類別 class

喬逸偉 (Yiwei Chiao)

1 前言

之前介紹了 JavaScript 的 *literal objects* (用在 `ball` 物件)；這一章，將進入 JavaScript 的 class*。

這一章將利用 *class* 將**擊球板** `paddle` 和**磚塊** `brick` 都轉換成**物件**；同時，為了遊戲進行方便，也加上了**開始**，**暫停**，等控制按鈕。

目前專案執行畫面，如下圖 1：

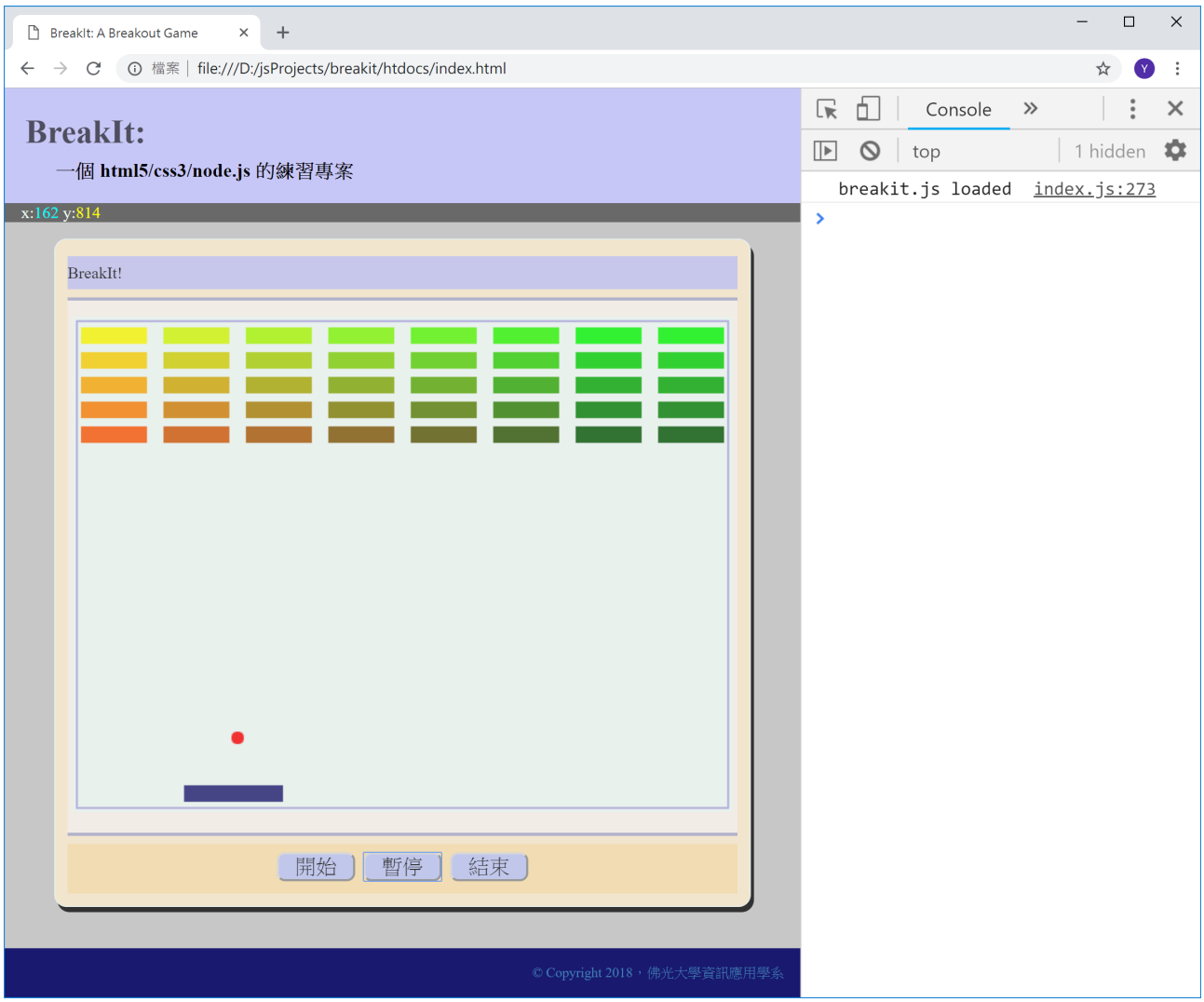

Figure 1: 移動的 paddle

1.1 JavaScript 的類別 (*class*)

先看遊戲內**磚塊** (*brick*) 目前的程式碼，共 36 行：

```
 1. class Brick {
 2.   constructor (x, y, color) {
 3.     this._x = x;
 4.     this._y = y;
 5.
 6.     this._width = 64;
 7.     this._height = 16;
 8.
 9.     this._color = color;
10.   }
11.
```

```
12.   get x() {
13.     return this._x;
14.   }
15.
16.   get y() {
17.     return this._y;
18.   }
19.
20.   get width() {
21.     return this._width;
22.   }
23.
24.   get height() {
25.     return this._height;
26.   }
27.
28.   paint (ctx) {
29.     ctx.save();
30.
31.     ctx.fillStyle = this._color;
32.     ctx.fillRect(this._x, this._y, this._width, this._height);
33.
34.     ctx.restore();
35.   }
36. };
```

