

包裝類別與工具類別(1)

TYIC 桃高資訊社

包裝類別

雖然在 **Java** 中幾乎所有東西都是物件，但基本資料型別卻不是這導致基本資料型別無法像物件一樣呼叫方法

所以出現了包裝類別(wrapper class)來解決這個問題

8 種基本資料型別對應了 8 種包裝類別，分別為：

Byte、**Short**、**Character**、**Integer**、

Long、**Float**、**Double**、**Boolean**

這些包裝類別皆位於 **java.lang** 套件內，所以可以直接使用包裝類別和字串一樣，具有不可變性，即不可更改內容

包裝類別

欲創建包裝類別，須呼叫包裝類別的公開靜態方法 `"valueOf"`

必有一個多載具有唯一參數，且為對應的基本資料型別

有些包裝類別有多載該方法，可能的參數有字串等

這個動作稱為裝箱(`boxing`)

而呼叫包裝類別的公開動態方法 `"xxxValue"` (`xxx`為基本資料型別)

將包裝類別變為基本資料型別


就被稱為拆箱(`unboxing`)

自動拆箱

包裝類別可以像基本資料型別一樣進行各式運算

```
public class Main1 {  
    public static void main(String[] args) {  
        System.out.println(100 + Integer.valueOf(200));  
        System.out.println(Integer.valueOf(100) / 200);  
        System.out.println(Integer.valueOf(100) % Integer.valueOf(200));  
    }  
}
```

自動拆箱



java

這是因為編譯器會在包裝類別運算前呼叫

"xxxValue" (xxx為基本資料型別) 方法，稱為自動拆箱

```
public class Main1 {  
    public static void main(String[] args) {  
        System.out.println(100 + Integer.valueOf(200).intValue());  
        System.out.println(Integer.valueOf(100).intValue() / 200);  
        System.out.println(Integer.valueOf(100).intValue() % Integer.valueOf(200).intValue());  
    }  
}
```

java

自動裝箱

將包裝類別賦值給
基本資料型別的變數時
也會自動拆箱
而將基本資料型別賦值
給包裝類別的變數時
則會自動裝箱

```
public class Main2 {  
    public static void main(String[] args) {  
        final Integer TWO_HUNDRED = 200;  
        System.out.println(add(100, TWO_HUNDRED));  
    }  
  
    public static Integer add(Integer a, Integer b) {  
        return a + b;  
    }  
}
```

自動裝箱 自動拆箱

java

```
public class Main2 {  
    public static void main(String[] args) {  
        final Integer TWO_HUNDRED = Integer.valueOf(200);  
        System.out.println(add(100, TWO_HUNDRED.intValue()));  
    }  
  
    public static Integer add(Integer a, int b) {  
        return Integer.valueOf(a + b);  
    }  
}
```

java



陣列

在 Java 中，陣列也是個物件
並且直接繼承 **Object**，但是沒有覆寫任何方法
建立陣列的方式如下：

```
new 元素型別[] {元素1, 元素2, ..., 元素n} // 第一種，指定陣列內容  
new 元素型別[陣列長度] // 第二種，無指定陣列內容
```

java

第一種創建方式指定了陣列的內容和長度(**length**)
大括號內填入不定數量的元素，以逗號分隔，元素的數量即為陣列長度
第二種創建方式只指定了陣列長度，並沒有指定內容，內容為預設值
兩種皆只能作為表達式，且陣列長度在創建陣列後不可變
陣列的型別為 "元素型別[]"

陣列

在賦值給陣列變數時

若使用第一種(指定內容)的方式創建陣列

則可以省略前方的 "new 元素型別[]"

```
元素型別[] 變數名稱 = new 元素型別[]{元素1, 元素2, ..., 元素n};  
元素型別[] 變數名稱 = {元素1, 元素2, ..., 元素n};
```

java

陣列沒有方法，但有一個不可變欄位 "length"，代表陣列長度

若要存取陣列的某個元素，需要透過下標運算子[] 來存取

```
陣列[索引值]
```

java

可將下標運算整體視為一個變數，像變數一樣操作

索引值

元素1

元素2

元素3

元素4

元素5

元素6

元素7

元素8

索引值0

索引值1

索引值2

索引值3

索引值4

索引值5

索引值6

索引值7

索引值(index)是用來表示陣列的某個元素的位置

從 0 開始編號，到陣列長度 - 1，所以第 1 個元素索引值為 0

第 n 個元素索引值為 $n - 1$

```
public class Main1 {  
    public static void main(String[] args) {  
        int[] arr = {6, 4, 5, 7, 2, 9};  
        for (int i = 0; i < arr.length; i++) {  
            System.out.print(arr[i] + " ");  
        }  
    }  
}
```

