

HTML5 & CSS3 実践入門

第6章 キャンバスの描画

第7章 オーディオとビデオの埋め込み

第8章 CSS3を使った魅せ方

やじゅ@静岡Developers勉強会

自己紹介／資料の場所

- やじゅ@静岡・・・漢字名は「八寿」
SL(大井川鉄道)が走っている所に在住。
2010年から現在までMicrosoft MVP VisualBasic
を受賞。

Twitter : yaju

<http://blogs.wankuma.com/yaju/>
<http://yaju3d.hatenablog.jp/>

- 資料の場所

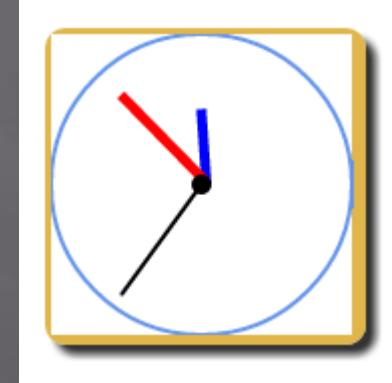
https://github.com/yaju/ShizuDev_HTML5
本のソースコードのダウンロード先
http://pragprog.com/titles/bhh5/source_code

今回のセッション方針

- ▣ 新人教育のための研修で
「概念を説明する時は、視覚（本・画面）や
聴覚（言葉）よりも、触覚（実物）で示す
のが一番早い」
と伺った。
実際に操作させて、実感させてこそその教育
座学では効果が薄い。

アジェンダ

- キャンバスの基礎
用語および基本的な使い方
- キャンバスの応用
オリジナル時計の作成
<http://randomibis.com/coolclock/>
- CSS 3 の概要
CSS 3 で出来るようになったことの紹介
- CSS 3 の応用
オリジナル時計に装飾
- オーディオタグとビデオタグの基礎
用語および基本的な使い方



※フォールバック(使用不可時の対処)は無視します。

<Canvas>要素の追加

ウェブページ上グラフィックを描画する<Canvas>要素が追加されました、これにより標準のHTMLとJavaScriptだけで、グラフやゲームグラフィックスなどの図形をすばやく表示できるようになります。

```
<canvas id="my_canvas" width="300" height="300">  
</canvas> // サイズ指定しない場合の初期値は300×150  
// 描画コンテキストの取得  
var canvas = document.getElementById('my_canvas');  
var context = canvas.getContext('2d');
```

canvasタグのチートシート

<http://simon.html5.org/dump/html5-canvas-cheat-sheet.html>

<Canvas>要素未対応

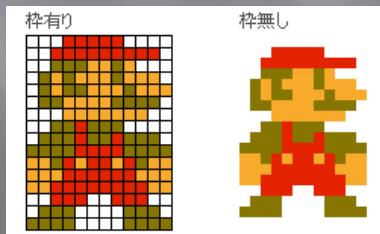
Canvas未対応でも線が引けた謎

- 1ピクセルにつき 1 個ずつ、指定色を背景にした SPANタグを生成

```
function(iColor, x, y, oSpan, iWidth) { appChild(oSpan,  
createSpan( (x - iWidth) * iPixSize, y * iPixSize, iWidth *  
iPixSize, iPixSize, aPalette[iColor] ) ); }
```

Google提供のExplorerCanvasライブラリもこの方式

- テーブルタグを使ってドット絵を表現（冗談w）



<http://mocheblog.blog22.fc2.com/blog-entry-387.html>

用語

□ コンテキスト

描画のための諸機能をまとめたオブジェクト

```
<canvas id="my_canvas" width="300" height="300"></canvas>
```

```
var canvas = document.getElementById('my_canvas');
```

```
var context = canvas.getContext('2d');
```

※2dは2次元(平面)、3dは3次元(WebGLで使用)

□ パス

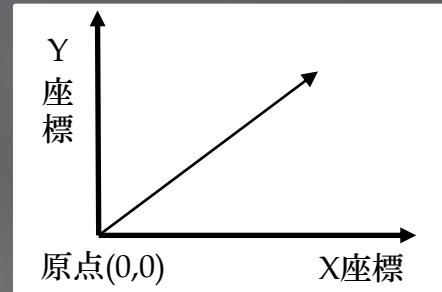
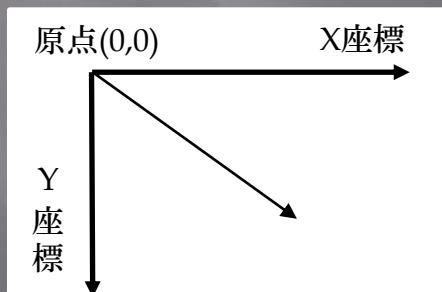
いくつかの線を組み合わせることによって作られた形状のこと

□ Canvasの座標系

左上が原点(0, 0)で右方向、下方向共に+（増加）する座標となる。

距離の単位はピクセルです。

数学で習った座標はデカルト座標といい、コンピュータと違い上方向共に+（増加）する座標となる。



キャンバスの描画機能

キャンバスの描画機能は基本的には、以下の3種類の図形+テキストと変形となります。

- ・矩形
- ・パス
- ・ビットマップ画像
- ・テキスト
- ・変形

パスには直線だけでなく円弧やベジェ曲線なども含められるので、ほぼすべての図形が描画可能です。

キャンバスの描画メソッドはあくまで描画する形状の情報のみをパラメータとして持ち、描画色などの指定はありません。

描画時のスタイル指定には別のメソッドが用意されています。このような構成をとることで少ないメソッドでも組み合わせによって多彩な表現を可能としています。

描画機能－矩形

□ 矩形の描画

- `strokeRect(x, y, width, height)`
 矩形の輪郭を描画します。
- `fillRect(x, y, width, height)`
 矩形を塗り潰します。
- `clearRect(x, y, width, height)`
 矩形を削除します。 (削除された部分は透明)

```
context.lineWidth = 1;          //線の太さ  
context.strokeStyle = 'rgb(255, 0, 0)'; //赤色  
context.strokeRect(20, 30, 60, 40);
```

