




陣列與字串類別

學習目標

- 陣列
 - 字串類別
- 

陣列 (Array)

陣列是相同類型的資料按順序組成的一種複合資料類型。通過陣列名稱加陣列指標來使用陣列中的資料。指標從0開始。

宣告陣列

宣告陣列包括陣列的名字、陣列包含的元素的資料類型。

宣告**一維陣列**有下列兩種格式：

陣列元素類型 陣列名稱[];
陣列元素類型 [] 陣列名稱;

宣告**二維陣列**有下列兩種格式：

陣列元素類型 陣列名稱[][];
陣列元素類型 [] [] 陣列名稱;

建立陣列

- 宣告陣列僅僅是給出了陣列名稱和元素的資料類型，要想使用陣列還必須為它分配記憶體空間，即創建陣列。
- 在為陣列分配記憶體空間時必須指明陣列的長度。格式如下：
陣列名稱字 =new 陣列元素的類型[陣列元素的個數];

例如：

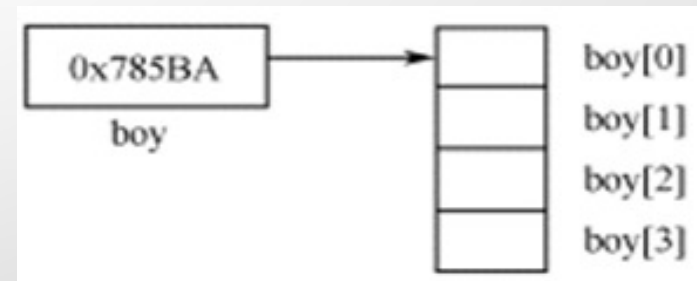
```
boy = new float[4];
```

宣告陣列和創建陣列可以一起完成，如

```
float boy[] = new float[4];
```

```
int man[ ][ ] = new int[3][4];
```

為陣列分配記憶體空間後，陣列boy獲得4個用來存放float類型資料的記憶體空間。



陣列 (Array)

陣列的創建和初始化

- 陣列元素的預設初始值與其類型有關，和相同類型的成員變數的預設初始值是相同的。

元素類型	初始值	元素類型	初始值
byte	0	short	0
int	0	long	0L
float	0.0F	double	0.0
char	'u0000'	boolean	false
物件引用	null		

陣列元素的使用

- 一維陣列通過指標符訪問自己的元素，如boy[0]， boy[1]等。
- 需要注意的是指標從0開始，因此，陣列若是7個元素，指標到6為止，如果你將來使用指標超過6將會發生異常。
- 二維陣列也通過指標符訪問自己的元素，如A[0][1]， A[1][2]等 。指標也是從0開始。

陣列的初始化

- 創建陣列後，系統會給每個陣列元素一個預設的值，如，float型是0.0。
- 我們在宣告陣列時同時也還可以給陣列的元素一個初始值，
- 如：

```
float boy[]={ 21.3F, 23.89F, 2.0F, 23F, 778.98F};
```

陣列的呼叫

- 陣列屬於呼叫型變數，因此兩個相同類型的陣列如果具有相同的呼叫，它們就有完全相同的元素。

例如 `INT [] A={1,2,3},B={4,5};`
 `A=B;`

- A中存放的呼叫就和B的相同，這時系統將釋放最初分配給陣列A的元素，
- 使得A的元素和B的元素相同，即A[0]，A[1]就是B[0]，B[1]，而最初分配給陣列A的三個元素已不復存在

用於陣列的for 迴圈

語法：

```
for(宣告語句 : 運算式) {  
  //代碼句子  
}
```

宣告語句：

宣告新的區域變數，其值與陣列元素的值相等，變數的類型和陣列元素的類型相同，其作用域限定在迴圈語句區塊內。

運算式：

陣列名稱，或者是返回值為陣列的方法。

用於陣列的for 迴圈

```
package ch5_1;
public class ch5_1 {
    public static void main(String[] args) {
        int[] numbers = { 10, 20, 30, 40, 50 };
        for (int x : numbers) {
            System.out.print(x);
            System.out.print(",");
        }
        System.out.print("\n");
        String[] names = { "字串1", "字串2", "字串3", "字串4" };
        for (String nag: names) {
            System.out.print(nag);
            System.out.print(",");
        }
    }
}
```

