HTML5游戏探索之路

杜欢（Huan Du）, MagnetJoy Games
Twitter: @huandu

HTML5与游戏

既然已经有了那么多与游戏相关的技术……

为什么使用HTML5?

HTML5与游戏

- HTML5的优势
  - 跨平台、标准化
    - 在桌面电脑/手持设备上具有完全相同的API
  - 播放方式最简单
    - 完全基于浏览器，无需任何插件
    - 能与现有的互联网产品无缝集成
  - 得到各大公司的广泛支持

- HTML5的不足
  - 糟糕的性能
  - 缺少为游戏开发而设计的API

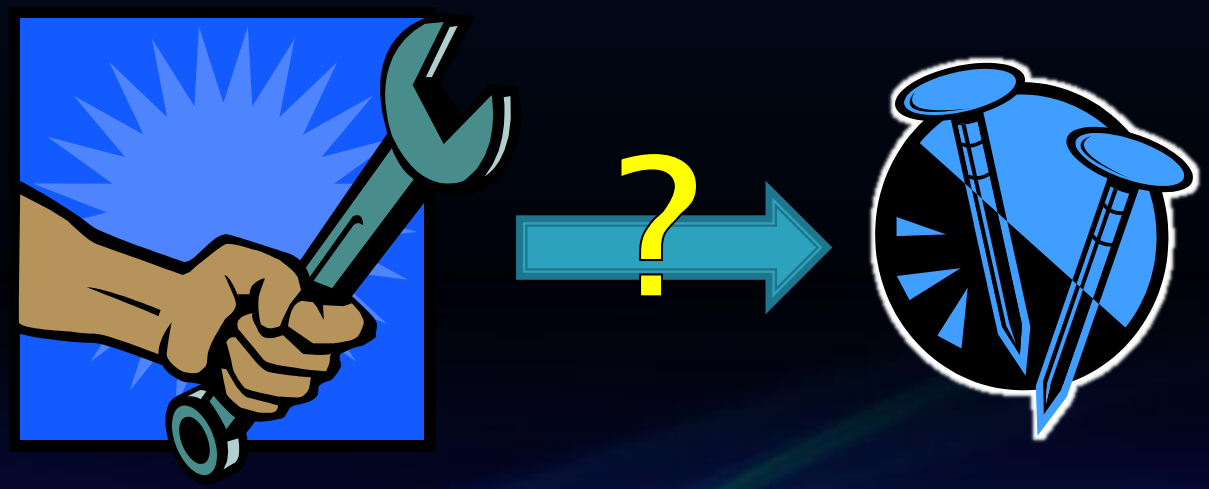
相关技术

- 经常用到的
  - HTML5 canvas
  - Javascript – 实现游戏逻辑
  - DOM，主要是DOM Event
  - SVG
  - HTML5 <audio>和<video>
- 偶尔用到的
  - WebGL
  - CSS Animation – key frames和timing function
  - Session和local storage

HTML5游戏的开发难点

HTML5

不是为开发游戏而设计的语言



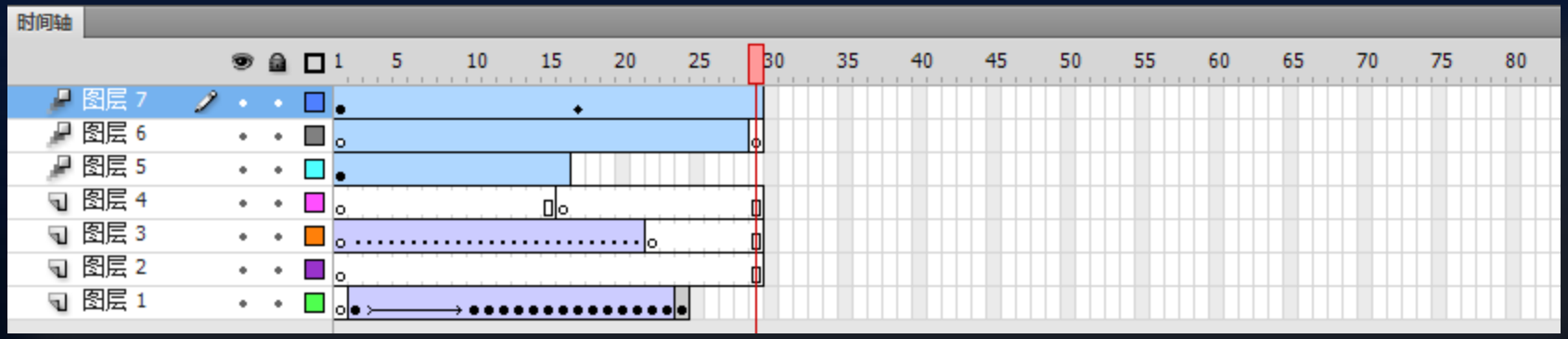
HTML5游戏的开发难点

- 无穷无尽的难点……
  - 时间轴
  - 键盘按键状态
  - 鼠标事件缓存
  - 关键帧与帧动画
  - 图层
  - 触控设备支持
  - 性能问题
  - ……