矩形はパスではありません。

そのため、beginPath() メソッドは不要です。

※パスには矩形のrectメソッドが存在します。

描画機能－色の指定

□ 色の指定

デフォルトは、黒

```
context.strokeStyle = 'red';
```

```
context.strokeStyle = 'rgb(255,0,0)';
```

```
context.strokeStyle = 'rgba(255,0,0,0.5)';
```

```
context.strokeStyle = '#ff0000'; //RRGGBB
```



最後のaはアルファ値で、0.0～1.0の数値
0.0で完全な透明、1.0で完全な不透明

描画機能—スタイルの指定

□ スタイルの指定

```
context.lineWidth = 3; //線の太さ
```

参照: http://www.nail-kobe.com/html5/html5_1402.html



□ 影を付ける

//影色の指定

```
context.shadowColor = 'gray';
```

//影を表示する座標

```
context.shadowOffsetX = 8;
```

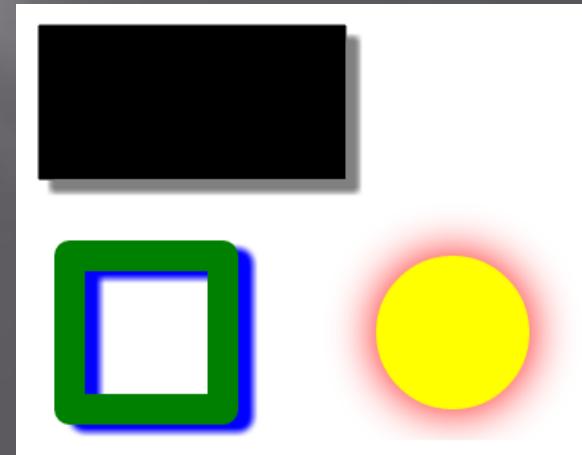
```
context.shadowOffsetY = 8;
```

//影のぼかし幅

```
context.shadowBlur = 5;
```

```
context.strokeRect(30, 30, 200, 100);
```

```
context.fillRect(30, 30, 200, 100);
```



描画機能—グラデーション①

■ 線形グラデーション

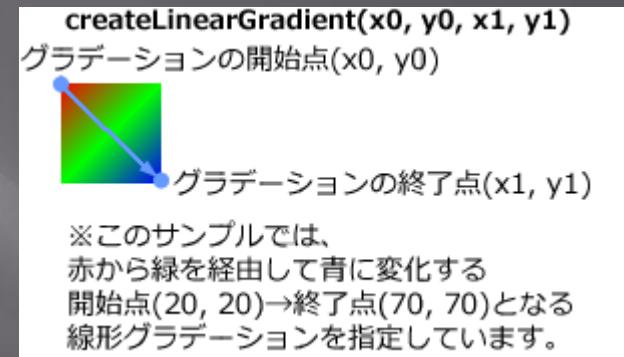
- `createLinearGradient(x0, y0, x1, y1)`

//線形グラデーションを指定する

```
var gradient = context.createLinearGradient(20,20,70,70);
```

//赤→緑→青のグラデーション色を3つ追加

```
gradient.addColorStop(0.0 , 'rgb(255,0,0)');
gradient.addColorStop(0.5 , 'rgb(0,255,0)');
gradient.addColorStop(1.0 , 'rgb(0,0,255)');
```

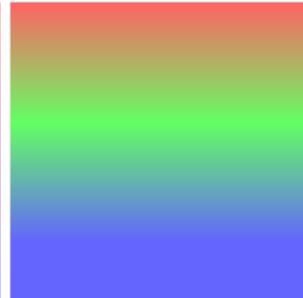


Linear Gradation 線形グラデーション

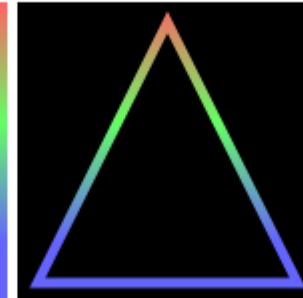
2 colors



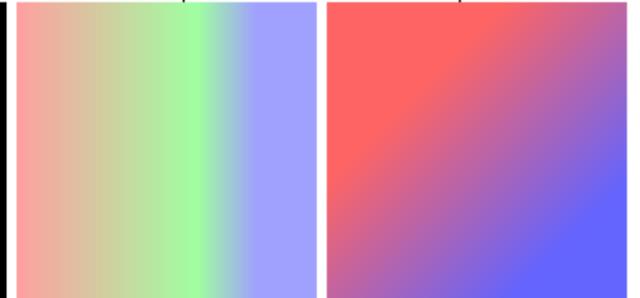
3 colors



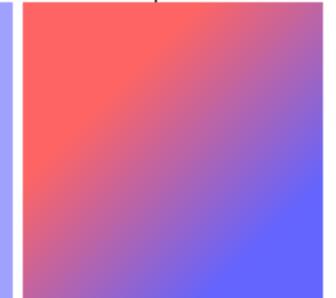
lines



Vertical Stripe



Skew Stripe



描画機能—グラデーション②

□ 円形グラデーション

- `createRadialGradient (x0, y0, r0, x1, y1, r1)`

//円形グラデーションを指定する：開始円=中心の座標(140,80)

