HTML5 Canvs性能测试笔记

目标:

建立基准测试数据，衡量优化效果

硬件:

1. iMac应用程序
2. XCodePad模拟器

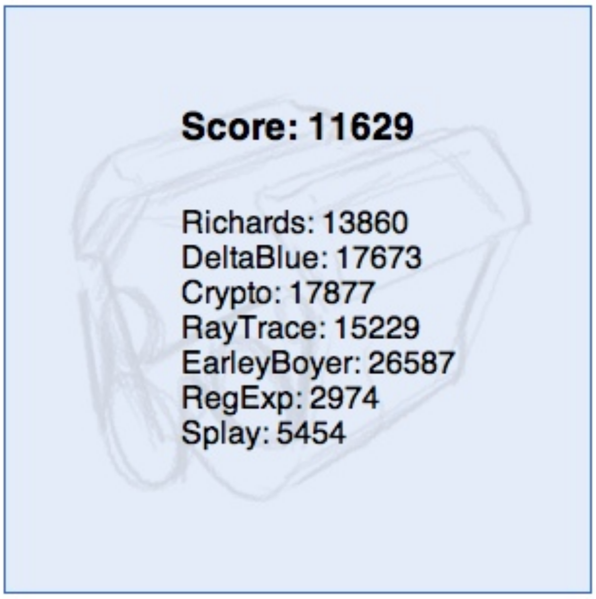
程序:

1.Chrome17
2.Safari5.1.3
3.Self Build MiniBrowser(WebKit-r107210)
4.模拟器上的Safari
5.IOSversion QQBrowser

1.纯JavaScript引擎性能测试基准数据:

测试工具：http://v8.googlecode.com/svn/data/benchmarks/v6/run.html

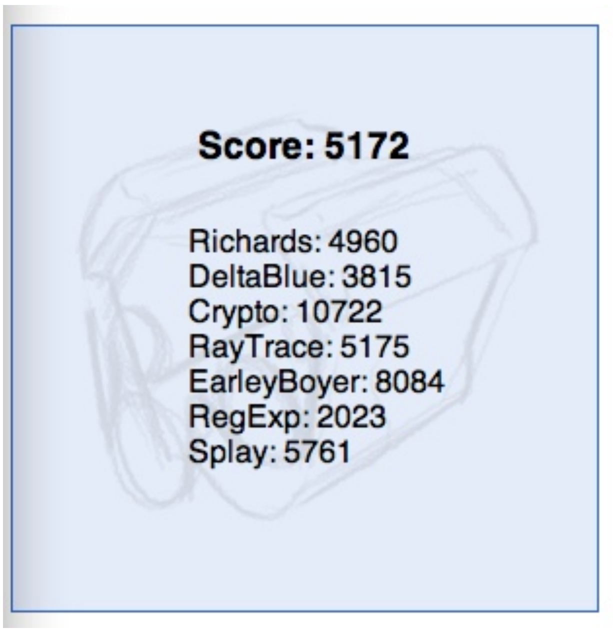
1.



Score: 11629

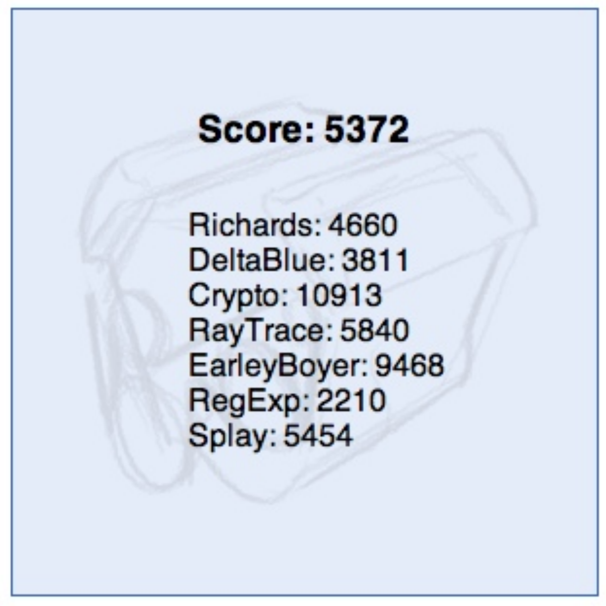
Richards: 13860
DeltaBlue: 17673
Crypto: 17877
RayTrace: 15229
EarleyBoyer: 26587
RegExp: 2974
Splay: 5454

2.



