

Outline

Module 1 : 大數據簡介

Module 2 : Hadoop Ecosystem介紹

Module 3 : Hadoop 平台安裝

Module 4 : Hadoop 分散式檔案系統 (HDFS)

Module 5 : Hadoop MapReduce

Module 6 : Apache Hive

Module 7 : Sqoop與Flume

Module 8 : Apache Spark

Module 9 : Spark 平台安裝

Module 10 : RDD – Resilient distributed dataset

Module 11 : Scala 程式開發基礎

Module 12 : Spark SQL 及 DataFrame

Module 13 : Spark 機器學習函式庫(MLlib)



Scala 語言簡介



- ▶ Scala 是 Scalable Language 的意思(可大可小的語言)
- ▶ Scala 是 **物件導向** 和 **函數式**語言的混合
 - 透過lambda expression讓程式更簡潔

```
Scala: List(1,2,3,4,5).foreach(x=>println("item %d".format(x)))
```

Java:

```
Int[] intArr = new Array[] {1,2,3,4,5};  
for (int x: intArr) println(String.format("item %d", x));
```

- ▶ scala可叫用Java及.NET龐大的函式庫
- ▶ 對**平行處理**有更佳的支援(函數式語言、actor model、akka)
- ▶ Spark 的原生實作語言，支援最完整、文件也最齊全

Scala常用語法整理

- ▶ import
 - import org.apache.spark.SparkContext
 - import org.apache.spark.rdd._ (引入rdd底下所有class)
 - import org.apache.spark.mllib.clustering.{ KMeans, KMeansModel } (只引入clustering底下兩個class)
- ▶ 變數宣告
 - val int1: Int = 5 (給值後不能再重新給值，否則會error)
 - var int2: Int = 5 (可不斷給新值)
 - val int = 5 (不宣告型別，由編譯器推斷)
- ▶ 方法定義(無回傳值)
 - def voidFunc(param1: Type, param2: Type2) = { ... }

```
def setLogger = {  
    Logger.getLogger(com).setLevel(Level.OFF)  
    Logger.getLogger(io).setLevel(Level.OFF)  
}
```

Scala常用語法整理

▶ 方法定義(單一回傳值)

- `def rtnFunc1(param1: Type, param2: Type2): Type3 = {
 val v1: Type3 = ...
 v1 //最後一行為回傳值，型態要和宣告相符
}`

▶ 方法定義(多回傳值)

- `def rtnFunc2(param1: Type, param2: Type2): (Type3, Type4) = {
 val v1: Type3 = ...
 val v2: Type4 = ...
 (v1, v2)
 //最後一行為回傳值，個數及型態要和宣告相符
}`

```
def getMinMax(intArr: Array[Int]): (Int, Int) = {  
    val min = intArr.min  
    val max = intArr.max  
    (min, max)  
}
```

Scala常用語法整理

▶ 取得方法執行結果

- `val res = rtnFunc1(param1, param2)` (針對單一回傳值方法，`res`可不宣告回傳型別)
- `val (res1, res2) = rtnFunc2(param1, param2)` (針對多回傳值方法，`res1, res2`可不宣告回傳型別)
- `val (_, res2) = rtnFunc2(param1, param2)` (針對多回傳值方法可用底線代表不使用之回傳值)

```
val (min,max)=getMinMax(intArr)  
val (_, max)=getMinMax(intArr)
```

▶ For Loop使用

- `for (i <- collection) { ... }`

▶ For Loop使用 (使用`yield`關鍵字產生集合物件)

- `val rtnArr = for (i <- collection) yield { ... }`

```
val intArr = Array(1,2,3,4,5,6,7,8,9)  
val multiArr=  
  for (i <- intArr; j <- intArr)  
  yield { i*j }  
//multiArr為長度81的陣列，內容為99乘法表
```

Scala常用語法整理

```
val intArr = Array(1,2,3,4,5,7,8,9)
val res=getMinMax(intArr) //res=(1,9)=>tuple
val min=res._1 //取得res第一個元素
val max=res._2 //取得res第二個元素
```

▶ Tuple的使用

- Tuple 是一種**固定長度**，但元素可以是**不同型別**的容器。
- 以val v=(v1,v2,v3...)方式宣告，透過v._1, v._2, v._3...取值
- 常搭配lambda語法一同使用

```
val intArr = Array((1,2,3),(4,5,6),(7,8,9)) //intArr為Tuple陣列
val intArr2=intArr.map(x=> (x._1 * x._2 * x._3))
//intArr2: Array[Int] = Array(6, 120, 504)
val intArr3=intArr.filter(x=> (x._1 + x._2 > x._3))
//intArr3: Array[(Int, Int, Int)] = Array((4,5,6), (7,8,9))
```

- 在lambda中常見以**底線(_)**代表值的方式以簡化寫法

```
val intArr = Array((1,2,3),(4,5,6),(7,8,9)) //intArr為Tuple陣列
def getThird(x:(Int,Int,Int)): Int = { (x._3) }
val intArr2=intArr.map(getThird(_))
val intArr2=intArr.map(x=>getThird(x)) //與上一行相同的對應寫法
//intArr2: Array[Int] = Array(3, 6, 9)
```

