Canvas
&
Js模块化开发

悟空

2015年10月10日

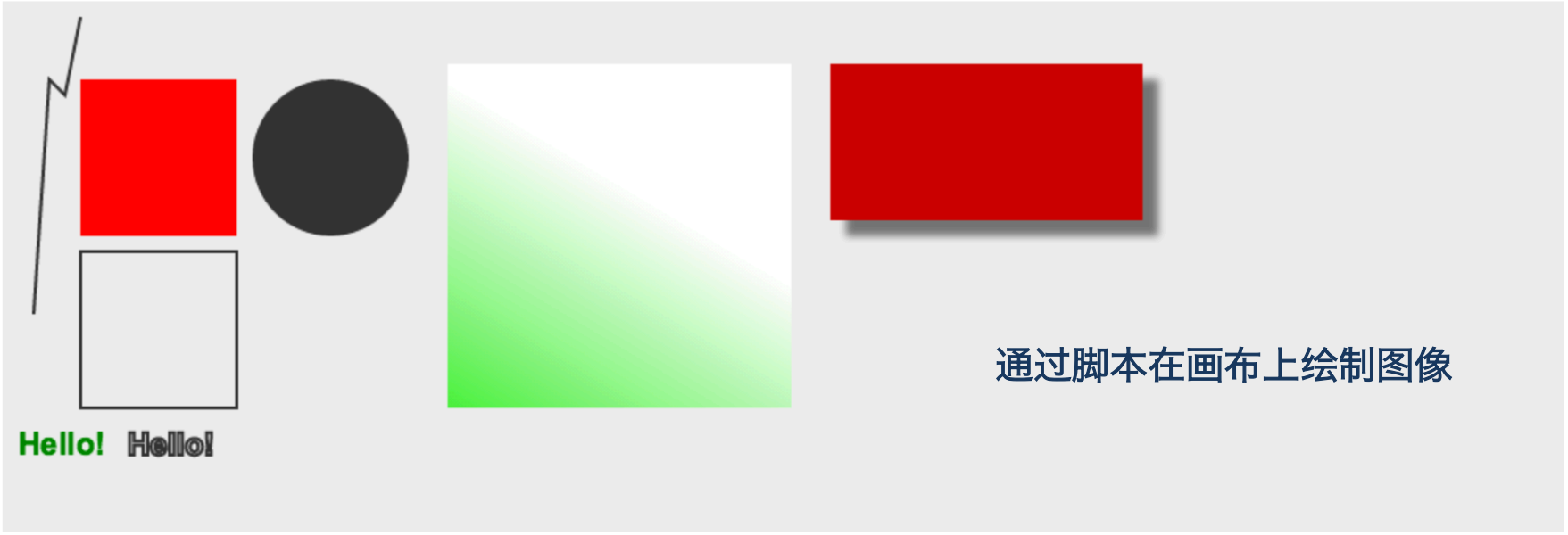
尚妆
SHOWJOY.COM
正 品 无 须 代 言

Canvas

Canvas元素是HTML5的一部分，允许**脚本语言**动态渲染位图像。

由js进行控制

位图（Bitmap），又称**栅格图**，是**使用像素阵列来表示的图像。**




```
<canvas id="myCanvas" width="400" height="200">
    您的浏览器不支持canvas!
</canvas>
```

<canvas> 标签来实现
Width、height分别表示画布区域的宽高

Canvas API

HTML：

```
<canvas id="myCanvas" width="400" height="200">
    您的浏览器不支持canvas!
</canvas>
```

