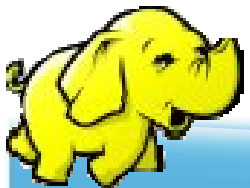


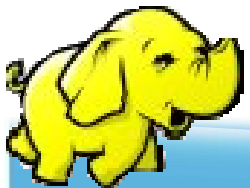
五、開發Hadoop Map/Reduce

程式設計者只需要解決”真實的”問題，
架構面留給MapReduce

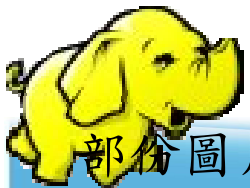
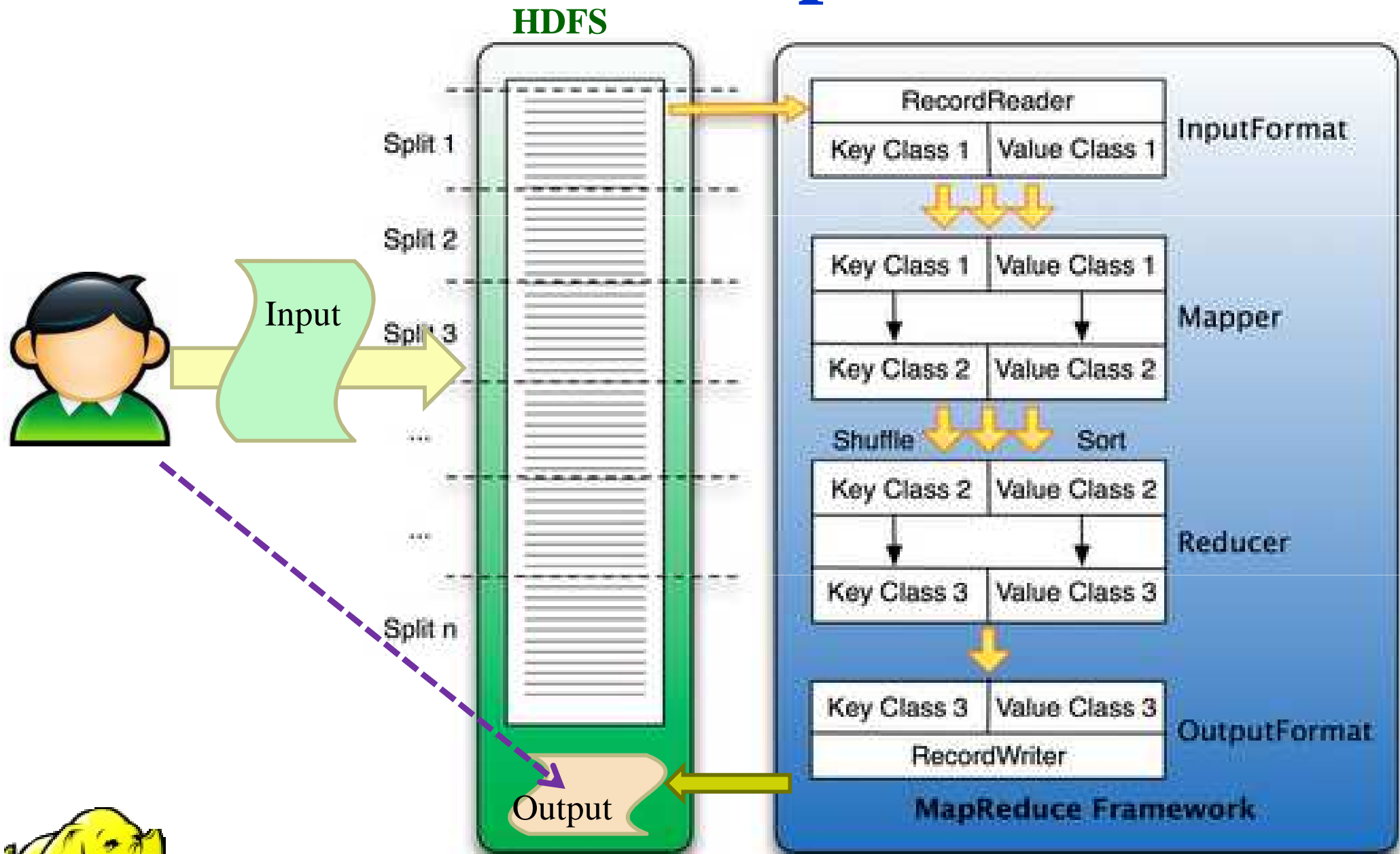