半径20、終了円=中心の座標(140,80)半径80

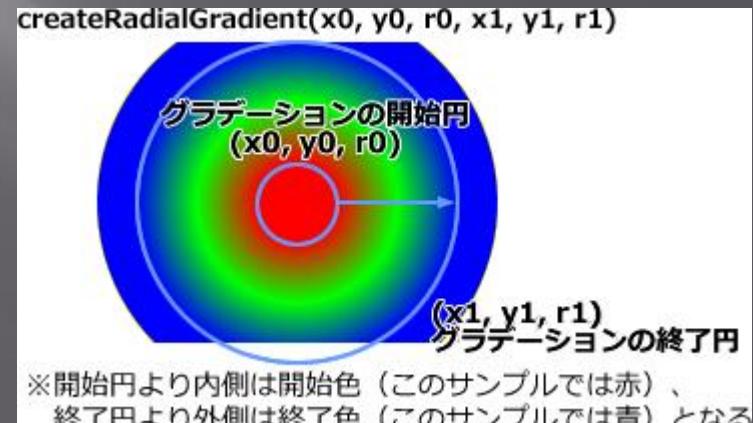
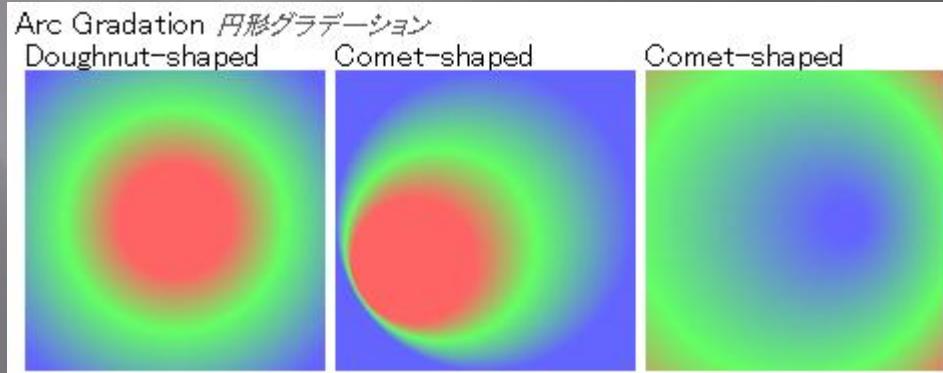
```
var gradient = context.createRadialGradient(140,80,20,140,80,80);
```

//赤→緑→青のグラデーション色を3つ追加

```
gradient.addColorStop(0.0 , 'rgb(255,0,0)');
```

```
gradient.addColorStop(0.5 , 'rgb(0,255,0)');
```

```
gradient.addColorStop(1.0 , 'rgb(0,0,255)');
```



描画機能－背景パターン

□ パターンにより繰り返し描画

- `createPattern(image, repeat)`

//img要素を生成

```
var image = new Image();
```

```
image.onload = function() {
```

//img要素の繰り返しパターン生成

```
var pattern = context.createPattern(image, "repeat");
```

//塗りつぶしパターンに指定

```
context.fillStyle = pattern;
```

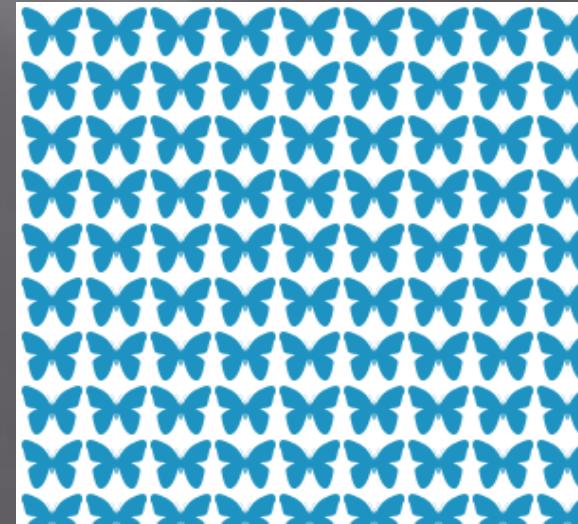
//canvasの描画

```
context.fillRect(0, 0, 300, 300);
```

```
};
```

//img要素読み込み

```
image.src = "butterfly.png"
```



描画機能－パス①

□ パスとは？

いくつかの線を組み合わせることによって作られた形状のこと
そのパスを構成する2つ以上の線を『サブパス』と言います。

□ パスの描画手順

1. beginPath()

パス描画の開始。beginPath()を呼び出すとそれまで描いていたパスはリセットされる。

2. canvasのAPIでパスを描画

パスを描画するには、描画Contextが持つメソッドを利用して描画。この段階ではまだ描画されない。

3. closePath()

パス描画の終了（これは必須ではありません）

4. stroke()、fill()を使用してグラフィックの表示

stroke()は線の描画(輪郭)、fill()はパスの内部を塗りつぶし

描画機能—パス②

□ パス定義の開始・終了

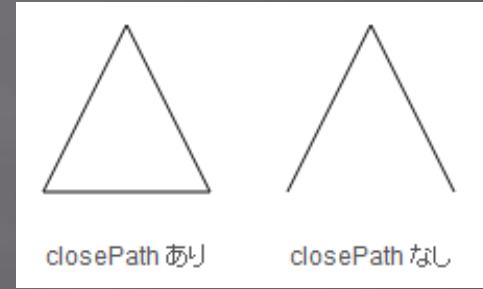
- beginPath()

パスの定義を開始します

- closePath()

パスの始点と終点を結び、閉じた図形にします

※パスが塗られた(fill)場合、図形は自動的に閉じられる



□ パスを使った三角形の描画

```
context.beginPath();           //パスのリセット
```

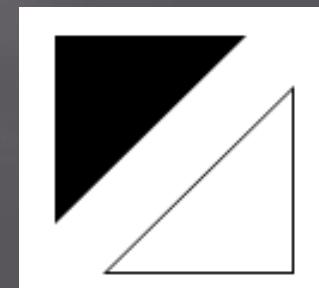
```
context.moveTo(10,90);       //開始支点
```

```
context.lineTo(50,10);       //線
```

```
context.lineTo(90,90);
```

```
context.closePath();          //パスを閉じる
```

```
context.stroke();             //線描画
```



