Appendix

Canvas API

B

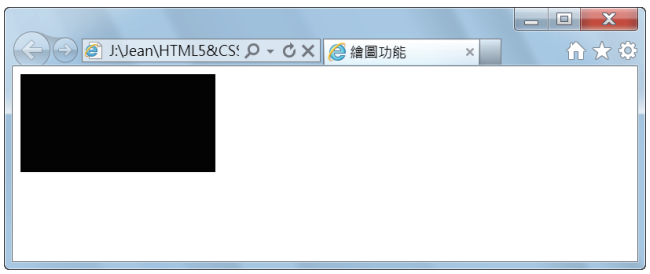
B-1 HTML5 的繪圖功能

我們在第 5-4 節中有簡單示範過如何使用 <canvas> 元素在網頁上建立繪圖區並進行繪圖，而在本章中，我們將進一步介紹 HTML5 提供的 Canvas API 及相關的程式設計技巧。由於 Canvas API 的功能非常豐富，包括線條樣式、填滿樣式、文字樣式、建立路徑、繪製矩形、繪製圖像、漸層、陰影等，因此，我們會介紹其中一些較具有代表性的 API，至於完整的 API，有興趣的讀者可以自行參考官方文件 HTML Canvas 2D Context (http://dev.w3.org/html5/2dcontext/)。

下面是一個例子，它會在網頁上建立繪圖區，然後填滿一個矩形。

```
01:<!doctype html>
02:<html>
03:  <head>
04:    <meta charset="utf-8">
05:    <title>繪圖功能</title>
06:  </head>
07:  <body>
08:    <canvas id="myCanvas" width="200" height="100"></canvas>
09:    <script>
10:      var canvas = document.getElementById("myCanvas");
11:      var context = canvas.getContext("2d");
12:      context.fillRect(0,0,200,100);
13:    </script>
14:  </body>
15:</html>
```

<\ 附錄 B\canvas1.html>



- 08：使用 <canvas> 元素在網頁上插入一塊寬度為 200 像素、高度為 100 像素的繪圖區 (註：若沒有指定大小，則繪圖區預設的寬度為 300 像素、高度為 150 像素)。
- 11：呼叫 <canvas> 元素的 getContext("2d") 方法取得 2D 繪圖環境。
- 12：呼叫繪圖環境的 fillRect(0,0,200,100) 方法將左上角座標為 (0, 0)、寬度為 200 像素、高度為 100 像素的矩形填滿色彩 (預設為黑色)，而此舉正好會填滿整個繪圖區，因為我們故意將矩形的大小設定成繪圖區的大小。

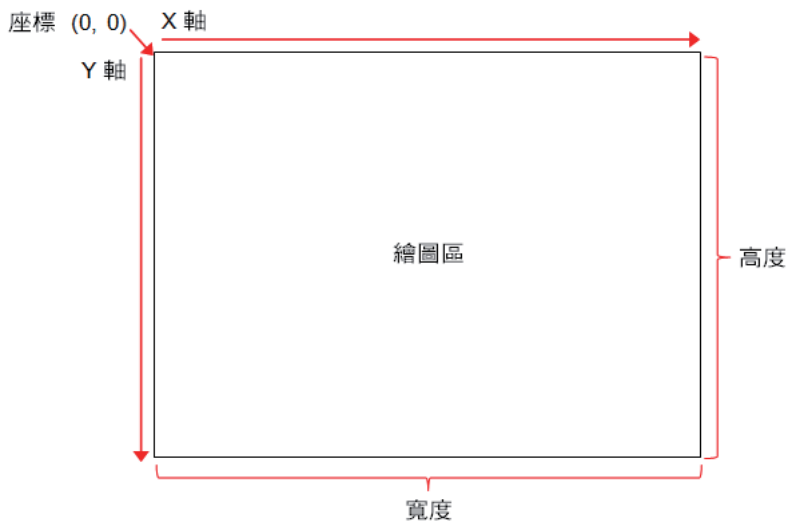
我們將在繪圖區進行繪圖的步驟歸納如下：

1. 使用 <canvas> 元素在網頁上插入繪圖區。

2. 呼叫 <canvas> 元素的 getContext("2d") 方法取得 2D 繪圖環境。

3. 呼叫繪圖環境的各種方法進行繪圖。

在進行繪圖之前，我們還要瞭解一下繪圖區的座標系統，原則上，座標 (0, 0) 是位於繪圖區的左上角，水平方向為 X 軸，垂直方向為 Y 軸，水平方向的寬度等於 <canvas> 元素的 width 屬性，垂直方向的高度等於 <canvas> 元素的 height 屬性，如下圖，要注意的是繪圖區沒有框線，預設為一塊空白的區域。



B-2 設定繪製樣式與填滿樣式

2D 繪圖環境提供了下列兩個屬性用來設定繪製樣式與填滿樣式：

- strokeStyle：繪製圖形時所使用的線條色彩或樣式，預設為 #000000 (黑色)。
- fillStyle：填滿圖形時所使用的色彩或樣式，預設為 #000000 (黑色)。

這兩個屬性的值可以是字串、CanvasGradients (漸層) 或 CanvasPatterns (圖樣)，字串必須是合法的 CSS 色彩，其語法如下，而漸層和圖樣留待第 B-4 節再做討論：

```
色彩名稱 | rgb(rr, gg, bb) | #rrggbb
```

- *色彩名稱*：以諸如 black、blue、gray、green、lime、maroon、navy、olive、orange、purple、red、silver、teal、white、yellow 等名稱指定色彩。
- rgb(*rr*, *gg*, *bb*)：以紅 (red)、綠 (green)、藍 (blue) 三原色的混合比例指定色彩，例如 rgb(100%, 0%, 0%) 表示紅 100%、綠 0%、藍 0%，也就是紅色。除了指定混合比例之外，我們也可以將紅、綠、藍三原色各自劃分為 0 ~ 255 共 256 個級數，改以級數來表示色彩，例如 rgb(100%, 0%, 0%) 在轉換成級數後會對應到 rgb(255, 0, 0)。
- *#rrggbb*：這是前一種指定方式的十六進位表示法，以 # 符號開頭，後面跟著三組十六進位數字，分別代表色彩的紅、綠、藍級數，例如 rgb(255, 0, 0) 在轉換成十六進位後會對應到 #ff0000。

下面是一個例子，它會將繪製樣式與填滿樣式分別設定成藍色和紅色。

```
<canvas id="myCanvas" width="400" height="300"></canvas>
<script>
  var canvas = document.getElementById("myCanvas");
  var context = canvas.getContext("2d");
  context.strokeStyle = "rgb(0, 0, 255)";
  context.fillStyle = "red";
</script>
```

