第 18 章 捨遺補缺

這一個章節的目的在於捨遺補缺，對於之前章節中有提到但沒說明的 Date 與日誌（Log）的 API，在這個章節中會加以說明，另外也將介紹訊息資源綁定，這讓您可以隨時調整文字訊息而不用重新編譯程式。

18.1 日期、時間

表面上看來，要取得系統的時間只要使用 java.util.Date 類別就可以了，但查閱 Date 後，發現很多方法都被標示為 "deprecated"？這是怎麼回事？事實上有些方法建議您改用 java.util.Calendar 上的一些對應方法了。

18.1.1 使用 Date

如果想要取得系統的時間，可以使用 System.currentTimeMillis() 方法，範例 18.1 可以取得系統目前的時間。

範例 18.1 CurrentTime.java

```java
package onlyfun.caterpillar;

public class CurrentTime {
    public static void main(String[] args) {
        System.out.println("現在時間 "
               + System.currentTimeMillis());
    }
}
```

執行結果：

```
現在時間 1118066309062
```

執行結果是一長串數字，這數字實際上表示從 1970 年 1 月 1 日 0 時 0 分 0 秒開始，到程式執行取得系統時間為止所經過的毫秒數，但這樣的數字讓人來看實在不太有意義，您可以使用 java.util.Date 類別來讓這個數字變的更有意義一些，範例 18.2 示範如何使用 Date 取得系統時間。

範例 18.2 DateDemo.java

```java
package onlyfun.caterpillar;

import java.util.Date;

public class DateDemo {
    public static void main(String[] args) {
        Date date = new Date();

        System.out.println("現在時間 "
                      + date.toString());
        System.out.println("自1970/1/1至今的毫秒數 "
                      + date.getTime());
    }
}
```

執行結果：

```
現在時間 Mon Jun 06 22:03:52 GMT+08:00 2005
自1970/1/1至今的毫秒數 1118066632890
```

當您生成 Date 物件時，會使用 System.currentTimeMillis() 來取得系統時間，而您使用 Date 實例的 toString() 方法時，會將取得的 1970 年 1 月 1 日至今的毫秒數轉為「星期 月 日 時：分：秒 西元」的格式；使用 Date 的 getTime() 方法則可以取得實際經過的毫秒數。

如果您想要對日期時間作格式設定，則可以使用 java.text.DateFormat 來作格式化，先來看看 DateFormat 的子類別 java.text.SimpleDateFormat 是如何使用的，範例 18.3 是個簡單示範。

範例 18.3 DateFormatDemo.java

```java
package onlyfun.caterpillar;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;

public class DateFormatDemo {
    public static void main(String[] args) {
        Date date = new Date();

        DateFormat dateFormat =
            new SimpleDateFormat("EE-MM-dd-yyyy");

        System.out.println(dateFormat.format(date));
    }
}
```

執行結果：

```
星期一-06-06-2005
```

DateFormat 會依電腦上的區域（Locale）設定顯示時間格式，"EE" 表示星期格式設定，"MM" 表示月份格式設定、"dd" 表示日期格式設定，而 "yyyy" 是西元格式設定，每個字元的設定都各有其意義，您可以參考 java.util.SimpleDateFormat 的 API 說明瞭解每個字元設定的意義。

您也可以直接使用 DateFormat 上的靜態 getDateTimeInstance() 方法來指定格式，getDateTimeInstance() 方法會傳回 DateFormat 的實例，之後可以重複使用這個實例來格式化 Date 物件，如範例 18.4 所示範的。

範例 18.4 DateTimeInstanceDemo.java

```java
package onlyfun.caterpillar;

import java.text.DateFormat;
import java.util.Date;

public class DateTimeInstanceDemo {
    public static void main(String[] args) {
        // 取得目前時間
        Date date = new Date();

        // 簡短資訊格式
        DateFormat shortFormat =
            DateFormat.getDateTimeInstance(
                DateFormat.SHORT, DateFormat.SHORT);
        // 中等資訊格式
        DateFormat mediumFormat =
            DateFormat.getDateTimeInstance(
                DateFormat.MEDIUM, DateFormat.MEDIUM);
        // 長資訊格式
        DateFormat longFormat =
            DateFormat.getDateTimeInstance(
```

```
                DateFormat.LONG, DateFormat.LONG);
        // 詳細資訊格式
        DateFormat fullFormat =
            DateFormat.getDateTimeInstance(
                DateFormat.FULL, DateFormat.FULL);

        System.out.println("簡短資訊格式：" +
                        shortFormat.format(date));
        System.out.println("中等資訊格式：" +
                        mediumFormat.format(date));
        System.out.println("長資訊格式：" +
                        longFormat.format(date));
        System.out.println("詳細資訊格式：" +
                        fullFormat.format(date));
    }
}
```

在使用 getDateTimeInstance() 取得 DateFormat 實例時，可以指定的引數是日期格式與時間格式，以上所指定的格式依訊息的詳細程度區分，執行結果如下：

```
簡短資訊格式：2005/6/6 下午 10:19
中等資訊格式：2005/6/6 下午 10:19:13
長資訊格式：2005年6月6日 下午10時19分13秒
詳細資訊格式：2005年6月6日 星期一 下午10時19分13秒 GMT+08:00
```