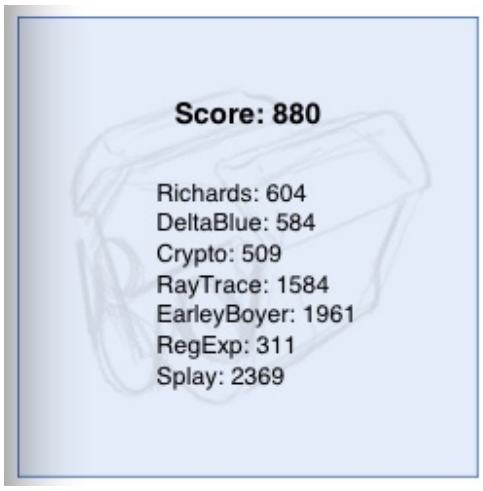
```
Score: 5172

Richards: 4960
DeltaBlue: 3815
Crypto: 10722
RayTrace: 5175
EarleyBoyer: 8084
RegExp: 2023
Splay: 5761
```

3.



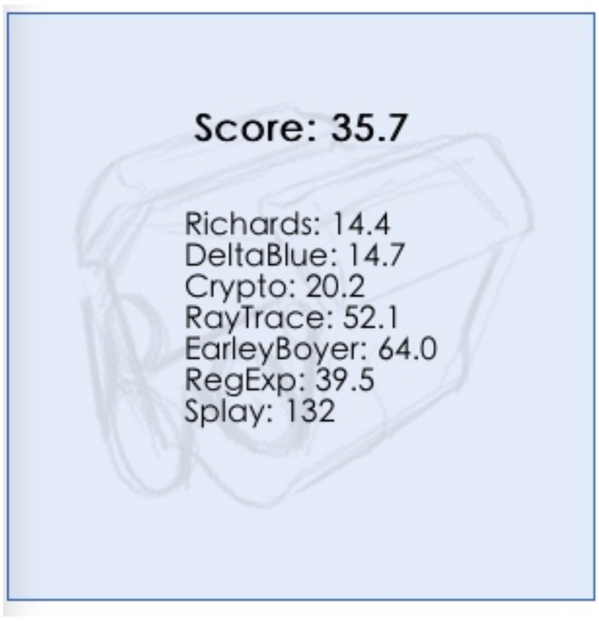
```
Score: 5372

Richards: 4660
DeltaBlue: 3811
Crypto: 10913
RayTrace: 5840
EarleyBoyer: 9468
RegExp: 2210
Splay: 5454
```

4.


Score: 880

Richards: 604
DeltaBlue: 584
Crypto: 509
RayTrace: 1584
EarleyBoyer: 1961
RegExp: 311
Splay: 2369

5.


Score: 35.7

Richards: 14.4
DeltaBlue: 14.7
Crypto: 20.2
RayTrace: 52.1
EarleyBoyer: 64.0
RegExp: 39.5
Splay: 132

2.WebCore解析性能数据:

CASE1： HTML5复杂页面解析（http://localhost:8080/html-parser.html）

1. Chrome

```
avg 2293.9
median 2303
```

2.Safari

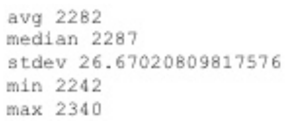
```
avg 1548.85
median 1605.5
```

3.MiniBrowser

```
avg 1626
```

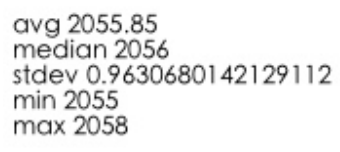
```
median 1619.5
```

4.iOS Safari



```
avg 2282
median 2287
stdev 26.67020809817576
min 2242
max 2340
```

5.QQBrowser



```
avg 2055.85
median 2056
stdev 0.9630680142129112
min 2055
max 2058
```

2.简单Canvas性能测试基准数据：
PEDDING

3.复杂Canvas性能测试基准数据：(需要大量重复Cavas绘制，大量填充操作)
http://localhost:8080/HTML5%20canvas%20performance%20test%20-%20Scott%20Porter.html