JavaScript：

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

if (canvas.getContext) {
    var ctx = canvas.getContext('2d');
}
```

getContext对象（上下文对象）

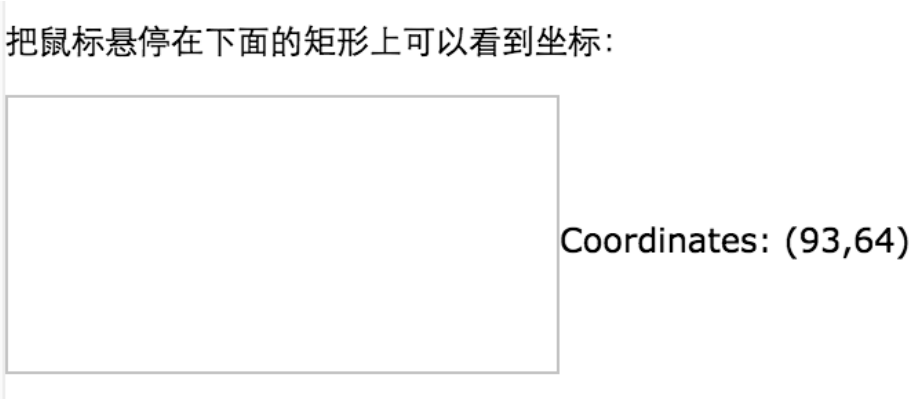
Canvas API定义在这个context对象上面，所以需要获取这个对象，方法是使用getContext方法。

getContext('2d')-- 平面图案

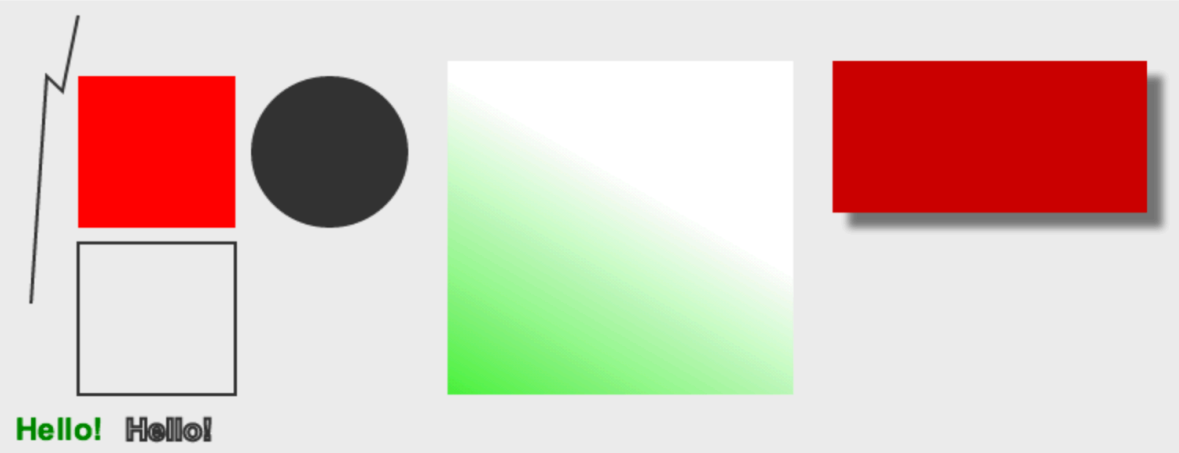
getContext('3d') -- WEBGL

画布提供了做图的空间，其中空间的点有对应的坐标(x, y)

左上角坐标为(0, 0)



Canvas API



直线　矩形

圆（扇形）　阴影

字符的描绘

……

路径的描绘

注：如果没有moveTo则第一个lineTo的位置作为起点，该点不绘制图像

```
ctx.beginPath(); // 开始路径绘制
ctx.moveTo(20, 20); // 设置路径起点，坐标为(20,20)
ctx.lineTo(200, 20); // 绘制一条到(200,20)的直线
ctx.lineWidth = 1.0; // 设置线宽
ctx.strokeStyle = "#CC0000"; // 设置线的颜色
ctx.stroke(); // 进行线的着色，这时整条线才变得可见
```

绘制了一个从(20, 20)开始到(200, 20)，粗细为1.0，颜色为红色的直线

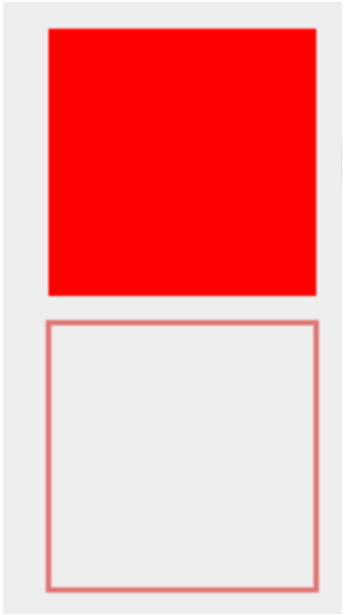
Canvas API

矩形的描绘

fillRect(x, y, width, height)方法用来绘制矩形

