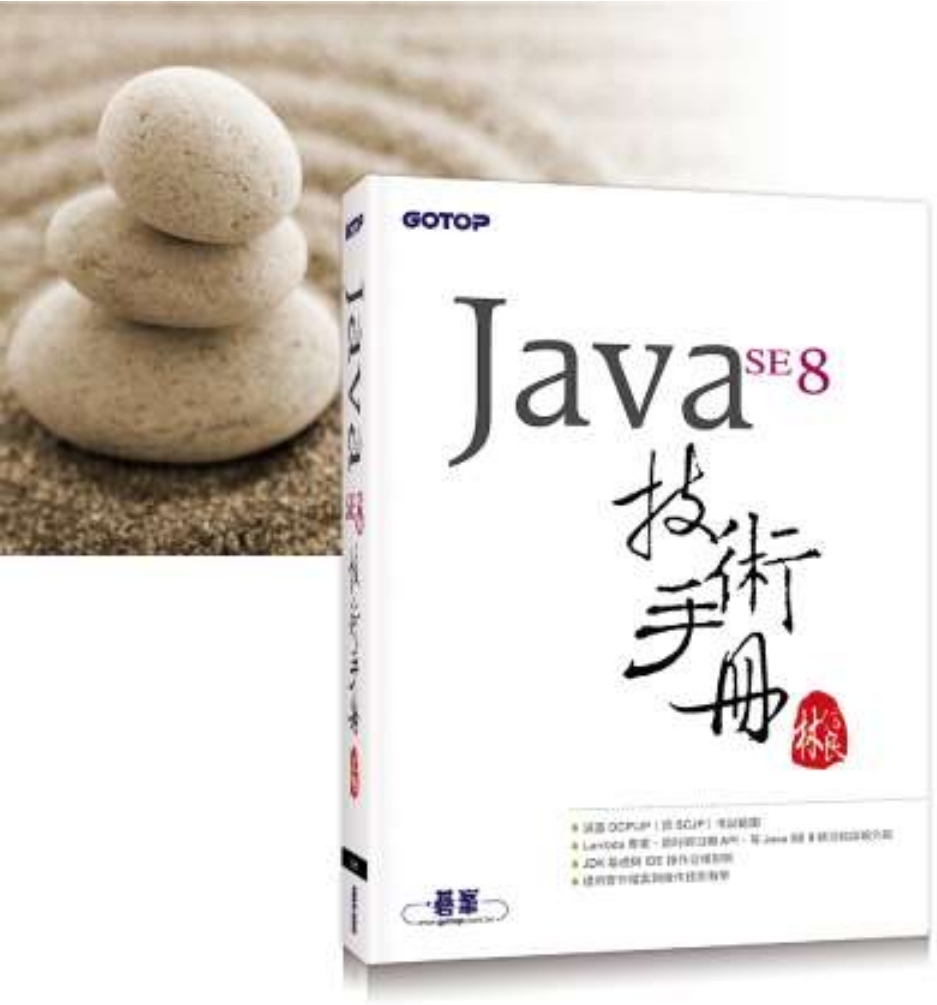


Java SE 8 技術手冊

林信良

- 涵蓋 OCP/JP（原 SCJP）考試範圍
- Lambda 專案、新時間日期 API、等 Java SE 8 新功能詳細介紹
- JDK 基礎與 IDE 操作交相對照
- 提供實作檔案與操作錄影教學