Scala常用語法整理

▶ Class

- Scala的Class用法和JAVA的Class大致相同，不過寫法可更簡潔
 - 成員變數或方法若沒明確指定飾字（如 private / protected）則預設為public
 - 成員變數可在Class的定義裡直接定義

```
Scala:
class Person(userID: Int, name: String) //定義兩個private的成員變數

class Person(val userID: Int, var name: String)
//定義兩個public的成員變數，userID只能給值一次
val person = new Person(102, John Smith)//設定成員變數值
person.userID //回傳102
```

第一個Person class定義的Java寫法:

```
public Class Person {
    private final int userID;
    private final String name;
    public Person(int userID, String name) {
        this.userID = userID;
        this.name = name;
    }
}
```

Scala常用語法整理

▶ Object

- Scala中沒有 static 變數與函式的概念，所有成員及方法均依存於 instance
- Scala中可透過Object來實作static的用法
 - Scala的Object其實是singleton的class instance

▶ Scala Object vs Class

- object用於定義utility或讓Spark執行之Driver Program
- class用於定義裝載資料之Entity

Scala Object:

```
object Utility {  
  def isNumeric(input: String): Boolean = input.trim()  
    .matches(s"[+-]?((\\d+(e\\d+)?[lL]?)|(((\\d+(\\.\\d*)?)|(\\.\\d+))(e\\d+)?[fF]?)))")  
  
  def toDouble(input: String): Double = {  
    val rtn = if (input.isEmpty() || !isNumeric(input)) Double.NaN else input.toDouble  
    rtn  
  }  
}
```

```
val d = Utility.toDouble(20) //可直接使用成員方法，不必先new
```


scala中常用的集合操作

- ▶ 宣告集合：
 - `val intArr = Array(1,2,3,4,5,7,8,9)`
- ▶ 集合串接
 - `val intArrExtra = intArr ++ Array(0,11,12)`
- ▶ `map`: 由現有集合內容建立新的集合
- ▶ `filter`: 取出現有集合中符合特定條件的資料，建立子集合
- ▶ `join`: 整合兩個Map中相同Key的所有資料，建立新的Map
- ▶ `sortBy`、`reverse`: 依據集合內容進行排序
- ▶ `take(N)`: 取出集合中前N筆資料建立新的集合

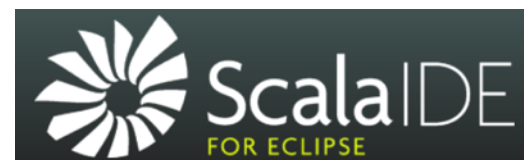
```
val intArr = Array(1,2,3,4,5,7,8,9)
val intArr2=intArr.map(_ * 2)
//intArr2: Array[Int] = Array(2, 4, 6, 8, 10, 12, 14, 16, 18)
val intArr3=intArr.filter(_ > 5)
//intArr3: Array[Int] = Array(6, 7, 8, 9)
val intArr4=intArr.reverse
//intArr4: Array[Int] = Array(9, 8, 7, 6, 5, 4, 3, 2, 1)
```

scala中常用的集合操作

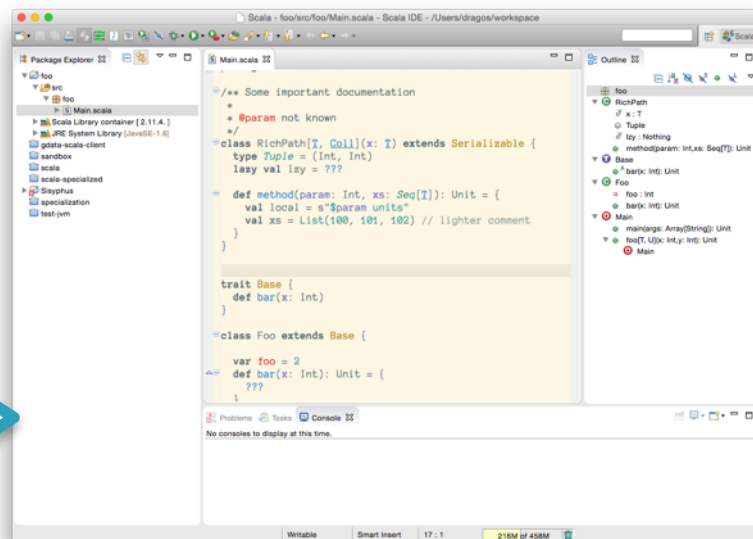
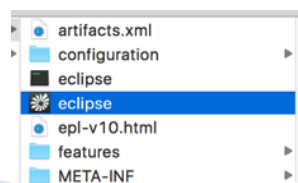
- ▶ sum: 取得數字集合的總合
 - val sum = Array(1,2,3,4,5,7,8,9).sum
- ▶ max: 取得數字集合中的最大值
 - val max = Array(1,2,3,4,5,7,8,9).max
- ▶ min: 取得數字集合中的最小值
 - val max = Array(1,2,3,4,5,7,8,9).min
- ▶ distinct: 取得集合中，不重覆的元素值

```
val intArr = Array(1,2,3,4,5,7,8,9)
val sum = intArr.sum
//sum = 45
val max = intArr.max
//max = 9
val min = intArr.min
//min = 1
val disc = Array(1,1,1,2,2,2,3,3).distinct
//disc = Array(1,2,3)
```