HTML5游戏的开发难点

幸好，难点总能逐个攻破

难点：时间轴



- HTML5的时间轴在哪里？

难点：时间轴

- 最简单的方案

```
1  function render() {
2      // implement rendering logic
3  }
4
5  window.setInterval(render, 40);
```

- 可惜……这个时间轴非常不准确
  - 即使在空闲状态下，setInterval()的精度量级为10ms
  - FPS并不稳定

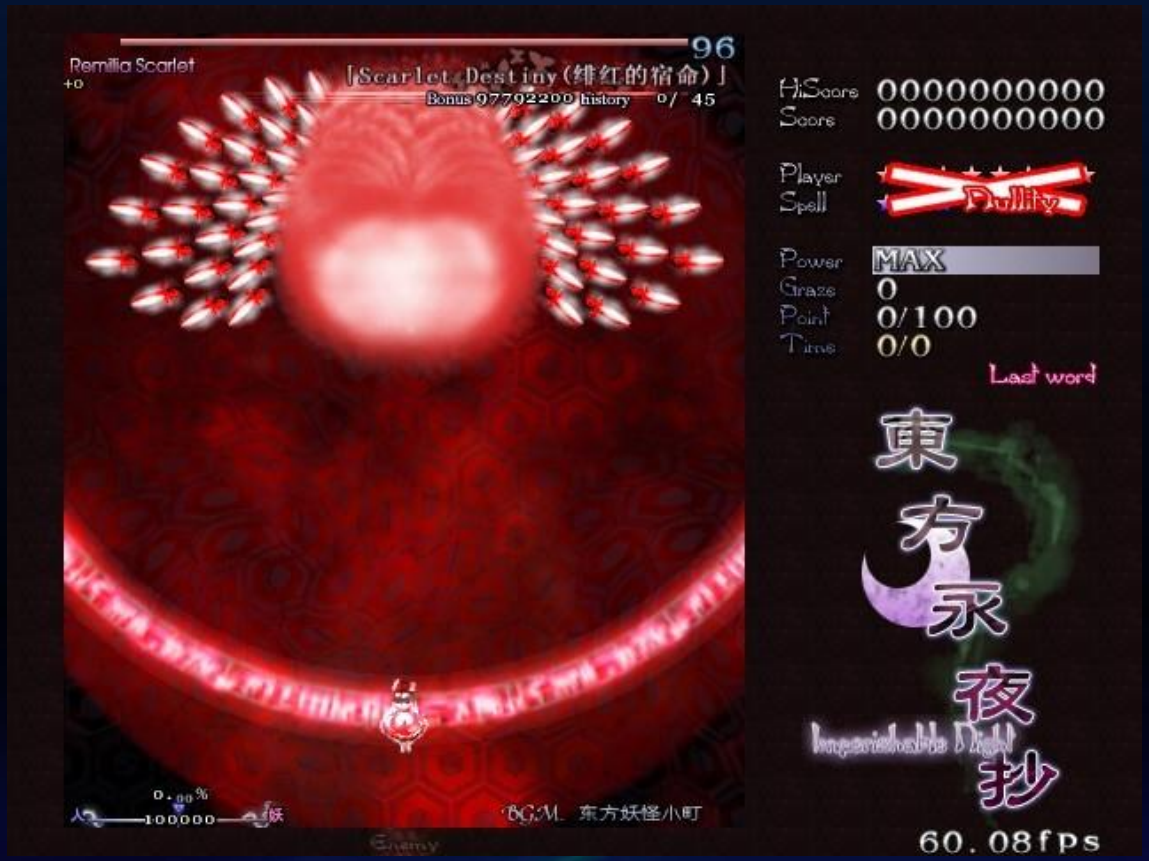
难点：时间轴

- 一个更好的方案
  - 在每一秒里面计算已渲染的帧数
  - 设置很小的回调间隔，仅在正确的时间进行渲染
  - 可以方便的得到FPS信息

```
var fps = 30,
    interval = 1000 / fps,
    now = Date.now(),
    lastSecond = Math.floor(now / 1000),
    frameCount = Math.ceil((now % 1000) / fps);

window.setInterval(function() {
    var now = Date.now(),
        second = Math.floor(now / 1000),
        milliSecond = now % 1000;

    if (second != lastSecond) {
        frameCount = 0;
        lastSecond = second;
    }

    if (frameCount < fps
        && milliSecond >= frameCount * interval) {
        frameCount++;

        // TODO: your render logic
    }
}, 1);
```