```
ctx.fillStyle = 'Red';
ctx.fillRect(50, 50, 100, 100);

ctx.strokeRect(50, 160, 100, 100);
```

fillRect绘制实心矩形
strokeRect绘制空心矩形



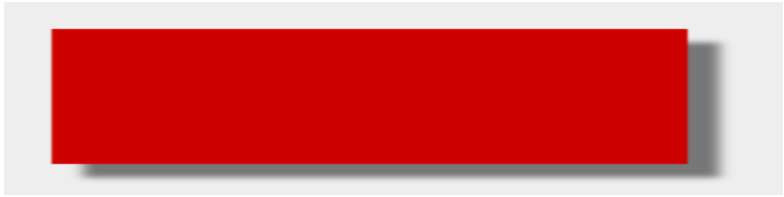
文本的绘制

fillText(string, x, y) 用来绘制文本


```
ctx.font = "Bold 20px Arial";
ctx.textAlign = "left";
ctx.fillStyle = "#008600";
ctx.fillText("Hello!", 10, 290);
ctx.strokeText("Hello!", 80, 290);
```

阴影的绘制

给某个绘制的图形添加shadow相关的方法实现



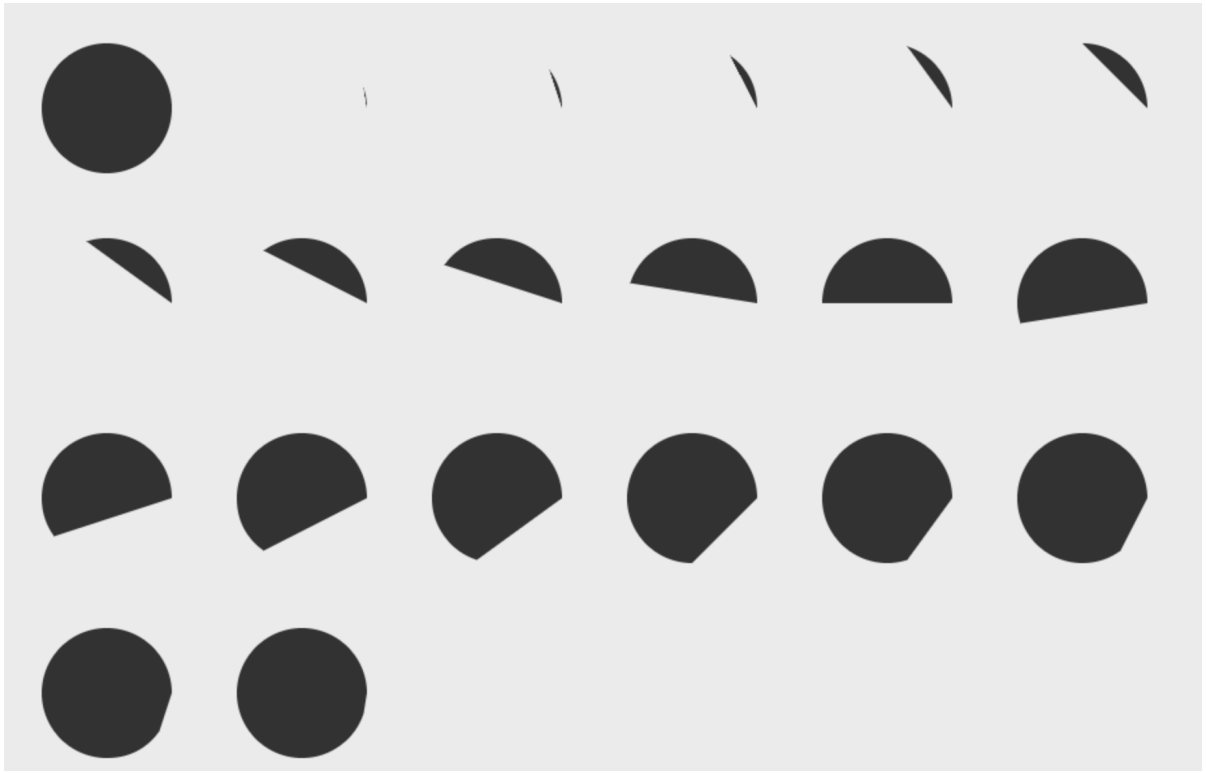
```
ctx.shadowOffsetX = 10; // 设置水平位移
ctx.shadowOffsetY = 10; // 设置垂直位移
ctx.shadowBlur = 5; // 设置模糊度
ctx.shadowColor = "rgba(0,0,0,0.5)"; // 设置阴影颜色