開發工具(IDE)使用介紹

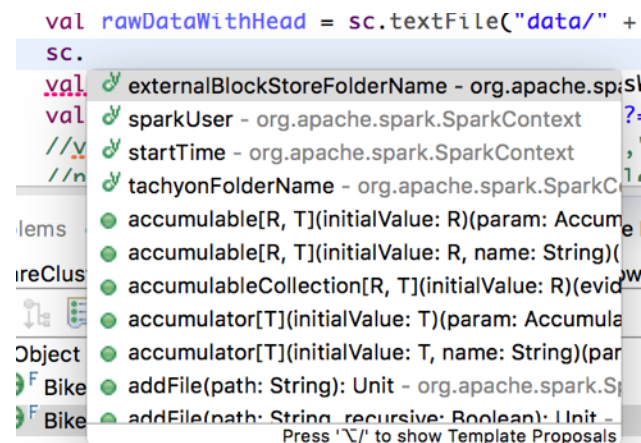


- ▶ spark-shell好用歸好用，但是土法鍊鋼、鑽木取火還是很辛苦
- ▶ ScalaIDE for eclipse 4.4.1
 - <http://scala-ide.org/download/sdk.html>
 - 請依自己電腦的作業系統下載適用版本
 - 進行安裝(解壓縮)
 - 建立捷徑
 - 開啟ScalaIDE



Scala IDE for eclipse

- ▶ 用於開發及偵錯Driver Program(word complete及breakpoint總能讓工程師感到安定…)
- ▶ 用於打包spark-shell可執行之jar檔
- ▶ 內容物
 - Eclipse 4.4.2 (Luna)
 - Scala IDE 4.4.1
 - Scala 2.11.8 and Scala 2.10.6
 - Sbt 0.13.8
 - Scala Worksheet 0.4.0
 - Play Framework support 0.6.0
 - ScalaTest support 2.10.0
 - Scala Refactoring 0.10.0
 - Scala Search 0.3.0
 - Access to the full Scala IDE ecosystem



```
val rawDataWithHead = sc.textFile("data/" +  
sc.  
val externalBlockStoreFolderName - org.apache.spark.  
val sparkUser - org.apache.spark.SparkContext  
//val startTime - org.apache.spark.SparkContext  
//val tachyonFolderName - org.apache.spark.SparkContext  
● accumulable[R, T](initialValue: R)(param: Accumulable[R, T])  
● accumulable[R, T](initialValue: R, name: String)(param: Accumulable[R, T])  
● accumulableCollection[R, T](initialValue: R)(evidence: Evidence[R, T])  
● accumulator[T](initialValue: T)(param: Accumulable[T, T])  
● accumulator[T](initialValue: T, name: String)(param: Accumulable[T, T])  
● addFile(path: String): Unit - org.apache.spark.SparkContext  
● addFile(path: String, recursive: Boolean): Unit - org.apache.spark.SparkContext  
Press 'Ctrl' to show Template Proposals
```

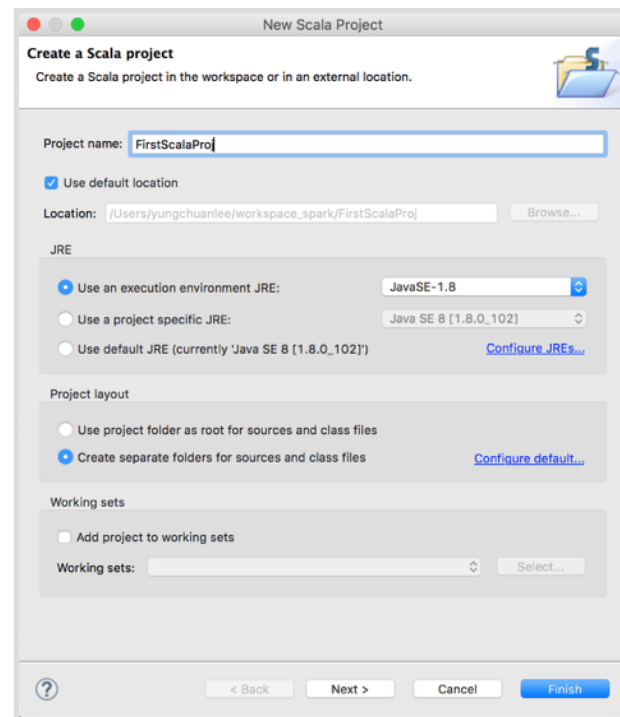
開始吧！用Scala IDE開發Driver Program

- ▶ 使用Scala IDE開發Driver Program有以下幾個步驟
 - 建立Scala Project
 - 設定Build Path
 - 加入Spark函式庫
 - 設定Scala版本
 - 新增package
 - 以package管理程式(視個人開發習慣)
 - 新增scala object
 - 實作邏輯
 - 執行debug程序
 - 匯出Jar檔
 - 透過spark-submit將程式交由Spark執行