1. Chrome： 64 fps
2. Safari： 229 fps
3. MiniBrowser： 231 fps
4. iOS Safari： 47 fps
5. QQBrowser：13 fps

附录：

一些Canvas相关的JS级别优化方法
http://www.html5rocks.com/en/tutorials/canvas/performance/

a. 输出到离屏Canvas上

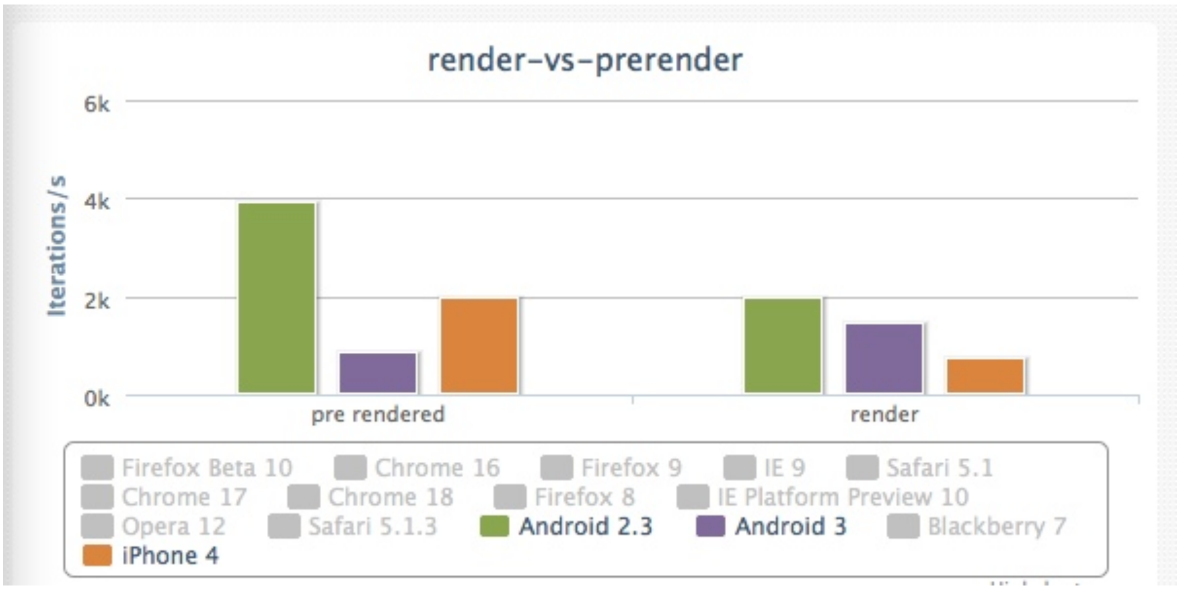
no pre-rendering:

```
// canvas, context are defined
function render() {
  drawMario(context);
  requestAnimationFrame(render);
}
```

pre-rendering:

```
var m_canvas = document.createElement('canvas');
m_canvas.width = 64;
m_canvas.height = 64;
var m_context = m_canvas.getContext(‘2d’);
drawMario(m_context);

function render() {
  context.drawImage(m_canvas, 0, 0);
  requestAnimationFrame(render);
}
```

性能差异：（测试工具 jsperf：http://jsperf.com/render-vs-prerender/3 ）



b. 尽量调用批量绘图

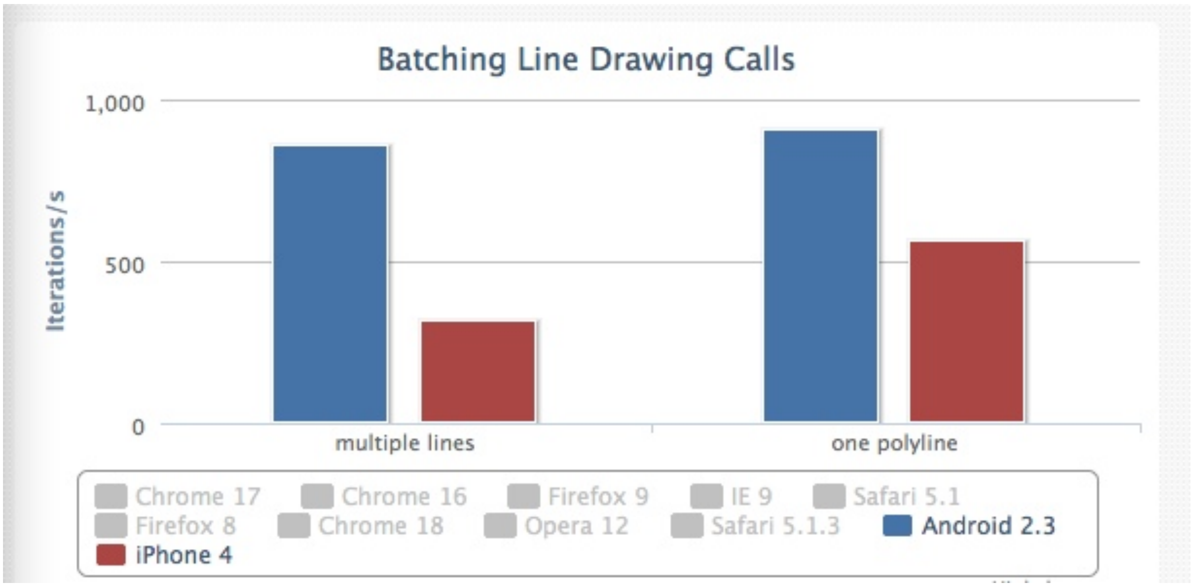
低效做法：

```
for (var i = 0; i < points.length - 1; i++) {
  var p1 = points[i];
  var p2 = points[i+1];
  context.beginPath();
  context.moveTo(p1.x, p1.y);
  context.lineTo(p2.x, p2.y);
  context.stroke();
}
```