ctx.fillStyle = "#CC0000";
ctx.fillRect(10,10,200,100);
```

Canvas API

圆形（扇形）的描绘

```
ctx.beginPath();
ctx.arc(100, 100, 50, 0, Math.PI*2, true);
ctx.fillStyle = '#333333';
ctx.fill();
```

arc方法的**x和y参数是圆心坐标，radius是半径，startAngle和endAngle则是扇形的起始角度和终止角度（以弧度表示）**，anticlockwise表示做图时应该**逆时针画（true）**还是**顺时针画（false）**。



从Math.PI*2 到 Math.PI*0 的变化

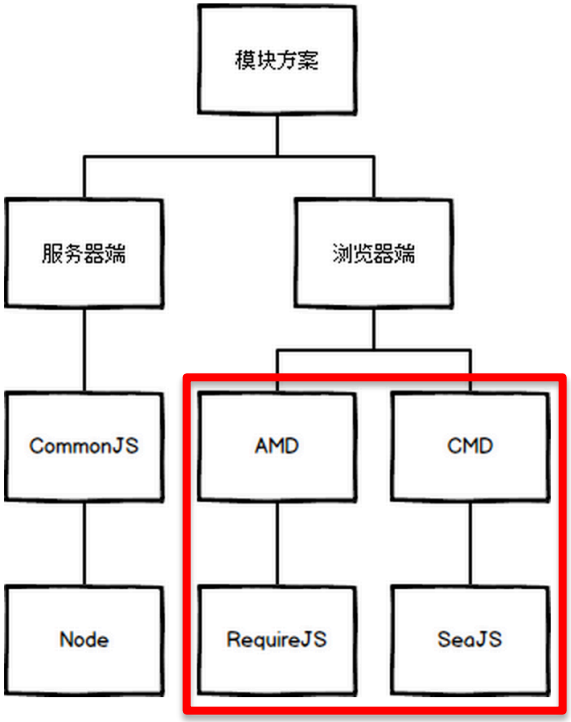
仅仅依靠arc方法是没有办法绘制出来扇形的，只能绘制出来填充了的圆弧。（如果需要绘制扇形会涉及到canvas的其他内容）

JavaScript模块化开发

AMD & CMD & CommonJS

前端代码的逻辑越来越复杂，通过原始的方法进行引用难以进行管理维护，而如果通过模块化开发的方式可以使逻辑更清晰，维护更简便。

```
define(function(require, exports){
  require('jqtransform');

  require('./bass/config');
  require('./bass/bass');

  require('./bass/widget/sidebar');
  require('./bass/widget/catMenu');
```



AMD规范 -- Require.js

Asynchronous Module Definition

CMD规范 -- Sea.js

Common Module Definition

difference?

AMD：提前执行（异步加载，依赖先执行）
CMD：延迟执行（运行到需加载，as lazy as possible）

定位的差异、遵循的规范不同、插件机制的不同等。

AMD & CMD

AMD规范 -- Require.js

Asynchronous Module Definition