碁峰資訊

CHAPTER

時間與日期

學習目標

- 建立時間與日期的認知
- 認識Date與Calendar
- 使用JDK8新的時間日期API
- 區分機器與人類時間概念

時間的度量

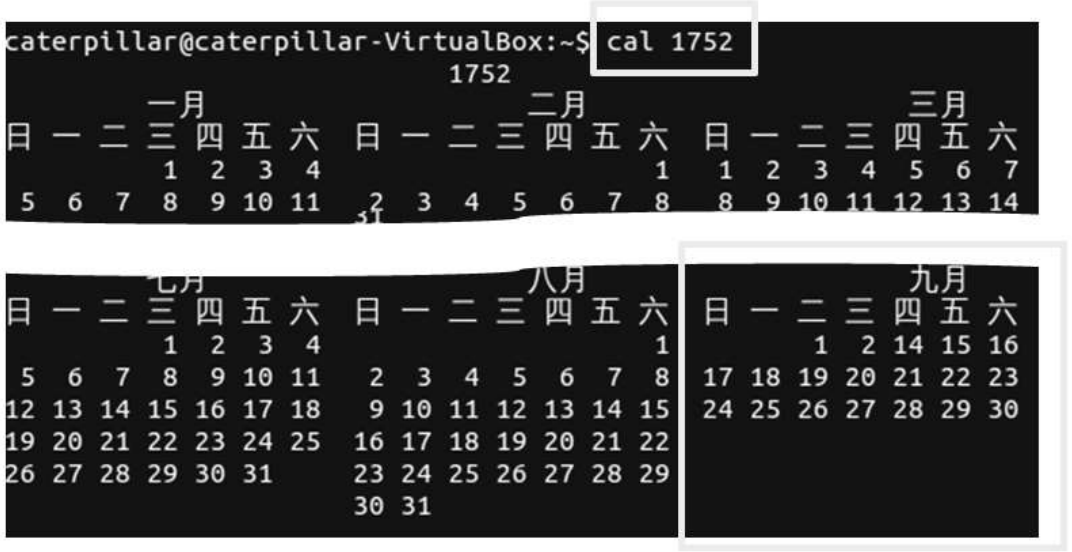
- 格林威治標準時間
- 世界時
- 國際原子時
- 世界協調時間
- Unix時間
- epoch

時間的度量

- 就目前來說，即使標註為GMT，實際上談到時間指的是UTC時間
- 秒的單位定義是基於TAI，也就是銫原子輻射振動次數
- UTC考量了地球自轉越來越慢而有閏秒修正
- Unix時間是1970年1月1日00:00:00 為起點而經過的秒數，不考慮閏秒

年曆簡介

- 儒略曆
- 格里高利曆



- ISO8601標準
  - 時間日期表示方法的標準，用以統一時間日期的資料交換格式

認識時區

- 牽涉到地理、法律、經濟、社會甚至政治等問題
  - UTC偏移（offset）
  - 有些國家的領土橫跨的經度很大，一個國家有多個時間反而造成困擾，因而不採取每15度偏移一小時的作法
  - 日光節約時間（Daylight saving time）、夏季時間（Summer time）
  - **台灣也曾實施過日光節約時間**

時間軸上瞬間的Date

- 取得系統時間，方法之一是使用 `System.currentTimeMillis()`
- 代表1970年1月1日0時0分0秒0毫秒至今經過的毫秒數
- 機器的時間觀點

時間軸上瞬間的Date

- Date也是偏向機器的時間觀點

```
Date date1 = new Date(currentTimeMillis());
Date date2 = new Date();

out.println(date1.getTime());
out.println(date2.getTime());
```

```
1398741380778
1398741380778
```

時間軸上瞬間的Date

- Date類別是從JDK1.0就已存在的API
  - 除了範例中使用的兩個建構式外，其他版本的建構式都已廢除
  - getTime()之外的getXXX()方法都廢棄了
  - setTime()（用來設置epoch毫秒數）外的setXXX()方法也都廢棄了
- **Date實例基本上建議只用來當作時間軸上的某一瞬間**

時間軸上瞬間的Date

- 不建議使用`toString()`來得知年月日等欄位資訊
- 有關於字串時間格式的處理，不再是`Date`的職責

格式化時間日期的DateFormat

- 字串時間格式的處理，職責落到了java.text.DateFormat身上
- 實作類別java.text.SimpleDateFormat
  - 直接建構SimpleDateFormat實例
  - 使用DateFormat的getDateInstance()、getTimeInstance()、getDateTimeInstance()等靜態方法

格式化時間日期的DateFormat

```
static void dateInstanceDemo(Date date) {
    out.println("getDateInstance() demo");
    out.printf("\tSHORT: %s%n", getDateInstance(LONG).format(date));
    out.printf("\tSHORT: %s%n", getDateInstance(SHORT).format(date));
}

static void timeInstanceDemo(Date date) {
    out.println("getTimeInstance() demo");
    out.printf("\tLONG: %s%n", getTimeInstance(LONG).format(date));
    out.printf("\tMEDIUM: %s%n", getTimeInstance(MEDIUM).format(date));
    out.printf("\tSHORT: %s%n",getTimeInstance(SHORT).format(date));
}

static void dateTimeInstanceDemo(Date date) {
    out.println("getDateTimeInstance() demo");
    out.printf("\tLONG: %s%n",
       getDateTimeInstance(LONG, LONG).format(date));
    out.printf("\tMEDIUM: %s%n",
       getDateTimeInstance(SHORT, MEDIUM).format(date));
    out.printf("\tSHORT: %s%n",
       getDateTimeInstance(SHORT, SHORT).format(date));
}
```

