绘画板 12——变形 (上)

作者：crick77
原文链接：https://ld246.com/article/1475030452778
来源网站：链滴

github地址： https://github.com/wangyuheng/painter

DEMO地址： http://painter.crick.wang/

变形

既然实现了拖拽效果，就可以在此基础上，实现另一个效果：变形。

HandlerBorder

在实现变形效果之前，先讲解一下HandlerBorder。这是在pick时，选中元素后，在元素周围出现的个黑框。 现在将其扩展为8个黑框，拖拽黑框，实现元素的变形效果。

先分析一下HandlerBorder的代码

```
(function() {

    var sideLength = 8;
    var sideWidth = {
        width: 1
    };

    var HandleBorder = function(svgDoc) {
        this.init(svgDoc);
    }

    HandleBorder.prototype = {
        constructor: HandleBorder,
        init: function(svgDoc) {
            this.currentSvgDoc = svgDoc;
            this.create();
            return this;
        },
    };

    HandleBorder.prototype.create = function() {
        var _this = this;

        _this.handleBorderGroup = _this.currentSvgDoc.group();

        _this.blockGroup = _this.handleBorderGroup.group();

        _this.rectLeftTop = this.blockGroup.rect(sideLength, sideLength).stroke(sideWidth);
        _this.rectLeftBottom = this.blockGroup.rect(sideLength, sideLength).stroke(sideWidth);
        _this.rectRightTop = this.blockGroup.rect(sideLength, sideLength).stroke(sideWidth);
        _this.rectRightBottom = this.blockGroup.rect(sideLength, sideLength).stroke(sideWidth);

        _this.rectLeftCenter = this.blockGroup.rect(sideLength, sideLength).stroke(sideWidth);
        _this.rectRightCenter = this.blockGroup.rect(sideLength, sideLength).stroke(sideWidth);
        _this.rectTopCenter = this.blockGroup.rect(sideLength, sideLength).stroke(sideWidth);
        _this.rectBottomCenter = this.blockGroup.rect(sideLength, sideLength).stroke(sideWidth);
```

```
    };

    HandleBorder.prototype.rebound = function(bbox) {
        var _this = this;

        var x1 = bbox.x;
        var y1 = bbox.y;
        var x2 = bbox.x2;
        var y2 = bbox.y2;
        _this.rectLeftTop.move(x1 - sideLength, y1 - sideLength);
        _this.rectLeftBottom.move(x1 - sideLength, y2);
        _this.rectRightTop.move(x2, y1 - sideLength);
        _this.rectRightBottom.move(x2, y2);

        _this.rectLeftCenter.move(x1 - sideLength, (y2 + y1 - sideLength) / 2);
        _this.rectRightCenter.move(x2, (y2 + y1 - sideLength) / 2);
        _this.rectTopCenter.move((x2 + x1 - sideLength) / 2, y1 - sideLength);
        _this.rectBottomCenter.move((x2 + x1 - sideLength) / 2, y2);

    };

    HandleBorder.prototype.show = function(svgEle) {
        if (!svgEle) {
            return;
        }
        this.currentElement = svgEle;
        this.handleBorderGroup.show();

        this.rebound(svgEle.bbox());
    };

    HandleBorder.prototype.hide = function() {
        this.handleBorderGroup.hide();
    };

    this.HandleBorder = HandleBorder;

})();
```