6 4 5 7 2 9 output java

```
public class Main2 {  
    public static void main(String[] args) {  
        int[] arr = {6, 4, 5, 7, 2, 9};  
        arr[5]++;  
        arr[4] = 0;  
        for (int i = 0; i <= arr.length - 1; i++) {  
            System.out.print(arr[i] + " ");  
        }  
    }  
}
```

6 4 5 7 0 10 output java

若存取的索引值超過最大索引值，則會出現錯誤

for-each

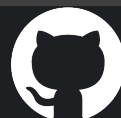
每次迭代(**iteration**)陣列都寫這麼長一個 **for** 迴圈，實在不便
所以在不需要索引值的情況下，可以使用「**for-each**」來替代

```
for (陣列元素型別 變數 : 陣列) {  
    陳述式...  
}
```

java

其中**變數**會在每次循環依序變為**陣列**中的**元素**

```
public class Main3 {  
    public static void main(String[] args) {  
        int[] arr = {6, 4, 5, 7, 2, 9};  
        for (int e : arr) {  
            System.out.print(e + " ");  
        }  
    }  
}
```



java

6 4 5 7 2 9

output

k 維陣列

剛剛所介紹的叫做一維陣列(1D array)

由 1 個索引值確定元素

而二維陣列(2D array)由 2 個索引值確定元素

k 維陣列由 k 個索引值確定元素

在 Java 中，二維陣列就是元素型別為一維陣列的一維陣列

k 維陣列就是元素型別為(k - 1)維陣列的一維陣列

```
public class Main4 {  
    public static void main(String[] args) {  
        int[][] arr = {{2, 1, 4}, {7, 4}, {8, 3, 6, 4}, {7}};  
        for (int[] subArr : arr) {  
            for (int e : subArr) {  
                System.out.print(e + " ");  
            }  
            System.out.println();  
        }  
    }  
}
```



java

```
2 1 4  
7 4  
8 3 6 4  
7
```

output

陣列工具類別

陣列的工具類別是 `java.util.Arrays`

當中定義了許多關於陣列的公開靜態方法，如：

`arrType copyOf(srcArray, newArrLength)`、`void sort(array)`


`void equals(arr1, arr2)`、`String toString(array)`

`void fill(array, value)`、`int binarySearch(array, value)`

更多方法可以在 `Java API` 中查找：[Java 21 API](#) 

```
import java.util.Arrays;

public class Main1 {
    public static void main(String[] args) {
        int[] arr1 = {1, 4, 7, 2, 5, 8, 3, 6, 9};
        int[] arr2 = Arrays.copyOf(arr1, arr1.length);
        System.out.println(Arrays.equals(arr1, arr2));
        Arrays.sort(arr1);
        System.out.println(Arrays.binarySearch(arr1, 2));
        System.out.println(Arrays.toString(arr1));
        int[] arr3 = new int[4];
        Arrays.fill(arr3, 6);
        System.out.println(Arrays.toString(arr3));
    }
}
```


java

```
true
4
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[6, 6, 6, 6]                                output
```

特別注意，呼叫 `binarySearch` 前
一定要先將陣列由小到大排序(`sort`)
這與二分搜尋法原理有關

不定長度引數

不定長度引數(**variable-length argument**)

是指傳入的**引數**數量不限制，可以有很多個，使用**不定長度參數**接收而一個**方法**只能有一個**不定長度參數**，且必須是最後一個**參數**

不定長度參數是一個**陣列**，內容為**不定長度引數**

不定長度參數也可以接收**陣列**，但互換則不行

```
修飾子 返回值型別 方法名稱(參數型別 參數名稱, ..., 不定長度參數型別... 不定長度參數名稱) {  
    陳述式...  
}
```

java

下方為 **java.io.PrintStream** 的 **printf** 方法定義
其使用到**不定長度參數**來接收不定數量的物件

```
public PrintStream printf(String format, Object ... args) {  
    return format(format, args);  
}
```

java

153 289 51 459

17

console

不定長度引數

194 2716 582 1746 9506 388^D

0

console

```
import java.util.Arrays;
import java.util.Scanner;

public class Main5 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int a = scanner.nextInt();
        int b = scanner.nextInt();
        int c = scanner.nextInt();
        int d = scanner.nextInt();
        System.out.println(gcd(a, b, c, d));
    }

    static int gcd(int a, int b) {
        if (b == 0) return a;
        return gcd(b, a % b);
    }

    static int gcd(int... nums) {
        if (nums.length == 1) return nums[0];
        return gcd(nums[0],
            gcd(Arrays.copyOfRange(nums,
                1, nums.length)));
    }
}
```



java

```
import java.util.Arrays;
import java.util.Scanner;

public class Main6 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int index = 0;
        int[] arr = new int[200];
        while (index < 200 && scanner.hasNextInt()) {
            arr[index++] = scanner.nextInt();
        }
        System.out.println(gcd(arr));
    }

    static int gcd(int a, int b) {
        if (b == 0) return a;
        return gcd(b, a % b);
    }

    static int gcd(int... nums) {
        if (nums.length == 1) return nums[0];
        return gcd(nums[0],
            gcd(Arrays.copyOfRange(nums,
                1, nums.length)));
    }
}
```



java

數學

在 **Java** 中，關於數學的東西多半定義在 **java.lang.Math**

其中的靜態方法有許多，下方為部分靜態方法

返回值型別皆為 **double**，且皆只有一個 **double** 參數：

基礎運算：**abs**(絕對值)、**sqrt**(開根號)、**cbrt**(開三次方根)

log10(常用對數)、**exp**(自然指數)、**log**(自然對數)