建立Scala Project

- ▶ 在Scala IDE中
 - 工具選單 FILE -> NEW -> Scala Project
 - 輸入project名稱：FirstScalaProj
 - JRE建議選用1.8版本(1.7亦可)
 - 其餘選項使用預設值即可
 - 直接 [Finish]



設定Build Path

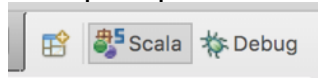
- ▶ 在Package Explorer或Project Explorer中，對FirstScalaProj按右鍵，選擇Build Path -> Configure Build Path

[Tips]:

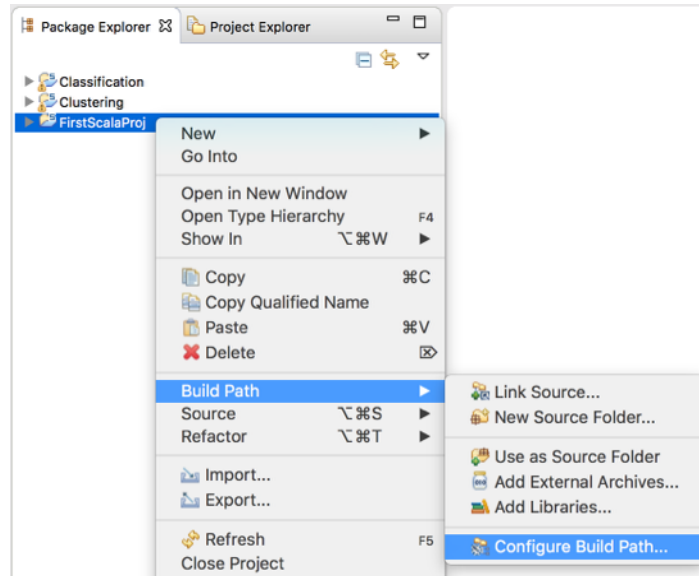
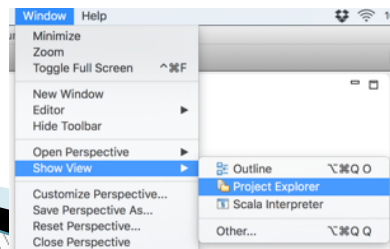
Q: 找不到Package / Project Explorer ?

A:

- ▶ 先確定目前是否在Scala perspective

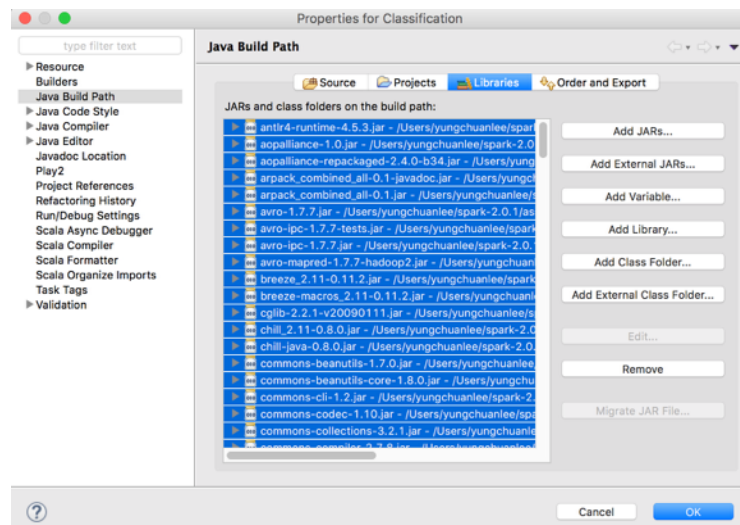


- ▶ 若已在Scala perspective仍找不到，則可透過工具列-> Window -> Show View



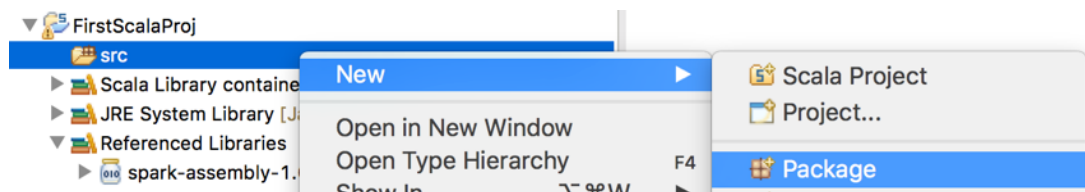
設定Build Path

- ▶ 開發Spark Driver Program需要對Build Path作以下設定：
 - 將所需的jar檔加入
 - 設定專案使用之Scala Library Container版本為2.11.8(IDE預設即為2.11.8，確認即可)
- ▶ 點選Configure Build Path後會出現設定Java Build Path的視窗，點選Libraries頁籤->Add External JARs...
 - Spark開發所需的Jar位於 Spark安裝目錄/assembly/target/scala-2.11/jars/ 底下
 - 全選所有的jar檔後，按確定
- ▶ 回到Java Build Path設定視窗，確認Scala Library Container的版本為 [2.11.8] ，按確定