結果:

10,20,30,40,50,
字串1,字串2,字串3,字串4,

一維數字陣列

```
package ch5_2;
public class ch5_2 {
    public static void main(String[] args) {
        int i, sum = 0;
        double average, total = 0.0; // 變數宣告
        int[] temp; // 宣告陣列變數
        int[] grades = {87, 78, 95}; // 建立int陣列
        double[] sales = new double[4]; // 建立double陣列
        sales[0] = 145.6; sales[1] = 178.9;
        sales[2] = 197.3; sales[3] = 156.7;
        temp = grades;
        // 使用迴圈顯示陣列值和計算總和
        for (int ele : temp) {
            sum += ele;
            System.out.print("| " + ele);
        }
    }
}
```

```
System.out.println("\n成績總分: " + sum);
// 使用迴圈顯示陣列值和計算平均
for ( i=0; i < sales.length; i++) {
    total += sales[i];
    System.out.print("| " + sales[i]);
}
System.out.println("\n業績總和: " + total);
average = total/(double)sales.length;
System.out.println("平均業績: " + average);
}
```

結果:

```
| 87| 78| 95
成績總分: 260
| 145.6| 178.9| 197.3| 156.7
業績總和: 678.5
平均業績: 169.625
```

一維字串陣列

```
public class ch5_3 {  
    public static void main(String[] args) {  
        // String物件陣列  
        String[] names = new String[5];  
        names[0] = "楊過";   names[1] = "小龍女";  
        names[2] = "金庸";   names[3] = "江小魚";  
        names[4] = "陳浩南";  
        // 使用foreach迴圈顯示陣列內容  
        for ( String name: names ) {  
            System.out.print( '\"'+name+\"\"(字串長度:");  
            System.out.println(name.length() + ")");  
        }  
    }  
}
```

結果:

```
"楊過"(字串長度:2)  
"小龍女"(字串長度:3)  
"金庸"(字串長度:2)  
"江小魚"(字串長度:3)  
"陳浩南"(字串長度:3)
```

length的使用

- 一維陣列

“陣列名稱字. length”的值就是陣列中元素的個數；

```
float [] a=new float[12];
```

a. length的值12;

- 二維陣列

“陣列名稱字. length”的值是它含有的一維陣列的個數

```
int [][] b=new int[3][6];
```

b. length的值是3。

length的使用

```
package ch5_5;
public class ch5_5 {
public static void main(String[] args) {
    int i, sum = 0; // 變數宣告
    double average, total = 0.0;
    int[] temp; // 宣告陣列變數
    // 建立int陣列
    int[] grades = {87, 78, 95};
    // 建立double陣列
    double[] sales = new double[4];
    sales[0] = 145.6; sales[1] = 178.9;
    sales[2] = 197.3; sales[3] = 156.7;
    temp = grades;
    // 使用迴圈顯示陣列值和計算總和
    for (int ele : temp) {
        sum += ele;
        System.out.print("/ " + ele);
    }
    System.out.println("\n成績總分: " + sum);
    for (i=0; i < sales.length; i++) {
        // 使用迴圈顯示陣列值和計算平均
        total += sales[i];
        System.out.print("/ " + sales[i]);
    }
}
```

```
System.out.println("\n業績總和: " + total);
average = total/(double)sales.length;
System.out.println("平均業績: " + average);
```

結果:

```
| 87 | 78 | 95
成績總分: 260
| 145.6 | 178.9 | 197.3 | 156.7
業績總和: 678.5
平均業績: 169.625
```

二維陣列

- 多維陣列
 - 與一維陣列宣告方式相同，以 `[]` 數目代表維度
 - 例：`int[][] xxx = new int[2][3];` *//2*3 array*
`int[][][] xxx = new int[2][2][2];` *//2*2*2*
 - 多維陣列其實就是陣列中的陣列
- 多維陣列宣告
 - `int[][] x = new int[2][5];`

x[0][0]	x[0][1]	x[0][2]	x[0][3]	x[0][4]
x[1][0]	x[1][1]	x[1][2]	x[1][3]	x[1][4]