Hadoop 的 MapReduce API 提供

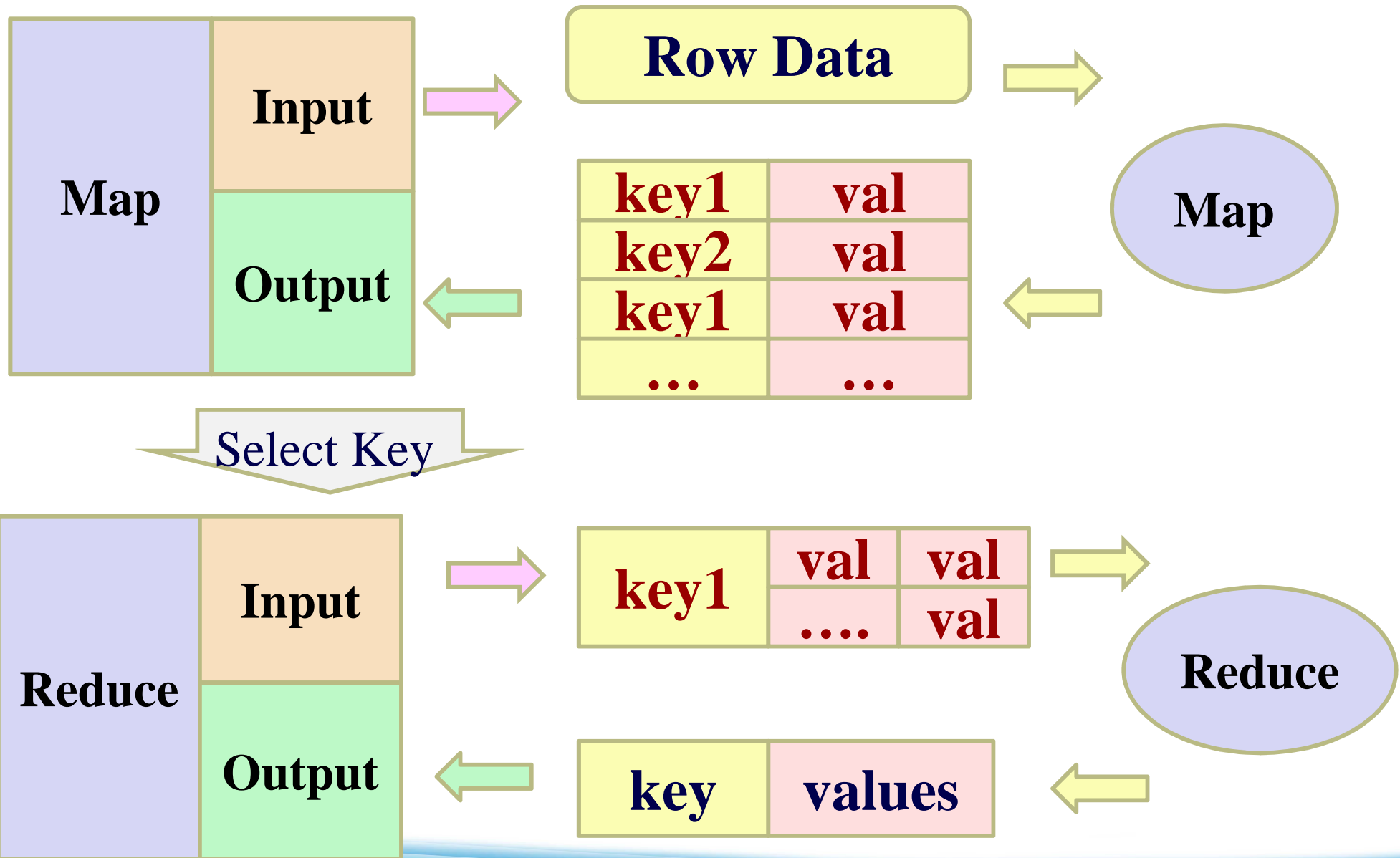
- 自動的平行化與工作分配
- 容錯特性
- 狀態監控工具
- 一個乾淨的抽象化(abstraction)供程式設計師使用



HDFS & MapReduce



<Key, Value> Pair



Program Prototype (v 0.20)



Class Mapper (v 0.20)

```
import org.apache.hadoop.mapreduce.Mapper;
```

```
1 class MyMap extends
```

```
Mapper < INPUT KEY Class, INPUT VALUE Class, OUTPUT KEY Class, OUTPUT VALUE Class >
```

```
2 {
```

```
3 // 全域變數區
```

```
4 public void map ( INPUT KEY Class key, INPUT VALUE Class value,
```

```
Context context ) throws IOException, InterruptedException
```

```
{
```

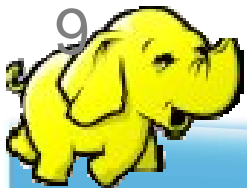
```
5 // 區域變數與程式邏輯區
```

```
6 context.write( NewKey, NewValue);
```

```
7 }
```

```
8 }
```

```
9
```



Class Reducer (v 0.20)

```
import org.apache.hadoop.mapreduce.Reducer;
```

```
1 class MyRed extends
```

```
    Reducer < INPUT KEY Class, INPUT VALUE Class, OUTPUT KEY Class, OUTPUT VALUE Class >
```

```
2 {
```

```
3 // 全域變數區
```

```
4 public void reduce ( INPUT KEY Class key, Iterable< INPUT VALUE Class > values,  
    Context context) throws IOException, InterruptedException
```

```
{
```