格式化時間日期的DateFormat

- 直接建構SimpleDateFormat的好處是，可使用模式字串自訂格式

```
DateFormat dateFormat = new SimpleDateFormat(
        args.length == 0 ? "EE-MM-dd-yyyy" : args[0]);
System.out.println(dateFormat.format(new Date()));

System.out.print("輸入出生年月日（yyyy-mm-dd）：");
DateFormat dateFormat = new SimpleDateFormat("yyyy-mm-dd");
Date birthDate = dateFormat.parse(new Scanner(System.in).nextLine());
Date currentDate = new Date();
long life = currentDate.getTime() - birthDate.getTime();
System.out.println("你今年的歲數為：" +
          (life / (365 * 24 * 60 * 60 * 1000L)));
```

處理時間日期的Calendar

- 想要取得某個時間日期資訊，或者是對時間日期進行操作，可以使用Calendar實例
- Calendar是個抽象類別
  - java.util.GregorianCalendar是其子類，實作了儒略曆與格里高利曆的混合曆

```
Calendar calendar = Calendar.getInstance();
out.println(calendar.get(Calendar.YEAR));   // 2014
out.println(calendar.get(Calendar.MONTH));  // 3
out.println(calendar.get(Calendar.DATE));   // 30
```

處理時間日期的Calendar

- 列舉值的一月是從0數字開始：

```
public final static int JANUARY = 0;
public final static int FEBRUARY = 1;
public final static int MARCH = 2;
public final static int APRIL = 3;
public final static int MAY = 4;
public final static int JUNE = 5;
public final static int JULY = 6;
public final static int AUGUST = 7;
public final static int SEPTEMBER = 8;
public final static int OCTOBER = 9;
public final static int NOVEMBER = 10;
public final static int DECEMBER = 11;
```

處理時間日期的Calendar

- 改變Calendar的時間

```
calendar.add(Calendar.MONTH,  1);        // Calendar 的時間加 1 個月
calendar.add(Calendar.HOUR,  3);         // Calendar 的時間加 3 小時
calendar.add(Calendar.YEAR, -2);         // Calendar 的時間減 2 年
calendar.add(Calendar.DATE, 3);          // Calendar 的時間加 3 天

calendar.roll(Calendar.DATE, 1);          // 只對日欄位加 1
```

處理時間日期的Calendar

- 預設的改曆時間為格里高利曆 1582 年 10 月 15 日星期五

```
Calendar calendar = Calendar.getInstance();
calendar.set(1582, Calendar.OCTOBER, 15);
out.println(calendar.getTime());          // 顯示格里高利曆 1582 年 10 月 15 日
calendar.add(Calendar.DAY_OF_MONTH, -1); // 往前一天
out.println(calendar.getTime());          // 顯示儒略曆 1582 年 10 月 4 日
```

- 單純地使用365 * 24 * 60 * 60 * 1000當作一年的毫秒數並用以計算使用者歲數是不對的

```
public static long yearsBetween(Calendar begin, Calendar end) {
    Calendar calendar = (Calendar) begin.clone();
    long years = 0;
    while (calendar.before(end)) {
        calendar.add(Calendar.YEAR, 1);
        years++;
    }
    return years - 1;
}

public static long daysBetween(Calendar begin, Calendar end) {
    Calendar calendar = (Calendar) begin.clone();
    long days = 0;
    while (calendar.before(end)) {
        calendar.add(Calendar.DATE, 1);
        days++;
    }
    return days - 1;
}
```

設定TimeZone

- 使用java.util.TimeZone的getDefault()來取得預設時區資訊

```
TimeZone timeZone = TimeZone.getDefault();
out.println(timeZone.getDisplayName());
out.println("\t時區 ID：" + timeZone.getID());
out.println("\t日光節約時數：" + timeZone.getDSTSavings());
out.println("\tUTC 偏移毫秒數：" + timeZone.getRawOffset());
```