三角運算：**sin**(正弦)、**cos**(餘弦)、**tan**(正切)

sin2(正弦平方)、**cos2**(餘弦平方)、**tan2**(正切平方)

asin(反正弦)、**acos**(反餘弦)、**atan**(反正切)

toRadians(角度轉弧度)、**toDegrees**(弧度轉角度)

其中三角函數、反三角函數使用的單位都是弧度

數學

返回值型別皆為 `long`，且皆只有一個 `double` 參數：

`round`(四捨五入)、`floor`(向下取整)、`ceil`(向上取整)

返回值型別皆為 `double`，且皆只有兩個 `double` 參數：

`pow`(指數)、`max`(取最大)、`min`(取最小)

以及限制數值範圍的方法，保證 `value` 不超過 `max` 或 `min`：

`double clamp(double value, double min, double max)`

該類別也定義了一些數學常數：

`PI`(3.141...)、`E`(2.718...)、`TAU`(2倍 `PI`，6.283...)

以上這些方法和欄位，在其他程式語言中也基本上都有提供

字串

在 Java 中，字串就是 `java.lang.String` 類別的實例
其有許多動態方法，如：`int length()`、`String strip()`

`String[] split(String regex)`

`String substring(int beginIndex, int endIndex)`

`boolean startsWith(String prefix)`

`boolean endsWith(String suffix)`

`boolean contains(String s)`、`char charAt(int index)`


`String replace(String target, String replacement)`

在大部分程式語言中，字串通常都有這些方法

字串

```
import java.util.Scanner;

// 找整數中連續 n 個數字和的最大值
public class Main3 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int count = scanner.nextInt();
        String maxString = "";
        int max = -1;
        String string = scanner.next();
        for (int i = 0; i <= string.length() - count; i++) {
            int sum = 0;
            for (int j = 0; j < count; j++) {
                sum += Character.getNumericValue(string.charAt(i + j));
            }
            if (sum > max) {
                max = sum;
                maxString = string.substring(i, i + count);
            }
        }
        System.out.println(maxString + "," + max);
    }
}
```



```
5
2147483647
74836,28      console
```

java

隨機

在 Java 中，可以使用 `java.util.Random` 來生成隨機數
其建構子有 `Random()` 和 `Random(long seed)`，其動態方法有
`xxx nextXxx()`、`xxx nextXxx(xxx upperBound)`
`xxx nextXxx(xxx lowerBound, xxx upperBound)`
其中 `xxx` 為基本資料型別
`Random` 為偽隨機(pseudorandom)生成器
即 `Random` 在同一種子(seed)所生成的隨機數序列相同
無參數建構子會使用當前 `unix` 時間作為種子



隨機

`java.lang.Math` 的
靜態方法 `double random()`
則可以生成
在半開區間 `[0, 1)` 間的隨機數
其內部也是使用了 `Random`

```
import java.util.Arrays;
import java.util.Random;

public class Main1 {
    public static void main(String[] args) {
        double[] arr1 = new double[5];
        double[] arr2 = new double[5];
        double[] arr3 = new double[5];
        double[] arr4 = new double[5];

        Random random1 = new Random();
        Random random2 = new Random(0);
        Random random3 = new Random(0);

        for (int i = 0; i < 5; i++) {
            arr1[i] = random1.nextDouble();
            arr2[i] = random2.nextDouble();
            arr3[i] = random3.nextDouble();
            arr4[i] = Math.random();
        }

        System.out.println(Arrays.toString(arr1));
        System.out.println(Arrays.toString(arr2));
        System.out.println(Arrays.toString(arr3));
        System.out.println(Arrays.toString(arr4));
    }
}
```

java

```
[0.5598715847357625, 0.450838353119576, 0.5623495846152963, 0.21469423361943507, 0.290874720158952]
[0.730967787376657, 0.24053641567148587, 0.6374174253501083, 0.5504370051176339, 0.5975452777972018]
[0.730967787376657, 0.24053641567148587, 0.6374174253501083, 0.5504370051176339, 0.5975452777972018]
[0.3564439755039498, 0.546236143641482, 0.7969334809641179, 0.4668859506985015, 0.9069712317195023]
```

output