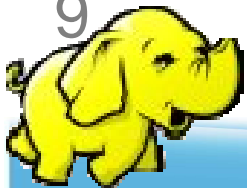
```
5 // 區域變數與程式邏輯區
```

```
6 context.write( NewKey, NewValue);
```

```
7 }
```

```
8 }
```

```
9
```



其他常用的設定參數

- 設定 Combiner

- ◆ `Job.setCombinerClass (...);`

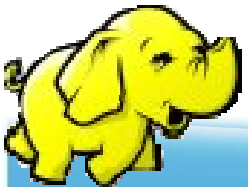
- 設定 output class

- ◆ `Job.setMapOutputKeyClass(...);`

- ◆ `Job.setMapOutputValueClass(...);`

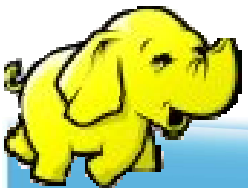
- ◆ `Job.setOutputKeyClass(...);`

- ◆ `Job.setOutputValueClass(...);`



Class Combiner

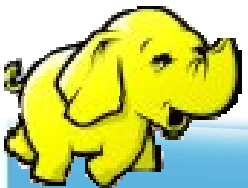
- 指定一個combiner，它負責對中間過程的輸出進行聚集，這會有助於降低從Mapper到Reducer數據傳輸量。
- 可不用設定交由Hadoop預設
- 也可不實做此程式，引用Reducer
- 設定
 - ◆ `JobConf.setCombinerClass(Class)`



範例一 (1) - mapper

```
public class HelloHadoop {  
  
    static public class HelloMapper extends  
        Mapper<LongWritable, Text, LongWritable, Text> {  
        public void map(LongWritable key, Text value, Context context)  
            throws IOException, InterruptedException {  
            context.write((LongWritable) key, (Text) value);  
        }  
    } // HelloReducer end
```

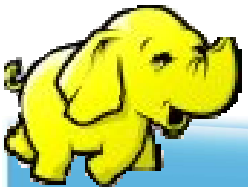
..(待續) ...



範例一 (2) - reducer

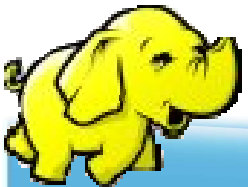
```
static public class HelloReducer extends
    Reducer<LongWritable, Text, LongWritable, Text> {
    public void reduce(LongWritable key, Iterable<Text> values,
        Context context) throws IOException, InterruptedException {
        Text val = new Text();
        for (Text str : values) {
            val.set(str.toString());
        }
        context.write(key, val);
    }
} // HelloReducer end
```

..(待續) ...



範例一 (3) - main

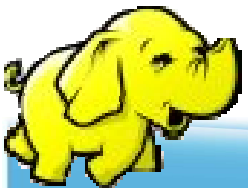
```
public static void main(String[] args) throws IOException,  
    InterruptedException, ClassNotFoundException {  
    Configuration conf = new Configuration();  
    Job job = new Job(conf, "Hadoop Hello World");  
    job.setJarByClass(HelloHadoop.class);  
    FileInputFormat.setInputPaths(job, "input");  
    FileOutputFormat.setOutputPath(job, new Path("output-hh1"));  
    job.setMapperClass(HelloMapper.class);  
    job.setReducerClass(HelloReducer.class);  
    job.waitForCompletion(true);  
    } // main end  
} // wordcount class end  
// 完
```



七、Hadoop 程式範例

7.1 : HDFS 操作篇

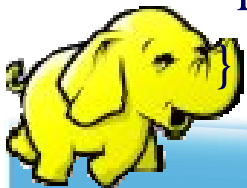
7.2 : MapReduce 運算篇



傳送檔案至HDFS

// 將檔案從local上傳到 hdfs , src 為 local 的來源
， dst 為 hdfs 的目的端

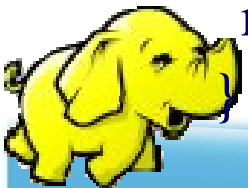
```
public class PutToHdfs {  
    static boolean putToHdfs(String src, String dst, Configuration conf) {  
        Path dstPath = new Path(dst);  
        try {  
            // 產生操作hdfs的物件  
            FileSystem hdfs = dstPath.getFileSystem(conf);  
            // 上傳  
            hdfs.copyFromLocalFile(false, new Path(src), new Path(dst));  
        } catch (IOException e) {  
            e.printStackTrace();  
            return false;  
        }  
        return true;  
    }  
}
```