`int x[2][5]`

二維陣列

- 多維陣列的初始化
 - 多維陣列也可以如一維陣列般，一邊宣告一邊給初值：

```
String[][] Month = {"January", "31"},  
                    {"February", "28"},  
                    {"March", "31"},  
                    {"April", "30"},  
                    {"May", "31"},  
                    {"June", "30"},  
                    {"July", "31"},  
                    {"August", "31"},  
                    {"September", "30"},  
                    {"October", "31"},  
                    {"November", "30"},  
                    {"December", "31"}};
```

二維陣列

```
package ch5_4;
public class ch5_4 {
public static void main(String[] args) {
    // 變數宣告
    int i, j, total = 0, sum;
    int[][] scores = { { 54, 68 }, // 建立二維陣列
                       { 67, 78 },
                       { 89, 93 } };

    // 使用巢狀迴圈計算總和
    for ( j = 0; j < scores.length; j++ ) {
        sum = 0;
        for ( i = 0; i < scores[j].length; i++ ) {
            System.out.print(scores[j][i] + " ");
            sum += scores[j][i];
            total += scores[j][i];
        }
        System.out.println("==>總得分: " + sum);
    }
    System.out.println("得分總和: " + total);
    // 建立二維物件陣列
    double[][] sales = new double[4][];
```


二維陣列

```
for ( i = 0; i < sales.length; i++)
    sales[i] = new double[2];
sales[0][0] = 123.4;    sales[0][1] = 143.5;
sales[1][0] = 142.3;    sales[1][1] = 198.4;
sales[2][0] = 234.6;    sales[2][1] = 200.5;
sales[3][0] = 167.1;    sales[3][1] = 150.4;
System.out.println("業績報表:");
System.out.println("季\t第一年\t第二年");
// 使用巢狀迴圈顯示陣列值
for ( j = 0; j < sales.length; j++ ) {
    System.out.print("第" + (j+1) + "季\t");
    for ( i = 0; i < sales[i].length; i++ )
        System.out.print(sales[j][i] + "\t");
    System.out.println();
}
}
```

結果:

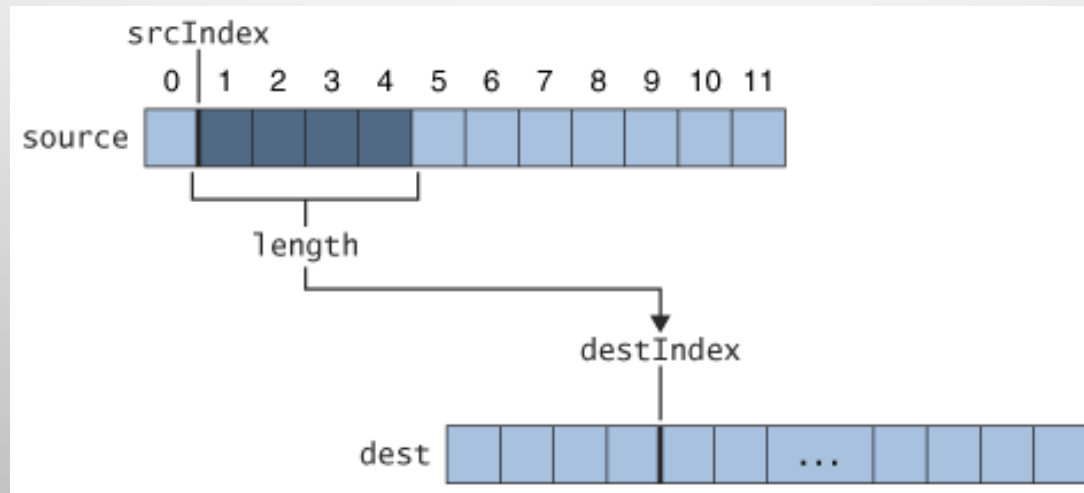
```
54 68 ==>總得分: 122
67 78 ==>總得分: 145
89 93 ==>總得分: 182
得分總和: 449
業績報表:
季      第一年    第二年
第1季    123.4    143.5
第2季    142.3    198.4
第3季    234.6    200.5
第4季    167.1    150.4
```