难点：键盘按键状态

- DOM Event不会直接告诉你那些键同时按下



难点：键盘按键状态

- 代码片段
  - 监听事件：keydown/keyup/blur
  - blur时清除状态
  - 注意：特殊按键 SHIFT/CTRL/ALT/CMD 状态无法保证正确

```
1   var specialKeys = {},
2       keyStates = {},
3       keyboardHandler = function(e) {
4           var isDown = e.type === 'keydown';
5           e.stopPropagation();
6           e.preventDefault();
7
8           if (!specialKeys[e.keyCode]) {
9               keyStates[e.keyCode] = isDown;
10          }
11      },
12      blurHandler = function(e) {
13          keyStates = {};
14      };
15
16  specialKeys[16] = specialKeys[17]
17      = specialKeys[18] = true;
18  canvas.addEventListener('keydown',
19      keyboardHandler, false);
20  canvas.addEventListener('keyup',
21      keyboardHandler, false);
22  canvas.addEventListener('blur',
23      blurHandler, false);
```

难点：鼠标状态

- 不要尝试立即处理鼠标事件
  - 每秒钟可产生的鼠标事件 > 130
  - 实际游戏的FPS <= 30即可
  - Javascript在执行时会阻塞UI线程，从而导致事件丢失

- 解决方案
  - 在缓存中记录每个鼠标事件
  - 每次渲染时处理缓存
  - 自动清空缓存中已经处理过的事件

难点：鼠标状态

- 代码片段
  - 循环缓冲区
  - 忽略重复的事件
  - 仅在一个元素上捕获鼠标事件

```
1   var MAX = 300, head = 0, tail = 0,
2       history = [],
3       mousemoveHandler = function(e) {
4           var last = history[head];
5           if (e.screenX == last.screenX
6               && e.screenY == last.screenY) {
7               // ignore duplication
8               return;
9           }
10
11          head = (head + 1) % MAX;
12          history[head] = {
13              // store attr in event...
14          };
15
16          if (tail == head)
17              tail = (tail + 1) % MAX;
18      },
19      traverseHistory = function(cb) {
20          var len = (head - tail + MAX) % MAX, i;
21          for (i = 0; i < len; i++)
22              cb(history[(i + tail) % MAX]);
23      };
24
25  canvas.addEventListener('mousemove',
26      mousemoveHandler, false);
```

难点：关键帧和帧动画

- 基本思路
  - 将关键帧事先绘制好，制作成图片
  - 配置帧出现的时间点、帧与帧之间的间隔
  - 将关键帧图片打包成整张图片

当前可见区域

难点：关键帧和帧动画

- Canvas 2d方案
  - 用drawImage绘制图片
  - 设置合适的clip 区域
  - 动态绘制的关键帧可使用getImageData/putImageData
- CSS Animation
  - @keyframes
  - Timing functions

难点：图层

- 图层的概念对游戏非常重要
  - 性能因素
    - 仅仅重绘必要部分
    - 缓存以固定模式变化的帧动画
  - “Movie Clip”
    - Play/Stop/Show/Hide
    - 封装游戏逻辑
  - 事件scope
    - 只处理落在图层中的事件