高效做法：

```
context.beginPath();
for (var i = 0; i < points.length - 1; i++) {
  var p1 = points[i];
  var p2 = points[i+1];
  context.moveTo(p1.x, p1.y);
  context.lineTo(p2.x, p2.y);
}
context.stroke();
```

性能数据：（http://jsperf.com/batching-line-drawing-calls/2）



c. 尽量避免Canvas状态机的状态切换，例如颜色，或者是填充模式

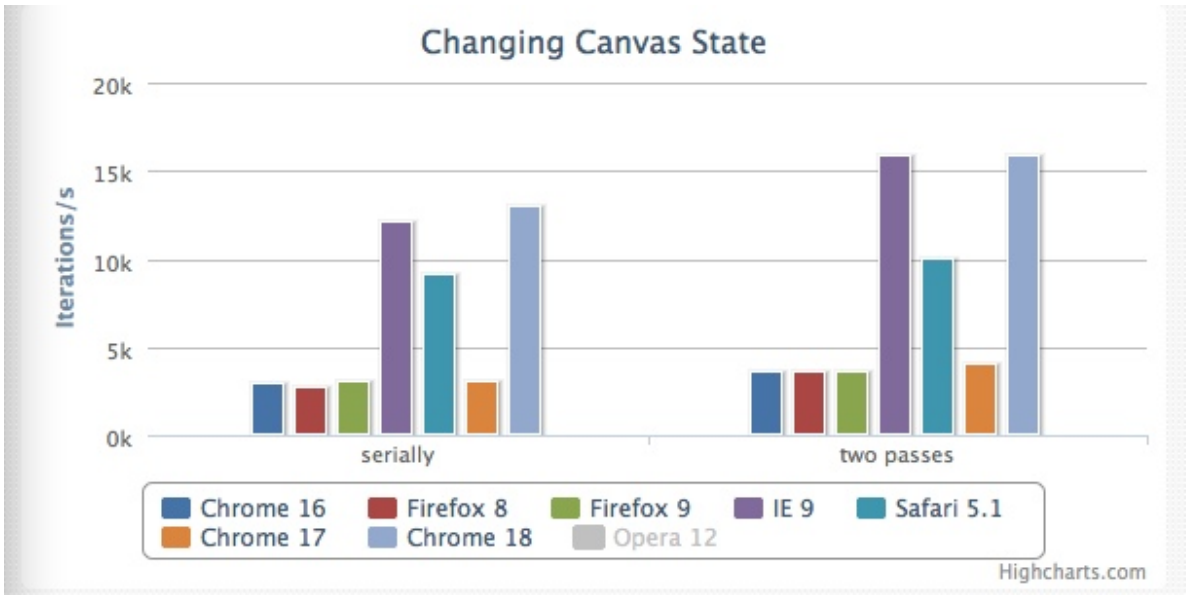
低效方式，太频繁的切换颜色和填充模式：

```
for (var i = 0; i < STRIPES; i++) {
  context.fillStyle = (i % 2 ? COLOR1 : COLOR2);
  context.fillRect(i * GAP, 0, GAP, 480);
}
```

高效方式：

```
context.fillStyle = COLOR1;
for (var i = 0; i < STRIPES/2; i++) {
  context.fillRect((i*2) * GAP, 0, GAP, 480);
}
context.fillStyle = COLOR2;
for (var i = 0; i < STRIPES/2; i++) {
  context.fillRect((i*2+1) * GAP, 0, GAP, 480);
```

