產生不同廠牌的車種        1. 測試的方式     可以看到，當我們要產生不同的車種，我們可以直接抽換成要產生的工廠，這樣就不會有問題了。 |
| 參考網站 |
| 1. <https://ithelp.ithome.com.tw/articles/10208955> |

## 8設計模式- Builder Pattern

|  |
| --- |
| Builder模式() |
| 簡介 |
|  |
| 參考網站 |
|  |

## 8設計模式- Reflection Factory Pattern

|  |
| --- |
| Reflection Factory模式(反射工廠模式) |
| 簡介 |
|  |
| 參考網站 |
| <https://blog.amowu.com/2009/08/factory-pattern.html> |

## 9設計模式- Singleton

|  |
| --- |
| Singleton模式(單例模式) |
| 簡介 |
|  |
| 參考網站 |
|  |

## 6技術框架- Hibernate

|  |
| --- |
| Hibernate |
| 簡介 |
|  |
| 優缺點 |
|  |

## 9自動化打包工具- Jenkins

|  |
| --- |
|  |
| 簡介 |
|  |
| 優缺點 |
|  |

## 11 Server- Oracle WebLogic Server

|  |
| --- |
|  |
| 簡介 |
|  |
| 優缺點 |
|  |

## 12 Server- WildFly JBoss

|  |
| --- |
| WildFly JBoss |
| 簡介 |
|  |
| 優缺點 |
|  |

## 13 Server- Apache Tomcat

|  |
| --- |
| Apache Tomcat |
| 簡介 |
|  |
| 優缺點 |
|  |

## 14 HTTP概念

|  |
| --- |
| HTTP |
| 簡介 |
|  |
| 優缺點 |
|  |

## 14 UML概念

|  |
| --- |
| UML |
| 簡介 |
|  |
| 優缺點 |
|  |