設定TimeZone

- 想要取得指定時區的TimeZone實例，可以使用ID字串

```
TimeZone taipeiTz = TimeZone.getTimeZone("Asia/Taipei");
TimeZone copenhagenTz = TimeZone.getTimeZone("Europe/Copenhagen");
```

設定TimeZone

- 想知道現在哥本哈根的時間

```
public static void main(String[] args) {
    TimeZone taipeiTz = TimeZone.getTimeZone("Asia/Taipei");
    Calendar calendar = Calendar.getInstance(taipeiTz);
    showTime(calendar);

    TimeZone copenhagenTz = TimeZone.getTimeZone("Europe/Copenhagen");
    calendar.setTimeZone(copenhagenTz);
    showTime(calendar);
}

static void showTime(Calendar calendar) {
    out.print(calendar.getTimeZone().getDisplayName());;
    out.printf(" %d:%d%n",
            calendar.get(Calendar.HOUR),
            calendar.get(Calendar.MINUTE));
}
```

JDK8新時間日期API

- Date實例真正代表的並不是日期，最接近的概念應該是時間軸上特定的一瞬間
- Date狀態仍是可變的
- 使用Calendar太麻煩、太痛苦了
- Calendar狀態可變
- JDK8中有了新的時間日期處理API，規格書為JSR310

機器時間觀點的API

- Date名稱看來像是人類的時間概念，實際卻是機器的時間概念
- 混淆機器與人類時間觀點會引發的問題之一像是日光節約時間

```
Calendar calendar = Calendar.getInstance();
calendar.set(1975, Calendar.MARCH, 31, 23, 59, 59);
out.println(calendar.getTime());          // Mon Mar 31 23:59:59 CST 1975
calendar.add(Calendar.SECOND, 1);         // 增加一秒
out.println(calendar.getTime());          // Tue Apr 01 01:00:00 CDT 1975
```

機器時間觀點的API

- 台灣已經不實施日光節約時間一段時間了，許多開發者並不知道過去有過日光節約時間
- 被名稱`Date`誤導它們代表日期
- 不該使用`Date`實例的`toString()`來得知人類觀點的時間資訊
- `Date`實例應該只代表機器觀點的時間資訊，真正可靠的資訊只有內含的epoch毫秒數

機器時間觀點的API

- 取得Date實例，下一步該獲取時間資訊應該是透過Date的getTime()取得epoch毫秒數

```
Calendar calendar = Calendar.getInstance();
calendar.set(1975, Calendar.MARCH, 31, 23, 59, 59);
out.println(calendar.getTime().getTime());      // 165513599484
calendar.add(Calendar.SECOND, 1);               // 增加一秒
out.println(calendar.getTime().getTime());      // 165513600484
```

機器時間觀點的API

- JDK8新時間日期處理API**清楚地將機器對時間的概念與人類對時間的概念區隔開來**
  - 對於機器相關的時間概念，設計了`Instant`類別
  - 代表自定義的Java epoch（1970 年 1 月 1 日）之後的某個時間點歷經的毫秒數
- 新舊API相容上
  - 呼叫`Date`實例的`toInstant()`取得`Instant`
  - 使用`Date`靜態方法`from()`將`Instant`轉`Date`

人類時間觀點的API

- 人類在時間概念的表達大多是籠統、片段的資訊
  - LocalDateTime、LocalDate、LocalTime
  - ZonedDateTime、OffsetDateTime
  - Year、YearMonth、Month、MonthDay

人類時間觀點的API

- LocalDateTime包括日期與時間
  LocalDate只有日期
- LocalTime只有時間
- 不具時區的時間與日期定義
- 基於ISO-8601年曆系統
- 只是對時間的描述

人類時間觀點的API

- 如果時間日期需要有帶有時區，可以基於LocalDateTime、LocalDate、LocalTime等來補齊缺少的資訊：

```
LocalTime localTime = LocalTime.of(0, 0, 0);
LocalDate localDate = LocalDate.of(1975, 4, 1);
ZonedDateTime zonedDateTime = ZonedDateTime.of(
        localDate, localTime, ZoneId.of("Asia/Taipei"));

out.println(zonedDateTime);
out.println(zonedDateTime.toEpochSecond());
out.println(zonedDateTime.toInstant().toEpochMilli());
```