描画機能－パス③

□ パスの定義

Canvas はパスの終端を「現在位置」として常に保存しており、ほとんどのメソッドはそれを始点として使用します。これにより、連続したパスを簡単に作成できます。

- `moveTo(x, y)`

パスを生成せずに現在位置を (x, y) に移動します。

- `lineTo(x, y)`

現在位置と (x, y) を結ぶ直線をパスに加えます。

実行後、現在位置は (x, y) に移動します。

- `rect(x,y,width,height)`

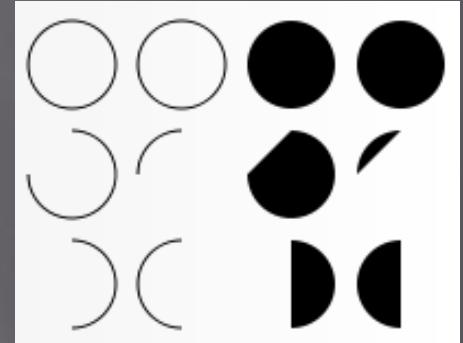
パスに、指定の矩形を表す閉じたサブパスを新規に追加します。

- `arc(x, y, radius, startAngle, endAngle, anticlockwise)`

(x, y) を中心とする半径 `radius` 円弧をパスに加えます。`startAngle` は開始角度、`endAngle` は終了角度です。いずれもラジアン単位です。

`anticlockwise` は `false` なら時計回り、`true` なら反時計回りにパスを生成します。他にも `arcTo(x1, y1, x2, y2, radius)` があります。

※橢円を描画するメソッドはありません。



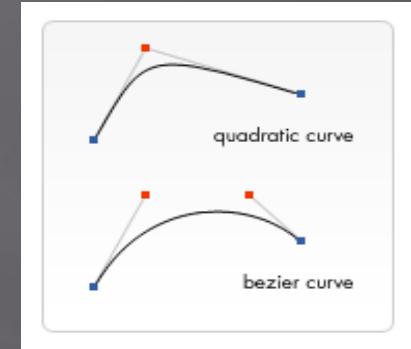
描画機能－パス④

□ パスの定義 続き

曲線の引くためには Flashのサイト

<http://hakuhin.jp/as/curve.html>

- quadraticCurveTo(cp1x, cp1y, x, y)
現在位置を始点、(x, y) を終点とする二次曲線を
パスに追加します。 (cp1x, cp1y) は制御点です。
実行後、現在位置は (x, y) に移動します。
- bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)
現在位置を始点、(x, y) を終点とする三次ベジェ
曲線をパスに追加します。 (cp1x, cp1y) と
(cp2x, cp2y) は制御点です。
実行後、現在位置は (x, y) に移動します。



描画機能－パス⑤

▣ パスの描画

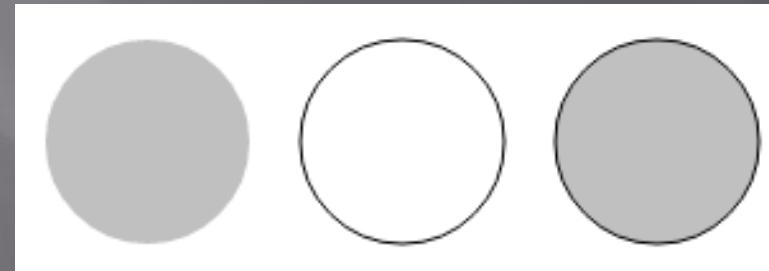
- `fill()`

直前に定義したパスを塗りつぶします。

- `stroke()`

直前に定義したパスの輪郭を描画します。

左からfillのみ、strokeのみ、fill+strokeの描画結果

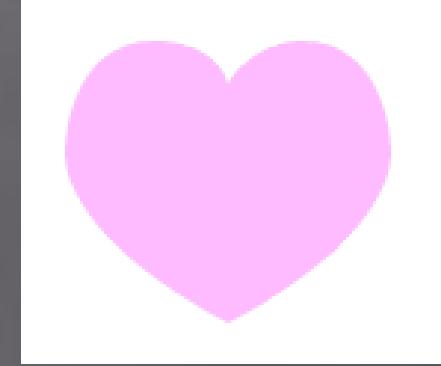


描画機能－パス⑥

□ 描画サンプル

ハートを描く

```
// ハート形のパスを生成
context.beginPath();
context.moveTo(75,40);
context.bezierCurveTo(75,37,70,25,50,25);
context.bezierCurveTo(20,25,20,62.5,20,62.5);
context.bezierCurveTo(20,80,40,102,75,120);
context.bezierCurveTo(110,102,130,80,130,62.5);
context.bezierCurveTo(130,62.5,130,25,100,25);
context.bezierCurveTo(85,25,75,37,75,40);
// ピンクで塗りつぶす
context.fillStyle = 'rgb(255,187,255)'
context.fill();
```



描画以外のパスの使い方①

□ Clip

```
context.save()  
// ハート形のクリッピングパスを生成  
context.beginPath();  
context.moveTo(75,40);  
context.bezierCurveTo(75,37,70,25,50,25);  
context.bezierCurveTo(20,25,20,62.5,20,62.5);  
context.bezierCurveTo(20,80,40,102,75,120);  
context.bezierCurveTo(110,102,130,80,130,62.5);  
context.bezierCurveTo(130,62.5,130,25,100,25);  
context.bezierCurveTo(85,25,75,37,75,40);  
context.clip();  
// イメージ描画  
var img = new Image();  
img.src = "cat.jpg";  
context.drawImage(img, 0, 0);  
// クリッピングを解除  
context.restore();  
context.drawImage(img, 150, 0);
```