從HDFS取回檔案

// 將檔案從hdfs下載回local, src 為 hdfs的來源,
dst 為 local 的目的端

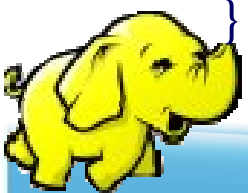
```
public class GetFromHdfs {  
    static boolean getFromHdfs(String src,String dst, Configuration conf) {  
        Path dstPath = new Path(dst);  
        try {  
            // 產生操作hdfs的物件  
            FileSystem hdfs = dstPath.getFileSystem(conf);  
            // 下載  
            hdfs.copyToLocalFile(false, new Path(src),new Path(dst));  
        } catch (IOException e) {  
            e.printStackTrace();  
            return false;  
        }  
        return true;  
    }  
}
```



檢查與刪除檔案

// checkAndDelete函式，檢查是否存在該資料夾，若有則刪除之

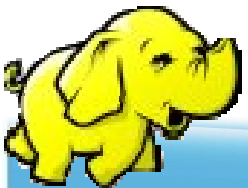
```
public class CheckAndDelete {
    static boolean checkAndDelete(final String path, Configuration conf) {
        Path dst_path = new Path(path);
        try {
            // 產生操作hdfs的物件
            FileSystem hdfs = dst_path.getFileSystem(conf);
            // 檢查是否存在
            if (hdfs.exists(dst_path)) {
                // 有則刪除
                hdfs.delete(dst_path, true);
            } } catch (IOException e) {
                e.printStackTrace();
                return false;
            }
            return true; }
}
```



七、Hadoop 程式範例

7.1 : HDFS 操作篇

7.2 : MapReduce 運算篇



範例二 (1) HelloHadoopV2

說明：

此程式碼比HelloHadoop 增加了

- * 檢查輸出資料夾是否存在並刪除
- * input 資料夾內的資料若大於兩個，則資料不會被覆蓋
- * map 與 reduce 拆開以利程式再利用

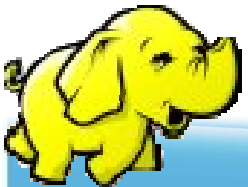
測試方法：

將此程式運作在hadoop 0.20 平台上，執行：

```
-----  
hadoop jar V2.jar HelloHadoopV2  
-----
```

注意：

1. 在hdfs 上來源檔案的路徑為 `"/user/$YOUR_NAME/input"`，請注意必須先放資料到此hdfs上的資料夾內，且此資料夾內只能放檔案，不可再放資料夾
2. 運算完後，程式將執行結果放在hdfs 的輸出路徑為 `"/user/$YOUR_NAME/output-hh2"`



範例二 (2)

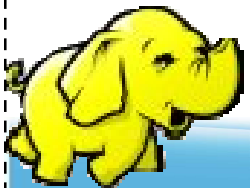
```
public class HelloHadoopV2 {
public static void main(String[] args) throws
    IOException,
    InterruptedException, ClassNotFoundException
    {
    Configuration conf = new Configuration();
    Job job = new Job(conf, "Hadoop Hello World
    2");
    job.setJarByClass(HelloHadoopV2.class);
    // 設定 map and reduce 以及 Combiner class
    job.setMapperClass(HelloMapperV2.class);
    job.setCombinerClass(HelloReducerV2.class);
    job.setReducerClass(HelloReducerV2.class);
    // 設定map的輸出型態
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(Text.class);
    // 設定reduce的輸出型態
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);
```

```
FileInputFormat.addInputPath
    (job, new Path("input"));
FileOutputFormat.setOutputPath
    (job, new Path("output-hh2"));
// 呼叫checkAndDelete函式，
// 檢查是否存在該資料夾，若有則刪除之
CheckAndDelete.checkAndDelete("output-hh2",
    conf);
boolean status = job.waitForCompletion(true);
if (status) {
    System.err.println("Integrate Alert Job Finished
    !");
} else {
    System.err.println("Integrate Alert Job Failed
    !");
    System.exit(1);
} } }
```

範例二 (3)

```
public class HelloMapperV2 extends
    Mapper <LongWritable, Text, Text,
    Text> {
    public void map(LongWritable key, Text
    value, Context context)
    throws IOException,
    InterruptedException {
    context.write(new Text(key.toString()),
    value);
    }
}
```

```
public class HelloReducerV2 extends
    Reducer<Text, Text, Text, Text> {
    public void reduce(Text key, Iterable<Text>
    values, Context context)
    throws IOException, InterruptedException {
    String str = new String("");
    Text final_key = new Text();
    Text final_value = new Text();
    // 將key值相同的values，透過 && 符號分隔
    之
    for (Text tmp : values) {
    str += tmp.toString() + " &&";
    }
    final_key.set(key);
    final_value.set(str);
    context.write(final_key, final_value);
    }
}
```



範例三 (1) HelloHadoopV3

說明：

此程式碼再利用了 HelloHadoopV2 的 map, reduce 檔，並且自動將檔案上傳到hdfs上運算並自動取回結果，還有提示訊息、參數輸入與印出運算時間的功能