```
// AMD 默认推荐的是
define(['./a', './b'], function(a, b) { // 依赖必须一
a.doSomething()
// 此处略去 100 行
b.doSomething()
...
})
```

```
seajs.config({
    alias: {
        'jqueryui': 'core/jqueryui/1.9.2/jquery-ui.js',
        'knockout': 'core/knockout/2.1.0/knockout-2.1.0.js',

        //plugin
        'customscroll': 'plugin/1.3.0/customscroll.js',

        'raty': 'plugin/1.3.0/jraty.js',

        'isotope': 'plugin/1.3.0/isotope.js',

        'jqtransform': 'plugin/1.13.2/jqtransform.js',
```

```
seajs.use('http://code.showjoy.net/view/ShowJoy-Assets/trunk/portal/src/global.js', function(){
    seajs.use(id, function(){
        if(callback){callback.apply(this, arguments);}
    });
});
```

CMD规范 -- Sea.js

Common Module Definition

```
// CMD
define(function(require, exports, module) {
var a = require('./a')
a.doSomething()
// 此处略去 100 行
var b = require('./b') // 依赖可以就近书写
b.doSomething()
// ...
})
```

vas-test-02.html × | global.js ×

```
define(function(require, exports){
    require('jqtransform');

    require('./bass/config');
    require('./bass/bass');

    require('./bass/widget/sidebar');
    require('./bass/widget/catMenu');

    var CommonUtil = require('./bass/util/commonUtil');
    var CookieUtil = require('./bass/util/cookieUtil');
```

JavaScript模块化开发

模块定义函数 define()

```
fn.define = function(id, deps, factory) {
    //code of function...
}
```

三个参数分别表示：

模块id ， 依赖模块数组 和 工厂函数

通常情况下只保留工厂函数一个参数

```
define(function(require, exports, module) {
    //code of the module...
});
```

只有一个参数的情况下，id和deps：
id会被默认赋值为此js文件的绝对路径。
deps一般在需要用到模块的时候require加载即可。

工厂方法的三个参数：

Require：模块加载函数，用于记载依赖模块。

```
var a = require('a');
```

```
var data1 = 1; //私有数据
```

Exports：接口点，将数据或方法定义在其上则将其暴露给外部调用。

```
exports.data2 = 2; //公共数据
```

Module：模块的元数据。 module是一个对象，存储了模块的元信息

.module.id——模块的ID。

.module.dependencies——一个数组，存储了此模块依赖的所有模块的ID列表。

.module.exports——与exports指向同一个对象。

JavaScript模块化开发

```
define(function(require, exports, module) {
    var a = require('a'); //引入a模块
    var b = require('b'); //引入b模块

    var data1 = 1; //私有数据

    var func1 = function() { //私有方法
        return a.run(data1);
    }

    exports.data2 = 2; //公共数据

    exports.func2 = function() { //公共方法
        return 'hello';
    }
});
```



```
define(function(require) {
    var a = require('a'); //引入a模块
    var b = require('b'); //引入b模块

    var data1 = 1; //私有数据

    var func1 = function() { //私有方法
        return a.run(data1);
    }

    return {
        data2: 2,
        func2: function() {
            return 'hello';
        }
    };
});
```

如果是只有返回值而没有任何定义的内容也可以直接简写成这种方式。

```
define({
    data: 1,
    func: function() {
        return 'hello';
    }
});
```

这种方式适合返回纯JSON数据

Seajs.use

页面加载一个或多个模块 seajs.use(id, callback?)