您也可以使用 getDateInstance() 取得 DateFormat 實例，並同時指定日期的區域顯示方式，指定時要使用一個 java.util.Locale（18.3.3 會介紹）實例作為引數，這邊改寫範例 18.4 為範例 18.5，讓程式改以美國的時間表示方式來顯示。

範例 18.5 DateTimeInstanceDemo2.java

```java
package onlyfun.caterpillar;

import java.text.DateFormat;
import java.util.Date;
import java.util.Locale;

public class DateTimeInstanceDemo2 {
    public static void main(String[] args) {
        // 取得目前時間
        Date date = new Date();

        // en: 英語系 US: 美國
        Locale locale = new Locale("en", "US");

        // 簡短資訊格式
        DateFormat shortFormat =
           DateFormat.getDateTimeInstance(
              DateFormat.SHORT, DateFormat.SHORT, locale);
        // 中等資訊格式
        DateFormat mediumFormat =
           DateFormat.getDateTimeInstance(
              DateFormat.MEDIUM, DateFormat.MEDIUM, locale);
        // 長資訊格式
        DateFormat longFormat =
           DateFormat.getDateTimeInstance(
              DateFormat.LONG, DateFormat.LONG, locale);
        // 詳細資訊格式
        DateFormat fullFormat =
           DateFormat.getDateTimeInstance(
                DateFormat.FULL, DateFormat.FULL, locale);

        System.out.println("short format: " +
```

```
                    shortFormat.format(date));
        System.out.println("media format: " +
                    mediumFormat.format(date));
        System.out.println("long format: " +
                    longFormat.format(date));
        System.out.println("full format: " +
                    fullFormat.format(date));
    }
}
```

```

執行結果：

```
short format: 6/6/05 10:29 PM
media format: Jun 6, 2005 10:29:30 PM
long format: June 6, 2005 10:29:30 PM GMT+08:00
full format: Monday, June 6, 2005 10:29:30 PM GMT+08:00
```

### 18.1.2 使用 Calendar

您可以使用 Date 來取得完整的日期時間並使用 toString() 顯示日期的文字描述，但如果您想要單獨取得某個時間或日期資訊的話該如何進行？例如您想知道現在的時間是 6 月的第幾天？

您要使用 java.util.Calendar 類別，但在使用這個類別上的一些方法之前，您要先知道 Calendar 的一些方法會取回 int 型態數字，而取回的數字其實是對應於 Calendar 中定義的常數，所以並不是您使用某個方法取得 1 這個數字，就會表示取得的時間是星期一。

想要取得現在的時間，首先要使用 Calendar 的 getInstance() 方法取得一個 Calendar 的實例，例如：

```
Calendar rightNow = Calendar.getInstance();
```

一個 Calendar 的實例是系統時間的抽象表示，您可以操作這個實例上的 get() 方法來取得您想要知道的時間資訊，例如您想知道現在是西元幾年，則可以使用 get() 方法並指定對應的常數，例如：

```
System.out.println(rightNow.get(Calendar.YEAR));
```

如果現在是2005年，則上例會顯示 2005 的數字，依照這個例子，假設現在的時間是 5 月份，而您現在想使用程式取得現在的月份，則下例您可能會有些困惑：

```
System.out.println(rightNow.get(Calendar.MONTH));
```

上面的程式片段會顯示 4 這個數字，而不是您預期的 5，事實上傳回的 4 並不是代表目前時間是 4 月份，而是對應於 Calendar.MAY 常數的值，Calendar 在月份上的常數值從一月 Calendar.JANUARY 開始是 0，到十二月 Calendar.DECEMBER 的 11，所以您如果想要顯示傳回值的真正意涵，可以如下撰寫：

```
String[] months =   {"一月", "二月", "三月", "四月",
                     "五月", "六月", "七月", "八月",
                     "九月", "十月", "十一月", "十二月"};
Calendar rightNow = Calendar.getInstance();
int monthConstant = rightNow.get(Calendar.MONTH);
System.out.println(months[monthConstant]);
```

同樣的，如果您想要取得星期資訊，要記得 Calendar 的星期常數從星期日 Calendar.SUNDAY 是 1，到星期六 Calendar.SATURDAY 是 7，由於對應的數並不是從0開始，所以如果要使用陣列來對應的話，第一個陣列的元素值就不包括資訊，例如：

```
String[] dayOfWeek = {"", "日", "一", "二",
                          "三", "四", "五", "六"};
Calendar rightNow = Calendar.getInstance();
int dayOfWeekConstant = rightNow.get(Calendar.DAY_OF_WEEK);
System.out.println(dayOfWeek[dayOfWeekConstant]);
```

總之您要記得 get() 傳回的 int 值是對應於 Calendar 的某個常數，如果怕搞混，使用 switch 來比對是個不錯的作法，範例 18.6 是一個實際應用的參
```