測試方法：

將此程式運作在hadoop 0.20 平台上，執行：

```
-----  
hadoop jar V3.jar HelloHadoopV3 <local_input> <local_output>  
-----
```

注意：

1. 第一個輸入的參數是在local的輸入資料夾，請確認此資料夾內有資料並無子目錄
2. 第二個輸入的參數是在local的運算結果資料夾，由程式產生不用事先建立，若有請刪除之



範例三 (2)

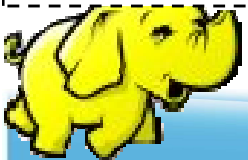
```
public class HelloHadoopV3 {
    public static void main(String[] args) throws
        IOException,
        InterruptedException, ClassNotFoundException
    {
        String hdfs_input = "HH3_input";
        String hdfs_output = "HH3_output";
        Configuration conf = new Configuration();
        // 宣告取得參數
        String[] otherArgs = new
            GenericOptionsParser(conf, args)
                .getRemainingArgs();
        // 如果參數數量不為2 則印出提示訊息
        if (otherArgs.length != 2) {
            System.err
                .println("Usage: hadoop jar
                    HelloHadoopV3.jar <local_input>
                    <local_output>");
            System.exit(2);
        }
    }
}
```

```
Job job = new Job(conf, "Hadoop Hello World");
job.setJarByClass(HelloHadoopV3.class);
// 再利用上個範例的map 與 reduce
job.setMapperClass(HelloMapperV2.class);
job.setCombinerClass(HelloReducerV2.class);
job.setReducerClass(HelloReducerV2.class);
// 設定map reduce 的key value輸出型態
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(Text.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(Text.class);
```

範例三 (2)

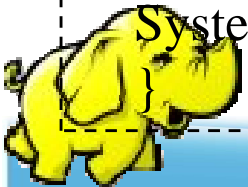
```
// 用 checkAndDelete 函式防止overhead的錯誤
CheckAndDelete.checkAndDelete(hdfs_input,
    conf);
CheckAndDelete.checkAndDelete(hdfs_output,
    conf);
// 放檔案到hdfs
PutToHdfs.putToHdfs(args[0], hdfs_input, conf);
// 設定hdfs 的輸入輸出來源路定
FileInputFormat.addInputPath(job, new
    Path(hdfs_input));
FileOutputFormat.setOutputPath(job, new
    Path(hdfs_output));
long start = System.nanoTime();
job.waitForCompletion(true);
```

```
// 把hdfs的結果取下
GetFromHdfs.getFromHdfs(hdfs_output, args[1],
    conf);
boolean status = job.waitForCompletion(true);
// 計算時間
if (status) {
    System.err.println("Integrate Alert Job Finished
        !");
    long time = System.nanoTime() - start;
    System.err.println(time * (1E-9) + " secs.");
} else {
    System.err.println("Integrate Alert Job Failed !");
    System.exit(1);
} } }
```



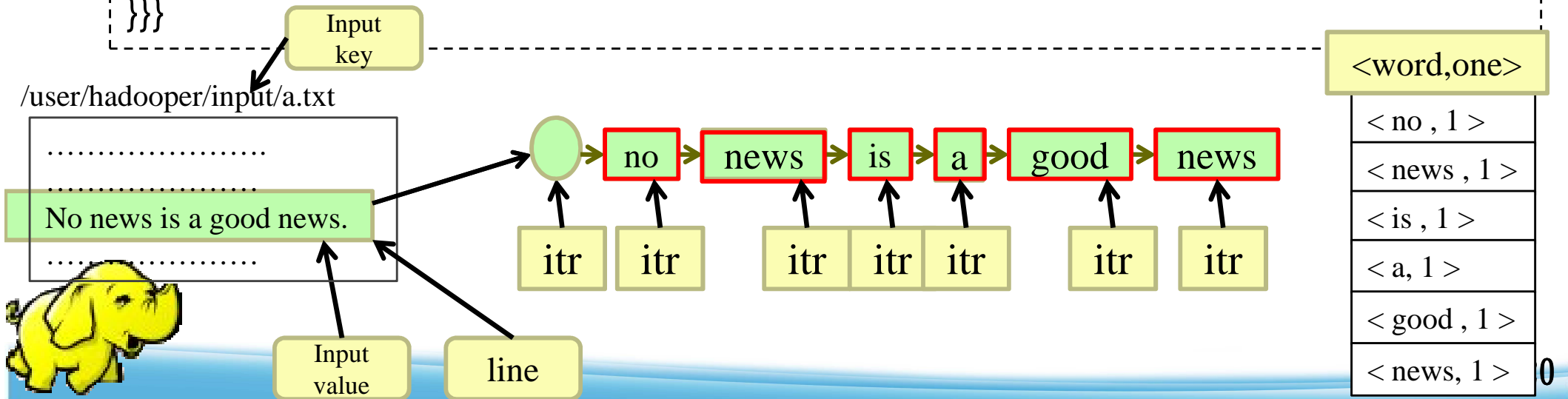
範例四 (1)