难点：图层

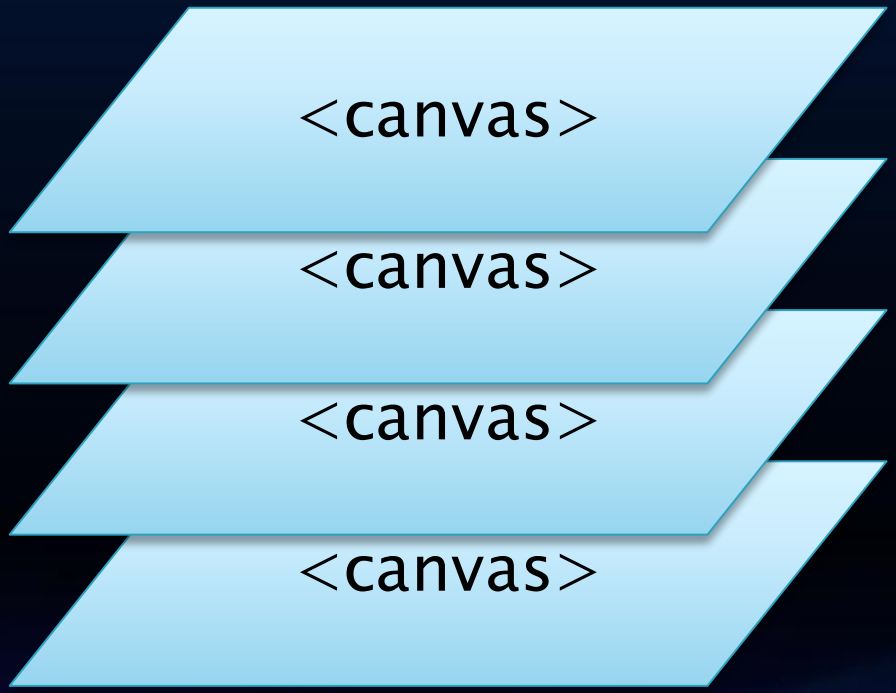
- 图层的两种实现思路
  - 共享canvas的逻辑图层
    - 使用同一个canvas
    - 将绘图代码按逻辑关系封装成“图层”
    - 难点：如何尽可能缩小重绘区域
    - 优点：
      - 可最大化绘图性能，方便实现timing function
    - 缺点：
      - 图层间无法完全独立，需要反复重绘同样内容
    - 适用场景
      - 存在大量动态图元的场景

难点：图层

- 图层的两种实现思路（续）
  - 基于DOM元素层级关系
    - 创建多个canvas结点
    - 独立渲染，用zIndex管理层级关系
    - 优点：
      - 图层互相不干扰，业务逻辑可完全独立
    - 缺点：
      - 动态渲染效率较低
    - 适用场景：
      - 背景动画/游戏菜单/独立的动画元件

难点：图层

- 基于DOM的图层：理想DOM结构，可惜做不到




```
<!- fake code-->
<canvas>
  <canvas>
    <canvas>
      <canvas></canvas>
    </canvas>
  </canvas>
</canvas>
```

难点：图层

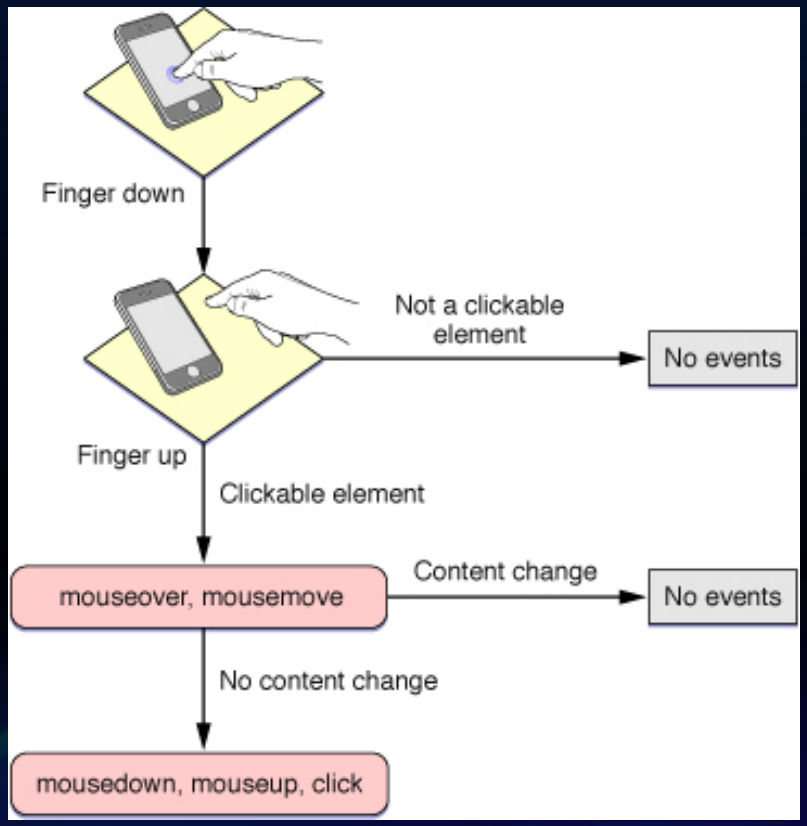
- 为什么不使用最自然的DOM布局
  - <canvas>不支持子节点
  - 缓存鼠标事件后，offsetX/Y和clientX/Y无法自动修正
  - DOM元素没有封装必要的方法 – play/stop/show/hide
  - 不能方便的嵌入游戏逻辑

