

# 空值與參考

TYIC 桃高資訊社

# null

在 Java 中，若欄位的型別不是基本資料型別  
則會在使用時就有可能會報錯，且為執行時期錯誤  
這是因為如果欄位的型別不是基本資料型別時，預設值為 **null**  
**null** 代表空值(空指標，**null pointer**)，也就是沒有這個物件  
理所當然的，沒有物件就沒辦法進行操作，也就引發錯誤

```
public class Main {  
    public static Integer integer;  
  
    public static void main(String[] args) {  
        System.out.println(integer + 6);  
        // Exception in thread "main" java.lang.NullPointerException:  
        // Cannot invoke "java.lang.Integer.intValue()"   
        // because "Main.integer" is null  
        // at Main.main(Main.java:5)  
    }  
}
```

java

# null

**null** 雖然不是任何一個類別或其子類別的實例  
但任何可以填入物件的地方都可以填入 **null**，表示物件不存在  
而因為編譯器和 IDE 可能檢查不出對 **null** 進行操作的問題  
所以 **null** 是個非常危險的存在，盡量不要使用  
空指標的發明人東尼·霍爾(Tony Hoare)在 2009 年曾說過：

*I call it my billion-dollar mistake.*

*It was the invention of the null reference in 1965.*

*我在1965年發明了空指標，*

*這是個十億美元的錯誤。*

*- Tony Hoare (2009)*

# 空值檢查

必須做好空值檢查(`null check`)，避免天外飛來一個 `null`  
欲檢查一物件是否為 `null`，可以直接使用比較運算子

```
public class Main {  
    public static void main(String[] args) {  
        Person person1 =  
            new Person(new PersonData("芳大同", 30));  
        System.out.println(person1);  
        Person person2 = new Person(null);  
        System.out.println(person2);  
    }  
}  
  
record PersonData(String name, int age) {  
}
```

```
Person{name=芳大同,age=30}  
Person{name=,age=0}
```

output

```
class Person {  
    PersonData data;  
  
    Person(PersonData data) {  
        if (data == null) {  
            data = new PersonData("", 0);  
        }  
        this.data = data;  
    }  
  
    @Override  
    public String toString() {  
        return "Person{" +  
            "name=" + data.name() +  
            ",age=" + data.age() +  
            '}';  
    }  
}
```



java


# 記憶體分配

當我們宣告變數或創建新實例時  
電腦就會分配記憶體空間讓我們儲存  
在 Java 中

8 個基本資料型別的記憶體大小視不同的 Java 虛擬機而定  
通常為 4 或 8 Bytes(位元組)  
其中 1 Byte 為 8 bits(位元)  
物件的記憶體大小通常與成員有關  
但實際上也是視不同的 Java 虛擬機而定

# 參考

記憶體每一格都有一個代表位址  
稱為記憶體位址(**memory address**)  
變數就是一個指向記憶體位址的東西  
變數讀取在獲取該記憶體位址的內容  
變數賦值即將內容寫入該記憶體位址  
Java 中，若變數型別為基本資料型別  
則在傳遞變數時傳遞的是值  
而若變數型別不是基本資料型別  
則在傳遞變數時傳遞的是記憶體位址



```
import java.util.Arrays;

public class Main {
    public static void main(String[] args) {
        int[] a = new int[]{1};
        int[] b = new int[]{a[0]};
        int[] c = a;
        int[] d = Arrays.copyOf(a, a.length);
        a[0]++;
        System.out.println(a[0]);
        System.out.println(b[0]);
        System.out.println(c[0]);
        System.out.println(d[0]);
    }
}
```

java

```
2
1
2
1
```

output

# 參考

假設下方的程式碼執行時記憶體配置如下：

記憶體內容	int[]{1}	int[]{1}	int[]{1}
記憶體位址	0xF000	0xF008	0xF010

- a 變數創建新物件，指向 0xF000
- a[0] 指向 int[] 的元素，傳遞值
- b 變數創建新實例，指向 0xF008
- c 變數參考物件 a，指向 0xF000
- d 變數創建新物件，指向 0xF010

```
import java.util.Arrays;

public class Main {
    public static void main(String[] args) {
        int[] a = new int[]{1};
        int[] b = new int[]{a[0]};
        int[] c = a;
        int[] d = Arrays.copyOf(a, a.length);
        a[0]++;
        System.out.println(a[0]);
        System.out.println(b[0]);
        System.out.println(c[0]);
        System.out.println(d[0]);
    }
}
```

2  
1  
2  
1 output

java

# 指標

指標就是一個指向記憶體位址的東西

一般的指標可以進行記憶體位址的運算，如往後移 4 Bytes

而變數就是一個不可以進行記憶體位址運算的指標

指標指向一個記憶體位址稱為參考(reference)

讀取指標指向的記憶體位址的內容稱為解除參考(dereference)

但因為 Java 的記憶體完全由 Java 虛擬機管理

所以 Java 中並沒有一般的指標，只有變數和空指標



# 陣列

陣列在記憶體中是連續的

也就是陣列中每個元素的記憶體位址皆相鄰

所以若要存取陣列中的元素

只需要對記憶體位址進行簡單的數學運算即可：

已知陣列元素記憶體大小為  $s$ ，索引值  $0$  元素之記憶體位址為  $a$

則陣列索引值  $n$  元素之記憶體位址即為  $a + s \times n$

陣列內容	1	2	3	4	5	6
陣列索引值	0	1	2	3	4	5
記憶體位址	0xF000	0xF008	0xF010	0xF018	0xF020	0xF028