這 36 行程式碼建立了一個**類別宣告** (*class declairation*)，並可以利用關鍵字 new 來建立真正的**物件實體** (*object instances*) 如，在 breakIt 物件裡新增的 reset 方法：

```
  let breakIt = {
    ...
 1.  reset: function () {
 2.    let width = 8;
 3.    let height = 5;
 4.
 5.    for (let x = 0; x < width; x ++) {
 6.      for (let y = 0; y < height; y++) {
 7.        this._bricks.push(new Brick(
 8.          (x * 80) + 8,
 9.          (y * 24) + 10,
10.          `rgb(${Math.floor(255 - 42.5 * x)}, ${Math.floor(255 - 42.5 *
11.        ));
12.      }
```

```
13.     }
14.
15.     this._paddle = new Paddle(272, 454);
16.
17.     this.paint();
18.   },
      ...
    };
```

1.1.1 Class

JavaScript 在**語法** (*syntax*) 上屬於 C 語言家族，受到 C 相當大的影響；宣告 class 也是利用 關鍵字 + 識別字 + {} 的方式。

```
 1. class Brick {
      ...
36. };
```

宣告了一個稱作 Brick 的**類別** (*class*)。

1.1.2 建構子 (constructor)

JavaScript 類別 (class) 的**建構函數** (constructor) 名字就直接稱作 constructor；

```
 2.   constructor (x, y, color) {
 3.     this._x = x;
 4.     this._y = y;
 5.
 6.     this._width = 64;
 7.     this._height = 16;
 8.
 9.     this._color = color;
10.   }
```

如上面的程式片段所示，constructor 可以接受參數，好在建構物件實體時直接初始化相對的屬性。

和 C++/C#/Java 不同，JavaScript 類別裡的屬性一樣**不**需要宣告；直接**賦值** (assign value)，JavaScript 就知道要為這個類別產生相應的屬性了；如第 3 到第 9 行所作的。注意，為了知道這個利用 **賦值** (assign value) 操作產生的屬性是給**物件實體** (*object instance*) 的，所以，前面都有 *this* 這個關鍵字。

這裡的屬性很簡單，基本上就是記錄**磚塊** (*brick*) 的 **左上角座標** (x, y)；**長寬** (width, height) 和 **顏色** (color)。

1.1.3 方法 (method)

宣告方法的文法和其它語言一樣，就是 識別字 () {}；如下：

```
28.   paint (ctx) {
29.     ctx.save();
30.
31.     ctx.fillStyle = this._color;
32.     ctx.fillRect(this._x, this._y, this._width, this._height);
33.
34.     ctx.restore();
35.   }
```

目前，Brick 類別就只有一個將**自己** (*this*) 繪出來的方法；注意它在查詢自己的**屬性**時，前面都加了 *this* 關鍵字，這樣 JavaScript 才知道去那兒找那些屬性。

1.1.4 getter 方法 (method)

物件導向 (object-oriented) 設計原理強調**封裝** (encapsulation)；因為**限制**資料的存取就可以簡化/減少資料不當存取的機會。

C++/C#/Java 利用 public/protected/private 等存取控制字來趨近這個目的；可是 JavaScript 沒有這些關鍵字。

另一方面，假定有一個屬性 x，代表 x 軸座標值；以 public/protected/private 這類的存取控制子來控制，x 要不是**可讀可寫**，要不就是**不可讀寫**；無法作到**唯讀** (*read-only*) 或**唯寫** (*write-only*)。

不難想到，很多情況下，其實真的只需要**讀取** (*read*) 而已；

因應這種觀察，有所謂的 getter/setter 函數被提出來。顧名思義，getter 函數就是**唯讀** (*read-only*)；而 setter 函數則是**唯寫** (*write-only*)；

以 Java 為例：

```
public class Point {
  private int x;
  private int y;

  public Point(int x, int y) {
      this.x = x;
      this.y = y;
  }

  // getter functions
  public int getX() {
```

```
        return x;
    }

    public int getY() {
        return y;
    }

    // setter functions
    public void setX(int x) {
        this.x = x;
    }

    public void setY(int y) {
        this.y = y;
    }
}
```

如上面程式碼，將**資料** (x, y) 設成 private，所以外界無法直接存取，再透過 getter/setter 函數來存取它們。

在 JavaScript 提供了更簡單的 getter/setter 方法。如：

```
 1. class Brick {
        ...
12.     get x() {
13.       return this._x;
14.     }
        ...
36. };
```

就定義了一個 getter 函數；而可以這麼使用：

```
    let brick = new Brick( 100, 100, 'red');

    console.log(`brick.x = ${brick.x}`);
```