考，可以顯示中文的星期與月份。

範例 18.6 CalendarDemo.java

```java
package onlyfun.caterpillar;

import java.util.Calendar;

public class CalendarDemo {
    public static void main(String[] args) {
        Calendar rightNow = Calendar.getInstance();

        System.out.println("現在時間是：");
        System.out.println("西元：" +
                   rightNow.get(Calendar.YEAR));
        System.out.println("月：" +
                   getChineseMonth(rightNow));
        System.out.println("日：" +
                   rightNow.get(Calendar.DAY_OF_MONTH));
        System.out.println("星期：" +
                   getChineseDayOfWeek(rightNow));
    }

    public static String getChineseMonth(Calendar rightNow) {
        String chineseMonth = null;

        switch(rightNow.get(Calendar.MONTH)) {
            case Calendar.JANUARY:
                chineseMonth = "一";
                break;
            case Calendar.FEBRUARY:
                chineseMonth = "二";
                break;
            case Calendar.MARCH:
                chineseMonth = "三";
                break;
            case Calendar.APRIL:
                chineseMonth = "四";
                break;
            case Calendar.MAY:
                chineseMonth = "五";
                break;
            case Calendar.JUNE:
                chineseMonth = "六";
                break;
            case Calendar.JULY:
                chineseMonth = "七";
                break;
            case Calendar.AUGUST:
                chineseMonth = "八";
                break;
            case Calendar.SEPTEMBER:
                chineseMonth = "九";
                break;
            case Calendar.OCTOBER:
                chineseMonth = "十";
                break;
            case Calendar.NOVEMBER:
                chineseMonth = "十一";
                break;
            case Calendar.DECEMBER:
                chineseMonth = "十二";
                break;
        }
```

```
        return chineseMonth;
    }

    public static String getChineseDayOfWeek(
                                Calendar rightNow) {
        String chineseDayOfWeek = null;

        switch(rightNow.get(Calendar.DAY_OF_WEEK)) {
            case Calendar.SUNDAY:
                chineseDayOfWeek = "日";
                break;
            case Calendar.MONDAY:
                chineseDayOfWeek = "一";
                break;
            case Calendar.TUESDAY:
                chineseDayOfWeek = "二";
                break;
            case Calendar.WEDNESDAY:
                chineseDayOfWeek = "三";
                break;
            case Calendar.THURSDAY:
                chineseDayOfWeek = "四";
                break;
            case Calendar.FRIDAY:
                chineseDayOfWeek = "五";
                break;
            case Calendar.SATURDAY:
                chineseDayOfWeek = "六";
                break;
        }

        return chineseDayOfWeek;
    }
}
```

```

執行結果：

```
現在時間是：
西元：2005
月：六
日：6
星期：一
```

事實上在 Java SE 6 中，對於 Calendar 新增了顯示名稱（Display Names），可以讓您直接根據 Locale 物件，顯示相對應的區域化訊息，詳細的介紹請目第 21 章關於 Java SE 6 新功能的介紹。

# 18.2 日誌（Logging）

如果您只是要作一些簡單的文件日誌（Log），可以考慮內建在 JDK 中的日誌 API，功能或許簡單了一些，但好處是它從 J2SE 1.4 之後就成為 Java SE 標準 API 的一員。

## 18.2.1 簡介日誌

程式中不免會出現錯誤，當錯誤發生時，您可以使用 System.err.println() 或是 System.out.println() 在文字模式（Console）中顯示訊息給使用者，如果是在視窗程式中，可能是使用訊息方塊，如果是在網頁程式中，則顯示一個錯誤訊息頁面，除了提供錯誤訊息之外，您可能還想將錯誤訊息以某種方式儲存下來，以供使用者或是程式人員除蟲（Debug）時使用。

在 Java SE 中的 java.util.logging 套件提供了一系列的日誌工具類別，如果您只是要簡單的記錄一些日誌訊息，就可以使用這些工具類別，它們在 J2SE 1.4 之後加入了標準 API 中，不必額外配置日誌元件就可以運行於標準的 Java 平台上是它的好處，無論是在文字模式簡單的顯示訊息，或是在檔案中輸出日誌，java.util.logging 下都提供了一些預設的工具類別。

要使用 Java SE 的日誌功能，首先要取得 java.util.logging.Logger 實例，這可以透過 Logger 類別的靜態 getLogger() 方法來取得，看看範例 18.7 的
```

例子，假設您的程式在啟動時必須提供引數給程式，如果使用者沒有提供引數則必須顯示警示訊息。

範例 18.7 LoggingDemo.java

```java
package onlyfun.caterpillar;

import java.util.logging.*;

public class LoggingDemo {
    public static void main(String[] args) {
        Logger logger = Logger.getLogger("LoggingDemo");

        try {
            System.out.println(args[0]);
        }
        catch(ArrayIndexOutOfBoundsException e) {
            logger.warning("沒有提供執行時的引數！");
        }
    }
}
```

執行結果：

```
2005/6/6 下午 11:52:19 onlyfun.caterpillar.LoggingDemo main
警告: 沒有提供執行時的引數！
```