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length != 2) {
        System.err.println("Usage: hadoop jar WordCount.jar <input> <output>");
        System.exit(2);
    }
    Job job = new Job(conf, "Word Count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    CheckAndDelete.checkAndDelete(args[1], conf);
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```



範例四 (2)

```
1 class TokenizerMapper extends Mapper<LongWritable, Text, Text,  
2 IntWritable> {  
3     private final static IntWritable one = new IntWritable(1);  
4     private Text word = new Text();  
5     public void map( LongWritable key, Text value, Context context )  
6         throws IOException , InterruptedException {  
7         String line = ((Text) value).toString();  
8         String tokenizer itr = new String tokenizer(line);  
9         while (itr.hasMoreTokens()) {  
            word.set(itr.nextToken());  
            context.write(word, one);  
        }  
    }  
}
```



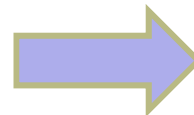
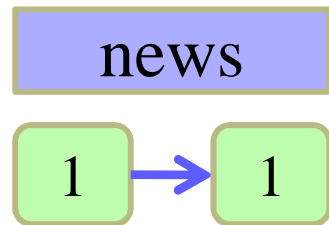
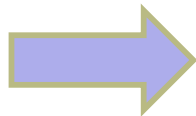
範例四 (3)

```
1 class IntSumReducer extends Reducer< Text, IntWritable, Text, IntWritable> {  
2     IntWritable result = new IntWritable();  
3     public void reduce( Text key, Iterable <IntWritable> values, Context context)  
4         throws IOException, InterruptedException {  
5         int sum = 0;  
6         for ( IntWritable val : values )  
7             sum += val.get();  
8         result.set(sum);  
9         context.write ( key, result);  
10    }  
11 }
```

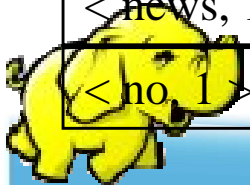
```
for ( int i ; i < values.length ; i ++ ){  
    sum += values[i].get()  
}
```



<word,one>
< a, 1 >
< good, 1 >
< is, 1 >
< news, 1 → 1 >
< no, 1 >



<key,SunValue>
< news , 2 >



範例五 (1) WordCountV2

說明：

用於字數統計，並且增加略過大小寫辨識、符號篩除等功能

測試方法：

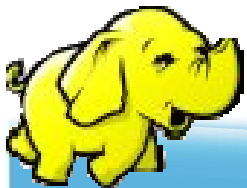
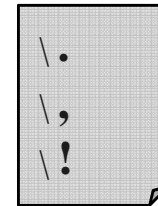
將此程式運作在hadoop 0.20 平台上，執行：

```
-----  
hadoop jar WCV2.jar WordCountV2 -Dwordcount.case.sensitive=false \  
<input> <output> -skip patterns/patterns.txt  
-----
```

注意：

1. 在hdfs 上來源檔案的路徑為 你所指定的 <input>
請注意必須先放資料到此hdfs上的資料夾內，且此資料夾內只能放檔案，不可再放資料夾
2. 運算完後，程式將執行結果放在hdfs 的輸出路徑為 你所指定的 <output>
3. 請建立一個資料夾 pattern 並在裡面放置pattern.txt，內容如

(一行一個，前置提示符號\) →



範例五 (2)

```
public class WordCountV2 extends Configured implements Tool {
    public static class Map extends MapReduceBase implements
        Mapper<LongWritable, Text, Text, IntWritable> {
        static enum Counters { INPUT_WORDS }
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();
        private boolean caseSensitive = true;
        private Set<String> patternsToSkip = new HashSet<String>();
        private long numRecords = 0;
        private String inputFile;
        public void configure(JobConf job) {
            caseSensitive = job.getBoolean("wordcount.case.sensitive", true);
            inputFile = job.get("map.input.file");
            if (job.getBoolean("wordcount.skip.patterns", false)) {
                Path[] patternsFiles = new Path[0];
                try {
                    patternsFiles = DistributedCache.getLocalCacheFiles(job);
                } catch (IOException ioe) {
                    System.err
                        .println("Caught exception while getting cached files: "
                            + StringUtils.stringifyException(ioe));
                }
                for (Path patternsFile : patternsFiles) {
                    parseSkipFile(patternsFile);
                }
            }
        }
    }
}
```

```
private void parseSkipFile(Path patternsFile) {
    try {
        BufferedReader fis = new BufferedReader(new FileReader(
            patternsFile.toString()));
        String pattern = null;
        while ((pattern = fis.readLine()) != null) {
            patternsToSkip.add(pattern);
        }
    } catch (IOException ioe) {
        System.err.println("Caught exception while parsing the cached
            file '" + patternsFile + "' : " + tringUtils.stringifyException(ioe));
    }
}

public void map(LongWritable key, Text value,
    OutputCollector<Text, IntWritable> output, Reporter reporter)
    throws IOException {
    String line = (caseSensitive) ? value.toString() : value.toString()
        .toLowerCase();
    for (String pattern : patternsToSkip)
        line = line.replaceAll(pattern, "");
    StringTokenizer tokenizer = new StringTokenizer(line);
    while (tokenizer.hasMoreTokens()) {
        word.set(tokenizer.nextToken());
        output.collect(word, one);
        reporter.incrCounter(Counters.INPUT_WORDS, 1);
    }
}
```