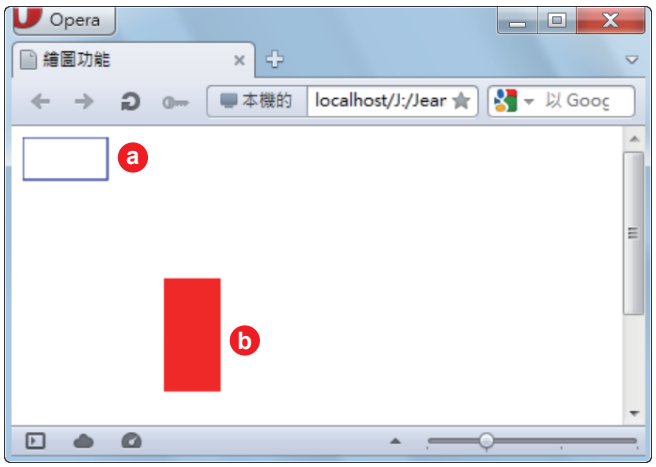
B-3 繪製矩形

2D 繪圖環境提供了下列幾個方法用來繪製矩形：

- strokeRect(*x*, *y*, *w*, *h*)：以目前的繪製樣式繪製矩形，該矩形的左上角座標為 (*x*, *y*)、寬度為 *w* 像素、高度為 *h* 像素。
- fillRect(*x*, *y*, *w*, *h*)：以目前的填滿樣式填滿矩形，參數的意義同上。
- clearRect(*x*, *y*, *w*, *h*)：清除矩形，參數的意義同上。

下面是一個例子 <\ 附錄 B\canvas2.html>。

```
<canvas id="myCanvas" width="400" height="300"></canvas>
<script>
  var canvas = document.getElementById("myCanvas");
  var context = canvas.getContext("2d");
  context.strokeStyle = "rgb(0, 0, 255)";
  context.strokeRect(0,0,60,30);
  context.fillStyle = "red";
  context.fillRect(100, 100, 40,80);
</script>
```



a 以藍色繪製矩形，左上角座標為 (0, 0)、寬度為 60 像素、高度為 30 像素

b 以紅色填滿矩形，左上角座標為 (100, 100)、寬度為 40 像素、高度為 80 像素

B-4 設定漸層與圖樣

2D 繪圖環境提供了下列幾個方法用來設定漸層：

- createLinearGradient(*x0*, *y0*, *x1*, *y1*)：從座標 (*x0*, *y0*) 往座標 (*x1*, *y1*) 建立直線變化的漸層。
- createRadialGradient(*x0*, *y0*, *r0*, *x1*, *y1*, *r1*)：以座標 (*x0*, *y0*) 為圓心、*r0* 為半徑的圓，和以 (*x1*, *y1*) 為圓心、*r1* 為半徑的圓之間建立圓形變化的漸層。
- addColorStop(*offset*, *color*)：設定漸層邊界的色彩，其中參數 *offset* 的值介於 0.0 ~ 1.0 之間，0.0 為漸層的起點，1.0 為漸層的終點，而參數 *color* 為漸層的色彩。

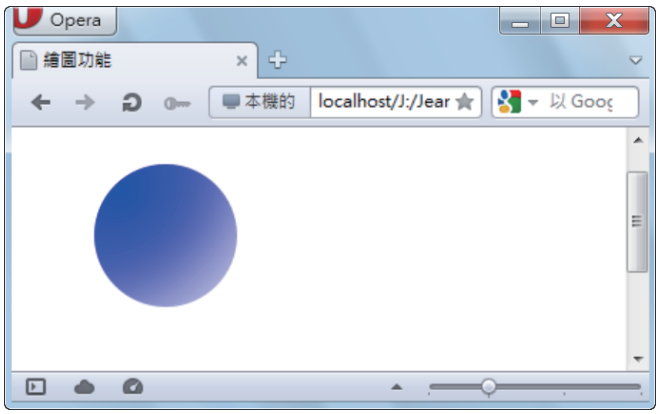
前兩個方法的傳回值都是一個 CanvasGradient 物件，可以指派給 strokeStyle 或 fillStyle 屬性，做為繪製樣式或填滿樣式。下面是一個例子 <\ 附錄 B\canvas3.html>，它會先建立一個藍白線性漸層，然後使用該漸層填滿矩形。

```
<canvas id="myCanvas" width="400" height="300"></canvas>
<script>
  var canvas = document.getElementById("myCanvas");
  // 取得 2D 繪圖環境
  var context = canvas.getContext("2d");
  // 建立線性漸層
  var gradient = context.createLinearGradient(0, 0, 100, 0);
  // 設定漸層起點為藍色
  gradient.addColorStop(0.0, "blue");
  // 設定漸層終點為白色
  gradient.addColorStop(1.0, "white");
  // 將填滿樣式指派為前面建立的漸層
  context.fillStyle = gradient;
  // 使用前面建立的漸層填滿矩形
  context.fillRect(0, 0, 100, 100);
</script>
```

下面是另一個例子 <\ 附錄 B\canvas4.html>，它會先建立一個藍白圓形漸層，然後使用該漸層填滿圓形。此處暫不解釋如何繪製圓形，詳見第 B-6-3 節的說明。

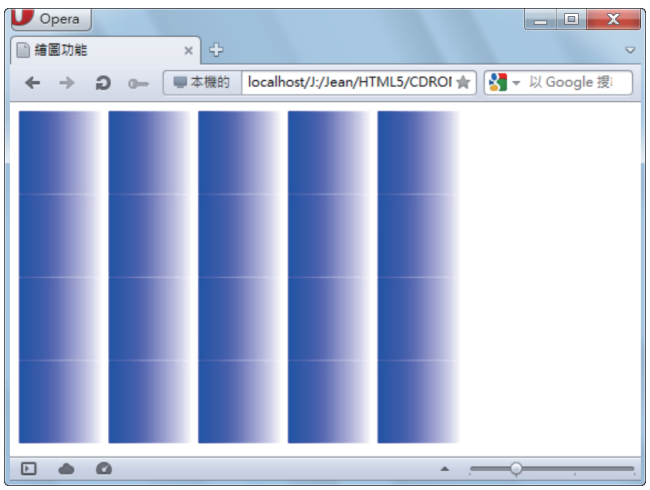
```
<canvas id="myCanvas" width="400" height="300"></canvas>
<script>
  var canvas = document.getElementById("myCanvas");
  // 取得 2D 繪圖環境
  var context = canvas.getContext("2d");
  // 建立圓形漸層
  var gradient = context.createRadialGradient(70, 70, 0, 20, 20, 200);
  // 設定漸層起點為藍色
  gradient.addColorStop(0.0, "blue");
  // 設定漸層終點為白色
  gradient.addColorStop(1.0, "white");
  // 將填滿樣式指派為前面建立的漸層
  context.fillStyle = gradient;
  // 使用前面建立的漸層填滿圓形
  context.arc(100, 100, 50, 0, 360 * Math.PI / 180, true);
  context.fill();
</script>
```

這個例子的瀏覽結果如下圖。



除了漸層之外，我們也可以指派圖樣給 strokeStyle 或 fillStyle 屬性，圖樣是一個 CanvasPattern 物件，可以呼叫 createPattern(*image*, *repetition*) 方法來產生，其中參數 *image* 為 <img>、<canvas> 或 <video> 元素的物件，而參數 *repetition* 有 repeat (雙向重複)、repeat-x (水平重複)、repeat-y (垂直重複)、no-repeat (不重複) 等值，預設為 repeat。下面是一個例子 <\ 附錄 B\canvas5.html>，它會先建立一個圖樣，然後使用該圖樣填滿整個繪圖區。

```
<canvas id="myCanvas" width="400" height="300"></canvas>
<script>
  var canvas = document.getElementById("myCanvas");
  var context = canvas.getContext("2d");
  // 建立 <img> 元素
  var image = document.createElement("img");
  // 待圖像載入完畢後，就建立圖樣並指派給填滿樣式，然後填滿整個繪圖區
  image.onload = function() {
    context.fillStyle = context.createPattern(this, "repeat");
    context.fillRect(0, 0, canvas.width, canvas.height);
  }
  // 讀取圖像
  image.src = "blue.jpg";
</script>
```