您簡單的透過 Logger 的靜態方法 getLogger() 取得 Logger 物件，給 getLogger() 的字串引數被用來作為 Logger 物件的名稱，之後就可以運用 Logger 實例上的方法進行訊息輸出，預設是從文字模式輸出訊息，除了您提供的文字訊息之外，Logger 還自動幫您收集了執行時相關的訊息，像是訊息輸出所在的類別與套件名稱，執行該段程式時的執行緒名稱以及訊息產生的時間等，這比下面這個程式所提供的訊息豐富的多了：

```java
package onlyfun.caterpillar;
public class LoggingDemo {
    public static void main(String[] args) {
        try {
            System.out.println(args[0]);
        }
        catch(ArrayIndexOutOfBoundsException e) {
            System.err.println("沒有提供執行時的引數！");
        }
    }
}
```

所以您不用親自實作日誌功能的程式，只要直接使用 java.util.logging 工具，就可以擁有一些預設的日誌功能，對於中大型系統來說，java.util.logging 套件下所提供的工具在功能上可能不足，但作為一個簡單的日誌工具，java.util.logging 套件下所提供的工具是可以考慮的方案。

18.2.2 日誌的等級

在進行訊息的日誌記錄時，依訊息程度的不同，您會設定不同等級的訊息輸出，您可以透過操作 Logger 上的幾個方法來得到不同等級的訊息輸出，範例 18.8 示範了各種等級的訊息輸出。

範例 18.8 LoggingLevelDemo.java

```java
package onlyfun.caterpillar;
```

```
import java.util.logging.*;

public class LoggingLevelDemo {
    public static void main(String[] args) {
        Logger logger = Logger.getLogger("loggingLevelDemo");

        logger.severe("嚴重訊息");
        logger.warning("警示訊息");
        logger.info("一般訊息");
        logger.config("設定方面的訊息");
        logger.fine("細微的訊息");
        logger.finer("更細微的訊息");
        logger.finest("最細微的訊息");
    }
}
```

```

執行結果：

```
2005/6/7 上午 09:10:39 onlyfun.caterpillar.LoggingLevelDemo main
嚴重的: 嚴重訊息
2005/6/7 上午 09:10:39 onlyfun.caterpillar.LoggingLevelDemo main
警告: 警示訊息
2005/6/7 上午 09:10:39 onlyfun.caterpillar.LoggingLevelDemo main
資訊: 一般訊息
```

依照不同的等級，您可以採用不同的訊息等級方法，程式中依程度高低撰寫，注意到 config() 方法及以下的訊息並沒有顯示出來，這是因為 Logger 的預設等級是 INFO，比這個等級更低的訊息，Logger 並不會將訊息輸出。

Logger 的預設等級是定義在執行環境的屬性檔 logging.properties 中，這個檔案位於 JRE 安裝目錄的 lib 目錄下，部份內容如下：

```
handlers= java.util.logging.ConsoleHandler
.level= INFO
java.util.logging.ConsoleHandler.level = INFO
```

Logger 預設的處理者（Handler）是 java.util.logging.ConsolerHandler，也就是將訊息輸出至文字模式，一個 Logger 可以擁有多個處理者，每個處理者可以有自己的訊息等級，在通過 Logger 的等級限制後，實際上還要再經過處理者的等級限制，所以在範例 18.8 中如果想要看到所有的訊息，則必須同時設定 Logger 與 ConsolerHandler 的等級，範例 18.9 示範了如何設定。

範例 18.9 LoggingLevelDemo2.java

```java

```
package onlyfun.caterpillar;

import java.util.logging.*;

public class LoggingLevelDemo2 {
    public static void main(String[] args) {
        Logger logger = Logger.getLogger("loggingLevelDemo2");
        // 顯示所有等級的訊息
        logger.setLevel(Level.ALL);

        ConsoleHandler consoleHandler = new ConsoleHandler();
        // 顯示所有等級的訊息
        consoleHandler.setLevel(Level.ALL);
        // 設定處理者為ConsoleHandler
        logger.addHandler(consoleHandler);

        logger.severe("嚴重訊息");
        logger.warning("警示訊息");
        logger.info("一般訊息");
        logger.config("設定方面的訊息");
        logger.fine("細微的訊息");
```

```
        logger.finer("更細微的訊息");
        logger.finest("最細微的訊息");
    }
}
```

```

Level.ALL 表示顯示所有的訊息，所以這一次的執行結果可顯示所有等級的訊息，如下所示：