1. HandleBorder有一个构造函数，调用create方法。create方法中创建了一个group，并在group创建8个矩形。

2. 在调用HandleBorder时，先new一个实例，然后调用其show方法，先将8个矩形的group显示出，再重新设定其位置rebound。

3. rebound方法根据元素边框坐标和小矩形的边长，计算其位置，分布在4个边角与4个边长中点。

变形原理

1. 将HandleBorder中的8个小矩形，作为操作框，绑定可拖拽效果。

2. 拖拽时，判断其移动距离与方向，使图形对应改变宽高，达到变形效果。

3. 为了避免拖拽时，操作框移动到任意位置，每次拖拽时，重新设置其位置rebound。

代码实现

实现拖拽

小矩形增加.draggable()方法调用。

```
_this.rectLeftTop = this.blockGroup.rect(sideLength, sideLength).stroke(sideWidth).draggable()

_this.rectLeftBottom = this.blockGroup.rect(sideLength, sideLength).stroke(sideWidth).dragga
le();
_this.rectRightTop = this.blockGroup.rect(sideLength, sideLength).stroke(sideWidth).draggabl
();
_this.rectRightBottom = this.blockGroup.rect(sideLength, sideLength).stroke(sideWidth).dragg
ble();

_this.rectLeftCenter = this.blockGroup.rect(sideLength, sideLength).stroke(sideWidth).draggab
e();
_this.rectRightCenter = this.blockGroup.rect(sideLength, sideLength).stroke(sideWidth).dragg
ble();
_this.rectTopCenter = this.blockGroup.rect(sideLength, sideLength).stroke(sideWidth).draggab
e();
_this.rectBottomCenter = this.blockGroup.rect(sideLength, sideLength).stroke(sideWidth).dra
gable();
```

绑定拖拽事件

以左侧中心点操作框 rectLeftCenter 为例

在dragstart时，获取鼠标所在坐标lastPoint

```
var lastPoint = null;
_this.rectLeftCenter.on("dragstart", function(event) {
    lastPoint = event.detail.p;
});
```

event.detail中包含4个成员变量，可以参考svg.draggable.js中的代码

```
this.el.fire('dragstart', {event: e, p: this.startPoints.point, m: this.m, handler: this})
```

绑定dragmove事件，并重新计算鼠标坐标。

```
_this.rectLeftCenter.on("dragmove", function() {
    var currPoint = event.detail.p;
    lastPoint = currPoint;
});
```

计算图形改变

dragmove时可以活动当前坐标和上次坐标，计算出x轴移动的距离

```
var dx = currPoint.x - lastPoint.x;
```

定义一个变量xLeft = 1;表明鼠标移动方向，向左移动，则为1， 否则为-1; 并在拖拽开始时，初始化

```
_this.rectLeftCenter.on("dragstart", function(event) {
    lastPoint = event.detail.p;
    xLeft = 1;
});
```

新的元素宽度为

```
var width = ele.width() - xLeft * dx;
```

判断width是否大于0，如果大于0，则表示正向移动， 则新坐标x为

```
var newX = ele.x() + xLeft * dx;
```

重新设定元素的x坐标和width，即可实现变形效果。

```
ele.x(newX).width(width);
```

如果为反向移动，则width小于 0，此时反转xLeft，坐标原点应为元素的bbox的x2点，并根据x的中做flip翻转。

```
xLeft = -xLeft;
ele.x(ele.bbox().x2).width(-width).matrix(new SVG.Matrix(ele).flip('x', ele.bbox().cx));
```

反转坐标复位

翻转之后的rebound无法复原，需要在rebound同样执行矩阵。

```
this.blockGroup.matrix(new SVG.Matrix(_this.currentElement));
```

限制拖拽轨迹

为了避免拖拽时，小矩形可以任意移动，则在拖拽时，执行rebound操作校准。svg.draggable.js默为提供此事件，扩展svg.draggable.js 增加如下方法。在DragHander.prototype.drag的结尾处增加

```
// so we can use it in the end-method, too
this.el.fire('afterdragmove', { event: e, p: p, m: this.m, handler: this });
```

并在小矩形中监听次事件，执行复位校准。

```
_this.rectLeftCenter.on("afterdragmove", function() {
    _this.rebound(_this.currentElement.bbox());
});
```