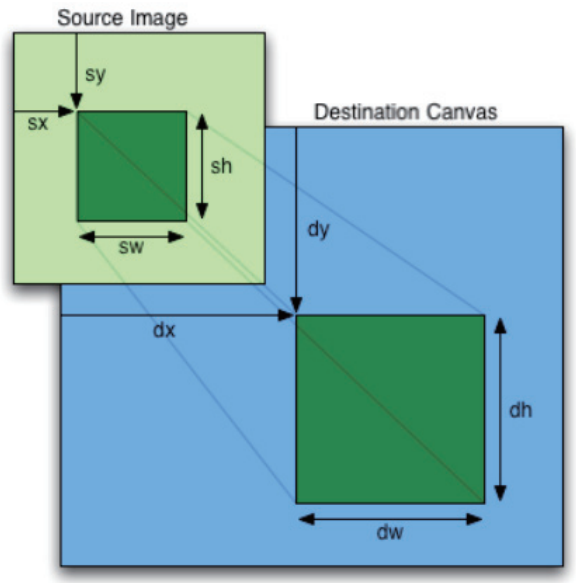


B-5 繪製圖像

2D 繪圖環境提供了下列幾個方法用來繪製圖像：

- drawImage(*image*, *dx*, *dy*)：以座標 (*dx*, *dy*) 為左上角，開始繪製參數 *image* 所指定的圖像，且圖像的大小為原始尺寸。
- drawImage(*image*, *dx*, *dy*, *dw*, *dh*)：以座標 (*dx*, *dy*) 為左上角，開始繪製參數 *image* 所指定的圖像，且圖像的大小為參數 *dw*、*dh* 所指定的寬度和高度。
- drawImage(*image*, *sx*, *sy*, *sw*, *sh*, *dx*, *dy*, *dw*, *dh*)：從參數 *image* 所指定的圖像切割出一部分來顯示，該部分的左上角座標為 (*sx*, *sy*)、寬度為 *sw*、高度為 *sh*。

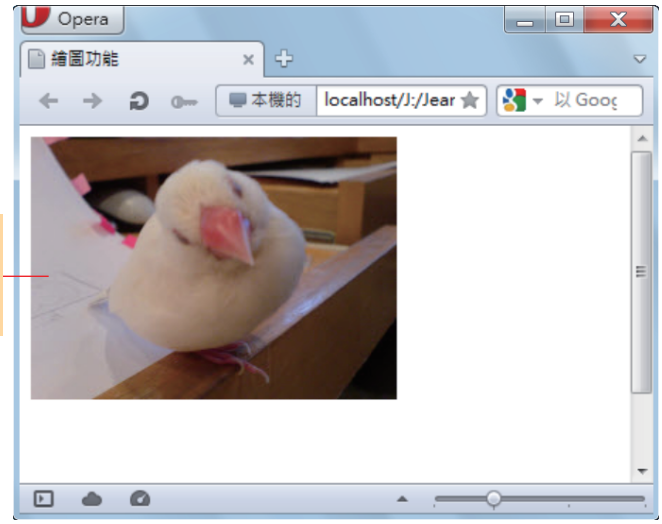
請注意，參數 *image* 必須為 HTMLImageElement、HTMLCanvasElement 或 HTMLVideoElement 其中一種型別的物件，也就是 <img>、<canvas> 或 <video> 元素的物件，同時為了幫助網頁設計人員瞭解前述參數的意義，HTML Canvas 2D Context (http://dev.w3.org/html5/2dcontext/) 提供了如下的示意圖。



下面是一個例子 <\ 附錄 B\canvas6.html>，它會先建立一個 <img> 元素，然後在繪圖區以座標 (0, 0) 為左上角，開始繪製參數 *image* 所指定的圖像，且圖像的大小為原始尺寸。

```
<canvas id="myCanvas" width="400" height="300"></canvas>
<script>
  var canvas = document.getElementById("myCanvas");
  var context = canvas.getContext("2d");
  //建立 <img> 元素
  var image = document.createElement("img");
  //待圖像載入後，就在繪圖區以座標 (0, 0) 為左上角，繪製原尺寸圖像
  image.onload = function() {
    context.drawImage(image, 0, 0);
  }
  //讀取圖像
  image.src = "bird.jpg";
</script>
```

這個例子的瀏覽結果如下圖。



在繪圖區繪製 blue.jpg 圖像，這是一張小鳥的照片

B-6 建立路徑與繪製圖形

在說明如何繪製圖形之前，我們要先瞭解何謂「路徑」(path)，這是零條、一條或多條子路徑的集合，而「子路徑」(subpath) 是連接一個或多個點的直線或曲線，一個封閉的子路徑指的是第一個點和最後一個點之間以直線連接起來。

我們將繪製圖形的步驟歸納如下，當子路徑包含少於兩個點時，將會被忽略，而不會被繪製出來：

1. 呼叫 beginPath() 方法重設目前的路徑，之前建立的路徑會被清除。

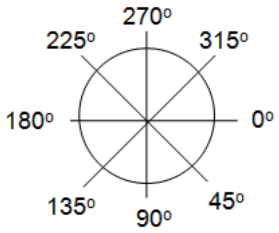
2. 呼叫繪圖環境的各種方法建立路徑。

3. 呼叫 stroke() 方法繪製路徑，或呼叫 fill() 方法填滿路徑。

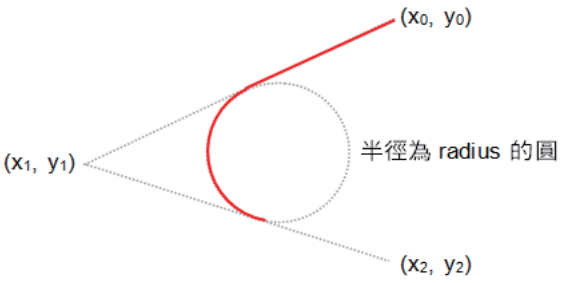
2D 繪圖環境提供了許多方法用來建立路徑與繪製圖形，比較重要的如下：

- beginPath()：重設目前的路徑，之前建立的路徑會被清除。
- closePath()：封閉目前的子路徑，也就是從最後一個點拉一條直線到第一個點，並將該子路徑標示為封閉的狀態。
- stroke()：以目前的繪製樣式繪製目前路徑的子路徑。
- fill()：以目前的填滿樣式填滿目前路徑的子路徑。
- moveTo(*x*, *y*)：移到座標為 (*x*, *y*) 的點，準備要建立一個新的子路徑。
- lineTo(*x*, *y*)：將座標為 (*x*, *y*) 的點加入目前的子路徑，同時以一條直線連接這個點與前一個點。
- rect(*x*, *y*, *w*, *h*)：加入新的矩形封閉子路徑，其左上角座標為 (*x*, *y*)、寬度為 *w* 像素、高度為 *h* 像素。
- quadraticCurveTo(*cpx*, *cpy*, *x*, *y*)：加入二次方貝茲曲線。