```
2005/6/7 上午 09:17:37 onlyfun.caterpillar.LoggingLevelDemo2 main
嚴重的: 嚴重訊息
2005/6/7 上午 09:17:37 onlyfun.caterpillar.LoggingLevelDemo2 main
嚴重的: 嚴重訊息
2005/6/7 上午 09:17:37 onlyfun.caterpillar.LoggingLevelDemo2 main
警告: 警示訊息
2005/6/7 上午 09:17:37 onlyfun.caterpillar.LoggingLevelDemo2 main
警告: 警示訊息
2005/6/7 上午 09:17:37 onlyfun.caterpillar.LoggingLevelDemo2 main
資訊: 一般訊息
2005/6/7 上午 09:17:37 onlyfun.caterpillar.LoggingLevelDemo2 main
資訊: 一般訊息
2005/6/7 上午 09:17:37 onlyfun.caterpillar.LoggingLevelDemo2 main
配置: 設定方面的訊息
2005/6/7 上午 09:17:37 onlyfun.caterpillar.LoggingLevelDemo2 main
細緻: 細微的訊息
2005/6/7 上午 09:17:37 onlyfun.caterpillar.LoggingLevelDemo2 main
更細緻: 更細微的訊息
2005/6/7 上午 09:17:37 onlyfun.caterpillar.LoggingLevelDemo2 main
最細緻: 最細微的訊息
```

如果您想要關閉所有的訊息，可以設定為 Level.OFF。

Logger 的 server()、warning()、info() 等方法，實際上是個便捷的方法，您也可以直接使用 log() 方法並指定等級來執行相同的作用，例如範例 18.10 的執行結果與範例 18.9 是一樣的。

### 範例 18.10 LoggingLevelDemo3.java

```java

```
package onlyfun.caterpillar;

import java.util.logging.*;

public class LoggingLevelDemo3 {
    public static void main(String[] args) {
        Logger logger = Logger.getLogger("loggingLevelDemo3");
        logger.setLevel(Level.ALL);

        ConsoleHandler consoleHandler = new ConsoleHandler();
        consoleHandler.setLevel(Level.ALL);
        logger.addHandler(consoleHandler);

        logger.log(Level.SEVERE, "嚴重訊息");
        logger.log(Level.WARNING, "警示訊息");
        logger.log(Level.INFO, "一般訊息");
        logger.log(Level.CONFIG, "設定方面的訊息");
        logger.log(Level.FINE, "細微的訊息");
        logger.log(Level.FINER, "更細微的訊息");
        logger.log(Level.FINEST, "最細微的訊息");
    }
}
```

```

## 18.2.3 Handler、Formatter
```

Logger 預設的輸出處理者（Handler）是 ConsolerHandler，ConsolerHandler 的輸出是使用 System.err 物件，而訊息的預設等級是 INFO，這可以在 JRE 安裝目錄下 lib 目錄的 logging.properties 中看到：

```
handlers= java.util.logging.ConsoleHandler
java.util.logging.ConsoleHandler.level = INFO
```

Java SE 提供了五個預設的 Handler：

```
java.util.logging.ConsoleHandler
```

以System.err輸出日誌。

```
java.util.logging.FileHandler
```

將訊息輸出至檔案。

```
java.util.logging.StreamHandler
```

以指定的OutputStream實例輸出日誌。

```
java.util.logging.SocketHandler
```

將訊息透過Socket傳送至遠端主機。

```
java.util.logging.MemoryHandler
```

將訊息暫存在記憶體中。

範例 18.11 示範如何使用 FileHandler 將訊息輸出至檔案中。

範例 18.11 HandlerDemo.java

```java
package onlyfun.caterpillar;

import java.io.IOException;
import java.util.logging.*;

public class HandlerDemo {
    public static void main(String[] args) {
        Logger logger = Logger.getLogger("handlerDemo");

        try {
            FileHandler fileHandler =
                          new FileHandler("%h/myLogger.log");
            logger.addHandler(fileHandler);
            logger.info("測試訊息");
        } catch (SecurityException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

執行的結果會在文字模式顯示訊息，並將結果輸出至檔案中，在指定輸出的檔案名稱時，您可以使用 '%h' 來表示使用者的家（home）目錄（Windows 系統中的使用者家目錄是在 "\Documents and Settings\使用者名稱" 目錄中），您還可以使用 '%t' 取得系統的暫存目錄，或者是加入 '%g' 自動為檔案編號，例如可以設定為 "%h/myLogger%g.log"，表示將 .log 檔儲存在使用者家目錄中，並自動為每一個檔案增加編號，範例 18.11 的檔案輸出的內容如下：

```
<?xml version="1.0" encoding="x-windows-950" standalone="no"?>
<!DOCTYPE log SYSTEM "logger.dtd">
<log>
    <record>
        <date>2005-06-07T09:25:52</date>
        <millis>1118107552609</millis>
        <sequence>0</sequence>
        <logger>handlerDemo</logger>
        <level>INFO</level>
        <class>onlyfun.caterpillar.HandlerDemo</class>
        <method>main</method>
        <thread>10</thread>
        <message>測試訊息</message>
    </record>
</log>
```

預設上 FileHandler 的輸出格式是 XML 格式，輸出格式是由 java.util.logging.Formatter 來控制，例如 FileHandler 的預設格式是 java.util.logging.XMLFormatter，而 ConsolerHandler 的預設格式是 java.util.logging.SimpleFormatter，您可以使用 Handler 實例的 setFormatter() 方法來設定訊息的輸出格式，例如：