人類時間觀點的API

- UTC 偏移量與時區的概念是分開的。
  `OffsetDateTime`單純代表 UTC 偏移量

```
LocalDate nowDate = LocalDate.now();
LocalTime nowTime = LocalTime.now();
OffsetDateTime offsetDateTime =
        OffsetDateTime.of(nowDate, nowTime, ZoneOffset.UTC);
```

人類時間觀點的API

- 如果只想表示2014年，可以使用Year
- 如果想表示2014/5，可以使用YearMonth
- 如果只想表示5月，可以使用Month
- 如果想表示5/4，可以使用MonthDay

```
for(Month month : Month.values()) {
    out.printf("original: %d\tvalue: %d\t%s%n",
            month.ordinal(), month.getValue(), month);
}
```

對時間的運算

- 某個日期起加上5天、6個月、3週後會的日期時間是什麼，並使用指定的格式輸出

```
Calendar calendar = Calendar.getInstance();
calendar.set(1975, Calendar.MAY, 26, 0, 0, 0);
calendar.add(Calendar.DAY_OF_MONTH, 5);
calendar.add(Calendar.MONTH, 6);
calendar.add(Calendar.WEEK_OF_MONTH, 3);
SimpleDateFormat df = new SimpleDateFormat("E MM/dd/yyyy");
out.println(df.format(calendar.getTime()));
```

對時間的運算

- JDK8新日期時間處理實現了流暢API（Fluent API）的概念

```
out.println(
    LocalDate.of(1975, 5, 26)
             .plusDays(5)
             .plusMonths(6)
             .plusWeeks(3)
             .format(ofPattern("E MM/dd/yyyy"))
);

out.println(
    LocalDate.of(1975, 5, 26)
             .plus(ofDays(5))
             .plus(ofMonths(6))
             .plus(ofWeeks(3))
             .format(ofPattern("E MM/dd/yyyy"))
);
```

對時間的運算

- 先前看過的HowOld範例，也可以使用新時間與日期API改寫

```
out.print("輸入出生年月日（yyyy-mm-dd）：");
LocalDate birth = LocalDate.parse(new Scanner(System.in).nextLine());
LocalDate now = LocalDate.now();
Period period = Period.between(birth, now);
out.printf("你活了 %d 年 %d 月 %d 日%n",
        period.getYears(), period.getMonths(), period.getDays());
```

對時間的運算

- 某個日期起加上5天、6個月、3週後會的日期時間是什麼，並使用指定的格式輸出

```
out.println(
    LocalDate.of(1975, 5, 26)
             .plus(5, DAYS)
             .plus(6, MONTHS)
             .plus(3, WEEKS)
             .format(ofPattern("E MM/dd/yyyy"))
);
```

對時間的運算

- 使用實作類別ChronoUnit的列舉實例來實作之前的CalendarUtil範例

```
LocalDate birth = LocalDate.of(1975, 5, 26);
LocalDate now = LocalDate.now();
out.printf("歲數：%d%n", ChronoUnit.YEARS.between(birth, now));
out.printf("天數：%d%n", ChronoUnit.DAYS.between(birth, now));
```

年曆系統設計

- java.time套件中的類別在需要採行年曆系統時都是採用單一的ISO8601年曆系統
- 如果需要其他年曆系統呢？需要明確採行java.time.chrono中等實作了java.time.chrono.Chronology介面的類別

年曆系統設計

java.time.chrono

Interface Chronology

All Superinterfaces:

`Comparable<Chronology>`

All Known Implementing Classes:

`AbstractChronology, HijrahChronology, IsoChronology, JapaneseChronology, MinguoChronology, ThaiBuddhistChronology`

年曆系統設計

- MinguoChronology就是中華民國年曆，也就是台灣通行的年曆系統
- 將西元年月日轉換為民國年月日

```
LocalDate birth = LocalDate.of(1975, 5, 26);
MinguoDate mingoBirth = MinguoDate.from(birth);
out.println(mingoBirth);  // Minguo ROC 64-05-26
```

- 想要同時表示民國日期與時間

```
out.println(
    MinguoDate.of(64, 5, 1)
              .atTime(LocalTime.of(3, 30, 0))); // Minguo ROC 64-05-01T03:30
```