- bezierCurveTo(*cp1x*, *cp1y*, *cp2x*, *cp2y*, *x*, *y*)：加入三次方貝茲曲線。
- arc(*x*, *y*, *radius*, *startAngle*, *endAngle* [, *anticlockwise*])：加入圓弧，該圓弧可以是完整的圓或圓的部分曲線，座標 (*x*, *y*) 為圓心，*radius* 為圓的半徑，*startAngle* 為開始繪製的弧度 (相對於 X 軸)，*endAngle* 為結束繪製的弧度，選擇性參數 *anticlockwise* 為繪製方向，預設為 false，表示順時針方向。下圖的角度供您參考，而弧度與角度的轉換公式為「弧度＝角度 × π ÷180」。



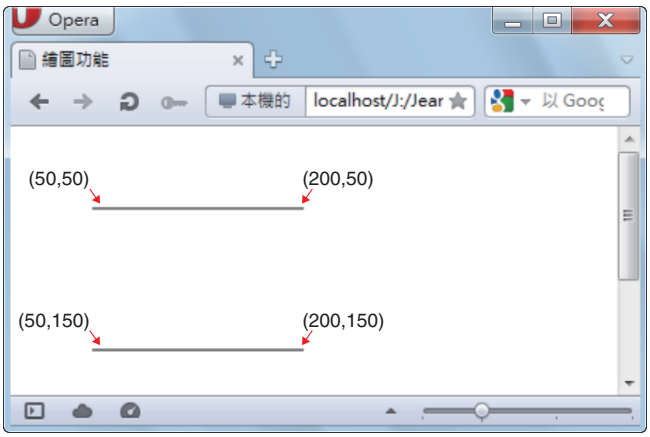
- arcTo(*x1*, *y1*, *x2*, *y2*, *radius*)：繪製直線及與其相切的圓弧，請您想像有兩條直線和一個圓，其中一條直線是從目前座標為 (*x0*, *y0*) 的點連接到座標為 (*x1*, *y1*) 的點，另一條直線是從座標為 (*x1*, *y1*) 的點連接到座標為 (*x2*, *y2*) 的點，而圓的半徑為 *radius*，且與這兩條直線各有一個切點，而 arcTo() 方法會繪製出從目前的點連接到第一個切點的直線，以及從第一個切點連接到第二個切點的圓弧，如下圖。



- clip()：指定目前路徑的裁剪區域。

B-6-1 繪製直線

我們直接以下面的例子 <\ 附錄 B\canvas7.html> 為您示範如何繪製直線，裡面有使用到 beginPath()、moveTo()、lineTo()、stroke() 等方法，瀏覽結果如下圖。




```
<canvas id="myCanvas" width="400" height="300"></canvas>
<script>
  var canvas = document.getElementById("myCanvas");
  var context = canvas.getContext("2d");
  // 重設目前的路徑
  context.beginPath();
  // 移到座標為 (50, 50) 的點
  context.moveTo(50, 50);
  // 將座標為 (200, 50) 的點加入目前的子路徑，同時以一條直線連接這個點與前一個點
  context.lineTo(200, 50);
  // 移到座標為 (50, 150) 的點
  context.moveTo(50, 150);
  // 將座標為 (200, 150) 的點加入目前的子路徑，同時以一條直線連接這個點與前一個點
  context.lineTo(200, 150);
  // 以目前的繪製樣式繪製目前路徑的子路徑
  context.stroke();
</script>
```

B-6-2 繪製矩形

我們直接以下面的例子 <\ 附錄 B\canvas8.html> 為您示範如何繪製矩形，裡面有使用到 beginPath()、rect()、fill() 等方法，瀏覽結果如下圖。

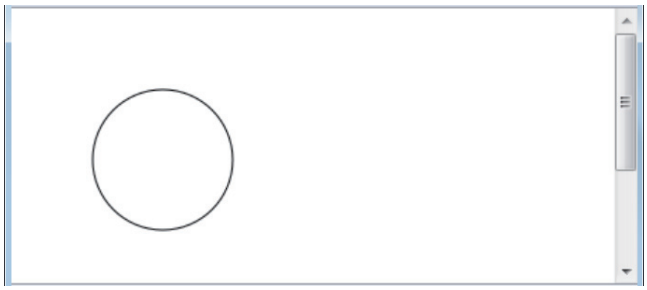
```
<canvas id="myCanvas" width="400" height="300"></canvas>
<script>
  var canvas = document.getElementById("myCanvas");
  var context = canvas.getContext("2d");
  // 重設目前的路徑
  context.beginPath();
  // 加入矩形封閉子路徑，其左上角座標為 (0, 0)、寬度為 100 像素、高度為 50 像素
  context.rect(0, 0, 100, 50);
  // 以目前的填滿樣式填滿目前路徑的子路徑
  context.fill();
</script>
```

若將 fill() 方法換成 stroke() 方法，就只會繪製矩形，而不會填滿矩形，如下圖。

B-6-3 繪製圓弧、圓形與扇形

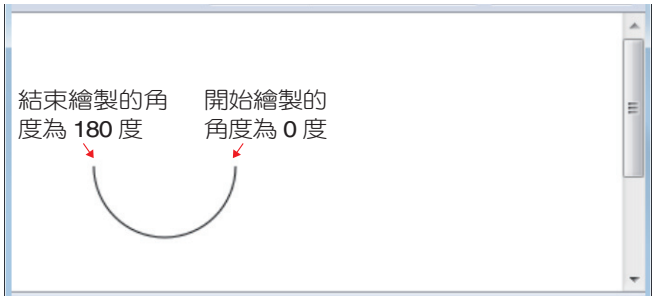
我們可以使用 arc() 方法繪製圓弧、圓形與扇形，下面是第一個例子 <\ 附錄 B\ canvas9.html>，它會繪製一個圓形，圓心的座標為 (100, 100)、半徑為 50 像素、開始繪製的弧度為 0、結束繪製的弧度為 360×π÷180、繪製方向為逆時針方向。

```
<canvas id="myCanvas" width="400" height="300"></canvas>
<script>
  var canvas = document.getElementById("myCanvas");
  var context = canvas.getContext("2d");
  context.beginPath();
  context.arc(100, 100, 50, 0, 360 * Math.PI / 180, true);
  context.stroke();
</script>
```



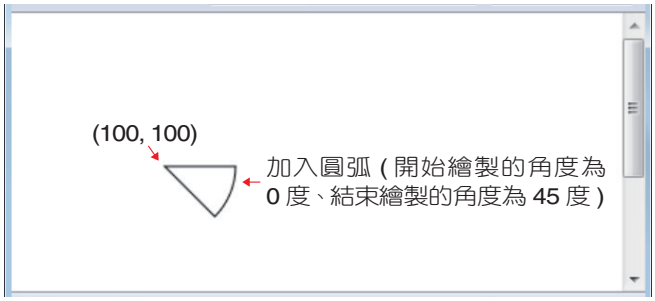
下面是第二個例子 <\ 附錄 B\canvas10.html>，它會繪製一個圓弧，圓心的座標為 (100, 100)、半徑為 50 像素、開始繪製的弧度為 0、結束繪製的弧度為 180×π÷180、繪製方向為順時針方向。

```
<canvas id="myCanvas" width="400" height="300"></canvas>
<script>
  var canvas = document.getElementById("myCanvas");
  var context = canvas.getContext("2d");
  context.beginPath();
  context.arc(100, 100, 50, 0, 180 * Math.PI / 180, false);
  context.stroke();
</script>
```