```
fileHandler.setFormatter(new SimpleFormatter());
```

如果 FileHandler 的 Formatter 設定為 SimpleFormatter，則輸出的日誌檔內容就是簡單的文字訊息，打開檔案看的話是與文字模式下看到的訊息內容相同。

18.2.4 自訂 Formatter

除了 XMLFormatter 與 SimpleFormatter 之外，您也可以自訂日誌的輸出格式，只要繼承抽象類別 Formatter，並重新定義其 format() 方法即可，format() 方法會傳入一個 java.util.logging.LogRecord 物件作為引數，您可以使用它來取得一些與程式執行有關的資訊。

範例 18.12 是個簡單的示範，當中自訂了一個簡單的 TableFormatter。

範例 18.12 TableFormatter.java

```java
package onlyfun.caterpillar;

import java.util.logging.*;

public class TableFormatter extends Formatter {
    public String format(LogRecord logRecord) {
        return
            "LogRecord info: " +
            logRecord.getSourceClassName() + "\n" +
            "Level\t|\tLoggerName\t|\tMessage\t|\n" +
            logRecord.getLevel() + "\t|\t" +
            logRecord.getLoggerName() + "\t|\t" +
            logRecord.getMessage() + "\t|\n\n";
    }
}
```

自訂好 Formatter 之後，接著就可以使用 Handler 實例的 setFormatter() 方法設定 Formatter 物件，如範例 18.13 所示。

範例 18.13 TableFormatterDemo.java

```java
package onlyfun.caterpillar;

import java.util.logging.*;
```

```
public class TableFormatterDemo {
    public static void main(String[] args) {
        Logger logger = Logger.getLogger("tableFormatter");

        try {
            for(Handler h : logger.getParent().getHandlers()) {
                if(h instanceof ConsoleHandler) {
                    h.setFormatter(new TableFormatter());
                }
            }

            logger.info("訊息1");
            logger.warning("訊息2");
        } catch (SecurityException e) {
            e.printStackTrace();
        }
    }
}
```

```

您取得預設的根（Root）Logger，並判斷出 ConsoleHandler 實例，之後設定 ConsoleHandler 實例的 Formatter 為您自訂的 TableFormatter，執行結果如下所示：

```
LogRecord info: onlyfun.caterpillar.TableFormatterDemo
Level | LoggerName | Message |
INFO | tableFormatter | 訊息1 |

LogRecord info: onlyfun.caterpillar.TableFormatterDemo
Level | LoggerName | Message |
WARNING | tableFormatter | 訊息2 |
```

### 18.2.5 Logger 階層關係

在使用 Logger 的靜態 getLogger() 方法取得 Logger 實例時，給 getLogger() 方法的名稱是有意義的，如果您給定 "onlyfun"，實際上您將從根（Root）logger 繼承一些特性，像是日誌等級（Level）以及根（Root）logger的處理者（Handler），如果您再取得一個 Logger 實例，並給定名稱 "onlyfun.caterpillar"，則這次取得的 Logger 將繼承 "onlyfun" 這個 Logger 的特性。

範例 18.14 可以看出，三個 Logger（包括根 logger）在名稱上的繼承關係。

#### 範例 18.14 LoggerHierarchyDemo.java

```java
package onlyfun.caterpillar;

import java.util.logging.*;

public class LoggerHierarchyDemo {
    public static void main(String[] args) {
        Logger onlyfunLogger = Logger.getLogger("onlyfun");
        Logger caterpillarLogger =
                     Logger.getLogger("onlyfun.caterpillar");

        System.out.println("root logger: "
                    + onlyfunLogger.getParent());
        System.out.println("onlyfun logger: "
                    + caterpillarLogger.getParent().getName());

        System.out.println("caterpillar logger: "
                   + caterpillarLogger.getName() + "\n");

        onlyfunLogger.setLevel(Level.WARNING);
        caterpillarLogger.info("caterpillar' info");
```
```

```
        caterpillarLogger.setLevel(Level.INFO);
        caterpillarLogger.info("caterpillar' info");
    }
}
```

```

getParent() 方法可以取得 Logger 的上層父 Logger，根 logger 並沒有名稱，所以您直接呼叫它的 toString() 以取得字串描述，當 Logger 沒有設定等級時，則使用父 Logger 的等級設定，所以在範例 18.14 中，onlyfunLogger 設定等級為 WARNING 時，caterpillarLogger 呼叫 info() 方法時並不會有訊息顯示（因為 WARNING 等級比 INFO 高），只有在 caterpillarLogger 設定了自己的等級為 INFO 之後，才會顯示訊息，執行結果如下：

```
root logger: java.util.logging.LogManager$RootLogger@757aef
onlyfun logger: onlyfun
caterpillar logger: onlyfun.caterpillar

2005/6/7 上午 09:50:01 onlyfun.caterpillar.LoggerHierarchyDemo main
資訊: caterpillar' info
```

在每一個處理者（Handler）方面，當每一個 Logger 處理完自己的日誌動作之後，它會向父 Logger 傳播，讓父 Logger 的處理者也可以處理日誌，例如 root logger 的處理者是 ConsoleHandler，所以若您的子 Logger 加入了 FileHandler 來處理完日誌之後，向上傳播至父 Logger 時，會再由 ConsolerHandler 來處理，所以一樣會在文字模式上顯示訊息，之前的範例 18.11 就是這樣的例子。