陣列複製

- **System** 的類別方法 **arraycopy()**

用法:

System.arraycopy(來源, 起始索引, 目的, 起始索引, 複製長度)



陣列複製

```
package ch5_6;
public class ch5_6 {
public static void main(String[] args) {
    int[] arr1 = { 11, 21, 32, 44, 35, 66, 17, 38 };
    int[] arr2 = new int[8];
    // System.arraycopy(來源, 起始索引, 目的, 起始索引, 複製長度)
    System.arraycopy(arr1, 0, arr2, 0, arr1.Length);
    for (int i : arr2) {
        System.out.print(i + " ");
    }
}
```

結果:

11 21 32 44 35 66 17 38

陣列複製

- 如果使用**JDK6**以上， `java.util.Arrays.copyOf()`方法，可複製指定的陣列。
- 用法:

`copyOf(int[] original, int newLength)`

`original`—原始陣列

`newLength`— 複製的長度

陣列複製

```
package ch5_7;
import java.util.Arrays;
public class ch5_7 {
    public static void main(String[] args) {
        int[] org = new int[] { 1, 2, 3, 4, 5 };
        System.out.println("原始陣列 " + "");
        for (int i = 0; i < org.length; i++) {
            System.out.print(org[i] + " ");
        }
        int[] copy = Arrays.copyOf(org, 7);
        System.out.println("\n原始陣列的複製:");
        for (int i = 0; i < copy.length; i++) {
            System.out.print(copy[i] + " ");
        }
    }
}
```

結果:

原始陣列

1 2 3 4 5

原始陣列的複製:

1 2 3 4 5 0 0

String物件

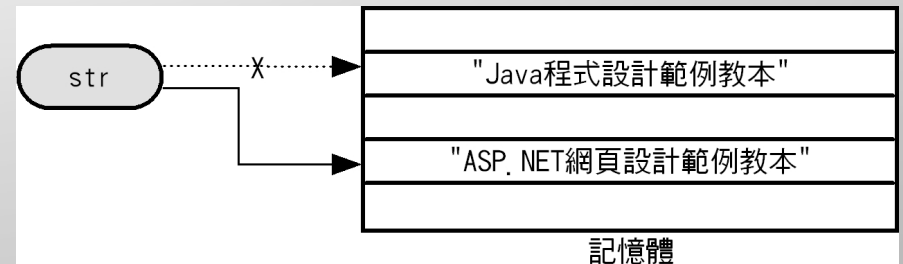
- Java字串就是String物件提供String字串類別和StringBuffer字串緩衝區類別來處理字串資料。
- String字串類別
建立的字串物件是final類別，即不能被繼承。
- StringBuffer字串緩衝區類別
字串物件能夠新增、插入和刪除字串內容。

String字串類別-字串是一種參考資料型態

- **Java**字串就是**String**物件，屬於一種參考資料型態，所以字串內容並不能更改，也就是說，一旦建立字串後，就無法改變其值，只能重新指定成新的字串值或另一個字串變數
- 如下所示：

```
String str = "Java程式設計範例教本";
```

```
str = "ASP.NET網頁設計範例教本";
```



String字串類別-建構式

- 字串使用字元集合與「"」雙引號括起來建立
`String str = "Java程式設計範例教本";`
- String物件使用建構子來建立String物件
`String str2 = new String("程式設計範例教本");`