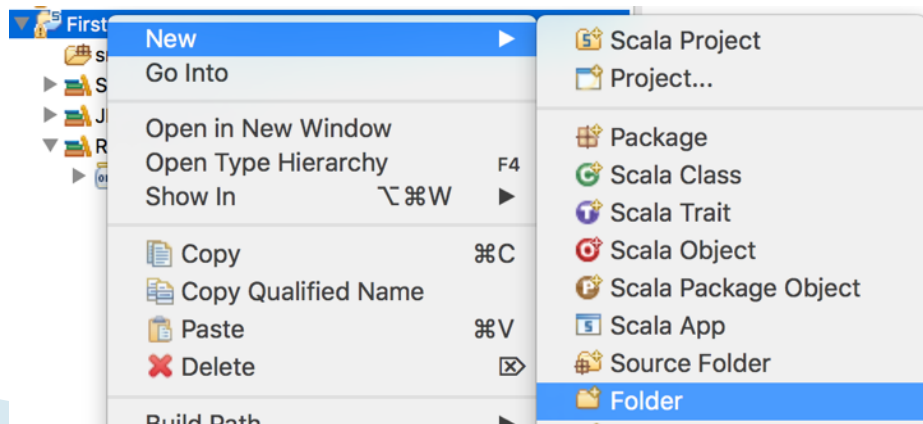


設定專案目錄、新增Package

- ▶ 在Package Explorer中展開FirstScalaProj，在src目錄下新增package
 - 對src目錄按右鍵->New->Package(或用快速鍵Ctrl + N)

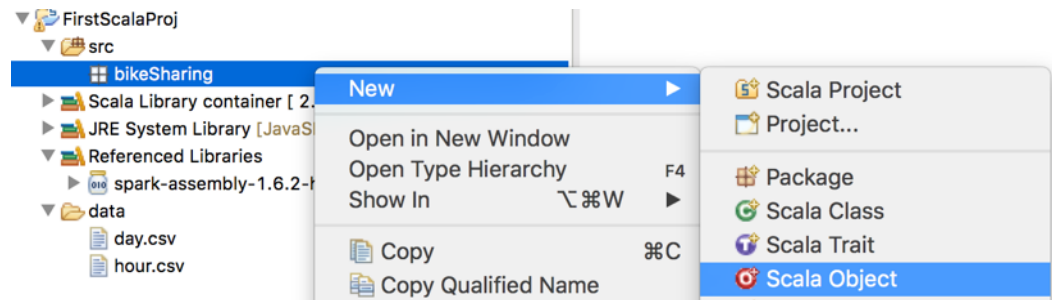


- 新增bikeSharing Package
- ▶ 在FirstScalaProj下新增data目錄(Folder)，以存放input檔案及輸出結果

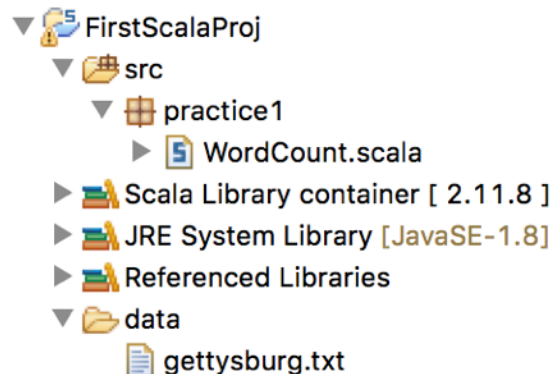


新增Scala Object

- ▶ 將要讀取的檔案(gettysburg.txt)copy至data目錄下
- ▶ 在bikeSharing Package下新增Scala Object : BikeCountSort



- ▶ 最後專案目錄結構：



WordCount的實作解析

```
package practice1
```

```
//spark lib
```

```
import org.apache.spark.SparkContext  
import org.apache.spark.SparkConf  
import org.apache.spark.rdd.RDD
```

import所需之
Library

```
//log
```

```
import org.apache.log4j.Logger
```

```
import org.apache.log4j.Level
```

以object來實作，並實作main方法

```
object WordCount {
```

```
  def main(args: Array[String]): Unit = {
```

```
    //設定Log顯示層級，減少Console資訊
```

```
    Logger.getLogger(org).setLevel(Level.ERROR) //mark for MLib INFO msg
```

```
    val sc = new SparkContext(new SparkConf().setAppName(WordCount).setMaster(local[*]))
```

```
    val rawRdd = sc.textFile("data/gettysburg.txt").flatMap { x=>x.split( ) }
```

```
    //不分大小寫(toLowerCase轉成全小寫)，不含空白(用filter篩選)
```

```
    val txtRdd = rawRdd.map { x => x.toLowerCase.trim }.filter { x => !x.equals( ) }
```

```
    val countRdd = txtRdd.map { x => (x, 1) } //先針對每個字組作成 (字組, 1)的Map
```

```
    val resultRdd = countRdd.reduceByKey { case (a, b) => a + b } //以ReduceByKey加總相同的字組
```

```
    val sortResRdd = resultRdd.sortBy((x => x._2), false) //以加總次數由大到小排序
```

```
    sortResRdd.take(5).foreach { println } //印出前五個
```