# 18.3 訊息綁定

程式中的一些文字訊息可以將之定義在一個屬性檔案中，而不一定要寫死在程式碼當中，如此在日後想要更改文字訊息時，只要更改文字檔案的內容而不用重新編譯程式，就可以在程式運行時顯示不同的訊息，這個小節也會有一些國際化（Internationalization）處理的簡單介紹。

## 18.3.1 使用 ResourceBundle

在程式中有很多字串訊息會被寫死在程式中，如果您想要改變某個字串訊息，您必須修改程式碼然後重新編譯，例如簡單顯示 "Hello!World!" 的程式就是如此：

```java

```
public class Hello {
     public static void main(String[] args) {
         System.out.println("Hello!World!");
     }
}
```

```

就這個程式來說，如果日後想要改變 "Hello!World!" 為 "Hello!Java!"，您就要修改程式碼中的文字訊息並重新編譯。

對於日後可能變動的文字訊息，您可以考慮將訊息移至程式之外，方法是使用 java.util.ResourceBundle 來作訊息綁定，首先您要先準備一個 .properties 檔案，例如 messages.properties，而檔案內容如下：

```
onlyfun.caterpillar.welcome=Hello
onlyfun.caterpillar.name=World
```

.properties 檔案必須放置在 Classpath 的路徑設定下，檔案中撰寫的是鍵（Key）、值（Value）配對，之後在程式中您可以使用鍵來取得對應的值，接著改寫 Hello 類別為範例 18.15。

### 範例 18.15 ResourceBundleDemo.java

```java

```
package onlyfun.caterpillar;

import java.util.ResourceBundle;

public class ResourceBundleDemo {
    public static void main(String[] args) {
```

```
        // 綁定messages.properties
        ResourceBundle resource =
                  ResourceBundle.getBundle("messages");
        // 取得對應訊息
        System.out.print(resource.getString(
                     "onlyfun.caterpillar.welcome") + "!");
        System.out.println(resource.getString(
                     "onlyfun.caterpillar.name") + "!");
    }
}
```

ResourceBundle 的靜態 getBundle() 方法會取得一個 ResourceBundle 的實例，所給定的引數名稱是訊息檔案的主檔名，getBundle() 會自動找到對應的 .properties 檔案，取得 ResourceBundle 實例後，可以使用 getString() 指定鍵來取得檔案中對應的值，執行結果如下：

```
Hello!World!
```

如果您日後想要改變顯示的訊息，只要改變 .properties 檔案的內容就可以了，例如可以將 messages.properties 檔案內容改為：

```
onlyfun.caterpillar.welcome=Oh
onlyfun.caterpillar.name=Java
```

無需重新編譯範例 18.15，直接執行程式，則輸出訊息如下：

```
Oh!Java!
```

> **良葛格的話匣子** 在撰寫 .properties 檔案的內容時，鍵的命名前可以加上單位、公司或是網址名，例如我的網址是 openhome.cc，則我可以將鍵命名為 cc.openhome.XXX，如此在訊息增多時可以減少鍵名的名稱衝突問題。

18.3.2 格式化訊息

程式在運行的過程中，有些訊息可能必須動態決定，而之前介紹訊息綁定時，.properties 檔案中的訊息則是靜態的，也就是固定的文字內容，除非修改 .properties 檔案內容並重新啟動程式，不然的話一些訊息內容無法隨著程式動態顯示。

您可以使用 java.text.MessageFormat 類別來輔助訊息的格式化，MessageFormat 接受一個字串模式（Pattern）指定，對於文字訊息中可能變動的部份，可以使用引數索引（Argument Index）先佔住文字位置，引數索引是 {0} 到 {9} 的非負整數，之後在使用 MessageFormat 實例的 format() 方法時，可以提供真正的引數來填充引數索引處的訊息。

舉個例子來說，您可以如下格式化文字訊息：

```
String message = "Hello! {0}! This is your first {1}!";
Object[] params =
        new Object[] {"caterpillar", "Java"};
MessageFormat formatter =
         new MessageFormat(message);
// 顯示格式化後的訊息
System.out.println(formatter.format(params));
```

MessageFormat 實例的 format() 方法會使用 params 陣列中的物件之 toString() 方法，將取得的 String 訊息依索引位置分別填入 {0} 到 {9} 的對應位置，執行 format() 方法後傳回的就是格式化後的訊息內容，就上面的程式片段而言，會顯示 "Hello! caterpillar! This is your first Java!" 訊息。

依照這樣的作法，如果您想在訊息綁定時也能進行訊息的格式化，讓一些訊息在程式運行過程中動態決定，則您可以在 .properties 中如下撰寫，例如撰寫 messages2.properties：

```
onlyfun.caterpillar.greeting=Hello! {0}! This is your first {1}!
```

接著您可以綁定 messages2.properties，並在程式中進行訊息的格式化，如範例 18.17 所示。

範例 18.17 MessageFormatDemo.java