使用new運算元來呼叫類別的建構式方法
使用字串來建立String物件。

字串類與字元陣列的關係

- 字元陣列不是字串類。
- 字元陣列可以轉化為字串類物件。
- 在輸出時，可以用陣列名稱呼叫整個陣列的所有元素。

字串類別的常用方法

String類別的方法

- `int length ()`

作用：返回字串的長度，即字串中包含的字元數。

說明：由於Java採用的是Unicode編碼，因此一個漢字的長度為1。

- `boolean equals (String AnotherString)`

作用：判斷字串是否相同

字串類別的常用方法

```
package ch5_8;
public class ch5_8 {
public static void main(String[] args) {
    // 陣列宣告
    char[] charArr = { ' ', 'J', 'a', 'v', 'a', ' ' };
    StringBuffer sb = new StringBuffer("use");
    String str = " JAVA "; // 使用String類別宣告字串
    // 使用建構子建立字串物件
    String str1, str2;
    str1 = new String(charArr); // 使用字元陣列
    str2 = new String("程式設計範例教本");
    System.out.println("str字串:\"\" + str + "\"");
    System.out.println("str1字串:\"\" + str1 + "\"");
    // 顯示字串長度
    System.out.print("str1長度:"+str1.length());
    System.out.println("/str2長度:"+str2.length());
    // 字串str與str1是否相等
    System.out.print("str與str1字串是否相等: ");
    System.out.println(str.equals(str1));
}
}
```

結果:

```
str字串:"  JAVA "
str1字串:" Java "
str1長度:6/str2長度:8
str與str1字串是否相等: false
```

- `boolean equalsIgnoreCase (String AnotherString)`

作用：判斷字串是否相同，忽略大小寫

- `boolean toUpperCase ()`

作用：將字元轉換為大寫。

- `boolean toLowerCase ()`

作用：將字元轉換為校小寫。

- `boolean trim ()`

作用：刪除字串前後空格。

說明：在進行字串比較時空格會對比較結果產生影響，因此有時為了正確比較，忽略空格的作用時可以用該方法。

```

package ch5_9;
public class ch5_9 {
    public static void main(String[] args) {
        char[] charArr = { ' ', 'J', 'a', 'v', 'a', ' ' };
        String str = " JAVA "; // 使用String類別宣告字串
        // 使用建構子建立字串物件
        String str1, str2, str3, str4;
        str1 = new String(charArr); // 使用字元陣列
        str2 = new String("程式設計範例教本");
        System.out.println("str字串:\"\" + str + "\"");
        System.out.println("str1字串:\"\" + str1 + "\"");
        System.out.println("str2字串:\"\" + str2 + "\"");
        System.out.print("str1長度:" + str1.length());
        System.out.println("/str2長度:" + str2.length());
        System.out.print("str1轉小寫:" + str1.toLowerCase());
        System.out.println("/str1轉大寫:" + str1.toUpperCase());
        System.out.print("str1刪除空白字元str1.trim(): ");
        System.out.println(str1.trim());
        System.out.print("str與str1字串是否相等: ");
        System.out.println(str.equals(str1));
        System.out.print("str與str1是否相等-不分大小寫: ");
        System.out.println(str.equalsIgnoreCase(str1));
    }
}

```

結果:

```

str字串:" JAVA "
str1字串:" Java "
str2字串:"程式設計範例教本"
str1長度:6/str2長度:8
str1轉小寫: java /str1轉大寫: JAVA
str1刪除空白字元str1.trim(): Java
str與str1字串是否相等: false
str與str1是否相等-不分大小寫: true

```

- `char charAt(int Position)`
作用：返回字串中指定位置字元
- `String substring (int start)`
作用：從指定位置開始取字串的子串。
- `String substring (int start, int end)`
作用：從指定位置開始取字串的指定區間的字串（不包括end位置的字元）
- `int indexOf (String aString)`
作用：物件字串中第一次出現aString的位置，如果字串中不包含指定子串，則返回-1。
- `int lastIndexOf (String aString)`
作用：物件字串中最後一次出現aString的位置，如果字串中不包含指定子串，則返回-1。

```

package ch5_10;
public class ch5_10 {
public static void main(String[] args) {
    char[] charArr = { ' ', 'J', 'a', 'v', 'a', ' ' };
    String str = " JAVA "; // 使用String類別宣告字串
    String str1, str2, str3, str4;
    str1 = new String(charArr); // 使用字元陣列
    str2 = new String("程式設計範例教本");
    System.out.println("str字串:\\" + str + "\\");
    System.out.println("str1字串:\\" + str1 + "\\");
    System.out.println("str2字串:\\" + str2 + "\\");
    System.out.print("str1 indexOf('a', 2): ");
    System.out.println(str1.indexOf('a', 2));
    System.out.print("str1 lastIndexOf('b', 2): ");
    System.out.println(str1.lastIndexOf('b', 2));
    System.out.print("英文str1.charAt(3): ");
    System.out.println(str1.charAt(3));
    System.out.print("中文str2.substring(2, 6): ");
    System.out.println(str2.substring(2, 6));
}
}