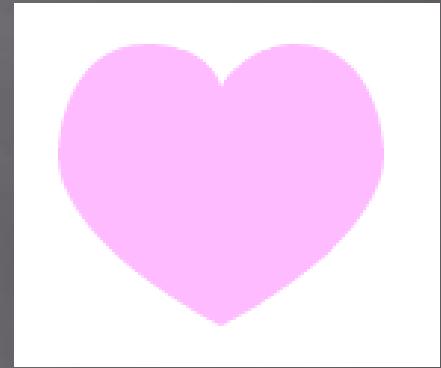


描画以外のパスの使い方②

■ 座標がパスの内側にあるかどうかを確認する

- `isPointInPath(x, y)`

```
canvas.addEventListener('mousemove', function(e) {  
    var rect = e.target.getBoundingClientRect();  
    mouse.x = e.clientX - rect.left;  
    mouse.y = e.clientY - rect.top;  
  
    if(mouse.x !== 0 || mouse.y !== 0) {  
        // ハート形のクリッピングパスを生成  
        context.beginPath();  
        context.moveTo(75,40);  
        context.bezierCurveTo(75,37,70,25,50,25);  
        context.bezierCurveTo(20,25,20,62.5,20,62.5);  
        context.bezierCurveTo(20,80,40,102,75,120);  
        context.bezierCurveTo(110,102,130,80,130,62.5);  
        context.bezierCurveTo(130,62.5,130,25,100,25);  
        context.bezierCurveTo(85,25,75,37,75,40);  
        if(context.isPointInPath(mouse.x, mouse.y))  
            context.fillStyle = 'rgb(255,0,0)'; // 内側なら赤に変更  
        else  
            context.fillStyle = 'rgb(255,187,255)';  
        context.fill();  
    } }, false);
```



描画機能—ビットマップ①

■ ラスタ形式とベクタ形式

- ・ラスタ形式とは、画像を、色のついた点(ドット)の羅列として表現したデータ
代表的な形式は、PNG、JPEG、GIF、BMP
- ・ベクタ形式とは、画像を点の座標と線や面の方程式など、図形情報の集まりとして表現する

代表的な形式は、S V G、アウトラインフォント、a i
http://server09.joeswebhosting.net/~xx5491/se3_diary/se3_diary/2.html

矩形やパスはベクタ形式ですが、C a n v a s に描画された時点でラスタ形式になります。

描画機能—ビットマップ②

□ イメージの描画

- `drawImage(image, dx, dy)`

canvas に指定のイメージを描画します。

<http://www.html5.jp/canvas/ref/method/drawImage.html>

`image`には、 `img`, `canvas`, `video` 要素が指定できます。

- `putImageData(imagedata, dx, dy)`

canvas に指定の `ImageData` オブジェクトのデータ
を描画します。

<http://www.htmq.com/canvas/putImageData.shtml>

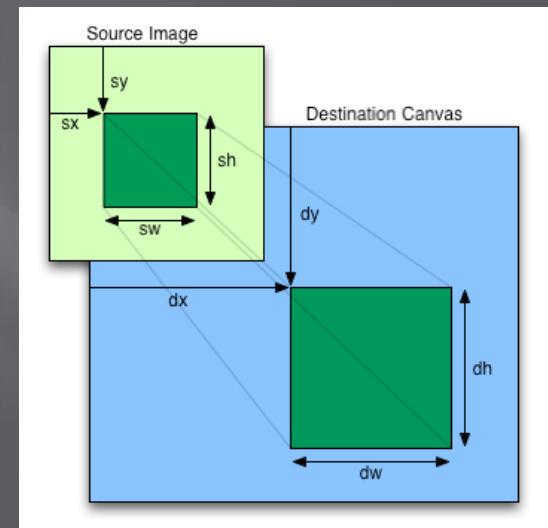
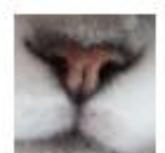
canvas内のピクセルは合成・透明化・影など非対象

描画機能—ビットマップ③

■ トリミング

- `drawImage(image, sx, sy, sw, sh, dx, dy, dw, dh)`
指定した位置に切り取り、拡大・縮小して描画できる。

```
var img = new Image();
img.src = "cat.jpg";
img.onload = function() {
    context.drawImage(img, 0, 0);
    context.drawImage(img, 60, 50, 50, 50, 200, 0, 60, 60);
}
```



描画機能—ビットマップ④

- イメージデータの取得と作成
 - `createImageData(sw, sh)`
指定された寸法の `ImageData` オブジェクトを返します。
生成される新規 `ImageData` オブジェクトは透明な黒です。
 - `getImageData(sx, sy, sw, sh)`
`canvas` の指定の矩形に対するイメージを含んだ
`ImageData` オブジェクトを返します。
画像から直接 `getImageData` でピクセルを取得できない
画像を一度キャンバスに描画する必要がある。

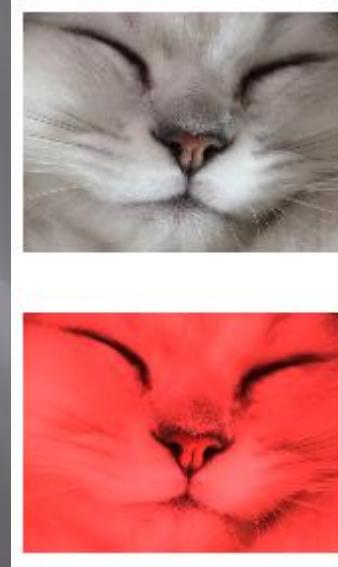
<http://www.html5.jp/canvas/ref/method/getImageData.html>

描画機能—ビットマップ⑤

■ ImageDataオブジェクトの編集

- ・赤色強調フィルター

```
img.onload = function() {
context.drawImage(img, 0, 0);
var w = img.width;
var h = img.height;
var imageData = context.getImageData(0, 0, w, h);
// ピクセル値を取得する
var pixelImage = context.createImageData(w, h);
// フィルター処理
for(var y=0; y<h; y++){
  for(var x=0; x<w; x++){
    var ptr = (y * w + x) * 4;
    // ピクセル処理する配列の要素位置を計算
    var R = imageData.data[ptr + 0];
    var G = imageData.data[ptr + 1];
    var B = imageData.data[ptr + 2];
    R = R * 2;
    if (R > 255) R = 255;
    G = Math.floor(G / 2);
    B = Math.floor(B / 2);
    pixelImage.data[ptr + 0] = R;
    pixelImage.data[ptr + 1] = G;
    pixelImage.data[ptr + 2] = B;
    pixelImage.data[ptr + 3] = 255;
  }
}
context.putImageData(pixelImage, 0, 150);
```



```
// グレースケールに変更してみて下さい
var myGray = parseInt((R + G + B) / 3);
pixelImage.data[ptr + 0] = myGray;
pixelImage.data[ptr + 1] = myGray;
pixelImage.data[ptr + 2] = myGray;
pixelImage.data[ptr + 3] = 255;
```