下面是第三個例子 <\ 附錄 B\canvas11.html>，它會繪製一個扇形，其中比較關鍵的步驟有三個，首先，呼叫 moveTo() 方法移到座標為 (100, 100) 的點，該點將做為扇形的中心點；接著，呼叫 arc() 方法加入一個圓弧，此時會自動從扇形的中心點拉一條直線到圓弧的起點；最後，呼叫 closePath() 方法封閉目前的子路徑，也就是從圓弧的終點拉一條直線到扇形的中心點。

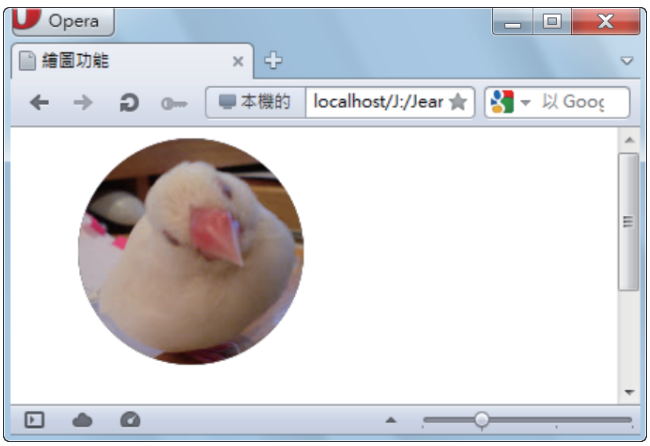
```
<canvas id="myCanvas" width="400" height="300"></canvas>
<script>
  var canvas = document.getElementById("myCanvas");
  var context = canvas.getContext("2d");
  context.beginPath();
  context.moveTo(100, 100);
  context.arc(100, 100, 50, 0, 45 * Math.PI / 180, false);
  context.closePath();
  context.stroke();
</script>
```



B-6-4 裁剪區域

我們可以使用 clip() 方法裁剪要顯示的範圍，下面是一個例子 <\ 附錄 B\ canvas12.html>，它先設定圓形的裁剪區域，然後載入圖像，再繪製裁剪區域。

```
<canvas id="myCanvas" width="400" height="300"></canvas>
<script>
  var canvas = document.getElementById("myCanvas");
  var context = canvas.getContext("2d");
  context.beginPath();
  context.moveTo(0, 100);
  context.arc(120, 80, 80, 0, 360 * Math.PI / 180, true);
  context.clip();
  //建立 <img> 元素
  var image = document.createElement("img");
  //待圖像載入後，就在繪圖區以座標 (0, 0) 為左上角，繪製原尺寸圖像
  image.onload = function() {
    context.drawImage(image, 0, 0);
  }
  //讀取圖像
  image.src = "bird.jpg";
  //繪製裁剪區域
  context.stroke();
</script>
```



B-7 設定線條樣式

2D 繪圖環境提供了下列幾個屬性用來設定線條樣式：

- lineWidth：線條的寬度，以像素為單位，預設為 1 像素，非大於 0 的有限值均會被忽略。
- lineCap：線條終點的樣式，有 butt (無樣式)、round (圓弧)、square (方形) 等值，預設為 butt，其它值均會被忽略。
- lineJoin：設定線條交叉接角的樣式，有 bevel (無樣式)、round (圓角)、miter (尖角) 等值，預設為 bevel，其它值均會被忽略。
- miterLimit：當 lineJoin 屬性的值等於 miter 時，miterLimit 屬性可以用來限制尖角的長度，預設為 10.0，其它諸如 0、負數、無限值或 NaN (Not a Number) 均會被忽略。

下面是一個例子 <\ 附錄 B\canvas13.html>，它會顯示如下圖的直線，且線條寬度為 15 像素、線條終點的樣式為圓弧。至於其它屬性和設定值，則請您自己試試看。

```
<canvas id="myCanvas" width="400" height="300"></canvas>
<script>
  var canvas = document.getElementById("myCanvas");
  var context = canvas.getContext("2d");
  context.lineWidth = "15";
  context.lineCap = "round";
  context.beginPath();
  context.moveTo(50, 50);
  context.lineTo(200, 50);
  context.stroke();
</script>
```

B-8 繪製文字與設定文字樣式

2D 繪圖環境提供了下列幾個方法用來繪製文字：

- fillText(*text*, *x*, *y* [, *maxWidth*])：以 fillStyle 屬性指定的填滿樣式，從座標為 (*x*, *y*) 處開始繪製參數 *text* 指定之文字，選擇性參數 *maxWidth* 可以指定字串寬度。

- strokeText(*text*, *x*, *y* [, *maxWidth*])：以 strokeStyle 屬性指定的繪製樣式，從座標為 (x, y) 處開始繪製參數 *text* 指定之文字的外框，選擇性參數 *maxWidth* 可以指定字串寬度。

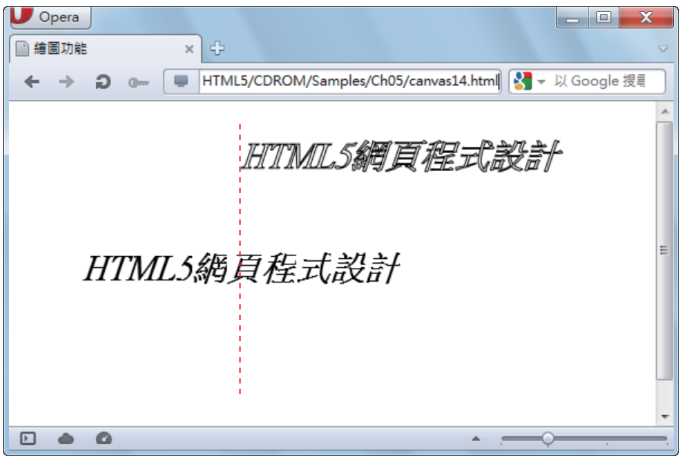
此外，2D 繪圖環境亦提供了下列幾個屬性用來設定文字樣式：

- font：以 CSS 語法設定文字的字型、大小 (*長度* | *絕對大小* | *相對大小* | *百分比*)、樣式 (normal | italic | oblique)、粗細 (normal | bold | bolder | lighter | 100 ~ 900)、行高 (normal | *數字* | *長度* | *百分比*) 等。

- textAlign：文字的水平對齊方式，有 start (以指定的座標為起點來顯示)、end (以指定的座標為終點來顯示)、left (以指定的座標為左側來顯示)、right (以指定的座標為右側來顯示)、center (以指定的座標為置中來顯示) 等值，預設為 start，其它值均會被忽略。

- textBaseline：文字的基準線，有 top、hanging、middle、alphabetic、ideographic、bottom 等值，效果如下圖 (參考自 HTML5 官方文件)，預設為 alphabetic。

下面是一個例子 <\ 附錄 B\canvas14.html>，它會在繪圖區繪製兩個字串，第一個字串是空心的 (只有繪製外框)，而且是以座標 (200, 50) 為起點來顯示，而第二個字串是填滿的，而且是以座標 (200, 150) 為置中來顯示。下圖的虛線是我們加上去的輔助線，用來標示 X 軸座標為 200 的位置供您參考比較。




```
<canvas id="myCanvas" width="500" height="300"></canvas>
<script>
  var canvas = document.getElementById("myCanvas");
  var context = canvas.getContext("2d");
  // 將字型設定為粗斜體、30 像素、新細明體
  context.font = "italic bold 30px 新細明體";
  // 將水平對齊方式設定為 start （以指定的座標為起點來顯示）
  context.textAlign = "start";
  // 使用前面設定的字型，以 (200, 50) 為起點來繪製文字的外框
  context.strokeText("HTML5 網頁程式設計", 200, 50);
  // 將水平對齊方式設定為 center （以指定的座標為置中來顯示）
  context.textAlign = "center";
  // 使用前面設定的字型，以 (200, 150) 為置中來繪製文字
  context.fillText("HTML5 網頁程式設計", 200, 150);
</script>
```