```

結果:

```

str字串:" JAVA "
str1字串:" Java "
str2字串:"程式設計範例教本"
str1 indexOf('a', 2): 2
str1 lastIndexOf('b', 2): -1
英文str1.charAt(4): v
中文str2.substring(2, 6): 設計範例

```

字串類別的常用方法

- `int indexOf (String aString , int start)`
作用：物件字串中在`start`之後第一次出現`aString`的位置，如果`start`之後不包含指定子串，則返回-1。
- `int compareTo (String aString)`
作用：物件字串與指定字串比較，若前者大返回正數、相等返回0；後者大返回負數。
- `int compareToIgnoreCase (String aString)`
作用：物件字串與指定字串比較，不區分大小寫；


```

package ch5_11;
public class ch5_11 {
public static void main(String[] args) {
    char[] charArr = { ' ', 'J', 'a', 'v', 'a', ' ' };
    String str = " JAVA "; // 使用String類別宣告字串
    String str1, str2, str3, str4;
    str1 = new String(charArr); // 使用字元陣列
    str2 = new String("程式設計範例教本");
    System.out.println("str字串:\"\" + str + "\"");
    System.out.println("str1字串:\"\" + str1 + "\"");
    System.out.println("str2字串:\"\" + str2 + "\"");
    System.out.print("英-字元indexOf(\"'a'\", 2): ");
    System.out.println(str1.indexOf('a', 2));
    System.out.print("英-字元lastIndexOf(\"'b'\", 2): ");
    System.out.println(str1.lastIndexOf('b', 2));
    System.out.print("中-字串indexOf(\"範例\"): ");
    System.out.println(str2.indexOf("範例"));
    System.out.print("中-字串lastIndexOf(\"範例\"): ");
    System.out.println(str2.lastIndexOf("範例"));
    System.out.print("比較str與str1字串: ");
    System.out.println(str.compareTo(str1));
    System.out.print("比較str與str1字串-不分大小寫: ");
    System.out.println(str.compareToIgnoreCase(str1));
}
}

```

結果:

```

str字串:" JAVA "
str1字串:" Java "
str2字串:"程式設計範例教本"
英-字元indexOf('a', 2): 2
英-字元lastIndexOf('b', 2): -1
中-字串indexOf("範例"): 4
中-字串lastIndexOf("範例"): 4
比較str與str1字串: -32
比較str與str1字串-不分大小寫: 0

```

字串運算與轉換

- 用 “+” 操作符對兩個字串所做的運算。
- “+” 可以連接任意多個字串
- “+” 可以將字串與其他類型進行連接並自動完成其他類型到字串類型的轉換。

StringBuffer類別

- StringBuffer類別物件是可變物件，當對它進行修改的時候不會像String那樣重新建立物件，而是在原來的物件空間內進行操作。
- StringBuffer類別物件只能通過構造函數來建立。雖然在任意時間點上它都包含某種特定的字元序列，但通過某些方法調用可以改變該序列的長度和內容。

StringBuffer所在套件

- `java.lang`，該類繼承自`java.lang.Object`，實現了`Serializable`、`CharSequence`介面。

StringBuffer建構式函數

- `public StringBuffer()`：創建一個空的`StringBuffer`類別的物件。
- `public StringBuffer(int length)`：創建一個長度為參數`length`的`StringBuffer`類別的物件。
- **※ 特別提示**：如果參數`length`小於0，將觸發`NegativeArraySizeException`異常。
- `public StringBuffer(String str)`：用一個已存在的字串來創建`StringBuffer`類別的物件。