ビットマップ描画の注意①

□ 画像のプリロード

Imageオブジェクトは、画像が読み込まれない限り、Imageオブジェクトとしての機能を果たすことができません。

ローカルで動作させる場合はいいが、Webサーバーを通すとエラーになることが多い。更新するとキャッシュされたりしてエラーにならない。

```
var img = new Image();
img.src = "cat.jpg";
context.drawImage(img, 0, 0);
```

・対処方法

画像が読み込まれるのを待ってから処理する

```
var img = new Image();
img.src = "cat.jpg?" + new Date().getTime();
//画像が読み込まれるのを待ってから処理を続行
img.onload = function() {
    context.drawImage(img, 0, 0);
}
```

ビットマップ描画の注意②

- `getImageData`はローカルでは取得できない
JavaScript の同一出身ポリシーに引っかかるため

<http://d.hatena.ne.jp/chiheisen/20100815/1281885412>

画像データを取得するだけなのに、なんで同一出身ポリシーを適用しているのかというと、不用意に画像がどこかに送信されるということを防ぐためだそうです。

- 対処方法

Web サーバーを経由する IISやApache

簡易Web サーバー 「HTTP Server Anywhere」 を使う

http://hp.vector.co.jp/authors/VA049605/http_server_anywhere.html

80ポートが使われていた場合、空いているポート例8080とする

`http://localhost:8080/redcolorfilter.html`

Macの方 <http://www.futomi.com/lecture/macosx/apache.html>

ビットマップ描画の注意③

□ ダブルバッファリングは不要？

ダブルバッファリングは、大量に描画するとチラつくため、非表示バッファに描画して、描画を終えると表示用に切り替える技法。

どうも、Canvasでは裏でダブルバッファリングされているようで大量に描画してもちらつくことがない。ブラウザによる？

HTML5 + Javascript + canvas でダブルバッファリングをする
<http://d.hatena.ne.jp/hypercrab/20111014/1318558535>

描画機能 - テキスト

□ テキスト

- `fillText(text, x, y [, maxWidth])`
- `strokeText(text, x, y [, maxWidth])`

指定のテキストを指定の位置に塗りつぶす、または輪郭描画します。
最大幅が与えられるとテキストはフィットするよう伸縮される。

```
context.font = "20pt Arial";
context.fillText("Sample String", 10, 50);
```

- `measureText(text)`

現在のフォントにおける指定テキストの長さを持った TextMetrics オブジェクトを返します。

```
metrics = context.measureText("文字列の幅")
metrics.width // テキストの幅を前もって返す
```

描画機能－変形①

□ 拡大縮小、回転などの変形処理

- `scale(x, y)`

指定の伸縮変形を適用して、変換マトリックスを変更します。

- `translate(x, y)`

指定の移動変形を適用して、変換マトリックスを変更します。

- `transform(m11, m12, m21, m22, dx, dy)`

指定のマトリックスを適用して、変換マトリックスを変更します。

- `setTransform(m11, m12, m21, m22, dx, dy)`

今までの変換マトリックスを初期化して、上記のtransformする。

参考サイト：

<http://primedesignworks.blogspot.com/2010/05/html5-canvas-transform.html>

<http://akibahideki.com/blog/html5-canvas-1/canvas.html>

描画機能－変形②

□ 楕円を描く

scaleメソッドを利用して楕円に変換する

//元の円を赤色で描く

```
context.beginPath();
```

```
context.arc(25, 50, 20, 0, 2 * Math.PI, false);
```

```
context.strokeStyle = "red";
```

```
context.stroke();
```

//横に2倍、縦に0.7倍した円を青色で描く

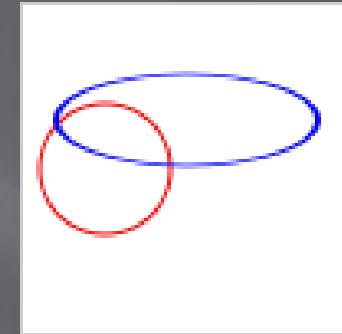
```
context.beginPath();
```

```
context.scale(2, 0.7);
```

```
context.arc(25, 50, 20, 0, 2 * Math.PI, false);
```

```
context.strokeStyle = "blue";
```

```
context.stroke();
```

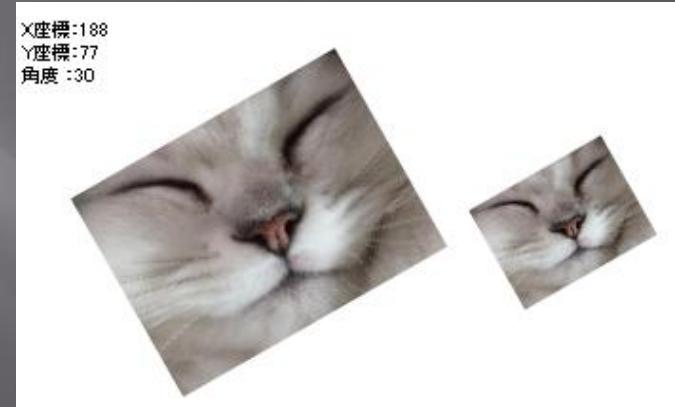


描画機能－変形③

□ 画像の変形

```
//setTransformにて表示  
//原点を画像の中心にして回転  
context.setTransform(Math.cos(rad), Math.sin(rad), -Math.sin(rad), Math.cos(rad), origin.x, origin.y);  
//原点を元に戻して描画  
context.drawImage(img, offset.x - origin.x, offset.y - origin.y);  
context.restore();
```

```
//縮小してtranslateとtransformにて表示  
context.save();  
//半分に縮小  
context.scale(0.5, 0.5);  
//原点を画像の中心にする  
context.translate(origin2.x, origin2.y)  
//回転  
context.transform(Math.cos(rad), Math.sin(rad), -Math.sin(rad), Math.cos(rad), 0, 0);  
//原点を元に戻す  
context.translate(-origin2.x, -origin2.y)  
//描画  
context.drawImage(img, offset2.x, offset2.y);  
context.restore();
```