```
// 加载一个模块
seajs.use('./a');

// 加载一个模块，在加载完成时，执行回调
seajs.use('./a', function(a) {
  a.doSomething();
});

// 加载多个模块，在加载完成时，执行回调
seajs.use(['./a', './b'], function(a, b) {
  a.doSomething();
  b.doSomething();
});
```

Seajs和DOM ready没有任何的关系

因此如果有些操作需要在DOM ready的情况下操作，则需要通过其他方法来保证。

JavaScript模块化开发 – Sea.js

Seajs.config 配置项

alias: (别名配置)

配置后可以在模块中用require('jQuery')进行调用

```
seajs.config({
    alias: {
        'jqueryui': 'core/jqueryui/1.9.2/jquery-ui.js',
        'knockout': 'core/knockout/2.1.0/knockout-2.1.0.js',

        //plugin
        'customscroll': 'plugin/1.3.0/customscroll.js',

        'raty': 'plugin/1.3.0/jraty.js',

        pe': 'plugin/1.3.0/isotope.js',

        nsform': 'plugin/1.13.2/jqtransform.js',
```

```
seajs.config({
    //设置路径
    paths: {
        ' gallery ': 'https://a.alipayobjects.com/gallery'
    },

    // 设置别名，方便调用
    alias: {
        ' underscore ': ' gallery /underscore'
    }
});
```

path: (路径设置)

这样的调用可以不受基础路径的影响。

```
define(function(require, exports, module) {
    var _ = require(' underscore ');
     //=> 加载的是 https://a.alipayobjects.com/gallery/underscore.js
});
```

JavaScript模块化开发 - 传统方式

```
//module1.js
var module1 = {
    run: function() {
        return $.merge(['module1'], $.merge(module2.run(), module3.run()));
    }
}

//module2.js
var module2 = {
    run: function() {
        return ['module2'];
    }
}

//module3.js
var module3 = {
    run: function() {
        return $.merge(['module3'], module4.run());
    }
}

//module4.js
var module4 = {
    run: function() {
        return ['module4'];
    }
}
```

模块一 依赖 模块二和模块三
模块三 依赖 模块四

HTML：

```
<head>
    <meta charset="UTF-8">
    <title>TinyApp</title>
    <script src="./jquery-min.js"></script>
    <script src="./module4.js"></script>
    <script src="./module2.js"></script>
    <script src="./module3.js"></script>
    <script src="./module1.js"></script>
</head>
```

随着依赖关系越来越复杂，所依赖的文件越来越多，维护成本也越来越高，可读性差。

JavaScript模块化开发 - seajs

```
<body>
    <p class="content"></p>
    <script src="./sea.js"></script>
    <script>
        seajs.use('./init', function(init) {
            init.initPage();
        });
    </script>
</body>
```

模块化的方式编写使得维护成本会有所降低，也可以更好的组织代码，使得代码更加的清晰。

```
define(function(require, exports, module) {

    //原jquery.js代码...

    module.exports = $.noConflict(true);
});
```

```
//init.js
define(function(require, exports, module) {
    var $ = require('jquery');
    var m1 = require('module1');

    exports.initPage = function() {
        $('.content').html(m1.run());
    }
});
```

```
//module1.js
define(function(require, exports, module) {
    var $ = require('jquery');
    var m2 = require('module2');
    var m3 = require('module3');

    exports.run = function() {
        return $.merge(['module1'], $.merge(m2.run(), m3.run()));
    }
});
```

```
//module2.js
define(function(require, exports, module) {
    exports.run = function() {
        return ['module2'];
    }
});
```

```
//module3.js
define(function(require, exports, module) {
    var $ = require('jquery');
    var m4 = require('module4');

    exports.run = function() {
        return $.merge(['module3'], m4.run());
    }
});
```

```
//module4.js
define(function(require, exports, module) {
    exports.run = function() {
        return ['module4'];
    }
});
```

JavaScript模块化开发 – ES6

一览ES6中的模块化开发

```
var privateVar = "this is a variable private to the module";
export var publicVar = "and this one is public";

export function returnPrivateVar() {
   return privateVar;
};
```

将左边的代码保存在**mymodule.js**中

使用：

```
import { returnPrivateVar, publicVar } from 'mymodule';
console.log(returnPrivateVar());
```

或者是：

```
import 'mymodule' as mm;
console.log(mm.returnPrivateVar());
```

THANKS FOR YOUR ATTENTION

尚妆

SHOWJOY.COM

正 品 无 须 代 言