B-9 設定陰影樣式

2D 繪圖環境提供了下列幾個屬性用來設定陰影樣式：

- ❖ shadowColor：陰影的色彩。
- ❖ shadowOffsetX：陰影的水平位移，預設為 0 像素，表示沒有水平位移。
- ❖ shadowOffsetY：陰影的垂直位移，預設為 0 像素，表示沒有垂直位移。
- ❖ shadowBlur：陰影的模糊度，預設為 0，表示不模糊，數字愈大就愈模糊。

下面是一個例子 <\ 附錄 B\canvas15.html>，它會以紅色填滿一個矩形，而且會加上陰影，瀏覽結果如下圖。

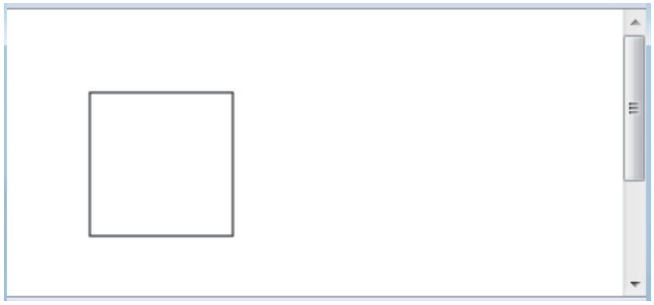
```
<canvas id="myCanvas" width="400" height="300"></canvas>
<script>
  var canvas = document.getElementById("myCanvas");
  var context = canvas.getContext("2d");
  context.shadowColor = "gray";           // 將陰影的色彩設定為灰色
  context.shadowOffsetX = 5;              // 將陰影的水平位移設定為 5 像素
  context.shadowOffsetY = 5;              // 將陰影的垂直位移設定為 5 像素
  context.shadowBlur = 10;                // 將陰影的模糊度設定為 10
  context.fillStyle = "red";                      // 將填滿樣式設定為紅色
  context.fillRect(50, 50, 60,80);        // 填滿一個矩形
</script>
```

B-10 變形

2D 繪圖環境提供了下列幾個方法用來進行變形，所謂「變形」是在建立形狀或路徑時，針對其座標進行矩陣運算，以達到縮放、旋轉、位移等效果：

- scale(*x*, *y*)：根據參數 *x* 指定的水平縮放倍率和參數 *y* 指定的垂直縮放倍率，進行縮小或放大，例如下面的敘述會繪製一個左上角座標為 (50, 50)、長寬均為 100 像素的矩形：

```
context.strokeRect(50, 50, 100, 100);
```

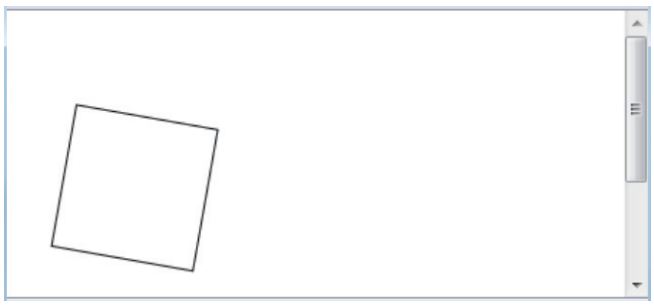


而下面的敘述會將矩形往水平方向放大 1.5 倍、往垂直方向縮小 0.5 倍，如欲恢復原來的縮放比例，可以使用 context.scale(1/1.5, 1/0.5);：

```
context.scale(1.5, 0.5);
context.strokeRect(50, 50, 100, 100);
```

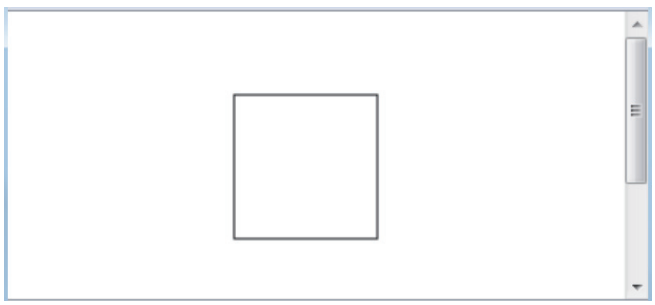
❖ rotate(*angle*)：根據參數 *angle* 指定的弧度往順時針方向進行旋轉，預設以左上角 (0, 0) 為圓心旋轉，例如下面的敘述會將矩形往順時針方向旋轉 10 度，如欲恢復原來的角度，可以使用 context.rotate(-10 * Math.PI / 180);，即旋轉 -10 度：

```
context.rotate(10 * Math.PI / 180);
context.strokeRect(50, 50, 100, 100);
```



❖ translate(*x*, *y*)：根據參數 *x* 指定的水平差距和參數 *y* 指定的垂直差距，進行座標轉移 (即位移)，例如下面的敘述會將矩形往水平方向位移 100 像素，如欲恢復原來的水平位置，可以使用 context.translate(-100, 0);：

```
context.translate(100, 0);
context.strokeRect(50, 50, 100, 100);
```

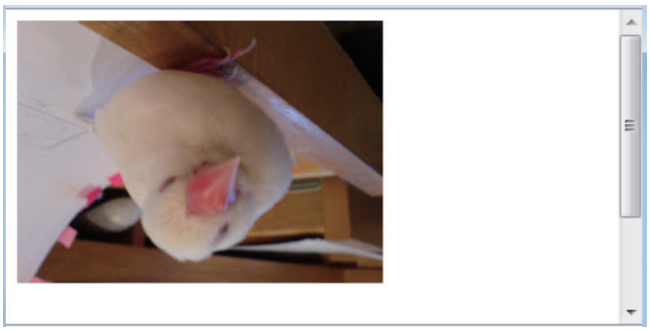


- transform(*a*, *b*, *c*, *d*, *e*, *f*)：將目前的座標矩陣乘以參數指定的矩陣，而參數所指定的矩陣如下。

$$\begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix}$$

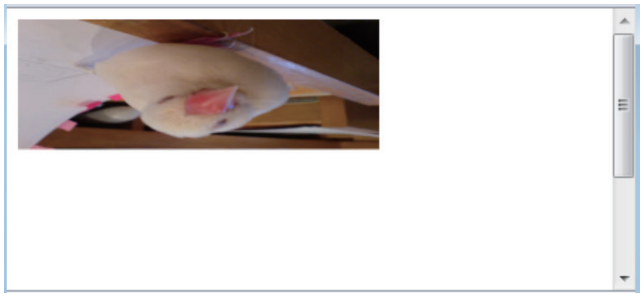
- setTransform(*a*, *b*, *c*, *d*, *e*, *f*)：重設座標矩陣，然後根據同樣的參數呼叫 transform(*a*, *b*, *c*, *d*, *e*, *f*) 方法。

下面的例子是在 <\附錄 B\canvas6.html> 加入 context.transform(1, 0, 0, -1, 0, image.height); 敘述，以產生鏡射圖像，瀏覽結果如下圖。