StringBuffer類別的方法

toString方法

格式：`public String toString()`

功能：轉換為String類別物件並返回。

由於大多數類中關於顯示方法的參數多為String類別物件，所以經常要將StringBuffer類對象轉換為String類別物件，再將它的值顯示出來。

StringBuffer類別的方法

- append方法

格式1：public StringBuffer append(boolean b)

格式2：public StringBuffer append(char c)

格式3：public StringBuffer append(int i)

格式4：public StringBuffer append(long l)

格式5：public StringBuffer append(float f)

格式6：public StringBuffer append(double d)

功能：

分別將boolean、char、int、long、float和double 6種類型的變數格式化成字串之後追加到StringBuffer類別物件的後面。

StringBuffer類別的方法

- append方法

格式7：`public StringBuffer append(String str)`

功能：將字串常量`str`追加到StringBuffer類別物件的後面。

格式8：`public StringBuffer append(char str[])`

功能：將字元陣列`str`追加到StringBuffer類別物件的後面。

格式9：`public StringBuffer append(char str[], int
offset, int len)`

功能：將字元陣列`str`，從第`offset`個開始取`len`個字元，追加到StringBuffer類別物件的後面。

StringBuffer類別的方法

insert方法

- 格式1：public StringBuffer insert(int offset, boolean b)
 - 格式2：public StringBuffer insert(int offset, char c)
 - 格式3：public StringBuffer insert(int offset, int i)
 - 格式4：public StringBuffer insert(int offset, long l)
 - 格式5：public StringBuffer insert(int offset, float f)
 - 格式6：public StringBuffer insert(int offset, double d)
 - 格式7：public StringBuffer insert(int offset, String str)
 - 格式8：public StringBuffer insert(int offset, char str[])
-
- 功能：將boolean、char、int、long、float、double類型的變數、String類的物件或字元陣列插入到StringBuffer類的物件中的第offset個字元位置。

StringBuffer類別的方法

delete方法

- 格式：`public StringBuffer delete (int start, int end)`
- 功能：刪除當前StringBuffer類對象中以start開頭，以end結束的子串。

length方法

- 格式：`public int length()`
- 功能：這個方法返回字串變數的長度，用法與String類的length方法類似。

```

package ch5_12;
public class ch5_12 {
    public static void main(String[] args) {
        StringBuffer sb = new StringBuffer("程式");
        char[] charArr = { 'J', 'a', 'v', 'a' };
        System.out.println("原始字串內容: " + sb);
        sb.append('-'); // 字元
        System.out.println("新增字元\ '-\ ': " + sb);
        sb.append(charArr, 0, 4); // 字元陣列
        System.out.println("新增字元陣列: " + sb);
        sb.append("程式範例教本");
        System.out.println("新增字串: " + sb);
        sb.deleteCharAt(2);
        System.out.println("刪除第3個字: " + sb);
        sb.delete(0, 2);
        System.out.println("刪除前2個字: " + sb);
        sb.insert(6, "設計");
        System.out.println("插入中文字串: " + sb);
        sb.insert(4, " SE");
        System.out.println("插入英文字串: " + sb);
        sb.setCharAt(5, 'E');
        System.out.println("取代字元: " + sb);
    }
}

```

```

        sb.replace(0, 7, "JDK SE");
        System.out.println("取代字串: " + sb);
        sb.reverse(); // 反轉字串
        System.out.println("反轉後字串: " + sb);
    }
}

```

結果:

原始字串內容: 程式
 新增字元 '-': 程式-
 新增字元陣列: 程式-Java
 新增字串: 程式-Java程式範例教本
 刪除第3個字: 程式Java程式範例教本
 刪除前2個字: Java程式範例教本
 插入中文字串: Java程式設計範例教本
 插入英文字串: Java SE程式設計範例教本
 取代字元: Java EE程式設計範例教本
 取代字串 : JDK SE程式設計範例教本
 反轉後字串 : 本教例範計設式程ES KDJ