- 实际DOM结构

```
<div>                    ← 父图层
  <canvas></canvas>
  <div>...</div>         ← 子图层
</div>
```

难点：触控设备支持

- 仅在iOS/Android上支持
  - touchstart/touchmove/touchend
  - 也需要先缓存，再处理
  - 必须调用preventDefault()
  - iOS文档 http://bit.ly/g8bBXl
  - W3C Editor's Draft http://bit.ly/h98JPr



难点：触控设备支持

- 代码片段
  - 监听touchstart和touchmove
  - 使用e.touches[0]
  - e.preventDefault()
- 多点触摸支持
  - changedTouches属性
  - iOS gesture事件

```
var MAX = 300, head = 0, tail = 0,
    history = [],
    touchHandler = function(e) {
        var last = history[head],
            touch = e.touches[0];
        e.preventDefault();

        if (touch.screenX == last.screenX
            && touch.screenY == last.screenY) {
            // ignore duplication
            return;
        }

        head = (head + 1) % MAX;
        history[head] = {
            // store attr in event...
        };

        if (tail == head)
            tail = (tail + 1) % MAX;
    };

canvas.addEventListener('touchstart',
    touchHandler, false);
canvas.addEventListener('touchmove',
    touchHandler, false);
```

难点：性能

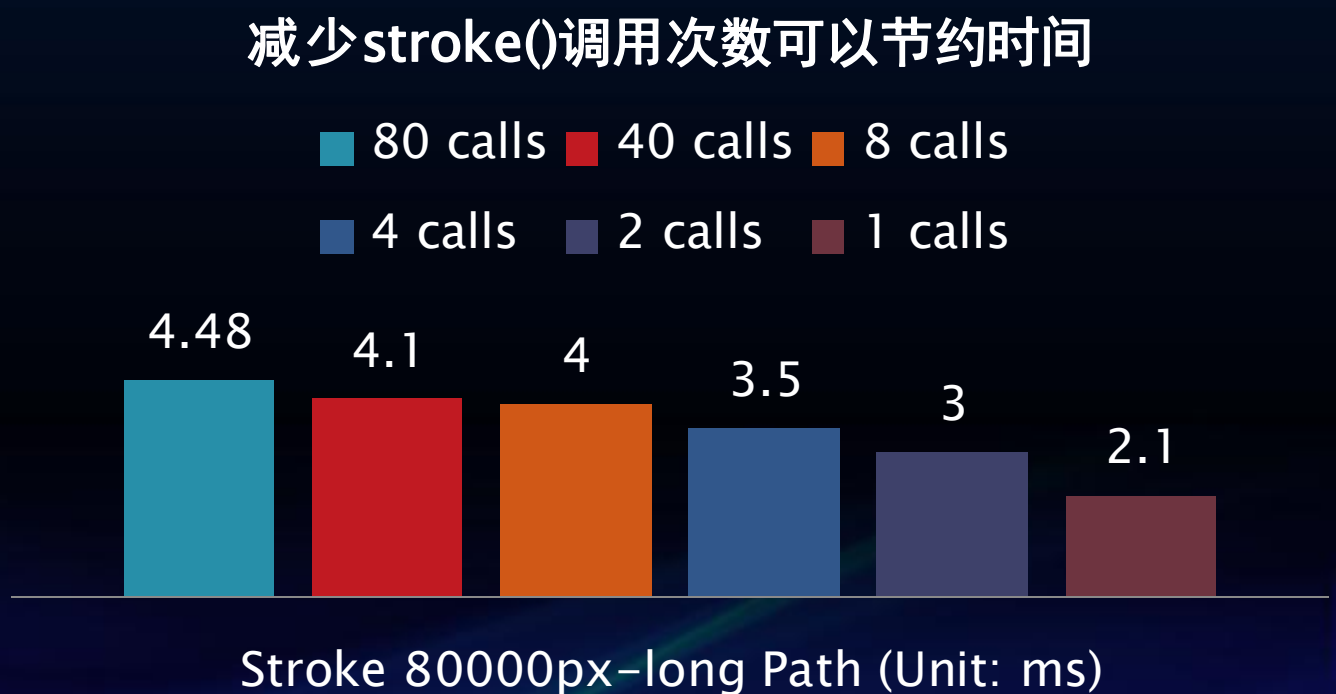
最大的性能瓶颈是

Canvas绘图效率

自身 ▼	总计	平均	调用	函数
84.74%	84.74%	0.00%	179600	▶ stroke
8.24%	100.00%	8.24%	1	(program)
2.68%	89.24%	0.00%	179600	▶ renderBubble
1.26%	1.26%	0.00%	898	▶ clearRect
1.25%	1.25%	0.00%	359200	▶ arc
0.90%	90.43%	0.00%	898	▶ (anonymous function)
0.30%	0.30%	0.00%	179600	▶ beginPath
0.27%	0.27%	0.00%	359200	▶ moveTo
0.08%	0.29%	0.00%	5610	▶ (anonymous function)
0.07%	0.11%	0.00%	5610	▶ (anonymous function)
0.05%	0.05%	0.00%	11220	▶ (anonymous function)

难点：性能

Tip1
减少调用stroke()/fill()/clip()的次数



难点：性能

Tip2

水平/竖直的直线绘图性能最佳

难点：性能

Tip3

尽量使用clip()缩小绘图区域

难点：性能

- 尽量不要使用canvas
  - Adobe Wallaby
    - 将Flash动画导出成 SVG + CSS3 Animation
  - DOM元素动画 – 古老但是反而有效