状態の保存と復帰

□ 状態の保存と復帰

- `save()`

現在の状態をスタックの最後に加えます。

- `restore()`

スタックの最後の状態を抜き出し、その状態をコンテキストに復元します。

保存と復帰対象

- 1.これまでに適用された変形(移動、回転、伸縮)
- 2.現在の切り抜きパス(`clip`)
- 3.プロパティの値 `strokeStyle`、`fillStyle`、`globalAlpha`、`lineWidth`、`lineCap`、`lineJoin`、`miterLimit`、`shadowOffsetX`、`shadowOffsetY`、`shadowBlur`、`shadowColor`、`globalCompositeOperation`

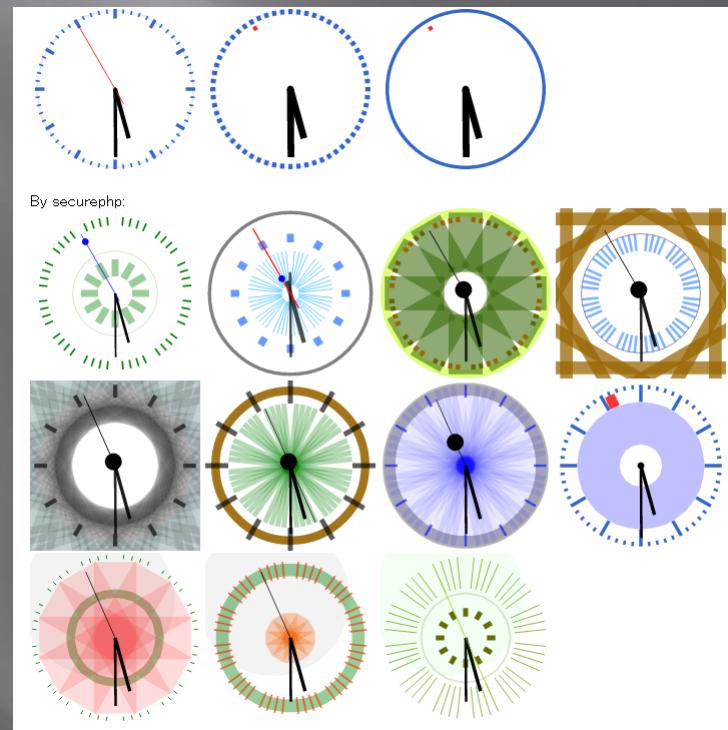
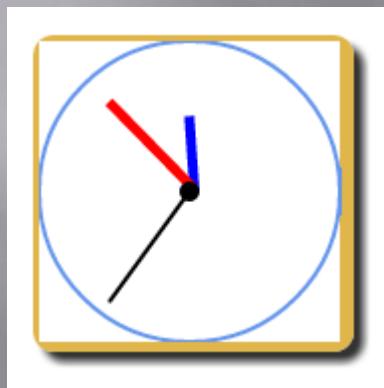
https://developer.mozilla.org/ja/Canvas_tutorial/Transformations

キャンバスの応用①

□ オリジナルの時計を作成

今まで習ったことを参考にカスタマイズする。

<http://randomibis.com/coolclock/>



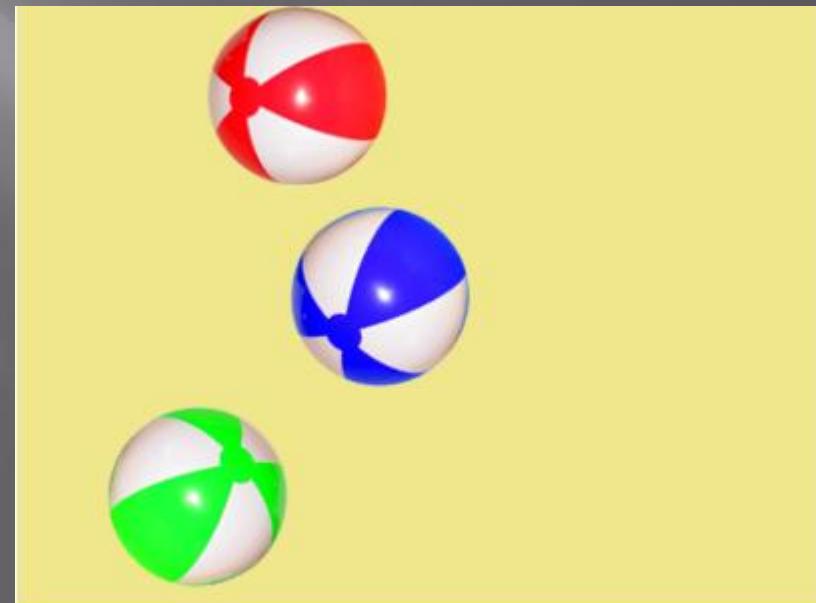
キャンバスの応用②

■ ボールの色変更と回転処理

赤いボールから色違いを作成

変形処理を使用して回転させる

- 1.getImageDataでボールのイメージデータを取得
- 2.赤成分を色変更して編集
- 3.編集結果を別キャンバスにputImageDataで描画
4. putImageDataでは透過描画されないため、drawImageで描画させる



CSS3の概要

□ CSS3で出来るようになったこと

- ・角を丸くする。
- ・背景にグラデーションをかける。
- ・対象物を、回転・拡大/縮小・歪ませる。
- ・文字やボックスに陰影をつける。
- ・コンテンツに合ったフォントが使える。
- ・アニメーションさせる。