性能比较：



d.只重绘改变的区域可以极大的提高性能

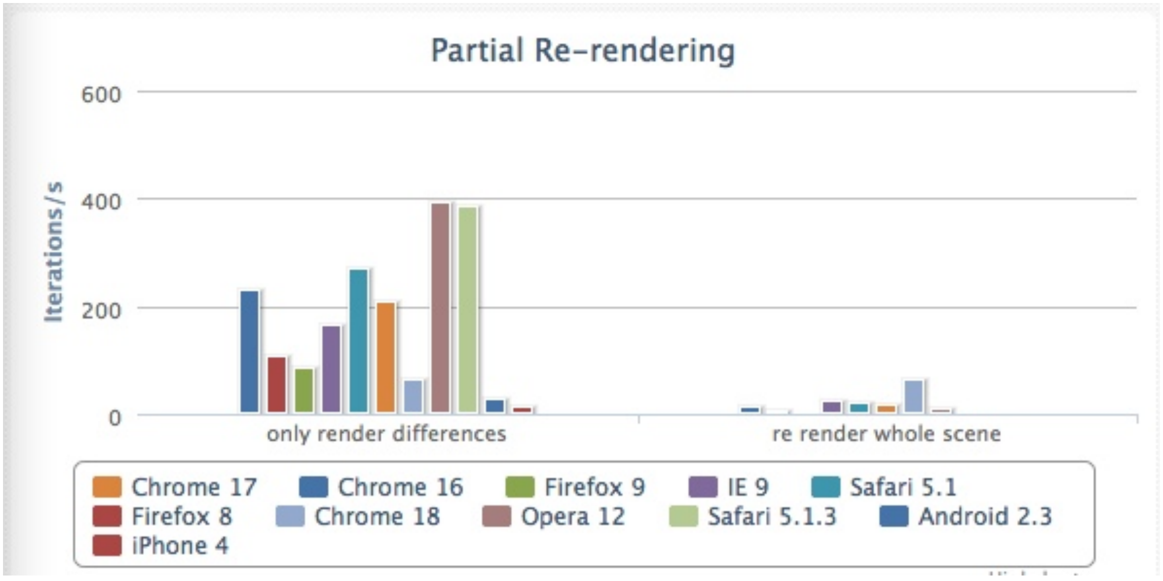
重绘整个Canvas:

```
context.fillRect(0, 0, canvas.width, canvas.height);
```

重绘变化区域:

```
context.fillRect(last.x, last.y, last.width, last.height);
```

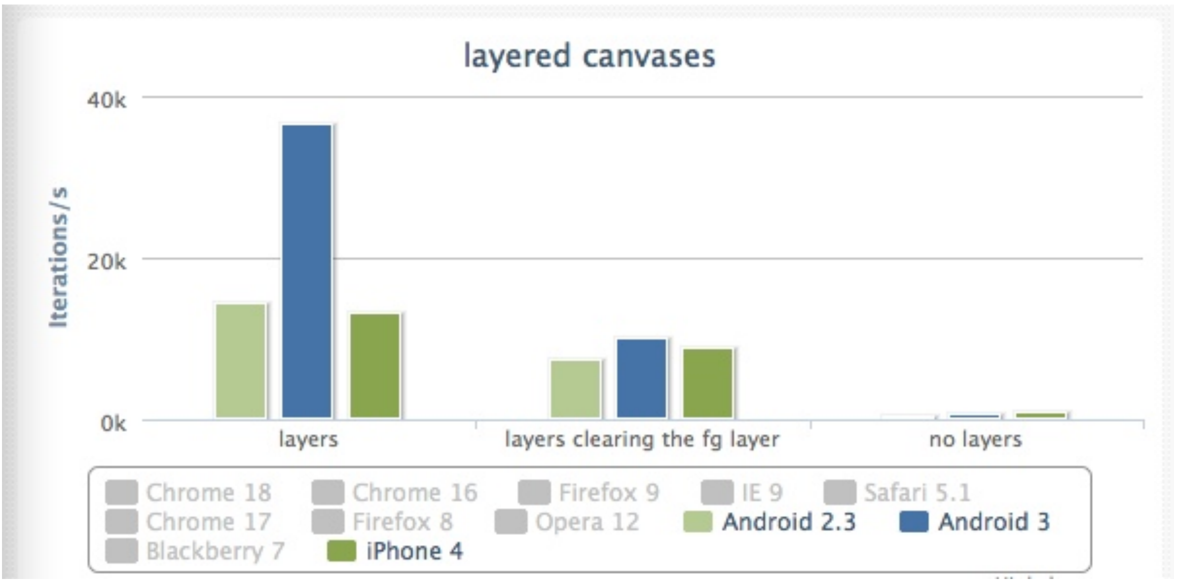
性能:



e.复杂的场景使用多个图层绘制，每个图层一个Canvas,利用GPU来计算图形融合，提高性能。

多个图层，利用Zindex控制图层之间的叠放孙序

```
<canvas id="bg" width="640" height="480" style="position: absolute; z-
index: 0"]]>
</canvas>
<canvas id="fg" width="640" height="480" style="position: absolute; z-
index: 1"]]>
</canvas>
```

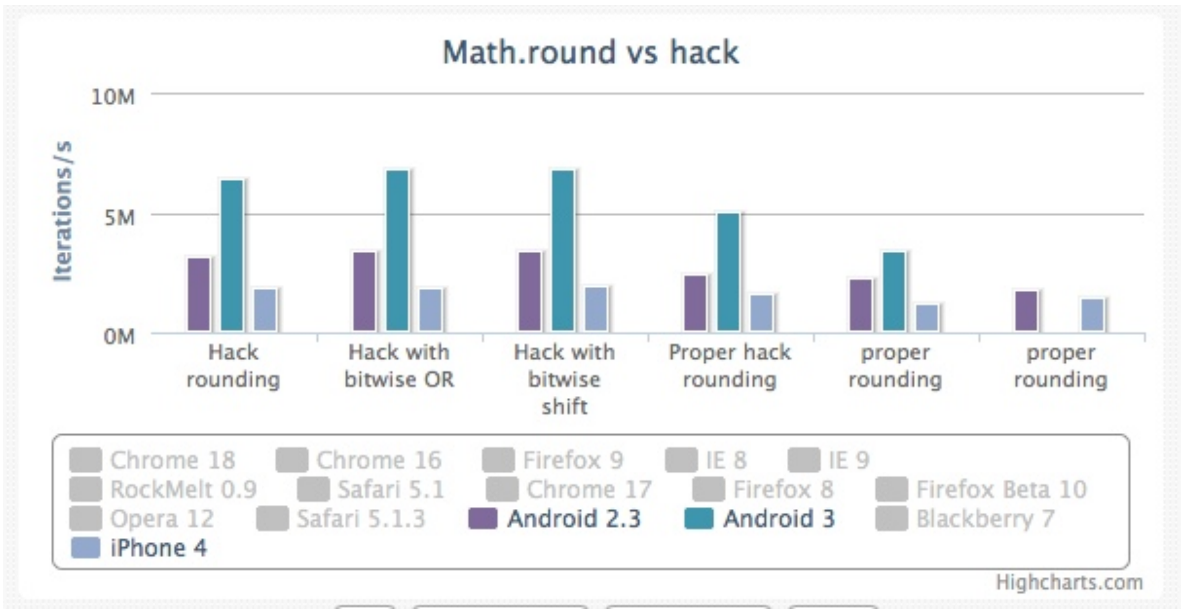


f: 避免阴影模糊，阴影模糊会比较大的消耗计算性能（在没有GPU的情况下）

g: 避免浮点数坐标
因为Canvas支持sub-pixel渲染，并且不能关闭这个特性，所以使用浮点坐标会自动进行anti-aliasing计算，这也会影响性能，最好在绘制之前把坐标转换成整数：
JS里面的标准转换方法是：　Math.floor / Math.round
但是还有一些更快的转换方法：

```
// With a bitwise or.
rounded = (0.5 + somenum) | 0;
// A double bitwise not.
rounded = ~~ (0.5 + somenum);
// Finally, a left bitwise shift.
rounded = (0.5 + somenum) << 0;
```

性能：



原因： javascript中的bit位操作都会首先把数字转换成Integer,再进行操作。

h: 使用新增加的requestAnimationFrame函数来调度绘制事件，这时目前比较推荐的方式，用于替代以前的SetInterval方法，这个函数也不能保证绝对精确的调度，所以还是要纪录事件的变化。

例程：

```
var x = 100;
var y = 100;
var lastRender = new Date();
function render() {
  var delta = new Date() - lastRender;
  x += delta;
  y += delta;
  context.fillRect(x, y, W, H);
  requestAnimationFrame(render);
}
render();
```

i: 移动设备上的Canvas实现基本上都比较慢，目前只有iOS5上的Safari实现了GPU加速。