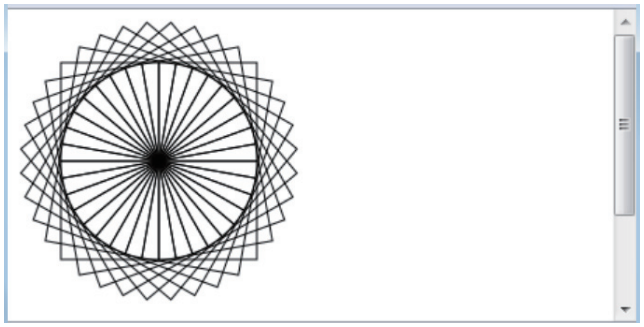


```
<canvas id="myCanvas" width="400" height="300"></canvas>
<script>
  var canvas = document.getElementById("myCanvas");
  var context = canvas.getContext("2d");
  var image = document.createElement("img");
  image.onload = function() {
    context.transform(1, 0, 0, -1, 0, image.height);  //呼叫transform() 方法產生鏡射圖像
    context.drawImage(image, 0, 0);
  }
  image.src = "bird.jpg";
</script>
```

若是將 context.transform(1, 0, 0, -1, 0, image.height); 敘述換成 context.transform(1, 0, 0, -0.5, 0, image.height*0.5);，則會得到如下圖的瀏覽結果 (<\ 附錄 B\canvas16.html>)。



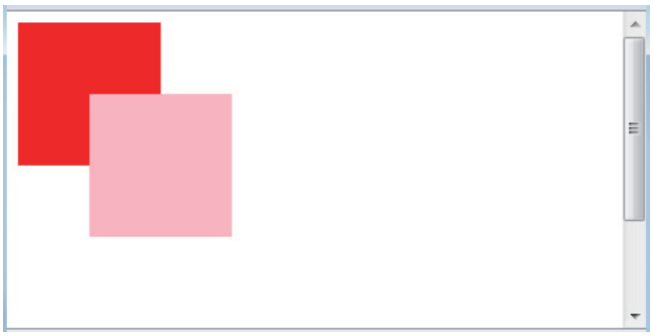
下面是另一個例子 <\ 附錄 B\canvas17.html>，它會繪製 36 個正方形，每個正方形之間的旋轉角度為 10 度，瀏覽結果如下圖。



```
<canvas id="myCanvas" width="400" height="300"></canvas>
<script>
  var canvas = document.getElementById("myCanvas");
  var context = canvas.getContext("2d");
  context.translate(100, 100);
  for (var i = 0; i < 36; i++){
    context.strokeRect(0, 0, 70, 70);
    context.rotate(10 * Math.PI / 180);
  }
</script>
```

B-11 重疊

本節所要討論的「重疊」指的是當新繪製的圖像與既有的圖像重疊時，新舊圖像之間該如何顯示。以下面的敘述為例 <\ 附錄 B\canvas18.html>，它會繪製兩個局部重疊的正方形，原先繪製的正方形為紅色，後來繪製的正方形為粉紅色，在預設的情況下，後來繪製的正方形會顯示在原先繪製的正方形上面，瀏覽結果如下圖。

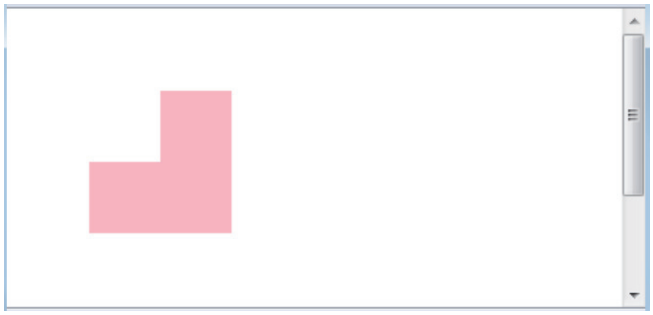


```
01:<canvas id="myCanvas" width="400" height="300"></canvas>
02:<script>
03:  var canvas = document.getElementById("myCanvas");
04:  var context = canvas.getContext("2d");
05:  context.fillStyle = "red";
06:  context.fillRect(0, 0, 100, 100);
07:  context.fillStyle = "pink";
08:  context.fillRect(50, 50, 100, 100);
09:</script>
```

若要自行設定重疊方式，可以使用 2D 繪圖環境提供的 globalCompositeOperation 屬性，該屬性的值如下，預設為 source-over：

- source-atop：顯示原先繪製的圖像，但重疊的部分會顯示後來繪製的圖像。
- source-in：只顯示重疊的部分，而且是後來繪製的圖像。

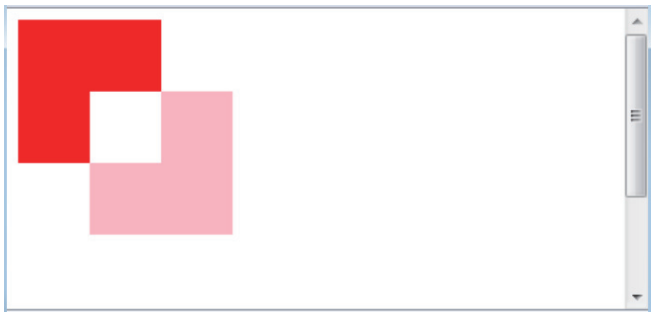
- source-out：只顯示後來繪製的圖像，但不顯示重疊的部分。以第 B-26 頁的程式碼為例，假設在第 05 行的前面加入 context.globalCompositeOperation = "source-out"; 敘述，會得到如下圖的瀏覽結果。



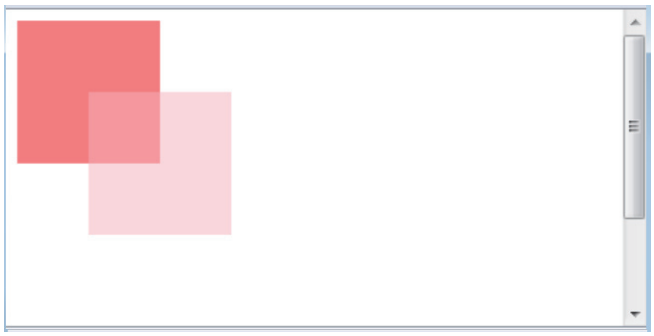
- source-over：後來繪製的圖像顯示在原先繪製的圖像上面，此為預設值。
- destination-atop：顯示後來繪製的圖像，但重疊的部分會顯示原先繪製的圖像。以第 B-26 頁的程式碼為例，假設在第 05 行的前面加入 context.globalCompositeOperation = "destination-atop"; 敘述，會得到如下圖的瀏覽結果。

- destination-in：只顯示重疊的部分，而且是原來繪製的圖像。
- destination-out：只顯示原來繪製的圖像，但不顯示重疊的部分。

- destination-over：原先繪製的圖像顯示在後來繪製的圖像上面。
- lighter：顯示原先繪製的圖像與後來繪製的圖像，但重疊的部分則顯示兩者融合後的色彩。
- copy：只顯示後來繪製的圖像。
- xor：顯示原先繪製的圖像與後來繪製的圖像，但重疊的部分則不顯示。以第 B-26 頁的程式碼為例，假設在第 05 行的前面加入 context.globalCompositeOperation = "xor"; 敘述，會得到如下圖的瀏覽結果。