代码

handle.border.js完整代码如下

```
(function() {

    var sideLength = 8;
    var sideWidth = {
        width: 1
    };

    var HandleBorder = function(svgDoc) {
        this.init(svgDoc);
    }

    HandleBorder.prototype = {
        constructor: HandleBorder,
        init: function(svgDoc) {
            this.currentSvgDoc = svgDoc;
            this.create();
            return this;
        },
    };

    HandleBorder.prototype.create = function() {
        var _this = this;

        _this.handleBorderGroup = _this.currentSvgDoc.group();

        _this.blockGroup = _this.handleBorderGroup.group();

        _this.rectLeftTop = this.blockGroup.rect(sideLength, sideLength).stroke(sideWidth).dragg
ble();
        _this.rectLeftBottom = this.blockGroup.rect(sideLength, sideLength).stroke(sideWidth).dr
ggable();
        _this.rectRightTop = this.blockGroup.rect(sideLength, sideLength).stroke(sideWidth).dra
gable();
        _this.rectRightBottom = this.blockGroup.rect(sideLength, sideLength).stroke(sideWidth).d
aggable();

        _this.rectLeftCenter = this.blockGroup.rect(sideLength, sideLength).stroke(sideWidth).dr
ggable();
        _this.rectRightCenter = this.blockGroup.rect(sideLength, sideLength).stroke(sideWidth).d
aggable();
        _this.rectTopCenter = this.blockGroup.rect(sideLength, sideLength).stroke(sideWidth).dr
ggable();
        _this.rectBottomCenter = this.blockGroup.rect(sideLength, sideLength).stroke(sideWidth)
draggable();


        var lastPoint = null;
        var xLeft;
        _this.rectLeftCenter.on("dragstart", function(event) {
```

```
        lastPoint = event.detail.p;
        xLeft = 1;
    });

    _this.rectLeftCenter.on("dragmove", function() {
        var currPoint = event.detail.p;
        var currPoint = event.detail.p;
        var dx = currPoint.x - lastPoint.x;

        var ele = _this.currentElement;
        var width = ele.width() - xLeft * dx;

        if (width > 0) {
            var newX = ele.x() + xLeft * dx;
            ele.x(newX).width(width);
        } else {
            //invert
            xLeft = -xLeft;
            ele.x(ele.bbox().x2).width(-width).matrix(new SVG.Matrix(ele).flip('x', ele.bbox().cx));
        }

        lastPoint = currPoint;
    });
    _this.rectLeftCenter.on("afterdragmove", function() {
        _this.rebound(_this.currentElement.bbox());
    });

};


HandleBorder.prototype.rebound = function(bbox) {
    var _this = this;

    var x1 = bbox.x;
    var y1 = bbox.y;
    var x2 = bbox.x2;
    var y2 = bbox.y2;
    _this.rectLeftTop.move(x1 - sideLength, y1 - sideLength);
    _this.rectLeftBottom.move(x1 - sideLength, y2);
    _this.rectRightTop.move(x2, y1 - sideLength);
    _this.rectRightBottom.move(x2, y2);

    _this.rectLeftCenter.move(x1 - sideLength, (y2 + y1 - sideLength) / 2);
    _this.rectRightCenter.move(x2, (y2 + y1 - sideLength) / 2);
    _this.rectTopCenter.move((x2 + x1 - sideLength) / 2, y1 - sideLength);
    _this.rectBottomCenter.move((x2 + x1 - sideLength) / 2, y2);

    this.blockGroup.matrix(new SVG.Matrix(_this.currentElement));
};

HandleBorder.prototype.show = function(svgEle) {
    if (!svgEle) {
        return;
    }
```

```
        this.currentElement = svgEle;
        this.handleBorderGroup.show();

        this.rebound(svgEle.bbox());
    };

    HandleBorder.prototype.hide = function() {
        this.handleBorderGroup.hide();
    };

    this.HandleBorder = HandleBorder;

})();
```

附

涉及到数学思维，总是绕不出去，只能不断的试错。

为每个操作框绑定事件太繁琐，是否可以通用？