CSS3なんて覚えなくても、これを使えば一瞬で作れちゃうんだよ。

<http://matome.naver.jp/odai/2132750704946539001>

CSS Transformsについてのメモ

<http://unformedbuiding.com/articles/learn-about-css-transforms/>

CSS3で注目のWebフォントについて今から勉強するためのサイトあつめました

<http://webdesignmatome.com/web-tips/webfont01>

CSS3 ドラえもん

<http://b.hatena.ne.jp/articles/201005/1118>

CSS3の応用

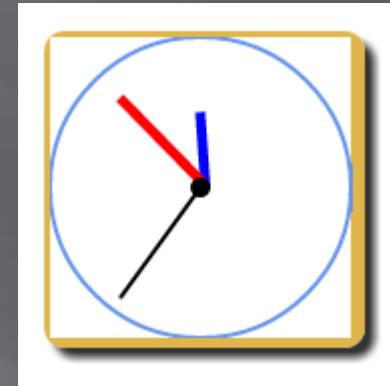
□ オリジナル時計に装飾

```
<link rel="stylesheet" href="clockstyle.css" type="text/css" media="screen">
```

clockstyle.cssの中身

```
body{padding: 40px;}
```

```
#frame{
    background-color: rgba(218,165,32,0.80);
    width: 160px;
    border-radius: 10px;      /* CSS3草案 */
    -webkit-border-radius: 10px; /* Safari,Google Chrome用 */
    -moz-border-radius: 10px;   /* Firefox用 */
    -moz-box-shadow: 5px 5px 5px #333;
    -webkit-box-shadow: 5px 5px 5px #333;
    -o-box-shadow: 5px 5px 5px #333;
    box-shadow: 5px 5px 5px #333;
}
```



オーディオタグとビデオタグの基礎①

■ オーディオタグの追加

- オーディオコーディック

<audio src="elvis.ogg">未サポートコメント</audio>

コーディック

AAC ・・・ iTunes Storeで使用している。ライセンス料発生

MP3 ・・・ 人気は高いが、特許問題がある。

Vorbis(OGG) ・・・ ボルビスと呼ぶ、ロイヤリティフリー。

<http://www.findmebyip.com/litmus>

HTML5 Audio Codecs

	MAC					WIN							
	SAFARI	FIREFOX	OPERA	CHROME		SAFARI	IE		FIREFOX	CHROME			
	5.1	8	9	11.1	15	17	5.1	6	7	8	9	8	15
Audio: ogg/vorbis	✗	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✓	✓
Audio: mp3	✓	✗	✗	✗	✓	✓	✓	✗	✗	✗	✓	✗	✓
Audio: wav	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓
Audio: AAC	✓	✗	✗	✗	✓	✓	✓	✗	✗	✗	✓	✗	✓

オーディオタグとビデオタグの基礎②

■ ビデオタグの追加

- ビデオタグ

<video src="sample.mp4">未サポートコメント</video>

コーディック

H.264 ・ ・ ・ デファクトスタンダード、特許権使用料問題あり。

Theora ・ ・ ・ ロイヤリティフリー。サブマリン特許懸念

VP8 ・ ・ ・ 完全ロイヤリティフリー。

<http://www.findmebyip.com/litmus>

HTML5 Video Codecs

	MAC					WIN				
	SAFARI	FIREFOX	OPERA	CHROME	SAFARI	IE	FIREFOX	CHROME		
	5.1	8 9	11.1	15 17	5.1	6 7 8 9	8	15		
Video: ogg/theora	✗	✓ ✓	✓	✓ ✓	✗	✗ ✗ ✗ ✗	✓	✓	70%	
Video: H.264	✓	✗ ✗ ✗	✗	✓ ✓	✓	✗ ✗ ✗ ✓	✗	✓	41%	
Video: WebM	✗	✓ ✓	✓	✓ ✓	✗	✗ ✗ ✗ ✗	✓	✓	45%	

オーディオタグとビデオタグの基礎③

□ H.264陣営 vs WebM陣営

<video>タグは死んだのか

HTML5ビデオフォーマットを巡る戦い

<http://blog.kazuhiroshibuya.com/archives/51245301.html>

□ videoの任意のフレームをcanvasに描画

ピクセル操作

<http://codezine.jp/article/detail/5615?p=4>

HTML5 videoの「映り込み」はCSS3または<canvas>で出来る！？

<http://akibahideki.com/blog/htmlcss/html5-videocss3.html>

videoの任意のフレームをそのままcanvasに描画させてみる

<http://d.hatena.ne.jp/favril/20100225/1267099197>

スライドや動画など、HTML5のサンプルいくつか

<http://kachibito.net/web-design/html5-sample.html>

Video映像をcanvasに加工描画

▣ 鏡像の生成

Videoタグの映像を反転かつ半分に縮小した後、半透明にして描画

```
var wid = video.width;  
var hei = video.height / 2;  
context.save();  
context.setTransform(1, 0, 0, -0.5, 0, hei);  
context.drawImage(video, 0, 0);  
context.fillStyle = "rgba(0,0,0,0.3)";  
context.fillRect(0, 0, wid, hei*2);  
context.restore();
```



気になるサイト

JavaScript、HTML5、CSSのソースコードを投稿・共有サイト

<http://jsdo.it/>

HTML5のCanvasに欠けているもの：フレームワーク

<http://www.atmarkit.co.jp/news/201003/30/canvas.html>

HTML5のCanvasをFlashライクに使う「EaselJS」を公開

<http://www.publickey1.jp/blog/11/html5canvasflasheaseljs.html>

ベクター画像描画JavaScriptフレームワーク「Paper.js」

<http://phpspot.org/blog/archives/2011/06/javascriptpaper.html>

F P S Frames Per Secondの処理　描画速度を見るため

<http://tech.kayac.com/archive/canvas-tutorial.html>

ご清聴ありがとうございましたm(_ _)m