範例五 (3)

```
if ((++numRecords % 100) == 0) {
    reporter.setStatus("Finished processing " + numRecords
        + " records " + "from the input file: " + inputFile);
} } }
public static class Reduce extends MapReduceBase implements
    Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterator<IntWritable> values,
        OutputCollector<Text, IntWritable> output, Reporter reporter)
        throws IOException {
        int sum = 0;
        while (values.hasNext()) {
            sum += values.next().get(); }
        output.collect(key, new IntWritable(sum)); } }
    public int run(String[] args) throws Exception {
        JobConf conf = new JobConf(getConf(), WordCount.class);
        conf.setJobName("wordcount");
        String[] otherArgs = new GenericOptionsParser(conf, args)
            .getRemainingArgs();
        if (otherArgs.length < 2) {
            System.out.println("WordCountV2 [-
                Dwordcount.case.sensitive=<false|true>] \\ ");
            System.out.println("    <inDir> <outDir> [-skip
                Pattern_file]");
            return 0; }
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);
```

```
        conf.setMapperClass(Map.class);
        conf.setCombinerClass(Reduce.class);
        conf.setReducerClass(Reduce.class);
        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);
        List<String> other_args = new ArrayList<String>();
        for (int i = 0; i < args.length; ++i) {
            if ("-skip".equals(args[i])) {
                DistributedCache
                    .addCacheFile(new Path(args[++i]).toUri(), conf);
                conf.setBoolean("wordcount.skip.patterns", true);
            } else {other_args.add(args[i]); } }
        FileInputFormat.setInputPaths(conf, new Path(other_args.get(0)));
        FileOutputFormat.setOutputPath(conf, new
            Path(other_args.get(1)));
        CheckAndDelete.checkAndDelete(other_args.get(1), conf);
        JobClient.runJob(conf);
        return 0; }
    public static void main(String[] args) throws Exception {
        int res = ToolRunner.run(new Configuration(), new WordCountV2(),
            args);
        System.exit(res);
    } }
```

範例六 (1) WordIndex

說明：

將每個字出於哪個檔案，那一行印出來

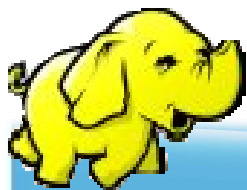
測試方法：

將此程式運作在hadoop 0.20 平台上，執行：

```
-----  
hadoop jar WI.jar WordIndex <input> <output>  
-----
```

注意：

1. 在hdfs 上來源檔案的路徑為 你所指定的 <input>
請注意必須先放資料到此hdfs上的資料夾內，且此資料夾內只能放檔案，
不可再放資料夾
2. 運算完後，程式將執行結果放在hdfs 的輸出路徑為 你所指定的
<output>



範例六 (2)

```
public class WordIndex {
    public static class wordindexM extends
        Mapper<LongWritable, Text, Text, Text> {
    public void map(LongWritable key, Text value,
        Context context)
        throws IOException, InterruptedException {

        FileSplit fileSplit = (FileSplit)
            context.getInputSplit();

        Text map_key = new Text();
        Text map_value = new Text();
        String line = value.toString();
        StringTokenizer st = new
            StringTokenizer(line.toLowerCase());
        while (st.hasMoreTokens()) {
            String word = st.nextToken();
            map_key.set(word);
            map_value.set(fileSplit.getPath().getName() +
                ":" + line);
            context.write(map_key, map_value);
        } } }
```

```
static public class wordindexR extends
    Reducer<Text, Text, Text, Text> {
    public void reduce(Text key, Iterable<Text>
        values,
        OutputCollector<Text, Text> output, Reporter
            reporter)
        throws IOException {
        String v = "";
        StringBuilder ret = new StringBuilder("\n");
        for (Text val : values) {
            v += val.toString().trim();
            if (v.length() > 0)
                ret.append(v + "\n");
        }
        output.collect((Text) key, new
            Text(ret.toString()));
    } }
```