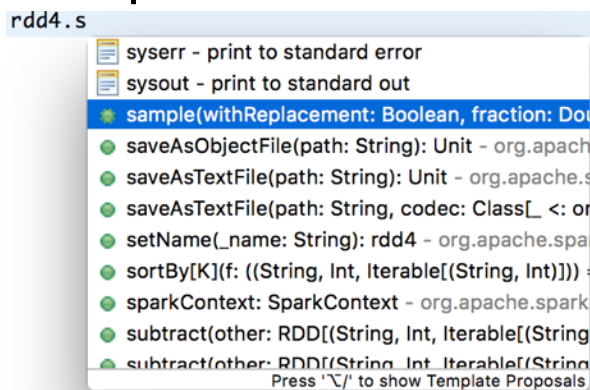
```
    sortResRdd.saveAsTextFile(data/wc_output)
```

以saveAsTextFile輸出結果至文字檔

```
  }  
}
```

善用IDE工具加速開發

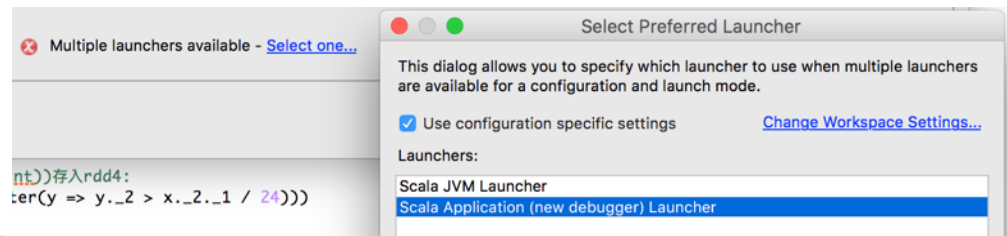
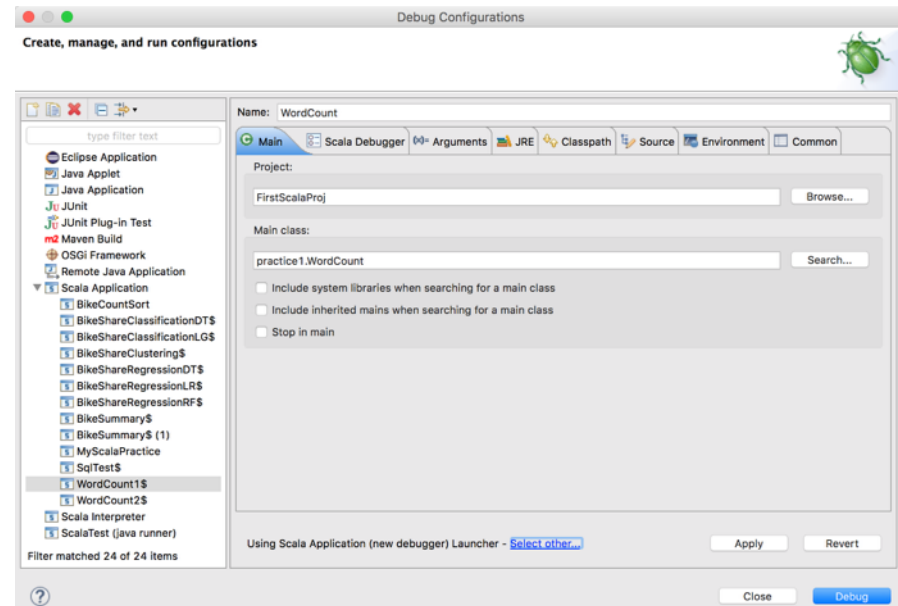
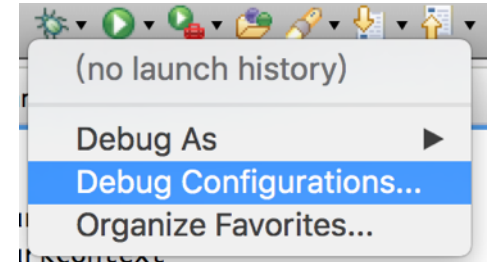
- ▶ 在編輯器中打字即會出現word complete視窗，或用**ALT + /**，亦可顯示word complete視窗



- ▶ **滑鼠指向變數**，會浮出視窗顯示變數型別(對於操作tuple的變數而言，知道型別是一大福音)