```java
```

```
package onlyfun.caterpillar;

import java.util.ResourceBundle;
import java.text.MessageFormat;

public class MessageFormatDemo {
    public static void main(String[] args) {
        try {
            // 綁定messages.properties
            ResourceBundle resource =
                  ResourceBundle.getBundle("messages2");

            String message = resource.getString(
                  "onlyfun.caterpillar.greeting");
            Object[] params =
                new Object[] {args[0], args[1]};
            MessageFormat formatter =
                 new MessageFormat(message);

            // 顯示格式化後的訊息
            System.out.println(formatter.format(params));
        }
        catch(ArrayIndexOutOfBoundsException e) {
            System.out.println("沒有指定引數");
        }
    }
}
```

```

程式運行時您必須給它兩個引數，分別作為 {0} 與 {1} 的訊息填充，執行時的一個例子如下：

```
java onlyfun.caterpillar.MessageFormatDemo caterpillar Java
Hello! caterpillar! This is your first Java.
```

### 18.3.3 國際化訊息

國際化的英文是 Internationalization，因為單字中總共有18個字母而首尾字元分別為 'I' 與 'N'，所以簡稱 I18N，國際化的目的是讓應用程式可以依地區不同而顯示不同的訊息，最基本的就是讓不同語系的使用者可以看到屬於自己語系的訊息，像是英文語系的看到英文內容，而中文語系的可以看到中文的內容。

為了在應用程式中表示一個區域，Java 提供有 java.util.Locale 類，一個 Locale 實例包括了語系資訊與區域資訊，例如 "en" 表示英文語系的國家，這個字母組合是在 ISO 639 中定義的，而區域資訊則是像 "US" 表示美國，這個字母組合則是在 ISO 3166 中定義的。

您可以這麼新增一個 Locale 的實例，用以表示中文語系 "zh"、台灣 "TW"：

```
Locale locale = new Locale("zh", "TW");
```

如何將 Locale 用於訊息綁定呢？當您使用 ResourceBundle.getBundle() 方法時，預設就會自動取得電腦上的語系與區域訊息，而事實上訊息檔案的名稱由 basename 加上語系與地區來組成，例如：

- basename.properties (預設)
- basename_en.properties
- basename*zh*TW.properties

沒有指定語言與地區的 basename 是預設的資源檔名稱，當沒有提供專用的語系、區域訊息檔案時，就會找尋預設的資源檔案。

如果您想要提供繁體中文的訊息，由於訊息資源檔必須是 ISO-8859-1 編碼，所以對於非西方語系的處理，必須先將之轉換為 Java Unicode Escape 格式，例如您可以先在 messages3*zh*TW.txt 中撰寫以下的內容：

```
onlyfun.caterpillar.welcome=哈囉
onlyfun.caterpillar.name=世界
```

然後使用 JDK 的工具程式 native2ascii 來轉換，例如：
```

```
native2ascii -encoding Big5 messages3_zh_TW.txt messages3_zh_TW.properties
```

轉換後的 messages3_zh_TW.properties 檔案內容會如下：

```
onlyfun.caterpillar.welcome=\u54c8\u56c9
onlyfun.caterpillar.name=\u4e16\u754c
```

將這個檔案放於 Classpath 可以存取的到的路徑位置，您也可以提供預設的訊息檔案 messages3.properties：

```
onlyfun.caterpillar.welcome=Hello
onlyfun.caterpillar.name=World
```

來測試一下訊息檔案，我所使用的作業系統是語系設定是中文，區域設定是台灣，來撰寫範例 18.18 進行測試。

範例 18.18 I18NDemo.java

```java
package onlyfun.caterpillar;

import java.util.ResourceBundle;

public class I18NDemo {
    public static void main(String[] args) {
        ResourceBundle resource =
                  ResourceBundle.getBundle("messages3");

        System.out.print(resource.getString(
                      "onlyfun.caterpillar.welcome") + "!");
        System.out.println(resource.getString(
                      "onlyfun.caterpillar.name") + "!");
    }
}
```

根據我作業系統的設定，執行程式時會使用預設的語系 "zh" 與區域設定 "TW"，所以就會找尋 messages_zh_TW.properties 的內容，結果會顯示以下的訊息：

```
哈囉!世界!
```

在使用 ResourceBundle.getBundle() 時可以給定 Locale 實例作為參數，例如若您想提供 messages_en_US.properties 檔案，並想要 ResourceBundle.getBundle() 取得這個檔案的內容，則可以如下撰寫：

```
Locale locale = new Locale("en", "US");
ResourceBundle resource =
        ResourceBundle.getBundle("messages", locale);
```

根據 Locale 物件的設定，這個程式片段將會取得 messages_ en_US.properties 檔案的訊息內容。

18.4 接下來的主題

到這邊為止，都使用一些簡單的範例來為您示範每個API的功能要如何使用，為了讓您實際了解到這些獨立的 API 如何組成一個完整的應用程式，下一個章節將實際製作一個文字編輯器，由於文字編輯器將使用到 Swing 來作為視窗介面，所以您也可以在下一個章節中了解到一些 Swing 視窗程式的基本概念。