除了設定重疊方式之外，我們也可以使用 2D 繪圖環境提供的 globalAlpha 屬性設定圖像的透明度，其值介於 0.0 (完全透明) 到 1.0 (不透明)。以第 B-26 頁的程式碼為例，假設在第 05 行的前面加入 context.globalAlpha = 0.5; 敘述，將透明度設定為 0.5，會得到如下圖的瀏覽結果。



B-12 像素運算

2D 繪圖環境提供了下列幾個屬性與方法用來處理單獨的像素，或許您會問，這有什麼用途呢？用途可妙了，透過處理單獨的像素，我們可以對圖像進行亮度調整、透明度調整或自訂濾鏡功能：

- createImageData(*sw*, *sh*)：根據參數 *sw*、*sh* 指定的寬度與高度，建立一個新的 ImageData 物件，該物件的屬性如下：
  - width：ImageData 物件的寬度，以像素為單位。
  - height：ImageData 物件的高度，以像素為單位。
  - data：ImageData 物件的像素資料，這是一個一維陣列，內容如下，其中 *r1*, *g1*, *b1*, *a1* 表示第一個像素的紅、綠、藍、透明度，*r2*, *g2*, *b2*, *a2* 表示第二個像素的紅、綠、藍、透明度，依此類推，而紅、綠、藍、透明度的值均為 0 ~ 255；至於像素的排列順序則是由圖像的左上角開始往右移，抵達最右邊後，往下一排像素並移到最左邊，繼續往右移，反覆此過程，直到抵達圖像的右下角：

```
{r1, g1, b1, a1, r2, g2, b2, a2, r3, g3, b3, a3, r4, g4, b4, a4,...}
```

- createImageData(*imagedata*)：根據和參數 *imagedata* 相同的寬度與高度，建立一個新的 ImageData 物件。
- getImageData(*sx*, *sy*, *sw*, *sh*)：取得繪圖區內指定之矩形的圖像資料，傳回值為一個 ImageData 物件，其中參數 *sx*、*sy* 為矩形的左上角座標，參數 *sw*、*sh* 為矩形的寬度與高度。
- putImageData(*imagedata*, *dx*, *dy* [, *dirtyX*, *dirtyY*, *dirtyWidth*, *dirtyHeight*])：將參數 *imagedata* 指定之 ImageData 物件，寫入繪圖區內座標為 (*dx*, *dy*) 處。若要指定 ImageData 物件的局部區域，可以使用選擇性參數指定該區域的左上角座標、寬度與高度。

下面是一個例子 <\ 附錄 B\canvas19.html>，它會逐一處理圖像的每個像素，將色彩反轉過來，然後顯示在繪圖區，瀏覽結果如下圖。

```
<canvas id="myCanvas" width="400" height="300"></canvas>
<script>
  var canvas = document.getElementById("myCanvas");
  var context = canvas.getContext("2d");
  var image = document.createElement("img");
  image.onload = function() {
    context.drawImage(image, 0, 0);
    // 取得代表圖像的 ImageData 物件
    var pixels = context.getImageData(0, 0, image.width, image.height);
    // 將 ImageData 物件內每個像素的 r、g、b 值反轉過來
    for (var i = 0; i < pixels.data.length; i += 4){
      pixels.data[i] = 255 - pixels.data[i];
      pixels.data[i + 1] = 255 - pixels.data[i + 1];
      pixels.data[i + 2] = 255 - pixels.data[i + 2];
    }
    // 將 ImageData 物件寫入繪圖區，即顯示反轉色彩過的圖像
    context.putImageData(pixels, 0, 0);
  }
  image.src = "bird.jpg";
</script>
```

B-13 儲存與回復繪圖區狀態

繪圖區狀態 (canvas state) 包含下列幾個項目：

- 目前的變形矩陣 (transformation matrix)。
- 目前的裁剪區域。
- strokeStyle、fillStyle、globalAlpha、lineWidth、lineCap、lineJoin、miterLimit、shadowOffsetX、shadowOffsetY、shadowBlur、shadowColor、font、textAlign、textBaseline、globalCompositeOperation 等屬性目前的值。

請注意，目前預設的路徑和目前的圖像並不包含在繪圖區狀態內，事實上，目前預設的路徑是永久存在的，除非您呼叫 beginPath() 方法加以重設。

2D 繪圖環境提供了下列兩個方法用來儲存與回復繪圖區狀態：

- save()：將目前的繪圖區狀態推入堆疊，即儲存繪圖區狀態。
- restore()：取出堆疊頂端的繪圖區狀態，即回復繪圖區狀態。

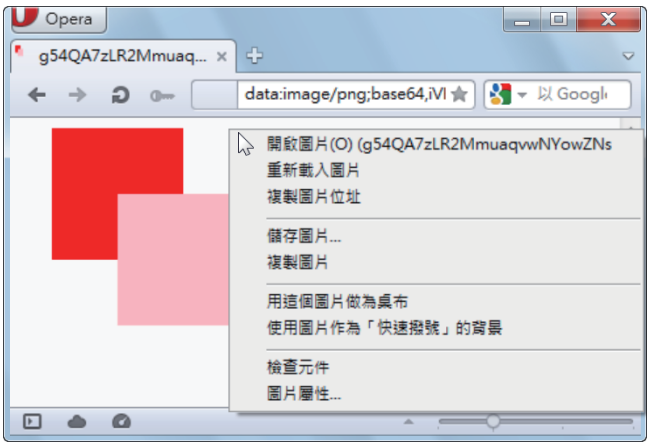
下面是一個例子 <\附錄 B\canvas20.html>，它會先儲存繪圖區狀態 (第 05 行)，接著將填滿樣式設定為銀色並填滿左上角的矩形 (第 06、07 行)，之後回復繪圖區狀態 (第 08 行)，此時，填滿樣式會回復為預設的黑色，再以黑色填滿右下角的矩形 (第 09 行)。

```
01:<canvas id="myCanvas" width="400" height="300"></canvas>
02:<script>
03:  var canvas = document.getElementById("myCanvas");
04:  var context = canvas.getContext("2d");
05:  context.save();
06:  context.fillStyle = "silver";
07:  context.fillRect(0, 0, 100, 100);
08:  context.restore();
09:  context.fillRect(50, 50, 100, 100);
10:</script>
```

B-14 匯出圖片

若要將繪圖區的圖片儲存成檔案，可以使用 <canvas> 元素提供的 toDataURL(optional *type* [, *any...args*]) 方法，選擇性參數 *type* 用來指定內容的類型，預設為 image/png。有些瀏覽器還會支援其它類型，例如 image/jpeg，此時可以加上第二個參數指定 JPEG 圖片的品質 (0.0 ~ 1.0)，省略不寫的話，表示為預設值。

下面是一個例子 <\ 附錄 B\canvas21.html>，它會呼叫 <canvas> 元素提供的 toDataURL() 方法將圖片儲存成 PNG 格式，當我們在圖片按滑鼠右鍵時，會出現如下圖的快顯功能表，供儲存圖片、複製圖片或將圖片設定為桌布。




```
<canvas id="myCanvas" width="400" height="300"></canvas>
<script>
  var canvas = document.getElementById("myCanvas");
  var context = canvas.getContext("2d");
  context.fillStyle = "red";
  context.fillRect(0, 0, 100, 100);
  context.fillStyle = "pink";
  context.fillRect(50, 50, 100, 100);
  window.location = context.canvas.toDataURL("image/png");
</script>
```