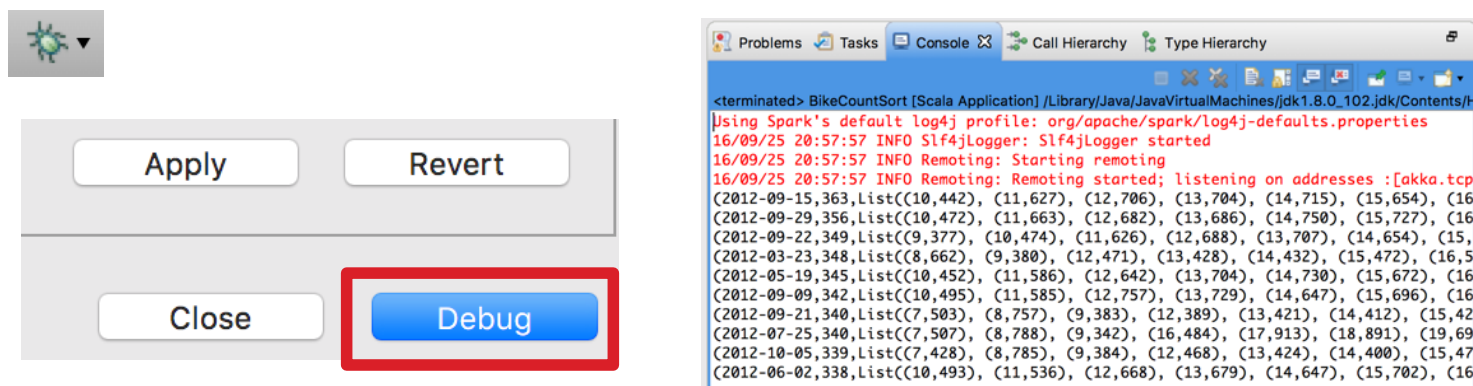
Debug Configuration

- ▶ 設定debug configuration 
 - 按小蟲icon，開啟Debug Configurations...
 - 在Scala Application下新增Debug設定：
 - Name : WordCount
 - Project : FirstScalaProj
 - Main Class :
practice1.WordCount
 - 設定Launcher



進行偵錯

- ▶ 完成偵錯設定後，可按小蟲icon或設定Debug Configuration視窗之Debug鈕執行偵錯，在console中可看到執行結果

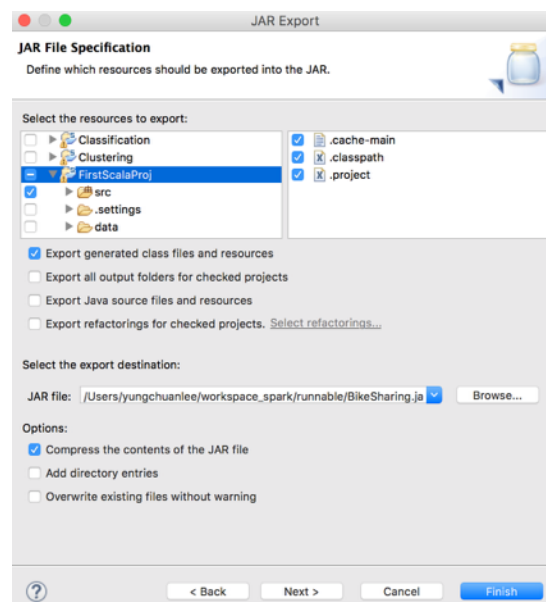
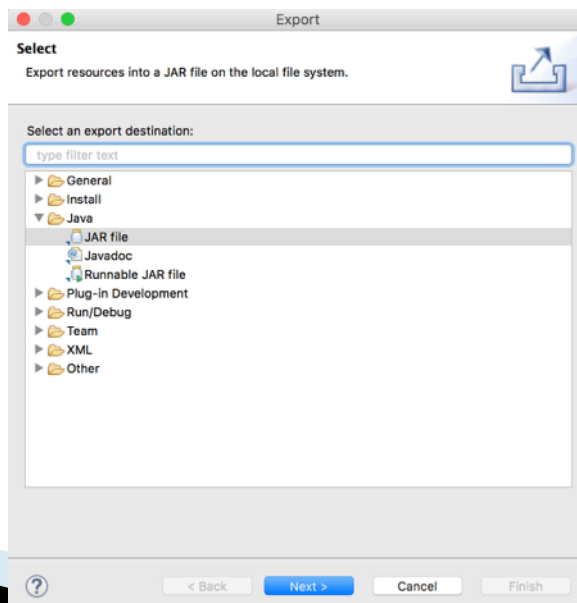


[Tips]

- 到data/output看看sortResRdd的內容是否順利寫入(查看part-xxxx檔案內容)
- 調整程式中Log Level設定，再執行看看console顯示訊息量的變化
- 重新執行前若不刪除output會發生什麼事情？