範例六 (2)

```
public static void main(String[] args) throws
    IOException,
    InterruptedException,
    ClassNotFoundException {
    Configuration conf = new Configuration();
    String[] otherArgs = new
        GenericOptionsParser(conf, args)
            .getRemainingArgs();
    if (otherArgs.length < 2) {
        System.out.println("hadoop jar
            WordIndex.jar <inDir> <outDir>");
        return;
    }
    Job job = new Job(conf, "word index");
    job.setJobName("word inverted index");
    job.setJarByClass(WordIndex.class);
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(Text.class);
```

```
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);
    job.setMapperClass(wordindexM.class);
    job.setReducerClass(wordindexR.class);
    job.setCombinerClass(wordindexR.class);
    FileInputFormat.setInputPaths(job, args[0]);
    CheckAndDelete.checkAndDelete(args[1],
        conf);
    FileOutputFormat.setOutputPath(job, new
        Path(args[1]));
    long start = System.nanoTime();
    job.waitForCompletion(true);
    long time = System.nanoTime() - start;
    System.err.println(time * (1E-9) + " secs.");
}}
```


範例七 (1) YourMenu

說明：

將之前的功能整合起來

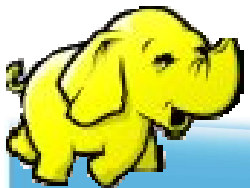
測試方法：

將此程式運作在hadoop 0.20 平台上，執行：

```
-----  
hadoop jar YourMenu.jar <功能>  
-----
```

注意：

1. 此程式需與之前的所有範例一起打包成一個jar檔



範例七 (2)

```
public class YourMenu {
    public static void main(String argv[]) {
        int exitCode = -1;
        ProgramDriver pgd = new ProgramDriver();
        if (argv.length < 1) {

            System.out.print("*****\n");
            + "歡迎使用 NCHC 的運算功能\n" + "指令：\n"
            + " Hadoop jar NCHC-example-*.jar <功能> \n" + "功能：\n"
            + " HelloHadoop: 秀出Hadoop的<Key,Value>為何\n"
            + " HelloHadoopV2: 秀出Hadoop的<Key,Value> 進階版\n"
            + " HelloHadoopV3: 秀出Hadoop的<Key,Value> 進化版\n"
            + " WordCount: 計算輸入資料夾內分別在每個檔案的
            字數統計\n"
            + " WordCountV2: WordCount 進階版\n"
            + " WordIndex: 索引每個字與其所有出現的所在列\n"
            + "*****\n");
        } else {
            try {
                pgd.addClass("HelloHadoop", HelloHadoop.class, "
                Hadoop hello world");
                pgd.addClass("HelloHadoopV2",
                HelloHadoopV2.class, " Hadoop hello world V2");
                pgd.addClass("HelloHadoopV3",
                HelloHadoopV3.class, " Hadoop hello world V3");
                pgd.addClass("WordCount", WordCount.class, "
                word count.");
                pgd.addClass("WordCountV2",
                WordCountV2.class, " word count V2.");
                pgd.addClass("WordIndex", WordIndex.class,
                "invert each word in line");
                pgd.driver(argv);
                // Success
                exitCode = 0;
                System.exit(exitCode);
            } catch (Throwable e) {
                e.printStackTrace();
            }
        }
    }
}
```

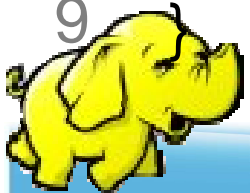
rogram Prototype (v 0.18)



Class Mapper (v0.18)

```
import org.apache.hadoop.mapred.*;

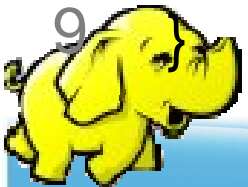
1 class MyMap extends MapReduceBase
  implements Mapper < INPUT KEY , INPUT VALUE , OUTPUT KEY , OUTPUT VALUE >
2 {
3   // 全域變數區
4   public void map ( INPUT KEY key, INPUT VALUE value,
      OutputCollector< OUTPUT KEY , OUTPUT VALUE > output,
      Reporter reporter) throws IOException
5   {
6     // 區域變數與程式邏輯區
7     output.collect( NewKey, NewValue);
8   }
9
```



Class Reducer (v0.18)

```
import org.apache.hadoop.mapred.*;

1 class MyRed extends MapReduceBase
  implements Reducer < INPUT KEY , INPUT VALUE , OUTPUT KEY , OUTPUT VALUE >
2 {
3   // 全域變數區
4   public void reduce ( INPUT KEY key, Iterator< INPUT VALUE > values,
      OutputCollector< OUTPUT KEY , OUTPUT VALUE > output,
      Reporter reporter) throws IOException
5   {
6     // 區域變數與程式邏輯區
7     output.collect( NewKey, NewValue);
8   }
9 }
```



Conclusions

- 以上範例程式碼包含
 - ◆ Hadoop 的key,value 架構
 - ◆ 操作Hdfs 檔案系統
 - ◆ Map Reduce運算方式
- 執行hadoop 運算時，程式檔不用上傳至hadoop 上，但資料需要再HDFS內
- 可運用範例七的程式達成連續運算
- Hadoop 0.20 與 Hadoop 0.18 有些API有些許差異，因此在網路上找到Hadoop的程式如果 compiler有錯，可以換換對應的Function試試