    - Absolute positioning: 改变left/top/right/bottom
    - CSS translate
  - webgl-2d
    - 使用webgl 渲染2d canvas

    https://github.com/corbanbrook/webgl-2d

HTML5游戏引擎

- 设计目标
  - 提供游戏API，减少重复劳动
  - 解决常见的开发难点
  - 让开发者专注于游戏的逻辑

HTML5游戏引擎

- 几个开源游戏引擎
  - The Render Engine
    - http://www.renderengine.com/
  - Cocos2d
    - http://cocos2d-javascript.org/
  - gin
    - https://github.com/huandu/gin
  - GameJS
    - http://gamejs.org/

HTML5游戏引擎

- 典型游戏引擎结构

Demos

Thanks

杜欢（Huan Du）, MagnetJoy Games
Twitter: @huandu

开源游戏引擎简介

- The Render Engine
  - 完全面向对象设计
    - 具有丰富的类库
    - 使用者必须从已有基类派生，采用面向对象的方法开发
  - 广泛的设备支持
    - 支持桌面/手机/移动设备/Wii等
    - 在IE 8及以下版本上使用FlashCanvas Pro代替canvas http://flashcanvas.net/
  - 支持帧动画，提供了一系列动画函数
  - 内建碰撞检测、Box2d、异步资源加载、音效支持
  - 精确的时间轴控制，可按帧或毫秒控制动画

开源游戏引擎简介

- Cocos2d
  - 完全面向对象设计，具有丰富的类库
  - 支持静态/动态打包代码和资源文件
    - 可使用编译脚本将js、图片、帧动画编译到一个js中
    - 同时提供cocos服务，动态将请求打包，方便调试
  - 支持帧动画，提供丰富的timing function
  - 内建碰撞检测、Box2d

开源游戏引擎简介

- gin
  - jQuery风格，用法简洁，插件式设计，易于扩展
  - 完全通过事件驱动
  - 支持桌面/手持设备
  - 支持鼠标/键盘/触控消息缓存
    - 详细记录每个消息的细节
    - 可方便的遍历所有消息
    - 屏蔽鼠标和触控消息的区别

gin正在持续开发中，欢迎大家在github上fork这个项目

```
1  $G('your-game-container-id', {}, {
2      render: function(e) {
3          // check if 'space' key pressed
4          if (e.keyState[0x20]) {
5              // do something
6          }
7
8          // check if mouse primary button pressed
9          // see DOM L3 Event attr buttons definition
10         if (e.buttons & 1) {
11             // do something
12         }
13
14         e.history.each(function(event) {
15             // work with all cached events
16         });
17     }
18 });
```

开源游戏引擎简介

- GameJS
  - Javascript版的PyGame（http://pygame.org）
  - 面向对象设计，提供众多工具类
  - 面向过程的使用方法
  - 支持键盘/鼠标/AJAX事件缓存
  - 内置碰撞检测、AJAX异步请求、资源异步加载支持