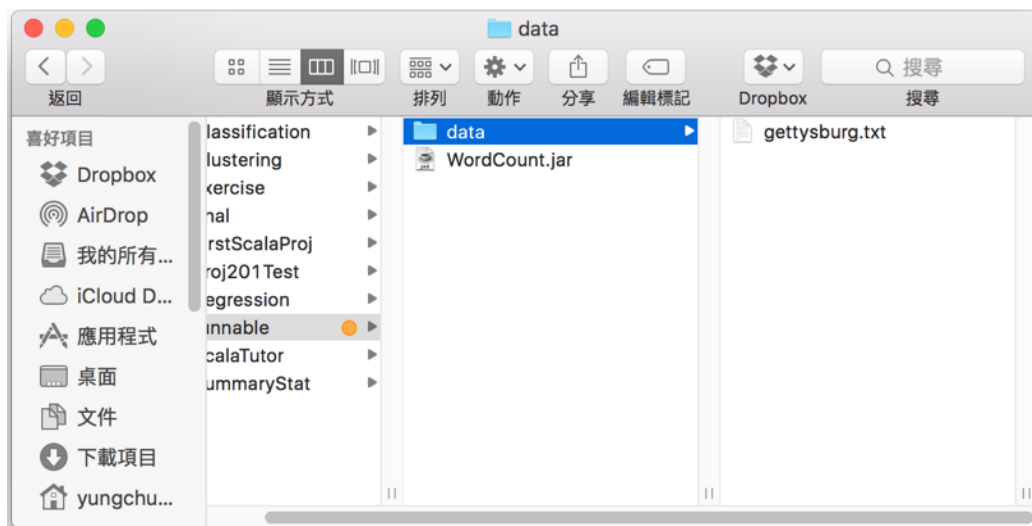
匯出JAR檔

- ▶ 使用Spark-Submit前，先將程式匯出成JAR檔，以簡化指令
 - 在Package Explorer中對FirstScalaProj按右鍵->Export...->Java/JAR file->勾選FirstScalaProj中的src目錄、輸入JAR File的匯出路徑



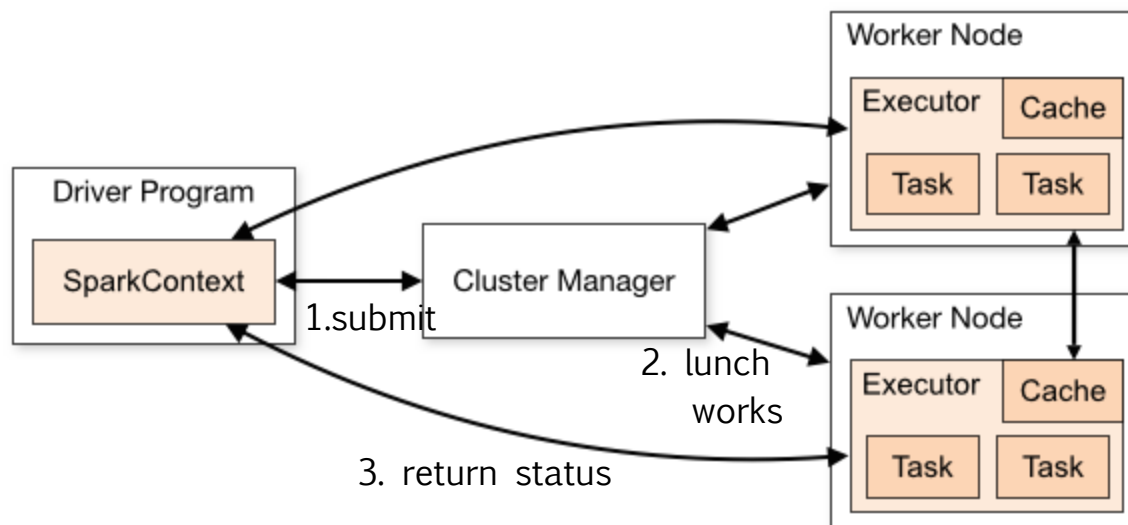
在Spark-submit之前…

- ▶ 要先將input檔案及output目錄放在JAR File的所在目錄
 - 以我們的程式寫法而言，需將data目錄放在JAR File的所在目錄



在Spark-submit之前……

▶ spark-submit指令運作流程



透過Spark-submit submit JOB

- ▶ 開啟Command Line視窗，**切換目錄至JAR File所在目錄**
- ▶ Spark-submit指令格式

```
./bin/spark-submit  
--class <main-class> (package名稱+scala object名稱)  
--master <master-url> (指定master之URL，本機模式為local[Worker thread num])  
--deploy-mode <deploy-mode> (指定Worker在Cluster或Client，不指定即Client)  
--conf <key>=<value> (指定Spark執行參數，可省略)  
... # other options  
<application-jar> (JAR檔名稱)  
[application-arguments] (指定Driver的main方式接收之參數，可省略)
```

Spark安裝目錄/bin/spark-submit --class practice1.WordCount --
master local[*] WordCount.jar

[Tips]:

- ▶ spark-submit需在JAR檔所在目錄呼叫，才能正確讀取data目錄裡的資料
- ▶ merge output的結果到一個檔案：
 - linux: cat data/output/part-* > res.txt
 - windows: type data\output\part-* > res.txt

Word Count變化一下...

- ▶ [Exercise] 在wordCount Package中新增WordCount2 Object，實作以下功能：
 - 讀取gettysburg.txt，以空格()切開字組，計算每個字組在文中出現的次數
 - 不分大小寫，不計入空白
 - 取得每個字組最後在文中出現的位置
 - Hint1: 取得每個字組在文中的位置(index)
 - `val posRdd=txtRdd.zipWithIndex()`
 - Hint2: 使用reduceByKey或groupByKey取得每個字組對應的最大index